А. Азарян, Н. Карабут, Т. Козикова,
О. Рибальченко, А. Трачук, Н. Шаповалова

ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ мовами C++, Visual Basic, Turbo Pascal

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Азарян А.А., Карабут Н.О., Козикова Т.П.,
Рибальченко О.Г., Трачук А.А., Шаповалова Н.Н.

НАВЧАЛЬНИЙ ПОСІБНИК
ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ МОВАМИ C++, VISUAL BASIC, TURBO PASCAL

Рекомендовано вченою радою
Криворізького національного університету
як навчальний посібник

Кривий Ріг
2014

ББК

В93

Рекомендовано до друку вченою радою
Криворізького національного університету
(протокол № від р.)

Рецензенти:

Азарян А.А., Карабут Н.О., Козикова Т.П., Рибальченко О.Г., Трачук А.А., Шаповалова Н.Н.

В93 Основи алгоритмізації та програмування: Навчальний посібник. – Кривий Ріг: Вид-во ОктанПринт, 2014. - 308 с.

ISBN

Посібник містить базові поняття про алгоритми та способи їх представлення. Наведена класифікація алгоритмів та приклади розв’язання класичних моделей обробки даних. Усі алгоритми реалізовані мовами C++, Visual Basic та Turbo Pascal. Про кожну з цих мов програмування викладено основні відомості. Посібник містить набір типових задач та їх розв’язки з повними кодами програм.

Для студентів вищих навчальних закладів III-IV рівнів акредитації.

ISBN

ЗМІСТ

ВСТУП

В наш час обчислювальна техніка має в своєму розпорядженні різні технічні засоби і майже необмежені можливості для виконання робіт, пов'язаних з логічною обробкою інформації і обчисленнями. Так, без обчислювальних машин немислиме вирішення трудомістких задач, вибір оптимальних варіантів проектів, раціональної структури пристроїв, ефективного використання комплексів машин і систем різного призначення.

Обчислювальна машина стала незамінним інструментом в руках інженера-дослідника. Вона відкриває великі можливості в нових розробках. До застосування швидкодіючих обчислювальних машин мистецтво дослідника полягало в основному у максимальному спрощенні задачі, виявленні і відкиданні тих чинників, які не впливають істотного на об'єкт, що вивчається. Розрахункова частина при розробці нових зразків техніки складала невелику частку і зводилася до наближених рішень. Задача розв'язувалася в основному шляхом проведення експерименту, що вимагало великих витрат праці, часу і матеріальних засобів. В наш час мистецтво дослідника полягає в тому, щоб якнайповніше і точніше охарактеризувати явище, що вивчається, і дати його строгий математичний опис.

Скоротити шлях від складних математичних рівнянь до отримання конкретних результатів, прискорити виконання обчислювальних операцій і звільнити людину від стомлюючої роботи допомагає обчислювальна машина.

Необхідність в своєчасній і якісній обробці всілякої інформації приводить в наш час до широкого використання обчислювальних машин для керування процесами і об'єктами в різних областях промисловості, транспорту, у військовій справі. Автоматичні і автоматизовані системи керування здійснюють збір, зберігання, передачу і переробку інформації, що відображає стан об'єктів, що регулюються.

Слід виділити область використання обчислювальних машин для виконання економічних розрахунків, науково обґрунтованого економічного аналізу діяльності промислових підприємств і різних галузей народного господарства.

Ведення виробничо-господарських робіт зобов'язує фахівців враховувати різноманітні чинники (наприклад, витрати праці на різні види продукції), порівнювати минулі витрати із теперішніми і майбутніми, оцінювати наявні ресурси і забезпечувати найвигідніші умови для розвитку господарства. Тому використання обчислювальних машин для оптимального планування, розрахунку всіх трудових витрат, собівартості продукції, що випускається, і фондів заробітної платні, визначення витрат основних і допоміжних матеріалів і т.д. дозволяє покращувати і удосконалювати адміністративно-господарську діяльність виробництва.

Матеріал посібника представлено таким чином, що він може бути добре засвоєний при різній кількості годин, що відводиться в учбових планах на вивчення курсів «Обчислювальна техніка», «Алгоритмізація та програмування».

1 ОСНОВИ АЛГОРИТМІЗАЦІЇ

1.1 Алгоритм. Способи представлення алгоритмів

Одним з базових понять інформатики, математики й кібернетики є поняття алгоритму як певного правила перетворення інформації. Воно містить вказівки про те, які операції в ході обробки даних і в якій послідовності необхідно виконати, щоб одержати шуканий результат або розв'язок завдання.

Кожний алгоритм повинен мати наступні основні властивості:

- *визначеність* (*детермінованість*) – запропоновані алгоритмом дії повинні бути визначені точно й однозначно (які-небудь довільні тлумачення виключаються). Дії являють собою детермінований, причинно-обумовлений процес, який скільки б раз не застосовувався до тих самим вихідних даних, повинен приводити до тих самих результатів виконання кожного кроку й усієї послідовності кроків у цілому;

- *дискретність* – процес, визначений алгоритмом, повинен мати переривчастий (дискретний) характер, тобто являти собою послідовність окремих елементарних кроків (дій або команд);

- *масовість* – застосовність алгоритму до будь-якого ряду вихідних даних, тобто забезпечення розв'язку будь-якого завдання з безлічі завдань деякого класу або одного типу;

- *результативність* – кожна дія повинна приводити до певного результату, і по закінченню виконання алгоритму має бути отриманий деякий результат;

- *кінцевість* – алгоритм повинен складатися з кінцевого числа кроків, кожний з яких вимагає для свого виконання кінцевий проміжок часу;

- *ефективність* – алгоритм повинен забезпечувати розв'язок за мінімальний час із мінімальними витратами оперативної пам'яті.

Враховуючи зазначені властивості алгоритму, сформулюємо наступне визначення:

Алгоритм – однозначна кінцева послідовність точно визначених кроків або дій, яка забезпечує розв'язок задачі за кінцевий час при мінімальному обсязі оперативної пам'яті.

Для представлення алгоритмів використовують такі способи:

- описовий (словесний опис);
- аналітичний;
- графічний;
- псевдокод;

- алгоритмічна мова.

Словесний опис алгоритму використовується в повсякденному житті у вигляді рецептів готування блюд, інструкцій до різних технічних і побутових приладів і пристроїв і т.п. Такому способу опису часто властива відсутність визначеності й неоднозначність дій внаслідок неоднозначності слів і визначень. Наприклад, «додайте трохи солі» (скільки, куди і як додати?), «за 10 хвилин до готовності вимкніть цибулю» (як визначити готовність?) і т.ін. Крім цього недоліку, словесний опис часто приводить до громіздких текстів, тому він застосовується лише для найпростіших алгоритмів.

Аналітичний спосіб представлення алгоритму використовується при розв'язку наукових і інженерних задач. Тут алгоритми описуються послідовністю розрахунків за математичними формулами.

Графічний спосіб, використовуючи різні геометричні фігури, дозволяє наочно зобразити послідовність здійснення різних етапів процесу і їх взаємозв'язок.

Запис алгоритму *алгоритмічною мовою* вимагає точного дотримання правил цієї мови, оскільки алгоритм має бути зрозумілим не тільки людині, але й комп'ютеру.

Псевдокод займає проміжне місце між словесним описом і алгоритмічною мовою. У цьому способі вживаються конструкції, близькі до алгоритмічної мови, але повне дотримання всіх правил не потрібно, оскільки вони призначені для розуміння людиною.

1.2 Схеми алгоритмів

Опис алгоритмів у вигляді блок-схем є найбільш наочним і найпоширенішим графічним способом представлення алгоритмів. Схеми алгоритмів відображають шлях даних при розв'язку задач і визначають етапи обробки, а також різні носії даних, які застосовуються. Схема складається з наступних символів (будемо їх традиційно називати блоками):

1) блоки даних, які можуть також вказувати вид носія даних;

2) блоки процесу, який слід виконати над даними;

3) лінії, що вказують потоки даних між процесами й (або) носіями даних;

4) спеціальні символи, використовувані для полегшення написання й читання схеми.

У стандарті визначені умовні позначки в схемах алгоритмів і встановлені правила виконання схем. У табл. 1 наведені графічні символи (блоки), які найчастіше використовуються при описі алгоритмів.

Правила застосування блоків

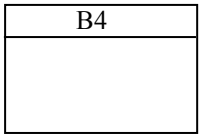
1. Блок призначений для графічної ідентифікації функції, яку він відображає, незалежно від тексту усередині цього блоку.

2. Блоки в схемі повинні бути розташовані рівномірно. Слід дотримуватися розумної довжини з'єднань і мінімального числа довгих ліній.
3. Кути й інші параметри, що впливають на відповідну форму символів не повинні змінюватися. Блоки повинні бути, по можливості, одного розміру.
4. Більшість блоків задумана так, щоб дати можливість включення тексту усередині блоку. Якщо обсяг тексту, що міститься усередині блоку, перевищує його розміри, слід використовувати блок коментаря.
5. У схемах може використовуватися ідентифікатор (номер, ім'я) блоків (блоку), який повинен розташовуватися ліворуч над блоком.
6. У схемах може використовуватися розгорнуте представлення, яке позначається за допомогою блоку зі смугою для процесу або даних. Блок зі смугою вказує, що в цьому ж комплекті документації в іншому місці є більш детальне представлення.

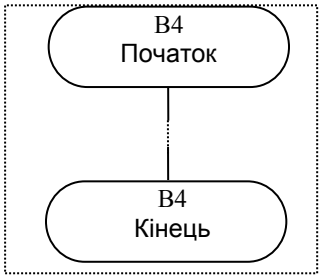
Блок зі смугою являє собою будь-який блок, усередині якого у верхній частині проведена горизонтальна лінія. Між цією лінією й верхньою лінією блоку розміщений ідентифікатор, що вказує на розгорнуте представлення даного блоку.

У якості першого й останнього блоку розгорнутого представлення має бути використаний блок покажчика кінця. Перший блок покажчика кінця повинен містити посилання, яке є також у блоці зі смугою.

Блок зі смугою



Розгорнуте представлення

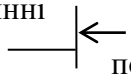


Правила виконання з'єднань:

1. Потоки даних у схемах показуються лініями, які проводять горизонтально або вертикально. Напрямок потоку зліва направо і зверху вниз вважається стандартним. У випадках, коли необхідно внести більшу ясність у схему (наприклад, при з'єднаннях), на лініях використовуються стрілки. Якщо потік має напрямок, відмінний від стандартного, стрілки повинні вказувати цей напрямок.

2. У схемах слід уникати перетинання ліній. Лінії, що перетинаються, не мають логічного зв'язку між собою, тому зміни напрямку в точках перетину не допускаються.

3. Дві або більше вхідних ліній можуть поєднуватися в одну вихідну лінію. Якщо дві або більше ліній поєднуються в одну лінію, то місце об'єднання повинне бути зміщене.

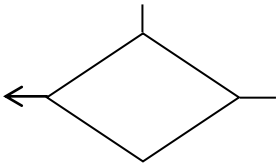
4. Лінії в схемах повинні  підходити до блоку або зліва, або зверху, а виходити або справа, або знизу. Лінії повинні бути спрямовані до центру блоку.

5. При необхідності лінії в схемах слід розривати для запобігання зайвих перетинань або занадто довгих ліній, а також, якщо схема складається з декількох сторінок. З'єднувач на початку розриву називається зовнішнім з'єднувачем, а з'єднувач наприкінці розриву – внутрішнім з'єднувачем. Посилання до сторінок можуть бути наведені разом із блоком коментаря для їхніх з'єднувачів.

Зовнішній з'єднувач Внутрішній з'єднувач

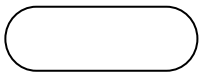
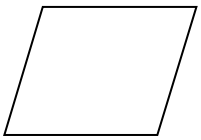
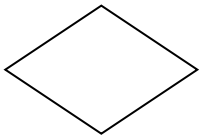
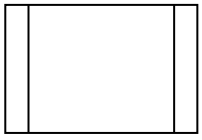
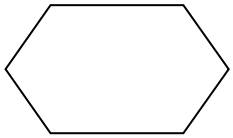
6. Кілька виходів із блоку Рішення слід показувати двома лініями від даного блоку до інших блоків

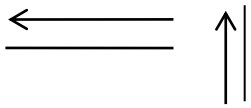
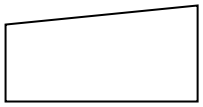
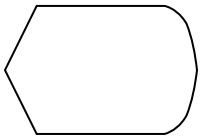
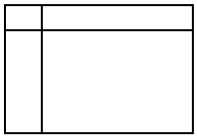
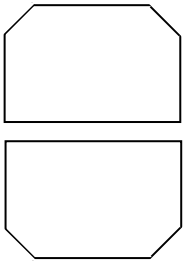
Довжину блоку треба вибирати з ряду 10, 15, 20 мм і можна збільшувати на число, кратне 5. Висота блоку в півтора раза менше довжини, крім блоків 1, 10, у яких висота дорівнює половині довжини.



Таблиця 1

Блоки схем алгоритмів

Найменування блоку	Графічне зображення	Функція блоку
1. Початок / Кінець		Блок відображає вихід у зовнішнє середовище й вхід із зовнішнього середовища (початок або кінець схеми)
2. Введення / Вивід		Блок відображає дані, носій даних не визначений
3. Обчислювальний		Блок відображає функцію обробки даних будь-якого виду (виконання певної операції або групи операцій, що приводить до зміни значення, форми або розміщення інформації)
4. Логічний		Блок відображає розв'язок або функцію **комутаційного** типу, що має один вхід і ряд альтернативних виходів, один і тільки один з яких може бути активізований після обчислення умов, визначених усередині цього блоку. Відповідні результати обчислення можуть бути записані по сусідству з лініями, що відображають ці шляхи.
5. Підпрограма		Блок відображає визначений процес, що складається з однієї або декількох операцій або кроків програми, які визначені в іншому місці
6. Переадрасація		Блок відображає модифікацію команди або групи команд із метою впливу на деяку наступну функцію (установка перемикача, модифікація індексного регістру або ініціалізація програми)
7. Коментар		Блок використовують для додавання описових коментарів або пояснювальних записів з метою пояснення або приміток. Пунктирні лінії в блоці коментаря пов'язані з відповідним блоком або можуть обводити групу блоків. Текст коментарів або приміток має бути поміщений біля обмежуючої фігури.

8. Лінія		Блок відображає потік даних або керування
9. З'єднувач		Блок відображає вихід у частину схеми й вхід з іншої частини цієї схеми й використовується для обриву лінії й продовження її в іншому місці. Відповідні блоки-з'єднувачі повинні містити одне і те ж саме унікальне позначення.
10. Ручне введення		Блок відображає дані, що вводяться вручну під час обробки із пристроїв будь-якого типу. *Будемо використовувати для позначення введення даних із клавіатури*
11. Дисплей		Блок відображає дані, представлені в формі, яка читається людиною, на носії у вигляді пристрою, що відображає інформацію (екран для візуального спостереження)
12. Документ		Блок відображає дані, представлені на носії в зручній для читання формі (машинограма, документ для оптичного або магнітного зчитування, рулон стрічки з підсумковими даними). *Будемо використовувати для позначення виводу даних на принтер*
13. Оперативний запам'ятовувальний пристрій		Блок відображає дані, що зберігаються в оперативному запам'ятовувальному пристрої.
14. Границя циклу		Блок, що полягає із двох частин, відображає початок і кінець циклу. Обидві частини блоку мають той самий ідентифікатор. Умови для ініціалізації, збільшення, завершення і т.ін. містяться усередині блоку на початку або наприкінці залежно від розташування операції, що перевіряє умову. *Будемо використовувати для позначення границь циклу*

У цілому спосіб запису алгоритму у вигляді схеми можна розглядати як певну алгоритмічну мову зі своєю системою позначень (словник мови) і правил для однотипного запису алгоритмів і їх виконання (синтаксис мови). Оскільки в навчальному курсі «Інформатика» студенти вирішують відносно нескладні завдання, то доцільно в навчальних цілях розробляти алгоритми з високим ступенем деталізації, що полегшує складання програми для розв'язку задачі на комп'ютері. Із цією метою необхідно уточнити (деталізувати) і «тлумачний» словник і синтаксис графічної мови

зображення алгоритму у вигляді схеми.

Блок Початок / Кінець уточнень не вимагає, будемо використовувати його для позначення початку й кінця процесу обробки даних.

Блок Введення / Вивід будемо використовувати для позначення послідовного введення або виводу даних, носій яких не визначений. Для конкретизації функції усередині блоку можна записувати слово Введення або Вивід, нижче якого будемо записувати список введення або виводу відповідно. Список введення або виводу складається з елементів, що відділяються один від одного символом «кома». Елементом списку введення може бути:

- ім'я простої змінної;
- ім'я змінної з індексом (елемент масиву).

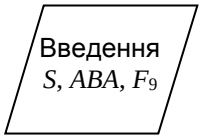
Елементом списку виводу може бути:

- ім'я простої змінної;
- ім'я змінної з індексом (елемент масиву);
- апостроф (послідовність символів, взята в лапки «»).

Під простою змінною, або просто змінною будемо розуміти деяку комірку пам'яті, тобто окреме місце для зберігання одного значення. В окремій комірці за час роботи алгоритму може побувати безліч різних констант (звідси назва – змінна). Такими комірками (електронними, магнітними, оптичними) оснащений комп'ютер. Змінні мають буквено-символьне позначення, наприклад, *S*, *ABA*, *F*, *n*, *a*1, *b*, *B*2. Водночас позначення змінної є адресою (номером, індексом) комірки, у якій будуть записуватися константи. Кожна з таких констант називається значенням змінної. Наприклад, *S* є змінною й адресою комірки *S* одночасно. З алгоритмічної точки зору поняття «змінна» і «адреса комірки» пам'яті є ідентичними.

Іншим різновидом змінних є так звана змінна з індексом або елемент масиву. Масив – це деяка сукупність комірок, об'єднана одним позначенням (іменем). Будь-який масив обов'язково має розмірність. Масиви бувають одномірними, двовимірними, тривимірними і т.ін. Для того щоб можна було відрізнити одну комірку масиву від іншої комірки цього ж масиву, їх нумерують. Нумерація комірок звичайно починається з 1 (необов'язково). Номер комірки масиву називається його індексом, а константа в комірці – елементом масиву.

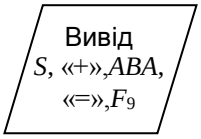
При введенні функцію блоку Введення / Вивід можна визначити наступними ключовими словами: **прочитати** дані з невизначеного носія інформації й **записати** (**запам'ятати**) у комірках



запам'ятовувального пристрою.

Наведений блок у схемі пропонує **прочитати** з носія дані в кількості трьох (нехай, наприклад, дані являють собою числові значення 11; 22; 33), причому, перше дане (11) **записати** (**запам'ятати**) у комірку з адресою *S*, друге (22) – у комірку з адресою *ABA*, третє (33) – у комірку з адресою F_9. Або іншими словами: **прочитати** з носія дані в кількості трьох, причому, перше дане (11) **запам'ятати** під іменем *S*, друге (22) – під іменем *ABA*, третє (33) – під іменем F_9. Тут F_9 – дев'ятий елемент одномірного масиву з іменем *F*.

При виводі функцію блокуВведення / Вивідможна визначити наступними ключовими словами: **прочитати** (витягнути) дані із комірок із зазначеними адресами запам'ятовувального пристрою й **відобразити** їх на невизначений носій інформації.



Наведений блок пропонує **прочитати** (викликати) із комірки з адресою *S* перше дане (11), із комірки з адресою *ABA* – друге дане (22), із комірки з адресою F_9 – третє дане (33) і **відобразити** їх на невизначеному носії інформації в зазначеному порядку разом з текстовими константами (у блоці виводу вони взяті в лапки). Або іншими словами: **прочитати** вміст комірок із адресою *S*, *ABA*, F_9 і **відобразити** його на невизначеному носії інформації в зазначеному порядку разом з текстовими константами. Або так: значення змінних *S*, *ABA*, F_9 **відобразити** на невизначеному носії інформації в зазначеному порядку разом з текстовими константами.

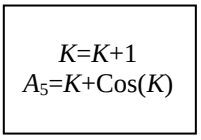
У тому випадку, якщо наведений блок виводу слідує в тій же схемі за попереднім блоком введення даних, результатом виконання зазначених дій є наступний запис: 11+22=33.

Не зайвим буде нагадати, що запис усередині блоку Введення / Вивід констант, покажчиків функцій, імен функцій і підпрограм, арифметичних і логічних виражень (наприклад, $X = A$, 10, $X+Yj+Zi$, $SIN(X)$, $Y<5$) буде трактуватися як помилка.

Блок Обчислювальний будемо використовувати для позначення виконання операції або групи операцій, у результаті яких змінюються значення. Функцію цього блоку можна трактувати ключовими словами: **обчислити значення** виразу, записаного праворуч від знака присвоювання (=), і **записати** обчислене значення в комірку, адреса якої зазначена ліворуч від знака присвоювання (=). Або іншими словами: **обчислити значення** виразу, записаного праворуч від знака присвоювання (=), і **запам'ятати** під іменем змінної (простої або з індексом), зазначеної ліворуч від знака присвоювання (=). Виходячи із цього, запис ліворуч від знака присвоювання замість імені змінної (простої або з індексом) будь-яких констант, покажчиків функцій, арифметичних і логічних виражень буде трактуватися як помилка.

Запис виду $Y = 5,5$ слід розуміти так: записати константу 5,5 у комірку з адресою Y (якщо до цієї операції в комірку вже була записана константа, то вона буде затерта константою 5,5). Розуміти й читати цей запис можна й так: змінній Y присвоїти значення 5,5.

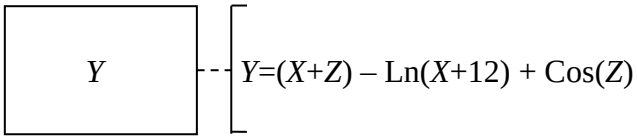
Запис виду $L = M$ слід розуміти так: прочитати константу, розташовану за адресою M, і скопіювати її в комірку з адресою L (при цьому константа із комірки M не видаляється й залишається такою ж, якою вона була до читання). Читати цей запис можна й так: змінній L



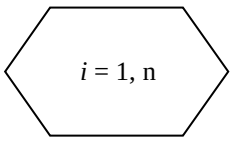
присвоїти значення змінної M (або просто: L присвоїти M).

Наприклад, при заданому значенні K= -1 блок, зображений вище, наказує значення змінної K збільшити на одиницю (-1+1=0) і отриманим значенням (0) замінити попереднє значення (-1), тобто в комірці з адресою K тепер зберігається значення 0. Другий оператор присвоювання наказує обчислити 0+Cos(0)=1 і отриманим значенням (1) замінити попереднє значення п'ятого елемента масиву з іменем A.

Блок Коментар уточнень не вимагає, будемо використовувати його для позначення зв'язку між елементом схеми й поясненням. Використання його доцільне й у тому випадку, якщо формули або написи повністю не поміщаються усередині блоку, наприклад:

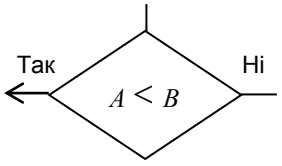


Блок Переадрасація будемо використовувати для позначення операцій з індексами, наприклад:



Блок Рішення використовується для позначення зміни напрямку виконання процесу залежно від деякої умови, записаної усередині блоку.

Функції блоку Рішення можна трактувати в такий спосіб: **обчислити** логічний вираз й здійснити перехід (**перейти**) на гілку у відповідності із значенням логічного виразу.



Пропонована інтерпретація не суперечить ДСТ, оскільки стандарт не поширюється на форму записів і позначень, що містяться усередині блоків або поруч із ними, та служать для уточнення виконуваних ними функцій.

Тут доречно нагадати деякі відомості про логічні вирази, записувані в якості умов усередині блоку Рішення. Логічний вираз – це правило для обчислення логічного значення. На відміну від арифметичного виразу, який має своє значення у вигляді числа, логічний вираз має своє значення у вигляді однієї із двох логічних констант – Так (Істина, True) чи Ні (Неправда, False), які так і називаються – логічні значення. Найпростішими логічними виразами є логічні константи, логічні змінні й відношення. Відношенням називається два арифметичні вирази, зв'язані одним із шести знаків операції порівняння: $<, \leq, =, \neq, >, \geq$. Наприклад, $x = 0$, $A \leq B$, $A / B - 3 = M$. Логічним значенням відношення є Так, якщо умова, виражена операцією порівняння, виконується, і – Ні, якщо ця умова не виконується.

Більш складні логічні вирази будуються за допомогою логічних операцій. Нижче наведені логічні операції й таблиця значень результатів їх виконання з операндами *A* і *B* логічного типу (результати обведені жирною рамкою).

Таблиця 2

<table>
<tr><th rowspan="2">Позначення операції</th><th rowspan="2">Назва операції</th><th rowspan="2">Приклад запису</th><th colspan="3">Таблиця результатів</th></tr>
<tr><td>A</td><td colspan="2">¬ A</td></tr>
<tr><td rowspan="3">¬</td><td rowspan="3">Логічне заперечення,
НЕ, інверсія</td><td rowspan="3">¬A</td><td>Так</td><td colspan="2">Ні</td></tr>
<tr><td>Ні</td><td colspan="2">Так</td></tr>
<tr><td rowspan="2">∧</td><td rowspan="2">Логічне множення,
І, кон'юнкція</td><td rowspan="2">A ∧ B</td><td rowspan="2">A</td><td colspan="2">B</td></tr>
<tr><td>Так</td><td>Ні</td></tr>
<tr><td rowspan="2">∨</td><td rowspan="2">Логічне додавання, АБО,
диз'юнкція</td><td rowspan="2">A ∨ B</td><td rowspan="2">A</td><td colspan="2">B</td></tr>
<tr><td>Так</td><td>Ні</td></tr>
<tr><td>Так</td><td>Так</td><td>Так</td><td>Так</td></tr>
<tr><td>Ні</td><td>Так</td><td>Так</td><td>Ні</td></tr>
</table>

Значення логічного виразу обчислюється зліва направо із урахуванням наступного пріоритету: арифметичні операції, операції відношення, логічні операції в послідовності НЕ, І, АБО.

Наприклад, необхідно визначити, чи належить точка M(X, Y) двозв'язній заштрихованій області (див. рисунок), що складається із прямокутника з вершинами (A, C), (A, D), (B, D), (B, C) і кола радіусом R.

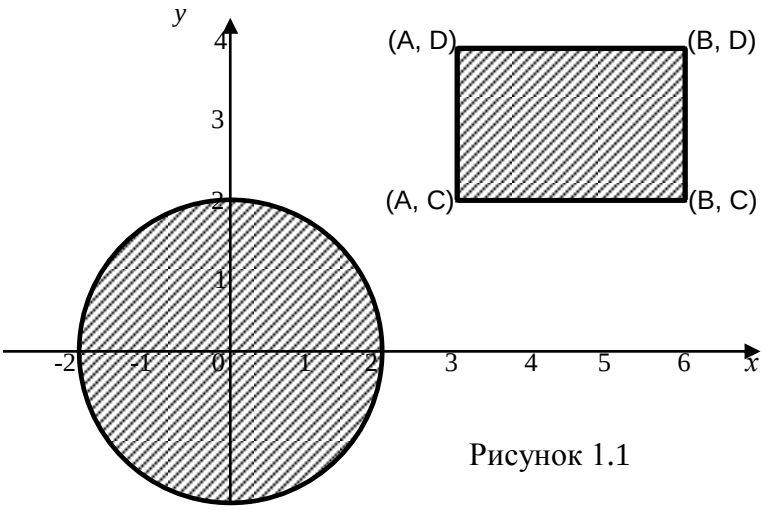


Рисунок 1.1

Для відповіді на поставлене питання необхідно записати наступний логічний вираз $X \geq A \wedge X \leq B \wedge Y \geq C \wedge Y \leq D \vee X^2 + Y^2 \leq R^2$ і обчислити його значення з урахуванням вищезгаданого пріоритету в зазначеному (числа над символами операцій) порядку

1 8 2 9 3 10 11 5 7 6

$X \geq A \wedge X \leq B \wedge Y \geq C \wedge Y \leq D \vee X^2 + Y^2 \leq R^2$.

Для довільних заданих значень параметрів A=3, B=6, C=2, D=4, R=2 обчислимо значення логічного виразу для трьох варіантів розташування точки M(X, Y), звівши дії в таблицю 3.

Таблиця 3

№ дії	X=1, Y=1 (точка належить колу)	X=2, Y=3 (точка не належить ні колу, ні прямокутнику)	X=4, Y=3 (точка належить прямокутнику)
1	1≥3 = Ні	2≥3 = Ні	4≥3 = Так
2	1≤6 = Так	2≤6 = Так	4≤6 = Так
3	1≥2 = Ні	3≥2 = Так	3≥2 = Так
4	1≤4 = Так	3≤4 = Так	3≤4 = Так
5	12+12 =2	22+32 =13	42+32 =25
6	22 = 4	22 = 4	22 = 4
7	2 ≤ 4 = Так	13 ≤ 4 = Ні	25 ≤ 4 Ні
8	Ні ∧ Так = Ні	Ні ∧ Так = Ні	Так ∧ Так = Так
9	Ні ∧ Ні = Ні	Ні ∧ Так = Ні	Так ∧ Так = Так
10	Ні ∧ Так = Ні	Ні ∧ Так = Ні	Так ∧ Так = Так
11	Ні ∨ Так = Так	Ні ∨ Ні = Ні	Так ∨ Ні = Так
ВІДПОВІДЬ	Належить області	Не належить області	Належить області

Порядок виконання дій, як і в математичних виразах, можна змінити проставлянням круглих дужок, причому, зайві пари дужок не є помилкою. Порядок обчислення усередині дужок виконується із урахуванням вищезгаданого пріоритету. Наприклад, можливі наступні записи

логічного виразу:

$$\begin{array}{l} 1\quad 3\quad 2\quad 10\quad 4\quad 6\quad 5\quad 11\quad 7\quad 9\ \ 8 \\ (X \geq A \wedge X \leq B) \wedge (Y \geq C \wedge Y \leq D) \vee (X^2 + Y^2 \leq R^2); \end{array}$$

$$\begin{array}{l} 1\quad 3\quad 2\quad 7\quad 4\quad 6\quad 5\quad 11\quad 7\quad 10\ \ 9 \\ ((X \geq A \wedge X \leq B) \wedge (Y \geq C \wedge Y \leq D)) \vee (X^2 + Y^2 \leq R^2); \end{array}$$

$$\begin{array}{l} 1\quad 3\quad 2\quad 7\quad 4\quad 6\quad 5\quad 11\quad 7\quad 10\ \ 9 \\ (((X \geq A) \wedge (X \leq B)) \wedge ((Y \geq C) \wedge (Y \leq D))) \vee ((X^2 + Y^2) \leq R^2). \end{array}$$

Правильне проставляння дужок не змінює остаточного значення логічного виразу. Пропонуємо переконатися в цьому, самостійно обчисливши значення логічного виразу в наведених вище варіантах запису.

Далі розглянемо складання схем різних видів обчислювальних процесів на конкретних прикладах.

2 ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ РІЗНИХ ВИДІВ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

Усі обчислювальні процеси можна представити у вигляді комбінації трьох основних складових:

- Лінійні процеси (або структури).

  У записі алгоритму підряд одна за одною написано кілька дій, які будуть виконуватися послідовно в такому ж порядку. Така конструкція в структурному програмуванні називається ПРОХОДЖЕННЯ.

- Структури, що розгалужуються.

  Умовна конструкція структурного програмування, що визначає розгалуження в порядку виконання дій. Називається ЯКЩО-ТО-ІНАКШЕ (дослівний англійський переклад IF-THEN-ELSE).

- Циклічні процеси (або структури).

  У структурному програмуванні передбачені циклічні конструкції трьох видів:

  1. Цикл із передумовою ПОКИ-ВИКОНУЙ (дослівний англійський переклад WHILE): поки істинна деяка умова, роби певні дії.
  2. Цикл із післяумовою ПОВТОРЮЙ-ПОКИ (дослівний англійський переклад DO-WHILE). Відрізняється від попереднього циклу тим, що тіло циклу повторюється не менш одного разу.
  3. Цикл із заздалегідь заданим числом повторень (FOR).

2.1 Графічне завдання лінійних обчислювальних процесів

Найпростішим обчислювальним процесом є лінійний, типова схема якого складається із запису трьох дій:

- введення початкових даних;
- обчислення за формулами;
- виведення результату.

Приклад 2.1.1 Після завершення виробництва деякого обладнання на заводському складі залишилися невикористаними комплектуючі вироби. Потрібно визначити загальну вартість невикористаних виробів

Розв'язок

1. Постановка задачі.

Результатом розв'язку є загальна вартість невикористаних виробів. Враховуючи спрощуюче допущення про те, що на складі перебувають вироби, які мають однакову вартість, для визначення результату повинні бути задані вартість 1 шт. виробу й кількість виробів (у шт.). Інших початкових даних для розв'язку не потрібно.

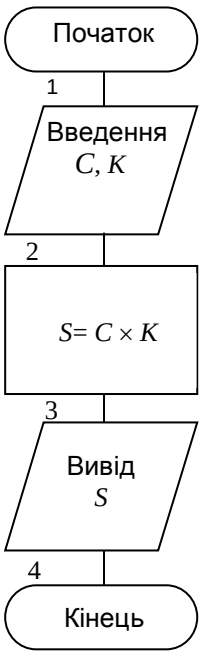
2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Ціна 1 шт. виробу	Дійсний	*C*	Початкове дане
Кількість виробів	Цілий	*K*	Початкове дане
Загальна вартість виробів	Дійсний	*S*	Результат

Таким чином, математичне формулювання завдання зводиться до обчислення за очевидною формулою $S = C \times K$.

3. Розробка алгоритму розв'язку задачі.



Оскільки отримана математична модель являє собою найпростіший арифметичний вираз, то ніяких спеціальних методів для реалізації моделі не потрібно.

Схема алгоритму складається із блоків, наведених у таблиці 1.

Усередині блоку Введення (блок 1) через кому записуються імена змінних *C* і *K*, зазначених у стовпчику Призначення таблиці імен змінних як початкові дані.

У блок 2 записується формула.

Примітка. Тут доречно нагадати, що знак «=» в інформатиці, крім операції порівняння двох величин, має й інше призначення. Він позначає оператор присвоювання й читається як «присвоїти», а розуміється як нове

значення змінної, яка стоїть ліворуч від знака «=». Для того щоб оператор присвоювання не плутати з операцією порівняння, його іноді позначають у вигляді «:=». У блок 3 записується ім'я *S* змінної-результату.

Перевіримо правильність алгоритму на довільних конкретних значеннях початкових даних:

Блок	Дія
	Початок
1	Введення: 5,2; 4
2	$S = 5{,}2 \times 4 = 20{,}8$
3	Вивід: 20,8
4	Кінець

Отриманий результат (20,8) збігається з очікуваним, обчисленим вручну для цього найпростішого алгоритму, отже, схема алгоритму розроблена вірно.

Надалі всі приклади алгоритмів будуть реалізовані трьома мовами програмування, C++, Visual Basic та Turbo Pascal, з коментарями там, де це необхідно.

C++

```
#include<iostream> //бібліотека для роботи з функціями введення/виводу
#include<clocale>//бібліотека для роботи з локалями
using namespace std; /*підключення простору імен std, у якому перебувають функції
введення/виводу*/
int main() //опис функції
{
  double c, s; //оголошення змінних дійсного типу
  int k; //оголошення змінної цілочисельного типу
  setlocale(LC_ALL, "UKR"); /* підключення локалей для відображення кирилиці*/
  cout<<"Введіть значення ціни за 1 шт. виробу "; /* вивід повідомлення на екран*/
  cin>>c; // запис значення в змінну c
  cout<<"Введіть кількість виробів "; // вивід повідомлення на екран
  cin>>k; // запис значення в змінну k
  s=k*c; //у змінну s записується добуток змінних k і c
  cout<<"Загальна вартість виробів = "<<s<<endl; //вивід значення змінної s
system("pause"); //системна затримка для перегляду результату
return 0; // те, що повертає функція
}
```

VB

```
Private Sub Command1_Click() ' оголошення процедури обробки натискання кнопки
  Dim s, c As Single, k As Integer ' оголошення змінних s,c дійсного типу та k - цілого типу
  c = CSng(Text1) ' запис у змінну c значення, зчитаного з форми з текстового поля Text1
  та перетвореного у дійсний тип
  k = CInt(Text2) ' запис у змінну k значення, зчитаного з форми з текстового поля Text2
  та перетвореного у цілий тип
  s = c * k ' запис у змінну s значення добутку змінних c та k
```

```
  Text3 = s ' виведення на форму у текстове поле Text3 значення змінної s
End Sub ' кінець процедури
```

TP

```
Program p_2_1_1; { заголовок програми }
Var { блок оголошення змінних }
  k: integer; { оголушується змінна k цілого типу }
  c,s:real; { оголушуються змінні s та c дійсного типу }
Begin { початок тіла програми }
  Writeln('Введіть значення вартості за одиницю виробу'); { Виведення тексту на екран }
  Readln(c); { зчитування з клавіатури та запис значення в змінну c }
  Writeln('Введіть кількість виробів');
  Readln(k);
  s = c * k  { змінній s присвоюється значення добутку змінних k та c }
  Writeln (s); { виведення значення змінної s на екран }
End. { кінець тіла програми }
```

Приклад 2.1.2 Необхідно розділити групу, що складається з *N* студентів, на дві приблизно рівні за кількостю підгрупи.

Розв'язок

1. Постановка задачі.

Результатом розв'язку є кількість студентів у першій і другій підгрупах. Для визначення результату повинна бути задана кількість студентів у групі.

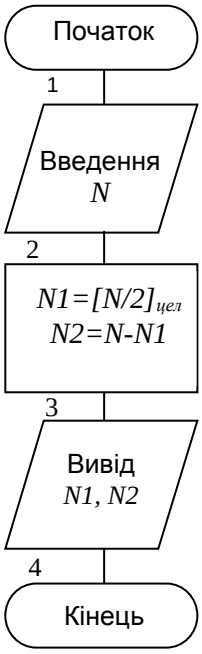
2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Кількість студентів у групі	Цілий	*N*	Початкове дане
Кількість студентів у першій підгрупі	Цілий	*N1*	Результат
Кількість студентів у другій підгрупі	Цілий	*N2*	Результат

Таким чином, математичне формулювання задачі зводиться до цілочисельного ділення кількості людей у групі на два й подальшому відніманні із загальної кількості результату ділення: N-$[N/2]_{\text{цілий}}$.

3. Розробка алгоритму розв'язку задачі.



C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  int N,N1,N2;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть кількість студентів у групі ";
  cin>>N;
  N1=N/2;/*цілочисельне ділення,так як обидва аргументи цілого типу*/
  N2=N-N1;/* від загальної кількості віднімаємо кількість студентів у першій групі*/
  cout<<"Кількість студентів у перші і другій групах відповідно: "<<N1<<N2<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
   Dim n, n1, n2 As Integer
' введення з форми вихідних даних та перетворення їх у цілочисельний тип
   n = CInt(Text1)
   n1 = n \ 2 ' запис у змінну n1 значення цілочисельного ділення змінної n навпіл
   n2 = n - n1
   Text2 = n1
   Text3 = n2
End Sub
```

TP

```
Program p_2_1_2;
Var
  n, n1, n2: integer;
Begin
  Writeln('Введіть кількість студентів у групі');
  Readln(n);
  n1 = n div 2  { запис у змінну n1 значення цілочисельного ділення змінної n навпіл }
  n2 = n - n1
  Writeln (n1, n2);
End.
```

2.2 Графічне завдання обчислювальних процесів, що розгалужуються

При розв'язку різних задач часто доводиться вибирати одну дію з декількох можливих. Така ситуація реалізується обчислювальним процесом, що розгалужується.

Обчислювальний процес називається процесом, що розгалужується, якщо для одержання кінцевого результату передбачається вибір одного з декількох можливих напрямків обчислень (гілок) залежно від результату перевірки проміжної умови.

Напрямок обчислень вибирається у відповідності зі значенням умови, записаної усередині блоку Рішення. Нагадаємо, що можливі два значення умови й два відповідні виходи: Так (Істина) – умова виконана; Ні (Неправда) – умова не виконана.

Обчислювальний процес, що розгалужується, у загальному вигляді зображується схемою, наведеною на рисунку 2.1*а*, і в окремому випадку – схемою, наведеною на рисунку 2.1*б*.

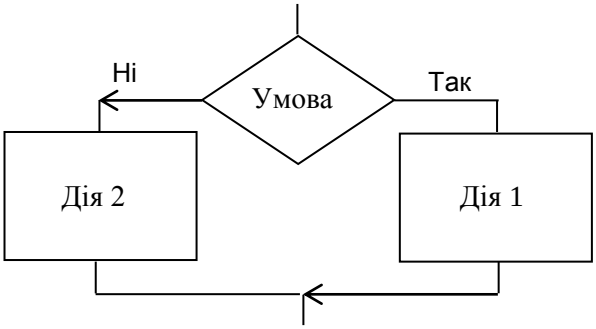


Рисунок 2.1*а*

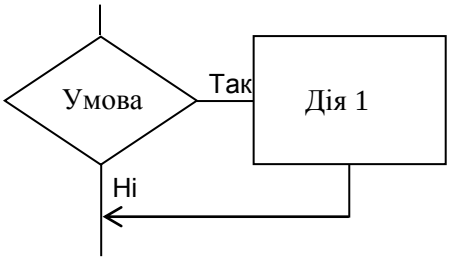


Рисунок 2.1*б*

Розглянемо декілька прикладів складення схеми алгоритма розв'язання простих задач.

Приклад 2.2.1 Відомо, що два студенти мають неоднаковий зріст. Визначити зріст вищого студента.

Розв'язок

1. Постановка задачі.

Результатом розв'язку задачі є значення зросту більш високого студента. Для визначення результату повинні бути задані значення зросту кожного із двох студентів. Інших початкових даних для розв'язку не потрібно.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане
Зріст другого студента	Дійсний	*S*2	Початкове дане

Математичне формулювання задачі полягає у визначенні максимального числа із двох заданих нерівних чисел.

3. Розробка алгоритму розв'язку задачі.

Розглянемо кілька можливих алгоритмів розв'язку задачі.

Алгоритм 1 Перше, що спадає на думку, – це реалізація очевидної формули

*Якщо S*1 > *S2, то вивести S1, інакше вивести S2* .

Ідея цього алгоритму полягає в наступному: шляхом порівняння двох значень визначається більше й виводиться в якості результату.

У цьому варіанті розв'язку змінні *S1* і *S2* є початковими даними й результатами, тому в стовпчику Призначення таблиці імен змінних необхідно це відзначити.

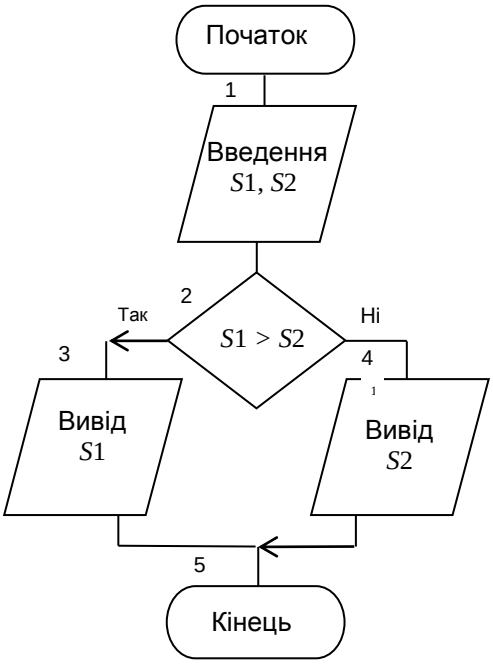
Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане, результат
Зріст другого студента	Дійсний	*S*2	Початкове дане, результат

Оскільки отримана математична модель являє собою найпростішу логічну умову, то ніяких спеціальних методів для реалізації моделі не потрібно.

У блок Введення (блок 1 для введення початкових даних) через кому записуються імена змінних *S*1 і *S*2, зазначених у таблиці імен змінних як початкові дані.

У блок Рішення (блок 2) записується логічна умова, що реалізує порівняння першого числа із другим.

Для виводу більшого значення, визначеного у блоці 2, у блок 3 записується ім'я змінної *S*1 (гілка Так), або у блок 4 записується, відповідно, ім'я змінної *S*2 (гілка Ні).



Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях початкових даних. Виконання дій по блок-схемі оформимо у вигляді таблиці. Незалежно від порядку введення значень (перше більше другого, або, навпаки, друге більше першого), при виводі необхідно одержати більше значення.

Блок	Варіант 1	Варіант 2
	Початок	Початок
1	Введення: 175; 164	Введення: 164; 175
2	175>164 Так	164>175 Ні
3	Вивід: 175	
4		Вивід: 175
5	Кінець	Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double s1, s2;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть значення зросту першого студента ";   cin>>s1;
  cout<<"Введіть значення зросту другого студента ";   cin>>s2;
          if(s1>s2) cout<<s1; //якщо s1більше s2,  виводиться s1
                    else cout<<s2; //інакше – s2
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
  Dim s1, s2 As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
```

```
  s1 = CSng(Text1)
  s2 = CSng(Text2)
  If s1 > s2 Then ' якщо s1 більше за s2, тоді...
  Text3 = s1 ' на форму в текстове поле виводиться значення s1
  Else ' інакше...
  Text3 = s2 ' на форму в текстове поле виводиться значення s2
  End If ' кінець умовного блоку If
End Sub
```

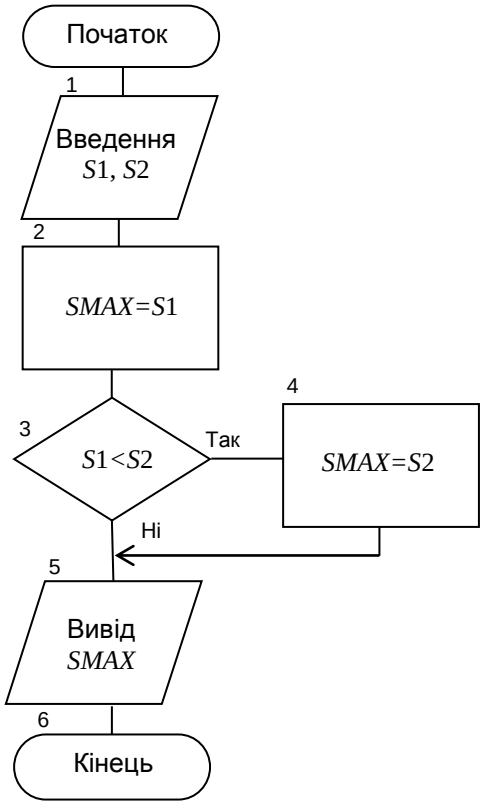
TP

```
Program p_2_2_1_a;
  var s1,s2:real;
begin
  writeln('Введіть зріст першого та другого студента');
  readln(s1,s2);
  if s1>s2 then writeln (s1) else writeln (s2); {якщо s1 більше s2,  виводиться s1, інакше – s2}
End.
```

Алгоритм 2

Структура даних – колишня.

Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане
Зріст другого студента	Дійсний	*S*2	Початкове дане
Найбільший зріст	Дійсний	*SMAX*	Результат



У блок 1 записуються імена змінних *S*1 і *S*2, що є початковими даними.

У блок 2 записується присвоювання змінній *SMAX* значення змінної *S*1.

У блоці 3 виконується присвоювання змінній *SMAX* значення змінної *S*1.

У блоці 4 виконується присвоювання змінній *SMAX* нового більшого значення *S*2.

У блоці 5 для виводу більшого значення вказується ім'я змінної-результату *SMAX*.

Перевіримо правильність алгоритму на конкретних значеннях, використаних у попередньому тестовому прикладі:

Блок	Варіант 1	Варіант 2
	Початок	Початок
1	Введення: 175; 164	Введення: 164; 175
2	*SMAX*=175	*SMAX*=164
3	175<164 Ні	164<175 Так
4		*SMAX*=175
5	Вивід: 175	Вивід: 175
6	Кінець	Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму розроблена вірно.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double S1,S2,SMAX;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть зрість першого і другого студента ";
  cin>>S1>>S2;
  SMAX=S1;//SMAX присвоюємо значення S1
  if(S1<S2) SMAX=S2;//якщо S1 менше S2, то в SMAX записати S2
  cout<<"Найбільший зріст = "<<SMAX<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim s1, s2, smax As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
s1 = CSng(Text1)
s2 = CSng(Text2)
smax = s1
If s1 < s2 Then smax = s2 ' якщо s1< s2, тоді змінній smax присвоюється значення s2
Text3 = smax
End Sub
```

TP

```
Program p_2_2_1_b;
  var s1,s2,smax:real;
begin
  writeln('Введіть зріст першого та другого студентів');
  readln(s1,s2);
  smax:=s1;
  if s1<s2 then smax:=s2;{якщо s1 менше за s2,smax присвоюємо значення  s2}
  writeln (smax);
  readln
 End.
```

У практиці обчислень часто виникають задачі, у яких з декількох альтернативних варіантів за відомими параметрами, що задовольняють деяким умовам, необхідно вказати задані атрибути, пов'язані із цими параметрами

Розглянемо кілька прикладів складання схеми алгоритму розв'язку подібних задач, використовуючи викладені вище ідеї пошуку максимального (мінімального) значення із двох заданих.

Приклад 2.2.2 Відомо, що два студенти мають неоднаковий зріст. Вказати ім'я більш високого студента.

Розв'язок

1. Постановка задачі.

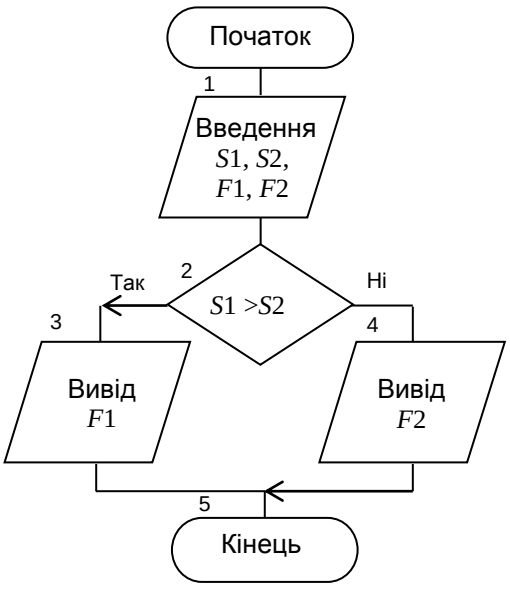
Результатом розв'язку задачі є не значення зросту, а ім'я більш високого студента. Для визначення результату повинні бути задані значення зросту й ім'я кожного із двох студентів. Інших початкових даних для розв'язку не потрібно.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане
Зріст другого студента	Дійсний	*S*2	Початкове дане
Ім'я першого студента	Строковий	*F*1	Початкове дане, результат
Ім'я другого студента	Строковий	*F*2	Початкове дане, результат

3. Розробка алгоритму розв'язку задачі.



У блок 1 записуються імена змінних, що є початковими даними.

У блок 2 записується логічна умова. У блок 3 записується ім'я змінної-результату *F*1, а в блок 4 – *F*2.

Перевіримо правильність алгоритму на конкретних значеннях: у Нікіфорова зріст 175 см, а в

Ніколаєнко – 164 см. Очевидно, що більш високий – Нікіфоров.

Блок	Варіант 1	Варіант 2
	Початок	Початок
1	Введення: 175; 164; Нікіфоров; Ніколаєнко	Введення: 164; 175; Ніколаєнко; Нікіфоров
2	175>164 Так	164>175 Ні
3	Вивід: Нікіфоров	
4		Вивід: Нікіфоров
5	Кінець	Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
#include<string>/*бібліотека для роботи із строковими даними типу string*/
using namespace std;
int main()
{
  double S1,S2;
  string F1,F2;//оголошення змінних типу string
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть зрість першого і другого студента ";
  cin>>S1>>S2;
  cout<<"Введіть фамілії цих студентів відповідно ";
  getline(cin,F1);//введення змінних типу string
  getline(cin,F2);
  if(S1>S2) cout<<"Найвищій студент = "<<F1<<endl;
  cout<<"Найвищій студент  = "<<F2<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim s1, s2 As Single, f1, f2 As String ' змінні f1, f2 оголешені текстового типу
' введення з форми вихідних даних та перетворення їх у дійсний та текстовий типи
s1 = CSng(Text1)
s2 = CSng(Text2)
f1 = Text3
f2 = Text4
If s1 > s2 Then
Text5 = f1
Else
Text5 = f2
End If
End Sub
```

TP

```
Program p_2_2_2;
  var s1,s2:real;
  f1,f2:string; {оголошення змінних текстового типу}
begin
  writeln('Введіть прізвище першого студента');
  readln (f1);
  writeln('Введіть його зріст ');
  readln(s1);
  writeln('Введіть прізвище другого студента');
  readln (f2);
  writeln('Введіть його зріст ');
  readln(s2);
  if s1>s2 then
  writeln (f1)
  else writeln (f2); {якщо s1 більше s2,  виводиться прізвище f1, інакше – f2}
End.
```

Для закріплення викладених ідей алгоритмів пошуку максимального (мінімального) значення із двох заданих чисел розглянемо кілька прикладів складання схеми алгоритму пошуку максимального (мінімального) значення для трьох заданих чисел.

Приклад 2.2.3 Відомо, що три студенти мають неоднаковий зріст. Визначити: а) зріст більш високого студента; б) ім'я більш високого студента.

Розв'язок задачі а)

1. Постановка задачі.

Результатом розв'язку задачі а) є значення зросту більш високого студента. Для визначення результату повинні бути задані значення зросту кожного із трьох студентів. Інших початкових даних для розв'язку не потрібно.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

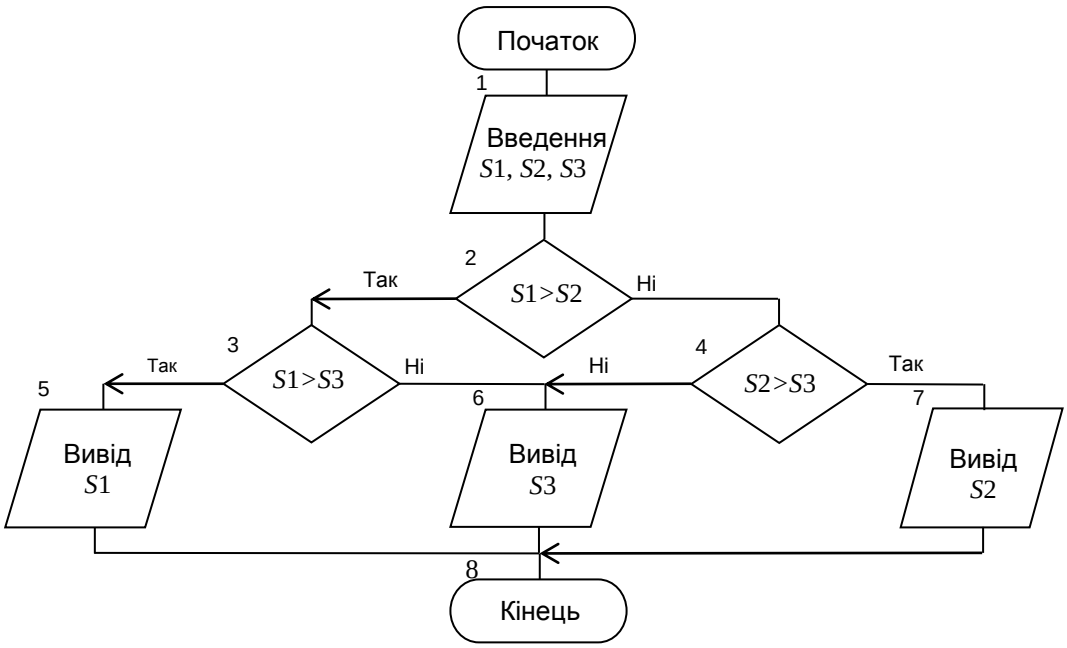
Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане, результат
Зріст другого студента	Дійсний	*S*2	Початкове дане, результат
Зріст третього студента	Дійсний	*S*3	Початкове дане, результат

3. Розробка алгоритму розв'язку задачі.

Розглянемо кілька можливих алгоритмів розв'язку задачі.

Алгоритм 1.а

У цьому варіанті розв'язку змінні *S1*, *S2* і *S3* є початковими даними й результатами, тому в стовпчику Призначення таблиці імен змінних необхідно це відзначити.



C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double S1,S2,S3;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть зрість першого,другого та  третього студента ";
  cin>>S1>>S2>>S3;
  if(S1>S2)//якщо S1 більше S2
  {
          if(S1>S3) cout<<"Найбільший зріст = "<<S1<<endl;/*якщо S1 більше S3, то вивести
S1*/
          else cout<<"Найбільший зріст = "<<S3<<endl;//інакше - S3
  }
  else if (S2>S3) cout<<"Найбільший зріст = "<<S2<<endl;/*інакше якщо S2 більше S3, то
вивести S2 */
  else cout<<"Найбільший зріст = "<<S3<<endl;//інакше - S3
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim s1, s2, s3 As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
s1 = CSng(Text1)
s2 = CSng(Text2)
s3 = CSng(Text3)
```

```
If s1 > s2 Then ' якщо s1 більше s2, тоді
 If s1 > s3 Then ' якщо s1 більше s3, тоді
 Text4 = s1 ' на форму в текстове поле виводиться значення s1
 Else ' інакше
 Text4 = s3 ' на форму в текстове поле виводиться значення s3
 End If ' кінець вкладеного умовного блоку
ElseIf s2 > s3 Then ' інакше, якщо s2 > s3
 Text4 = s2 ' на форму в текстове поле виводиться значення s2
 Else' інакше
 Text4 = s3 ' на форму в текстове поле виводиться значення s3
End If ' кінець умовного блоку
End Sub
```

TP

```
Program p_2_2_3_a;
  var s1,s2,s3:real;
begin
  writeln('Введіть зріст першого, другого та третього студентів');
  readln(s1,s2,s3);
  if s1>s2 then
  begin  {якщо s1 більше s2, то}
    if s1>s3 then writeln(s1); {якщо s1 більше s3 то виводиться зріст s1}
  end
  else if s2>s3 then writeln(s2) {інакше, якщо s2 більше s3, то виводиться зріст s2}
  else writeln (s3); {інакше виводиться зріст s3}
End.
```

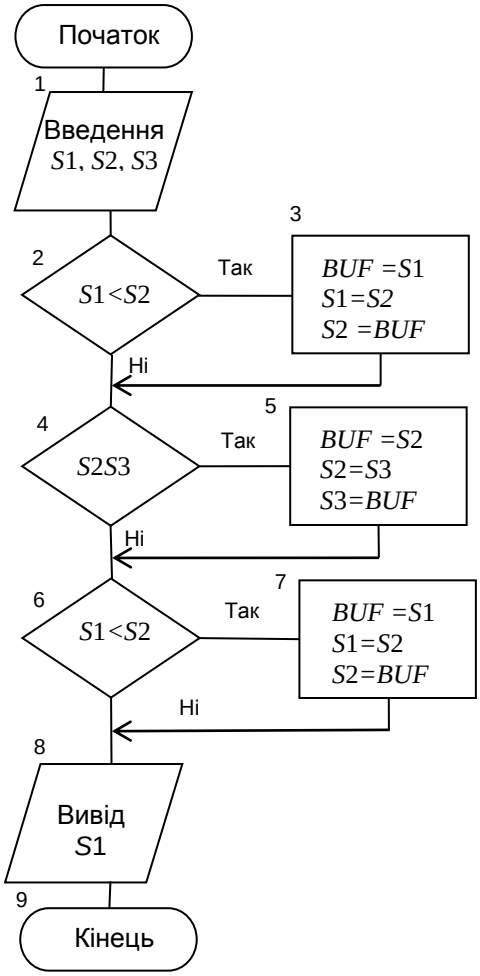
Алгоритм 2.а

1. Постановка задачі.

2. Побудова математичної моделі.

Складемо таблицю імен змінних для розв'язку задачі а) і задачі б) за допомогою алгоритму сортування трьох чисел за спаданням.

Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане, результат
Зріст другого студента	Дійсний	*S*2	Початкове дане
Зріст третього студента	Дійсний	*S*3	Початкове дане
Ім'я першого студента	Строковий	*F*1	Початкове дане, результат
Ім'я другого студента	Строковий	*F*2	Початкове дане
Ім'я третього студента	Строковий	*F*3	Початкове дане
Буферна змінна для обміну числовими значеннями	Дійсний	*BUF*	Допоміжна
Буферна змінна для обміну строковими значеннями	Строковий	*BUFF*	Допоміжна



Опишемо ідею алгоритму сортування чисел, наприклад, у порядку спадання. Спочатку порівняємо перше число із другим, обмінюючи їх місцями з використанням допоміжної змінної *BUF*, якщо вони не впорядковані за спаданням. Такі ж самі дії виконуємо з другим і третім числами, в результаті чого найменше число опиниться на останньому місці. Для наступного перегляду кількість неупорядкованих чисел стала на одиницю меншою, тобто два. У результаті повторного порівняння першого й другого чисел усі три числа будуть упорядковані за спаданням.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
 double S1,S2,S3,BUF;
 setlocale(LC_ALL, "UKR");
 cout<<"Введіть зріст першого,другого та  третього студента ";
 cin>>S1>>S2>>S3;
 if(S1<S2)
 {
        BUF=S1;
        S1=S2;
        S2=BUF;
 }
 if(S2<S3)
 {
        BUF=S2;
        S2=S3;
        S3=BUF;
 }
  if(S1<S2)
```

```
  {
            BUF=S1;
            S1=S2;
            S2=BUF;
  }

          cout<<"Найбільший зріст = "<<S1<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim s1, s2, s3, buf As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
s1 = CSng(Text1)
s2 = CSng(Text2)
s3 = CSng(Text3)
If s1 < s2 Then ' якщо s1 меньше s2, тоді - зміна місцями значень s1 та s2 через буферну змінну
buf = s1
s1 = s2
s2 = buf
End If
If s2 < s3 Then ' якщо s2 меньше s3, тоді - зміна місцями значень s2 та s3 через буферну змінну
buf = s2
s2 = s3
s3 = buf
End If
If s1 < s2 Then ' якщо s1 меньше s2, тоді - зміна місцями значень s1 та s2 через буферну змінну
buf = s1
s1 = s2
s2 = buf
End If
Text4 = s1 ' на форму в текстове поле виводиться значення змінної, що знаходиться на першому місці в відсортованій за зменшенням послідовності
End Sub
```

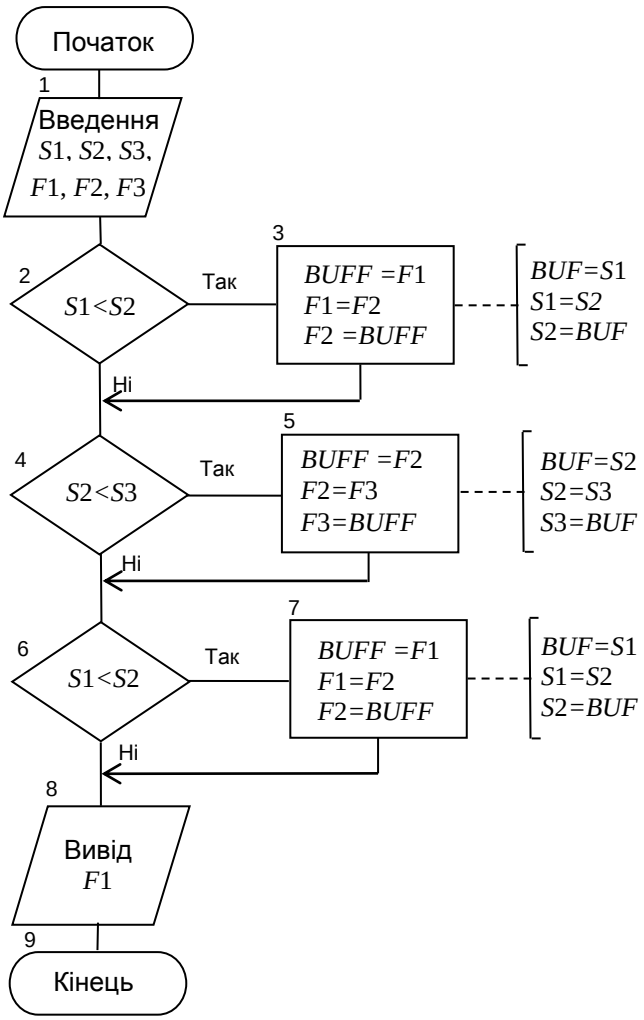
TP

```
Program p_2_2_3_b;
  var s1,s2,s3,buf:real;
begin
  writeln('Введіть зріст першого, другого та третього студентів');
  readln(s1,s2,s3);
  if s1<s2 then {якщо s1 меньше s3, тоді..}
  begin {зміна місцями значень s1 та s2 через буферну змінну}
      buf:=s1;
```

```
    s1:=s2;
    s2:=buf;
  end;
  if s2<s3 then {якщо s2 меньше s3 тоді…}
  begin  {зміна місцями значень s2 та s3 через буферну змінну}
    buf:=s2;
    s2:=s3;
    s3:=buf;
  end;
  if s1<s2 then {якщо s1 меньше s2 тоді…}
  begin {зміна місцями значень s1 та s2 через буферну змінну}
    buf:=s1;
    s1:=s2;
    s2:=buf;
  end;
  writeln (s1); {виводиться значення змінної, що знаходиться на першому місці в відсортованій за зменшенням послідовності }
End.
```

Алгоритм 2.б

Для розв'язку задачі б) у блок-схему Алгоритму 2.а необхідно внести зміни, результат яких представлений в наступній блок-схемі.



C++

```
#include<iostream>
#include<clocale>
#include<string>
using namespace std;
int main()
{
 double S1,S2,S3,BUF;
 string F1,F2,F3,BUFF;
 setlocale(LC_ALL, "UKR");
 cout<<"Введіть зріст першого,другого та  третього студента ";
 cin>>S1>>S2>>S3;
 cout<<"Введіть фамілії першого,другого та  третього студента ";
 getline(cin,F1);
 getline(cin,F2);
 getline(cin,F3);
 if(S1<S2)
 {
        BUFF=F1;
        F1=F2;
        F2=BUFF;
        BUF=S1;
        S1=S2;
        S2=BUF;
 }
 if(S2<S3)
 {
        BUFF=F2;
        F2=F3;
        F3=BUFF;
        BUF=S2;
        S2=S3;
        S3=BUF;
 }
  if(S1<S2)
 {
        BUFF=F1;
        F1=F2;
        F2=BUFF;
        BUF=S1;
        S1=S2;
        S2=BUF;
 }

      cout<<"Студент з найбільшим зростом = "<<F1<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim s1, s2, s3, buf As Single, f1, f2, f3, buff As String
' введення з форми вихідних даних та перетворення їх у дійсний та текстовий типи
s1 = CSngь(Text1)
s2 = CSng(Text2)
s3 = CSng(Text3)
f1 = Text4
f2 = Text5
f3 = Text6
If s1 < s2 Then ' якщо s1 меньше s2, тоді міняються місцями прізвища першого та другого студентів та …
buff = f1
f1 = f2
f2 = buff
buf = s1 ' міняються місцями s1 та s2
s1 = s2
s2 = buf
End If
If s2 < s3 Then ' якщо s2 меньше s3, тоді міняються місцями прізвища третього та другого студентів та …
buff = f2
f2 = f3
f3 = buff
buf = s2 ' міняються місцями s3 та s2
s2 = s3
s3 = buf
End If
If s1 < s2 Then ' якщо s1 меньше s2, тоді міняються місцями прізвища першого та другого студентів та …
buff = f1
f1 = f2
f2 = buff
buf = s1 ' міняються місцями s1 та s2
s1 = s2
s2 = buf
End If
Text7 = f1
```

TP

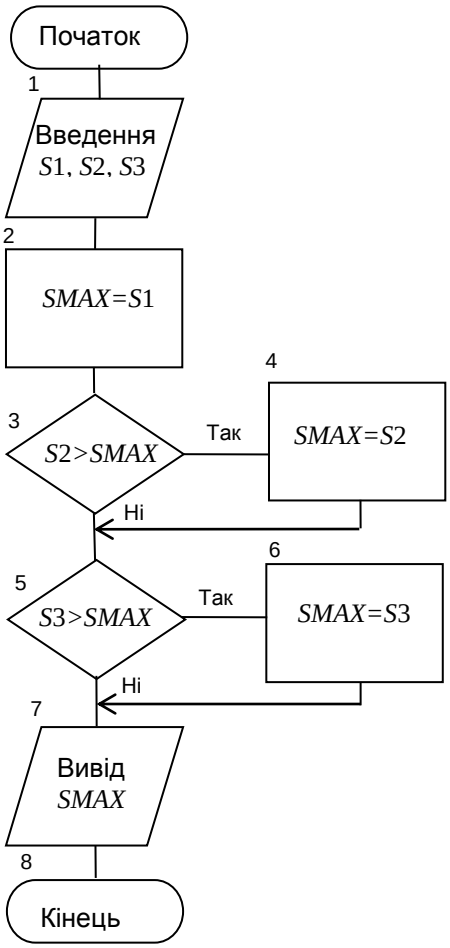
```
Program p_2_2_3_c;
  var s1,s2,s3,buf:real;
  f1,f2,f3,buff:string;
begin
  writeln('Введіть прізвище першого студента');
  readln(f1);
  writeln('Введіть його зріст');
  readln(s1);
  writeln('Введіть прізвище другого студента');
  readln(f2);
```

```
  writeln('Введіть його зріст');
  readln(s2);
  writeln('Введіть прізвище третього студента');
  readln(f3);
  writeln('Введіть його зріст');
  readln(s3);
  if s1<s2 then {якщо s1 меньше s2, тоді…}
  begin  { міняються місцями прізвища першого та другого студентів}
    buff:=f1;
    f1:=f2;
    f2:=buff;
    buf:=s1; { міняються місцями s1 та s2}
    s1:=s2;
    s2:=buf;
  end;
  if s2<s3 then {якщо s2 меньше s3 тоді…}
  begin  { міняються місцями прізвища третього та другого студентів}
    buff:=f2;
    f2:=f3;
    f3:=buff;
    buf:=s2; { міняються місцями s2 та s3}
    s2:=s3;
    s3:=buf;
  end;
  if s1<s2 then {якщо s1 меньше s2 то}
  begin { міняються місцями прізвища, що знаходяться на першому та другому місцях в послідовності}
    buff:=f1;
    f1:=f2;
    f2:=buff;
    buf:=s1; { міняються місцями s1 та s2}
    s1:=s2;
    s2:=buf;
  end; writeln (f1); {виводиться прізвище, що знаходиться на першому місці в послідовності}
End.
```

Алгоритм 3.а

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Зріст першого студента	Дійсний	*S*1	Початкове дане
Зріст другого студента	Дійсний	*S*2	Початкове дане
Зріст третього студента	Дійсний	*S*3	Початкове дане
Найбільший зріст	Дійсний	*SMAX*	Допоміжна
Ім'я першого студента	Строковий	*F*1	Початкове дане
Ім'я другого студента	Строковий	*F*2	Початкове дане
Ім'я третього студента	Строковий	*F*3	Початкове дане
Ім'я студента з більшим зростом	Строковий	*FMAX*	Результат



C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double S1,S2,S3,SMAX;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть зріст першого,другого та  третього студента ";
  cin>>S1>>S2>>S3;
  SMAX=S1;
  if(S2>SMAX) SMAX=S2;
  if(S3>SMAX) SMAX=S3;
        cout<<"Найбільший зріст = "<<SMAX<<endl;
system("pause");
return 0;
}
```

VB

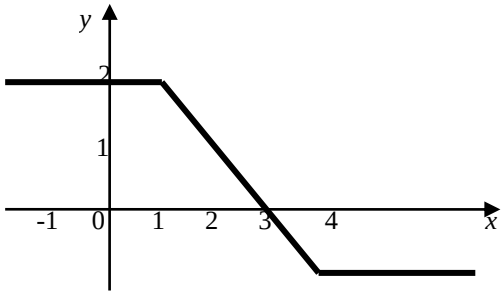
```
Private Sub Command1_Click()
Dim s1, s2, s3, smax As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
```

```
s1 = CSng(Text1)
s2 = CSng(Text2)
s3 = CSng(Text3)
smax = s1 ' змінній smax привласнюється значення s1
If s2 > smax Then smax = s2 ' якщо s2 більше за smax, тоді змінній smax привласнюєтьс значення s2
If s3 > smax Then smax = s3 ' якщо s3 більше за smax, тоді змінній smax привласнюєтьс значення s3
Text4 = smax  ' виведення на форму в текстове поле найбільшого зросту smax
End Sub
```

TP

```
Program p_2_2_3_d;
  var s1,s2,s3,smax:real;
begin
  writeln('Введіть зріст першого, другого та третього студентів');
  readln(s1,s2,s3);
  smax:=s1; { змінній smax привласнюється значення s1 }
  if s2>smax then smax:=s2; { якщо s2 більше за smax, тоді змінній smax привласнюєтьс значення s2 }
  if s3>smax then smax:=s3; {якщо s3 більше за smax тоді змінній smax привласнюєтьс значення s3 }
  writeln (smax); { виведення найбільшого зросту smax }
End.
```

Приклад 2.2.4 Скласти алгоритм обчислення значення функції, заданої графіком, для будь-якого заданого значення аргументу.



Розв'язок

1. Постановка задачі.

Результатом розв'язку є значення функції. Для його визначення повинне бути задане значення аргументу. Інших початкових даних для розв'язку не потрібно.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Значення аргументу	Дійсний	X	Початкове дане
Значення функції	Дійсний	Y	Результат

Значення функції визначається в такий спосіб:

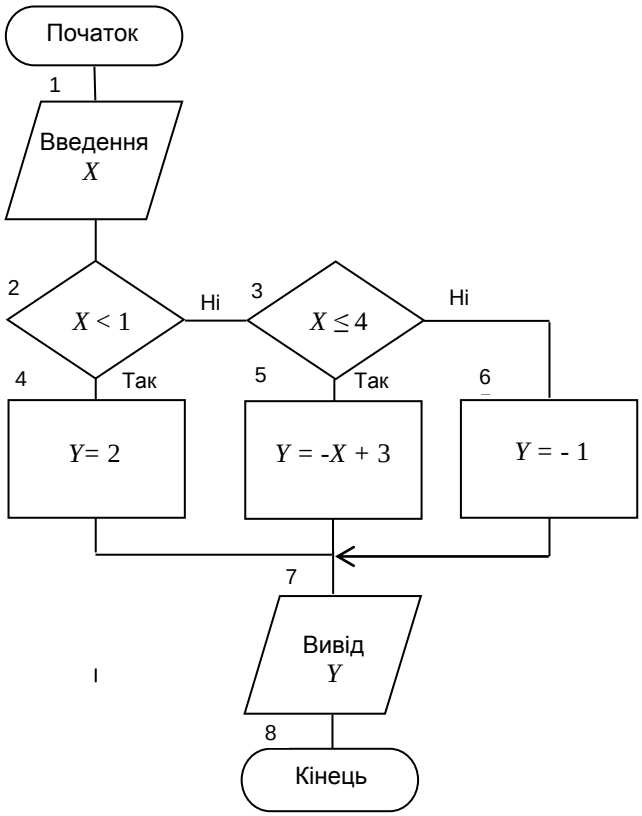
$$Y = \begin{cases} 2, & \text{якщо } X < 1, \\ -X + 3, & \text{якщо } 1 \le X \le 4, \\ -1, & \text{якщо } X > 4. \end{cases}$$

Отримана математична модель ніяких спеціальних методів для реалізації не вимагає.

3. Розробка алгоритму розв'язку задачі.

Оскільки для одержанні кінцевого результату, передбачається вибір одного із трьох можливих напрямків обчислень, використовуватимо алгоритм, що розгалужується. Залежно від результату перевірки значення змінної *X* змінній *Y* присвоюватиметься відповідне значення.

Реалізація алгоритму, що розгалужується, розв'язок даної задачі можливий у вигляді декількох варіантів блок-схем, один з яких наведений нижче.



C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double X,Y;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть значення X ";
  cin>>X;
  if(X<1) Y=2;
  else if(X<=4) Y=-X+3;
  else Y=-1;
          cout<<"Y = "<<Y<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim x, y As Single
x = CSng(Text1)
If x < 1 Then ' якщо x<1, тоді змінній y привласнюється значення 2
y = 2
ElseIf x <= 4 Then ' інакше, якщо x <= 4, y привласнюється значення: -x + 3
y = -x + 3
Else: y = -1 ' інакше y привласнюється значення:-1
End If
Text2 = y ' виведення на форму значення y
End Sub
```

TP

```
Program p_2_2_4;
  var x,y:real;
begin
writeln('Введіть x');
  readln(x); {введення значення x}
  if x<1 then y:=2 else {якщо x<1, тоді змінній y привласнюється значення 2, інакше, }
  if x<=4 then y:=-x+3 else y:=-1; {якщо x<=4, змінній y привласнюється значення -x+3 в протележному випадку: -1}
  writeln (y:4:3); { виведення значення y}
End.
```

Приклад 2.2.5 Визначити чи можна в прямокутний отвір з відомими розмірами сторін вставити трубу із заданим зовнішнім діаметром.

Розв'язок

1. Постановка задачі.

Результатом розв'язку є текстове повідомлення «можна» або «не можна» вставити трубу в прямокутний отвір. Для цього повинні бути задані розміри сторін прямокутного отвору й зовнішній діаметр труби. Інших початкових даних для розв'язку не потрібно.

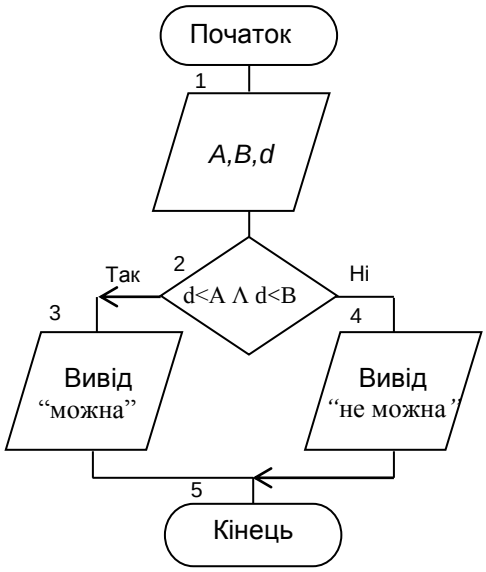
2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Довжина прямокутного отвору	Дійсний	*A*	Початкове дане
Ширина прямокутного отвору	Дійсний	*B*	Початкове дане
Зовнішній діаметр труби	Дійсний	*d*	Початкове дане

Трубу можна вставити в прямокутний отвір, якщо її зовнішній діаметр менше, ніж кожна сторона прямокутного отвору.

3. Розробка алгоритму розв'язку задачі.



C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
  double A,B,d;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть довжину та ширину прямокутного отвору ";
  cin>>A>>B;
  cout<<"Введіть зовнішній діаметр труби ";
  cin>>d;
 if(d<A&&d<B) cout<<"Можна"<<endl;/*якщо d менше A (логічне І) d менше B, то "можна"
*/
 else cout<<"Не можна"<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim a, b, d As Single
' введення значень a,b,d
a = CSng(Text1)
b = CSng(Text2)
d = CSng(Text3)
If (d < a) And (d < b) Then Text4 = "Можна" Else Text4 = "Не можна" ); ' якщо d<a та d<b,
```

```
на форму виводиться «Можна», інакше - «Не можна»
End Sub
```

TP

```
Program p_2_2_5;
var a,b,d:real;
begin
writeln('Введіть значення трьох сторін прямокутного отвору');
readln(a,b,d); {введення значень a,b,d}
if (d<a)and(d<b) then writeln('Можна') else writeln ('Не можна'); {якщо d<a та d<b, виводиться «Можна», інакше - «Не можна»}
End.
```

Незважаючи на важливість алгоритмів, що розгалужуються, які дозволяють знаходити найбільше (найменше) значення серед декількох заданих, визначати попадання в зазначені діапазони даних, стан прапорців (ключів, перемикачів), найбільша ефективність застосування комп'ютера для реалізації складних математичних моделей, безсумнівно, досягається при багаторазових циклічних обчислювальних процесах.

2.3 Графічне представлення циклічних обчислювальних процесів

У програмах, пов'язаних з обробкою даних або іншими складними обчисленнями, часто виконуються циклічно повторювані дії.

Наприклад, при необхідності присвоїти значення кільком сотням змінних важко і безглуздо «вручну» записувати в тексті програми сотні операторів введення або присвоювання. Цикли дозволяють записати такі дії в компактній формі. Тому вони є однією з найважливіших алгоритмічних структур.

Цикл - це послідовність кроків, яка виконується *неодноразово*. Процес називається ***циклічним***, якщо в ньому має місце багаторазове повторення тих самих дій або обчислення за тими самими формулами, причому кожного разу обчислення проводиться після підстановки в ці формули нових вихідних значень.

Змінна, яка керує рухом циклічного обчислювального процесу, називається ***змінною (параметром) циклу***. Цей параметр приймає нові значення при кожному повторенні циклу.

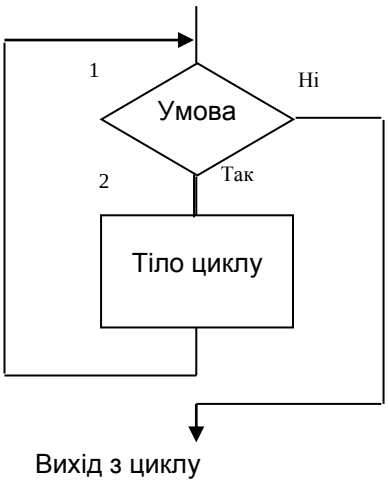
Параметрами циклу також називаються конкретні значення змінної циклу, які визначають характеристики циклічного процесу. До параметрів циклу відносять початкове і кінцеве значення змінної циклу і крок її зміни в процесі виконання алгоритму.

Цикл називається ***детермінованим***, якщо кількість повторень тіла циклу заздалегідь відомо або може бути визначено на підставі вихідних даних.

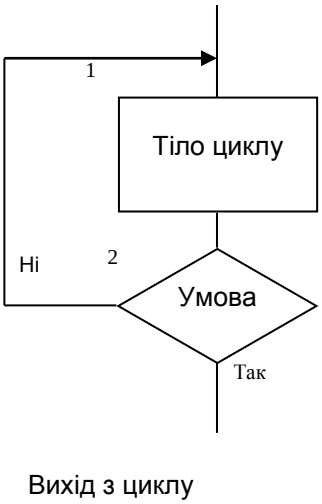
В організації циклу можна виокремити наступні етапи:

- підготовка (ініціалізація) циклу;
- виконання обчислень циклу (тіло циклу);
- модифікація параметрів;
- перевірка умови закінчення циклу.

Порядок виконання етапів організації циклу може змінюватися. Залежно від розташування перевірки умови закінчення циклу розрізняють цикли з нижнім і верхнім закінченням. Для циклу з нижнім закінченням (так званий *цикл з післяумовою*) тіло циклу виконується як мінімум один раз, оскільки спочатку виконуються обчислення, а потім перевіряється умова виходу з циклу. У випадку циклу з верхнім закінченням (так званий цикл *з передумовою*) тіло циклу може не виконатися жодного разу у випадку, якщо умова виходу одразу ж виконується .



Цикл з передумовою



Цикл з післяумовою

Цикл з параметром

Розглянемо докладніше складання схеми циклічного алгоритму на прикладі.

Приклад 2.3.1 Обчислити значення функції $y = ax^2$ для ряду значень аргументу *x*, що змінюється від початкового значення *xn* до кінцевого значення *xk* з кроком *dx*.

Рішення

1. Постановка задачі.

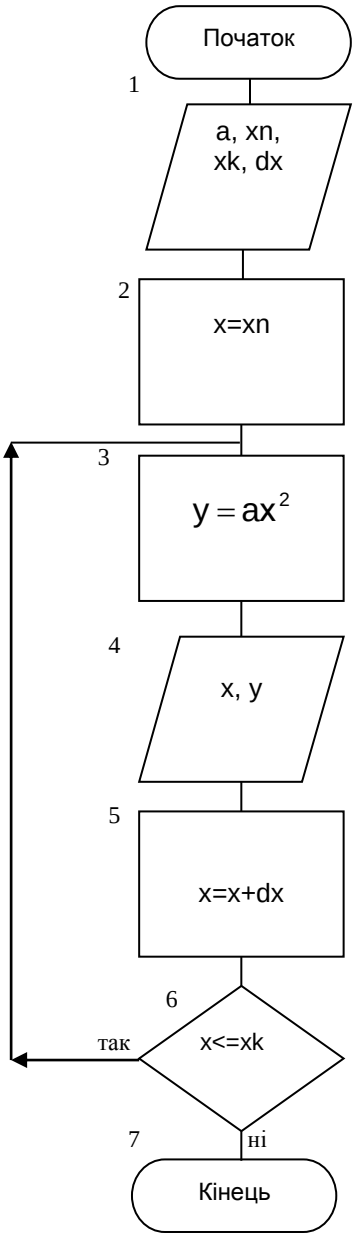
Результатом розв'язку задачі є таблиця значень аргументу *x* і функції *y*. У ході виконання завдання аргумент обчислюваної функції має змінюватися від початкового значення *xn* на величину кроку *dx* при кожному повторенні циклу. Розв'язок задачі має завершитися при досягненні аргументом *x* його кінцевого значення *xk*. Отже, для вирішення завдання мають бути задані початкове, кінцеве значення аргументу і крок. Також необхідним вихідним даним є значення коефіцієнта *a*.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім’я	Призначення
Початкове значення аргументу	Дійсний	*xn*	Вихідне дане
Кінцеве значення аргументу	Дійсний	*xk*	Вихідне дане
Крок зміни аргументу	Дійсний	*dx*	Вихідне дане
Коефіцієнт	Дійсний	*a*	Вихідне дане
Поточне значення аргументу	Дійсний	*x*	Результат
Значення функції	Дійсний	*y*	Результат

Перше значення аргументу функції *x=xn*. Кожне нове значення аргументу будемо обчислювати за формулою *x=x+dx*. Значення функції будемо обчислювати за заданою формулою $y = ax^2$.



3. Розробка алгоритму розв'язання задачі.

Змінною циклу в цій реалізації рішення задачі є значення аргументу функції *x*.

Для реалізації алгоритму будемо використовувати варіант циклу з нижнім закінченням, тобто цикл з післяумовою.

У блоці 1 через кому записуємо імена змінних, зазначених у таблиці імен змінних як вихідні дані.

Блок 2 називається початковою частиною циклічного алгоритму. У цьому блоці задається початкове значення змінної циклу *x* (виконується ініціалізація).

Блоки 3-4 - це тіло циклу, вони виконують обчислення значення заданої функції і виводять отриманий результат.

У блоці 5 відбувається модифікація змінної циклу *x*.

Блок 6 - це логічна частина циклу. Якщо змінна циклу *x* не перевищила своє кінцеве значення *xk*, то виконується чергове проходження тіла циклу, інакше цей блок забезпечує вихід з циклу.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *a* = 2; *xn* = 1; *xk* = 5; *dx* = 1. Виконання дій за блок-схемою оформимо у вигляді таблиці.

Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5
	Початок				
1	Введення: 2; 1; 5; 1				
2	x=1				
3	y=2	y=8	y=18	y=32	y=50
4	Виведення: 1 2	Виведення: 2 8	Виведення: 3 18	Виведення: 4 32	Виведення: 5 50
5	x=2	x=3	x=4	x=5	x=6
6	2<=5 так (перехід на блок 3)	3<=5 так (перехід на блок 3)	4<=5 так (перехід на блок 3)	5<=5 так (перехід на блок 3)	6<=5 ні (перехід на блок 7)
7					Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream> //бібліотека для роботи з функціями вводу/виведення
#include<clocale>//бібліотека для роботи з локалями
using namespace std; /* підключення простору імен std, в якому знаходяться ф-ції
вводу/виведення */
int main() //опис функції
{
  double xn,xk,dx,a,x,y; //оголошення змінних дійсного типу
  setlocale(LC_ALL, "UKR"); /* підключення локалей для відображення української мови */
  cout<<"Введіть значення коефіцієнта "; // виведення повідомлення на екран
  cin>>a; // запис значення в змінну a
  cout<<"Введіть початкове значення аргументу ";
  cin>>xn;
  cout<<"Введіть кінцеве значення аргументу ";
  cin>>xk;
  cout<<"Крок зміни аргументу ";
  cin>>dx;
x=xn;
do //робити
{
        y=a*x*x; //розрахунок функції
        cout<<" x = "<<x<<"  y = "<<y<<endl;/* виведення x="аргумент" y="значення
функції" */
        x+=dx; //збільшення аргументу на заданий крок dx
} while (x<=xk);//поки виконується умова x<=xk


system("pause"); //системна затримка для перегляду результату
return 0; //те, що повертає функція
}
```

VB

```
Private Sub Command1_Click()
Dim a, xn, nk, x, dx, y As Single
' введення з форми вихідних даних та перетворення їх у дійсний тип
a = CSng(Text1)
```

```
xn = CSng(Text2)
xk = CSng(Text3)
dx = CSng(Text4)
x = xn ' аргумент приймає своє початкове значення
Do ' виконується тіло циклу
    y = a * x ^ 2 ' обчислення функції
    Print x, y ' виведення на форму значень аргументу та функції від цього аргументу
    x = x + dx ' наступне значення аргументу обчислюється як попереднє, збільшене на величину кроку dx
Loop While x <= xk ' доки логічна умова x<=xk істинна, повторюється виконання тіла циклу, виконання програми повертається до оператора Do
End Sub
```

TP

```
program p_2_3_1;
uses CRT; {підключення модуля }
var x, a, xn, xk, dx, y:real;
begin
    clrscr; {очищення екрану}
    writeln('Ввести значення параметрів xn, xk, dx, a ');
    readln(xn, xk, dx, a);
    x:=xn;
    repeat {оператор циклу, що примушує тіло циклу повторюватись… }
        y:=a*sqr(x); { тіло циклу }
        writeln('x=',x:4:2,'y=':5,y:4:2);
        x:=x+dx;
    until x>xk; { доки логічна умова не прийме істинне значення }
    readln { очікування натискання клавіши}
end.
```

Залежно від способу модифікації змінної циклу, від способу контролю закінчення циклу, від способу отримання остаточного результату розрізняють такі основні типи циклічних алгоритмів:

1. Простий цикл:
   а) простий цикл;
   б) простий цикл з лічильником.
2. Цикл с переадресацією.
3. Ітераційний цикл.
4. Комбіновані цикли:
   а) цикли з накопиченням;
   б) вкладені цикли;
   в) комбіновані складні цикли.

Розв'язок задачі, розглянутий у прикладі 2.3.1, відноситься до простих циклів.

Простим циклом називається такий вид циклічних алгоритмів, коли відомий закон модифікації змінної циклу, причому змінна циклу збігається з аргументом обчислюваних математичних залежностей, а число повторень циклу заздалегідь відомо.

Ознаки простого циклу.

1. Спосіб модифікації змінної циклу полягає в тому, що кожне її наступне значення обчислюється на підставі попереднього за деяким законом, наприклад $x=x+dx$. Цей закон називається рекурентною (повторюваною) формулою. Якщо закон модифікації змінної циклу буде складнішим, виду $x=\varphi(x)$, то структура циклу не зміниться.
2. Результати обчислень отримуємо і виводимо при кожному проходженні циклу.
3. Контроль закінчення циклу здійснюється порівнянням поточного значення змінної циклу з її кінцевим значенням.

При заданих значеннях параметрів змінної циклу *xn*, *xk*, *dx* кількість *ітерацій* (повторень тіла циклу) визначається за формулою:

$$N = \left[\frac{xk - xn}{dx} \right]_{\text{ціл}} + 1$$

Оскільки кількість повторень циклу в простому циклі може бути обчислена заздалегідь, то контроль закінчення циклу можна організувати шляхом підрахунку вже виконаних циклів і порівнянням цієї кількості із заздалегідь підрахованим або заданим значенням.

Простий цикл з лічильником - це такий вид циклу, коли відома загальна кількість циклів і вводиться додаткова змінна-лічильник, яка враховує число здійснених проходів тілом циклу, а вихід з даної структури алгоритму здійснюється при досягненні значенням лічильника встановленого значення.

Тобто при реалізації простого циклу з лічильником відбувається розділення змінної циклу та математичного аргументу. Всі інші ознаки простого циклу зберігаються, а закон модифікації змінної циклу змінюється. Рекурентна формула в цьому випадку приймає вигляд $i=i+1$.

Вирішимо задачу з **прикладу 2.3.1** ще раз з використанням схеми простого циклу з лічильником.

Рішення

1. Постановка задачі не змінюється.
2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім’я	Призначення
Початкове значення аргументу	Дійсний	*xn*	Вихідне дане
Кінцеве значення аргументу	Дійсний	*xk*	Вихідне дане
Крок зміни аргументу	Дійсний	*dx*	Вихідне дане
Коефіцієнт	Дійсний	*a*	Вихідне дане
Кількість повторень циклу	Цілий	*N*	Допоміжна змінна
Лічильник	Цілий	*i*	Допоміжна змінна
Поточне значення аргументу	Дійсний	*x*	Результат
Значення функції	Дійсний	*y*	Результат

На додачу до формул математичної моделі з попереднього методу рішення необхідні такі записи. Початкове значення змінної циклу-лічильника *i*=1. Кожне наступне значення лічильника обчислюємо за формулою *i=i+1*. Кількість повторень циклу обчислимо за наведеною вище формулою.

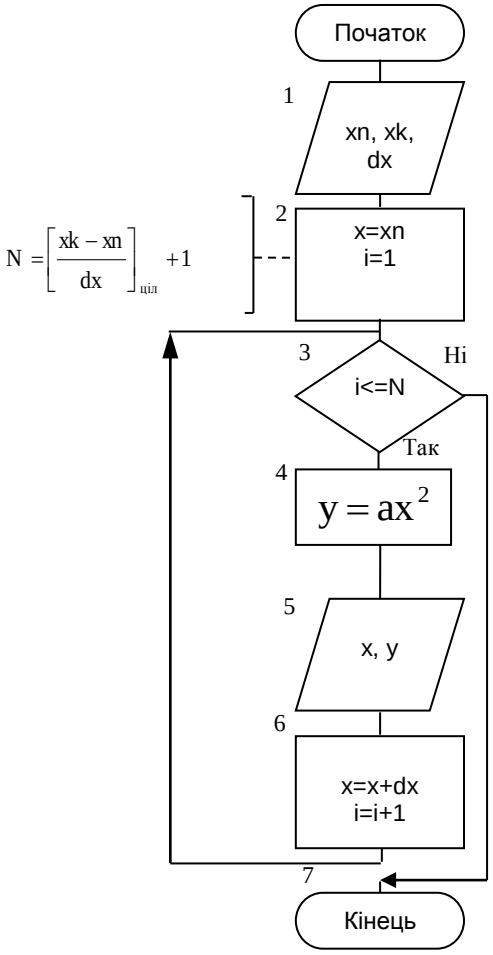
3. Розробка алгоритму розв'язання задачі.

Для реалізації алгоритму будемо використовувати варіант циклу з верхнім закінченням, тобто цикл з передумовою.

У блоці 2 обчислимо кількість повторень циклу *N*, призначимо змінній-лічильнику початкове значення *i*=1 і задамо початкове значення математичного аргументу *x=xn*.

Блок 3 - це логічна частина циклу. Якщо змінна-лічильник не перевищила своє кінцеве значення *N*, то виконується чергове проходження тіла циклу, інакше цей блок забезпечує вихід із циклу.

Блоки 4-5 - це тіло циклу, вони виконують обчислення значення заданої функції і виводять отриманий результат. У блоці 6 відбувається модифікація змінної циклу *i* та збільшення аргументу функції *x* на величину кроку, після чого здійснюється повернення на блок 3.



Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *a* = 2; *xn* = 1; *xk* = 5; *dx* = 1. Виконання дій за блок-схемою оформимо у вигляді таблиці.

Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6
	Початок					
1	Введення: 2; 1;5;1					
2	x=1; i=1; N=(5-1)/1+1=5					
3	1<=5 так (перехід на блок 4)	2<=5 так (перехід на блок 4)	3<=5 так (перехід на блок 4)	4<=5 так (перехід на блок 4)	5<=5 так (перехід на блок 4)	5<=6 ні (перехід на блок 7)
4	y=2	y=8	y=18	y=32	y=50	
5	Виведення: 1 2	Виведення: 2 8	Виведення: 3 18	Виведення: 4 32	Виведення: 5 50	
6	x=2; i=2 (повернення на блок 3)	x=3; i=3 (повернення на блок 3)	x=4; i=4 (повернення на блок 3)	x=5; i=5 (повернення на блок 3)	x=6; i=6 (повернення на блок 3)	
7						Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream> //бібліотека для роботи з функціями вводу/виведення
#include<clocale>//бібліотека для роботи з локалями
using namespace std; /* підключення простору імен std, в якому знаходяться ф-ції
вводу/виведення */
int main() //опис функції
{
  double xn,xk,dx,a,x,y; //оголошення змінних дійсного типу
  int N,i;
setlocale(LC_ALL, "UKR"); /* підключення локалей для відображення української мови */
  cout<<"Введіть значення коефіцієнта "; // виведення повідомлення на екран
  cin>>a; // запис значення в змінну a
  cout<<"Введіть початкове значення аргументу ";
  cin>>xn;
  cout<<"Введіть кінцеве значення аргументу ";
  cin>>xk;
  cout<<"Крок зміни аргументу ";
  cin>>dx;
x=xn;
N=(xk-xn)\dx+1; //розрахунок кількості повторів розрахунку функції
i=1; //початкове значення лічильника
do //робити
{
        y=a*x*x; //розрахунок функції
        cout<<" x = "<<x<<"  y = "<<y<<endl;/* виведення x="аргумент" y="значення
функції" */
        x+=dx; //збільшення аргументу на заданий крок dx
       i+=1; //збільшення лічильника одиницю
```

```
} while (i<=N);//поки виконується умова i<=N


system("pause"); //системна затримка для перегляду результату
return 0; //те, що повертає функція
}
```

VB

```
Private Sub Command1_Click()
Dim a, xn, nk, x, dx, y As Single, i, n As Integer
a = CSng(Text1)
xn = CSng(Text2)
xk = CSng(Text3)
dx = CSng(Text4)
n = (xk - xn) \ dx + 1 ' знаходження цілої частини частки від ділення різниці кінцевого та початкового значень аргументу на його прирощення
x = xn
i = 1
Do While i <= n ' доки логічна умова істинна, виконується тіло циклу
    y = a * x ^ 2 обчислення y
    Print x, y ' виведення x та y
    x = x + dx ' збільшення x на величину dx
    i = i + 1 ' збільшення змінної циклу на оиницю
Loop ' повернення до перевірки логічної умови
End Sub
```

TP

```
program p_2_3_1_a;
uses CRT;
var x, a, xn, xk, dx, y:real; i, n:integer;
begin
    clrscr;
    writeln(' Введіть змінні xn, xk, dx, a ');
    readln(xn,xk,dx,a); {введення змінних xn,xk,dx,a }
    x:=xn; i:=1;
    n:=trunc((xk-xn)/dx)+1; {знаходження цілої частини частки від ділення різниці кінцевого та  початкового значень аргументу на його прирощення }
    While i<=n do {доки логічна умова i<=N виконується, тіло циклу повторюється}
    Begin {початок тіла циклу }
        y:=a*sqr(x); {обчислення y }
        writeln('x=',x:4:2,'y=':5,y:4:2); {виведення x та y }
        x:=x+dx; i:=i+1; {збільшення x на величину dx та i на одиницю }
    end; { кінець тіла циклу }
    readln
end.
```

Цикл з переадресацією застосовується при розв'язанні задач, де в якості вихідних даних використовуються переважно масиви даних.

Знову розглянемо рішення задачі обчислення таблиці значень функції, дещо змінивши умову. Нехай потрібно розв'язати задачу обчислення значень функції $y = ax^2$ для невпорядкованої послідовності аргументу *x*, що налічує сто елементів $x_1, x_2, \ldots, x_{100}$. Всі отримані значення необхідно зберігати до кінця роботи програми (**Приклад 2.3.2**).

Оскільки за відсутності закономірності зміни аргументу *x* рекурентну формулу скласти неможливо, то в такому випадку алгоритм простого циклу непридатний до використання. Для вирішення задачі необхідно запровадити деяку закономірність штучно.

Всю задану послідовність значень аргументу організовують у структуру, яка називається ***масивом***, і позначають загальним ім'ям, а кожне окреме значення послідовності позначається тим же ім'ям з уточненням порядкового номера. Тоді в загальному вигляді окреме значення аргументу з номером *i* називається елементом масиву і позначається, наприклад, x_i.

Масивом називається сукупність даних одного типу, об'єднаних загальним ім'ям і структурованих системою індексів.

Характеристики масива:

- тип - спільний тип всіх елементів масиву;
- розмірність (ранг) - кількість індексів масиву;
- діапазон зміни індексу (індексів) - визначає кількість елементів у масиві.

Одновимірний масив - це масив, в якому елементи нумеруються одним індексом.

Якщо в масиві зберігається таблиця значень (матриця), то такий масив називається ***двовимірним***, а його елементи нумеруються двома індексами - номером рядка і стовпця відповідно, наприклад, x_{ij}.

Звернутися до кожного елементу масиву можна за номером (індексом або кількома індексами). В якості індексу елемента масиву використовується ціла константа або змінна цілочисельного типу.

Обробка масивів виконується при зміні індексів елементів.

Оскільки елементи масиву в області пам'яті записані послідовно, то індекс елемента пов'язаний з адресою комірки пам'яті, де він розташований. Зміна індексу елемента викликає зміну адреси комірки, тобто ***переадресацію***.

Цикли, в яких закономірність модифікації керуючої змінної полягає в переадресації, називаються ***циклами з переадресацією***.

Ознаки цикла з переадресацією:

1. Кількість виконань циклів відомо заздалегідь.

2. Математична закономірність зміни аргументу відсутня.
3. Обробці підлягає сукупність значень аргументу, зведена в упорядкований масив.
4. В якості змінної циклу використовуються індекси елементів масиву.
5. Алгоритм має риси простого циклу з лічильником.

Складемо схему алгоритму прикладу 2.3.2.

Рішення

1. Постановка задачі.

Результатом розв'язку задачі є таблиця значень аргументу *x* і функції *y*, збережена в пам’яті комп'ютера. У ході виконання задачі аргумент і обчислені значення функції мають бути представлені у вигляді одновимірних масивів з *N* елементів. Змінюючи номер елемента масиву від початкового значення 1 до кінцевого значення *N* з кроком 1, при кожному повторенні циклу матимемо доступ до чергового значення аргументу, обчислювати значення функції і зберігати його в масиві *y*. Рішення задачі має завершитися при досягненні змінною циклу *i* її кінцевого значення *N*. Отже, для вирішення задачі мають бути задані кількість елементів масиву *N* і масив значень аргументу *x*. Також необхідним вихідним даним є значення коефіцієнта *a*.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім’я	Призначення
Масив аргументів	Дійсний	*x*	Вихідне дане
Кількість елементів масива	Цілий	*N*	Вихідне дане
Коефіцієнт	Дійсний	*a*	Вихідне дане
Лічильник	Цілий	*i*	Допоміжна змінна
Масив значень функції	Дійсний	*y*	Результат

Значення функції будемо обчислювати за заданою формулою $y_i = ax_i^2$.

3. Розробка алгоритму розв’язання задачі.

Для реалізації алгоритму будемо використовувати варіант циклу з параметром. У цьому різновиді циклічних алгоритмів застосовується блок Переадрасація (блок 2), який поєднує в собі три алгоритмічні дії:

1) надає змінній циклу *i* початкове значення, в нашій задачі 1;

2) перевіряє умову, чи не досягла змінна циклу свого кінцевого значення *N*; Якщо результат перевірки умови «істина», то виконується тіло циклу, інакше здійснюється вихід з циклу.

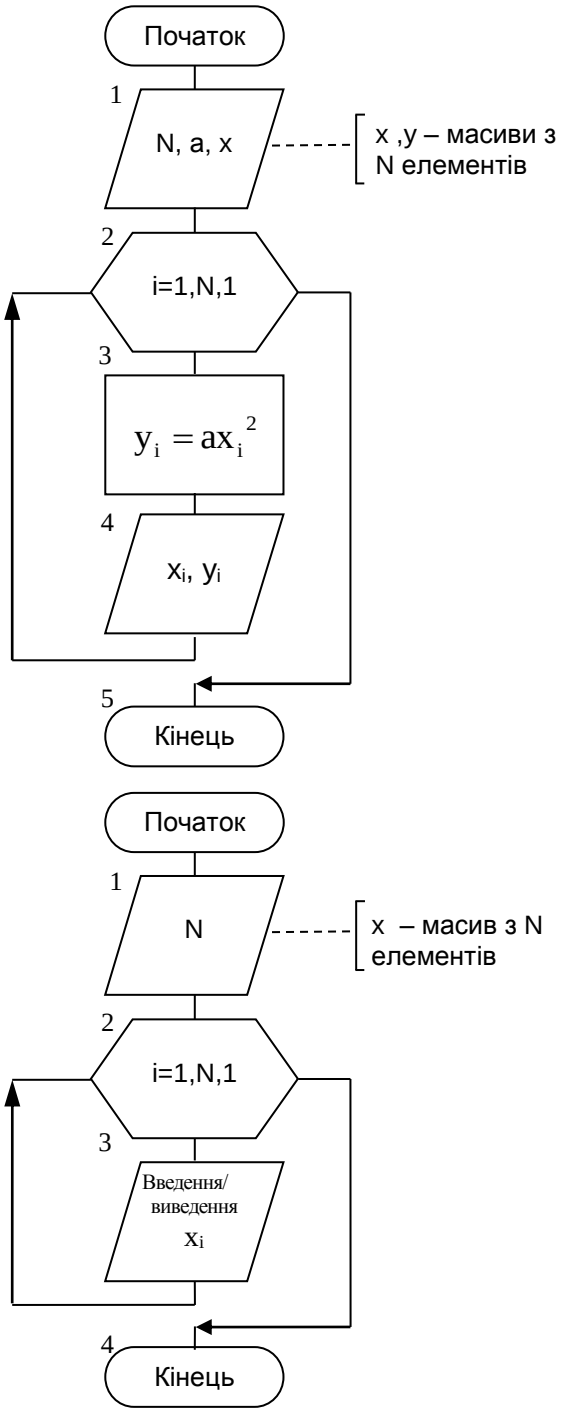
3) після виконання тіла циклу змінює змінну циклу на зазначену величину кроку, в нашій задачі 1.

Можна сказати, що цикл з параметром працює аналогічно циклу з передумовою.

Тіло циклу утворено блоками 3 і 4, які забезпечують обчислення значення функції і виведення отриманих результатів.

У блоці 1 зазначений список введення, що містить кількість елементів масиву *N*, коефіцієнт *a* й сам масив *x*. Слід розуміти, що таке позначення введення масиву є умовним. Насправді, введення і виведення елементів масиву - це не одноразова дія, а циклічний процес, який зображений у наведеній блок-схемі.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *a*=2; *N*=5; *x*=(2;4;6;8;10). Виконання дій за блок-схемою оформимо у вигляді таблиці.



Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6
	Початок					
1	Введення: 5;2;4;6;8;10					
2	i=1; 1<=5 так (перехід на блок 3)	i=2; 2<=5 так (перехід на блок 3)	i=3; 3<=5 так (перехід на блок 3)	i=4; 4<=5 так (перехід на блок 3)	i=5; 5<=5 так (перехід на блок 3)	i=6; 6<=5 ні (перехід на блок 5)
3	y=8	y=32	y=72	y=128	y=200	
4	Виведення: 2 8 (повернення на блок 2)	Виведення: 4 32 (повернення на блок 2)	Виведення: 6 72 (повернення на блок 2)	Виведення: 8 128 (повернення на блок 2)	Виведення: 10 200 (повернення на блок 2)	
5						Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{

  double a;
  int N,i;
  setlocale(LC_ALL, "UKR");
  cout<<"Введіть кількість елементів масиву ";
  cin>>N;
  double *x = new double [N];//оголошення динамічного масиву x з N елементів
  double *y = new double [N];//оголошення динамічного масиву y з N елементів
  cout<<"Введіть коефіцієнт ";
  cin>>a;

  for(i=1;i<=N;i++) //введення масиву x
  {
            cout<<"x["<<i<<"]=";
            cin>>x[i];
  }

  for(i=1;i<=N;i++)
  {
            y[i]=a*x[i]*x[i]; //разрахунок формули
            cout<<"x["<<i<<"] = "<<x[i]<<"  y["<<i<<"] = "<<y[i]<<endl;
          //виведення результату
  }

delete []x;//очищення динамічної пам'яті
delete []y;

system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim x(), y(), a As Single  ' об'явлення динамічних масивів x та y
Dim n, i As Integer
n = CInt(Text2) ' введення з форми кількості елементів в масивах x та y
ReDim x(n) ' визначення розмірності масиву x
ReDim y(n) ' визначення розмірності масиву y
a = CSng(Text1)
' введення елементів масиву
For i = 1 To n Step 1 'змінна циклу i приймає значення від одиниці до n з кроком, що дорівнює
```

```
одиниці
    x(i) = CSng(InputBox("Введіть " & i & "-ий елемент масиву аргументів"))
Next i ' змінна циклу збільшується на величину кроку та управління передається структурі For
For i = 1 To n Step 1 'змінна циклу і приймає значення від одиниці до n з кроком, що дорівнює одиниці
' тіло циклу
    y(i) = a * x(i) ^ 2
    Print x(i), y(i)
Next i ' змінна циклу збільшується на величину кроку та управління передається структурі For
End Sub
```

TP

```
program p_2_3_2;
uses CRT;
var x, y:array [1..100]of real; i, n, a:integer;
begin
    clrscr;
    writeln("Вводимо змінні a і N");
    readln (a,n); {введення змінних a та n }
    for i:=1 to n do {змінна циклу і змінюється від 1 до n з кроком 1 }
        begin {початок тіла циклу }
        writeln(' Введіть значення аргументів в масив x');
        readln(x[i]); {введення значень до масиву аргументів x(i) }
        y[i]:=a*sqr(x[i]); {обчислення y(i) }
        writeln('x=',x[i]:4:2,'y=':5,y[i]:4:2); {виведення x(i) та y(i) }
    end;
    readln
end.
```

Цикли з накопиченням.

У практиці розв’язання задач дуже поширені обчислення сум з великою кількістю доданків і добутків з великою кількістю множників. Ці обчислення виконуються за циклічними алгоритмами, в яких у кожному циклі обчислюється один доданок (або множник), а результат накопичується від циклу до циклу шляхом додавання чергового доданка до вже накопиченої часткової суми (або шляхом множення чергового множника на вже накопичений добуток). Такі алгоритми називають ***циклами з накопиченням***.

Для організації циклу з накопиченням необхідно виділити спеціальну ***змінну-накопичувач*** і в початковій частині алгоритму присвоїти їй початкове значення. У разі накопичення суми її початковим значенням має бути 0 (***нуль***). При накопиченні добутку початковим значенням

накопичувача повинна бути 1 (***одиниця***).

У тілі циклу необхідно використовувати блок накопичення, що виконує операцію:

$S=S+X$ або $S=S*Y$,

де S – змінна-накопичувач;

X та Y – доданок та множник, відповідно.

Остаточний результат накопичення суми або добутку буде готовий тільки після закінчення виконання циклу.

Приклад 2.3.3 В одновимірному масиві, що складається з 7 елементів, знайти суму додатних елементів.

Рішення

1. Постановка задачі.

Результатом рішення задачі є сума додатних елементів одновимірного масиву. Для розрахунку і зберігання результату буде використовуватися змінна-накопичувач S. Змінюючи індекс i елемента масиву x від початкового значення 1 до кінцевого значення 7 з кроком 1, при кожному повторенні циклу будемо перевіряти, чи є значення елементу масиву додатним. Якщо «так», то змінну S будемо сумувати зі значенням елемента масиву. Якщо «ні», то такий елемент масиву будемо пропускати. Отже, для рішення задачі має бути заданий масив x з 7 чисел.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім’я	Призначення
Масив з 7 чисел	Дійсний	x	Вихідне значення
Змінна циклу	Цілий	i	Допоміжна змінна
Сума додатних елементів	Дійсний	S	Результат

Початкове значення суми додатних елементів дорівнює 0. Для накопичення суми будемо використовувати формулу $S=S+x_i$.

3. Розробка алгоритму розв’язання задачі.

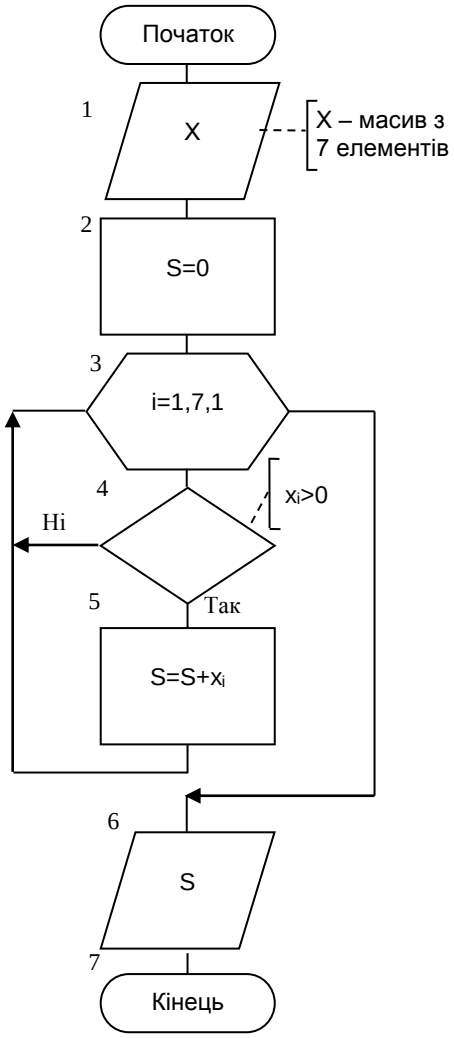
Для реалізації алгоритму будемо використовувати варіант циклу з параметром.

У блоці 1 здійснюється введення елементів одновимірного масиву. У блоці 2 змінній S присвоюється початкове значення.

У блоці 3 задаємо параметри роботи циклу з переадресацією. Блоки 4-5 є тілом циклу. Блок 4 Рішення перевіряє, чи є поточний елемент масиву x додатним. Якщо відповідь «так», то в блоці 5 відбувається накопичення суми додатних елементів *S*.

Блок 6 служить для виведення результатів рішення задачі.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *x*=(-1;3;-5;7;8;9;-10). Виконання дій за блок-схемою оформимо у вигляді таблиці.



Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6	Цикл 7	Цикл 8
	Початок							
1	Введення: (-1;3; -5;7;8; 9;-10)							
2	S=0							
3	i=1 1<=7 так (перехід на блок 4)	i=2 2<=7 так (перехід на блок 4)	i=3 3<=7 так (перехід на блок 4)	i=4 4<=7 так (перехід на блок 4)	i=5 5<=7 так (перехід на блок 4)	i=6 6<=7 так (перехід на блок 4)	i=7 7<=7 так (перехід на блок 4)	i=8 8<=7 ні (перехід на блок 6)
4	-1>0 ні (перехід на блок 3)	3>0 так (перехід на блок 5)	-5>0 ні (перехід на блок 3)	7>0 так (перехід на блок 5)	8>0 так (перехід на блок 5)	9>0 так (перехід на блок 5)	-10>0 ні (перехід на блок 3)	
5		S=0+3=3 (повернення на блок 3)		S=3+7=10 (повернення на блок 3)	S=10+8=18 (повернення на блок 3)	S=18+9=27 (повернення на блок 3)		
6								Виведення 27
7								Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
```

```
using namespace std;
int main()
{

  double S;
  double x[8];
  int i;
  setlocale(LC_ALL, "UKR");

  cout<<"Введіть елементи масиву x ";

  for(i=1;i<=7;i++) //введення масиву x
  {
          cout<<"x["<<i<<"]=";
          cin>>x[i];
  }

  S=0;

  for(i=1;i<=7;i++)//з першого по 7-ий елемент масиву x з кроком 1 робити
          if (x[i]>0) S+=x[i];//якщо x>0, то накопичувати суму

  cout<<"S="<<S<<endl;//виведення результату

system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim x(7), s As Single ' об'явлення одновимірного масиву x, що складається з семи елементів
Dim i As Integer
' введення елементів масиву x
For i = 1 To 7 Step 1
    x(i) = CSng(InputBox("Введіть " & i & "-ий елемент масиву"))
Next i
For i = 1 To 7 ' змінна циклу і приймає значення від одиниці до 7 з кроком, що дорівнює одиниці. Якщо крок дорівнює одиниці, ключове слово Step можна не писати.
    If x(i) > 0 Then s = s + x(i)
' якщо тіло циклу складається з однієї лексеми або одного оператора, ключове слово Next можна не писати, змінна циклу автоматично збільшується на величину кроку
Print s
End Sub
```

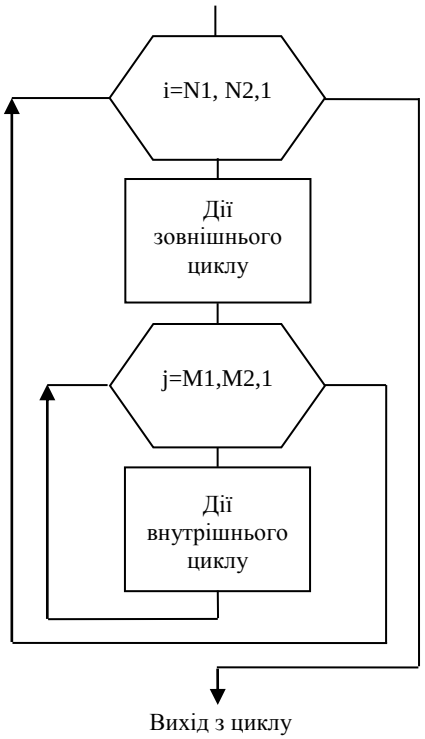
TP

```
program p_2_3_3;
uses CRT;
var x:array [1..7] of real; {об'явлення одновимірного масиву x, що складається з семи дійсних елементів }
```

```
    i:integer; s:real;
begin
  clrscr;
  for i:=1 to 7 do begin {починаючи з першого по сьомий, вводяться елементи масиву x }
    writeln('Введіть елемент масиву x');
    readln(x[i]);
  end;
  s:=0; { початкове значення суми дорівнює нулю }
  for i:=1 to 7 do begin { починаючи з першого по сьомий, додатні елементи масиву сумуються }
    if x[i]>0 then s:=s+x[i];
  end;
  writeln(s); {виведення значення суми }
  readln
end.
```

Вкладені цикли

У тілі будь-якого циклічного алгоритму можуть знаходитися цикли. При цьому цикл, що містить в собі інший, називають зовнішнім, а цикл, що знаходиться у тілі першого, - внутрішнім (або вкладеним). Правила організації зовнішнього і внутрішнього циклу такі ж, як і для простого циклу.



Зверніть увагу, при складанні алгоритмів вкладених циклів необхідно дотримувати наступну додаткову умову: *всі дії внутрішнього циклу повинні повністю розташовуватися у тілі зовнішнього циклу*.

При вкладенні циклів внутрішній цикл виконується ***повністю*** від початкового до кінцевого значення параметра при ***незмінному*** значенні параметра ***зовнішнього*** циклу.

Потім значення параметра ***зовнішнього*** циклу змінюється на ***величину кроку***, і знову від початку і до кінця виконується ***вкладений*** цикл. І так до тих пір, поки значення параметра зовнішнього циклу не стане більше кінцевого значення, визначеного в зовнішньому циклі.

Обробка матриць

Найчастіше вкладені цикли застосовуються в алгоритмах розв’язання задач з використанням багатовимірних масивів даних. Розглянемо цю тему докладніше на прикладі двовимірних масивів, які часто називають матрицями.

При обробці матриць є такі типові варіанти задач:

- знаходження будь-якої величини для всього масиву;
- знаходження будь-якої величини для кожного рядка матриці;

- знаходження будь-якої величини для кожного стовпця матриці.

Правила складання алгоритмів при розв'язанні задач з двовимірними масивами:

1. Для обробки *n*-мірних масивів використовуються вкладені цикли з глибиною вкладення *n*, причому логічний блок перевірки заданої умови і обчислювальний блок розміщуються у тілі внутрішнього циклу.

2. При знаходженні шуканої величини для всієї матриці початкове присвоєння здійснюється один раз до входу в зовнішній цикл, а результат виводиться після виходу із зовнішнього циклу.

3. Якщо шукана величина визначається для рядків або стовпців, то початкове присвоєння і виведення результату здійснюється число разів, рівне числу зовнішніх циклів. Блок початкового присвоєння розміщується на вході у внутрішній цикл, а блок виведення результатів - на виході з внутрішнього циклу.

4. Якщо шукана величина визначається для рядків, то зовнішній цикл організовується за рядками, а внутрішній - за стовпцями. Якщо шукана величина визначається для стовпців, то зовнішній цикл організовується за стовпцями, а внутрішній - за рядками.

При обробці матриць часто зустрічаються задачі, дії в яких здійснюються з елементами, розташованими певним чином відносно головної або побічної діагоналі, вертикальної або горизонтальної вісі симетрії матриці. Правильно скласти алгоритм рішення задачі у такому випадку допоможе таблиця, наведена нижче.

№	Розташування елементів матриці	Умова при послідовній обробці	Обробка матриць		Примітка
			за рядками	за стовпцями	
1	На головній діагоналі	i=j	i:=1, n	j:=1, n	Матриця NxN 1 цикл
2	На побічній діагоналі	i+j=n+1	i:=1, n	j:=1, n	Матриця NxN
3	Вище головної діагоналі	i<j	i:=1, n-1 j:=i+1, n	j:=2, n i:=1, j-1	Матриця NxN
4	Нижче головної діагоналі	i>j	i:=2, n j:=1, n-1	j:=1, n-1 i:=j+1, n	Матриця NxN
5	Вище побічної діагоналі	i+j<n+1	i:=1, n-1 j:=1, n-i	j:=1, n-1 i:=1, n-j	Матриця NxN
6	Нижче побічної діагоналі	i+j>n+1	i:=2, n j:=n+2-i,n	j:=2, n i:=n+2-j,n	Матриця NxN
7	Правіше вертикальної вісі симетрії	j>(n+1)/2=[n1]	i:=1, m j:=n1, n	j:=n1, n i:=1, m	Матриця прямокутна MxN
8	Нижче горизонтальної вісі симетрії	i>(m+1)/2=[m1]	i:=m1, m j:=1, n	j:=1, n i:=m1, m	

Приклад 2.3.4 Визначити добуток додатних і кількість від'ємних елементів у прямокутній матриці *A*, що складається з *m* рядків і *n* стовпців. Прийняти, що нульових елементів у матриці немає.

Рішення

1. Постановка задачі.

Для рішення цієї задачі необхідно перебрати в будь-якому порядку елементи матриці *A*, з додатних елементів у процесі перебору накопичити добуток, а для від'ємних елементів організувати лічильник, що підраховує їх кількість.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Двовимірний масив	Дійсний	*A*	Вихідне дане
Кількість рядків	Цілий	*M*	Вихідне дане
Кількість стовпців	Цілий	*N*	Вихідне дане
Змінна циклу за рядками	Цілий	*i*	Допоміжна змінна
Змінна циклу за стовпцями	Цілий	*j*	Допоміжна змінна
Добуток додатних елементів	Дійсний	*P*	Результат
Кількість від'ємних елементів	Цілий	*K*	Результат

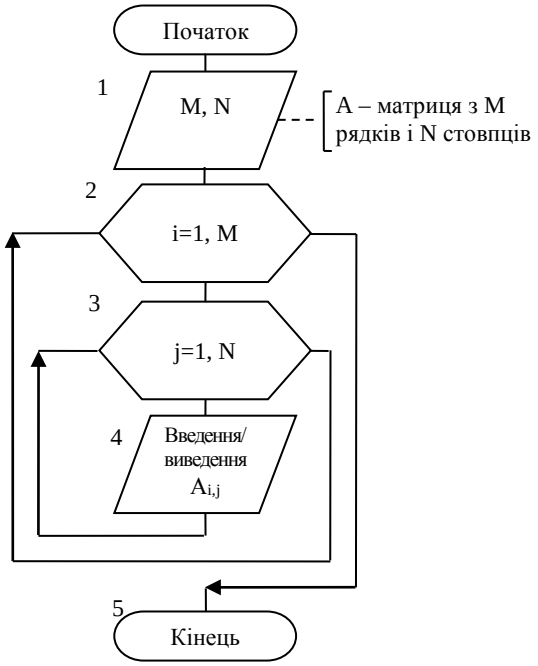
Початкове значення добутку додатних елементів дорівнює 1. Для накопичення добутку будемо використовувати формулу $P=P*A_{i,j}$. Початкове значення кількості від'ємних елементів дорівнює 0. Для підрахунку кількості будемо використовувати формулу $K=K+1$.

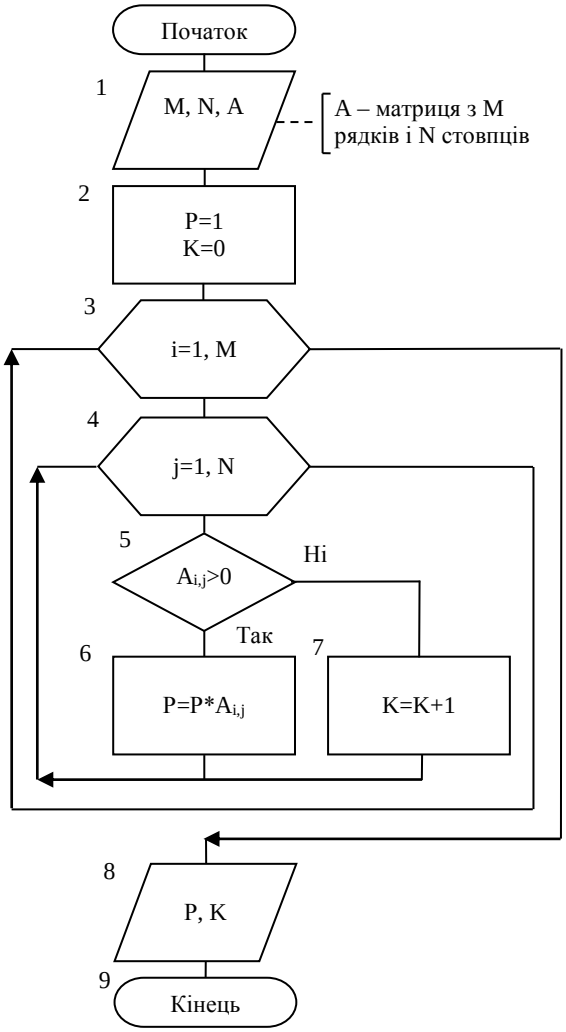
3. Розробка алгоритму розв'язання задачі.

Для реалізації алгоритму будемо використовувати варіант вкладених циклів з параметром.

У блоці 1 виконується введення кількості рядків і стовпців матриці, а також введення значень елементів двовимірного масиву.

Потрібно пам’ятати, що таке позначення введення двовимірного масиву є умовним. Насправді, введення і виведення елементів матриці - це не одноразова дія, а циклічний процес, що здійснюється у вкладених циклах. Блок-схема введення-виведення двовимірного масиву зображена на рисунку.





У блоці 2 змінним *P* і *K* присвоюються початкові значення.

У блоці 3 задаємо параметри роботи зовнішнього циклу з переадресацією. У блоці 4 задаємо параметри роботи внутрішнього циклу з переадресацією. Блоки 5-7 є тілом внутрішнього циклу. Блок 5 Рішення перевіряє, чи є поточний елемент масиву *A* додатним. Якщо відповідь «так», то в блоці 6 відбувається накопичення добутку додатних елементів *P*. Якщо відповідь «ні», то в блоці 7 відбувається підрахунок кількості від’ємних елементів *K*.

Блок 8 використовується для виведення результатів рішення задачі.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *M*=2, *N*=2, *A*=(-1;3;5;-7). Виконання дій за блок-схемою оформимо у вигляді таблиці.

Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6	Цикл 7
	Початок						
1	Введення: 2;2; (-1;3; 5;-7)						
2	P=1 K=0						
3	i=1 1<=2 так (перехід на блок 4)			i=2 2<=2 так (перехід на блок 4)			i=3 3<=2 ні (перехід на блок 8)
4	j=1 1<=2 так (перехід на блок 5)	j=2 2<=2 так (перехід на блок 5)	j=3 3<=2 ні (перехід на блок 3)	j=1 1<=2 так (перехід на блок 5)	j=2 2<=2 так (перехід на блок 5)	j=3 3<=2 ні (перехід на блок 3)	
5	-1>0 ні (перехід на блок 7)	3>0 так (перехід на блок 6)		5>0 так (перехід на блок 6)	-7>0 ні (перехід на блок 7)		
6		P=1*3=3 (перехід на блок 4)		P=3*5=15 (перехід на блок 4)			
7	K=0+1=1 (перехід на блок 4)				K=1+1=2 (перехід на блок 4)		
8							Виведення 15; 2
9							Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{

  double P;
  int i,j,M,N,K;
  setlocale(LC_ALL, "UKR");

  cout<<"M= "; cin>>M;
  cout<<"N= "; cin>>N;

  double **A = new double* [M+1];/*виділення динамічної пам'яті під матрицю A[M][N]*/
        for(i=1;i<=M;i++)
                A[i] = new double [N+1];


  for(i=1;i<=M;i++) //введення матриці A по рядкам
          for(j=1;j<=N;j++)
```

```
 {
        cout<<"A["<<i<<"]["<<j<<"]=";
        cin>>A[i][j];
 }
 P=1;
 K=0;

 for(i=1;i<=M;i++) //обробка матриці
        for(j=1;j<=N;j++)
              if(A[i][j]>0) P*=A[i][j];
              else K++;
   cout<<"Добуток додатніх елементів = "<<P<<endl;//виведення результату
 cout<<"Кількість від'ємних елементів = "<<K<<endl;
for(i=1;i<=M;i++)//очищення динамічної пам'яті
       delete A[i];
delete []A;

system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim a(), p As Single ' об’явлення динамічного масиву а
Dim m, n, i, j, k As Integer
m = CInt(Text1) ' введення з форми кількості рядків масиву а
n = CInt(Text2) ' введення з форми кількості стовпців масиву а
ReDim a(m, n) ' визначення розмірності та розміру масиву а
' введення елементів матриці по рядках
For i = 1 To m ' змінна циклу і, що відповідає номеру рядка матриці змінюється від одиниці до m, та утворює зовнішній цикл
For j = 1 To n ' змінна циклу j, що відповідає номеру стовпця матриці змінюється від одиниці до n, та утворює вкладений цикл
       a(i, j) = CSng(InputBox("Введіть(" & i & "," & j & ") елемент матриці"))
   Next j
Next i
' підрахунок добутку та кількості елементів, що відповідають завданим умовам
p = 1 ' початкове значення добутку додатних елементів матриці дорівнює 1, k = 0 ' а кількість від’ємних - 0
' обробка матриці по рядках, тобто змінна вкладеного циклу змінюється швидше, ніж змінна зовнішнього
For i = 1 To m
   For j = 1 To n
      If a(i, j) > 0 Then p = p * a(i, j) Else k = k + 1
   Next j
Next i
Print p, k
```

TP

```
program p_2_3_4;
uses CRT;
```

```
var A:array [1..100,1..100] of real; {об'явлення дійсного двовимірного масиву/матриці A, що складається з 100 рядків та 100стовпців }
      i, j, m, n, k:integer; p:real;
begin
   clrscr;
   writeln('Введіть значення m,n, та матрицю A');
   readln(m,n); { введення кількості рядків m та кількості стовпців n }
for i:=1 to m do { зовнішній цикл по рядках, номер рядка змінюється від 1 до m з кроком 1 }
       for j:=1 to n do { вкладений  цикл по стовпцях, номер стовпця змінюється від 1 до n }
          readln(a[i,j]); { введення елементів матриці A }
   k:=0; p:=1; {початкове значення добутку додатних елементів матриці дорівнює 1, а кількість від'ємних - 0 }
   for i:=1 to m do { зовнішній цикл по рядках, номер рядка змінюється від 1 до m з кроком 1 }
       for j:=1 to n do { вкладений  цикл по стовпцях, номер стовпця змінюється від 1 до n }
          if A[i,j]>0 then p:=p*A[i,j] else k:=k+1; {якщо елемент додатній,  накопичується добуток p, інакше – кількість від'ємних елементів к збільшується на одиницю}
   writeln (p, k); { виведення значення p та к }
   readln
end.
```

Приклад 2.3.5 Піднести в квадрат елементи матриці *A(M,M)*, що лежать вище побічної діагоналі.

Рішення

1. Постановка задачі.

Для вирішення цього завдання необхідно організувати зовнішній і внутрішній цикл обробки матриці таким чином, щоб обробляти тільки ті елементи матриці *A*, які лежать вище побічної діагоналі.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім'я	Призначення
Двовимірний масив	Дійсний	*A*	Вихідне дане та результат
Кількість рядків і стовпців	Цілий	*M*	Вихідне дане
Змінна циклу за рядками	Цілий	*i*	Допоміжна змінна
Змінна циклу за стовпцями	Цілий	*j*	Допоміжна змінна

Для піднесення в квадрат елементів матриці скористаємося формулою $A_{i,j}=A_{i,j}^2$.

3. Розробка алгоритму розв'язання задачі.

Для реалізації алгоритму будемо використовувати варіант вкладених циклів з параметром.

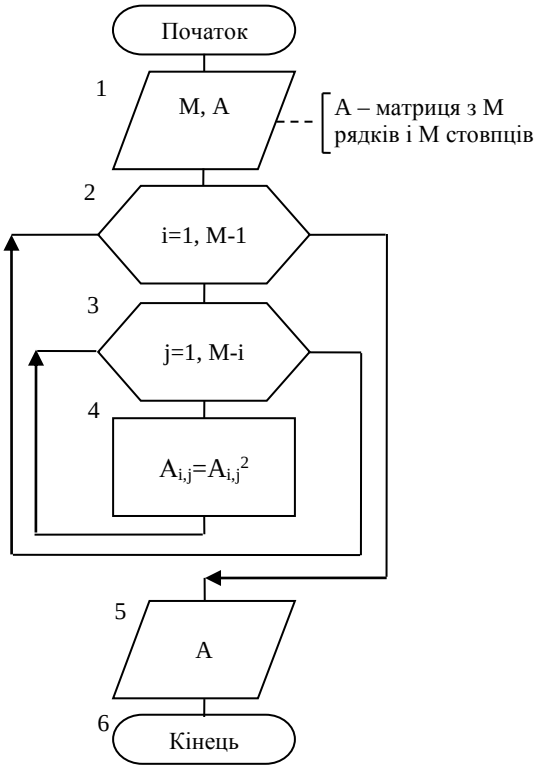
У блоці 1 виконується введення кількості рядків і стовпців матриці, а також введення значень елементів двовимірного масиву.

У блоці 2 задаємо параметри роботи зовнішнього циклу з переадресацією, а в блоці 3 задаємо параметри роботи внутрішнього циклу з переадресацією таким чином, щоб обробці підлягали тільки елементи, що лежать вище побічної діагоналі (див. довідкову таблицю).

У блоці 4 вибрані елементи матриці підносимо в квадрат $A_{i,j}=A_{i,j}^2$.

У блоці 5 перетворену матрицю виводимо.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад M=3; A=(2;4;6;5;3;7;1;8;9). Виконання дій за блок-схемою оформимо у вигляді таблиці.



Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6
	Початок					
1	Введення: 3; (2; 4; 6 5; 3; 7 1; 8; 9)					
2	i=1 1<=2 так (перехід на блок 3)			i=2 2<=2 так (перехід на блок 3)		i=3 3<=2 ні (перехід на блок 5)
3	j=1 1<=2 так (перехід на блок 4)	j=2 2<=2 так (перехід на блок 4)	j=3 3<=2 ні (перехід на блок 2)	j=1 1<=1 так (перехід на блок 4)	j=2 2<=1 ні (перехід на блок 2)	
4	$A_{1,1}=2^2=4$ (перехід на блок 3)	$A_{1,2}=4^2=16$ (перехід на блок 3)		$A_{2,1}=5^2=25$ (перехід на блок 3)		
5						Виведення (4; 16; 6 25; 3; 7 1; 8; 9)
6						Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
using namespace std;
int main()
{
```

```
 int i,j,M;
 setlocale(LC_ALL, "UKR");

 cout<<"M= "; cin>>M;

 double **A = new double* [M+1];/*виділення динамічної пам'яті під матрицю A[M][M]*/
        for(i=1;i<=M;i++)
                A[i] = new double [M+1];

 for(i=1;i<=M;i++) //введення матриці A по рядкам
          for(j=1;j<=M;j++)
 {
          cout<<"A["<<i<<"]["<<j<<"]=";
          cin>>A[i][j];
 }

 for(i=1;i<M;i++) //обробка матриці
          for(j=1;j<=M-i;j++)
                  A[i][j]=A[i][j]*A[i][j];

  for(i=1;i<=M;i++) //ввиведення матриці A
  {
           for(j=1;j<=M;j++)
      cout<<A[i][j]<<'\t';
   cout<<endl;
 }

for(i=1;i<=M;i++)//очищення динамічної пам'яті
          delete A[i];
delete []A;

system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim a() As Single, m, i, j As Integer
m = CInt(Text1)
ReDim a(m, m)
' введення елементів матриці по рядках
For i = 1 To m
    For j = 1 To m
        a(i, j) = CSng(InputBox("Введіть (" & i & "," & j & ") елемент матриці"))
    Next j
Next i
' виведення елементів вихідної матриці по рядках, формування строкового представлення матриці
matr = ""
For i = 1 To m
    For j = 1 To m
```

```
        matr = matr + CStr(a(i, j)) + "   "
    Next j
    matr = matr + vbCrLf   ' наприкінці кожного рядка додається символ кінця строки та переведення каретки
Next i
' обробка матриці, та формування нової матриці, шляхом піднесення до квадрата кожного елемента
For i = 1 To m - 1
    For j = 1 To m - i
        a(i, j) = a(i, j) ^ 2
    Next j
Next i
' виведення елементів результуючої матриці по рядках, формування строкового представлення матриці
matr1 = ""
For i = 1 To m
    For j = 1 To m
        matr1 = matr1 + CStr(a(i, j)) + "   "
    Next j
matr1 = matr1 + vbCrLf
Next i
End Sub
```

TP

```
program p_2_3_5;
uses CRT;
var A:array [1..100,1..100]of real; i, j, m:integer;
begin
    clrscr;
    writeln('Введіть кількість рядків/стовпців квадратної матриці A');
    readln(m);
    for i:=1 to m do
        for j:=1 to m do
            readln(a[i,j]); {введення матриці A}
    { виведення на екран початкової матриці A}
    for i:=1 to m do begin
        for j:=1 to m do
            write(A[i,j]:4:1);
        writeln;
    end;
    for i:=1 to m-1 do
        for j:=1 to m-i do
            A[i,j]:=sqr(A[i,j]); {елементи матриці A, що знаходяться вище побічної діагоналі, підводяться до квадрату }
    { виведення на екран результуючої матриці A}
    for i:=1 to m do begin
        for j:=1 to m do
            write(A[i,j]:4:1);
    writeln; {виводимо построково матрицю A}
    end;
    readln
end.
```

Ітераційні цикли

Ітераційним називається такий вид простого циклу, який має наступні ознаки:

1. Кількість повторень циклу заздалегідь невідома, тому що вихід з циклу визначається отриманим результатом.

2. Значення, отримані в поточному циклі, стають вихідними даними для наступного циклу.

3. Остаточне значення результату набувається одночасно з виходом із циклу.

Подібні обчислення найчастіше виникають при рішенні задач методом ітерацій. Хорошим прикладом ітераційного циклу також є задачі на обчислення суми ряду або визначення величини чергового члена ряду із заданою точністю.

Приклад 2.3.6 Визначити номер того члена ряду, абсолютна величина якого не перевищує деякого заданого малого числа ε ($10^{-3} > \varepsilon > 10^{-4}$).

$$\tilde{o} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \ldots + (-1)^{n-1}\frac{x^n}{n!} + \ldots$$

Рішення

1. Постановка задачі.

Вираз $(-1)^{n-1}\frac{x^n}{n!}$ називається формулою загального члена ряду. З її допомогою можна визначити значення будь-якого *n*-го члена ряду, підставляючи в неї величину аргументу *x* і замість *n* номер потрібного члена ряду.

Задача зводилася б до простого обчислення чергового члена ряду до досягнення умови виходу з циклу, але:

- у знаменнику формули загального члена є факторіал;
- потрібно (-1) підносити в довільний ступінь.

Як ми вже знаємо, накопичення факторіала - це окремий циклічний процес накопичення добутку, що ускладнює розв’язання задачі.

Крім того, у деяких мовах програмування безпосередньо піднести до ступіню число (-1) просто неможливо, потрібно застосовувати спеціальні методи.

Вирішити ці проблеми можна за допомогою знаходження та використання рекурентної формули для ряду.

Рекурентна формула для ряду - це вираз, що пов'язує будь-які два сусідніх члена ряду.

Знайдемо рекурентну формулу для заданого ряду. Введемо наступні позначення: U_n, U_{n+1} – попередній і наступний члени ряду (сусіди).

Записуємо формулу загального члена ряду для попереднього члена:

$$U_n = (-1)^{n-1}\frac{x^n}{n!}$$

Записуємо формулу загального члена ряду для наступного члена:

$$U_{n+1} = (-1)^{n+1-1}\frac{x^{n+1}}{(n+1)!} = (-1)^n\frac{x^{n+1}}{(n+1)!}$$

Тоді рекурентна формула виводиться як частка від ділення наступного члена ряду на попередній:

$$R = \frac{U_{n+1}}{U_n} = \frac{(-1)^n x^{n+1}}{(n+1)!}\cdot\frac{n!}{(-1)^{n-1}x^n} = -\frac{x}{n+1}$$

Перетворення у формулі ведуться з урахуванням того, що:

- при діленні дробів показники ступенів віднімаються;
- старше значення факторіала необхідно розкласти до молодшого:

$$(n+1)! = (n+1)\cdot n!$$

Тепер, знаючи попередній член ряду і рекурентну формулу, можна знайти наступний член ряду, повторюючи ці дії до досягнення заданої точності.

2. Побудова математичної моделі.

Складемо таблицю імен змінних.

Змінна	Тип	Ім’я	Призначення
Аргумент	Дійсний	*X*	Вихідне дане
Точність	Дійсний	*E*	Вихідне дане
Значення члена ряду	Дійсний	*U*	Допоміжна змінна
Номер члена ряду	Цілий	*n*	Результат

3. Розробка алгоритму розв’язання задачі.

Для реалізації алгоритму будемо використовувати варіант простого циклу з верхнім закінченням, тобто цикл з передумовою.

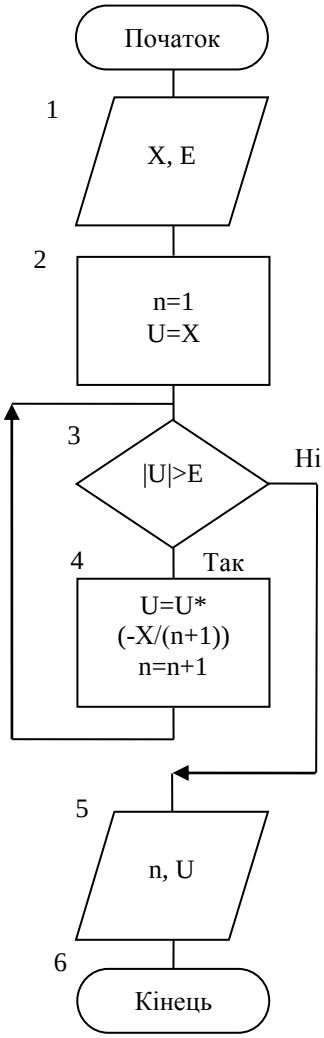
У блоці 1 виконується введення вихідних даних: аргументу *X* і точності *E*.

У блоці 2 виконуються початкові присвоєння: значенню члена ряду *U* присвоюємо значення першого доданка ряду x, номеру члена ряду *n* присвоюємо значення 1.

У блоці 3 перевіряється умова продовження циклу. Якщо вона виконується, то переходимо до блоку 4, де за допомогою рекурентної формули обчислюємо значення наступного члена ряду. Номер члена ряду відповідно збільшується на 1.

Якщо умова у блоці 3 не виконується, це означає, що обчислення досягли заданої точності. Тому здійснюється перехід до блоку 5 і виведення отриманих результатів.

Перевіримо правильність алгоритму на конкретних, довільно взятих значеннях вихідних даних, наприклад *X*=1; *E*=0,001. Виконання дій за блок-схемою оформимо у вигляді таблиці.



Блок	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5	Цикл 6	Цикл 7
	Початок						
1	Введення: 1; 0,001						
2	U=1 n=1						
3	|1|>0,001 так (перехід на блок 4)	|-0,5|>0,001 так (перехід на блок 4)	|0,167|>0,001 так (перехід на блок 4)	|-0,0417|>0,001 так (перехід на блок 4)	|0,0083|>0,001 так (перехід на блок 4)	|-0,0014|>0,001 так (перехід на блок 4)	|0,0002|>0,001 ні (перехід на блок 5)
4	U=-0,5 n=2 (перехід на блок 3)	U=0,167 n=3 (перехід на блок 3)	U=-0,0417 n=4 (перехід на блок 3)	U=0,0083 n=5 (перехід на блок 3)	U=-0,0014 n=6 (перехід на блок 3)	U=0,0002 n=7 (перехід на блок 3)	
5							Виведення 7; 0,0002
6							Кінець

Отриманий результат збігається з очікуваним, отже, схема алгоритму складена вірно.

C++

```
#include<iostream>
#include<clocale>
#include<cmath>//бібліотека для роботи з математичними функціями
using namespace std;
int main()
{
```

```
  double X,E,U;
  int n;
  setlocale(LC_ALL, "UKR");
    cout<<"Аргумент X = "; cin>>X;
  cout<<"Точність E = "; cin>>E;
  U=X;
  n=1;
  while(fabs(U)>E)// fabs – модуль від числа дійсного типу
  {
          U*=-X/(n+1);//розрахунок наступного члена ряду
          n++;//збільшення кількості на 1
  }
  cout<<"n="<<n<<endl;//виведення результату
  cout<<"U="<<U<<endl;
system("pause");
return 0;
}
```

VB

```
Private Sub Command1_Click()
Dim x, e, u As Single, n As Integer
x = CSng(Text1)
e = CSng(Text2)
u = x  ' перший член ряду дорівнює x
n = 1 ' його номер - 1
Do While Abs(u) > e ' доки логічна умова істинна, виконується тіло циклу
    u = u * (-x / (n + 1)) ' наступний член ряду обчислюється як попередній, помножений на спільний співмножник
    n = n + 1 ' номер поточного члена ряду збільшується на одиницю
Loop ' кінець циклу, повернення до перевірки умови
Print n, u
End Sub
```

TP

```
program p_2_3_6;
uses CRT;
var x,u,e:real; n:integer;
begin
    clrscr;
    writeln('Введіть значення x і точність обчислень e');
    readln(x,e);
    u:=x; n:=1; {перший член ряду дорівнює x, його номер -1 }
    While  abs(u)>e do {доки виконується логічна умова, тіло циклу повторюється}
    begin { початок тіла циклу }
        u:=u*(-x/(n+1)); {наступний член ряду обчислюється як попередній, помножений на спільний співмножник}
        n:=n+1; { номер поточного члена ряду збільшується на одиницю }
    end; { кінець тіла циклу }
    writeln('n=',n,'u=':5,u:4:2);
    readln
end.
```

3 МОВИ ПРОГРАМУВАННЯ

3.1 C++

3.1.1 Вступ

Мова C++ виникла на початку 1980-х років, коли співробітник фірми Bell Labs Бьерн Страуструп створив ряд удосконалень до мови C під власні потреби.

C++ поєднує властивості як високорівневих, так і низькорівневих мов. У порівнянні з його попередником- мовою C,- найбільша увага приділена підтримці об'єктно-орієнтованого й узагальненого програмування. Підтримує такі парадигми програмування як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування, забезпечує модульність, роздільну компіляцію, обробку виключень, абстракцію даних, оголошення типів (класів) об'єктів, віртуальні функції. Стандартна бібліотека включає, у тому числі, загальновживані контейнери й алгоритми.

C++ широко використовується для розробки програмного забезпечення, будучи одним із самих популярних мов програмування. Область його застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для систем, що вбудовуються, високопродуктивних серверів, а також розважальних додатків (ігор).

3.1.2 Робота в середовищі Visual Studio 2008

Стартова сторінка Стартова сторінка (рис. 3.1) містить декілька Web-Частин. Почнемо з лівого верхнього кута. Тут є область недавніх проектів **Recent Projects.** Звідси ви можете запустити проект, над яким недавно працювали, або створити новий. Нижче знаходиться область **Getting Started** (Для початківців). Ця Web-Частина корисна в тому випадку, якщо ви шукаєте можливості навчання. Тут можна почати з вивчення питань для початківця, нових можливостей, або розділу **"How Do I...?"** (" Як мені зробити, щоб...?"). Ще нижче знаходиться область останніх новин Visual Studio. Нарешті, у центрі сторінки знаходяться останні новини й пропозиції від MSDN.

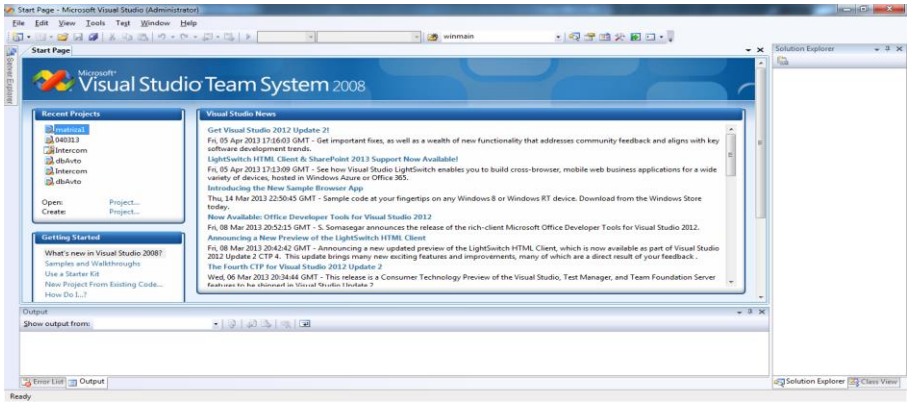


Рисунок 3.1 – Стартова сторінка

Опції запуску

Якщо вам просто не подобається стартова сторінка або ви волієте напряму попадати в проект, то ви можете настроїти те, що відбувається при завантаженні інтегрованого середовища розробки. У діалоговім вікні **Options (Tools | Options)** виберіть вузол **Environment**, а потім **Startup**. На рисунку 3.2 показані деякі з опцій, наявних для запуску середовища.

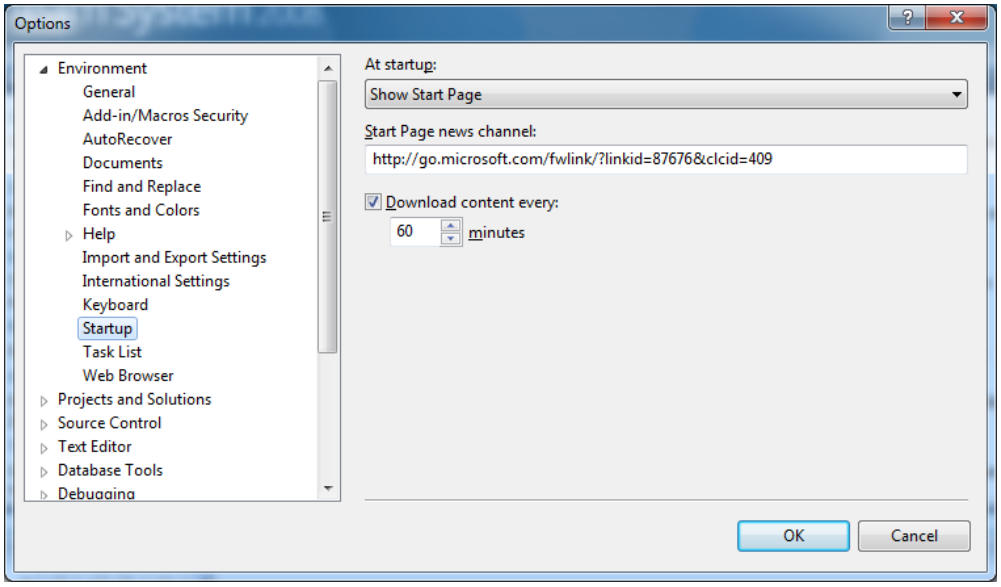


Рисунок 3.2 – Опції запуску середовища Visual Studio 2008

Тут ви можете настроїти - звідки брати новини для вашої стартової сторінки. Ви можете також дати вказівки середовищу завантажити останнє рішення, показати діалогове вікно нового проекту або вікно відкриття проекту, відкрити домашню сторінку вашого браузера, або не робити взагалі нічого (показати порожнє середовище). Ви можете також настроїти — як часто буде ваш контент автоматично обновлятися із сервера.

Ваш перший проект

Наступний природній крок— створити ваш перший проект. Це швидко познайомить вас із деякими основними функціональними можливостями керування проектами й файлами інтегрованого середовища. З меню **File** ви можете створити новий проект. Проекти — це просто шаблони, які групують файли для виконуваних додатків під Windows, Office або мобільні пристрої.

Зверніть увагу, що компонування інтегрованого середовища майже не має специфіки. Для ваших перших додатків вона буде схожою (поки ви все не настроїте).

Для створення нового проекту (рис. 3.3) вибираємо **File/New/Project** або натискаємо комбінацію клавіш **Ctrl+Shift+N**.

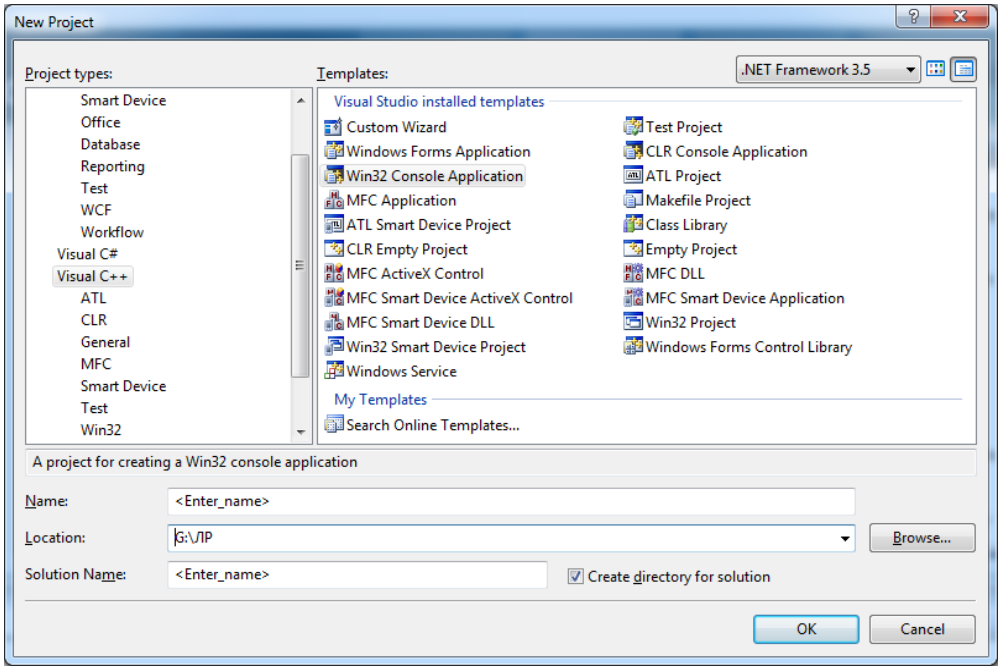


Рисунок 3.3 – Створення нового проекту

Вибираємо закладку **Visual C++**. У правому вікні вказуємо **Win32 Console Application** (або **Empty Project**).

У поле **«Name»** пишемо назву проекту. Вибір імені проекту може бути досить довільним: припустимо використовувати числове значення, а також припустиме ім'я задавати через букви російського алфавіту.

У поле **«Location»** записуємо шлях для збереження проекту або вибираємо шлях через кнопку **«Browse»**. За замовчуванням проект зберігається в папці **Projects**.

Після натискання кнопки **«Ok»** відкривається вікно **«Майстра створення додатка»** (рисунок 3.4).

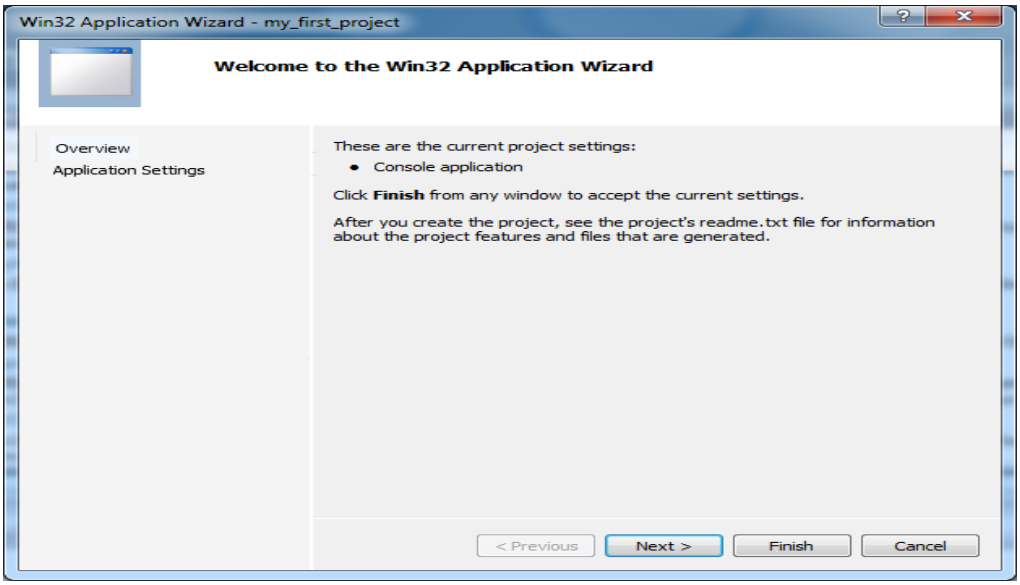


Рисунок 3.4 – Майстер створення додатка

Після звертання до сторінки **Application Settings** або після натискання кнопки «**Next**» відкривається наступне вікно налаштувань додатка.

У додаткових опціях (**Additional options**) слід поставити галочку в поле **Empty project** (порожній проект) і зняти (забрати) галочку в поле **Precompiled header**, як видно на рисунку 3.5.

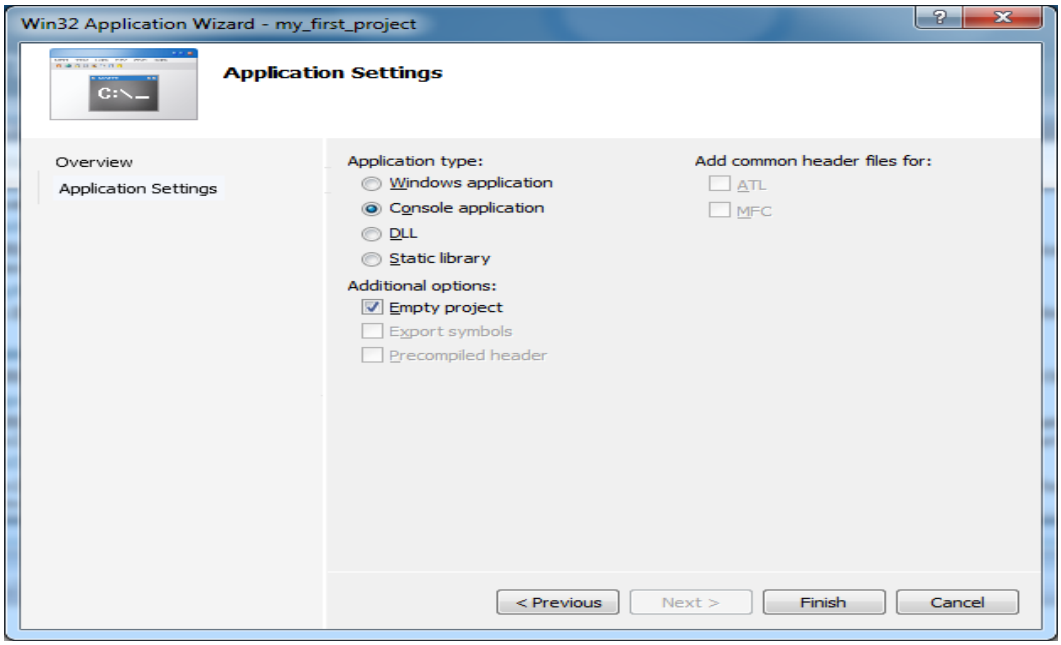


Рисунок 3.5 – Додаткові опції

Натискаючи на клавішу **«Finish»** створюємо наш проект.

Якщо на рисунку ви вибрали **«Empty Project»**, а не «Win32 Console Application», то буде створений такий же порожній проект тільки без «Майстра створення додатків».

Отже, проект створений і має вигляд (рисунок 3.6).

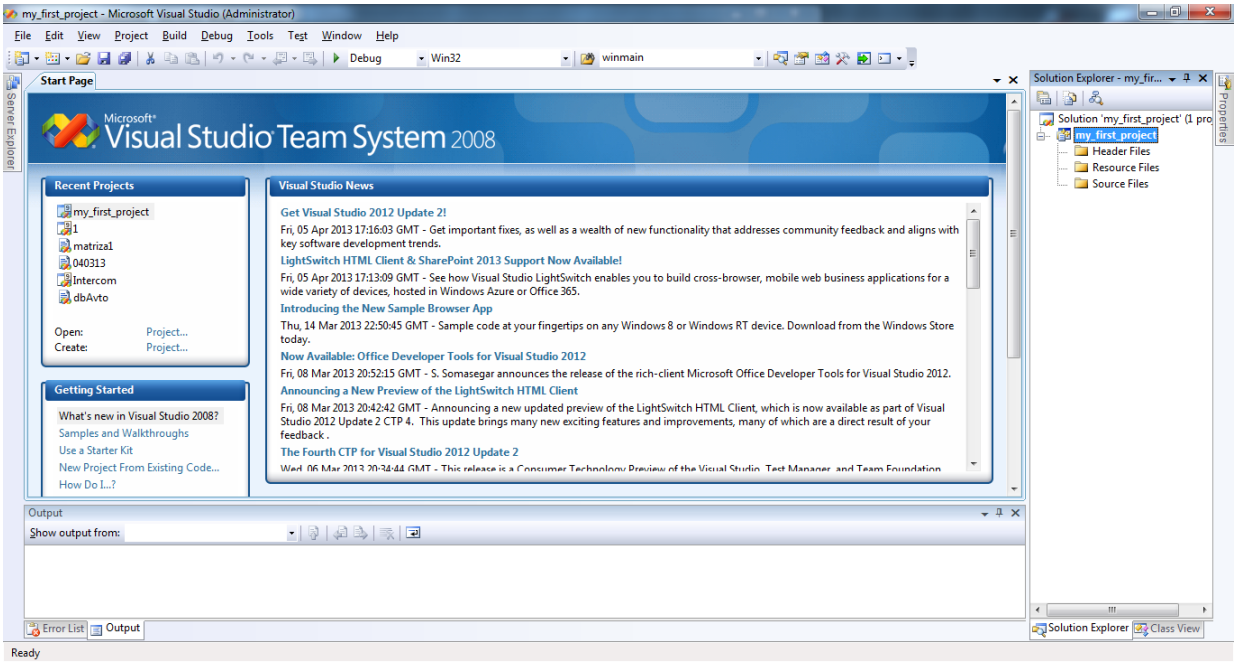


Рисунок 3.6 – Новий проект Visual Studio 2008

Додати новий елемент до проекту: Правою кнопкою миші відкриваємо меню **«Sours Files»** і вибираємо **«Add/New Item»**. Відкривається вікно вибору типу файлу для підключення до проекту. Вибираємо файл C++ і вказуємо ім'я (рисунок 3.7).

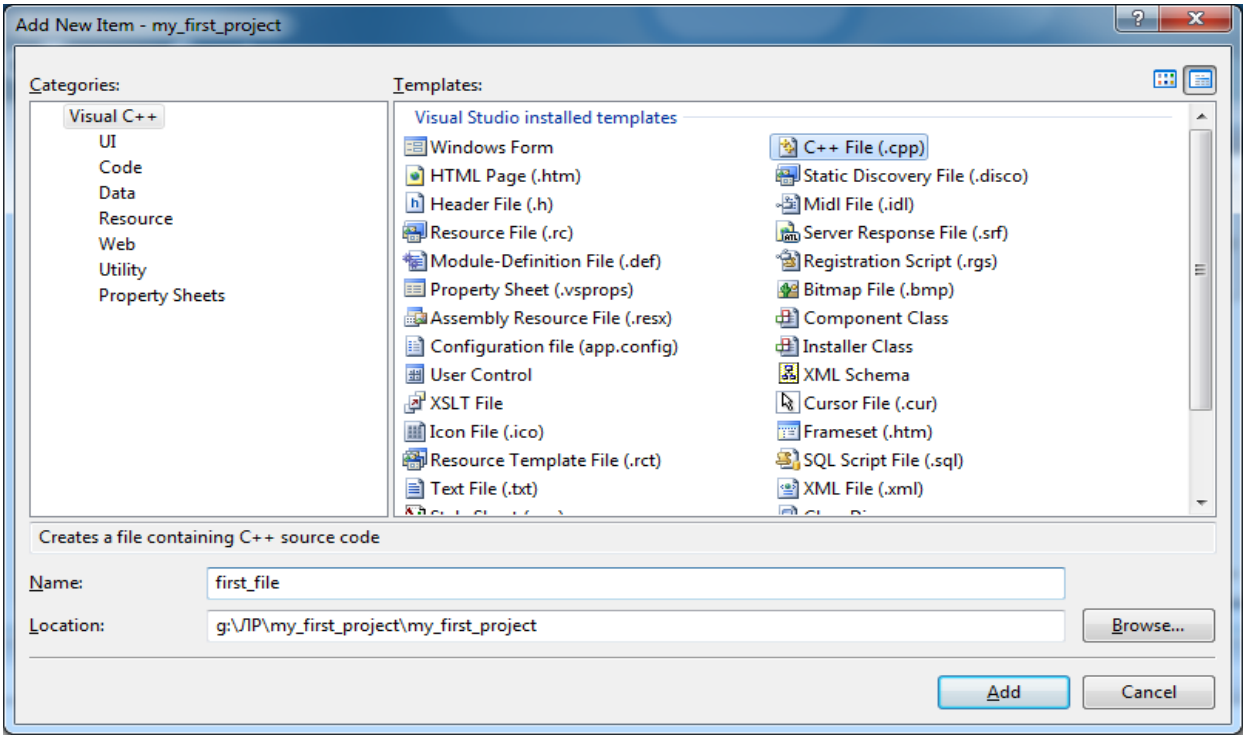


Рисунок 3.7 – Додавання нового елемента до проекту

При натисканні на клавішу **«Add»** додаємо наш файл до проекту.

Доданий файл відображається в дереві **«Solution Explorer»** під вузлом **«Source Files»** (файли з вихідним кодом), і для нього автоматично відкривається редактор (рисунок 3.8).

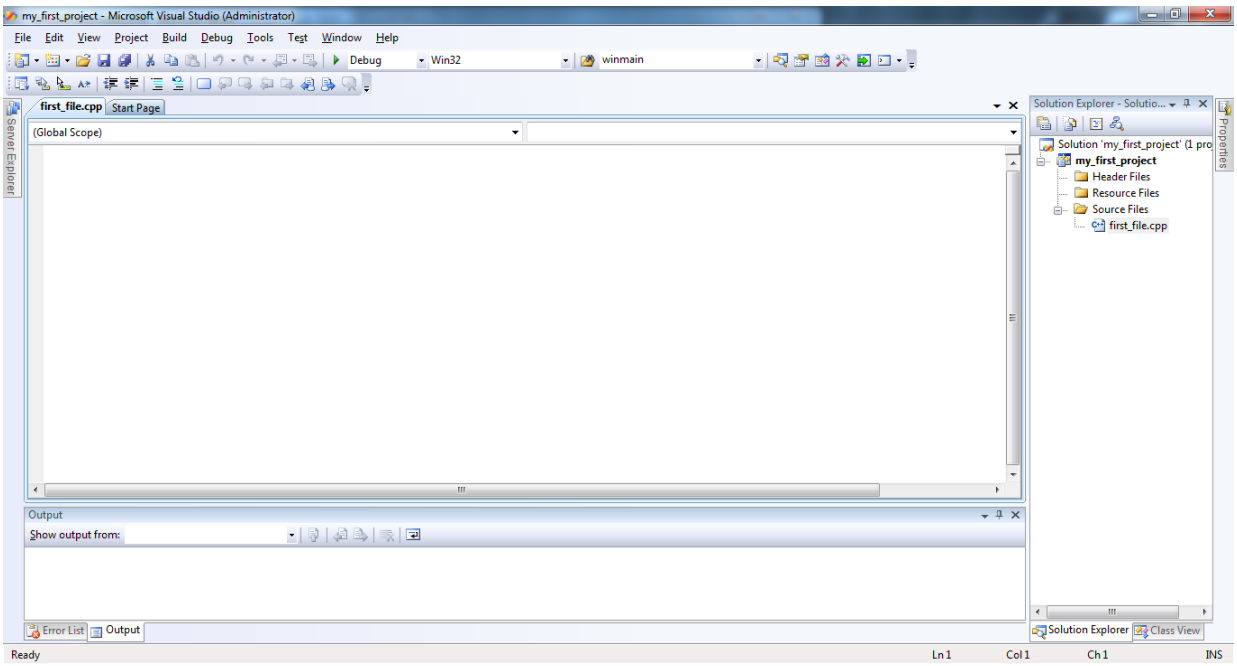


Рисунок 3.8 – Редактор коду для доданого файлу

Приклад: Вивести на екран повідомлення «Привіт, світ!».

Лістинг програми:

```
#include<iostream> //бібліотека для функції cout
#include<clocale>//бібліотека для функції setlocale
using namespace std;
int main()
{
    setlocale(LC_ALL,"RUS");//установка  шрифту
    cout<<"Привіт, світ!"<<endl;// Вивід на екран строкової константи,
                                //endl - перехід на новий рядок

system("pause");
return 0;
}
```

Примітка:

// - коментує один рядок

/* … */ - коментує n-ну кількість рядків

На рисунку 3.9 показаний результат роботи програми.

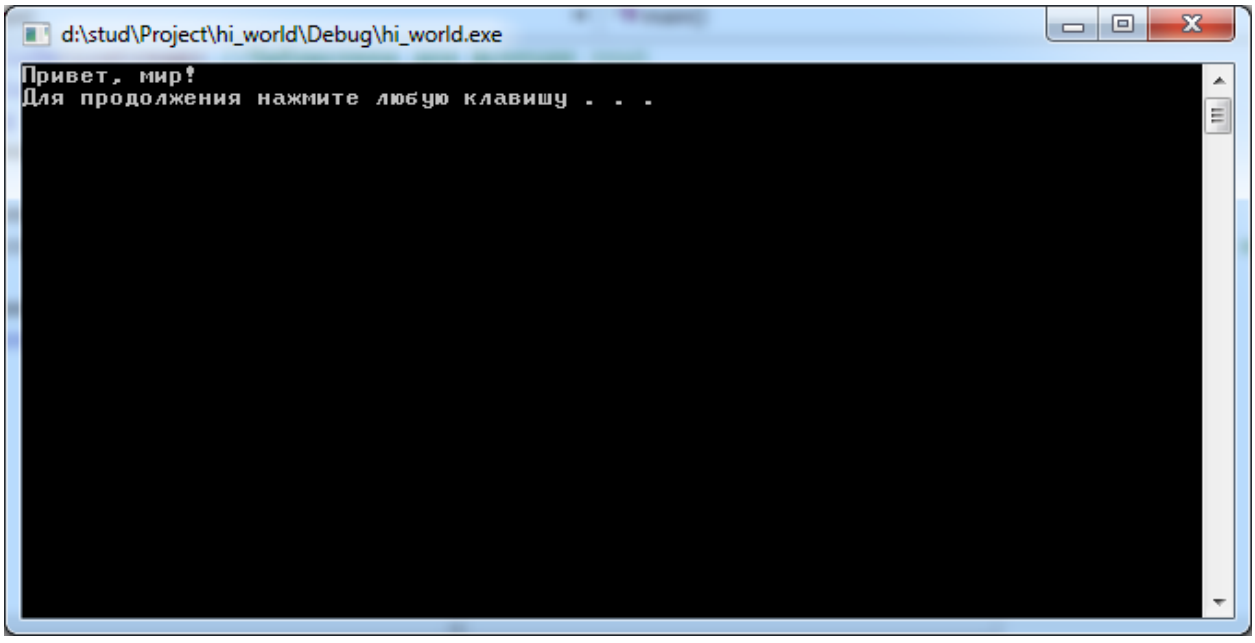


Рисунок 3.9 – Вивід повідомлення «Привіт, світ!» на екран

Збереження проекту

Для того щоб зберегти проект вибираємо пункт меню File/Save або File/Save All.

3.1.3 Структура програми

Програма мовою С++ складається з директив препроцесора, вказівок компілятору, оголошень змінних і/або констант, оголошень і визначень функцій.

Оголошення змінної

Оголошення змінної задає ім'я й атрибути змінної. Атрибутами змінної можуть бути тип, кількість елементів (для масивів), специфікація класу пам'яті, а також *ініціалізатор*. ***Ініціалізатор*** – це константа відповідного типу, що задає значення, яке присвоюється змінній при створенні.

Оголошення змінної має наступний синтаксис: [<*специфікація класу пам'яті*>] <*тип*> <*ім'я*> [= < *ініціалізатор*>] [,<*ім'я*> [= < *ініціалізатор* >] **...**];

Приклади оголошення змінних

int x;	// Оголошення змінної цілого типу без *ініціалізатора*
double y = exp(1);	// Змінна дійсного типу ініціалізується числом e. // exp(x) – функція, що обчислює e^x.
int a, b = 0;	// Оголошення двох змінних цілого типу. Змінна *b* ініціалізується значенням 0.

У мові С++ немає обмежень на кількість символів в імені. Однак деякі частини реалізації недоступні авторові компілятора, і вони іноді накладають такі обмеження.

Константи

У мові С++ введена концепція обумовлених користувачем констант для вказівки на те, що значення не можна змінити безпосередньо. Це може бути корисно в декількох відносинах. Наприклад, багато об'єктів не міняються після ініціалізації; використання символічних констант приводить до більш зручного використання в супроводі коду, чим застосування літералів безпосередньо в тексті програми; покажчики часто використовуються тільки для читання, але не для запису; більшість параметрів функцій читаються, але не перезаписуються.

Щоб оголосити об'єкт константою, в оголошення потрібно додати ключове слово *const*. Так як константі не можна присвоювати значення, вона повинна бути ініціалізованою.

const int a = 100;	// *a* є константою	
const int b[] = {1, 2, 3, 4, 5};	// Усі *b[i]* є константами	
const int c;	// Помилка – немає ініціалізатора!	

Типовим є використання констант як розміру масивів і міток в інструкції *case*.

Відзначимо, що *const* модифікує тип, тобто обмежує можливе використання об'єкта, але не вказує спосіб розміщення константного об'єкта. Простим і типовим використанням константи є той випадок, коли значення константи відомо під час компіляції й під неї не потрібне виділення пам'яті. Для масиву констант, як правило, потрібне виділення пам'яті, тому що, у загальному випадку, компілятор не в змозі визначити, до якого елемента масиву відбувається звернення у виразі.

Оголошення typedef

Оголошення, що починається із ключового слова *typedef*, уводить нове ім'я для типу, не для змінної даного типу. Метою такого оголошення часто є призначення короткого синоніма для часто використовуваного типу. Наприклад, при частому застосуванні unsigned char можна ввести синонім uchar.

```
typedef unsigned char    // Тепер uchar – синонім для unsigned
uchar;                   char
```

Імена, що вводяться за допомогою *typedef*, є **синонімами, а не новими типами**. Отже, старі типи можна використовувати разом з їхніми синонімами. Якщо вам потрібні різні типи з однаковою семантикою або з однаковим представленням, зверніться до перерахувань або класів.

Оголошення й визначення функції

Оголошення функції задає ім'я функції, тип значення, що вертається, і кількість і типи параметрів, які повинні бути присутніми при виклику функції. Вказівка void у якості значення, що вертається, означає, що функція не повертає значення.

Визначенням функції є оголошення функції, у якому є присутнім тіло функції. Визначення функції має наступний синтаксис:
<*тип*> <*ім'я*> **(**<*список формальних параметрів*>**) {** [<*оголошення*>] [<*оператори*>] **}**

Типи у визначенні й оголошеннях функції повинні збігатися. Однак, імена параметрів не є частиною типу й не зобов'язані збігатися.

Усі функції в програмі існують на глобальному рівні й не можуть бути вкладені одна в одну.

Серед функцій виділяється одна головна функція, яка повинна мати ім'я *main*. З неї починається виконання програми, звичайно вона керує виконанням програми, організовуючи

виклики інших функцій. Для того щоб програма могла бути скомпільована й виконана, вона повинна містити, принаймні, визначення функції *main*.

Приклади визначення функції

```
double Cube(double x);                          // // Оголошення (прототип) функції
void main() { printf("%lf\n", Cube(5));         // Головна функція програми, яка друкує 5 в кубі
}
double Cube(double x) { return x * x * x; }     // Функція одного дійсного аргументу, яка повертає дійсне значення, // рівне кубу аргументу
```

При виклику функції виділяється пам'ять під її формальні параметри, і кожному формальному параметру привласнюється значення відповідного фактичного параметра. Семантика передачі параметрів ідентична семантиці ініціалізації. Зокрема, перевіряється відповідність типів формальних і фактичних параметрів і при необхідності виконуються або стандартні, або визначені користувачем перетворення типів. Існують спеціальні правила для передачі масивів як параметрів, засоби для передачі параметрів, відповідність типів для яких не перевіряється, і засоби для завдання параметрів за замовчуванням.

Функція повинна повертати значення, якщо вона не оголошена як void. І навпаки – значення не може бути повернено з функції, якщо вона оголошена як void. Як і передача параметрів, семантика повернення значення з функції ідентична семантиці ініціалізації значення, що повертається, задається інструкцією return.

```
int  f1() { } void f2() { }                     // Помилка – не повертається значення // Правильно
int f3() { return 1; } void f4() { return 1; }  // Правильно // Помилка – значення повертається у функції void
int  f5() { return; } void f6() { return; }     // Помилка – не зазначене значення, що повертається, // Правильно
```

Функція з типом void не може повертати значення. Однак виклик функції з типом void не дає значення, так що функція з типом void може використовувати виклик функції з типом void як вираз в інструкції return.

```
void g() { ... } void h() { return g(); }       // Правильно
```

Така форма інструкції return важлива при написанні шаблонів функцій, коли тип значення, що повертається, є параметром шаблону.

Препроцесор

Препроцесор – це програма, яка обробляє текст вашої програми до компілятора. Таким чином, на вхід компілятора потрапляє текст, який може відрізнятися від того, який бачите Ви. Робота препроцесора управляється директивами. За допомогою препроцесора можна виконувати наступні операції:

- включення в програму текстів із зазначених файлів;
- заміна ідентифікаторів послідовностями символів;
- макропідстановка, тобто заміна позначення параметризованим текстом, що формується препроцесором з урахуванням конкретних аргументів;
- виключення із програми окремих частин тексту (умовна компіляція).

Включення файлів

Включення файлів проводитися за допомогою директиви *#include*, яка має наступний синтаксис:

#include <*шлях*> **#include** *"шлях"*

Кутові дужки тут є елементом синтаксису.

Директива *#include* включає вміст файлу, шлях до якого заданий, у компільований файл замість рядка з директивою. Якщо шлях укладений у кутові дужки, то пошук файлу здійснюється в стандартних директоріях. Якщо шлях взятий у лапках й заданий повністю, то пошук файлу здійснюється в заданій директорії, а якщо шлях повністю не заданий – у поточній директорії. За допомогою цієї директиви Ви можете включати в текст програми як стандартні, так і свої файли.

Крім того, як було зазначено вище, у мові C++ ряд функцій, такі як функції введення/виводу, динамічного розподілу пам'яті і т.ін., не є елементом мови, а входять у стандартні бібліотеки. Для того щоб користуватися функціями стандартних бібліотек, необхідно в текст програми включати так звані заголовні файли (в описі кожної функції вказується, який заголовний файл необхідний для неї). Це також робиться за допомогою директиви препроцесора *#include*.

Макропідстановки

Макропідстановки реалізуються директивою *#define*, яка має наступний синтаксис:

#define <*ідентифікатор*> <*текст*> **#define** <*ідентифікатор*>(<*список параметрів*>) <*текст*>

Директива *#define* заміняє всі входження *ідентифікатора* у вихідному файлі на *текст*, наступний в директиві за *ідентифікатором*. Цей процес називається макропідстановкою. *Ідентифікатор* заміняється лише в тому випадку, якщо він являє собою окрему лексему. Наприклад, якщо *ідентифікатор* є частиною рядка або більш довгого ідентифікатора, він не заміняється.

3.1.4 Типи даних C++

Літерали

У C++ є набір вбудованих типів даних для представлення цілих і дійсних чисел, символів, а також тип даних "символьний масив", який служить для зберігання символьних рядків. Тип char служить для зберігання окремих символів і невеликих цілих чисел. Він займає один машинний байт. Типи short, int і long призначені для представлення цілих чисел. Ці типи різняться тільки діапазоном значень, які можуть приймати числа, а конкретні розміри перерахованих типів залежать від реалізації. Звичайно short займає половину машинного слова, int – одне слово, long – одне або два слова. В 32-бітних системах int і long, як правило, одного розміру.

Типи float, double і long double призначені для чисел із плаваючою крапкою й різняться точністю представлення (кількістю значущих розрядів) і діапазоном. Звичайно float (одинарна точність) займає одне машинне слово, double (подвійна точність) – два, а long double (розширена точність) – три.

char, short, int і long разом становлять *цілі типи*, які, у свою чергу, можуть бути *знаковими* (signed) і *беззнаковими* (unsigned). У знакових типах самий лівий біт служить для зберігання знака (0 – плюс, 1 – мінус), а біти, що залишилися, містять значення. У беззнакових типах усі біти використовуються для значення. 8-бітовий тип signed char може представляти значення від -128 до 127, а unsigned char – від 0 до 255.

Коли в програмі зустрічається деяке число, наприклад 1, то це число називається *літералом*, або *літеральною константою*. Константою, тому що ми не можемо змінити його значення, і літералом, тому що його значення фігурує в тексті програми. Літерал є неадресованою величиною: хоча реально він, звичайно, зберігається в пам'яті машини, немає ніякого способу довідатися його адресу. Кожний літерал має певний тип. Так, 0 має тип int, 3.14159 – тип double.

Літерали цілих типів можна записати в десятковому, восьмеричному й шістнадцятиричному вигляді. Ось як виглядає число 20, представлене десятковим, восьмеричним і шістнадцятиричним літералами:

20// десятковий

024// восьмеричний

0x14 // шістнадцятиричний

Якщо літерал починається з 0, він трактується як восьмеричний, якщо з 0x або 0X, то як шістнадцятиричний. Звичний запис розглядається як десяткове число.

За замовчуванням усі цілі літерали мають тип signed int. Можна явно визначити цілий літерал як тип long, приписавши наприкінці числа букву L (використовується як прописна L, так і рядкова l, однак для зручності читання не слід вживати рядкову: її легко переплутати з 1(одиницею)).

Буква U (або u) наприкінці визначає літерал як unsigned int, а дві букви – UL або LU – як тип unsigned long. Наприклад:

128u 1024UL 1L 8Lu

Літерали, що представляють дійсні числа, можуть бути записані як з десятковою крапкою, так і в науковій (експонентній) нотації. За замовчуванням вони мають тип double. Для явної вказівки типу float потрібно використовувати суфікс F або f, а для long double - L або l, але тільки у випадку запису з десятковою крапкою. Наприклад:

3.14159F 0/1f 12.345L 0.0

3el 1.0E-3E 2.1.0L

Слова true і false є літералами типу bool.

Літеральні символьні константи записуються як символи в одинарних лапках. Наприклад:

'a' '2' ',' ' ' (пробіл)

Спеціальні символи (табуляція, повернення каретки) записуються як escape-послідовності. Визначені наступні послідовності (вони починаються із символу зворотної косої риски):

новий рядок \n

горизонтальна табуляція \t

забій \b

вертикальна табуляція \v

повернення каретки \r

прогін аркуша \f

дзвінок \a

зворотна коса риска \\

питання \?

одиночні лапки \'

подвійні лапки \"

escape-послідовність загального виду має форму \ooo, де ooo – від однієї до трьох восьмеричних цифр. Це число є кодом символу. Використовуючи Ascii-Код, ми можемо написати наступні літерали:

\7 (дзвінок) \14 (новий рядок)

\0 (null) \062 ('2')

Символьний літерал може мати префікс L (наприклад, L'a'), що означає спеціальний тип wchar_t – двобайтовий символьний тип, який застосовується для зберігання символів національних алфавітів, якщо вони не можуть бути представлені звичайним типом char, як, наприклад, китайські або японські літери.

Строковий літерал – рядок символів, взятий в подвійні лапки. Такий літерал може займати й кілька рядків, у цьому випадку наприкінці рядка ставиться зворотна коса риска. Спеціальні символи можуть бути представлені своїми escape-послідовностями. Ось приклади строкових літералів:

"" (порожній рядок)

"a"

"\ncc\toptions\tfile.[cc]\n"

"a multi-line \

string literal signals its \

continuation with a backslash"

Фактично строковий літерал являє собою масив символьних констант, де за згодою мов C и C++ останнім елементом завжди є спеціальний символ з кодом 0 (\0).

Літерал 'A' задає єдиний символ A, а строковий літерал "A" – масив із двох елементів: 'A' і \0 (порожнього символу).

Раз існує тип wchar_t, існують і літерали цього типу, позначувані, як і у випадку з окремими символами, префіксом L:

L"a wide string literal"

Строковий літерал типу wchar_t – це масив символів того ж типу, завершений нулем.

Якщо в тесті програми йдуть підряд два або кілька строкових літералів (типу char або wchar_t), компілятор з'єднує їх в один рядок.

Ключові слова

Таблиця 4 - Ключові слова C++

asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	export
extern	false	float	for	friend
goto	goto	inline	int	long
mutable	namespace	new	operator	private
protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_cast
short short	signed	sizeof	static	static_cast
struct	switch	template	this	throw
typedef	true	try	typeid	typename
union	voidunion	using	virtual	void

Щоб текст вашої програми був більш зрозумілим, ми рекомендуємо дотримуватися загальноприйнятих правил про імена об'єктів:

- ім'я змінної звичайно пишеться малими літерами, наприклад index (для порівняння: Index – це ім'я типу, а INDEX – константа, визначена за допомогою директиви препроцесора #define);
- ідентифікатор повинен нести будь-який зміст, пояснюючи призначення об'єкта в програмі, наприклад: birth_date або salary;

якщо таке ім'я складається з декількох слів, як, наприклад, birth_date, то прийнято або розділяти слова символом підкреслення (birth_date), або писати кожне наступне слово з великої букви (birthDate).

Тип даних bool

Тип bool використовується винятково для зберігання результатів логічних виразів. У логічного виразу може бути один із двох результатів true або false. True - якщо логічний вираз правда, false- якщо логічний вираз неправда.

Константі true еквівалентні всі числа від 1 до 255 включно, тоді як константі false еквівалентно тільки одне ціле число — 0.

Тип даних char

Тип даних char - це цілий тип даних, який використовується для представлення символів. Тобто, кожному символу відповідає певне число з діапазону [0;255]. Тип даних char також ще називають символьним типом даних, тому що графічне представлення символів у C++ можливо завдяки char. Для представлення символів в C++ типу даних char приділяється один байт, в одному байті — 8 біт, тоді піднесемо двійку до степеня 8 і одержимо значення 256 — кількість символів, яку можна закодувати. Таким чином, використовуючи тип даних char можна відобразити будь-який з 256 символів. Усі закодовані символи представлені в таблиці ASCII.

ASCII (від англ. American Standard Code for Information Interchange) — американський стандартний код для обміну інформацією.

Цілочисельні типи даних

Приставки цілочисельних типів даних:

short - приставка укорочує тип даних, до якого застосовується, шляхом зменшення розміру займаної пам'яті;

long - приставка подовжує тип даних, до якого застосовується, шляхом збільшення розміру займаної пам'яті;

unsigned (без знака) - приставка збільшує діапазон позитивних значень у два рази, при цьому діапазон негативних значень у такому типі даних зберігатися не може.

Так, що, по суті, ми маємо один цілочисельний тип для представлення цілих чисел — це тип даних int. Завдяки приставкам short, long, unsigned з'являється деяка різноманітність типів даних int, що різняться розміром займаної пам'яті й (або) діапазоном прийнятих значень.

Типи даних із плаваючою крапкою

У C++ існують два типи даних із плаваючою крапкою: float і double. Типи даних із плаваючою крапкою призначені для зберігання чисел із плаваючою крапкою. Типи даних float і double можуть зберігати як позитивні, так і негативні числа із плаваючою крапкою. У типі даних float розмір займаної пам'яті у два рази менше, чим у типі даних double, а значить і діапазон прийнятих значень теж менше. Якщо тип даних float оголосити із приставкою long, то діапазон прийнятих значень стане дорівнює діапазону прийнятих значень типу даних double. В основному, типи даних із плаваючою крапкою потрібні для розв'язку завдань із високою точністю обчислень, наприклад, операції із грошима.

Таблиця 5 — Типи даних C++

Тип	байт	Діапазон прийнятих значень
цілочисельний (логічний) тип даних		
bool	1	0/ 255
цілочисельний (символьний) тип даних		
char	1	0/ 255
цілочисельні типи даних		
short int	2	-32 768 / 32 767
unsigned short int	2	0/65 535
int	4	-2 147 483 648 / 2 147 483 647
unsigned int	4	0/4 294 967 295
long int	4	-2 147 483 648 / 2 147 483 647
unsigned long int	4	0/4 294 967 295
типи даних із плаваючою крапкою		
float	4	-2 147 483 648.0/2 147 483 647.0
long float	8	-9 223 372 036 854 775 808 .0/9 223 372 036 854 775 807.0
double	8	-9 223 372 036 854 775 808 .0/9 223 372 036 854 775 807.0

Покажчики

Покажчик – це об'єкт, що містить адресу іншого об'єкта, що й дозволяє побічно маніпулювати цим об'єктом. Звичайно покажчики використовуються для роботи з динамічно створеними об'єктами, для побудови зв'язаних структур даних, таких, як зв'язані списки й ієрархічні дерева, і для передачі у функції більших об'єктів – масивів і об'єктів класів – у якості параметрів.

Кожний покажчик асоціюється з деяким типом даних, причому їх внутрішнє представлення не залежить від внутрішнього типу: і розмір пам'яті, яку займає об'єкт типу покажчик, і діапазон значень у них однаковий . Різниця полягає в тому, як компілятор сприймає адресований об'єкт. Покажчики на різні типи можуть мати те саме значення, але область пам'яті, де розміщаються відповідні типи, можуть бути різними:

- покажчик на int, що містить значення адреси 1000, спрямований на область пам'яті 1000-1003 (в 32-бітній системі);
- покажчик на double, що містить значення адреси 1000, спрямований на область пам'яті 1000-1007 (в 32-бітній системі).

Ось кілька прикладів:

```
int          *ip1, *ip2;
complex<double>  *cp;
```

```
string          *pstring;
vector<int>     *pvec;
double          *dp;
```

Покажчик позначається зірочкою перед іменем. У визначенні змінних списком зірочка повинна стояти перед кожним покажчиком (див. вище: ip1 і ip2). У прикладі нижче lp – покажчик на об'єкт типу long, а lp2 – об'єкт типу long:

```
long *lp, lp2;
```

У наступному випадку fp інтерпретується як об'єкт типу float, а fp2 – покажчик на нього:

```
float fp, *fp2;
```

Оператор розіменування (*) може відділятися пробілами від імені й навіть безпосередньо примикати до ключового слова типу. Тому наведені визначення синтаксично правильні й цілком еквівалентні:

```
string *ps;
string* ps;
```

Однак рекомендується використовувати перший варіант написання: другий здатний ввести в оману, якщо додати до нього визначення ще однієї змінної через кому:

```
//увага: ps2 не покажчик на рядок!
string* ps, ps2;
```

Можна припустити, що й ps, і ps2 є покажчиками, хоча покажчик – тільки перший з них.

Якщо значення покажчика рівно 0, виходить, він не містить ніякої адреси об'єкта.

Нехай задана змінна типу int:

```
int ival = 1024;
```

Нижче приводяться приклади визначення й використання покажчиків на int pi і pi2:

```
//pi ініціалізований нульовою адресою
int *pi = 0;

// pi2 ініціалізований адресою ival
int *pi2 = &ival;
```

```
// правильно: pi і pi2 містять адресу ival
pi = pi2;

// pi2 містить нульову адресу допустимо, ми прагнемо
pi2 = 0;
```

Покажчику не може бути привласнена величина, що не є адресою:

```
// помилка: pi не може приймати значення int
pi = ival
```

Точно так само не можна привласнити покажчику одного типу значення, що є адресою об'єкта іншого типу. Якщо визначені наступні змінні:

```
double dval;
double *ps = &dval;
```

то обидва вирази присвоювання, наведені нижче, спричинять помилку компіляції:

```
// помилки компіляції
// неприпустиме присвоювання типів даних: int* <== double*
pi = pd
pi = &dval;
```

Справа не в тому, що змінна pi не може містити адреси об'єкта dval – адреси об'єктів різних типів мають ту саму довжину. Такі операції змішання адрес заборонені свідомо, тому що інтерпретація об'єктів компілятором залежить від типу покажчика на них.

Звичайно, бувають випадки, коли нас цікавить саме значення адреси, а не об'єкт, на який він указує (допустимо, ми прагнемо зрівняти цю адресу з якоюсь іншою). Для дозволу таких ситуацій уведений спеціальний покажчик void, який може вказувати на будь-який тип даних, і наступні вирази будуть вірними:

```
// правильно: void* може містити
// адреси будь-якого типу
void *pv = pi;
pv = pd;
```

Тип об'єкта, на який вказує void*, невідомий, і ми не можемо маніпулювати цим об'єктом. Усе, що ми можемо зробити з таким покажчиком, – привласнити його значення іншому покажчику або зрівняти з якою-небудь адресною величиною.

Для того щоб звернутися до об'єкта, маючи його адресу, потрібно застосувати операцію розіменування, або непряму адресацію, позначувану зірочкою (*). Маючи наступні визначення змінних:

```
int ival = 1024;, ival2 = 2048;
int *pi = &ival;
```

ми можемо читати й зберігати значення ival, застосовуючи операцію розіменування до покажчика pi:

```
// непряме присвоювання змінної ival значення ival2
*pi = ival2;

// непряме використання змінної ival як rvalue і lvalue
 *pi = abs(*pi); // ival = abs(ival);
*pi = *pi + 1; // ival = ival + 1;
```

Коли ми застосовуємо операцію узяття адреси (&) до об'єкта типу int, то одержуємо результат типу int*

```
int *pi = &ival;
```

Якщо ту ж операцію застосувати до об'єкта типу int* (покажчик на int), ми одержимо покажчик на покажчик на int, тобто int**. int** – це адреса об'єкта, яка містить адресу об'єкта типу int. Проводячи розіменування ppi, ми одержуємо об'єкт типу int* , що містить адресу ival. Щоб одержати сам об'єкт ival, операцію розіменування до ppi необхідно застосувати двічі.

```
int **ppi = &pi;
int *pi2 = *ppi;
cout << "Значення ival\n"
    << "явне значення: " << ival << "\n"
    << "непряма адресація: " << *pi << "\n"
    << "двічі непряма адресація: " << **ppi << "\n"
    << endl;
```

Покажчики можуть бути використані в арифметичних виразах. Зверніть увагу на наступний приклад, де два вирази роблять зовсім різні дії:

```
int i, j, k;
int *pi = &i;
// i = i + 2
```

```
*pi = *pi + 2;
// збільшення адреси, що міститься в pi, на 2
pi = pi + 2;
```

До покажчика можна додавати ціле значення, можна також віднімати з нього. Додавання до покажчика 1 збільшує значення, що міститься в ньому, на розмір області пам'яті, що приділяється об'єкту відповідного типу. Якщо тип char займає 1 байт, int – 4 і double – 8, то додавання 2 до покажчиків на char, int і double збільшить їхнє значення відповідно на 2, 8 і 16. Як це можна інтерпретувати? Якщо об'єкти одного типу розташовано в пам'яті один за одним, то збільшення покажчика на 1 приведе до того, що він буде вказувати на наступний об'єкт. Тому арифметичні дії з покажчиками найчастіше застосовуються при обробці масивів; у будь-яких інших випадках вони чи навряд виправдані.

Клас string

Для того щоб використовувати об'єкти класу string, необхідно підключити відповідний заголовний файл:

```
#include <string>
```

Ось приклад рядка, представлений об'єктом типу string і ініціалізований рядком символів:

```
#include <string>
string st( "Ціна пляшки вина\n" );
```

Довжину рядка повертає функція-член size() (довжина не включає завершальний нульовий символ).

```
cout << "Довжина "
    << st
    << ": " << st.size()
    << " символів, включаючи символ нового рядка\n";
```

Друга форма визначення рядка задає порожній рядок:

```
string st2; // порожній рядок
```

Як ми дізнаємося, чи порожній рядок? Звичайно, можна зрівняти його довжину з 0:

```
if ( ! st.size() )
// правильно: порожній
```

Однак є й спеціальний метод empty(), що повертає true для порожнього рядка й false для непорожного:

```
if ( st.empty() )
// правильно: порожній
```

Третя форма конструктора ініціалізує об'єкт типу string іншим об'єктом того ж типу:

```
string st3( st );
```

Рядок st3 ініціалізований рядком st. Як ми можемо переконатися, що ці рядки збігаються? Скористаємося оператором порівняння (==):

```
if ( st == st3 )
// ініціалізація спрацювала
```

Як скопіювати один рядок в інший? За допомогою звичайної операції присвоювання:

```
st2 = st3; // копіюємо st3 в st2
```

Для конкатенації рядків використовується операція додавання (+) або операція додавання із присвоюванням (+=). Нехай дано два рядки:

```
string s1( "hello, " );
string s2( "world\n" );
```

Ми можемо одержати третій рядок, що полягає з конкатенації перших двох, у такий спосіб:

```
string s3 = s1 + s2;
```

Якщо ж ми прагнемо додати s2 у кінець s1, ми повинні написати:

```
s1 += s2;
```

Операція додавання може провести конкатенацію об'єктів класу string не тільки між собою, але й з рядками вбудованого типу. Можна переписати приклад, наведений вище, так, щоб спеціальні символи й розділові знаки представлялися вбудованим типом, а значимі слова – об'єктами класу string:

```
const char *pc = ", ";
string s1( "hello" );
string s2( "world" );
```

```
string s3 = s1 + pc + s2 + "\n";
```

Подібні вирази працюють тому, що компілятор знає, як автоматично перетворювати об’єкти вбудованого типа в об’єкти класа string. Можливе й просте привласнення вбудованої строки об’єкту string:

```
string s1;
const char *pc = "a character array";
s1 = pc; // правильно
```

Зворотне перетворення, однак, не працює. Спроба виконати наступну ініціалізацію рядка вбудованого типу викличе помилку компіляції:

```
char *str = s1; // помилка компіляції
```

Щоб здійснити таке перетворення, необхідно явно викликати функцію-член із трохи дивною назвою c_str():

```
char *str = s1.c_str(); // майже правильно
```

Функція c_str() повертає покажчик на символьний масив, що містить рядок об'єкта string у тому виді, у якому вона перебувала б у вбудованому строковому типі.

Наведений вище приклад ініціалізації покажчика char *str усе ще не зовсім корректний c_str() повертає покажчик на константний масив, щоб запобігти можливості безпосередньої модифікації вмісту об'єкта через цей покажчик, що має тип

```
const char *
```

Правильний варіант ініціалізації виглядає так:

```
const char *str = s1.c_str(); // правильно
```

До окремих символів об'єкта типу string, як і вбудованого типу, можна звертатися за допомогою операції узяття індексу. Ось, наприклад, фрагмент коду, що замінює всі крапки символами підкреслення:

```
string str( "fa.disney.com" );
int size = str.size();

for ( int ix = 0; ix < size; ++ix )
  if ( str[ ix ] == '.' )
```

```
str[ ix ] = '_';
```

Попередній приклад реалізується також викликом однієї- єдиної функції replace():

```
replace( str.begin(), str.end(), '.', '_' );
```

Ця функція пробігає діапазон від begin() до end(), які повертають покажчики на початок і кінець рядка, і заміняє елементи, рівні третьому своєму параметру, на четвертий.

Тип «масив»

Масив – це набір елементів одного типу, доступ до яких проводиться по індексу – порядковому номеру елемента в масиві. Наприклад:

```
int ival;
```

визначає ival як змінну типу int, а інструкція

```
int ia[ 10 ];
```

задає масив з десяти об'єктів типу int. До кожного із цих об'єктів, або *елементів масиву*, можна звернутися за допомогою операції узяття індексу:

```
ival = ia[ 2 ];
```

привласнює змінній ival значення елемента масиву ia з індексом 2. Аналогічно

```
ia[ 7 ] = ival;
```

привласнює елементу з індексом 7 значення ival.

Визначення масиву складається зі специфікатора типу, імені масиву й розміру. Розмір задає кількість елементів масиву (не менше 1) і береться у квадратні дужки. Розмір масиву потрібно знати вже на етапі компіляції, а отже, він має бути константним виразом, хоча не обов'язково задається літералом. Ось приклади правильних і неправильних визначень масивів:

```
extern int get_size();
// buf_size i max_files константи
const int buf_size = 512, max_files = 20;
int staff_size = 27;
// правильно: константа
char input_buffer[ buf_size ];
// правильно: константний вираз: 20 - 3
```

```
char *filetable[ max_files-3 ];
// помилка: не константа
double salaries[ staff_size ];
// помилка: не константний вираз
int test_scores[ get_size() ];
```

Об'єкти buf_size і max_files є константами, тому визначення масивів input_buffer і filetable правильні. А ось staff_size – змінна (хоча й ініціалізована константою 27), виходить, salaries[staff_size] неприпустимо. (Компілятор не в змозі знайти значення змінної staff_size у момент визначення масиву salaries.)

Вираз max_files-3 може бути обчислений на етапі компіляції, отже, визначення масиву filetable[max_files-3] синтаксично правильно.

Нумерація елементів починається з 0, тому для масиву з 10 елементів правильним діапазоном індексів є не 1 – 10, а 0 – 9. Ось приклад перебору всіх елементів масиву:

```
int main()
{
  const int array_size = 10;
  int ia[ array_size ];
  for ( int ix = 0; ix < array_size; ++ ix )
  ia[ ix ] = ix;
}
```

При визначенні масив можна явно ініціалізувати, перерахувавши значення його елементів у фігурних дужках, через кому:

```
const int array_size = 3;
int ia[ array_size ] = { 0, 1, 2 };
```

Якщо ми явно вказуємо список значень, то можемо не вказувати розмір масиву: компілятор сам підрахує кількість елементів:

```
// масив розміру 3
int ia[] = { 0, 1, 2 };
```

Коли явно зазначені й розмір, і список значень, можливі три варіанти. При збігу розміру й кількості значень усе очевидно. Якщо список значень коротше, чим заданий розмір елементи масиву, що залишилися, ініціалізуються нулями.

Якщо ж у списку більше значень, компілятор виводить повідомлення про помилку:

```
// ia ==> { 0, 1, 2, 0, 0 }
const int array_size = 5;
int ia[ array_size ] = { 0, 1, 2 };
```

Символьний масив може бути ініціалізований не тільки списком символьних значень у фігурних дужках, але й строковим літералом. Однак між цими способами є деяка різниця. Допустимо,

```
const char cal[] = {'C', '+', '+' };
const char cal2[] = "C++";
```

Розмірність масиву ca1 рівна 3, масиву ca2 – 4 (у строкових літералах ураховується завершальний нульовий символ). Наступне визначення викличе помилку компіляції:

```
// помилка: рядок "Daniel" складається з 7 елементів
const char ch3[ 6 ] = "Daniel";
```

Масиву не може бути привласнене значення іншого масиву, неприпустима й ініціалізація одного масиву іншим. Крім того, не дозволяється використовувати масив посилань. Ось приклади правильного й неправильного вживання масивів:

```
const int array_size = 3;
int ix, jx, kx;
// правильно: масив покажчиків типу int*
int *iar [] = { &ix, &jx, &kx };
// error: масиви посилань неприпустимі
int &iar[] = { ix, jx, kx };
int main()
{
  int ia3{ array_size ]; // правильно
  // помилка: вбудовані масиви не можна копіювати
  ia3 = ia;
  return 0;
}
```

Щоб скопіювати один масив в іншій, доведеться проробити це для кожного елемента окремо:

```
const int array_size = 7;
int ia1[] = { 0, 1, 2, 3, 4, 5, 6 };
int main()
{
  int ia3[ array_size ];
  for ( int ix = 0; ix < array_size; ++ix )
  ia2[ ix ] = ia1[ ix ];
  return 0;
```

}

У якості індексу масиву може виступати будь-який вираз, що дає результат цілого типу. Наприклад:

```
int someval, get_index();
ia2[ get_index() ] = someval;
```

Підкреслимо, що мова С++ не забезпечує контролю індексів масиву – ні на етапі компіляції, ні на етапі виконання. Програміст сам повинен стежити за тим, щоб індекс не вийшов за межі масиву. Помилки при роботі з індексом досить поширені. На жаль, не так важко зустріти приклади програм, які компілюються й навіть працюють, але проте містять фатальні помилки, що рано або пізно приводять до краху.

Багатомірні масиви

У С++ є можливість використовувати багатом Об'єкт іа визначений ірні масиви, при оголошенні яких необхідно вказати праву границю кожного виміру в окремих квадратних дужках. Ось визначення двовимірного масиву:

```
int ia[ 4 ][ 3 ];
```

Перша величина (4) задає кількість рядків, друга (3) – кількість стовпців. Об'єкт іа визначений як масив із чотирьох рядків по три елементи в кожному. Багатомірні масиви теж можуть бути ініціалізованими:

```
int ia[ 4 ][ 3 ] = {
   { 0, 1, 2 },
   { 3, 4, 5 },
   { 6, 7, 8 },
   { 9, 10, 11 }
};
```

Внутрішні фігурні дужки, що розбивають список значень на рядки, необов'язкові й використовуються, як правило, для зручності читання коду. Наведена нижче ініціалізація в точності відповідає попередньому прикладу, хоча менш зрозуміла:

```
int ia[4][3] = { 0,1,2,3,4,5,6,7,8,9,10,11 };
```

Наступне визначення ініціалізує тільки перші елементи кожного рядка. Елементи, що залишилися, будуть дорівнювати нулю:

```
int ia[ 4 ][ 3 ] = { {0}, {3}, {6}, {9} };
```

Якщо ж вилучити внутрішні фігурні дужки, результат виявиться зовсім іншим. Усі три елементи першого рядка й перший елемент другої одержать зазначене значення, а інші будуть неявно ініціалізовані 0.

```
int ia[ 4 ][ 3 ] = { 0, 3, 6, 9 };
```

При звертанні до елементів багатомірного масиву необхідно використовувати індекси для кожного вимірювання (вони беруться у квадратні дужки). Так виглядає ініціалізація двовимірного масиву за допомогою вкладених циклів:

```
int main()
{
  const int rowsize = 4;
  const int colsize = 3;
  int ia[ rowsize ][ colsize ];
  for ( int = 0; i < rowsize; ++i )
  for ( int j = 0; j < colsize; ++j )
  ia[ i ][ j ] = i + j j;
}
```

Конструкція

```
ia[ 1, 2 ]
```

є припустимою з погляду синтаксису C++, однак означає зовсім не те, чого чекає недосвідчений програміст. Це аж ніяк не оголошення двовимірного масиву 1 на 2. Агрегат у квадратних дужках – це список виразів через кому, результатом якого буде останнє значення 2. Тому оголошення ia[1,2] еквівалентно ia[2]. Це ще одна можливість припуститися помилки.

Клас vector

Використання класу vector є альтернативою застосуванню вбудованих масивів. Цей клас надає набагато більше можливостей, тому його використання прийнятніше. Однак зустрічаються ситуації, коли не обійтися без масивів вбудованого типу. Клас vector, як і клас string, є частиною стандартної бібліотеки C++.

Для використання вектора необхідно включити заголовний файл:

```
#include <vector>
```

Існують два абсолютно різні підходи до використання вектора, назвемо їх ідіомою масиву й ідіомою STL. У першому випадку об'єкт класу vector використовується точно так само, як масив вбудованого типу. Визначається вектор заданої розмірності:

```
vector< int > ivec( 10 );
```

що аналогічно визначенню масиву вбудованого типу:

```
int ia[ 10 ];
```

Для доступу до окремих елементів вектора застосовується операція узяття індексу:

```
void simp1e_examp1e()
{
  const int e1em_size = 10;
  vector< int > ivec( e1em_size );
  int ia[ e1em_size ];
  for ( int ix = 0; ix < e1em_size; ++ix )
  ia[ ix ] = ivec[ ix ];
  //
}
```

Ми можемо отримати розмірність вектора, використовуючи функцію size(), і перевірити, чи порожній вектор, за допомогою функції empty(). Наприклад:

```
void print_vector( vector<int> ivec )
{
  if ( ivec.empty() )
    return;
  for ( int ix=0; ix< ivec.size(); ++ix )
  cout << ivec[ ix ] << ' ';
}
```

Елементи вектора ініціалізуються значеннями за замовчуванням. Для числових типів і покажчиків таким значенням є 0. Якщо в якості елементів виступають об'єкти класу, то ініціатор для них задається конструктором за замовчуванням. Однак ініціатор можна задати і явно, використовуючи форму:

```
vector< int > ivec( 10, -1 );
```

Усі десять елементів вектора будуть рівні -1.

Масив вбудованого типу можна явно ініціалізувати списком:

```
int ia[ 6 ] = { -2, -1, ПРО, 1, 2, 1024 };
```

Для об'єкта класу vector аналогічна дія неможлива. Однак такий об'єкт може бути ініціалізований за допомогою масиву вбудованого типу:

```
// 6 елементів іа копіюються в ivec
```

```
vector< int > ivec( ia, ia+6 );
```

Конструкторові вектора ivec передаються два покажчики – покажчик на початок масиву ia і на елемент, що іде за останнім. У якості списку початкових значень припустимо вказувати не весь масив, а деякий його діапазон:

```
// копіюються 3 елемента: ia[2], ia[3], ia[4]
vector< int > ivec( &ia[ 2 ], &ia[ 5 ] );
```

Ще однією відмінністю вектора від масиву вбудованого типу є можливість ініціалізації одного об'єкта типу vector іншим і використання операції присвоювання для копіювання об'єктів. Наприклад:

```
vector< string > svec;
void init_and_assign()
{
  // один вектор ініціалізується іншим
  vector< string > user_names( svec );
  // один вектор копіюється в інший
  svec = user_names;
}
```

Говорячи про ідіому STL, ми маємо на увазі зовсім інший підхід до використання вектора. Замість того щоб відразу задати потрібний розмір, ми обумовлюємо порожній вектор:

```
vector< string > text;
```

Потім додаємо до нього елементи за допомогою різних функцій. Наприклад, функція push_back() вставляє елемент у кінець вектора. Ось фрагмент коду, що зчитує послідовність рядків зі стандартного введення і додає їх у вектор:

```
string word;
  while ( cin >> word ) {
    text.push_back( word );
    // ...
  }
```

Клас complex

Клас комплексних чисел complex – ще один клас зі стандартної бібліотеки. Зазвичай, для його використання потрібно включити заголовний файл:

```
#include <complex>
```

Комплексне число складається із двох частин – дійсної й уявної. Уявна частина являє собою квадратний корінь із негативного числа. Комплексне число прийнято записувати у вигляді

2 + 3i

де 2 – дійсна частина, а 3i – уявна. Ось приклади визначень об'єктів типу complex:

```
// чисто уявне число: 0 + 7-i
comp1ex< double > purei( 0, 7 );
// уявна частина рівна 0:3 + Oi
comp1ex< float > rea1_num( 3 );
// і дійсна, і уявна частина рівні 0:0 + 0-i
comp1ex< long double > zero;
// ініціалізація одного комплексного числа іншим
comp1ex< double > purei2( purei );
```

Оскільки complex, як і vector, є шаблоном, ми можемо конкретизувати його типами float, double і long double, як у наведених прикладах. Можна також визначити масив елементів типу complex:

```
complex< double > conjugate[ 2 ] = {
  complex< double >( 2, 3 ),
  complex< double >( 2, -3 )
};
```

Ось як визначаються покажчик і посилання на комплексне число:

```
complex< double > *ptr = &conjugate[0];
complex< double > &ref = *ptr;
```

Комплексні числа можна складати, віднімати, множити, ділити, порівнювати, одержувати значення дійсної й уявної частини.

Клас pair

Клас pair (пари) стандартної бібліотеки C++ дозволяє нам визначити одним об'єктом пари значень, якщо між ними є будь-який семантичний зв'язок. Ці значення можуть бути однакового або різного типу. Для використання даного класу необхідно включити заголовний файл:

```
#include <utility>
```

Наприклад, інструкція

```
pair< string, string > author( "James", "Joyce" );
```

3.1.5 Оператори C++

Оператори керують процесом виконання програми. Набір операторів мови C++ містить усі керуючі конструкції структурного програмування.

Складений оператор обмежується фігурними дужками. Усі інші оператори закінчуються крапкою з комою.

Порожній оператор – ;

Порожній оператор – це оператор, що складається тільки із крапки з комою. Він може з'явитися в будь-якім місці програми, де по синтаксису потрібен оператор. Виконання порожнього оператора не змінює стану програми.

Складений оператор – {...}

Дія складеного оператора полягає в послідовнім виконанні операторів, що містяться в ньому, за винятком тих випадків, коли будь-який оператор явно передає керування в інше місце програми.

Оператор обробки виключень

try { <*оператори*> } **catch** (<*оголошення виключення*>) { <*оператори*> } **catch** (<*оголошення виключення*>) { <*оператори*> } **...** **catch** (<*оголошення виключення*>) { <*оператори*> }

Умовний оператор

if (<*вираз*>) <*оператор 1*> [**else** <*оператор 2*>]

умова міститься в круглих дужках. Вона може бути виразом, як у цьому прикладі:

```
if(a+b>c) { ... }
```

або інструкцією оголошення з ініціалізацією:

```
if ( int ival = compute_value() ){...}
```

Спробуємо для ілюстрації застосування інструкції if реалізувати функцію min(), що повертає найменший елемент вектора. Водночас наша функція буде підраховувати число елементів, рівних мінімуму. Для кожного елемента вектора ми повинні проробити наступне:

1. Зрівняти елемент із поточним значенням мінімуму.
2. Якщо елемент менше, присвоїти поточному мінімуму значення елемента й скинути лічильник в 1.
3. Якщо елемент дорівнює поточному мінімуму, збільшити лічильник на 1.
4. А якщо ні, то нічого не робити.

5. Після перевірки останнього елемента повернути значення мінімуму й лічильника.

Необхідно використовувати дві інструкції if:

```
if ( minval > ivec[ i ] )...// нове значення minval
if ( minval == ivec[ i ] )...// однакові значення
```

Досить часто програміст забуває використовувати фігурні дужки, якщо потрібно виконати кілька інструкцій залежно від умови:

```
if ( minval > ivec[ i ] )
minval = ivec[ i ];
occurs = 1; // не належить  if!
```

Таку помилку важко побачити, оскільки відступи в записі мають на увазі, що й minval=ivec[i], і occurs=1 входять в одну інструкцію if. Насправді ж інструкція

```
occurs = 1;
```

не є частиною if і виконується безумовно, завжди скидаючи occurs в 1. Ось як повинна бути складена правильна if-інструкція (точне положення відкриваючої фігурної дужки є приводом для нескінченних суперечок):

```
if ( minval > ivec[ i ] )
{
  minval = ivec[ i ];
  occurs = 1;
}
```

Друга інструкція if виглядає так:

```
if ( minval == ivec [ i ] )
  ++occurs;
```

Зауважимо, що порядок проходження інструкцій у цьому прикладі вкрай важливий. Якщо ми будемо порівнювати minval саме в такій послідовності, наша функція завжди буде помилятися на 1:

```
if ( minval > ivec[ i ] ) {
  minval = ivec[ i ];
  occurs = 1;
}
// якщо minval тільки що одержала нове значення,
// то occurs буде на одиницю більше, ніж потрібно
if ( minval == ivec[ i ] )
  ++occurs;
```

Виконання другого порівняння не обов'язково: той самий елемент не може одночасно бути й менше й рівний minval. Тому з'являється необхідність вибору одного із двох блоків залежно від умови, що реалізується інструкцією if-else, другою формою if-інструкції. Її синтаксис виглядає в такий спосіб:

```
if ( умова )
  інструкція1
else
  інструкція2
```

інструкція1 виконується, якщо умова дійсна, інакше переходимо до інструкція2. Наприклад:

```
if ( minval == ivec[ i ] )
  ++occurs;
else
  if ( minval > ivec[ i ] ) {
    minval = ivec[ i ];
    occurs = 1;
  }
```

Тут інструкція2 сама є if-інструкцією. Якщо minval менше ivec[i], ніяких дій не проводиться.

У наступному прикладі виконується одна із трьох інструкцій:

```
if ( minval < ivec[ i ] )
{} // порожня інструкція
else
if ( minval > ivec[ i ] ) {
  minval = ivec[ i ];
  occurs = 1;
}
else // minval == ivec[ i ]
++occurs;
```

Складені інструкції if-else можуть служити джерелом неоднозначного тлумачення, якщо частин else більше, ніж частин if. До якого з if віднести дану частину else? (Цю проблему іноді називають проблемою висячого else). Наприклад:

```
if ( minval <= ivec[ i ] )
  if ( minval == ivec[ i ] )
    ++occurs;
else {
  minval = ivec[ i ];
  occurs = 1;
}
```

Судячи з відступів, програміст припускає, що else ставиться до найпершого, зовнішньому if. Однак у C++ неоднозначність висячих else дозволяється співвіднесенням їх з останнім, що зустрілися if. Таким чином, у дійсності попередній фрагмент означає наступне:

```
if ( minval <= ivec[ i ] ) {
  if ( minval == ivec[ i ] )
    ++occurs;
  else {
    minval = ivec[ i ];
    occurs = 1;
  }
}
```

Одним зі способів вирішення даної проблеми є взяття внутрішнього if у фігурні дужки:

```
if ( minval <= ivec[ i ] ) {
  if ( minval == ivec[ i ] )
    ++occurs;
}
else {
  minval = ivec[ i ];
  occurs = 1;
}
```

У деяких стилях програмування рекомендується завжди вживати фігурні дужки при використанні інструкцій if-else, щоб не допустити можливості неправильної інтерпретації коду.

Ось перший варіант функції min(). Другий аргумент функції буде повертати кількість входжень мінімального значення у вектор. Для перебору елементів масиву використовується цикл for. Але ми припустилися помилки в логіці програми. Чи вдасться Вам її помітити?

```
#include <vector>
int min( const vector<int> &ivec, int &occurs )
{
  int minval = 0;
  occurs = 0;
  int size = ivec.size();
  for ( int ix = 0; ix < size; ++ix ) {
    if ( minval == ivec[ ix ] )
      ++occurs;
    else
    if ( minval > ivec[ ix ] ) {
      minval = ivec[ ix ];
      occurs = 1;
    }
  }
```

```
  return minval;
}
```

Звичайно функція повертає тільки одне значення. Однак згідно з нашою специфікацією в точці виклику повинне бути відомо не тільки саме мінімальне значення, але й кількість його входжень у вектор. Для повернення другого значення ми використовували параметр типу посилання. Будь-яке присвоювання значення посиланню occurs змінює значення змінної, на яку вона посилається:

```
int main()
{
  int occur_cnt = 0;
  vector< int > ivec;
 // occur_cnt одержує значення occurs
  // з функції min()
  int minval = min( ivec, occur_cnt );
  // ...
 }
```

Альтернативою використанню параметра-посилання є застосування об'єкта класу pair. Функція min() могла б повертати два значення в одній парі:

```
// альтернативна реалізація
// за допомогою пари

#include <uti1ity>
#include <vector>
typedef pair<int,int> min_va1_pair;

min_va1_pair
min( const vector<int> &ivec )
{
  int minval = 0;
  int occurs = 0;
  // те ж саме ...
  return make_pair( minval, occurs );
}
```

На жаль, і ця реалізація містить помилку. Де ж вона? Правильно: ми ініціалізували minval нулем, тому, якщо мінімальний елемент вектора більше нуля, наша реалізація поверне нульове значення мінімуму й нульове значення кількості входжень.

Програму можна змінити, ініціалізувавши minval першим елементом вектора:

```
int minval = ivec[0];
```

Тепер функція працює правильно. Однак у ній виконуються деякі зайві дії, що знижують її ефективність.

```
// виправлена версія min()
// залишаються можливості для оптимізації ...
int minval = ivec[0];
occurs = 0;
int size = ivec.size();
for ( int ix = 0; ix < size; ++ix )
{
  if ( minval == ivec[ ix ] )
  ++occurs;
  // ...
```

Оскільки іх ініціалізується нулем, на першій ітерації циклу значення першого елемента порівнюється із самим собою. Можна ініціалізувати іх одиницею й уникнути непотрібного виконання першої ітерації. Однак при оптимізації коду ми допустили іншу помилку (напевно, потрібно було все залишити як було!). Чи вдасться Вам її помітити?

```
// оптимізована версія min(),
// на жаль, що містить помилку...
int minval = ivec[0];
occurs = 0;
int size = ivec.size();
for ( int ix = 1; ix < size; ++ix )
{
 if ( minval == ivec[ ix ] )
 ++occurs;
  // ...
```

Якщо ivec[0] виявиться мінімальним елементом, змінна occurs не одержить значення 1. Звичайно, виправити це дуже просто, але спочатку треба знайти помилку:

```
int minval = ivec[0];
occurs = 1;
```

На жаль, подібного роду недогляди зустрічаються не так рідко: програмісти теж люди й можуть помилятися. Важливо розуміти, що це неминуче, і бути готовим ретельно тестувати й аналізувати свої програми.

Ось остаточна версія функції min() і програма main(), що перевіряє її роботу:

```
#include <iostream>
#include <vector>
int min( const vector< int > &ivec, int &occurs )
```

```
{
  int minval = ivec[ 0 ];
  occurs = 1;
  int size = ivec.size();
  for ( int ix = 1; ix < size; ++ix )
  {
    if ( minval == ivec[ ix ] )
      ++occurs;
    else
      if ( minval > ivec[ ix ] ){
        minval = ivec[ ix ];
        occurs = 1;
      }
  }
  return minval;
}

int main()
{
  int ia[] = { 9,1,7,1,4,8,1,3,7,2,6,1,5,1 };
  vector<int> ivec( ia, ia+14 );
  int occurs = 0;
  int minval = min( ivec, occurs );
  cout << "Мінімальне значення: " << minval
       << " зустрічається: " << occurs << " раз.\n";
  return 0;
}
```

Результат роботи програми:

Мінімальне значення: 1 зустрічається: 5 разів.

Довгі ланцюжки інструкцій if-else важкі для сприйняття й, таким чином, є потенційним джерелом помилок.

```
if ( ch == 'a' ||
  ch == 'A' )
    ++acnt;
else
if ( ch == 'e' ||
  ch == 'E' )
    ++ecnt;
else
  if ( ch == 'i' ||
  ch == 'I' )
    ++icnt;
```

```
else
  if ( ch == 'o' ||
    ch == '0' )
      ++ocnt;
else
  if ( ch == 'u' ||
    ch == 'U' )
      ++ucnt;
```

У якості альтернативи таким ланцюжкам C++ присвячує інструкцію switch.

Оператор-Перемикач

switch (<*вираз*>) { **case** <*константний вираз 1*>: <*оператори 1*> **case** <*константний вираз 2*>: <*оператори 2*> **... case** <*константний вираз N*>: <*оператори N*> [**default**: <*оператори*>] }

Оператор-Перемикач призначено для вибору одного з декількох альтернативних шляхів виконання програми. Обчислення оператора-перемикача починається з обчислення *вираз*, після чого керування передається *операторові*, позначеному *константним виразом*, рівним обчисленому значенню *виразу*. Вихід з оператора-перемикача здійснюється оператором break. Якщо значення *виразу* не рівно жодному *константному виразу*, то керування передається *операторові*, позначеному ключовим словом *default*, якщо він є.

Оператор циклу з післяумовою

do <*оператор*> **while** <*вираз*>;

У мові C++ цей оператор відрізняється від класичної реалізації циклу з післяумовою тим, що при істинності *виразу* відбувається продовження роботи циклу, а не вихід із циклу.

Представимо, що нам треба написати програму, що переводить милі в кілометри. Структура програми виглядає так:

```
int val;
bool more = true; // фіктивне значення, потрібне для
// початку циклу

while ( more ) {
   val = getvalue();
  val = convertvalue(val);
  printvalue(val);
  more = domore();
 }
```

Проблема полягає в тому, що умова обчислюється в тілі циклу. for і while вимагають, щоб значення умови рівнялося true до першого входження в цикл, інакше тіло не виконається жодного разу. Це означає, що ми повинні забезпечити таку умову до початку роботи циклу. Альтернативою може стати використання do while тіла, що гарантує виконання, циклу хоча б один раз.

Ось як виглядає попередній приклад з використанням циклу do while:

```
do {
  val = getvalue();
  val = convertvalue(val);
  printvalue(val);
  } while domore();
```

На відміну від інших інструкцій циклів, do while не дозволяє оголошувати об'єкти у своїй частині умови. Ми не можемо написати:

```
// помилка: оголошення змінної
// в умові не дозволяється
do {
  // ...
  mumble( foo );
} while ( int foo = get_foo() ) // помилка
```

тому що до умовної частини інструкції do while ми дійдемо тільки після першого виконання тіла циклу.

Оператор покрокового циклу

for ([<*початковий вираз*>]; [<*умовний вираз*>]; [<*вираз збільшення*>]) <*оператор*>

Цикл for звичайно використовується для обробки структур даних, що мають фіксовану довжину, таких, як масив або вектор:

```
#include <vector>
int main() {
int ia[ 10 ];
for ( int ix = 0; ix < 10; ++-ix )
  ia[ ix ] = ix;
vector<int> ivec( ia, ia+10 );
vector<int>::iterator iter = ivec.begin() ;

for ( ; iter != ivec.end(); ++iter )
  *iter *= 2;
return 0;
}
```

Інструкція-Ініціалізації може бути або виразом, або інструкцією оголошення. Звичайно вона використовується для ініціалізації змінної значенням, яке збільшується в ході виконання циклу. Якщо така ініціалізація не потрібна або виконується десь в іншім місці, цю інструкцію можна замінити порожньою (див. другий з наведених нижче прикладів). Ось приклади правильного використання інструкції-ініціалізації:

```
// index і iter визначені в іншім місці
for ( index =0; ...
for ( ; /* порожня інструкція */ ...
for ( iter = ivec.begin(); ...
for ( int 1o = 0,hi = max; ...
for ( char *ptr = getstr(); ...
```

Умова використовується для керування циклом. Поки умова при обчисленні дає true, інструкція продовжує виконуватися. Виконувана в циклі інструкція може бути як простою, так і складеною. Якщо ж найперше обчислення умови дає false, інструкція не виконується жодного разу. Правильні умови можна записати так:

```
(... index < arraysize; ... )
(... iter != ivec.end(); ... )
(... *stl++ = *st2++; ... )
(... char ch = getnextchar(); ... )
```

Вираз обчислюється після виконання інструкції на кожній ітерації циклу. Звичайно його використовують для модифікації змінної, ініціалізованої в інструкції-ініціалізації. Якщо найперше обчислення умови дає false, вираз не виконується жодного разу. Правильні вирази виглядають у такий спосіб:

```
( ... ...; ++-index )
( ... ...; ptr = ptr->next )
( ... ...; ++i, --j, ++cnt )
( ... ...; ) // порожній вираз
```

Для наведеного нижче циклу for

```
const int sz = 24;
int ia[ sz ];
vector<int> ivec( sz );

for ( int ix = 0; ix < sz; ++ix ) {
  ivec[ ix ] = ix;
  ia[ ix ]= ix;
 }
```

порядок обчислень буде наступним:

1. Інструкція-Ініціалізації виконується один раз перед початком циклу. У даному прикладі оголошується змінна ix, яка ініціалізується значенням 0.

2. Обчислюється умова. Якщо вона рівна true, виконується складена інструкція тіла циклу. У нашім прикладі, поки ix менше sz, значення ix присвоюється елементам ivec[ix] і ia[ix]. Коли значенням умови стане false, виконання циклу припиниться. Якщо найперше обчислення умови дасть false, складена інструкція виконуватися не буде.

3. Обчислюється вираз. Як правило, його використовують для модифікації змінної, що фігурує в інструкції-ініціалізації, що й перевіряється в умові. У нашім прикладі ix збільшується на 1.

Ці три кроки являють собою повну ітерацію циклу for. Тепер кроки 2 і 3 будуть повторюватися доти, поки умова не стане рівною false, тобто ix виявиться рівним або більшим sz.

В інструкції-ініціалізації можна визначити кілька об'єктів, однак усі вони повинні бути одного типу, тому що інструкція оголошення допускається тільки одна:

```
for ( int ival = 0, *pi = &ia, &ri = val;
  ival < size;
  ++iva1, ++pi, ++ri )
  // ...
```

Оголошення об'єкта в умові набагато важче правильно використовувати: таке оголошення повинне хоча б раз дати значення false, інакше виконання циклу ніколи не припиниться. Ось приклад, хоча й трохи надуманий:

```
#include <iostream>
int main()
{
  for ( int ix = 0;
    bool done = ix == 10;
    ++ix )
      cout << "ix: " << ix << endl;
}
```

Видимість усіх об'єктів, визначених усередині круглих дужок інструкції for, обмежена тілом циклу. Наприклад, перевірка iter після циклу спричинить помилку компіляції :

```
int main()
{
  string word;
  vector< string > text;
  // ...
  for ( vector< string >::iterator
```

```
        iter = text.begin(),
        iter_end = text.end();
        iter != text.end(); ++iter )
      {
        if ( *iter == word )
        break;
        // ...
    }

    // помилка: iter і iter_end невидимі
      if ( iter != iter_end )
      // ...
```

Оператор розриву

break;

Оператор розриву перериває виконання операторів while, do, for і switch. Він може міститися тільки в тілі цих операторів. Керування передається оператору програми, що іде за перерваним. Якщо оператор розриву записаний усередині вкладених операторів while, do, for, switch, то він завершує оператор, що тільки безпосередньо охоплює його.

Оператор продовження

continue;

Оператор продовження передає керування на наступну ітерацію в операторах циклу while, do, for. Він може міститися тільки в тілі цих операторів. В операторах do і while наступна ітерація починається з обчислення умовного виразу. В операторі for наступна ітерація починається з обчислення виразу приросту, а потім відбувається обчислення умовного виразу.

Наприклад,

```
#include <iostream>
main()
{
static int a[]={1,2,-3,4,-5,6};
int i,n,s;
n=6; s=0;
for(i=0; i<n; i++)
{
if(a[i]<=0)
continue; /*пропуск 0*/
      s+=a[i];
   }
cout<<"сумма = "<<s;
```

}

В даному прикладі оператор *continue* використовується для пропуску від'ємних елементів масиву, додаючи лише додатні елементи.

Оператор повернення

return [<*вираз*>];

Оператор повернення закінчує виконання функції, у якій він міститься, і повертає керування в функцію, що її викликала. Керування передається в точку визивної функції, що безпосередньо іде за оператором виклику. Значення *виразу*, якщо воно задане, обчислюється, приводиться до типу, оголошеного для функції, що містить оператор повернення, і вертається в визивну функцію. Якщо *вираз* опущений, то значення, що вертається функцією, не визначене.

Приклад:

```
int sum (int a, int b)
{ renurn (a+b); }
```

Функція sum має два формальні параметри a і b типу int. Оператор renurn повертає результат роботи функції sum, а саме добуток елементів a і b.

З формальної точки зору оператори break, continue і return не є операторами структурного програмування. Однак їх використання в обмежених кількостях виправдане, коли вони спрощують розуміння програми й дозволяють уникати великих вкладених структур.

Арифметичні оператори

Таблиця 6 – Арифметичні оператори

Оператор		**Синтаксис**
Присвоювання		a = b
Додавання		a + b
Віднімання		a – b
Унарний плюс		+a
Унарний мінус		-a
Множення		a * b
Ділення		a / b
Операція модуль (залишок від цілочисельного ділення)		a % b
Інкремент	префіксний	++a
	суффіксний	a++

Оператор		Синтаксис
Декремент	префіксний	--a
	суффіксний	a--

Оператори порівняння

Таблиця 7 - Оператори порівняння

Оператор	Синтаксис
Рівність	a == b
Нерівність	a != b
Більше	a > b
Менше	a < b
Більше або рівно	a >= b
Менше або рівно	a <= b

Логічні оператори

Таблиця 8 - Логічні оператори

Оператор	Синтаксис
Логічне заперечення NOT	!a
Логічне І	a && b
Логічне АБО	a || b

Побітові оператори

Таблиця 9 - Побітові оператори

Оператор	Синтаксис
Побітова інверсія	~a
Побітове І	a & b
Побітове АБО	a | b
Побітове виключає АБО (XOR)	a b
Побітовий лівий зсув	a << b
Побітовий правий зсув	a >> b

Складене присвоювання

Таблиця 10 - Складене присвоювання

Оператор	Синтаксис	Значення
Присвоювання з підсумовуванням	a += b	a = a + b
Присвоювання з відніманням	a -= b	a = a - b
Присвоювання із множенням	a *= b	a = a * b

Присвоювання з діленням	a /= b	a = a / b
Присвоювання по модулю	a %= b	a = a % b
Присвоювання з побітовим I	a &= b	a = a & b
Присвоювання з побітовим АБО	a |= b	a = a | b
Присвоювання з побітовим виключаючим АБО (XOR)	a = b	a = a b
Присвоювання з побітовим зсувом уліво	a <<= b	a = a << b
Присвоювання з побітовим зсувом вправо	a >>= b	a = a >> b

Оператори роботи з покажчиками й членами класу

Таблиця 11 - Оператори роботи з покажчиками й членами класу

Оператор	**Синтаксис**
Звертання до елемента масиву	a[b]
Непряме звернення («об'єкт на який вказує *a*»)	*a
Посилання («адреса *a*»)	&a
Звертання до члена структури («член *b* об'єкт, на який вказує *a*»)	a->b
Звертання до члена структури («член *b* об'єкта *a*»)	a.b
Член, на який вказує *b*, в об'єкті на який вказує *a*	a->*b
Член, на який вказує *b*, в об'єкті *a*	a.*b

Таблиця пріоритетів визначає пріоритет у послідовності виразів, коли порядок обчислень не був явно заданий дужками.

3.1.6 Пріоритет операцій

Таблиця 12 - Пріоритет операцій

Пріоритет	**Оператор**	**Опис**	**Асоціативність**
1 Найвищий	::	Зміна області видимості	Немає
2	++	Суффіксний інкремент	Зліва направо
	--	Суффіксний декремент	
	()	Виклик функції	
	[]	Узяття елемента масиву	
3	++	Префіксний інкремент	Справа наліво
	--	Префіксний декремент	
	+	Унарний плюс	

Пріоритет	Оператор	Опис	Асоціативність
	-	Унарний мінус	
	!	Логічне НІ	
	~	Побітове НІ	
	*	Розіменування покажчика	
	&	Узяття адреси	
4	*	Множення	
	/	Ділення	
	%	Узяття модуля (залишок від ділення)	
5	+	Додавання	
	-	Віднімання	
6	<<	Побітовий зсув вліво	
	>>	Побітовий зсув вправо	
7	<	Менше	
	<=	Менше або рівно	
	>	Більше	
	>=	Більше або рівно	
8	==	Рівність	
	!=	Нерівність	
9	&	Побітове І	
10	^	Побітове виключаюче АБО (XOR)	
11	|	Побітове АБО	
12	&&	Логічне І	
13	||	Логічне АБО	
14	?:	Умовний оператор	Справа наліво
15	=	Пряме присвоювання	
	+=	Присвоювання з додаванням	
	-=	Присвоювання з відніманням	
	*=	Присвоювання із множенням	
	/=	Присвоювання з діленням	
	% =	Присвоювання з узяттям залишку від ділення	
	<<=	Присвоювання з побітовим зсувом уліво	
	>>=	Присвоювання з побітовим зсувом вправо	
	&=	Присвоювання з побітовим І	
	=	Присвоювання з побітовим XOR	
	|=	Присвоювання з побітовим OR	
16	,	Оператор Comma	Зліва направо

3.1.7 Математичні функції C++

У C++ визначені в заголовному файлі <cmath> функції виконуючі деякі часто використовувані математичні завдання. Наприклад, знаходження кореня, піднесення в ступінь,

sin(), cos() і багато інших. У таблиці 13 показані основні математичні функцій, прототипи яких містяться в заголовному файлі <cmath>.

Таблиця 13 — Математичні функції в С++

Функція	Опис	Приклад
abs(a)	модуль або абсолютне значення від **a**	abs(-3.0)= 3.0 abs(5.0)= 5.0
sqrt(a)	корінь квадратний з **a,** причому **a** не негативно	sqrt(9.0)=3.0
pow(a, b)	піднесення **a** в ступінь **b**	pow(2,3)=8
ceil(a)	округлення **a** до найменшого цілого, але не менше ніж **a**	ceil(2.3)=3.0 ceil(-2.3)=-2.0
floor(a)	округлення **a** до найбільшого цілого, але не більше чим **a**	floor(12.4)=12 floor(-2.9)=-3
fmod(a, b)	обчислення залишку від a/b	fmod(4.4, 7.5) = 4.4 fmod(7.5, 4.4) = 3.1
exp(a)	обчислення експоненти $\mathbf{e^a}$	exp(0)=1
sin(a)	**a** задається в радіанах	
cos(a)	**a** задається в радіанах	
log(a)	натуральний логарифм **a**(основою є експонента)	log(1.0)=0.0
log10(a)	десятковий логарифм **a**	Log10(10)=1
asin(a)	арксинус **a**, де **-1.0 < a < 1.0**	asin(1)=1.5708

Приклад використання арифметичних операцій та математичних функцій.

```
#include<iostream>
int main()
{
int a, b;
double z;
cin>>a>>b;//вводимо значення змінних
if(a==b) z=sqrt(a);//якщо а дорівнює b, тоді z=корінь квадратний від а
if(a>b) z=abs(b); //якщо а більше b, тоді z=модуль b
if(a<=b) z=fmod(a,b);//якщо а менше або дорівнює b, тоді  z= fmod(a,b)
if(a!=b && b>10) z=exp(b);// якщо а недорівнює b І b більше 10, тоді z=…
z*=a;//z=z*a
cout<<z;//виведення результату на екран
system(“pause”);
return 0;
}
```

В даному прикладі використовуються логічні операції: рівність, більше, менше оба рівно, нерівно, логічне І. Складене присвоювання, а саме присвоювання з множенням. Математичні функції: квадратного кореня, модуля, залишку від цілочисельного ділення, експоненти.

Примітка: розгорнуті приклади Ви знайдете в розділі задач.

3.2 VISUAL BASIC

3.2.1 Загальні відомості

Visual Basic - це остання версія однієї з популярних мов програмування. В даний час за допомогою Visual Basic можна швидко створювати додатки, що працюють в середовищі Windows для будь-якої області комп'ютерних технологій: бізнес-додатки, мультимедіа, додатки типу клієнт - сервер і додатки управління базами даних. Крім того, Visual Basic є вбудованою мовою для додатків Microsoft Office. Багато розробників додатків також використовують Visual Basic в якості внутрішньої мови своїх додатків.

Visual Basic являє собою інтегроване середовище розробки, яке містить набір інструментів, що полегшують і прискорюють процес розробки додатків. Причому процес розробки полягає не в написанні програми (програмного коду), а в проектуванні програми. Додаток формується засобами графічного редагування (компонування), що дозволяє звести процес створення програмного коду до мінімуму.

Як і у всіх сучасних системах візуального проектування, в Visual Basic застосовується об'єктно-орієнтований підхід до програмування. Будь-який додаток, написаний на Visual Basic, являє собою сукупність об'єктів.

Об'єкт - деяка сутність, яка чітко проявляє свою поведінку і є представником деякого класу подібних собі об'єктів. Майже все, з чим проводиться робота в VB, є об'єктами. Наприклад: Форма, Командна кнопка, Текстове поле і т. д.

Кожен об'єкт характеризується:

• властивостями;

• методами;

• подіями.

Властивість - це атрибут об'єкта, що має ім'я. Властивості визначають характеристики об'єкта (колір, положення на екрані, стан об'єкта).

Методи - це дії або завдання, які виконує об'єкт (те, що можна робити з об'єктами).

Класом об'єктів в об'єктно-орієнтованих мовах програмування називається загальний опис таких об'єктів, для яких характерна наявність безлічі загальних властивостей і загальних дій, які здатні виконувати ці об'єкти (наприклад, клас Командна кнопка - загальний опис кнопок у вікнах додатків). Вони повинні мати безліч загальних властивостей та інших характеристик (наприклад подій, однакових для всіх цих об'єктів: клацання мишею).

Додаток, що створюється у середовищі Visual Basic, називається проектом. Програмний проект - це сукупність частин, що складають майбутній Windows-додаток. Будь-який проект повинен обов'язково складатися з екранних форм (хоча б однієї) і програмних модулів (хоча б одного). Visual Basic зберігає кожен проект в окремому файлі з розширенням vbp.

Екранна форма - це графічне представлення Windows-додатка разом з вмістом цього вікна. Зміст включає в себе:

• сукупність властивостей цього вікна з їх значеннями;

• сукупність, об'єктів, що знаходяться в цьому вікні;

• сукупність властивостей цих об'єктів з їх значеннями.

У Visual Basic екранна форма зберігається в окремому файлі з розширенням frm.

Програмний модуль - це програмний код (текст деякої програми), що зберігається в окремому файлі. Він може використовуватися при вирішенні найчастіше однієї, а іноді й кількох задач. Ім'я цього файлу має розширення bas.

Програмний код проекту існує не сам по собі, він прив'язаний до окремих об'єктів екранної форми. Частина коду, яка відноситься тільки до одного об'єкту, в свою чергу може складатися з декількох фрагментів-процедур.

У Visual Basic (VB) програмний код майже завжди прив'язується до якоїсь події, яка є сигналом до початку роботи програми. Наприклад, натискання мишею по будь-якому об'єкту екранної форми; завантаження нової екранної форми; переміщення покажчика миші уздовж смуги прокрутки; натискання будь-якої клавіші на клавіатурі.

Спочатку проектується екранна форма, потім встановлюються події, які відбуватимуться в працюючому додатку, і тільки потім програмуються дії, пов'язані з цими подіями.

Подія - це характеристика класу об'єкта, що описує зовнішній вплив, на який реагує об'єкт цього класу під час роботи програми.

Більшість процедур, з яких складається програмний код VB, прив'язані до подій і називаються процедурами-подіями.

Створення будь-якого додатку складається з наступних етапів:

1) постановка завдання. Опис принципу роботи майбутнього додатку, видів екранних форм (вікон) цього додатка;

2) розробка інтерфейсу. Створення екранних форм програми з усіма об'єктами, що знаходяться на цих формах, і властивостями цих об'єктів;

3) програмування. Визначення того, які події будуть відбуватися в процесі роботи програми, складання алгоритмів процедур для цих подій і написання програми (програмних кодів) цих процедур;

4) налагодження програми. Усунення логічних помилок в процедурах і досягнення того, щоб додаток працював задовільно в середовищі проектування;

5) збереження проекту і при бажанні - компіляція (перетворення проекту в виконуємий додаток, здатний працювати самостійно за межами середовища проектування).

Додаток може працювати в режимі компіляції або інтерпретації. У режимі інтерпретації всі інструкції мовою Basic, з яких складається програма, будуть виконуватися системою Visual Basic безпосередньо в процесі їх читання комп'ютером рядок за рядком у середовищі розробки.

У режимі компіляції спочатку здійснюється налагодження програми за допомогою інтерпретатора, потім вона повністю транслюється (перекладається) з мови високого рівня (Basic) на мову низького рівня (мова машинних команд комп'ютера), тобто компілюється.

Скомпільована програма поміщається у файл з розширенням exe. Цей файл може бути запущений на виконання самостійно, без підтримки середовища Visual Basic.

3.2.2 Екранні елементи середовища Visual Basic

Запуск середовища програмування Visual Basic

Для того, щоб запустити середовище програмування, необхідно виконати послідовність: **Пуск → Програми → Microsoft Visual Basic 6.0. → Microsoft Visual Basic 6.0.**

З'являється вікно з трьома вкладками: **New** (Нове), **Existing** (Існуюче), **Recent** (Останнє).

Рисунок 3.10 - Вікно додавання об'єкта

За допомогою вікна можна створити новий проект або відкрити існуючий.

1. Створення нового проекту: **New → Standard EXE → Відрити**.

Створюється проект майбутньої програми, який може містити форми, модулі та інші компоненти.

2. Відкрити існуючий проект: **Existing → Відкрити потрібну папку → Вибрати потрібний проект → Відкрити.**

3. Відкрити об'єкт, який раніше відкривали на даному комп'ютері: **Recent → Вибрати потрібний проект → Відкрити.**

Вікно середовища програмування Visual Basic

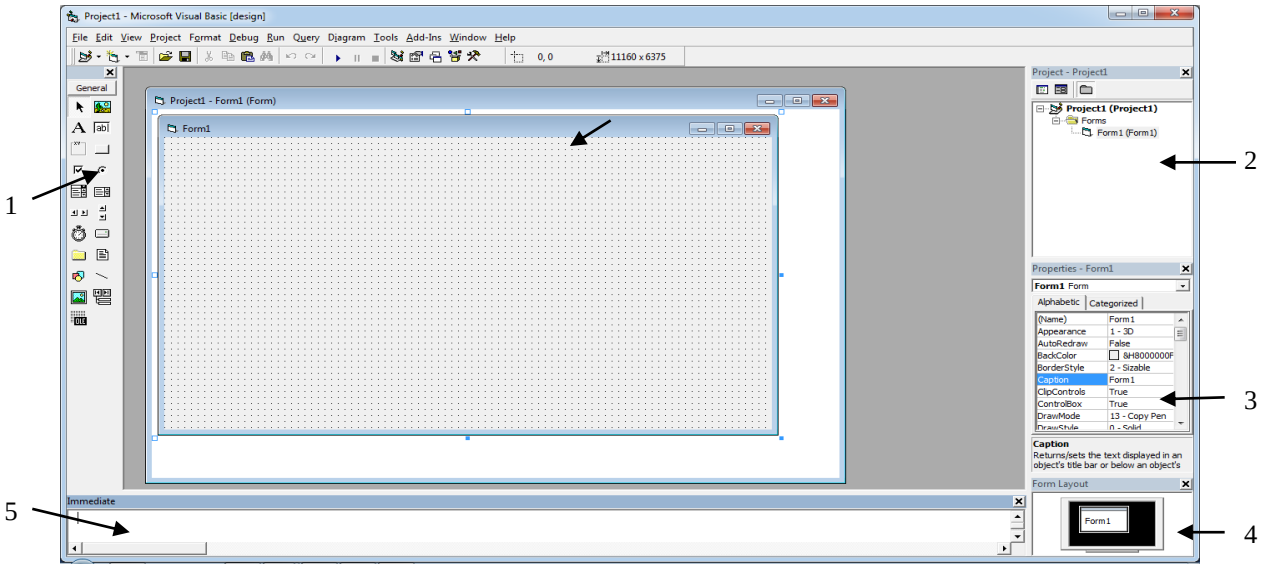


Рисунок 3.11 - Вікно середовища програмування Visual Basic

1) панель елементів управління;
2) вікно провідника проекту;
3) вікно властивостей поточного елемента керування або форми;
4) вікно розміщення форми на екрані монітора;
5) вікно для введення команд. Команди виконуються відразу після введення;
6) вікна, що містять форми, модулі та інші елементи проекту.

Якщо які-небудь з перерахованих елементів не видно, то їх можна вивести на екран (або приховати) за допомогою меню **View** (Вид):

1) **View → ToolBox** (Панель інструментів);
2) **View → Project Explorer** (Провідник проекту);
3) **View → Properties Window** (Вікно властивостей);
4) **View → Form Layout Window** (Вікно розміщення форми);

5) **View → Immediate Window** (Вікно невідкладного).

Вікно форми можна вивести на екран двічі клацнувши у вікні провідника проекту по значку або імені форми.

Збереження проекту

Проект додатка зберігається в окремому файлі і також в окремих файлах зберігаються елементи проекту.

При першому збереженні вказуються імена файлів для всіх елементів проекту. Оскільки проект складається з декількох файлів, то для нього краще створити окрему папку.

Порядок збереження проекту, що містить одну форму:

File → Save Form As ... → Створення нової папки → Ввести имя папки → Enter → двічі натиснути по папці (папка відкриється) → ввести ім'я файлу форми → **Зберегти** → ввести ім'я файлу проекту → **Зберегти.**

При повторному збереженні досить натиснути на кнопку (зберегти проект).

Робота з елементами середовища програмування

Елементи середовища програмування - це невеликі вікна, які виводять різну інформацію і дозволяють управляти складовими частинами проекту.

Вікна можна вивести на екран або приховати за допомогою меню **View** (див. вище). Всі вікна середовища програмування можна перетягнути за заголовок до будь-якого краю екрану.

Розглянемо кожне з них.

Project Explorer (Провідник проекту) - відображає групи об'єктів (Форми, Модулі). У групах знаходяться безпосередньо самі об'єкти: **форми , модулі .**

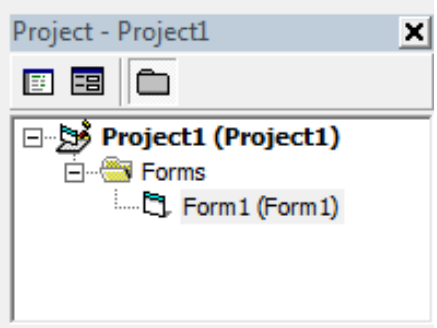


Рисунок 3.12 - Провідник проекту

У вікні після значка об'єкта вказується ім'я файлу, в якому він збережений.

Додати новий об'єкт можна за допомогою меню **Project.**

Для того, щоб вибрати об'єкт і відкрити його вікно, необхідно у вікні провідника проекту двічі натиснути на назву потрібного об'єкту або натиснути кнопку (Показати об'єкт) у вікні провідника проекту.

Щоб відкрити вікно коду об'єкта, достатньо клацнути на назву об'єкту правою кнопкою і в контекстному меню вибрати рядок **View Code** або натиснути кнопку (Показати код) у вікні провідника проекту.

Properties Window (Вікно властивостей) - відображає властивості поточного об'єкта (форми або елементів управління: *кнопок, списків, перемикачів*). Щоб зробити елемент поточним, необхідно натиснути по ньому.

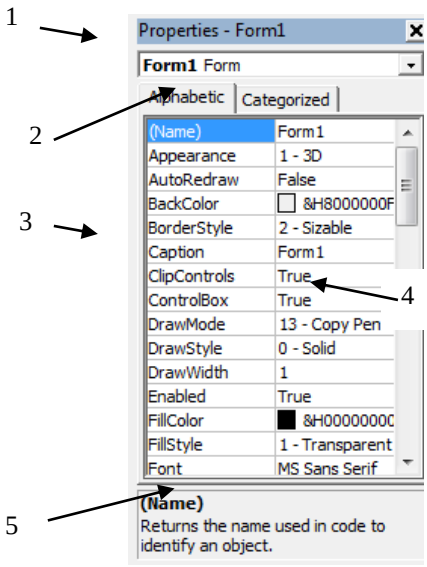


Рисунок 3.13 - Вікно властивостей

1 - ім'я об'єкта, властивості якого відображаються;

2 - вкладки: **Alphabetic** (Алфавіт) і **Categorized** (Категорії) змінюють порядок сортування властивостей: за алфавітом, за категоріями;

3 - назва властивості;

4 - значення властивості;

5 - коментар поточної властивості.

Список, що розкривається у верхній частині вікна **Properties**, який відображає імена і типи об'єктів, включених в додаток і розташованих на формі, називається списком об'єктів. Спочатку тут міститься лише інформація про форму. Але в міру того як об'єкти додаються, Visual Basic реєструє їх в цьому списку.

Нижче списку об'єктів слідує поле параметрів. Тут можна визначити конкретне значення для обраної властивості.

 Поняття властивості (**properties**) в Visual Basic пов'язано з механізмом формального опису атрибутів об'єкта. Кожен об'єкт має специфічні властивості, які визначають зовнішній вигляд і поведінку об'єкта в додатку. Деякі властивості мають фіксований набір значень (їх можна дізнатися, натиснувши першу з кнопок в полі параметрів). Якщо ж кнопка справа в цьому полі з трьома крапками, то це означає, що натискання на кнопку ініціює виклик діалогового вікна з вибором необхідного значення. Прикладами можуть служити вибір кольорів, а також вибір файлів для завантаження картинок. Інші властивості можуть мати практично необмежений набір значень. Це, наприклад, завдання імені або назви форми.

Значення властивостей можуть бути логічними, тобто мати значення:

True - тобто **Так, Істина, 1**.

False - тобто **Ні, Неправда, 0.**

Ці два значення змінюються подвійним клацанням по рядку потрібної властивості.

Form Layout Window (Вікно розміщення форми на екрані) - показує, як буде розташована форма на екрані після запуску програми.

У цьому вікні на зображеному екрані монітора можна перетягувати форму мишею.

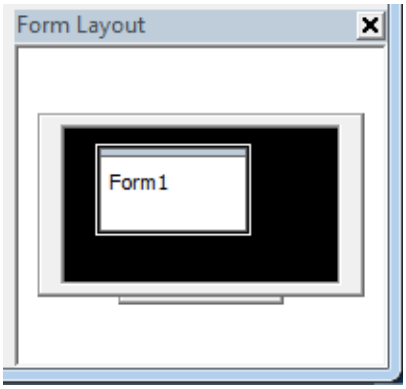


Рисунок 3.14 - Вікно розміщення форми

3.2.3 Форма

Загальні відомості

Форма - це вікно майбутньої програми. При створенні нового проекту Visual Basic відкриває порожню форму і привласнює їй заголовок **Form1**. Форма слугує своєрідним полотном, на якому розміщуються різні частини програми - об'єкти або елементи управління. Форма є об'єктом, тому має свої *властивості, методи, події*. У проекті може міститися кілька форм.

Форма зберігається в окремому файлі (точніше в 2-х файлах з різними розширеннями).

Форми бувають:

1. **Форма (Form)** - звичайна форма, використовувана в нескладних програмах.

2. **Основна форма (MDI Form)** - це форма, яка може містити дочірні (вкладені) форми. У додатку може бути тільки одна така форма.

3. **Дочірня форма (Child)** - міститься тільки всередині основної форми. Таких форм у додатку може бути кілька.

4. **Форма діалогу (Dialog)** - з'являється на екрані на короткий час, служить для введення або виведення інформації, не змінюється в розмірах і знаходиться поверх інших вікон.

Запуск і зупинка створюваного додатка

Запустити програму (запуск) або F5. При цьому з'являється стартова форма, тобто вікно створюваної програми і його значок в панелі завдань.

Призупинити програму Пауза (Ctrl + Pause Break) Використовується для налагодження.

Продовжити виконання програми: або F5.

Зупинити програму Стоп або закрити вікно запущеної програми

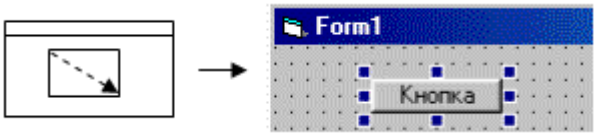
Створення елементів управління на формі

Для організації інтерфейсу між користувачем і програмою служать:

Елементи управління додаються за допомогою *Панелі Інструментів* **View** (Вид) → **ToolBox** (Панель Інструментів):

Рисунок 3.15 - Панель інструментів

Створити елемент управління: виберіть потрібний елемент в панелі інструментів (натиснувши по кнопці елемента) і утримуючи ліву кнопку перемістіть мишу по діагоналі на формі:



Виділити елемент управління: натисніть по потрібному елементу мишею.

Після виділення елементу управління або форми у вікні властивостей можна змінити його властивості.

Видалити елемент управління: виділіть елемент керування та натисніть клавішу Delete.

3.2.4 Елементи управління

Елементи управління - це об'єкти, які служать для організації інтерфейсу між користувачем і комп'ютером. Наприклад: *кнопки, списки, перемикачі.*

Елемент управління - це об'єкт, що має свої *властивості, методи, події.*

 **Pointer** (покажчик). Активізація цього елемента означає, що можна редагувати форму: додавати нові об'єкти, перетягувати їх на інші місця або змінювати розміри об'єктів, використовуючи один із маркерів.

 **Picture Box** (вікно рисунка) і  **image Box** (вікно зображення). Використання цих інструментів дозволяє розміщувати графічну інформацію в певних ділянках форми. Вікна рисунків краще підходять для динамічного середовища (наприклад, анімаційні ефекти). Вікна зображень зручніше використовувати в статиці, тобто коли створену картинку не передбачається змінювати.

 **Label** (мітка). Мітки - це поля, які заповнюються програмістом текстовою інформацією і недоступні користувачеві для редагування. Вміст мітки визначається її властивістю *Caption*. Пряме введення тексту на мітці під час виконання програми не допускаються.

Властивості:

- **Caption** - текст напису;
- **Font** - шрифт, його розмір, накреслення;
- **Alignment** - вирівнювання тексту: *Left* (вліво), *Right* (вправо), *Center* (по центру);
- **ToolTipText** - підказка, що з'являється при наведенні покажчика миші на елемент керування;

- **ForeColor** - колір тексту;
- **BackColor** - колір фону.

 **Text Box** (текстове вікно). Текстові вікна - це екранні області, в які користувач може вводити текст.

Властивості:

Text - містить символи, які ввів користувач. Інші властивості аналогічні елементу Label.

 **Command Button** (командна кнопка), **Check Box** (прапорець), **Option Button** (перемикач). Ці три об'єкти з точки зору програміста практично однакові, але для користувача вони відрізняються за зовнішнім виглядом і за призначенням. Натискання командної кнопки зазвичай активізує якусь операцію, а прапорці та перемикачі, повідомляючи поточний стан того чи іншого параметра, дають можливість змінювати його. Натискання прапорця встановлює або скидає певний параметр: якщо на квадратику прапорця значок **X**, параметр активний (включений). Будь-який прапорець функціонально незалежний від інших прапорців. Перемикачі, навпаки, завжди працюють в групі і дозволяють вибрати будь-який варіант з кількох можливих. Коли один перемикач активний, решта відключені.

 **Frame** (рамка). Служить для розділення у вікні різних груп об'єктів. Стосовно до перемикачів рамки впливають на поведінку кнопок. Для інших об'єктів рамки виступають в ролі візуального роздільника і функції, регулюючої доступ до групи об'єктів.

 **List Box** (список) - перелік варіантів, обираних натисканням миші. У занадто довгий список, який не виводиться на екран цілком, Visual Basic додає лінійки прокрутки. Для коректної роботи з цим об'єктом необхідно, щоб його висота була мінімум три рядки. Вміст списку не можна задати на етапі проектування. Щоб ввести в нього елемент, необхідно використовувати метод *Additem*. Ініціалізацію списку краще проводити в процедурі обробки події **Load** - вона належить формі, що містить цей список.

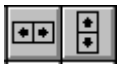
 **Combo Box** (комбінований список). Названий так тому, що, об'єднуючи текстове вікно (тут воно називається полем введення) із звичайним списком, утворює єдиний елемент управління. Однак, на відміну від звичайного, комбінований список не дозволяє розміщувати елементи в кілька колонок.

Властивості:

о **Text** - вміст рядка, введене користувачем або вибране зі списку;

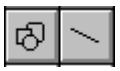
о **List** - рядки списку (багаторядкова властивість);

о **ListIndex** - номер обраного користувачем рядка (нумерація починається з нуля, якщо ніякий рядок не було обрано, то властивість дорівнює -1).

 **Horizontal Scroll Bar** (горизонтальна лінійка прокрутки), **Vertical Scroll Bar** (вертикальна лінійка прокрутки). Діють абсолютно однаково, тільки в різних напрямках. Ці об'єкти дозволяють дізнаватися про позицію движка на лінійці і контролювати діапазон його дії і дискретність переміщення движка.

 **Timer** (таймер). Об'єкт, здатний ініціювати події через регулярні проміжки часу.

 **Drive List Box** (список дисків), **Directory List Box** (список каталогів), **File List Box** (список файлів). Кнопки дозволяють створювати і настроювати діалогові вікна, призначені для взаємодії з файловою системою. Кожен з них управляє окремим компонентом файлової системи. Об'єкт **Drive List Box** - це список, що розкривається, містить імена всіх дисків в системі. Об'єкт **Directory List Box** виводить на екран список всіх підкаталогів в поточному каталозі. Об'єкт **File List Box** відображає всі (або з заданим розширенням) файли в поточному каталозі.

 **Shape** (фігура), **Line** (лінія). Також як і вікна зображень служать, головним чином, для створення фону у вікні форми. Інструмент **Line** дозволяє створювати прості прямі лінії. Модифікуючи властивості ліній, можна змінювати їх розмір, колір і стиль. Інструмент **Shape** за замовчуванням дозволяє за замовчуванням малювати тільки прямокутники, але, змінивши властивість *Shape*, можна викреслювати окружності, еліпси і прямокутники з округленими кутами. Крім того, допускається підбір кольору і зафарбовування замкнутих фігур.

3.2.5 Створення програмного коду

Поняття програмного коду

Для того, щоб програма виконувала приписані їй дії, наприклад, обчислювала, виводила результат, реагувала на дії користувача, наприклад, на натиснення кнопок, вибір рядків зі списку, необхідний програмний код.

Програмний код-це набір слів і символів мови програмування.

Слова і символи повинні бути записані строго за правилами мови, без орфографічних і пунктуаційних помилок. Саме точне написання дозволить комп'ютеру однозначно зрозуміти і виконати програму.

Вікно програмного коду

Програмний код записується у вікні коду. Таке вікно є у кожної форми.

Коли створюється зовнішній вигляд форми, і на неї вносяться інструменти, Visual Basic автоматично готує шаблони процедур для обробки подій.

Відкрити вікно коду:

1 спосіб - в вікні **Project Explorer** (Провідник Проекту) натиснути правою кнопкою по потрібній формі і в меню, вибрати**View Code** (Показати код).

Вікно коду може бути і не пов'язане з формою. Окреме вікно коду називається **Module** (Модуль)  . Модулі у вікні **Project Explorer** згруповані в групу **Modules**. Для відкриття вікна з кодом модуля потрібно у вікні **Project Explorer** двічі натиснути по імені модуля.

3 *спосіб* - двічі натиснути по елементу управління на формі або по самій формі у вікні форми. При цьому не тільки відкривається вікно коду, але і створюється *процедура обробки події* (див. нижче).

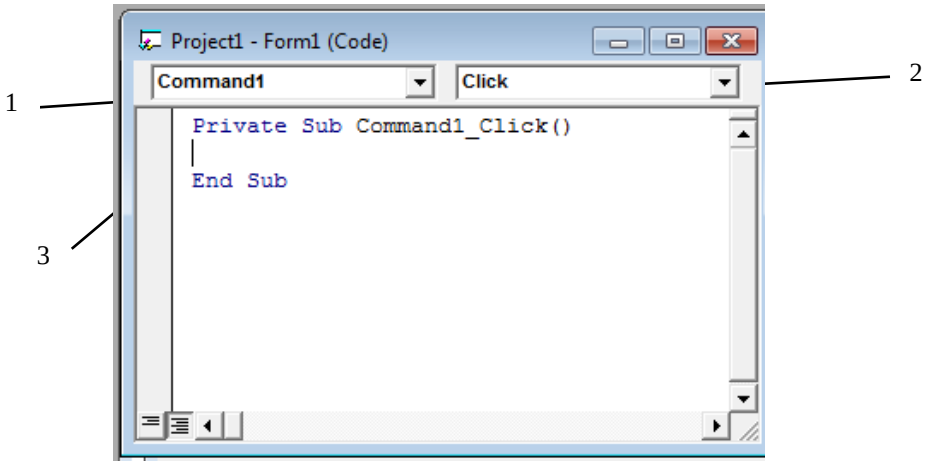


Рисунок 3.16 - Вікно програмного коду

1) список елементів управління;
2) список подій елементів управління;
3) процедура (код).

Процедури

Програмний код складається з окремих процедур, кожна з яких виконує свої певні для неї дії. Процедура - це відокремлений фрагмент програмного коду.

Процедури бувають:

а) **процедури обробки подій**. Виконуються при виникненні якої-небудь події в якому-небудь елементі управління (або формі);

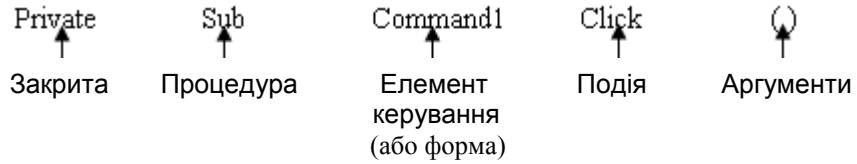
b) **довільні процедури**. Вони не пов'язані з подіями і можуть бути викликані з будь-якої іншої процедури і виконані в будь-який час.

Структура процедури

Процедура складається з наступних елементів:

- заголовок процедури - відзначає початок процедури, її тип, призначення (подія).

Приклад заголовка процедури, яка виконується при натисканні мишею по кнопці з ім'ям **Command1**.



Private - закрита, тобто процедура належить тільки даній формі або модулю і не може бути використана іншими формами, модулями. Якщо це слово опустити, то процедура буде відкрита.

Sub - процедура.

Елемент управління (або ім'я форми): тут вказується точне ім'я елемента, що зберігається у властивості **Name.**

Подія - найменування події. Ось деякі події:

Click - натискання мишею;

DblClick - подвійне натискання мишею;

KeyPress-натискання клавіші;

Load - завантаження форми (при запуску програми або відкритті нової форми);

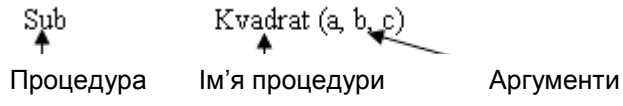
UnLoad - вивантаження форми (при закритті форми, завершенні програми);

Activate - активізація форми (при натисканні по формі, коли її заголовок підсвічується);

Deactivate - деактивізація форми (при натисканні по іншій формі).

Аргументи - це вихідні дані, передані процедурі для обробки.

У довільних процедур заголовок наступний:



Ім'я процедури повинно бути унікально, повинно починатися з літери, не повинно містити пробілів та інших знаків, крім знака підкреслення. На ім'я відбувається виклик процедури, коли необхідно її виконати.

- закінчення процедури - закінчує програмний код процедури ***End Sub***
- тіло процедури - це рядки між заголовком і закінченням. Їх кількість необмежена. Рядки містять розпорядження, які повинні виконатися при виклику процедури (виникненні події).

Створення процедури

Для створення процедури можна скористатися одним з наведених способів:

1 спосіб - двічі натисніть по потрібному елементу управління або формі. Відкриється вікно коду, а в ньому з'явиться заголовок і закінчення процедури. Якщо необхідна інша подія, то її вибирають за допомогою списку у верхньому правому куті вікна коду;

2 *спосіб* - відкрийте вікно коду і введіть потрібні рядки з клавіатури.

Виклик процедур на виконання

Щоб виконалася процедура обробки події, ця подія має статися.

Для виконання довільної процедури в тілі іншої процедури вказують ім'я цієї процедури (викликають її).

Private Sub Command1_Click ()

Kvadrat

End Sub

Тут при натисненні на кнопку **Command** виникає подія **Click** (натискання мишею) і викликається і виконується процедура **Kvadrat.**

Код процедури виконується зверху вниз.

3.2.6 Дані

Класифікація даних

У мові Visual Basic введені десять основних типів даних, які представлені в таблиці.

Таблиця 14

Тип даних	Опис	Діапазон значень	Займана пам'ять
Byte	Двійкові дані	Від 0 до 255	1 байт
Boolean	Логічний	True або False	2 байта
Integer	Цілі числа	Від -32768 до 32767	2 байта
Long	Цілі числа (довгі)	Від -2147483648 до +2 147 483 647	4 байта
String	Символьний	Від 0 до 2000000000	1 байт на 1 символ

Тип даних	Опис	Діапазон значень	Займана пам'ять
		символів	
Currency	Число з фіксованою десятковою крапкою	Від -22337203685477,58 до +922337203685477,58	8 байтів
Date	Дата	Від January 1, 100 до December 31, 9999	8 байтів
Single	Речовинні числа	Від ± 1.4 * 10^{-45} до + 3.4 * 104^{-38}	4 байта
Double	Речовинні числа	Від + 4.94 * 10^{-324} до + 1.79 * 10^{-308}	8 байтів
Variant	Довільний тип	Будь-який з перерахованих вище	Залежить від значення

Константи

Константа - дане, значення якого однозначно визначається написанням і не може бути змінено.

Для зберігання постійних величин Visual Basic дозволяє оголосити константи, тобто виділити ділянки пам'яті, вміст яких не змінюється. Оголошення констант здійснюється оператором

[Public | Private] **Const** *ім'я* [As type] = *вираз*

Ім'я констант прийнято записувати прописними буквами (правила запису імен див. нижче).

Public - константу можна використовувати в будь-яких процедурах і функціях;

Private - константу можна використовувати тільки всередині модуля, в якому вона визначена.

Тип константи можна не оголошувати. За замовчуванням приймається тип, що займає найменший обсяг пам'яті. Тому краще явно вказувати тип спеціальними символами в операторах оголошення констант. Використовувані символи показані в таблиці 15:

Таблиця 15

Символ оголошення типу	**Тип даних**
%	Integer
&	Long
1	Single
#	Double
@	Currency
$	String

Імена використовуються для позначення об'єктів у програмі (константа є об'єктом програми). Правила виникнення імен:

• першим символом імені повинна бути латинська буква;

• ім'я може включати тільки латинські букви, цифри і знак підкреслення (_);

• ім'я може містити не більше 40 символів;

• ключові слова або **Reserved word** (Зарезервовані слова) не можуть використовуватися як імена.

Хорошим тоном при програмуванні на будь-якій мові є осмислений вибір імен для об'єктів програми (привласнювати об'єктам імена, відповідні контексту і несучі описове навантаження).

Оператор оголошення

Оператор оголошення резервує в пам'яті місце для зберігання даних певного типу та організації і привласнює йому ім'я, по якому проводиться звернення до даних.

Оператор має вигляд:

Dim *ім'я* **As** *тип, ім'я* **As** *тип* **...**

ім'я - ім'я об'єкта (ім'я змінної, масиву);

тип - тип даних.

При описі імен прописні і малі літери сприймаються однаково. Однак після визначення посилання на цю змінну повинні відповідати останній формі запису (проводиться автоматичне перетворення тексту програми).

Visual Basic допускає використання імен без оголошення їх типу (в цьому випадку автоматично визначається тип, вимагає для розміщення мінімальної пам'яті), проте доцільно і є ознакою хорошого тону явно оголошувати типи використовуваних даних.

Існує поняття області дії (**scope**) даних, що визначає можливість доступу до них або даним (наприклад, до змінної) в окремих процедурах однієї форми або в процедурах, що відносяться до різних форм однієї програми.

Якщо оператор оголошення якої-небудь змінної знаходиться всередині процедури обробки події, то доступ до цієї змінної (можливість її використання) можливий тільки в рамках даної процедури. Така змінна називається локальною (**local**).

Для того щоб одна і та ж змінна могла використовуватися в різних процедурах однієї форми, оператор оголошення змінних має бути поміщений в розділ загальних оголошень (**general**), доступ до якого відкривається натисканням миші по елементу **general** списку вікна **Object** форми.

Оголошена таким чином змінна має статус діючої на рівні модуля (**modul-level variable**) і може використовуватися (доступна) в будь-якій процедурі даної форми. Для того щоб одна і та ж змінна могла використовуватися в процедурах різних форм однієї програми вона повинна бути

оголошена як глобальна змінна (**global variable**). Використовується ключове слово **Global** замість **Dim**.

Оператори оголошення глобальних змінних поміщаються в модулях коду (**code modules**) і ці змінні можуть використовуватися у всій програмі.

Масиви

Масив - упорядкований набір однотипних даних, позначений одним ім'ям. Масив може будуватися з однотипних змінних, однорідних типів даних (однакових записів), однотипних елементів, тип яких визначає користувач.

Найбільш простий масив - це список елементів даних; такого роду масив називається *простим* або *одномірним*, масивом. Подібний масив можна представити у вигляді черги, де кожному елементу черги присвоюється не тільки порядковий номер, але і його конкретне значення.

Масив оголошується вже розглянутим оператором **Dim**

Dim *ім'я* (розмір) **As** *тип*

Інший варіант завдання масиву: вказати перший і останній номери елементів масиву:

Dim *ім'я* (*початкове значення індексу* **To** *кінцеве значення індексу*) **As** *тип*

Елементи масиву займають пов'язану послідовну область в пам'яті машини. Багатовимірні масиви також займають лінійну послідовну область пам'яті. При цьому важливе значення має спосіб упорядкування елементів багатовимірних масивів, який відрізняється для різних мов програмування.

У Visual Basic багатовимірні масиви упорядковуються в пам'яті машини так, що швидше змінюється лівий індекс. Наприклад, двовимірний масив (у прикладних математичних програмах матриці представляються двовимірним масивом) *A(2,3)*, що складається з 12 елементів, розташовується в пам'яті машини таким чином: *A(0,0), A(1,0), A(2 , 0), A(0,1), A(1,1), A(2,1), A(0,2), A(1,2), A(2,2), A(0, 3), A(1,3), A(2,3)* (якщо даний масив представляє матрицю, то в пам'яті машини вона впорядковується за стовпцями).

Порядок створення двовимірного масиву той же, що і одновимірного, з тією лише різницею, що, вказуючи його розмір, потрібно вказати два значення - кількість рядків і стовпців:

Dim *ім'я* (кількість рядків, кількість стовпців) **As** *тип*

Розмір тривимірного масиву визначатиметься трьома числами і так далі:

Dim *ім'я* (X, Y, Z) **As** *тип*

При створенні масивів, у тому числі і багатовимірних, для зберігання значення кожного елемента виділяється оперативна пам'ять (навіть якщо це нульові значення або пусті рядки). Таким чином, створюючи великий масив, відбувається різке зменшення обсягу вільної пам'яті, що може негативно відобразитися на роботі програми. Тому створювати багатовимірні масиви слід лише в міру необхідності. Подібні масиви називаються статичними (**static**), тому що число елементів у масиві не змінюється.

Вибір розміру масиву може бути ускладнений, якщо невідомо, скільки даних буде введено в масив, або якщо обсяг даних, що збираються для масиву, значно змінюється. Для подібних ситуацій Visual Basic підтримує особливий тип масивів, званий динамічним (**dynamic**) масивом.

Динамічні масиви створюються за допомогою оператора **Dim, Private, Public** або **Static**, причому список розмірностей опускається, потім їх розмір встановлюється за допомогою оператора **ReDim** під час виконання процедури.

Оператор **ReDim** має наступний синтаксис:

ReDim [Preserve] *ім'я* (вимірювання масиву) [**As** *тип*],

де необов'язкове ключове слово **Preserve** призводить до того, що Visual Basic зберігає дані в наявному масиві, коли змінюється розмір масиву з допомогою **ReDim.**

Необхідно використовувати окремий оператор **As** *тип* для кожного масиву, який визначається.

Елементи створеного масиву не містять жодних даних. Щоб зберегти в масиві якесь значення, потрібно вказати, якому елементу воно має бути присвоєно. Використання елементів масивів відбувається за допомогою звернення до них через ім'я масиву і вказівки індексів.

3.2.7 Вирази

Вирази використовуються для операцій над даними. Залежно від даних і використовуваних операцій виразу можна розділити на арифметичні, логічні і символьні. Вираз можна визначити

Операнд [знак операції *операнд*] [знак операції *операнд*] ...,

де в залежності від типу виразу використовуються відповідні операнди і знаки операцій.

Арифметичний вираз

Використовуються наступні знаки операцій:

+ - додавання (2.36 +12.5);

- віднімання (231-49);

* - множення (3 * 2);

^ - піднесення до ступеня (1 ^ 2);

/ - ділення з плаваючою крапкою (3/2, результат 1.5);

\ - цілочисельне ділення (3/2, результат 1);

Mod - обчислення залишку (7 Mod 4, результат 3).

Пріоритет виконання операції (в порядку зменшення пріоритету): зведення в ступінь, множення і ділення з плаваючою точкою, цілочисельне ділення, обчислення залишку, додавання і віднімання.

Обчислення в виразах проводяться зліва направо.

Дужки змінюють пріоритет.

Операндами виразів можуть бути:

• константа (Integer, Long, Currency, Single, Double, Variant);

• змінна (Integer, Long, Currency, Single, Double, Variant);

• елемент масиву (Integer, Long, Currency, Single, Double, Variant);

• звернення до стандартної функції (див. нижче);

• звернення до процедури - функції (див. нижче);

• арифметичний вираз в дужках.

Логічний вираз

Логічні вирази використовуються в математичній логіці і їх також називають булевими виразами, по імені математика Дж. Буля.

Використовуються наступні знаки логічних операцій:

Not - логічне заперечення НЕ;

And - логічне множення І;

Or - логічне додавання АБО;

Xor - виняткове АБО;

Eqv - логічна еквівалентність;

Imp - логічна імплікація.

Логічні операції поєднують логічні величини, які можуть приймати два значення: **True** (Істина) або **False** (Неправда). Результат логічної операції також приймає одне з двох значень: **True** (Істина) або **False** (Неправда).

Пріоритет виконання операцій (у порядку зменшення пріоритету): Not, And, Or, Xor, Eqv, Imp.

Операндами логічного виразу є:

• логічні константи;

• логічні змінні;

• звернення до функцій, що повертає логічні значення;

• вираз відношення;

• укладені в дужки логічні вирази.

Вирази відношення складаються з двох арифметичних або символьних виразів, об'єднаних знаками операцій відношення:

> - більше;

< - менше;

> = - більше або дорівнює;

<= - менше або дорівнює;

= - дорівнює;

<> - не дорівнює.

Вираз приймає значення або **True**, або **False**.

Подвійні нерівності для правильного їх обчислення необхідно записувати з використанням знаків логічних операцій. Коли арифметичні дані перетворюються до логічного типу, то **0** перетвориться в **False**, а інші значення перетворюються в **True**. При перетворенні логічного типу до арифметичного, **False** перетвориться до **0**, а **True** до **-1.**

Символьний вираз

У Visual Basic визначена одна операція з символьними даними - конкатенація (зчеплення), що дозволяє об'єднувати декілька рядків в одну. Знак операції - "+" або "&".

Операндами символьного виразу можуть бути:

• символьна константа;

• символьна змінна;

• елемент символьного масиву (**string**);

• звернення до процедури-функції, що повертає символьне значення;

• звернення до стандартної функції, що повертає символьне значення.

3.2.8 Стандартні функції

У Visual Basic є широкий набір вбудованих (стандартних) функцій, що полегшує написання програм. Є математичні функції, для обробки рядків, для роботи з часом і датами, для фінансових розрахунків.

Вбудовані функції різняться тим, що деякі повертають розраховане значення, інші не повертають. Звернення до функцій, які повертають розраховане значення, є операндом виразу.

Звернення до вбудованої функції, що повертає значення того чи іншого типу, має відповідати виразу, в якому до неї звертаються. Наприклад, в арифметичному виразі можна звертатися до функцій, що повертають значення арифметичних типів, в символьному - символьного типу.

Звернення до функцій, які не повертають розраховане значення, є окремими операторами програми. Наприклад, запис окремого оператора.

Для звернення до деяких вбудованих функцій потрібно задавати значення аргументу (наприклад, **Sin**(X+2), де X+2 вираз, що визначає значення аргументу). Для інших вбудованих функцій аргумент задавати не потрібно (наприклад, **Now**).

Прикладами математичних функцій є:

Atn - повертає арктангенс;

Sin - повертає синус;

Cos - повертає косинус;

Tan - повертає тангенс;

Exp - повертає $e \wedge x$;

Log - повертає натуральний логарифм;

Sqr - повертає квадратний корінь;

Rnd - повертає випадкове число;

Sgn - повертає знак числа:

Fix - повертає округлене число.

Прикладами строкових функцій є:

StrComp - порівнює два рядки;

Lease - перетворює рядок у нижній регістр;

Ucase - перетворює рядок у верхній регістр;

Spase - створює рядок пробілів;

String - створює рядок символів;

Len - визначає довжину рядка;

Instr - шукає підрядок;

Right - виділяє праву частину рядка;

Left - виділяє ліву частину рядка;

Asc - повертає ASCII код символу;

Str - перетворює число в рядок;

Val - перетворює рядок у число.

Прикладами функцій дати і часу є:

Date - встановлює і повертає поточну дату;

Time - встановлює і повертає поточний час;

DateSerial - перетворює в послідовну дату три цілих числа (день, місяць, рік);

Day - перетворює послідовну дату в день місяця;

Month - перетворює послідовну дату в місяць року;

Year - перетворює послідовну дату на рік.

Повні відомості про вбудовані функції і правила їх застосування можна знайти в довідковій системі Visual Basic.

3.2.9 Оператори

Програма на Visual Basic складається з процедур (будь-яка програма складається хоча б з однієї процедури). Процедури складаються з операторів.

Оператором (**Statement**) є синтаксично повний опис конкретної команди (аналог пропозиції російською або іншою мовою), яка виражає одну дію або визначення.

Одному оператору відповідає один рядок програми. Однак можна використовувати розділовий знак двокрапка (**:**), щоб помістити більше ніж один оператор в рядку програми.

Оператори програми виконуються послідовно зверху вниз (зліва направо для операторів на одному рядку), якщо інші оператори (переходу, управління, звернення до функцій або процедурам, див. нижче) не викликають зміни послідовності їх виконання.

Рядки програми можуть, бути позначені мітками (**Linelabel**) або номерами (**Linenumber**).

Мітка (**Label**) позначає наступний рядок програми. Мітка може включати не більше 40 символів (перший обов'язково буква) і закінчується двокрапкою (**:**), не може бути ключовим словом. Мітка може починатися в будь-якій позиції рядка, якщо їй не попередує ніякий символ.

Номер рядка (**Lilienumber**) позначає наступний рядок програми. Номер рядка може включати не більше 40 десяткових цифр і не закінчується двокрапкою. Номер рядка може починатися в будь-якій позиції рядка, якщо йому не попередує ніякий символ. У рамках однієї процедури номери рядків не можуть повторюватися.

Програму легше читати і налагоджувати, якщо оператори програми забезпечені коментарями. Коментарі починаються з апострофа ('), за яким можна розміщувати будь-які зауваження в тексті програми. Якщо коментарі розташовуються на кількох рядках, то кожен рядок потрібно починати з апострофа.

Оператор переходу

Оператор переходу має вигляд:

GoTo {мітка | номер рядка}

і викликає перехід до виконання оператора, з вказаною міткою або номером рядка.

Слід зазначити, що використання оператора переходу в програмах є ознакою низької кваліфікації програміста і його бажано уникати.

Інший оператор переходу дозволяє перейти до виконання виділеної групи операторів (так звана внутрішня процедура). Синтаксис його використання наступний:

GoSub {мітка | номер рядка}

Return - цей оператор викликає перехід до виконання групи операторів, початок якій зазначено міткою або номером рядка. Останній оператор групи є оператор **Return** (**Return** і **GoSub** - ключові слова).

Оператор присвоювання

Оператор привласнення (**assignment statement**) має наступний вигляд:

[**Let**] {змінна | елемент масиву} = вираз

Змінної або елементу масиву в лівій частині оператора присвоюється значення обчисленого виразу в лівій частині.

При використанні оператора привласнення слід дотримуватися таких правил:

• Якщо в лівій частині оператора використовується змінна або елемент масиву символьного типу (**String**), то вираз в правій частині має бути теж символьним;

• Якщо в лівій і правій частинах оператора використовуються арифметичні дані (**Integer, Long, Single, Double, Currency**) але різних типів, то тип правої частини перетвориться до типу лівої частини. Результатом привласнення значення речовинної константи 2.5 змінної цілого типу (I = 2.5) буде 2 (тобто в комірці пам'яті відведеної для змінної I буде зберігатися значення 2).

• Змінній або елементу масиву типу **Variant** в лівій частині може відповідати будь-який тип виразу в правій частині (в комірці пам'яті для зберігання даних типу **Variant** зберігається не тільки значення, але і його тип). Однак такого присвоєння бажано уникати.

Опція **Let** в операторі використовується для привласнення значення одного даного користувальницького типу іншому, за умови, що типи елементів обох даних збігаються.

Умовний оператор

Як правило, алгоритми обробки інформації та програми, що їх реалізують містять перевірки будь-яких умов, від яких залежить наступна дія. Для цього призначений умовний оператор. У Visual Basic є дві форми оператора **If:** *однорядкова і багаторядкова***.**

Однорядкова форма:

If *умова* **Then** *оператори* [**Else** *оператори*]

Багаторядкова форма:

If *умова* **Then**
оператори
оператори
.....
[**ElseIf** *умова* **Then**
оператори
оператори
.....]
.....
[**Else**
оператори
оператори
.....]
End If

Блоків **ElseIf** може бути *скільки завгодно* чи *не бути зовсім*. Блок **Else**, якщо він є, то *один* і *стоїть останнім*.

Керуюча структура Select Case

Структура **Select Case** застосовується, коли одна величина бере участь у кількох логічних порівняннях і визначає, який блок операторів буде виконуватися Алгоритм такого множинного порівняння можна запрограмувати і з використанням логічного структурного оператора, але застосування структури **Select Case** ефективніше.

Найбільш часто структура **Select Case** застосовується в тих випадках, коли порівнювана величина є цілим числом (наприклад, для вибору блоків операторів програми в залежності від обраної альтернативи діалогу).

Синтаксис оператора **Select Case**:

Select Case перевіряємий вираз
[**Case** значення, значення .....
[Оператори
оператори
.....]]
[Case значення, значення .....
[Оператори
оператори
.....]]
.....
[**Case Else**
[Оператори
оператори

.....]]
End Select

Цикли

В алгоритмах обробки інформації та програмах що їх реалізують, широко використовуються цикли - повторювані однакові обчислення.

Для реалізації такого роду програм в Visual Basic є спеціальні засоби - оператори циклу.

Оператори циклу діляться на 2 види: **Do** і **For**. Оператори виду **Do** існують в 5 варіантах:

1. **Do …. Loop**
2. **Do …. Loop While**
3. **Do …. Loop Until**
4. **Do While …. Loop**
5. **Do Until …. Loop**

Оператори виду **For** зустрічаються в 2 варіантах: з додатним і від'ємним кроком.

Оператор циклу Do-Loop

Синтаксис оператора наступний:

Do
оператори
оператори
.....
Loop

Пара **Do** (роби) - **Loop** (петля, повернення до **Do**) визначають початок і кінець оператора циклу, *оператори* всередині неї є тілом циклу.

3.2.8.5.2 Оператор циклу Do-Loop While

Синтаксис оператора:

Do
оператори
оператори
.....
Loop While *умова продовження виконання циклу*

While визначає виконання *операторів,* що входять в цикл, поки логічний вираз що стоїть слідом, приймає значення **True** (істина);

Оператор циклу Do-Loop Until

Синтаксис оператора:

Do

оператори
оператори
.....
Loop Until *умова припинення виконання циклу*

Until визначає виконання *операторів*, що входять в цикл, до тих пір, поки логічний вираз що стоїть слідом, не прийме значення **True** (істина);

Оператор циклу Do While .... Loop

Синтаксис оператора:

Do While *умова продовження виконання циклу*
оператори
оператори
.....
Loop

Запис умови на початку або в кінці циклу визначає, де ця умова (задається логічним виразом) буде перевірятися.

Коли умова перевіряється на початку циклу, цикл виконується, якщо умова виконується (значення логічного виразу дорівнює **True**).

Оператор циклу Do Until .... Loop

Синтаксис оператора:

Do Until *умова припинення виконання циклу*
оператори
оператори
.....
Loop

Оператор циклу For-Next

Синтаксис оператора наступний:

For *параметр циклу = початкове значення* **To** *кінцеве значення* [**Step** *крок*]
оператори
оператори
.....
Next *параметр циклу*

Пара **For-Next** визначають початок і кінець оператора циклу. *Оператори* між ними (блок операторів) повторюються стільки разів, скільки визначено заданими *початковим значенням, кінцевим значенням і кроком*.

Параметр циклу - арифметична змінна, не може бути елементом масиву або елементом типу даних, що визначає користувач.

Початкове значення, кінцеве значення і *крок* визначають значення, які приймає параметр циклу при роботі програми - на першому кроці параметр циклу приймає початкове значення, після виконання *операторів*, що входять в цикл (блок операторів), параметр циклу змінюється на величину кроку (виконується оператор **Next**), знову виконуються оператори, що входять в цикл, параметр циклу змінюється на величину кроку і т.ін., поки параметр циклу не прийме послідовно всі свої значення.

Після того, як параметр циклу прийме всі свої значення і відповідне число раз виконається блок операторів в циклі, буде виконуватися наступний за **Next** оператор.

При використанні оператора циклу необхідно дотримуватися правил:

- слід уникати зміни значення параметра циклу в будь-яких операторах всередині циклу;
- передача управління на оператори всередині циклу (крім першого) з будь-яких операторів поза циклом заборонена.

Існує два варіанти використання оператора **For-Next**: з додатнім кроком і від'ємним.

У першому випадку *початкове значення* **менше** *кінцевого значення* і *крок* **додатній**.

У другому - *початкове значення* **більше** *кінцевого значення* і *крок* **від'ємний.**

Спільне використання операторів циклу і умовного операторів

При одночасному використанні в процедурах і функціях операторів циклу і умовних операторів повинно виконуватися так зване правило вкладеності.

Якщо серед операторів, що виконуються в циклі (цикли **For-Next** і **Do-Loop**), мається умовний оператор (**If-Endlf**), то умовний оператор повинен цілком міститися всередині циклу (між операторами **For-Next** або **Do-Loop**).

Якщо в **Then**-блоці або **Else**-блоці умовного оператора **If-End lf** є оператори, що виконуються в циклі (цикли **For-Next** і **Do-Loop**), то ці цикли повинні цілком знаходитися в цих блоках.

3.3 TURBO PASCAL

3.3.1 Основні відомості про мову програмування Turbo Pascal

Історія розробки мови Pascal

Алгоритмічна мова Turbo Pascal була розроблена швейцарським ученим Никлаусом Віртом в 1967 році як мова для навчання процедурному програмуванню. Назва мові дана на честь видатного французького математика, фізика, літератора і філософа Блеза Паскаля.

У даний час – це середа розробки програмного забезпечення. Назва Borland Pascal була зарезервована для дорогих варіантів поставки (з більшою кількістю бібліотек і початковим кодом стандартної бібліотеки), оригінальна дешева і широко відома версія продавалася як Turbo Pascal. Назва Borland Pascal також використовується в більш широкому значенні – як позначення діалекту мови Pascal від фірми Borland.

Інтегрована інструментальна оболонка

Інтегроване середовище програмування розроблено фірмою Borland. Середовище програмування характеризується високою швидкістю компіляції програм, ретельно продуманою та зручною середою для роботи. Виявлення помилок при компіляції програми спричиняє останов обробки програми з видачею на екран повідомлення про характер помилки.

Таким чином, наявність в одному середовищі редактора, компілятора і відладчика значно збільшує ефективність розробки програм.

Для ефективного управління процесом обробки програми користувачу необхідно мати уявлення про технологію проходження програми в інтегрованому середовищі.

Виконуються послідовно такі дії:

1. Створюється початковий текст програми.
2. Програма записується на диск у вигляді файлу.
3. Програма запускається на компіляцію.
4. Знаходяться та усуваються синтаксичні помилки.
5. Програма запускається на виконання.

Запуск інтегрованого середовища програмування здійснюється таким чином:

1. Знаходимо директорію з ім'ям TP7 (Turbo Pascal сьома версія).
2. Знаходимо файл turbo.exe.

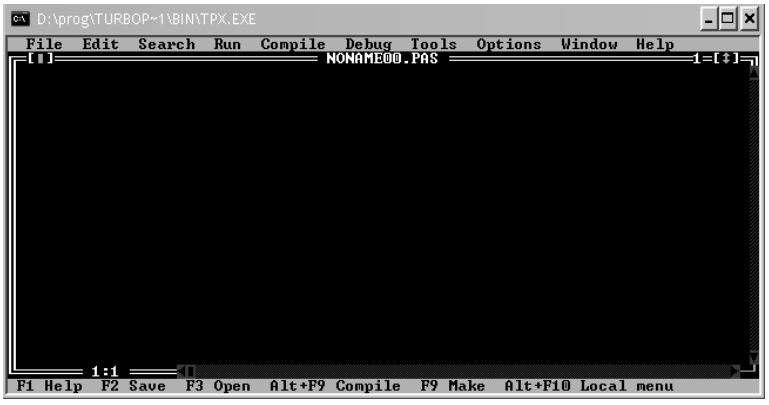


Рисунок 3.17- Інтегрована інструментальна оболонка Turbo Pascal

При натисненні клавіші Enter відбудеться запуск програми, що встановлює інтегроване середовище програмування з видачею на екран основного меню.

Після завантаження системи екран розділений на три частини: основне (або робоче) вікно, головне меню і рядок, в якому указується призначення основних функціональних клавіш. Перехід з основного вікна в головне меню і назад здійснюється за допомогою клавіші F10.

У робочому вікні здійснюється набір тексту програми, запуск же відбувається таким чином: вихід в меню, вибір пункту Run – Run.

Для того, щоб зберегти програму, необхідно: вийти в меню, вибрати File – Save (Save as..), у вікні, що з'явилося, ввести ім'я файлу і натиснути клавішу Enter.

Для всіх елементів основного меню виводиться перелік підрежимів. Для того, щоб активізувати основне меню (верхній рядок), необхідно натиснути клавішу F10, далі, переміщаючись за допомогою клавіш управління курсором, вибирається потрібний режим (опція), потім, натискуючи клавішу Enter, відкривається падаюче меню, що містить підрежими.

Все управління середою Turbo Pascal здійснюється в основному за допомогою системи меню, що розвертаються.

Головне меню містить фактично лише зміст додаткових меню.

FILE (Файл) – дія з файлами і вихід з системи;

EDIT (Редагувати) – відновлення зіпсованого рядка і операція з тимчасовим буфером;

SEARCH (шукати) – пошук тексту, процедури, функції або місця помилки;

RUN (робота) – прогін програми;

COMPILE (компіляція) – компіляція програми;

DEBUG (відладка) – відладка програми;

TOOLS (інструменти) – виклик допоміжних програм (утиліт);

OPTIONS (варіанти) – установка параметрів середовища;

WINDOW (вікно) – робота з вікнами;

HELP (допомога) – звернення до довідкової служби.

Основні файли пакету Turbo Pascal:

- Turbo.exe – інтегроване середовище програмування;
- Turbo.hlp – файл, що містить дані для оперативної підказки;
- Turbo.tp – файл конфігураційної системи;
- Turbo.tpl – бібліотека стандартних модулів Turbo Pascal.

Для роботи в графічному режимі необхідні Graph.tru – модуль з графічними процедурами і функціями Turbo Pascal, декілька файлів з розширенням *.BGI – драйвери різних типів відеосистем ПК, декілька файлів з розширенням *.CHR, що містять векторні шрифти.

Вихід з системи програмування: вихід в меню, пункт File – Exit.

Алфавіт та словник мови Turbo Pascal

Текст програми на мові програмування Turbo Pascal є послідовністю рядків, що складаються з символів, утворюючих алфавіт мови.

Алфавітом мови програмування називають набір символів (дозволений до використовування і сприйманий компілятором), за допомогою якого можуть бути утворені величини, вирази і оператори даної мови.

Алфавіт мови Pascal включає всі символи, представлені в таблиці кодів, яка завантажена в оперативну пам'ять або зберігається в ПЗП комп'ютера. Кожному символу алфавіту відповідає індивідуальний числовий код від 0 до 255. Символи з кодами від 0 до 127 є так званою основною таблицею кодів ASCII (American Code for Information Interchange – Американський стандартний код для обміну інформацією). Їх склад і порядок визначені американським стандартом на коди обміну інформацією (ідентичні для всіх IBM – сумісних комп'ютерів).

До алфавіту мови Turbo Pascal належать:

1. Символи, що використовуються для складання ідентифікаторів (імен):

– латинські рядкові і прописні букви **A...Z, a...z ;**

– арабські цифри від **0** до **9;**

– символ підкреслення **_;**

2. Символи роздільники:

– пропуск, основне призначення якого розділення ключових слів і імен;

– управляючі символи (ASCII – коди від 0 до 31). Ці символи можуть застосовуватися при описі рядкових і символьних констант. Управляючі символи з ASCII– кодом 9 (табуляція), також кодами 10 і 13 (замикаюче рядок) використовуються як роздільники при написанні програм;

– спецсимволи: +–*/=^<>(){}[ ] . : ; # $.

3. Складові символи – це група символів, які сприймаються компілятором як єдине ціле: **<= => := (* *) (. .) .**

4. “Невживані” символи, чи символи так званої «розширеної» таблиці ASCII кодів, тобто символи, що мають коди від 128 до 255 (в цій області знаходяться символи алфавіту російської мови та символи псевдографіки на IBM – сумісних комп'ютерах), а також деякі символи з

основної таблиці ASCII (наприклад: & ! % “ та інші). Їх можна використовувати в тексті коментарів і у вигляді значень констант рядків або констант символів.

Символи з мови використовуються для побудови базових елементів –лексем. В Turbo Pascal визначені наступні класи лексем:

1) Службові (ключові або зарезервовані) слова: Begin, End, Var, Type, Label, Const, If, Then, Else, For, Do, While, Repeat та інші.

Службові слова не можна використовувати не за призначенням. Вони не можуть використовуватися як ідентифікатори.

2) Імена (або ідентифікатори) вводяться програмістом для позначення (в програмі) змінних, констант, типів, міток, процедур, функцій, об'єктів, моделей, полів в записах і т.п. Вони формуються тільки з букв і цифр, причому першою повинна бути буква або знак підкреслення. Довжина імені може бути довільною, але компілятор сприймає тільки перші 63 символи.

Ідентифікатори вводяться в програму за допомогою описів.

3) Зображення – група лексем, що позначають числа, символьні рядки і деякі інші значення.

4) Знаки операцій, які формуються з одного або декількох спеціальних символів або службових слів:

а) арифметичні операції: + (додавання), – (віднімання), * (множення), div(ділення цілих чисел), mod (залишок від ділення двох цілих чисел);

б) операції відношення: < (менше), > (більше), <= (не більше), >= (не менше), = (рівно), <> (не рівно);

в) логічні операції: and – логічне і, or – логічне або, not – логічне не, xor – виняткове або;

г) операції над множинами: * – перетин множин, + об'єднання множин, – віднімання множин, IN – приналежність множині.

5) Роздільники, які формуються із спеціальних символів.

6) Коментарі – довільна послідовність символів, у тому числі і російських букв, укладених у фігурні дужки {.} або (* . *), призначена для пояснень в програмі. Коментарі можуть знаходитися між будь–якими двома лексемами програми.

7) Пропуск, що не має графічного зображення, використовується для відділення лексем один від одного.

Правила побудови ідентифікаторів мови Pascal.

При створенні програм слід дотримуватись загальних правил побудови ідентифікаторів мови Turbo Pascal.

1. У ідентифікаторах можуть використовуватися тільки латинські букви, цифри і знак підкреслення. Використовування пропусків, крапок і інших спеціальних символів, а також букв російського алфавіту **неприпустимо.**

2. Ідентифікатори повинні начинтися **тільки** з букви або символу підкреслення.

3. Максимальна допустима довжина ідентифікаторів – 127 символів.

4. Регістр букв в ідентифікаторах значення **не має**, проте рекомендується виділяти прописними буквами смислові частини ідентифікатора. Наприклад, для позначення кількості книг ідентифікатор NumberOfBooks є більш наочним, ніж ідентифікатор numberofbooks.

Ідентифікатори рекомендується вибирати так, щоб вони передавали значення тієї або іншої величини. Очевидно, що ім'я ідентифікатора NumberOf Books незрівнянно більш інформативне, ніж ім'я ідентифікатора N.

3.3.2 Види та типи даних

Кожний тип даних характеризується так званим кардинальним числом – кількістю різних значень, що належать типу. Для кожного типу даних має бути строго визначений набір операцій, які можна застосовувати при обробці даних цього типу.

Рішення задач на ПК – це процес збору, обробки і передачі інформації. Тому задача будь–якої програми полягає в обробці даних. В Turbo Pascal дані діляться на константи і змінні.

Оголошення констант розміщуються в тексті програми після зарезервованого слова const в наступному форматі:

ідентифікатор = значення

У Turbo Pascal застосовується декілька стандартних видів констант:

- **Цілочисельні** константи. Це число не повинне містити десяткової точки.
- **Дійсні** константи**.** Можуть бути визначені числами, записаними в десятковому форматі даних з використанням десяткової точки.
- **Символьні** константи**.** Можуть бути визначені за допомогою деякого символу (ув'язненого в апострофи).
- **Рядкові** константи**.** Можуть бути визначені послідовністю довільних символів (ув'язнених в апострофи).

- **Константи,** що типізуються**.** Є ініціалізовані змінні, які можуть використовуватися в програмах нарівні із звичайних змінних. Кожній константі, що типізується, ставиться у відповідність ім'я, тип і початкове значення.

  Наприклад:

  year = 2001;
  symb = '?';
  money = 57.23;

У мові Pascal є декілька констант, до яких можна звертатися без попереднього оголошення. Такі константи називають *зарезервованими.* Розглянемо деякі з них:

• True. Цій константі відповідає логічне значення "Істина".

• False. Цій константі відповідає логічне значення "Неправда"**.**

Змінною називають елемент програми, який призначений для зберігання, корекції і передачі даних усередині програми. Змінна подібно скриньці, яка можна заповнити різними значеннями, що не можна зробити з константою.

Змінна характеризується ім'ям, типом і значенням.

Всі змінні, що використовуються в програмі, повинні бути оголошені в розділі опису з вказівкою їх типу. Обов'язковий опис типу приводить до надмірності в тексті програм, але така надмірність є важливим допоміжним засобом розробки програм і розглядається як необхідна властивість сучасних алгоритмічних мов високого рівня.

Тип змінної визначається типом даних, які зберігатимуться в цій змінній. Тип змінної обов'язково має бути визначений до того, як над нею будуть виконані які–небудь дії.

Ієрархія типів даних в мові Turbo Pascal слідуюча:

- Прості
  - Порядкові
    - Цілі
    - Логічні
    - Символьні
    - Перераховувані
    - Інтервальні
  - Дійсні
- Структуровані
  - Масиви
  - Рядки

- Множини
- Записи
- Файли

- Покажчики

Порядкові типи характеризуються тим, що кожний з них має кінцеве число можливих значень і з кожним з них можна зіставити деяке ціле число – порядковий номер значення. До порядкових типів відносяться:

– цілий тип;

– логічний (або булевий) тип;

– символьний тип;

– тип, що перераховує (або перераховуваний);

– обмежений (або інтервальний) тип, якого також називають тип–діапазон.

До будь–якого з них застосовна стандартна функція Turbo Pascal ORD(X), результатом якої є порядковий номер значення X. До порядкових типів можна також застосовувати функції:

PRED(X) – повертає попереднє значення порядкового типу;

SUCC(X) – повертає наступне значення порядкового типу.

Цілі типи даних

Ця група типів позначає безліч цілих чисел в різних діапазонах. Діапазон можливих значень цілих типів залежить від їх внутрішнього уявлення, яке може займати один, два або чотири байти.

Над цілими значеннями допустимі наступні арифметичні операції: + – складання, – – віднімання * – множення, / – розподіл і дві додаткові операції "типу розподіл", а саме, Div – розподіл без остачі, з відкиданням дробової частини і Mod – узяття залишку від цілочисельного розподілу.

При вживанні до цілих значень всіх цих операцій, окрім / – виходить результат цілого типу, а розподіл (/) завжди дає дійсний результат.

Логічні типи змінних **BOOLEAN** мають два значення **TRUE** і **FALSE**, займають один байт пам'яті.

Над значеннями цього типу допустимі операції порівняння, причому False < True.

Для них справедливі правила:

ORD(False)= True;

ORD(True)= False;

SUCC(False)= True;

PRED(True)= False.

З логічним типом пов'язані логічні операції: AND (И), OR (АБО), NOT (НЕ).

Символьні типи змінних **CHAR** можуть приймати значення з безлічі символів ASCII кодів, займають один байт пам'яті. Ця множина складається з 256 різних символів, впорядкованих певним чином. Воно містить символи рядкових і прописних букв, цифр і інших символів, включаючи спеціальних керівників символи. Кожному символу приписується ціле число в діапазоні від 0 до 255. Це число служить кодом внутрішнього представлення символу, його повертає функція ORD.

Якщо символьне значення має графічне зображення, то воно зображається в програмі відповідним знаком, укладеним в апострофи (одинарні лапки): 'A', 'B' ., 'a', 'b' ., '1', '2' ., '*', '+' і т.ін.

Якщо символ не має графічного зображення, то використовують іншу форму запису: #K, де К – цілочисельний код символу.

Наприклад: #13 – Enter; #27 – Esc; #8 – Backspace.

Оскільки символи впорядковані, то до типа Char застосовні операції порівняння, наприклад: 'A' < 'M'; 'A' < 'a' і т. п., а також стандартні функції:

CHR(b) – перетворить вираз b типу byte в символ і повертає його своїм значенням.

Наприклад: Chr(90) повертає як результат символ 'Z'.

ORD(S) – повертає як результат код символу S в таблиці символів ASCII.

Наприклад: Ord('Z') повертає код, рівний 90.

UPCASE(CH) – повертає прописну латинську букву, якщо CH – рядкова латинська буква, інакше повертає сам символ CH.

Наприклад: Upcase('z') повертає символ 'Z'.

PRED(S) – повертає символ, передуючий символу S.

SUCC(S) – повертає символ, наступний за символом S.

Тип даних**, що перераховує** (або перераховуваний)**,** задається списком значень (об'єктів), які можуть приймати змінні цього типу. При цьому кожний об'єкт має ім'я. Відповідність між значеннями перераховуваного типа і порядковими номерами цих значень встановлюється порядком переліку: перше значення в списку одержує порядковий номер 0, друге – 1 і т.ін. Максимальне число об'єктів в перераховуваному типі рівне 65366 значень.

До значень типу, що перераховує, застосовні стандартні функції Ord, Pred, Succ, а також операції відносин.

Змінні цього типу підвищують наочність програми і дозволяють автоматично контролювати допустимість значень змінних.

Розглянемо приклади порядкових типів.

1. Опис днів тижня:

TypeDays=(Monday,Tuesday,Wednsday,Thday,Friday,Sutterday,Sunday).

2. Опис місяців року:

TypeYear=(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec)

Таблиця 16 - Прості типи даних мови Turbo Pascal

Ідентифікатор	Довжина (байт)	Діапазон значень	Операції
Цілі типи			
integer	2	–32768..32767	+, –, /, *, Div, Mod,>=, <=, =, <>, <, >
byte	1	0..255	+, –, /, *, Div, Mod,>=, <=, =, <>, <, >
word	2	0..65535	+, –, /, *, Div, Mod,>=, <=, =, <>, <, >
shortint	1	–128..127	+, –, /, *, Div, Mod,>=, <=, =, <>, <, >
longint	4	–2147483648.. 2147483647	+, –, /, *, Div, Mod,>=, <=, =, <>, <, >
Дійсні типи			
real	6	$2{,}9x10^{-39} – 1{,}7x10^{38}$	+, –, /, *,>=, <=, =, <>, <, >
single	4	$1{,}5x10^{-45} – 3{,}4x10^{38}$	+, –, /, *,>=, <=, =, <>, <, >
double	8	$5x10^{-324} – 1{,}7x10^{308}$	+, –, /, *,>=, <=, =, <>, <, >
extended	10	$3{,}4x10^{-4932} – 1{,}1x10^{4932}$	+, –, /, *,>=, <=, =, <>, <, >
Логічний тип			
boolean	1	true, false	Not, And, Or, Xor
Символьний тип			
char	1	все символи кода ASCII	+,>=, <=, =, <>, <, >

До **дійсних типів даних** відноситься підмножина дійсних чисел, які можуть бути представлені у форматі з фіксованою крапкою і з плаваючою десятковою крапкою. Числа з фіксованою крапкою записуються у вигляді цілої і дробової частин числа.

Наприклад: 5.45, –0.001, 17.0, –19.1919, 0.143.

Запис числа не може починатися і закінчуватися крапкою.

Числа з плаваючою крапкою використовуються для запису чисел, що змінюються в широкому діапазоні значень (від дуже маленьких до дуже великих) і представляються в експоненціальному вигляді: mE+p, де m – мантиса (ціле або дробове число), E означає 10 в ступені, p – порядок (ціле число).

Наприклад, 5.18E+2 = 5.18 * 102 = 518.

Числа з плаваючою крапкою: 0.547E+3, 5.47E+2, 54.7E+1, 547.0E0, 5470E–1, 54700E–2 представляють одне і теж число 547.

Над значеннями дійсних типів допустимі чотири арифметичні операції: * , /, +, –.

У таблиці 1 приведені всі розглянуті прості типи даних мови Turbo Pascal, об'єм пам'яті, необхідний для зберігання однієї змінної вказаного типу, безліч допустимих значень і застосовні операції.

3.3.3 Арифметичні операції і стандартні функції

З операцій і операндів складаються вирази. Більшість операцій в мові Turbo Pascal є бінарними, тобто містять два операнди. Решта операцій є унарними і містять тільки один операнд.

У бінарних операціях використовується звичне представлення алгебри, наприклад: a+b. В унарних операціях операція завжди передує операнду, наприклад: –b.

Таблиця 17 - Арифметичні операції мови Turbo Pascal

+	складання;
–	віднімання;
*	множення;
/	ділення;
div	цілочисельне ділення;
mod	отримання залишку від цілочисельного ділення.

У складніших виразах порядок, в якому виконуються операції, відповідає пріоритету операцій.

Таблиця 18 - Пріоритет операцій

Операція	Пріоритет	Вид операції
@, not	перший (вищий)	унарна операція
* /, div, mod, and, shl, shr	другий	операція множеня, ділення,зсув
+, –, or, xor	третій	операція складаня
| = <> < > <= >=, in	четвертий (низький)	операція відншення

Для визначенні старшинства операцій є три основні правила:

1. По–перше, операнд, що знаходиться між двома операціями з різними пріоритетами, зв'язується з операцією, що має більш високий пріоритет.

2. По–друге, операція, що знаходиться між двома операціями з рівними пріоритетами, зв'язується з тією операцією, яка знаходиться зліва від нього.

3. По–третє, вираз, укладений в дужки, перед виконанням обчислюється, як окремий операнд.

Аргументами всіх функцій (окрім функції π, що не має аргументу) можуть бути дійсні і цілі числа. Значеннями всіх перерахованих функцій є дійсні числа.

Таблиця 19 - Стандартні математичні функції мови Turbo Pascal

Звертання	Тип аргументу	Тип результату	Функція
Abs(x)	Цілий, дійсний	Цілий, дійсний	Модуль аргументу
Arctan(x)	Цілий, дійсний	Дійсний	Арктангенс
Cos(x)	Цілий, дійсний	Дійсний	Косинус
Exp(x)	Цілий, дійсний	Дійсний	e^x – експонента
Frac(x)	Цілий, дійсний	Дійсний	Дробова частина x
Int(x)	Цілий, дійсний	Дійсний	Ціла частина x
Ln(x)	Цілий, дійсний	Дійсний	Натуральний логарифм
Random		Дійсний	Псевдовипадкове число [0,1]
Random(x)	Цілий	Цілий	Псевдовипадкове число [0,x]
Round(x)	Дійсний	Цілий	Округлення до найближчого цілого
Sin(x)	Цілий, дійсний	Дійсний	Синус
Sqr(x)	Цілий, дійсний	Дійсний	Квадрат x
Sqrt(x)	Цілий, дійсний	Дійсний	Коріння квадратне з x
Trunc(x)	Дійсний	Цілий	Найближче ціле, не перевищуюче x по модулю

Арифметичні вирази будуються з арифметичних констант, змінних, функцій і операцій над ними. Всі дані, що входять в арифметичні вирази, повинні бути одного типу, хоча іноді допускається використовувати в од ному виразі дані цілого і дійсного типів.

При побудові арифметичних виразів слід враховувати наступні правила:

1. Вираз записується в строчку. Наприклад, вираз: $\frac{2\cdot a\cdot x+3\cdot b\cdot y-4\cdot x}{2.5\cdot(a+b+c)}$ на Turbo Pascal записуватиметься таким чином:

(2*a*x+3*b*y–4*x)/(2.5*(a+b+c))

2. Дужки в арифметичних виразах тільки круглі. Число дужок, що відкриваються, повинне дорівнювати числу дужок, що закриваються.

3. Не можна записувати два знаки операцій підряд, без дужок, наприклад у виразі $\frac{3\cdot a+b}{-x}$ слід записати: (3*a+b)/(–x).

4. Порядок виконання арифметичних операцій зліва направо відповідно до старшинства операцій

Розглянемо порядок обчислення на прикладі:

$$\frac{(a\cdot\sin x+b\cdot\cos y)^2}{(a^2+b^2)\cdot\sin(\frac{x}{y})}$$

Представлений вираз обчислюватиметься в наступному порядку:

1) Обчислення функції Sin(x);

2) a*Sin(x);

3) Cos(y);

4) b*cos(y);

5) a*Sin(x)+b*Cos(y);

6) SQR(a*Sin(x)+b*Cos(y)) – набуто значення в чисельнику;

7) Обчислення x/y;

8) Sin(x/y);

8) a*a;

9) b*b;

10) SQR(a*a+b*b);

11) SQR(a*a+b*b)*Sin(x/y) – набуто значення в знаменнику;

12) чисельник/знаменник = отриманий результат.

У Turbo Pascal є 4 логічні операції: заперечення –NOT, логічне множення –AND, логічне складання – OR, що виключає «або» –XOR . Використані позначення: T – true, *F* – false.

Пріоритети операцій: not, and, or, xor. Операції відношення (= <> .) мають більш високий пріоритет, ніж логічні операції, тому їх слід брати в дужки при використовуванні по відношенню до них логічних операцій.

Таблиця 20 - Логічні операції

A	B	Not A	A and B	A or B	A xor B
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T

3.3.4 Основи побудови програм мовою Turbo Pascal

Загальна структура програм в Turbo Pascal

Програма на алгоритмічної мові Turbo Pascal є послідовністю операторів, за допомогою яких реалізується алгоритм рішення задачі. Будь–яка програма на мові TurboPascal складається з двох основних розділів: розділу описів даних і розділу операторів, і закінчується завжди символом «**.**».

Розділ описів може включати підрозділи опису міток, констант, типів, змінних, а також підпрограм, реалізовуваних у вигляді процедур або функцій. Якщо в програмі використовуються стандартні або бібліотечні модулі (Unit), то першою повинна стояти директива Uses, в якій перераховуються модулі, що використовуються.

Окрім двох основних розділів в програму можна і потрібно включати коментарі: пояснення до програми, дані про розробників і т.п. Частина програми, укладена у фігурні дужки { }, є **коментарем**. Ця частина ігнорується компілятором і ніяк не відображається на роботі програми. Альтернативно, коментар можна обмежувати парою символів кругла дужка + зірочка (* *).

Загальна структура програми на мові Turbo Pascal має вигляд:

Кожний з підрозділів розділу описів починається своїм ключовим словом. Проте, не всі підрозділи обов'язкові.

Оператори в програмі відділяються один від одного знаком ";". Запис операторів в рядку може починатися з будь–якої позиції. В одному рядку можна записати декілька операторів, а один оператор може бути записаний в декількох рядках.

```
Program      <ім’я програми>;
Use          < Розділ описи модулів>;
Label   <Розділ описи міток>;
Const   <Розділ описи констант>;
Type    < Розділ описи типів>;
Var          < Розділ описи змінних>;
Function     <Розділ описи функцій>;
Procedure    < Розділ описи процедур>;
```

розділ описів даних

розділ операторів

BEGIN

{Текст основної програми}

END.

У заголовку програми після службового слова Program приводиться ім'я програми (ідентифікатор). Ім'я може мати будь–яку довжину, з них для компілятора мають значення перші 63 символи. Заголовок програми виконує чисто декоративні функції і служить для сумісності з іншими компіляторами мови Turbo Pascal або задоволення естетичних запитів програміста.

Розділ оголошення модулів в Turbo Pascal дає можливість підключати об'єкти, описані у іншому місці, що використовуються в програмі. Такі об'єкти називаються модулями.

За ключовим словом USES в розділі опису модулів записується список імен стандартних (Crt, Dos, Printer, Graph, Turbo3, Graph3, Overlay) і призначених для користувача бібліотечних модулів, що використовуються:

USES < список стандартних і призначених для користувача модулів, що використовуються >;

Приклад:

Uses Crt, Dos, MyMod;

Мітка в програмі є правильним ідентифікатором або будь–яким цілим без знаку від 1 до 9999. Мітки повинні бути описані в підрозділі Label. Кожна мітка описується тільки один раз в кожній програмній одиниці (основній програмі або підпрограмах).

Label мітка_____; або Label мітка1, мітка2 ., міткаN;

У програмі мітка ставиться перед оператором, на якого передається управління і відділяється від нього символом ":".

Приклади опису міток:

Label m1, m2, met1, l1, lab, 125;

Будь–яка змінна, що використовується в програмі (і підпрограмах) повинна бути визначена (описана) в підрозділі Var **розділу описів**, кожна змінна описується тільки один раз в кожній програмній одиниці.

Визначення змінної повинне містити ім'я змінної і її тип, розділені двокрапкою.

VAR ім'я змінної : тип;

Приклади:

Var x:real; i: byte;

S: char; b : boolean;

Days: 1..31;

Змінні одного типу записуються один за одним через коми:

Var a, b, з : real;

I,j,k : byte;

m, months:1..12;

Для змінних, описаних в кожній програмній одиниці, відводиться певний об'єм пам'яті.

Змінні, описані в основній (головної) програмі, називають **глобальними змінними.** Загальний об'єм пам'яті, відведений під глобальні змінні, не повинен перевищувати 64 Кбайта.

Змінні, описувані в підпрограмах, називаються **локальними змінними.**

Опис типів

У найпростіших випадках тип змінних указується явно, при їх описі в розділі Var:

Var Ім'я змінної: тип;

Можна зіставити типу деяке ім'я і описати його в розділі Type:

Type Ім'я типу = Тип;

Наприклад:

Type Diapason = 1..1000;

T_days = 1..31;

T_symbol = 'a'..'z';

T_Month = (j,f,mr,ap,may,jn,jl,ag,s,o,n,d);

Це дає можливість програмісту визначати і використовувати свої власні типи, а не стандартні.

Далі можна імена типів, введені в підрозділі Type, використовувати в підрозділі Var:

Var

i, n :Diapason;

Day: T_days;

Sim, ch: T_symbol;

Mes: T_Month;

Опис простих констант і констант, що типізуються

Прості константи можуть бути задані явно своїм значенням (0.5, 0, 100, 3.14, 'A', –5 і т.п.) або позначені ім'ям і в цьому випадку константи повинні бути описані в підрозділі Const:

Const Ім'я константи = Значення;

Наприклад:

Const N=200; A=0.5; sym='*';

Як значення константи можуть бути використані цілі і дійсні числа, рядки символів, ідентифікатори інших констант, константні вирази.

Наприклад:

Const Max=100; Min=10; S=(Max+Min) div 2;

Окрім простих констант використовують так звані константи або змінні, що типізуються, із стартовим значенням. Вони займають проміжне положення між простими константами і змінними, що відображається в наступних їх властивостях:

1. Константи, що типізуються, описуються в підрозділі Const своїм ім'ям.
2. Вони, як і константи, набувають своє початкове значення.
3. Аналогічно змінним, вони мають тип, який задається при їх описі.
4. Вони можуть, як змінні, набувати нові значення, в процесі роботи програми.

Таким чином, назва "константа" достатньо умовна.

Константи, що типізуються, можна використовувати як звичні змінні, але їм привласнюються початкові значення.

Опис констант, що типізуються:

Const {Ім'я константи, що типізується}: Тип = Константа, що типізується;

У свою чергу, поняття константи, що типізується, може включати одне з:

- Звична константа
- Константа посилального типу
- Ідентифікатор програми
- Зображення масиву
- Зображення множини
- Зображення запису
- Зображення об'єкту

Приклади:

Const Max:integer=999; Min:real=–0.01; Index:1..1000=1;

Приклади складніших констант, що типізуються, будуть приведені при описі відповідних типів в розділі 2.

«**Процедура»** і «**функція»** – терміни, вживані в Turbo Pascal для позначення спеціальним чином оформленої послідовності команд (підпрограми). Доступ до такої підпрограми може бути здійснений з будь–якого місця основного блоку програми, а також з будь–якої процедури або функції, опис яких слідує нижче. Розділ описів процедур і функцій є текстом процедур і функцій, які будуються за правилами, аналогічними правилами побудови програми.

3.3.5 Оператори Turbo Pascal

Розділ операторів слідує відразу за розділом описів і завжди полягає в операторні дужки, визначувані ключовими словами **Begin ... End.**

Робота програми починається саме з першого оператора основного блоку програми.

Основна частина програми на мові Turbo Pascal є послідовністю операторів, кожний з яких проводить деяку дію над даними.

Існує два основні види операторів: прості і структурні.

Простим оператором є оператор, який не містить в собі інших операторів.

Одним з простих операторів є **оператор привласнення.** Він наказує виконати вираз, заданий в його правій частині, і привласнити результат змінної, ідентифікатор якої розташований в лівій частині.

Оператор привласнення має вигляд:

Змінна:= Значення;

Зліва в операторі привласнення завжди стоїть ім'я змінної, а справа – те, що є її значенням (це може бути конкретне значення, арифметичний або логічний вираз, виклик функції, або інша змінна). Після виконання операції привласнення змінна зліва набуває нове значення. Необхідно стежити за сумісністю типів даних, що беруть участь в операції привласнення.

Приведемо деякі приклади операторів привласнення:

A:=0.5; – змінною A буде привласнено конкретне значення, рівне 0,5.

X:= 2*A+1; – змінною X, після обчислення правої частини, буде привласнено знайдене значення.

Складовий оператор задає порядок виконання операторів, що є його елементами. Вони повинні виконуватися в тому порядку, в якому вони записані. Складові оператори обробляються, як один оператор, що має вирішальне значення там, де синтаксис Turbo Pascal допускає використовування тільки одного оператора. Оператори полягають в обмежувачі begin і end, і відділяються один від одного крапкою з комою.

Приведемо приклад складового оператора:

```
begin
Z := X;
X := Y;
Y := Z;
end;
```

3.3.6 Процедури введення і виведення даних

Під введенням даних мається на увазі передача програмі на обробку якої-небудь інформації ззовні. Виведення даних – це зворотний процес, коли результати обробки інформації передаються програмою на різні пристрої комп'ютера (екран монітора, принтер, жорсткий диск і ін.).

Процедура введення даних використовується у вигляді:

а) read (a1, a2, a3 ., an) – кожне значення, що вводиться, привласнюється послідовності змінних a1, a2, a3,.., an;

б) readln(a1, a2, a3,.., an) – кожне значення, що вводиться, привласнюється послідовності змінних a1, a2, a3 ,.., an, після чого відбувається перехід на новий рядок (наступний оператор введення вводитиме дані з нового рядка);

в) readln – перехід на новий рядок при введенні даних.

Процедура виводу Write проводить виведення даних.

Загальний вигляд: Write(<список вивода>);

У списку виводу можуть бути представлені вирази допустимих типів даних (integer, real, char і т.ін.) і довільний текст, укладений в апострофи.

Наприклад, Write(‘Привіт’); Write(34.7); Write(45+55); Write(b, d);

Процедура Writeln аналогічна процедурі Write. Відмінність в тому, що після виведення останнього в списку виразу курсор переходить на початок нового рядка.

У процедурах виводу Write і Writeln є можливість запису виразу, що визначає ширину поля виводу. Ширина поля виводу визначається типом пристрою, що використовується в даному ЕОМ. Форма представлення змінних, що виводяться, визначається типом змінних: значення величин цілого типу виводиться в звичній формі; значення величин дійсного типу – у вигляді нормалізованого числа дійсного типу з порядком: значення логічного типу – у вигляді логічних значень **TRUE** і **FALSE**; значення символьних змінних – у вигляді відповідних символів.

Загальний вид запису операторів при виведенні значень цілого типу:

Write(b:m); writeln(b:m);

де: b – ім'я змінної, що виводиться; m – константа або вираз цілого типу, що відводиться під значення.

При виведенні значень дійсного типу з фіксованою крапкою указується ширина поля, що відводиться під все значення і під дробову частину числа.

Загальний вид запису операторів виглядає таким чином:

Write(b:m:n); writeln(b:m:n);

де: m – поле, що відводиться під запис значення; n – частина поля, що відводиться під дробову частину числа.

Наприклад: **Write(a:8:3);**

У даному випадку під значення a виділяється вісім позицій, три з яких відводиться під дробову частину числа. Якщо при виведенні дійсних значень не указується кількість позицій, відведених під дробову частину числа, то результат виходить в нормалізованому вигляді з десятковим порядком.

Можна задавати кількість пропусків. Для цього необхідно записати оператор виводу у вигляді **Write(‘ ‘:q),** де q – константа цілого типу, вказуюча число пропусків.

Приклад розміщення інформації при виводі:

```
Program primer;
const pi=3.141592; t=401; w=true; sim=’d’;
begin
writeln(‘pi=’, pi:8:6);
writeln(t:6 ‘ ‘:6, w:4 ‘ ‘:6, sim:1)
end.
```

У результаті виконання програми на екрані з'явиться результат:

```
Pi=3.141592
_ _ _401_ _ _ _ _ _true_ _ _ _ _ _d
```

При необхідності виведення даних **на друкарський пристрій** програму слід організувати з оголошенням модуля підключення принтера:

```
Program {ім'я програми};
Uses Printer;
…..
Write (Lst <список ввода>);
….
```

При цьому друкуюче пристрій повинне бути готовим до роботи (дані на екран дисплея виводитися не будуть).

Програмування алгоритмів структури, що розгалужуються

У мові Turbo Pascal використовуються два оператори для реалізації умовних переходів – IF і CASE, а також оператор безумовного переходу GOTO. Вони дозволяють порушити послідовний порядок виконання інструкцій програми.

Оператор умовного переходу в Turbo Pascal має вигляд:

if *умова* then *оператор 1* else *оператор 2*;

де: *умова* – це логічний вираз, залежно від якого вибирається одна з двох альтернативних гілок алгоритму. Якщо значення умови істинне (TRUE), то виконуватиметься *оператор 1*, записаний після ключового слова then. Інакше буде виконаний *оператор 2*, наступний за словом else, при цьому *оператор 1* пропускається. Після виконання вказаних операторів програма переходить до виконання команди, що стоїть безпосередньо після оператора if (базова структура «галуження»).

Необхідно пам'ятати, що перед ключовим словом else крапка з комою не ставиться.

Частина else в операторі if може бути відсутня:

if *умова* then *оператор 1*;

Тоді у разі невиконання логічної умови управління відразу передається оператору, що стоїть в програмі після конструкції if (структура «обхід»).

Слід пам'ятати, що синтаксис мови допускає запис тільки одного оператора після ключових слів then і else, тому групу інструкцій обов'язково треба об'єднувати в складовий оператор (оздоблювати операторними дужками begin ... end). Інакше виникає частіше за все логічна помилка програми, коли компілятор мови помилок не видає, але програма проте працює неправильно.

Приклади.

```
if x > 0 then modul := x else modul := –x;

if k > 0 then WriteLn('k – число додатнє');

if min > max then
begin
```

```
t := min;
min := max;
max := t;
end;
```

Проте, часто виникають ситуації, коли доводиться здійснювати вибір одного з декількох альтернативних шляхів виконання програми. Не дивлячись на те, що такий вибір можна організувати за допомогою оператора if .. then, зручніше скористатися спеціальним оператором вибору. Його формат:

case *вираз* **of**
варіант : *оператор*;
...
варіант : *оператор*;
end;
або
case *вираз* **of**
варіант : *оператор*;
...
варіант : *оператор*;
else *оператор*
end;

Вираз, який записується після ключового слова case, називається ***селектором***, воно може бути будь–якого перераховуваного типу. *Варіант* складається з однієї або більшої кількості констант або діапазонів, розділених комами. Вони повинні належати до того ж типу, що і селектор, причому неприпустимо більше однієї згадки *варіанту* в записі інструкції case. З перерахованої безлічі *операторів* буде вибраний тільки той, перед яким записаний *варіант*, співпадаючий із значенням селектора. Якщо такого варіанту немає, виконується оператор, наступний за словом else (якщо він є).

При використовуванні оператора Case повинні виконуватися наступні правила:

1. Вираз–селектор може мати тільки простий порядковий тип (цілий, символьний, логічний).

2. Всі константи, які передують операторам альтернатив, повинні мати той же тип, що і селектор.

3. Всі константи в альтернативах повинні бути унікальні в межах оператора вибору.

Форми запису оператора:

Селектор інтервального типу:

```
Case I
1..10 : writeln (‘число в діапазоні 1–10’);
11.. 20 : writeln (‘число в діапазоні 11–20’);
else writeln (‘число зовні меж потрібних діапазонів’)
end;
```

Селектор цілого типу:

```
Case I
1 : біля:= I+10;
2 : біля:= I+20;
3:  біля:= I +30;
end;
```

Приклад

```
case ch
'A'..'Z', 'a'..'z' : WriteLn('Буква');
'0'..'9': WriteLn('Цифра');
'+', '–', '*', '/' : WriteLn('Оператор');
else WriteLn('Спеціальний символ')
end;
```

Циклічні алгоритми.Оператори циклів

У мові Turbo Pascal є три різні оператори, за допомогою яких можна запрограмувати фрагменти програм, що повторюються: з параметром, з передумовою і з постумовою.

Оператор циклу з постумовою **REPEAT** складається із заголовка, тіла і умови закінчення **UNTIL**.

REPEAT <Оператор 1>;
<Оператор 2>;
. . . . . .
<Оператор N>;
UNTIL <умова – логічний вираз>;

Оператори, укладені між словами **REPEAT . . . UNTIL**, є тілом циклу. На початку виконується тіло циклу, потім перевіряється умова виходу з циклу. Якщо результат булевого виразу рівний False, то цикл активізується ще раз, якщо результат True – відбувається вихід з циклу. Принаймні один з операторів тіла циклу повинен впливати на значення умови, інакше цикл виконуватиметься нескінченно.

Оператор циклу з передумовою **WHILE** аналогічний оператору **REPEAT**, але перевірка умови виконання циклу проводиться на початку оператора.

WHILE<умова> **DO**<тіло циклу>;

<умова> – булевий вираз <тіло циклу>– простий або складовий оператор. Перед кожним виконанням тіла циклу обчислюється значення виразу умови. Якщо результат рівний True, то тіло циклу виконується і знову обчислюється вираз умови, якщо результат рівний False, відбувається вихід з циклу і перехід до першого після WHILE оператора.

Якщо число повторень тіла циклу наперед відоме, то частіше за все застосовується *оператор циклу з параметром*:

FOR <параметр циклу> := <поч. знач.> **TO** <кін. знач.> **DO** <оп–р>

або

FOR <параметр циклу> := <поч.знач.> **DOWNTO** <кін.знач.> **DO**<оп–р>

де <параметр циклу> – величина, яка змінюється в циклі (змінна типа Integer); <поч.знач.> – величина, задаюча початкове значення параметра циклу; <кін.знач.> – величина, задаюча кінцеве значення параметра циклу.

Крок нарощування параметра циклу строго рівний 1. При заміні зарезервованого слова TO на DOWNTO крок нарощування параметра циклу рівний (–1).

Для більш гнучкого управління операторами циклів For, While, Repeat в Turbo Pascal введено дві процедури:

BREAK – реалізує негайний вихід з циклу; дія процедури полягає в передачі управління оператору, що стоїть відразу після останнього оператора циклу.

CONTINUE – забезпечує дострокове завершення чергового проходу циклу, що еквівалентне передачі управління в самий кінець циклічного оператора**.**

Типові алгоритми обробки масивів

Масив – це впорядковані дані одного типа, що складаються із змінних (елементів масиву) та подані під одним ім’ям. Масивом часто позначають характеристики об'єктів одного типа, що мають однакові одиниці виміру.

Масив може бути багатовимірним (без обмеження), але займати не більше 65520 байт.

Слід відрізняти розмірність масиву від розміру масиву. Якщо розмірність масиву – це, по суті, число індексів, що визначають положення елементу в масиві, то розмір масиву – це число елементів в масиві.

Таким чином, якщо об'єкти одного типа позначити ім'ям, наприклад "A", то елементи масиву будуть A[1], A[2] і так далі.

У квадратних дужках вказаний номер елементу.

Порядковий номер елементу масиву, зазвичай не несе жодній інформації про значення елементу, а показує розташування елементу серед інших. До елементів масиву можна звертатися лише по їх номеру (індексу).

Значення елементам масиву привласнюються також як і іншим змінним з врахуванням типа масиву. Якщо елементи масиву мають один індекс, то масив називається одновимірним або лінійним, або масив – вектор. Значення елементів одновимірного масиву зазвичай виводять на екран або папір у вигляді стовпця або рядка.

Синтаксис опису масиву має вигляд:

var <ім'я масиву>: array[<список індексних типів>] of <тип елементу>;

Приклади опису масивів в розділі змінних:

```
VAR  M1: array[1..200] of integer;
A1: array[100..200] of real;
ch1: array['A'..'Z'] of char;
ch2: array[0..255] of char;
{ M1,  A1 – одновимірні масиви цілих і речових чисел }
 { ch1,  ch2 – одновимірні масиви символів }
```

Приклади привласнення значення:

```
M1[1]:=5; M1[2]:=8; M1[3]:= M1[1]; M1[100]:=32;
A1[101]:=0.2; A1[102]:=2.31;
ch1['B']:=’C’; ch2[1]:=’!’
```

Для опису масиву скористаємося розділом типів:

type < ім'я типа>= array[<список індексних типів>]

of <тип елементу>;

var <ім'я масиву> : <ім'я типа>;

де ім'я типа - ідентифікатор.

Якщо відома залежність, по якій змінюються значення елементів масиву, то привласнення значень зручно проводити в операторах циклу з параметром або з умовою. Наприклад, привласнимо значення елементам масиву "y" по залежності: y=sin(x), де x= Pi * i/180, 0<= i <=180 .

```
For i:= 0 to 180 Do в[i]:= sin(Pi * i/180);
```

Привласнимо значення семи елементам масиву "A" оператором Readln:

```
For i:= 1 to 7 Do begin
Write( ' Введіть A[ ', i ' ]= ' );
Readln(A [i]) end;
```

Виведення значень ста елементів масивів "S" і "A" в три пари колонок проведемо операторами:

```
For i:= 1 to 100 do  begin Write('(', s[i]:11',(', a[i]:8,  '(');
if (i mod 3) = 0 Then Writeln;
if (i mod 60) = 0 then readln end;
```

В цьому випадку дані таблиці повністю не уміщаються на екрані і можна затримати прокрутку екрану при виведенні даних, застосовуючи оператора Readln після виводу, наприклад, 20 рядків.

У циклі зручно визначати суму елементів масиву, найбільший (найменший) елемент і створювати нові масиви, що задовольняють деякій умові, наприклад:

```
s:= 0; for i:= 1 to 100 do  s:= s + a[i]; {s – сума елементів масиву }
a_max:= a[1];  for i:= 1 to 100 do { пошук найбільшого елементу a[j]}
if a[i]> a_max then begin a_max:= a[i];  j:= i  end;


j:= 0; k:= 0;
for i:=1 to 100  do {створення нових масивів з елементами: b[j] >=0, з[k] <0}
if a[i] >= 0  then begin  j:= j+1;  b[j]:= a[i] end
else begin  k:= k+1;  з[k]:= a[i] end;
j:= 0;  k:= 8;
for i:= 1 to 100 do{створення масиву номерів "M" для елементів: a[i]> a[k]}
if a[i]> a[k]  then begin  j:= j+1;  M[j]:= i   end;
```

Двовимірні масиви

Двовимірні масиви мають елементи, впорядковані по двох індексах і часто називаються матрицями.

У Turbo Pascal при описі багатовимірного масиву діапазони зміни індексів перераховуються через коми, наприклад:

Var A: array[1..30, 1..7] of byte;

Умова вигляду «дана матриця розміру M x N» означає, що спочатку дається фактичний розмір двовимірного масиву–матриці (кількість рядків M і кількість стовпців N), а потім наводяться елементи цього масиву (кількість елементів рівна MxN). Якщо в завданні явно не вказується, які значення можуть приймати розміри вихідної матриці, то передбачається, що і число рядків, і число стовпців може мінятися в межах від 2 до 10. Порядкові номери початкового рядка і початкового стовпця матриці вважаються рівними 1. Введення і виведення елементів матриці здійснюються по рядках.

Квадратною матрицею порядку M називається двовимірний масив–матриця розміру M x M.

Наведемо приклад опису двовимірного масиву (матриці) з N рядків і M стовпців:

```
const N=10; M=10;
type T_el = integer;
```

Перший спосіб.

```
Var C:array[1..N,1..M] of T_el;
```

Другий спосіб.

```
Type matr= array[1..N,1..M] of T_el;
Var C:matr;
```

У 1–ом і 2–ом способах доступ до елементу і–того рядка і j–того стовпця позначається як C[i,j].

Третій спосіб.

```
Type vect= array[1..M] of T_el;
 matr= array[1..N] of vect;
Var C:matr;
```

Цей спосіб, окрім доступу до елементу C[i,j]або C[i][j], дає доступ до рядка. Наприклад, доступ до і–того рядка позначається як C[i]. При такому описі матриця трактується як вектор векторів.

Матриці як масиву виділяється суцільна область пам'яті. Компілятор з алгоритмічної мови Turbo Pascal використовує відрядкове розміщення елементів матриці в пам'яті комп'ютера. Це означає, що рядки матриці розташовуються послідовно один за одним незалежно від способу опису.

Пам'ять матриці виділяється компілятором статично, тобто у момент компіляції. Матриці C[1..N,1..M] буде виділено N*M*p байт, де p – кількість байт елементу типа T_el.

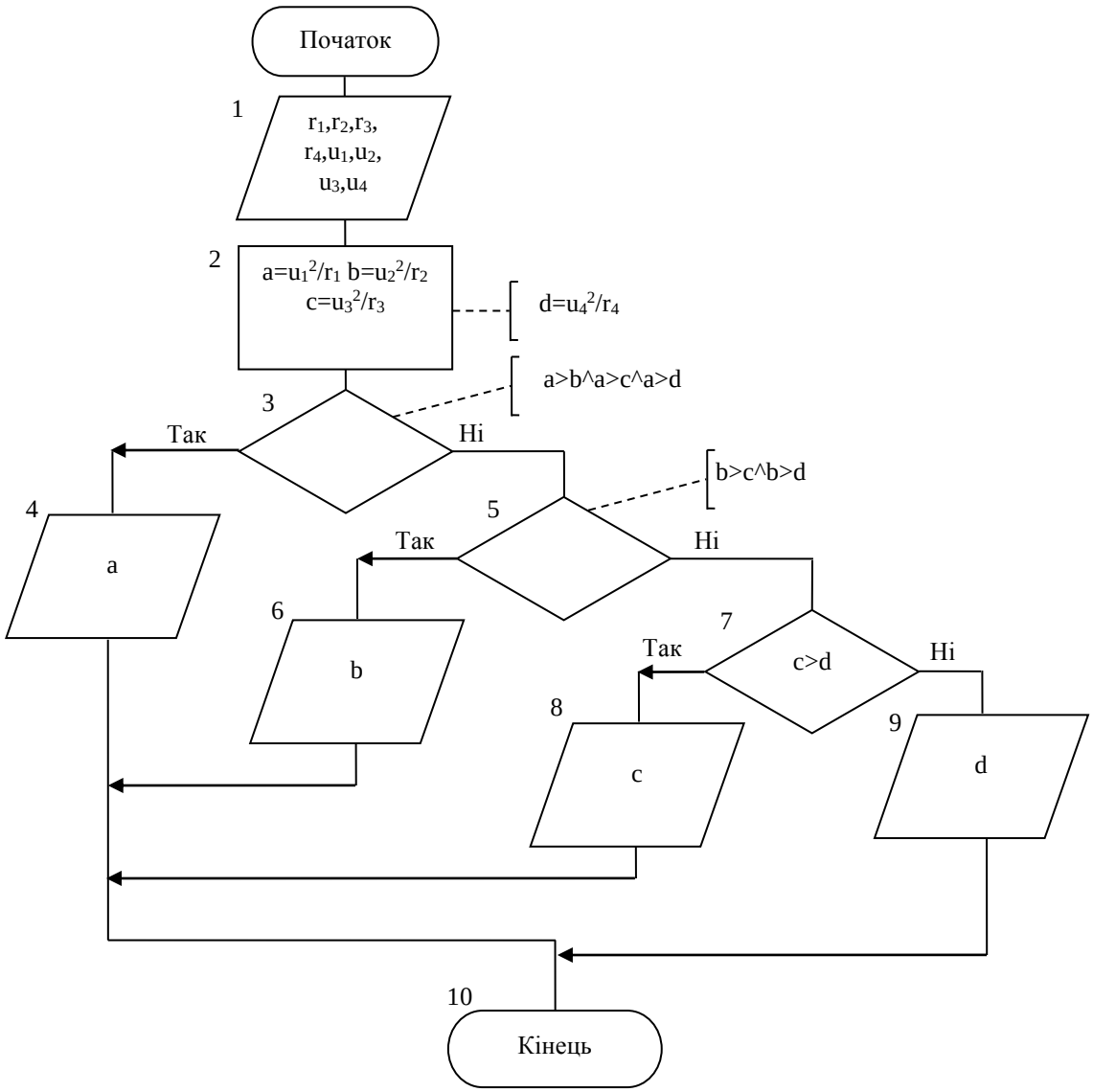
4 РЕШЕБНИК

4.1 Алгоритми, що розгалужуються

4.1.1 Пошук екстремума за значенням

Задача 1.

Чотири токарні верстати обладнані двигунами з електричними опорами *R1, R2, R3, R4.* Напруга на них, відповідно:*U1, U2, U3, U4*. Вказати значення потужності самого потужного верстата, враховуючи, що потужність розраховується за формулою $P=U^2/R$.



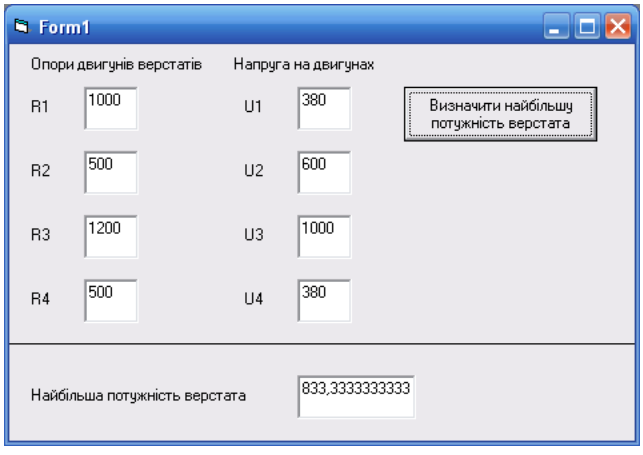
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
```

```
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        double r1,r2,r3,r4,u1,u2,u3,u4,a,b,c,d;
        cout<<"Введите значения сопротивления и напряжения:";
        cout<<"R1=";cin>>r1;
        cout<<"R2=";cin>>r2;
        cout<<"R3=";cin>>r3;
        cout<<"R4=";cin>>r4;
        cout<<"U1=";cin>>u1;
        cout<<"U2=";cin>>u2;
        cout<<"U3=";cin>>u3;
        cout<<"U4=";cin>>u4;
        a=u1*u1/r1;
        b=u2*u2/r2;
        c=u3*u3/r3;
        d=u4*u4/r4;
        if(a>b && a>c && a>d) cout<<a;
         else if(b>c && b>d) cout<<b;
         else if (c>d) cout<<c;
         else cout<<d;
        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim R1, R2, R3, R4, U1, U2, U3, U4, A, B, C, D As Single
R1 = CSng(Text1)
R2 = CSng(Text2)
R3 = CSng(Text3)
R4 = CSng(Text4)
U1 = CSng(Text5)
U2 = CSng(Text6)
U3 = CSng(Text7)
```



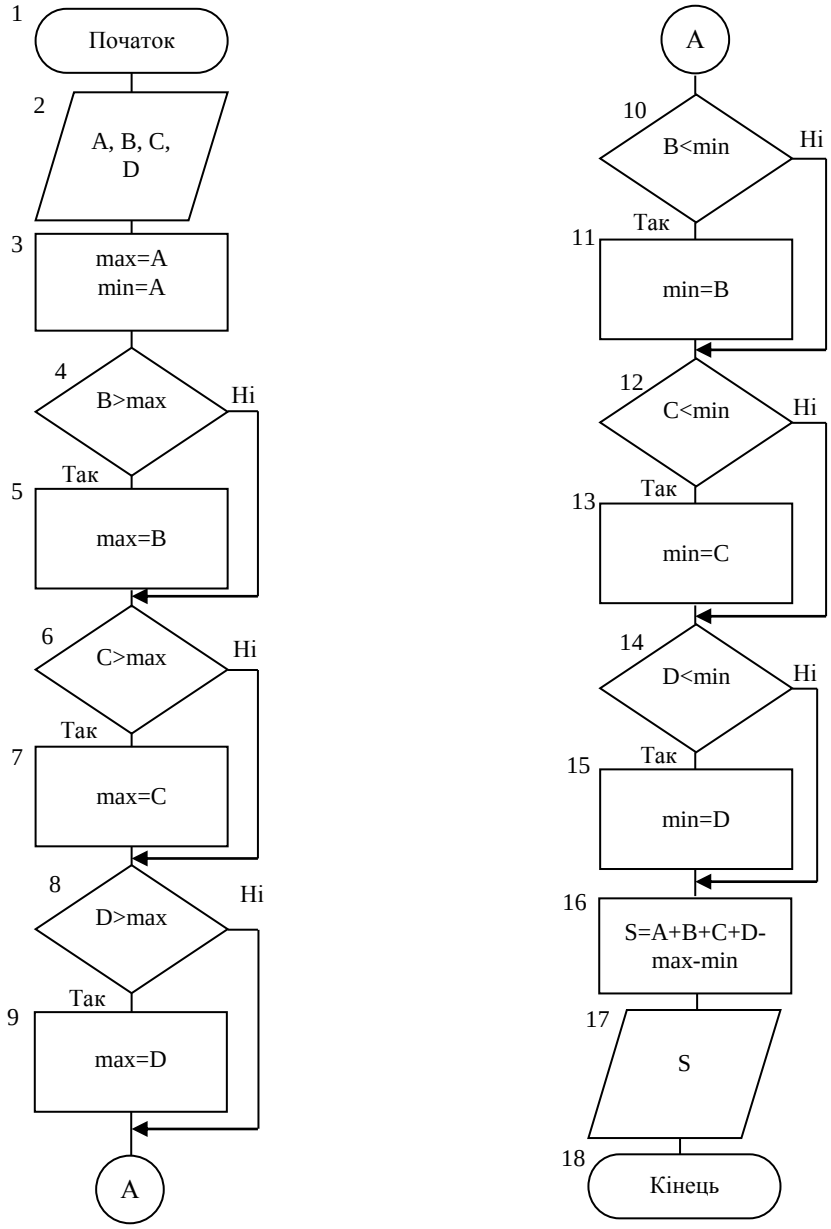
```
U4 = CSng(Text8)
A = U1 ^ 2 / R1
B = U2 ^ 2 / R2
C = U3 ^ 2 / R3
D = U4 ^ 2 / R4
If A > B And A > C And A > D Then
Text9 = CStr(A)
ElseIf B > C And B > D Then
Text9 = CStr(B)
ElseIf C > D Then
Text9 = CStr(C)
Else
Text9 = CStr(D)
End If
End Sub
```

TP

```
Program p1;
Var
r1, r2, r3, r4, u1, u2, u3, u4: real;
a, b, c, d: real;
Begin
Writeln (‘Введіть r1, r2, r3, r4, u1, u2, u3, u4’);
Readln (r1, r2, r3, r4, u1, u2, u3, u4);
a:=sqr(u1)/r1;
b:=sqr(u2)/r2;
c:=sqr(u3)/r3;
d:=sqr(u4)/r4;
If (a>b) and (a>c) and (a>d) then
writeln (‘a=’,a) else If (b>c) and (b>d) then writeln (‘b=’,b) else If (c>d) then writeln (‘c=’,c) else writeln ((‘d=’,d);
End.
```

Задача 2.

Із заданої послідовності чотирьох чисел виключити найменше і найбільше, а значення суми чисел, що лишилися, присвоїти змінній *S*.



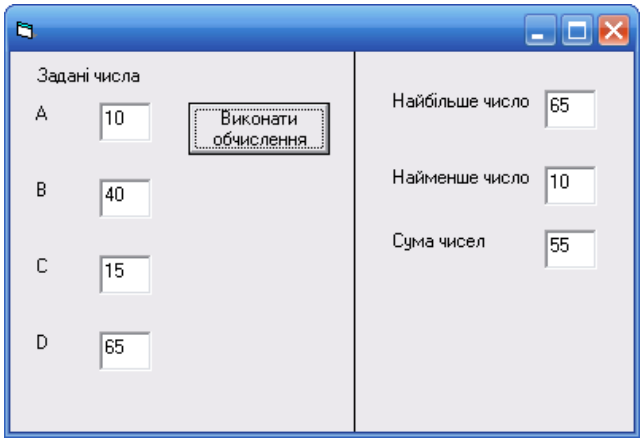
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
```

```
   SetConsoleOutputCP(1251);
        double A,B,C,D,max,min,S;
        cout<<"Введите 4 числа:"<<endl;
        cout<<"A=";cin>>A;
        cout<<"B=";cin>>B;
        cout<<"C=";cin>>C;
        cout<<"D=";cin>>D;

max=A;
min=A;
if(B>max) max=B;
if(C>max) max=C;
if(D>max) max=D;
if(B<min) min=B;
if(C<min) min=C;
if(D<min) min=D;

S=A+B+C+D-max-min;
cout<<"S="<<S<<endl;

        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
a = CSng(Text1)
b = CSng(Text2)
c = CSng(Text3)
d = CSng(Text4)
Max = a: Min = a
If b > Max Then Max = b
If c > Max Then Max = c
If d > Max Then Max = d
If b < Min Then Min = b
```



```
If c < Min Then Min = c
If d < Min Then Min = d
s = a + b + c + d - Max - Min
Text5 = CStr(Max)
Text6 = CStr(Min)
Text7 = CStr(s)
End Sub
```

TP

```
Program p2;
Var
a, b, c, d, max, min: real;
Begin
Writeln (‘введіть a, b, c, d );
Readln (a, b, c, d);
max:=a;
min:=a;
If b>max then max:=b else
    If c>max then max:=c else
        If d>max then max:=d else
If b<min then min:=b else
If c<min then min:=c else
If d<min then min:=d
S:=a+b+c+d-max-min;
Writeln (‘s=’,s:6:2);
End.
```

4.1.2 Пошук екстремума за змінною

Задача 3.

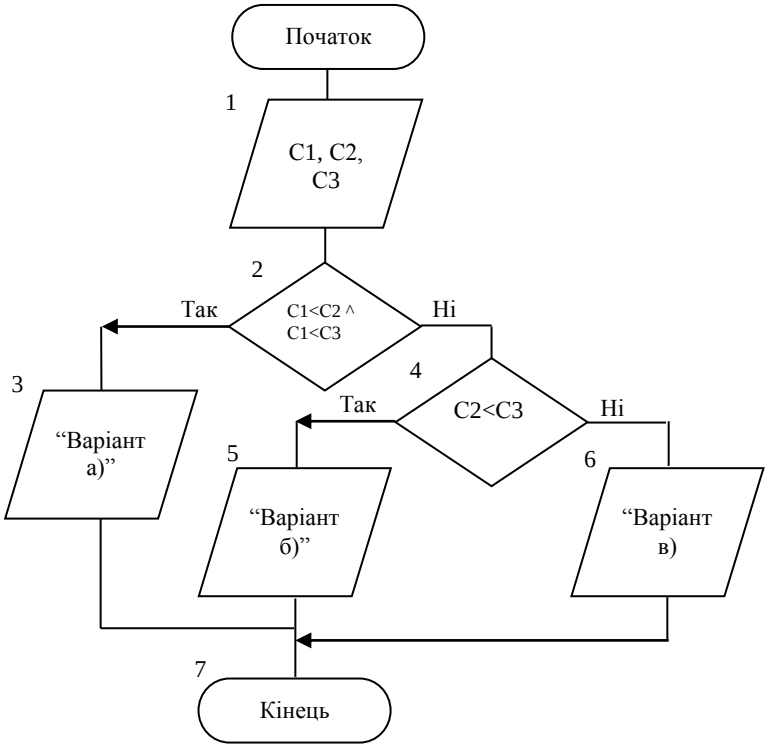
При будівництві телевізійної башти можливі варіанти:

а) монтаж на землі з підйомом у вертикальне положення;

б) вертикальний монтаж за допомогою пересувного крана;

в) вертикальний монтаж за допомогою вертольота.

Визначити найбільш економічний варіант, якщо відома вартість кожного.

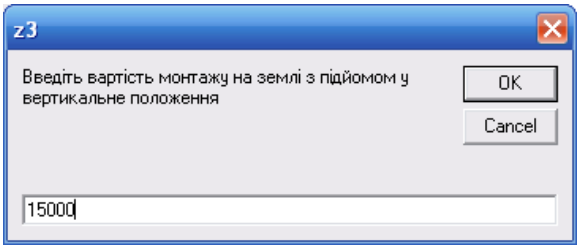


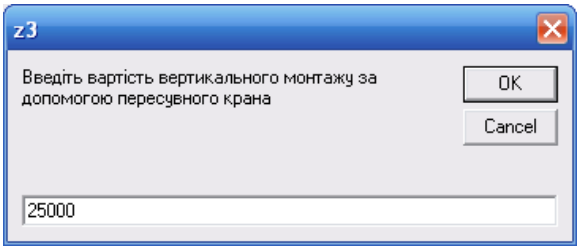
C++

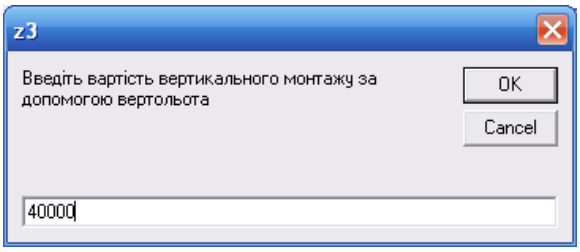
```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double C1,C2,C3;
	cout<<"Введите стоимость вариантов строительства:"<<endl;
	cout<<"C1=";cin>>C1;
	cout<<"C2=";cin>>C2;
	cout<<"C3=";cin>>C3;

if(C1<C2 && C1<C3) cout<<"Вариант а)";
else if (C2<C3) cout<<"Вариант б)";
else cout<<"Вариант в)";

	system("pause");
	return 0;
}
```

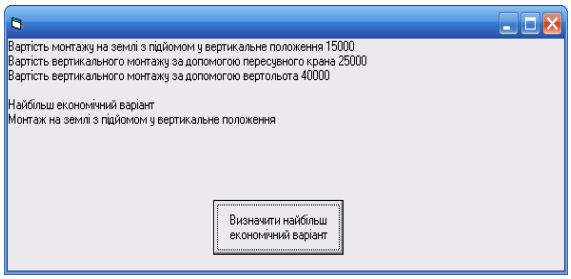
VB

```
Option Explicit
Private Sub Command1_Click()
Dim C1, C2, C3 As Single
C1 = CSng(InputBox("Введіть вартість монтажу на землі з підйомом у вертикальне положення"))
Print "Вартість монтажу на землі з підйомом у вертикальне положення"; C1
C2 = CSng(InputBox("Введіть вартість вертикального монтажу за допомогою пересувного крана"))
Print "Вартість вертикального монтажу за допомогою пересувного крана"; C2
C3 = CSng(InputBox("Введіть вартість вертикального монтажу за допомогою вертольота"))
Print "Вартість вертикального монтажу за допомогою вертольота"; C3
Print
Print "Найбільш економічний варіант"
If C1 < C2 And C1 < C3 Then
Print "Монтаж на землі з підйомом у вертикальне положення"
ElseIf C2 < C3 Then
Print "Вертикальний монтаж за допомогою пересувного крана"
Else
Print "Вертикальний монтаж за допомогою вертольота"
End If
End Sub
```









TP

```
Program p3;
Var
c1, c2, c3: real;
Begin
Writeln (‘введіть c1, c2, c3);
```

```
Readln (c1, c2, c3);
If (c1<c2) and (c1<c3) then writeln (‘варіант а’)
else If c2<c3 then writeln (‘варіант б’) else writeln (‘варіант в’);
End.
```

Задача 4.

При виготовлені складної деталі гідравлічного пресу можливі наступні варіанти технологічного процесу:

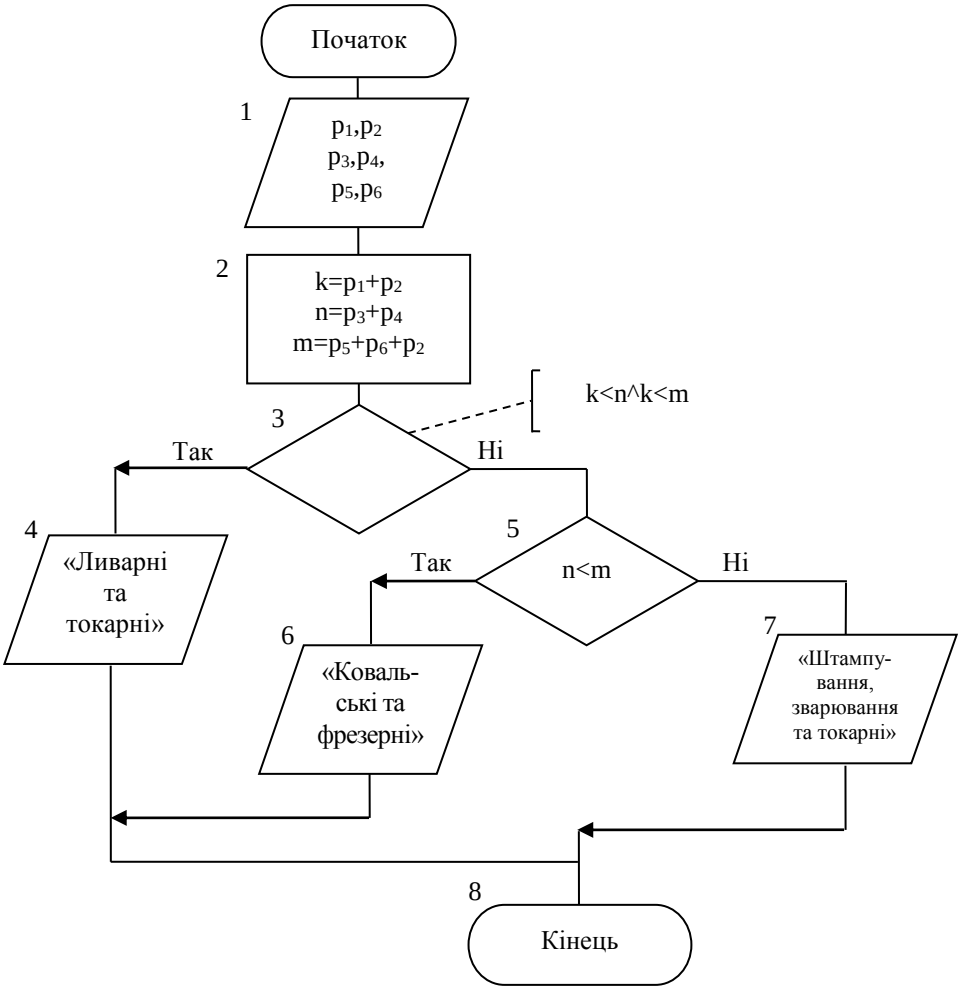
- ливарні роботи, а потім токарні;
- ковальські роботи, а потім фрезерування;
- штампування, а потім послідовно зварювання та токарні роботи.

Вартість цих складових:

ливарні роботи – $p1$; токарні – $p2$; ковальські – $p3$;

фрезерні - $p4$; штампування – $p5$; зварювання – $p6$.

Визначити найбільш ощадливий варіант виготовлення деталі.



C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double p1,p2,p3,p4,p5,p6,k,m,n;
	cout<<"Введите стоимость работ:"<<endl;
	cout<<"p1=";cin>>p1;
	cout<<"p2=";cin>>p2;
	cout<<"p3=";cin>>p3;
	cout<<"p4=";cin>>p4;
	cout<<"p5=";cin>>p5;
	cout<<"p6=";cin>>p6;
 k=p1+p2;
 n=p3+p4;
 m=p5+p6+p2;

if(k<n && k<m) cout<<"Литейные и токарные";
else if (n<m) cout<<"Кузнечные и фрезерные";
else cout<<"Штамповка, сварка и токарные";

	system("pause");
	return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim P1, P2, P3, P4, P5, P6, K, N, M As Single
P1 = CSng(Text1)
P2 = CSng(Text2)
P3 = CSng(Text3)
P4 = CSng(Text4)
```

```
P5 = CSng(Text5)
P6 = CSng(Text6)
K = P1 + P2
N = P3 + P4
M = P5 + P6 + P2
If K < N And K < M Then
Label2 = "Ливарні і токарні"
ElseIf N < M Then
Label2 = "Ковальські та фрезерні"
Else
Label2 = "Штампування, зварювання і токарні"
End If
End Sub
```

TP

```
Program p4;
Var
p1,p2, p3, p4, p5, p6: real;
k, n, m: real;
Begin
Writeln (‘Введіть p1, p2, p3, p4, p5, p6’);
Readln (p1, p2, p3, p4, p5, p6);
k:=p1+p2;
n:=p3+p4;
m=p5+p6+p2;
If (k<n) and (k<m) then
writeln (‘Ливарні та токарні’)
else If (n<m) then writeln (‘Ковальські та фрезерні’) else writeln (‘Штампування, зварювання та токарні’);
End.
```

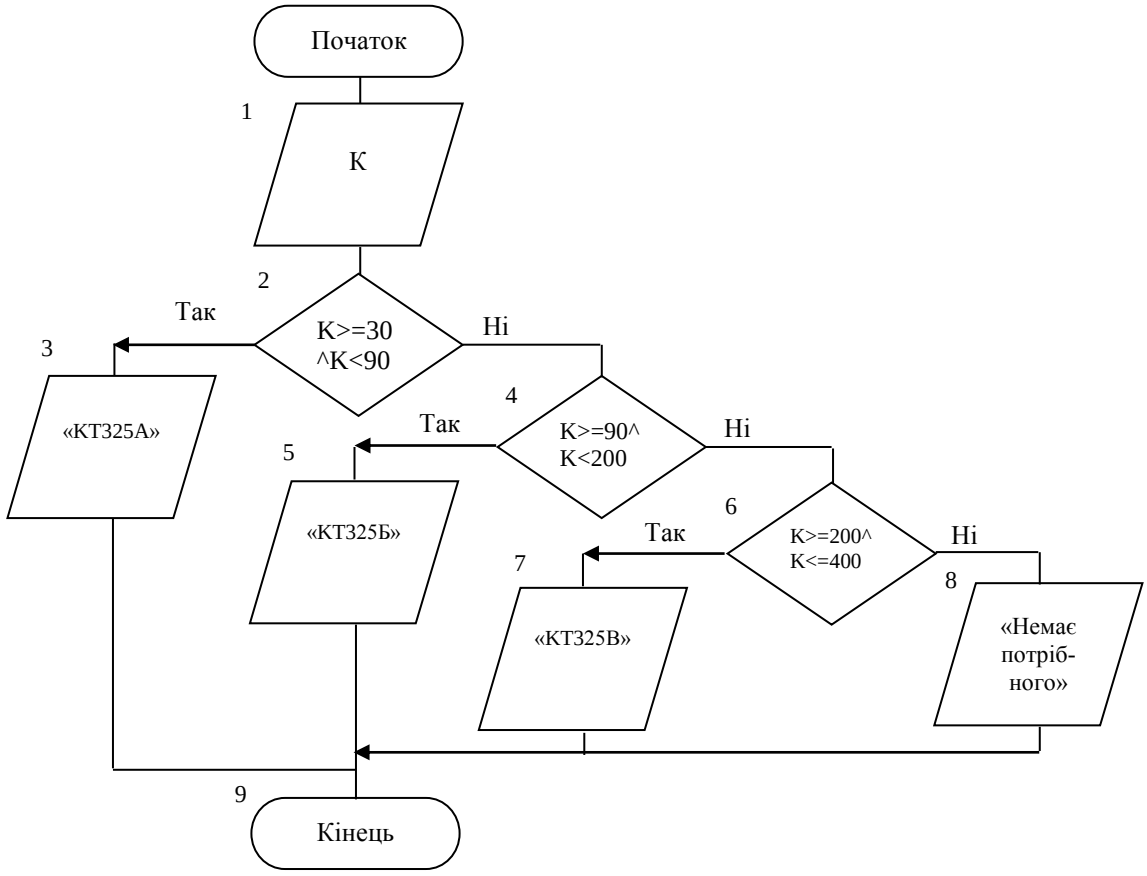
4.1.3 Попадання в заданий інтервал

Задача 5.

Транзистор серії КТ325 має наступні параметри:

Номер	Тип транзистора	Коефіцієнт підсилення	Гранична частота, МГц
1	КТ325А	30-90	0-1
2	КТ325Б	90-200	0-5
3	КТ325В	200-400	0-10

За заданим значенням коефіцієнта підсилення визначити тип транзистора. У випадку можливості вибору двох транзисторів віддати перевагу високочастотному.



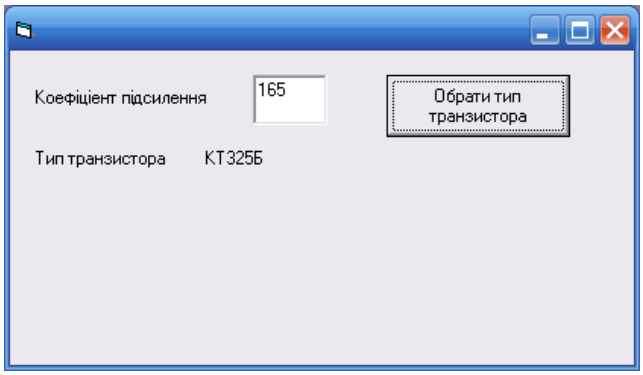
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
```

```
        double K;
        cout<<"Введите коэффициента усиления :"<<endl;
        cin>>K;
if(K>=30 && K<90) cout<<"КТ325А";
if (K>=90 && K<200) cout<<"КТ325Б";
if(K>=200 && K<=4000) cout<<"КТ325В";
else cout<<"Нет подходящего";
        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim K As Single
K = CSng(text1)
If K >= 30 And K < 90 Then
Label2 = "КТ325А"
ElseIf K >= 90 And K < 200 Then
Label2 = "КТ325Б"
ElseIf K >= 200 And K <= 400 Then
Label2 = "КТ325В"
Else
Label2 = "Немає потрібного"
End If
End Sub
```



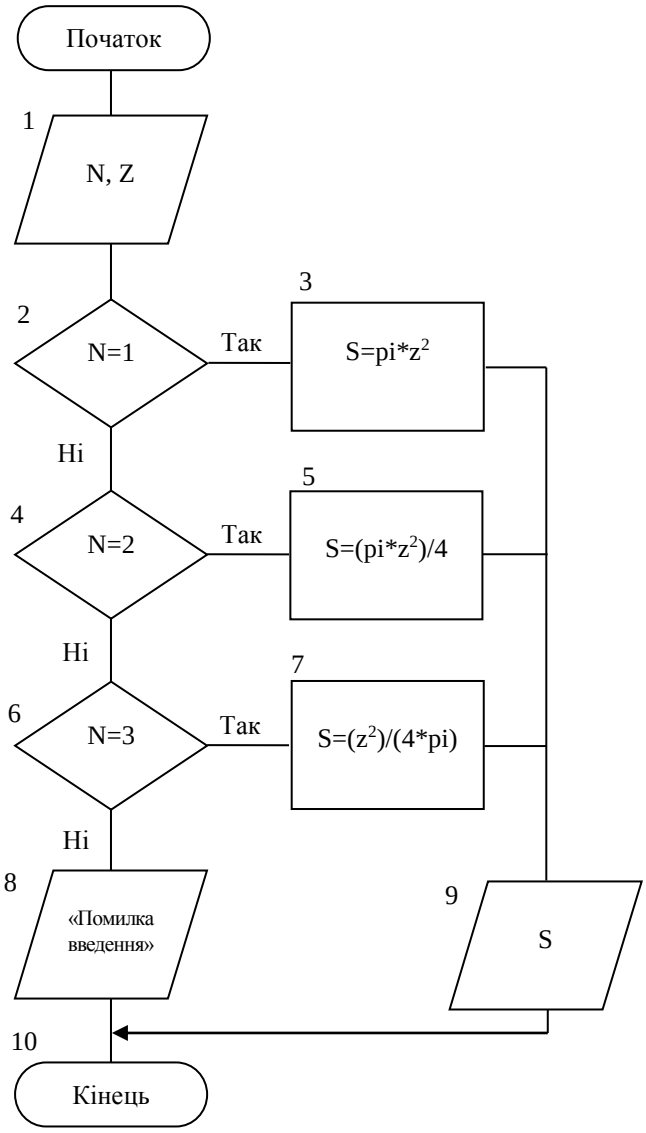
TP

```
Program p5;
Var
k: real;
Begin
Writeln (‘введіть k’);
Readln (k);
If (k>=30) and (k<90) then writeln (‘КТ325А’)
```

```
else
    If (k>=90) and (k<200) then writeln (‘КТ325Б’) else
        If (k>=200) and (k<=400) then writeln (‘КТ325В’) else
writeln (‘Немає підходящого’);
End.
```

Задача 6.

Елементами круга є радіус (перший елемент), діаметр (другий елемент) і довжина кола (третій елемент). Обчислити за заданим номером елемента і його значенням площу круга.



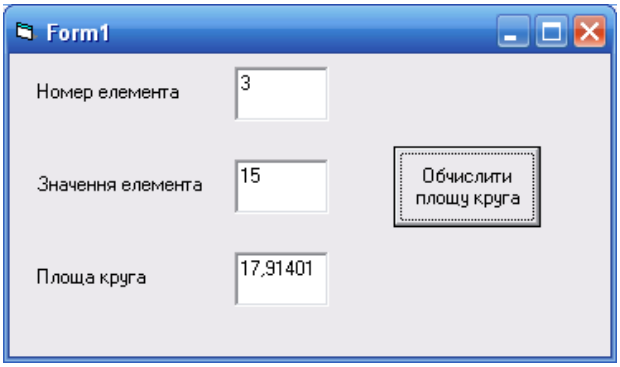
C++

```
#define _USE_MATH_DEFINES
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
```

```
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double z,S;
	int N;
	cout<<"Введите номер элемента :"<<endl;
	cin>>N;
	cout<<"Введите значение элемента :"<<endl;
	cin>>z;

	switch(N){
		case 1: {
			S=M_PI*z*z;
			break;
				}
		case 2: {
			S=(M_PI*z*z)/4;
			break;
				}
      case 3: {
			S=(z*z)/(4*M_PI);
			break;
				}
		  default: {
			cout<<"Ошибка ввода";
			break;}
	}
	cout<<"S="<<S<<endl;

	system("pause");
	return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim N As Integer, Z, S As Single
N = CInt(Text1)
Z = CSng(Text2)
If N = 1 Then
S = 3.14 * Z ^ 2: Text3 = CStr(S)
ElseIf N = 2 Then
S = (3.14 * Z ^ 2) / 4: Text3 = CStr(S)
ElseIf N = 3 Then
S = Z ^ 2 / 4 / 3.14: Text3 = CStr(S)
Else
Text3 = "Помилка введення"
End If
End Sub
```



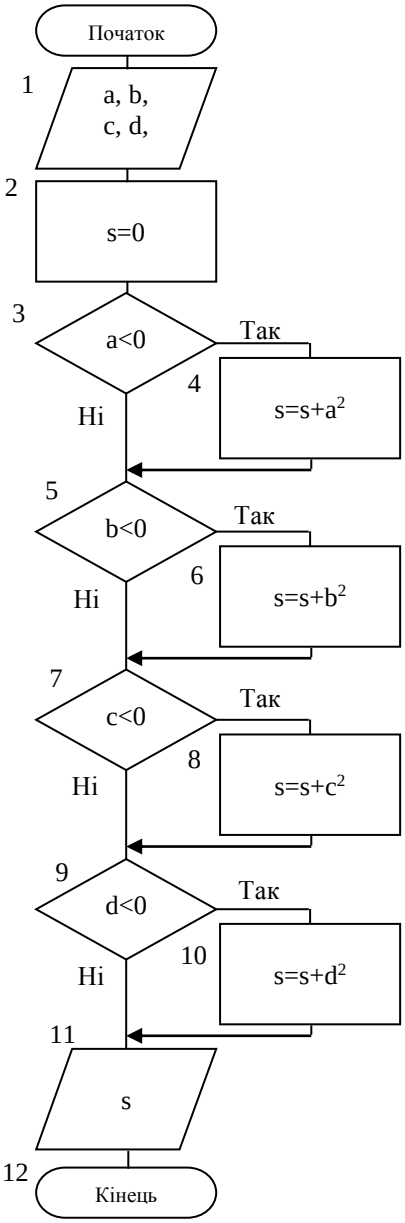
TP

```
Program p6;
Var
N, z, s: real;
Begin
Writeln (‘введіть n, z’);
Readln (n, z);
If n=1 then s:=pi*sqr(z);
If n=2 then s:=(pi*sqr(z))/4;
If n=3 then s:= sqr(z)/(4*pi);
writeln (‘s=’, s) else writeln (‘помилка вводу’);
End.
```

4.1.4 Підрахунок кількостей

Задача 7.

Обчислити суму квадратів від'ємних елементів із заданої послідовності 4 чисел.



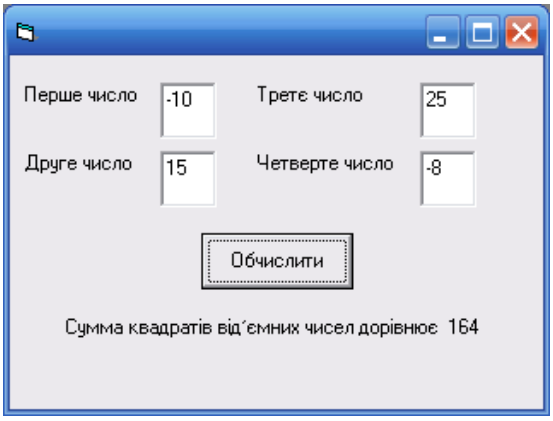
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double a,b,c,d,s;
```

```
        cout<<"Введите 4 числа :"<<endl;
        cin>>a>>b>>c>>d;

s=0;
if(a<0) s+=a*a;
if(b<0) s+=b*b;
if(c<0) s+=c*c;
if(d<0) s+=d*d;

        cout<<"s="<<s<<endl;

        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim A, B, C, D, S As Single
A = CSng(Text1)
B = CSng(Text2)
C = CSng(Text3)
D = CSng(Text4)
S = 0
If A < 0 Then S = S + A ^ 2
If B < 0 Then S = S + B ^ 2
If C < 0 Then S = S + C ^ 2
If D < 0 Then S = S + D ^ 2
Label5 = Label5 + CStr(S)
End Sub
```



TP

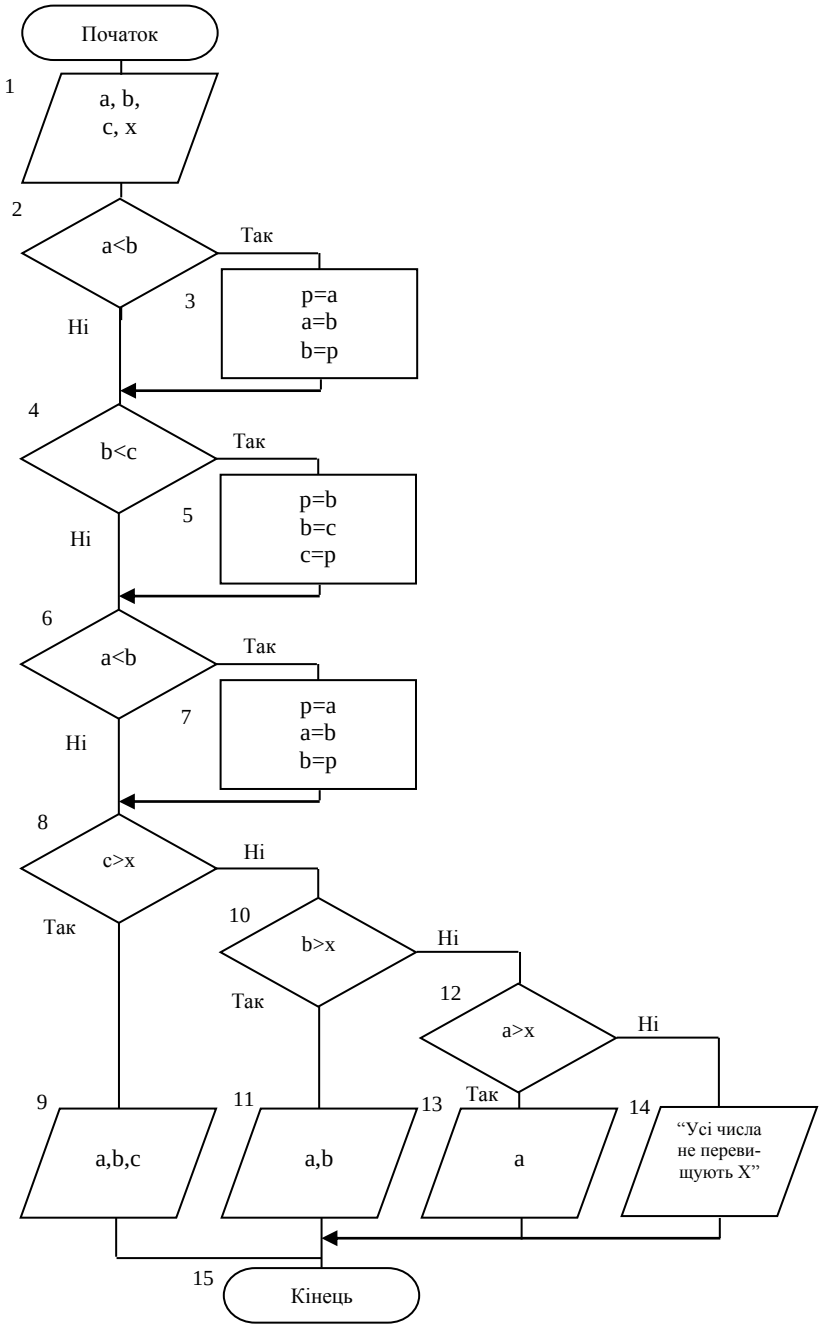
```
Program p7;
Var
a, b, c, d, s: real;
Begin
```

```
Writeln (‘введіть a, b, c, d’);
Readln (a, b, c, d);
S:=0;
If a<0 then s:=s +sqr(a) else
If b<0 then s:=s +sqr(b) else
If c<0 then s:=s +sqr(c) else
If d<0 then s:=s +sqr(d) else
writeln (‘s=’,s);
End.
```

4.1.5 Сортировка за значенням

Задача 8.

Розташувати в порядку зменшення ті з трьох заданих чисел, які більші заданої константи.



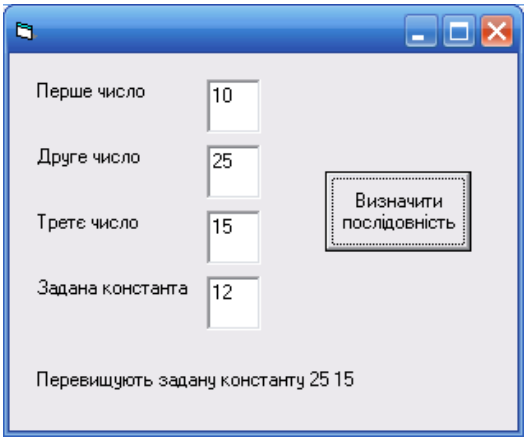
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
```

```
   SetConsoleOutputCP(1251);
        double a,b,c,x,p;
        cout<<"Введите 3 числа :"<<endl;
        cin>>a>>b>>c;
   cout<<"Введите константу X:"<<endl;
        cin>>x;

if(a<b)
{
        p=a;
        a=b;
        b=p;
}
if(b<c)
{
        p=b;
        b=c;
        c=p;
}
if(a<b)
{
        p=a;
        a=b;
        b=p;
}

if(c>x) cout<<a<<" "<<b<<" "<<c;
else if(b>x) cout<<a<<" "<<b;
else if (a>x) cout<<a;
else cout<<"Все числа не превышают X";

        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
a = CSng(Text1)
b = CSng(Text2)
c = CSng(Text3)
x = CSng(Text4)
If a < b Then p = a: a = b: b = p
If b < c Then p = b: b = c: c = p
If a < b Then p = a: a = b: b = p
If c > x Then
Label5 = "Перевищують задану константу" + " "
+ CStr(a) + " " + CStr(b) + " " + CStr(c)
ElseIf b > x Then
Label5 = "Перевищують задану константу" + " "
+ CStr(a) + " " + CStr(b)
ElseIf a > x Then
Label5 = "Перевищують задану константу" + " "
+ CStr(a)
Else
Label5 = "Всі числа не перевищують задану
константу"
End If
End Sub
```



TP

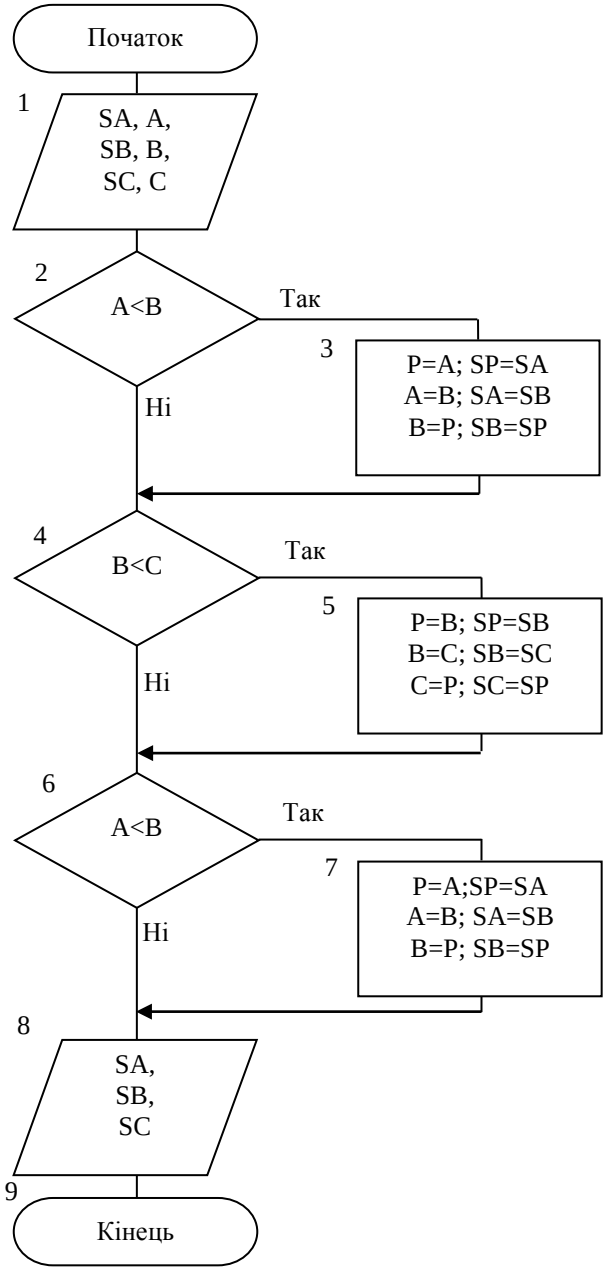
```
Program p8;
Var
a, b, c, x, p: real;
Begin
Writeln (‘введіть a, b, c, x’);
Readln (a, b, c, x);
If a<b then begin p:=a; a:=b; b:=p; end else
If b<c then begin p:=b; b:=c; c:=p; end else
If a<b then begin p:=a; a:=b; b:=p; end else
If c>x then Writeln (‘a, b, c’ ) else
If b>x then Writeln (‘a, b’ ) else
```

```
If a>x then Writeln (‘a’ ) else writeln (‘всі числа не перевищують x’);
End.
```

4.1.6 Сортировка за змінною

Задача 9.

Три спортсмена під час змагань набрали нерівні кількості балів. Покажіть, як розташуються прізвища спортсменів у підсумковій таблиці змагань, якщо перемагає той, хто набрав більше балів.



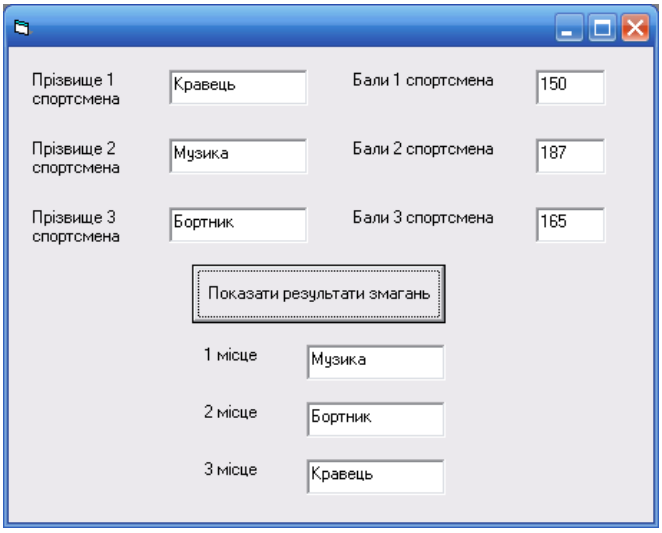
C++

```
#include <string>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double A,B,C,P;
	string SA,SB,SC,SP;
	cout<<"Введите фамилии спортсменов и кол-во набранных баллов :"<<endl;
	cout<<"Фамилия: "; cin>>SA;
	cout<<"Балл: ";cin>>A;
	cout<<"Фамилия: "; cin>>SB;
	cout<<"Балл: ";cin>>B;
	cout<<"Фамилия: "; cin>>SC;
	cout<<"Балл: ";cin>>C;

if(A>B)
{
	P=A; SP=SA;
	A=B; SA=SB;
	B=P; SB=SP;
}
if(B>C)
{
	P=B; SP=SB;
	B=C; SB=SC;
	C=P; SC=SP;
}
if(A>B)
{
	P=A; SP=SA;
	A=B; SA=SB;
	B=P; SB=SP;
```

```
}

cout<<SA<<" "<<SB<<" "<<SC<<endl;
        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim a, b, c, p As Single, sa, sb, sc, sp As String
sa = Text1
sb = Text2
sc = Text3
a = CSng(Text4)
b = CSng(Text5)
c = CSng(Text6)
If a < b Then p = a: a = b: b = p: sp = sa: sa = sb: sb = sp
If b < c Then p = b: b = c: c = p: sp = sb: sb = sc: sc = sp
If a < b Then p = a: a = b: b = p: sp = sa: sa = sb: sb = sp
Text7 = sa
Text8 = sb
Text9 = sc
End Sub
```



TP

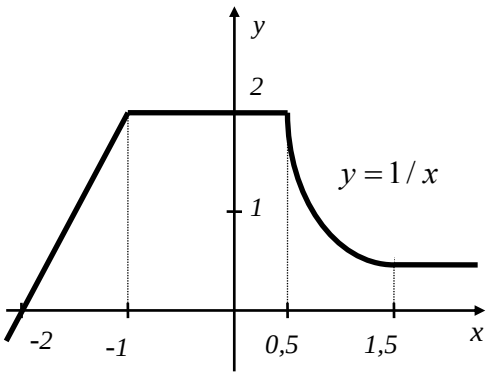
```
Program p9;
Var
sa, sb, sc, a, b, c: real;
sp, p: real;
Begin
Writeln (‘Введіть sa, sb, sc, a, b, c’);
Readln (sa, sb, sc, a, b, c);
```

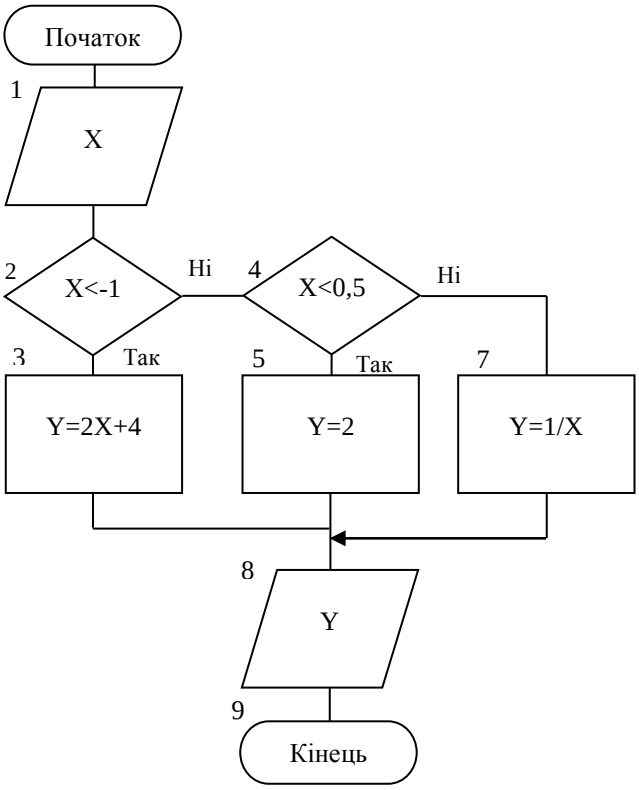
```
If (a>b) then begin
p:=a; sp:=sa;
a:=b; sa:=sb;
b:=p; sb:=sp;
end;
If (b>c) then begin
p:=b; sp:=sb;
b=c; sb=sc;
c=p; sc=sp;
end;
If (a>b) then begin
p:=a; sp:=sa;
a:=B; sa:=sb;
b:=p; sb:=sp;
end;
writeln (sa, sb, sc);
End.
```

4.1.7 Геометрична задача з графіком

Задача 10.

Скласти алгоритм розрахунку значення функції, заданої графіком, для будь-якого заданого значення аргумента.



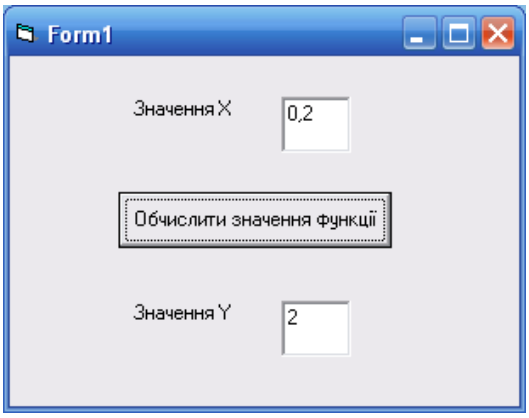


C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double X,Y;
	cout<<"Введите значение аргумента :"<<endl;
	cin>>X;
	if(X<-1) Y=2*X+4;
else if(X<=0.5) Y=2;
else Y=1/X;
cout<<"Y= "<<Y<<endl;
	system("pause");
	return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim X, Y As Single
X = CSng(Text1)
If X < -1 Then
Y = 2 * X + 4
ElseIf X <= 0.5 Then
Y = 2
Else
Y = 1 / X
End If
Text2 = CStr(Y)
End Sub
```



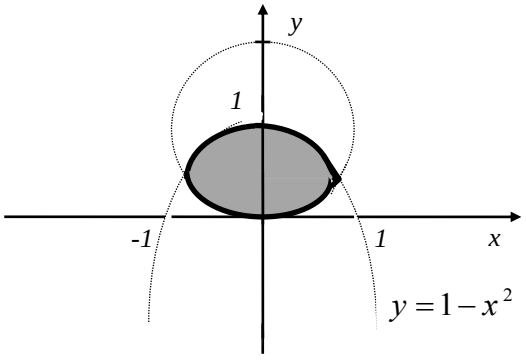
TP

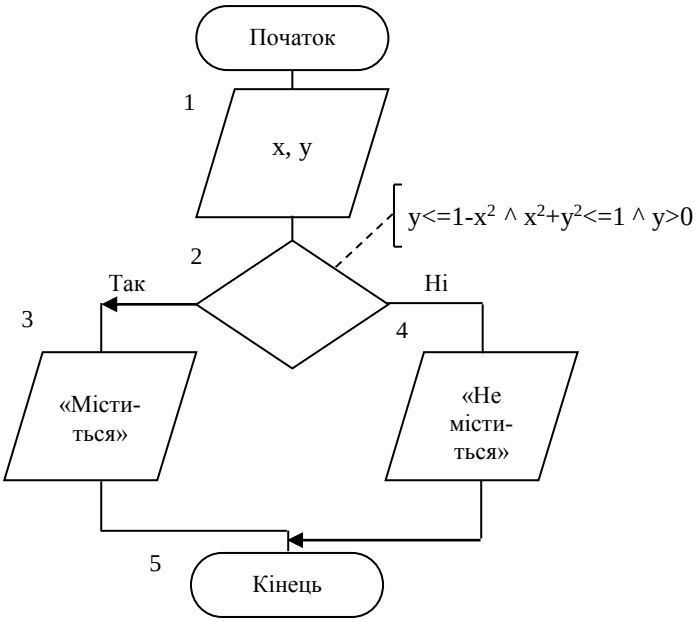
```
Program p10;
Var
x, y: real;
Begin
Writeln (‘введіть x’);
Readln (x);
If begin x<(-1) then Y:=2*x+4 else begin if x<=0.5 then Y:=2 else y:=1/x;
writeln (‘y=’,y:6:2); end; end;
End.
```

4.1.8 Геометрична задача з фігурою

Задача 11.

Перевірити, чи міститься точка з заданими координатами усередині області, що наведена на рисунку.



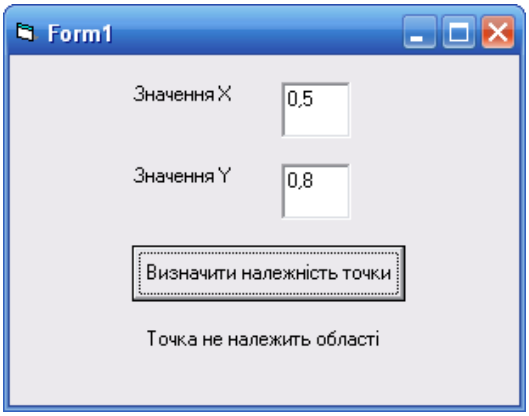


C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double x,y;
	cout<<"Введите координаты :"<<endl;
	cin>>x>>y;

if(y<=(1-x*x) && (x*x+y*y)<=1 && y>0) cout<<"Принадлежит";
else cout<<"Не принадлежит";

	system("pause");
	return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim X, Y As Single
X = CSng(Text1)
Y = CSng(Text2)
If Y <= 1 - X ^ 2 And X ^ 2 + Y ^ 2 <= 1 And Y > 0 Then
Label3 = "Точка належить області"
Else
Label3 = "Точка не належить області"
End If
End Sub
```



TP

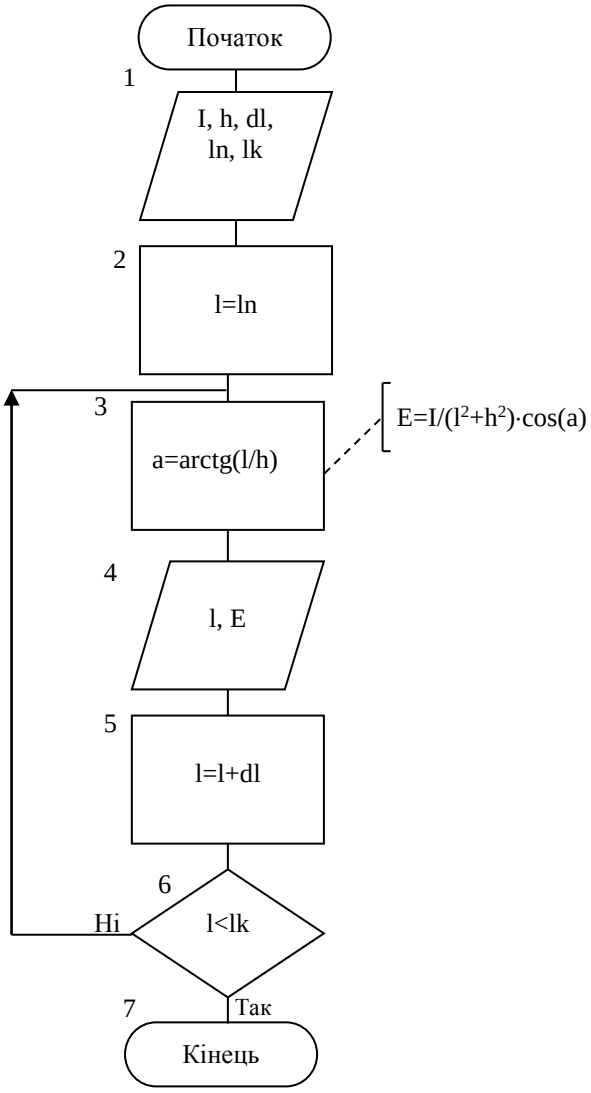
```
Program p11;
Var
x, y: real;
Begin
Writeln (‘введіть x, y’);
Readln (x, y);
If (y<=1-sqr(x)) and (sqr(x)+sqr(y)<=1) and (y>0) then
writeln (‘Належить’) else writeln (‘Не належить’);
End.
```

4.2 Циклічні алгоритми

4.2.1 Прості цикли

Задача 12.

Кімната освітлюється лампою, яка розміщена на відстані H від поверхні стола. Вважаючи лампу точечним джерелом з силою світла I, визначити освітленість $E = \frac{I}{r^2} \cdot \cos\alpha$ поверхні стола на відстані l = 10, 20, ... , 200 см від точки проекції лампи на стіл.



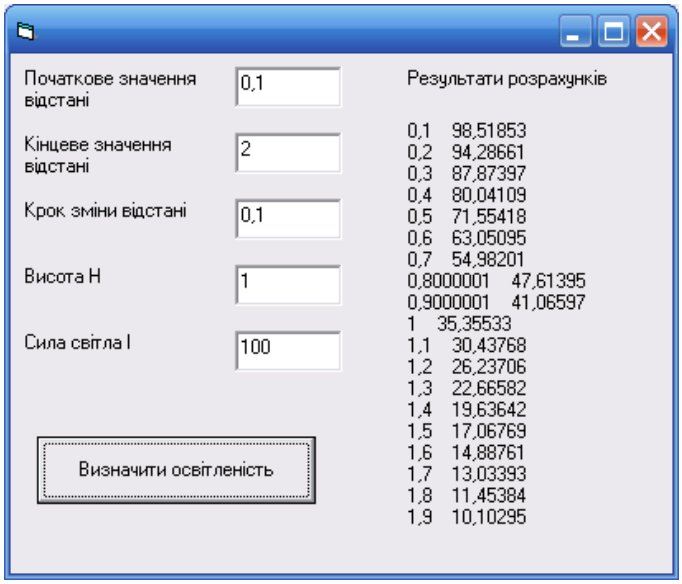
C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double H,I,E,a;
	int dl,ln,lk,l;
	cout<<"H = "; cin>>H;
```

```
        cout<<"I = "; cin>>I;
        cout<<"Расстояние от точки проекции"<<endl;
        cout<<"Начало интервала = "; cin>>ln;
        cout<<"Конец интервала = "; cin>>lk;
        cout<<"Шаг = "; cin>>dl;

l=ln;
do
{
        a=atan(1/H);
        E=I/(l*l + H*H)*cos(a);
        cout<<l<<" "<<E<<endl;
        l+=dl;
}
while(l<lk);

        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim LN, LK, DL, L, H, I, A, E As Single
LN = CSng(Text1)
LK = CSng(Text2)
DL = CSng(Text3)
H = CSng(Text4)
I = CSng(Text5)
For L = LN To LK Step DL
A = Atn(L / H)
E = I / (L ^ 2 + H ^ 2) * Cos(A)
Label7 = Label7 & CStr(L) & "    " & CStr(E) & vbCrLf
Next L
End Sub
```



TP

```
Program p12;
Var
I, dl, ln, lk, h, E, a, l: real;
Begin
Writeln (‘Введіть I, dl, ln, lk, h’);
Readln (I, dl, ln, lk, h);
l:=ln;
Repeat
a:=arctan(l/h);
E:=I/(sqr(l)+sqr(h))*cos(a);
Writeln (‘l=’,l:6:2,’E=’,E:6:2, a:6:2);
l:=l+dl;
Until l>lk;
End.
```

Простий цикл

Задача 13.

Лінією електропередач постійної напруги передається електроенергія потужністю

$$P = \frac{U^2 \cdot R_Н}{(R_Н + R_Л)^2}$$

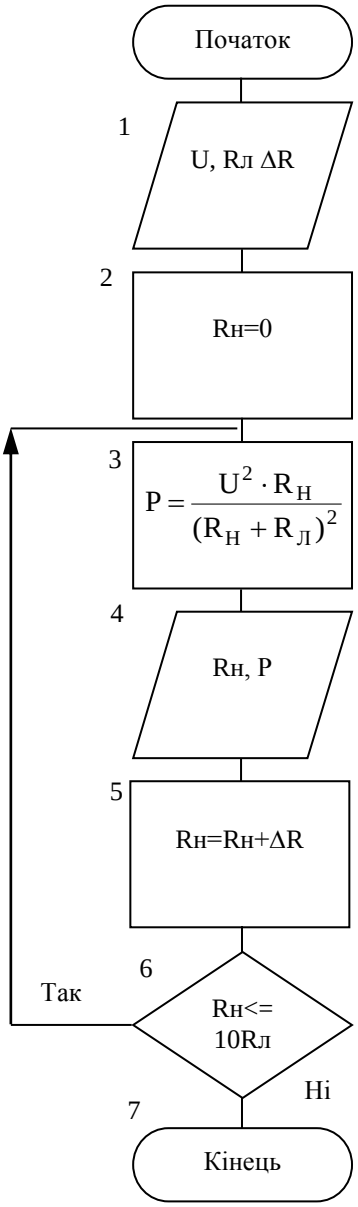
де U – напруга на початку лінії;

$R_Н$ – опір навантаження;

$R_Л$ – опір лінії.

Розрахувати таблицю залежності $P = f(R_Н)$

змінюючи $R_Н$ від 0 до $10R_Л$ з кроком ΔR.



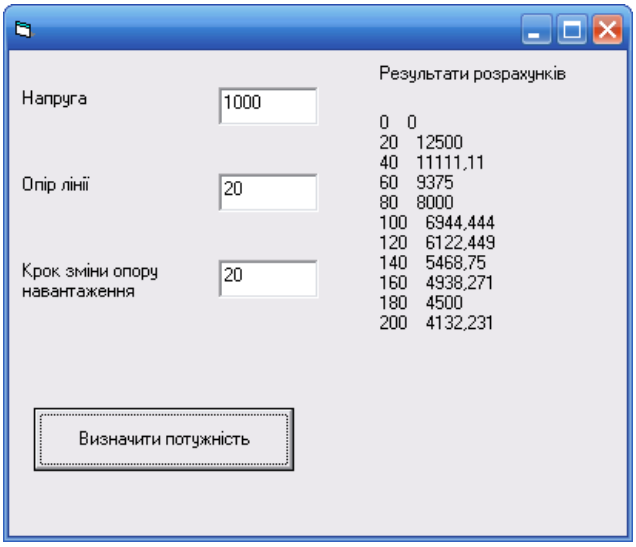
C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        double U,Rl,dR,Rn,P;
        cout<<"U = "; cin>>U;
        cout<<"Rl = "; cin>>Rl;
        cout<<"dR = "; cin>>dR;
```

```
Rn=0;
do
{
        P=(U*U*Rn)/pow(Rn+Rl,2);
        cout<<Rn<<" "<<P<<endl;
        Rn+=dR;
}
while(Rn<=10*Rl);

        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim U, RN, RL, DR, P As Single
U = CSng(Text1)
RL = CSng(Text2)
DR = CSng(Text3)
For RN = 0 To 10 * RL Step DR
P = U ^ 2 * RN / (RN + RL) ^ 2
Label4 = Label4 & CStr(RN) & "    " & CStr(P) & vbCrLf
Next RN
End Sub
```



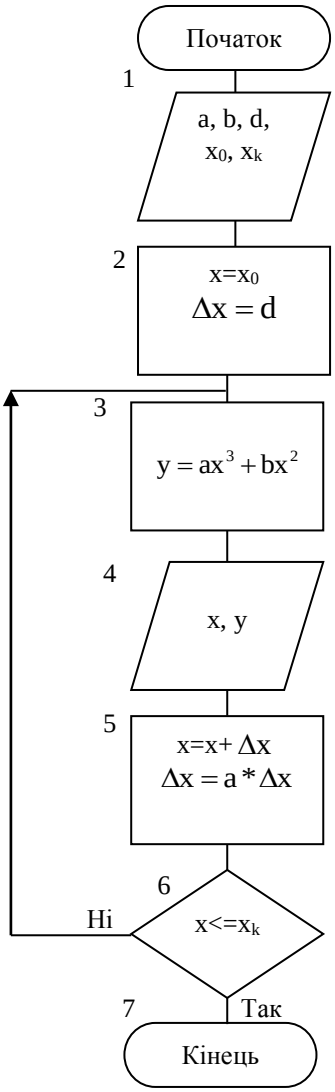
TP

```
Program p13;
Var
u, dR, Rl, P: real;
Rn: integer;
Begin
Writeln (‘Введіть u, Rl, dR’);
```

```
Readln (u, Rl, dR);
Rn:=0;
Repeat
P:=(sqr(u)*Rn)/sqr(Rn+Rl);
Writeln (‘Rn=’, Rn:6:2,’P=’, p:6:2);
Rn:=Rn+dR;
Until Rn>10*Rl;
End.
```

Задача 14.

Розрахувати таблицю значень функції $y = ax^3 + bx^2$ для значень $x_0 \le x \le x_k$ з кроком, що змінюється за законом $\Delta x_{i+1} = a*\Delta*_i$, де $\Delta x_0 = d < x_k$.

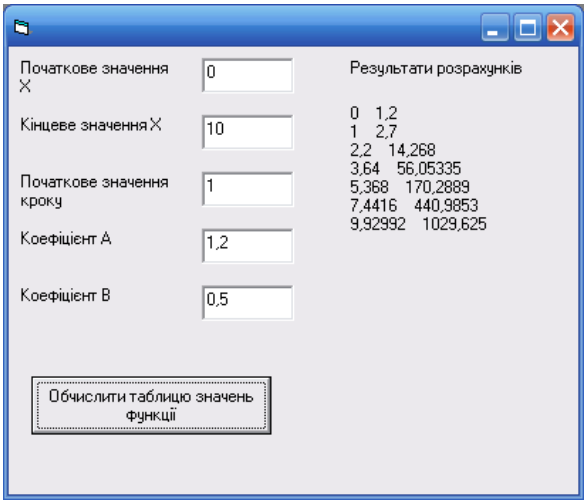


C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	double a,b,d,x0,xk,y,x,dx;
	cout<<"a = "; cin>>a;
	cout<<"b = "; cin>>b;
	cout<<"d = "; cin>>d;
	cout<<"x0 = "; cin>>x0;
	cout<<"xk = "; cin>>xk;

x=x0;
dx=d;
do
{
	y=a*pow(x,3)+b*pow(x,2);
	cout<<x<<" "<<y<<endl;
	x+=dx;
	dx*=a;
}
while(x<=xk);

	system("pause");
	return 0;
}
```

VB

```
Option Explicit
Private Sub Command1_Click()
Dim X, X0, XK, D, DX, A, B, Y As Single
X0 = CSng(Text1)
XK = CSng(Text2)
D = CSng(Text3)
A = CSng(Text4)
B = CSng(Text5)
DX = D
X = X0
Do
Y = A + X ^ 3 + B * X ^ 2
Label7 = Label7 & CStr(X) & "    " & CStr(Y) &
vbCrLf
X = X + DX
DX = A * DX
Loop Until X > XK
End Sub
```



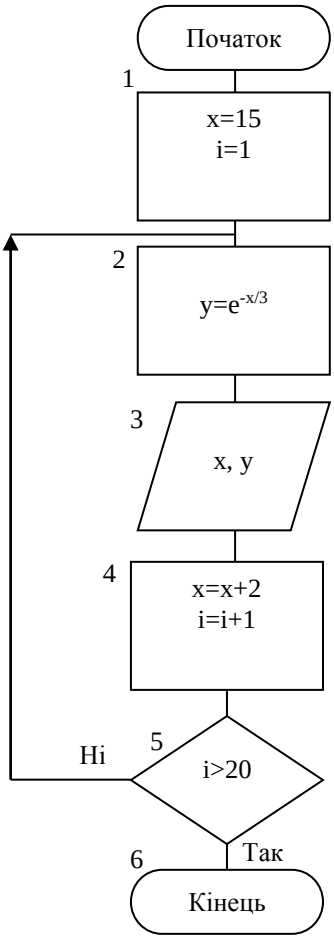
TP

```
Program p14;
Var
a, b, d, x0, xk, y, x, dx: real;
Begin
Writeln ('Введіть a, b, d, x0, xk');
Readln (a, b, d, x0, xk);
x:=x0;
dx:=d;
While x<=xk do
Begin
y:=(a*sqr(x)*x)+(b*sqr(x));
Writeln ('x=', x:6:2,'y=',y:6:2);
x:=x+dx;
dx:=a*dx;
end;
End.
```

Постий цикл з лічильником

Задача 15.

Обчислити значення функції $y = e^{-x/3}$ для 20 непарних чисел натурального ряду, починаючи з 15.



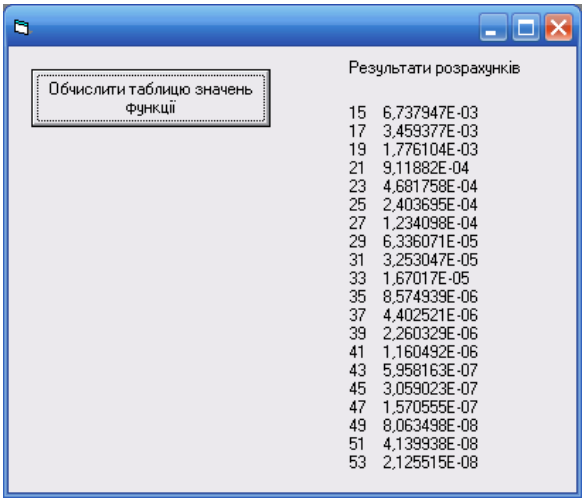
C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int x,i;
        double y;

x=15;
```

```
i=1;
do
{
        y=exp(-(double)x/3);
        cout<<x<<" "<<y<<endl;
        x+=2;
        i++;
}
while(i<20);

        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim X, Y As Single, I As Integer
X = 15
For I = 1 To 20
Y = Exp(-X / 3)
Label7 = Label7 & CStr(X) & "    " & CStr(Y) & vbCrLf
X = X + 2
Next I
End Sub
```



TP

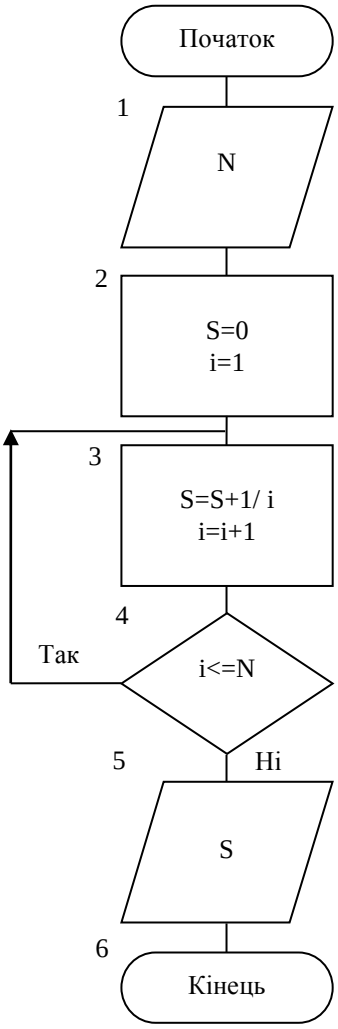
```
Program p15;
Var
n, i: integer; y, x: real;
Begin
x:=15;
i:=1;
Repeat
```

```
Y:=exp(-x/3);
Writeln (‘x=’, x:6:2,’y=’,y:6:2);
x:=x+2;
I:=i+1;
Until i>20;
End.
```

Простий цикл з накопиченням

Задача 16.

Обчислити суму *N* доданків за формулою $S = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N}$

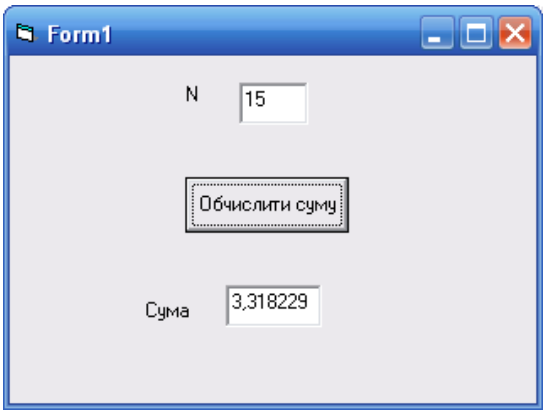


C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,i;
	double S;
   cout<<"N="; cin>>N;
S=0;
i=1;
do
{
	S+=1.0/i;
	i++;
}
while(i<=N);

cout<<"S="<<S;

system("pause");
	return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim N, I As Integer, S As Single
N = CInt(Text1)
S = 0
I = 1
Do
S = S + 1 / I
I = I + 1
```



```
Loop Until I > N
Text2 = CStr(S)
End Sub
```

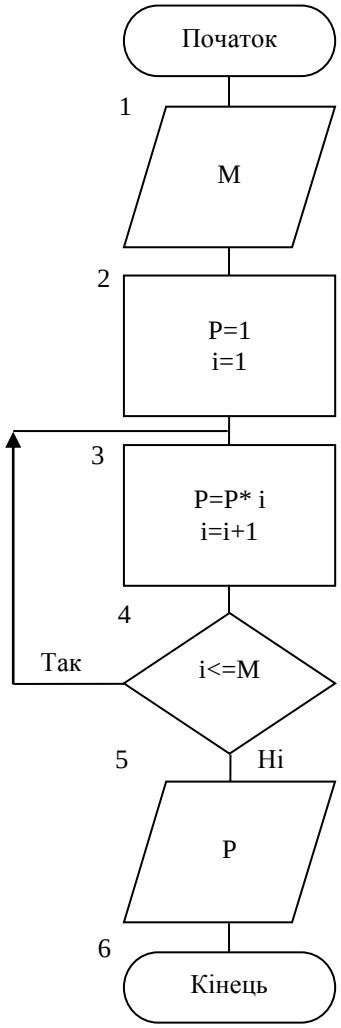
TP

```
Program p16;
Var
n, i: integer; s: real;
Begin
Writeln (‘Введіть масив n’);
Readln (n);
s:=0;
i:=1;
While i<=n do
Begin
s:=s+1/i;
i:=i+1;
End;
Writeln (‘s=’, s:6:2);
End.
```

Задача 17.

Обчислити значення факторіала *M*!

$$M!=1*2*3*4*\ldots..*M$$



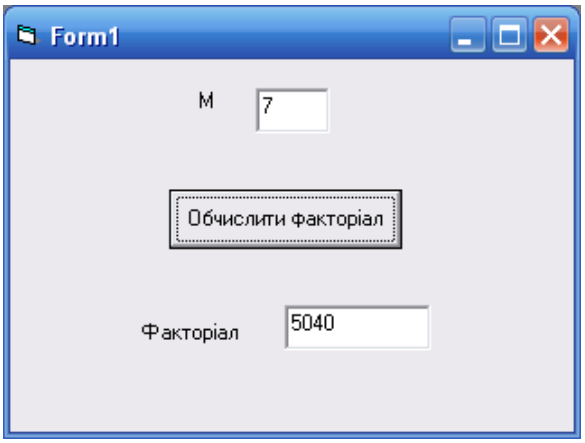
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int M,P,i;
   cout<<"M="; cin>>M;
```

```
P=1;
i=1;
do
{
        P*=i;
        i++;
}
while(i<=M);

cout<<"P="<<P;

system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Private Sub Command1_Click()
Dim M, I As Integer, P As Single
M = CInt(Text1)
P = 1
I = 1
Do
P = P * I
I = I + 1
Loop Until I > M
Text2 = CStr(P)
End Sub
```



TP

```
Program p17;
Var
p, m, i: integer;
Begin
Writeln (‘Введіть m’);
```

```
Readln (m);
p:=1;
i:=1;
While i<=m do
Begin
p:=p*I;
i:=i+1;
End;
Writeln (‘p=’, p);
End.
```

4.2.2 Обробка одновимірних масивів

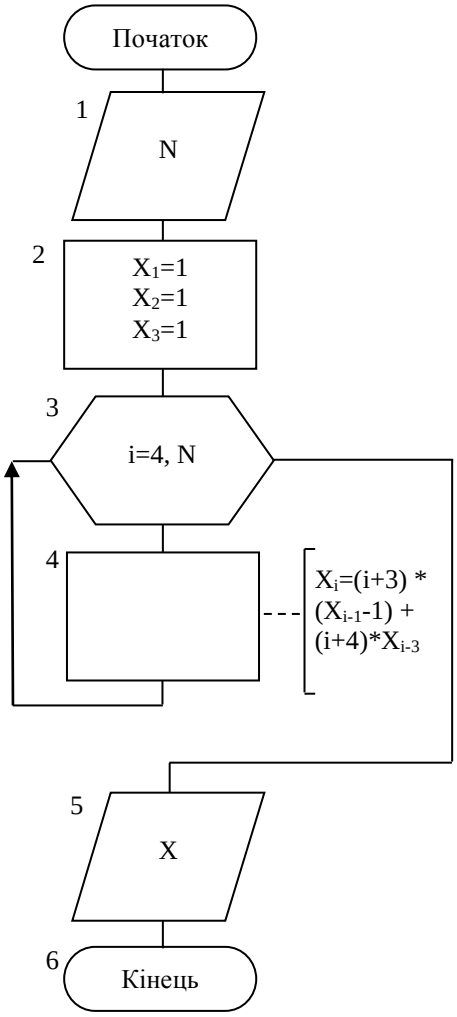
Формування одновимірного масива

Задача 18.

Послідовність $x_1, x_2, ... x_n$ утворена за законом:

$x_1 = x_2 = x_3 = 1; \quad x_i = (i+3)\cdot(x_{i-1}-1) + (i+4)\cdot x_{i-3}, i = 4,5,...n$

Одержати $x_1, x_2, ..., x_n$.

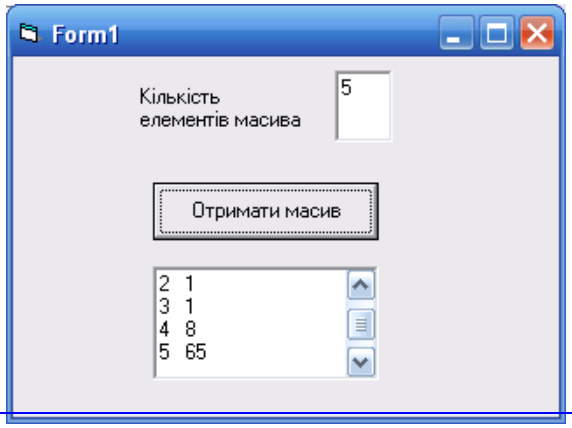


C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int N,i;

   cout<<"N="; cin>>N;
        double *x = new double [N+1];
x[1]=1;
x[2]=1;
x[3]=1;

for(i=4;i<=N;i++)
x[i]=(i+3)*(x[i-1]-1)+(i+4)*x[i-3];

for(i=1;i<=N;i++)
 cout<<"x["<<i<<"]="<<x[i]<<endl;

delete []x;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Private Sub Command1_Click()
Dim X() As Single, I, N As Integer
N = CInt(Text1)
ReDim X(N)
X(1) = 1
X(2) = 1
```



```
X(3) = 1
For I = 4 To N
X(I) = (I + 3) * (X(I - 1) - 1) + (I + 4) * X(I - 3)
Next I
For I = 1 To N
Text2 = Text2 + CStr(I) + "   " + CStr(X(I)) +
vbCrLf
Next I
End Sub
```

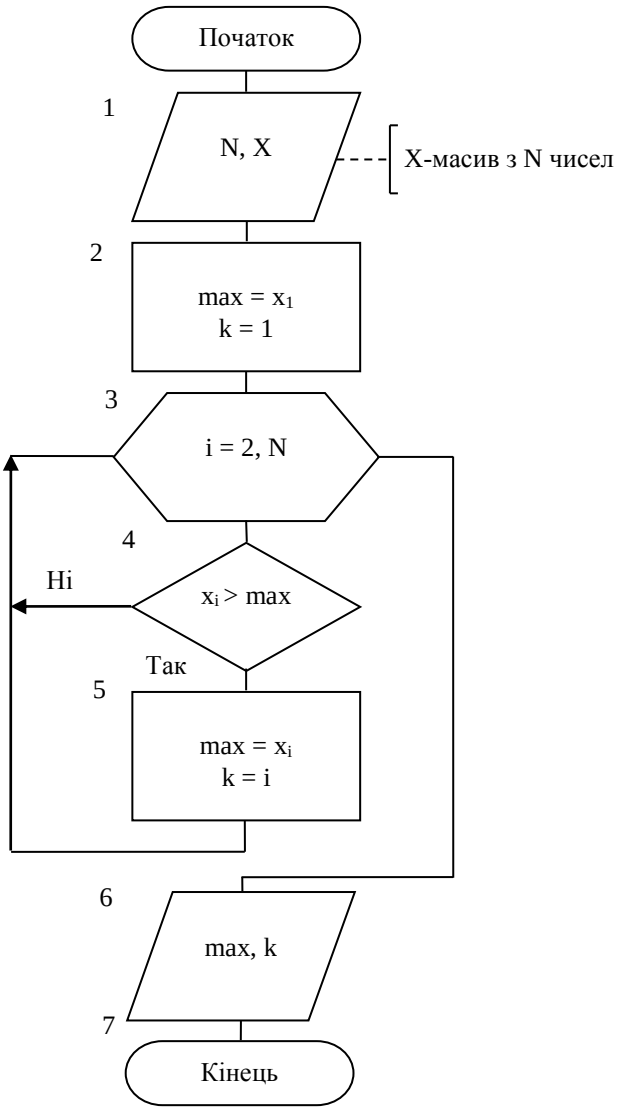
TP

```
Program p18;
Var
X: array [1..100] of real;
n, i: integer;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
x[1]:=1;
x[2]:=1;
x[3]:=1;
For i:=4 to n do
x[i]:=(i+3)*(x[i-1]-1)+(i+4)*(x[i-3]);
For i:=1 to n do
Writeln (‘x[i]=’, x[i]:6:2);
End.
```

Пошук екстремума в масиві

Задача 19.

Знайти максимальний елемент масива ***X*** та його номер.



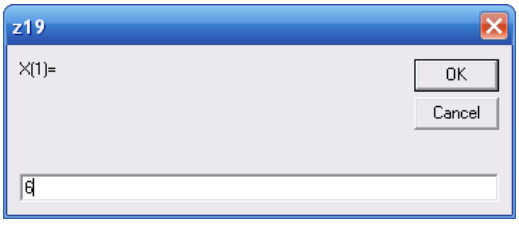
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,i,k;
	double max;
```

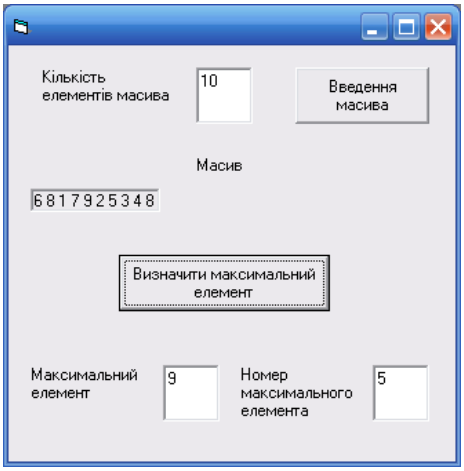
```
   cout<<"N="; cin>>N;
        double *x = new double [N+1];

        for(i=1;i<=N;i++)
        {
                cout<<"x["<<i<<"]=";
                cin>>x[i];
        }
max=x[i];
k=1;
for(i=2;i<=N;i++)
if(x[i]>max)
{
        max=x[i];
        k=i;
}

cout<<"max = "<<max<<" k = "<<k<<endl;

delete []x;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim X() As Single, Max As Single, I, N, Nmax As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim X(N)
Label2 = ""
For I = 1 To N
X(I) = CSng(InputBox("X(" & I & ")="))
Label2 = Label2 + CStr(X(I)) + " "
```



```
Next I
End Sub

Private Sub Command2_Click()
Max = X(1)
Nmax = 1
For I = 1 To N
If X(I) > Max Then Max = X(I): Nmax = I
Next I
Text2 = CStr(Max)
Text3 = CStr(Nmax)
End Sub
```



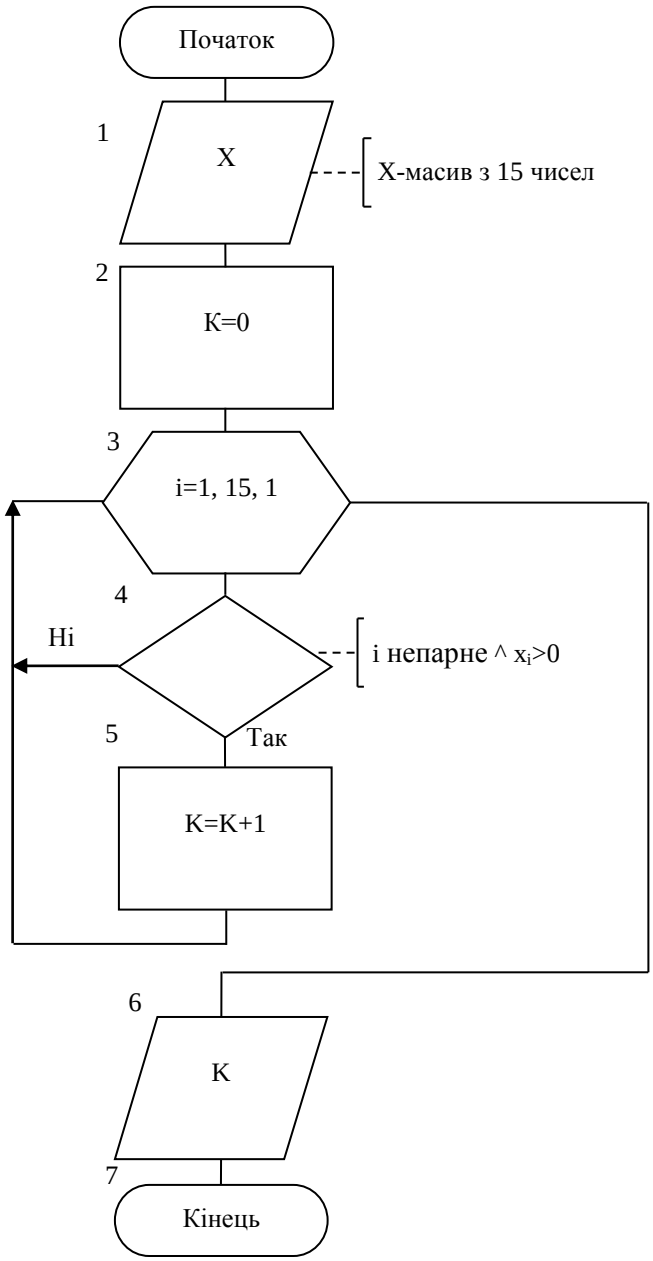
TP

```
Program p19;
Var
X: array [1..100] of real;
n, i, k: integer; max: real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
max:=x[1];
k:=1;
For i:=2 to n do
If x[i]>max then
Begin
max:=x[i];
k:=i;
end;
Writeln (‘max=’, max:6:2, ‘k=’,k); End.
```

Накопичення в масиві

Задача 20.

В одновимірному масиві, що складається з 15 елементів, знайти кількість додатніх елементів, що мають непарні номери.



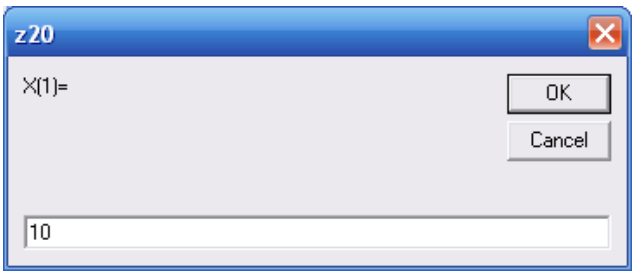
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
```

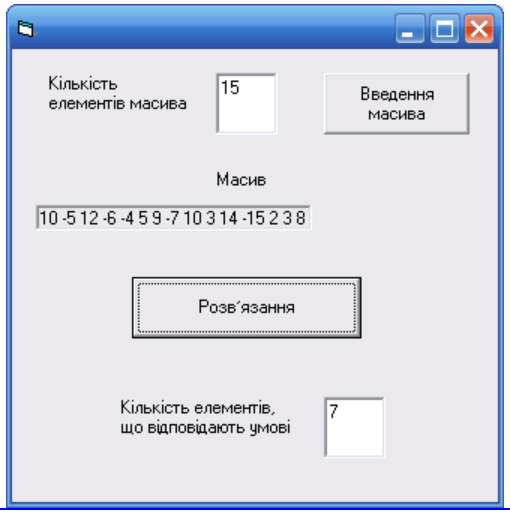
```
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int i,K;
        double x[16];

        for(i=1;i<=15;i++)
        {
                cout<<"x["<<i<<"]=";
                cin>>x[i];
        }

K=0;
for(i=1;i<=15;i++)
if(i%2!=0 && x[i]>0) K++;

cout<<"K = "<<K<<endl;

system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim X() As Single, I, N, K As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim X(N)
Label2 = ""
For I = 1 To N
X(I) = CSng(InputBox("X(" & I & ")="))
Label2 = Label2 + CStr(X(I)) + " "
Next I
End Sub

Private Sub Command2_Click()
```



```
K = 0
For I = 1 To N
If X(I) > 0 And I Mod 2 = 1 Then K = K + 1
Next I
Text2 = CStr(K)
End Sub
```

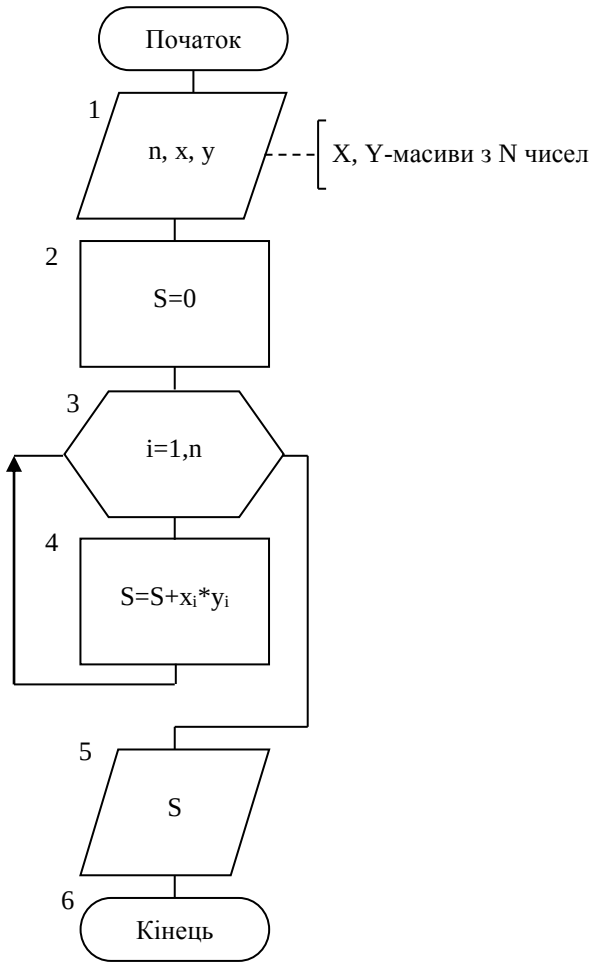


TP

```
Program p20;
Var
X: array [1..15] of real;
K, i: integer;
Begin
Writeln (‘Введіть масив X’);
For i:=1 to 15 do
Readln (x[i]);
K:=0;
For i:=1 to 15 do
Begin
If (I mod 2 =0) and (x[i]>0) then k:=k+1;
end;
Writeln (‘k=’,k);
End.
```

Задача 21.

Задані два вектори своїми компонентами x (x_1, x_2, ..., x_n) та y (y_1, y_2, ..., y_n). Обчислити скалярний добуток заданих векторів як суму добутків однойменних координат.



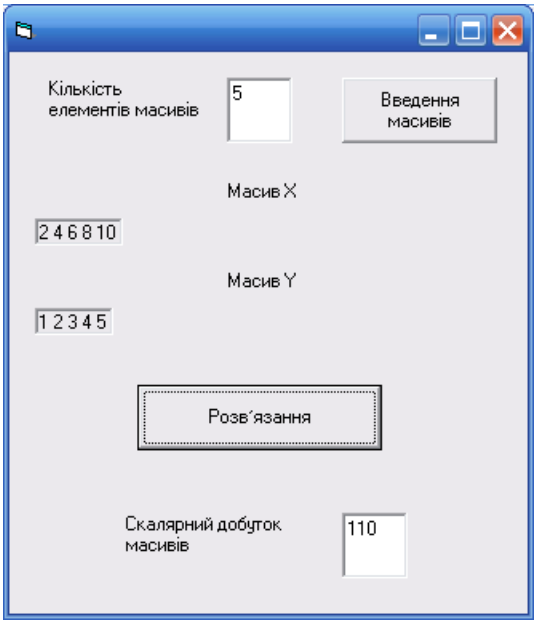
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int n,i,S;
	cout<<"n=";cin>>n;
double *x = new double [n+1];
double *y = new double [n+1];
```

```
for(i=1;i<=n;i++)
        {
                cout<<"x["<<i<<"]=";
                cin>>x[i];
                cout<<"y["<<i<<"]=";
                cin>>y[i];
        }

S=0;
for(i=1;i<=n;i++) S+=x[i]*y[i];

cout<<"S = "<<S<<endl;

delete []x;
delete []y;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim X(), Y(), S As Single, I, N As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim X(N), Y(N)
Label2 = ""
For I = 1 To N
X(I) = CSng(InputBox("X(" & I & ")="))
Label2 = Label2 + CStr(X(I)) + " "
Next I
Label5 = ""
For I = 1 To N
Y(I) = CSng(InputBox("Y(" & I & ")="))
Label5 = Label5 + CStr(Y(I)) + " "
```



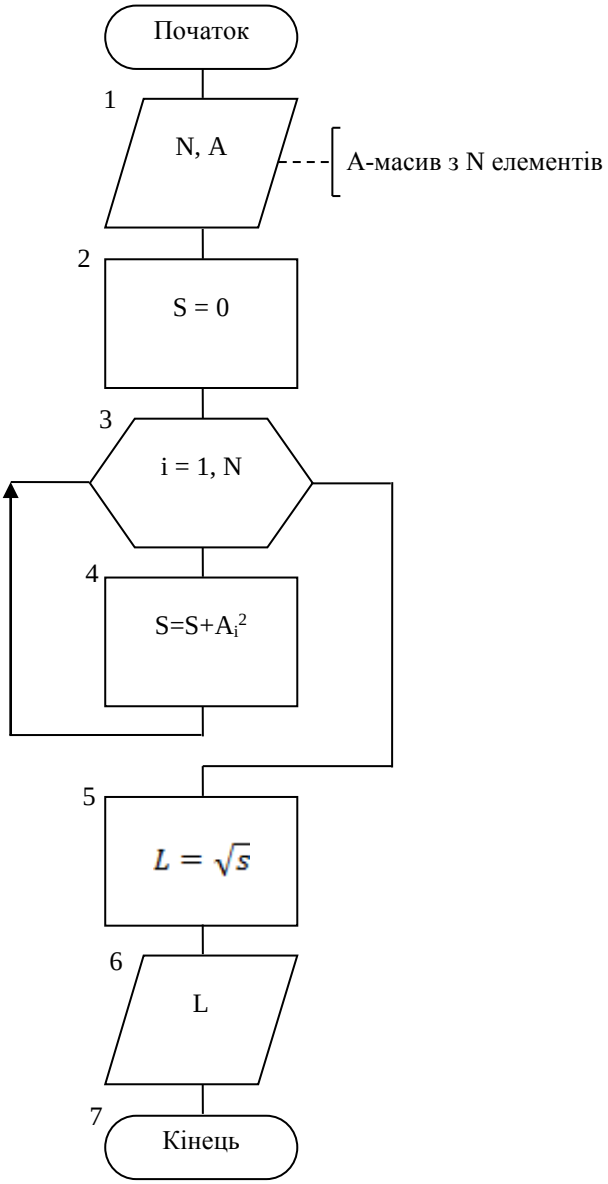
```
Next I
End Sub

Private Sub Command2_Click()
S = 0
For I = 1 To N
S = S + X(I) * Y(I)
Next I
Text2 = CStr(S)
End Sub
```

TP

```
Program p21;
Var
X, Y: array [1..100] of real;
n, i: integer; s:real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
Writeln (‘Введіть масив y’);
For i:=1 to n do
Readln (y[i]);
s:=0;
For i:=1 to n do
s:=s+x[i]*y[i];
Writeln (‘s=’, s:6:2);
End.
```

Задача 22.

Визначити довжину вектора, що складається з *N* елементів.



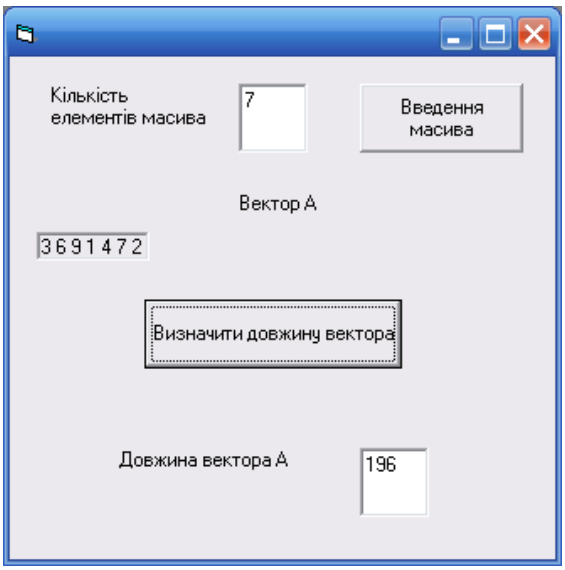
C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
```

```
	int N,i;
	double L,S;
	cout<<"N=";cin>>N;
double *A = new double [N+1];

for(i=1;i<=N;i++)
	{
		cout<<"A["<<i<<"]=";
		cin>>A[i];
	}
S=0;
for(i=1;i<=N;i++) S+=A[i]*A[i];

L=sqrt(S);
cout<<"L = "<<L<<endl;

delete []A;
system("pause");
	return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim A(), S As Single, I, N As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim A(N)
Label2 = ""
For I = 1 To N
A(I) = CSng(InputBox("A(" & I & ")="))
Label2 = Label2 + CStr(A(I)) + " "
Next I
End Sub

Private Sub Command2_Click()
```



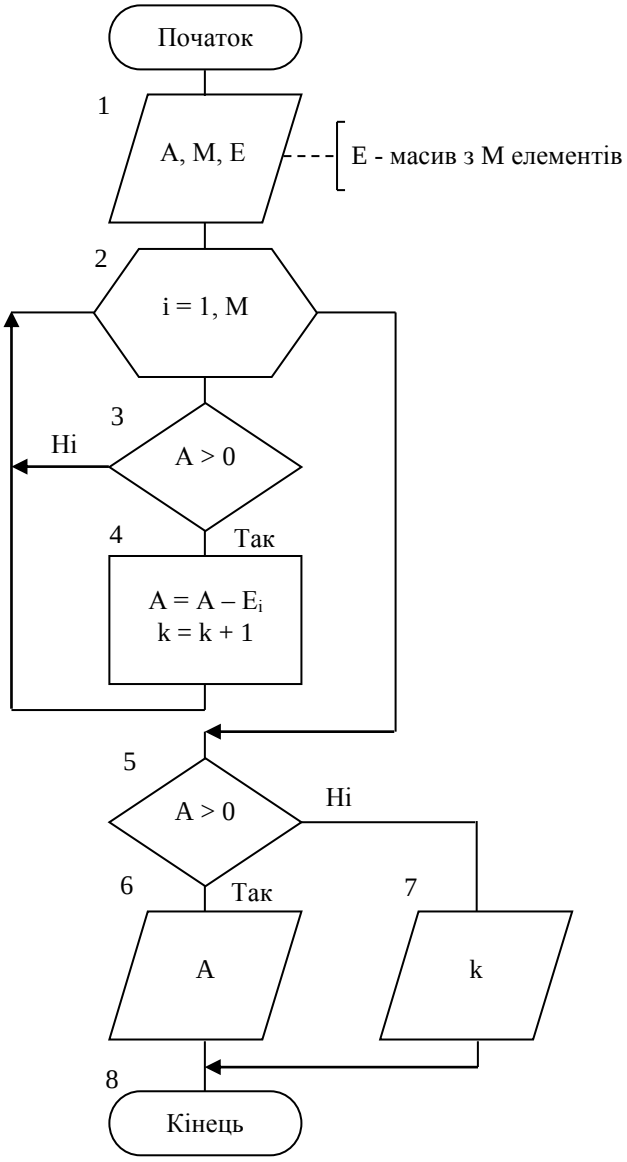
```
S = 0
For I = 1 To N
S = S + A(I) ^ 2
Next I
Text2 = CStr(S)
End Sub
```

TP

```
Program p22;
Var
a: array [1..100] of real;
n, i: integer; s, l: real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив a’);
For i:=1 to n do
Readln (a[i]);
s:=0;
For i:=1 to n do
s:=s+sqr(a[i]);
l:=sqrt(s);
Writeln (‘l=’, l:6:2);
End.
```

Задача 23.

Танкер доставив A тис. тон нафти. В порту є M нафтосховищ з емностями $E_1, E_2, \ldots, E_M$ (тис. тон). Нафтосховища заповнюються з танкера послідовно, починаючи з першого. Визначити, скільки нафтосховищ достатньо для прийому нафти. У випадку неможливості прийому всієї нафти підрахувати, скільки її залишилось в танкері й видрукувати текст “В танкері залишилось [кількість] тис. тон”.



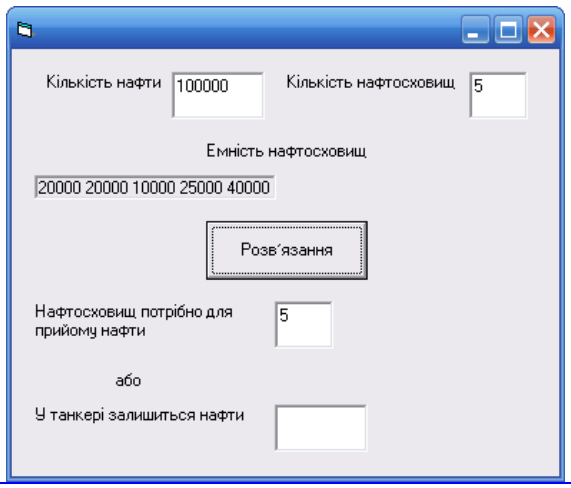
C++

```
#include<iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
	int M,k(0);
	double A; cout<<" Укажите сколько тыс.тонн нефти доставил танер "<<endl;
cout<<" A = ";
cin>>A;
cout<<" Укажите количество нефтехранилищ в порту "<<endl;
cout<<" M = ";
```

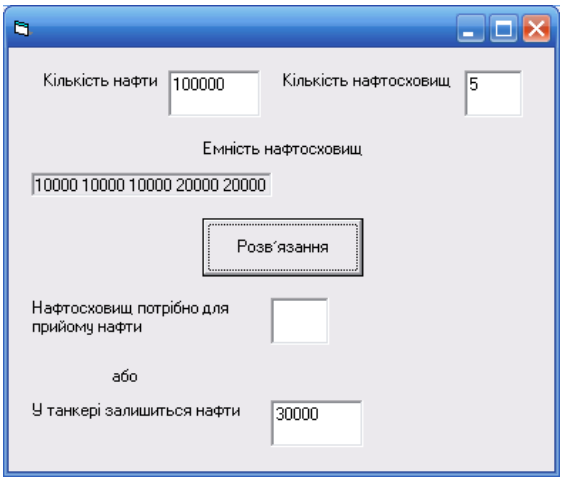
```
cin>>M;
cout<<" Укажите емкости нефтехранилищ "<<endl;

double *E = new double [M+1];
for(int i=1;i<=M;i++)
{
        cout<<" E["<<i<<"]= ";
        cin>>E[i];
}

for(int i=1;i<=M;i++)
 if(A>0)
 {
         A-=E[i];
         k++;
 }

 if (A>0) cout<<" В танкере осталось "<<A<<" тыс.тонн "<<endl<<endl;
  else cout<<" Для приема нефти достаточно "<<k<<" нефтехранилищ "<<endl<<endl;

 delete []E;
        system("pause");
        return 0;
        }
```

VB

```
 Option Explicit
Option Base 1
Dim E(), A, S As Single, I, M, K As Integer
Private Sub Command1_Click()
A = CSng(Text1)
M = CInt(Text2)
ReDim E(M)
Label2 = ""
For I = 1 To M
```



```
E(I) = CSng(InputBox("E(" & I & ")="))
Label2 = Label2 + CStr(E(I)) + " "
Next I
K = 0
For I = 1 To M
If A > 0 Then
A = A - E(I)
K = K + 1
End If
Next I
If A > 0 Then
Text4 = CStr(A)
Else
Text3 = CStr(K)
End If
End Sub
```



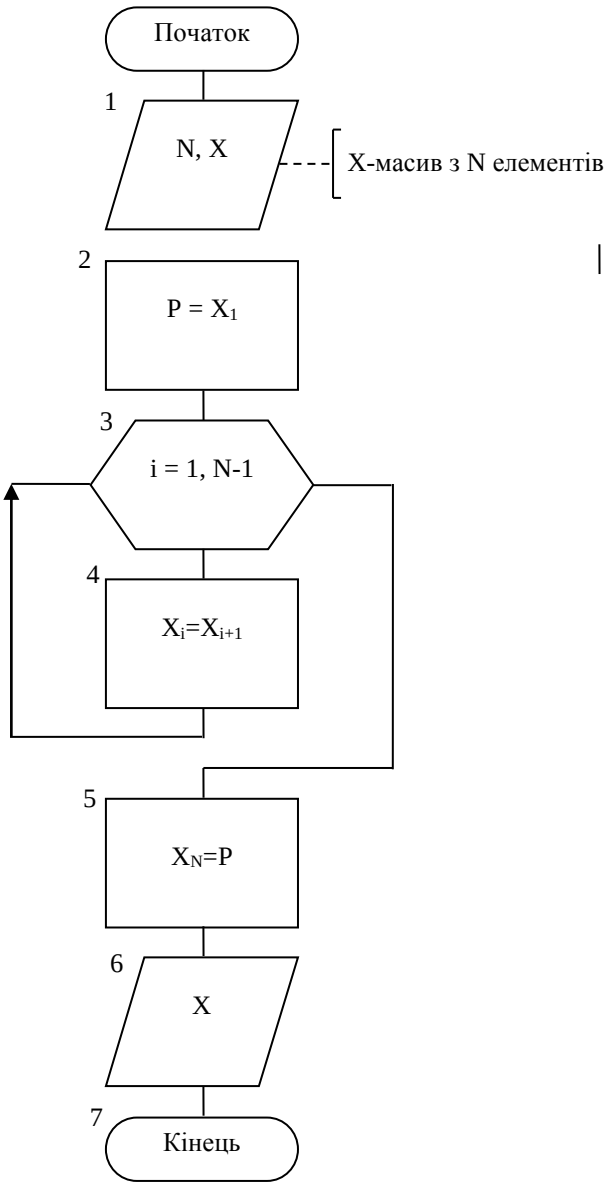
TP

```
Program p23;
Var
e: array [1..100] of real;
m, I, k: integer; a: real;
Begin
Writeln ('Введіть m');
Readln (m);
Writeln ('Введіть масив e');
For i:=1 to n do
Readln (e[i]);
For i:=1 to m do
begin
If a>0 then a:=a+(e[i]);
k:=k+1;
end;
If a>0 then Writeln ('a=', a:6:2) else Writeln ('k=',k);
End.
```

Перестановка елементів в масиві

Задача 24.

Виконати циклічне зрушення елементів масива вліво на одну позицію.



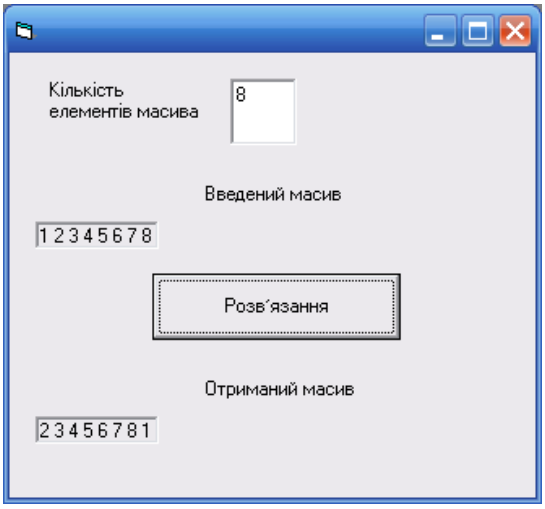
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int N,i;
```

```
        double P;
        cout<<"N=";cin>>N;
double *X = new double [N+1];

for(i=1;i<=N;i++)
        {
                cout<<"X["<<i<<"]=";
                cin>>X[i];
        }

P=X[1];
for(i=1;i<N;i++) X[i]=X[i+1];

X[N]=P;

for(i=1;i<=N;i++)
                cout<<"X["<<i<<"]="<<X[i]<<endl;

delete []X;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim X(), P As Single, I, N As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim X(N)
Label2 = ""
For I = 1 To N
X(I) = CSng(InputBox("X(" & I & ")="))
Label2 = Label2 + CStr(X(I)) + " "
Next I
P = X(1)
```



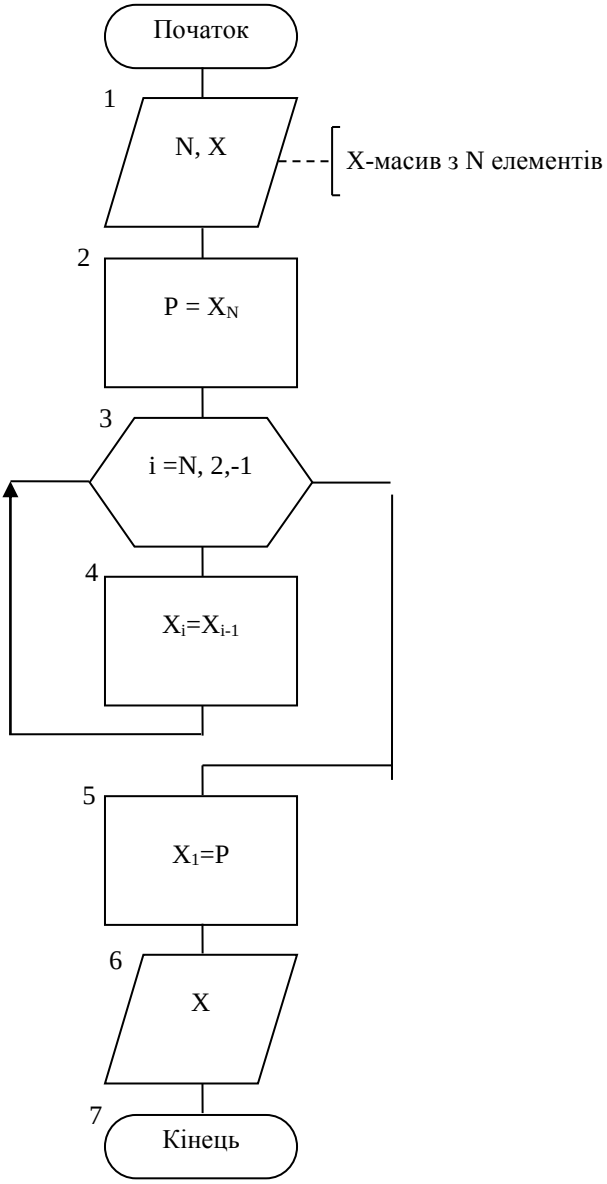
```
For I = 1 To N - 1
X(I) = X(I + 1)
Next I
X(N) = P
Label5 = ""
For I = 1 To N
Label5 = Label5 + CStr(X(I)) + " "
Next I
End Sub
```

TP

```
Program p24;
Var
X: array [1..100] of real;
n, i: integer; p:real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
p:=x[1];
For i:=1 to n-1 do
x[i]:= x[i+1];
x[n]:=p;
For i:=1 to n do
Writeln (‘x=’, x[i]:6:2);
End.
```

Задача 25.

Виконати циклічне зрушення елементів масива вліво на одну позицію.



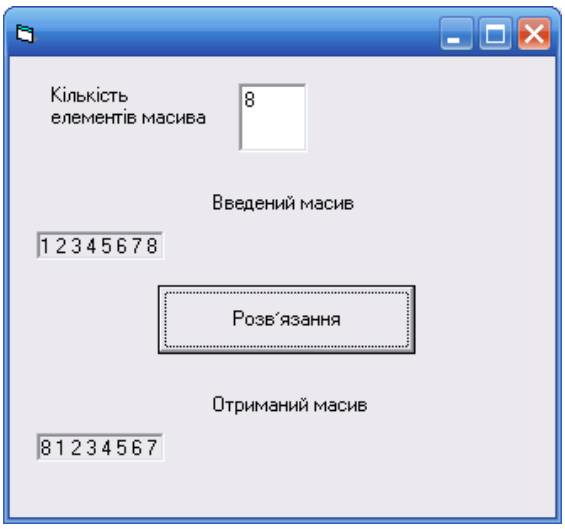
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int N,i;
```

```
        double P;
        cout<<"N=";cin>>N;
double *X = new double [N+1];

for(i=1;i<=N;i++)
        {
                cout<<"X["<<i<<"]=";
                cin>>X[i];
        }

P=X[N];
for(i=N;i>=2;i--) X[i]=X[i-1];

X[1]=P;

for(i=1;i<=N;i++)
                cout<<"X["<<i<<"]="<<X[i]<<endl;

delete []X;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim X(), P As Single, I, N As Integer
Private Sub Command1_Click()
N = CInt(Text1)
ReDim X(N)
Label2 = ""
For I = 1 To N
X(I) = CSng(InputBox("X(" & I & ")="))
Label2 = Label2 + CStr(X(I)) + " "
Next I
P = X(N)
```



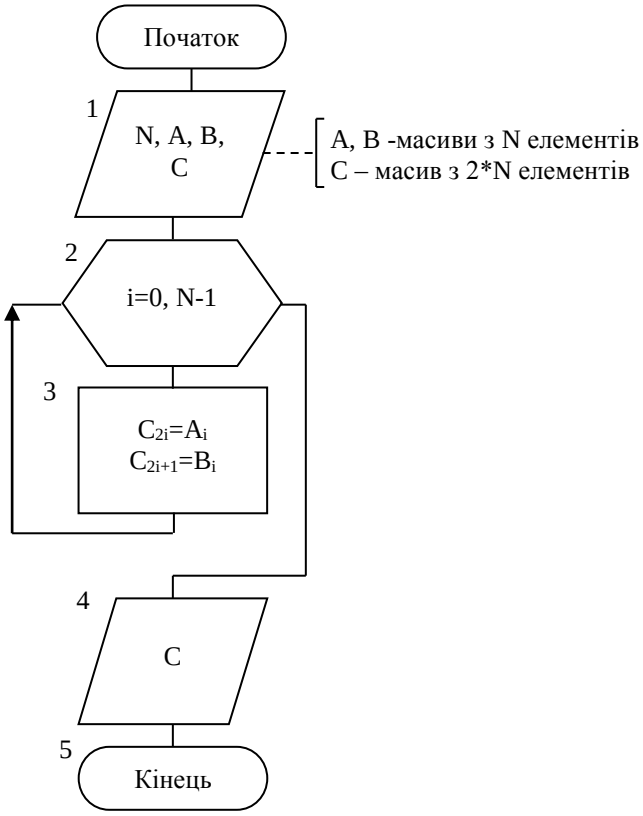
```
For I = N To 2 Step -1
X(I) = X(I - 1)
Next I
X(1) = P
Label5 = ""
For I = 1 To N
Label5 = Label5 + CStr(X(I)) + " "
Next I
End Sub
```

TP

```
Program p25;
Var
X: array [1..100] of real;
n, i: integer; p:real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
p:=x[n];
For i:=n downto 2 do
x[i]:= x[i-1];
x[1]:=p;
For i:=1 to n do
Writeln (‘x=’,x[i]:6:2);
End.
```

Задача 26.

За заданими векторами $A(a_1, a_2, \dots, a_n)$ та $B(b_1, b_2, \dots, b_n)$ отримати вектор $C(a_1, b_1, a_2, b_2, \dots, a_n, b_n)$.



C++

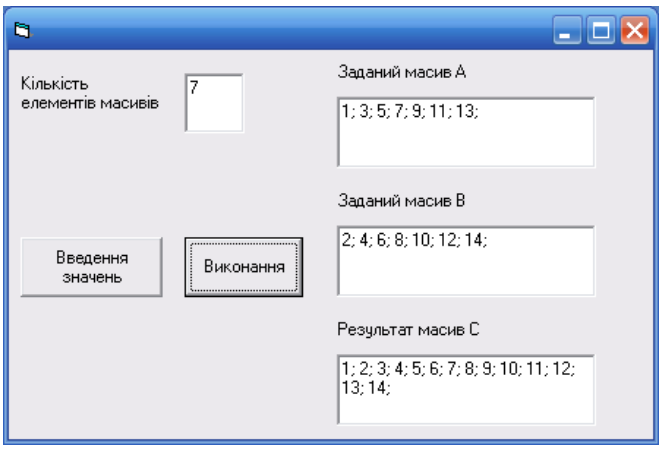
```
#include<iostream>
#include<Windows.h> ()
using namespace std;
int main()
{
        SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
        int n;
        cout<<" Введите количество элементов массива: ";
        cin>>n;

double *A = new double [n];
double *B = new double [n];
double *C = new double [2*n];
```

```
cout<<endl<<"Элементы массива A:"<<endl;
for(int i=0;i<n;i++)//ввод массива A
{
        cout<<" A["<<i+1<<"]= ";
        cin>>A[i];
}

cout<<endl<<"Элементы массива B:"<<endl;
for(int i=0;i<n;i++)//ввод массива B
{
        cout<<" B["<<i+1<<"]= ";
        cin>>B[i];
}

cout<<endl<<endl;
cout<<" Массив A:  ";
for(int i=0;i<n;i++)
        cout<<A[i]<<"\t";

cout<<endl<<endl;
cout<<" Массив B:  ";
for(int i=0;i<n;i++)
        cout<<B[i]<<"\t";

for(int i=0;i<n;i++)
{
        C[i*2]=A[i];
        C[i*2+1]=B[i];
}
cout<<endl<<endl;
cout<<" Массив C:  ";
for(int i=0;i<2*n;i++)
   cout<<C[i]<<"\t";
cout<<endl<<endl;
 delete []A;
 delete []B;
 delete []C;
```

```
        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a, b, c() As Single, i, n As Integer
Private Sub Command1_Click()
n = CInt(kolvo)
ReDim a(n), b(n), c(2 * n)
For i = 1 To n
a(i) = CSng(InputBox("a(" & i & ")="))
Next i
For i = 1 To n
b(i) = CSng(InputBox("b(" & i & ")="))
Next i
End Sub
Private Sub Command2_Click()
rez = ""
For i = 1 To n
rez = rez + CStr(a(i)) + "; "
Next i
rez2 = ""
For i = 1 To n
rez2 = rez2 + CStr(b(i)) + "; "
Next i
For i = 1 To n
c(2 * i - 1) = a(i)
c(2 * i) = b(i)
Next i
rez1 = ""
For i = 1 To 2 * n
rez1 = rez1 + CStr(c(i)) + "; "
Next i
End Sub
```

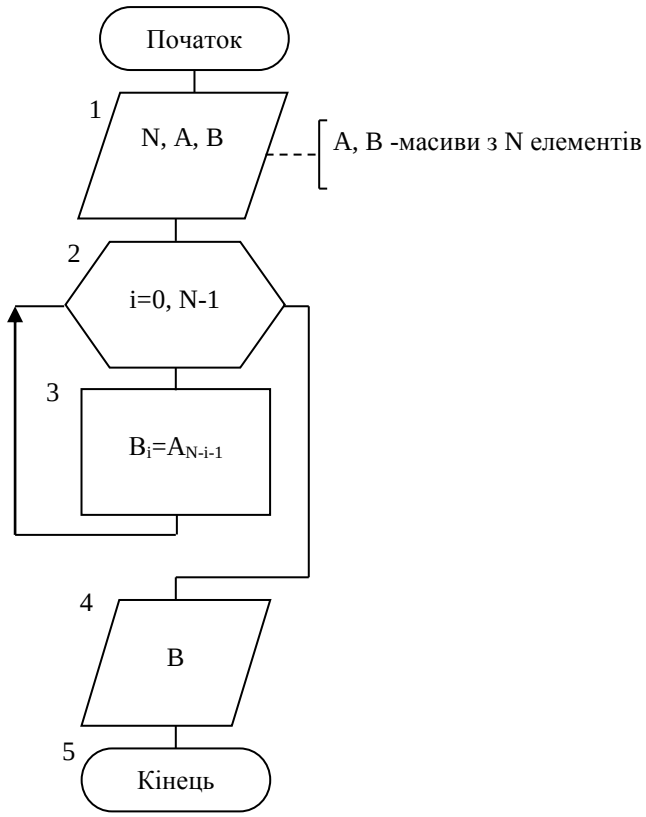


TP

```
Program p26;
Var
a, b: array [1..100] of real;
c: array [1..100] of real;
n, i: integer;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив a’);
For i:=1 to n do
Readln (a[i]);
Writeln (‘Введіть масив b’);
For i:=1 to n do
Readln (b[i]);
For i:=1 to n do
Begin
c[2*i-1]:=a[i];
c[2*i]:=b[i];
end;
For i:=1 to n*2 do
Writeln (‘c=’,c[i]:6:2);
End.
```

Задача 27.

Елементи масива $A(a_1, a_2, \ldots, a_n)$ задані. Необхідно отримати вектор $B(a_n, a_{n-1}, \ldots, a_2, a_1)$.



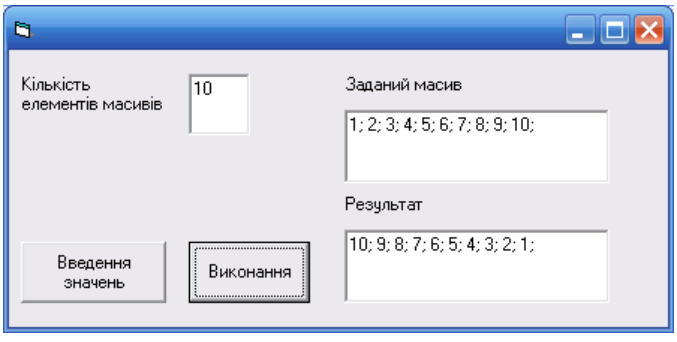
C++

```
#include<iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
	int n;
	cout<<" Введите количество элементов массива: ";
	cin>>n;

double *A = new double [n];
double *B = new double [n];
```

```
for(int i=0;i<n;i++)
{
        cout<<" A["<<i+1<<"]= ";
        cin>>A[i];
}
cout<<endl<<endl;
cout<<" Массив A:  ";
for(int i=0;i<n;i++)
        cout<<A[i]<<"\t";

for(int i=0;i<n;i++)
  B[i]=A[n-i-1];

cout<<endl<<endl;
cout<<" Массив B:  ";
for(int i=0;i<n;i++)
  cout<<B[i]<<"\t";

cout<<endl<<endl;
 delete []A;
 delete []B;
        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a(), b() As Single, p As Single, i, n As Integer
Private Sub Command1_Click()
n = CInt(kolvo)
ReDim a(n), b(n)
For i = 1 To n
a(i) = CSng(InputBox("a(" & i & ")="))
Next i
```



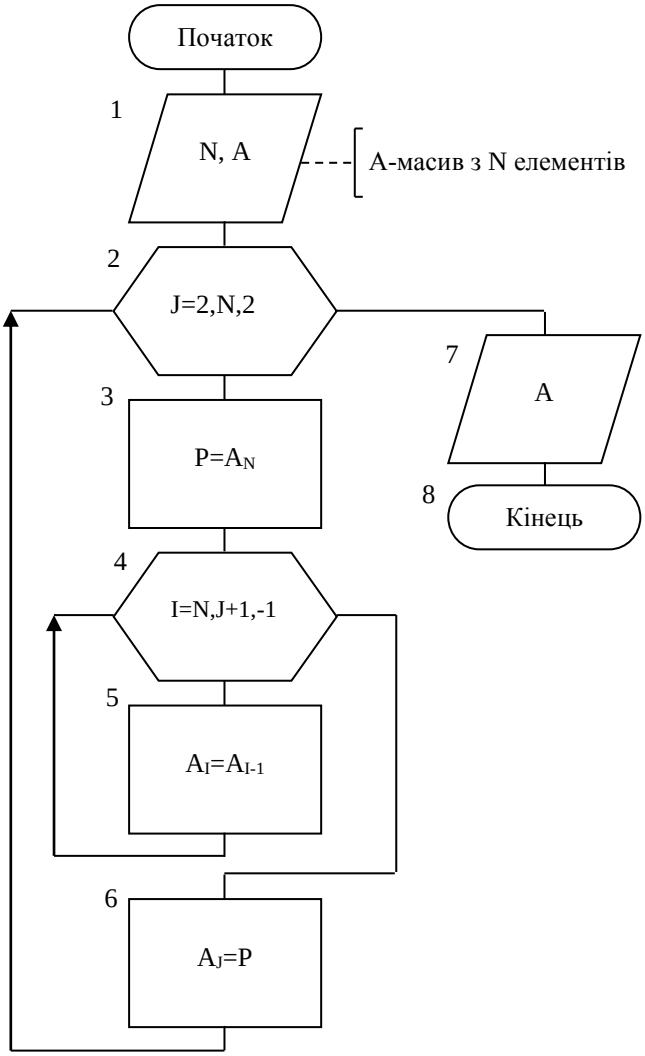
```
End Sub
Private Sub Command2_Click()
rez = ""
For i = 1 To n
rez = rez + CStr(a(i)) + "; "
Next i
For i = 1 To n
b(i) = a(n - i + 1)
Next i
rez1 = ""
For i = 1 To n
rez1 = rez1 + CStr(b(i)) + "; "
Next i
End Sub
```

TP

```
Program p27;
Var
a: array [1..100] of real;
b: array [1..100] of real;
n, i: integer;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив a’);
For i:=1 to n do
Readln (a[i]);
For i:=0 to n-1 do
b[i] :=a[n-i-1];
For i:=1 to n do
Writeln (‘b=’,b[i]:6:2);
End.
```

Задача 28.

Заданий масив $A(N)$. Отримати масив: a_1, a_N, a_2, a_{N-1}, a_3, a_{N-2}…..



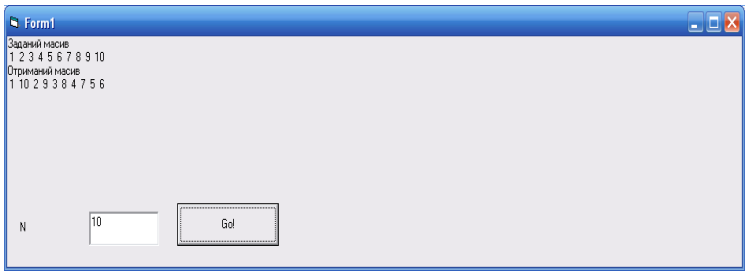
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,i,j;
	double P;
	cout<<"N=";cin>>N;
double *A = new double [N+1];
```

```
for(i=1;i<=N;i++)
	{
		cout<<"A["<<i<<"]=";
		cin>>A[i];
	}

for(j=2;j<=N;j+=2)
{
	P=A[N];
	for(i=N;i>=j+1;i--)
		A[i]=A[i-1];
	A[j]=P;
}

for(i=1;i<=N;i++)
		cout<<"A["<<i<<"]="<<A[i]<<endl;

delete []A;
system("pause");
	return 0;
}
```

VB

```
 Private Sub cmdGO_Click()
Dim A() As Integer, i, N As Integer
N = CInt(txtN)
ReDim A(1 To N)
Print "Заданий масив"
For i = 1 To N
A(i) = i
Print A(i);
Next
Print
For j = 2 To N Step 2
p = A(N)
```



```
For i = N To j + 1 Step -1
A(i) = A(i - 1)
Next i
A(j) = p
Next j
Print "Отриманий масив"
For i = 1 To N
Print A(i);
Next
End Sub
```

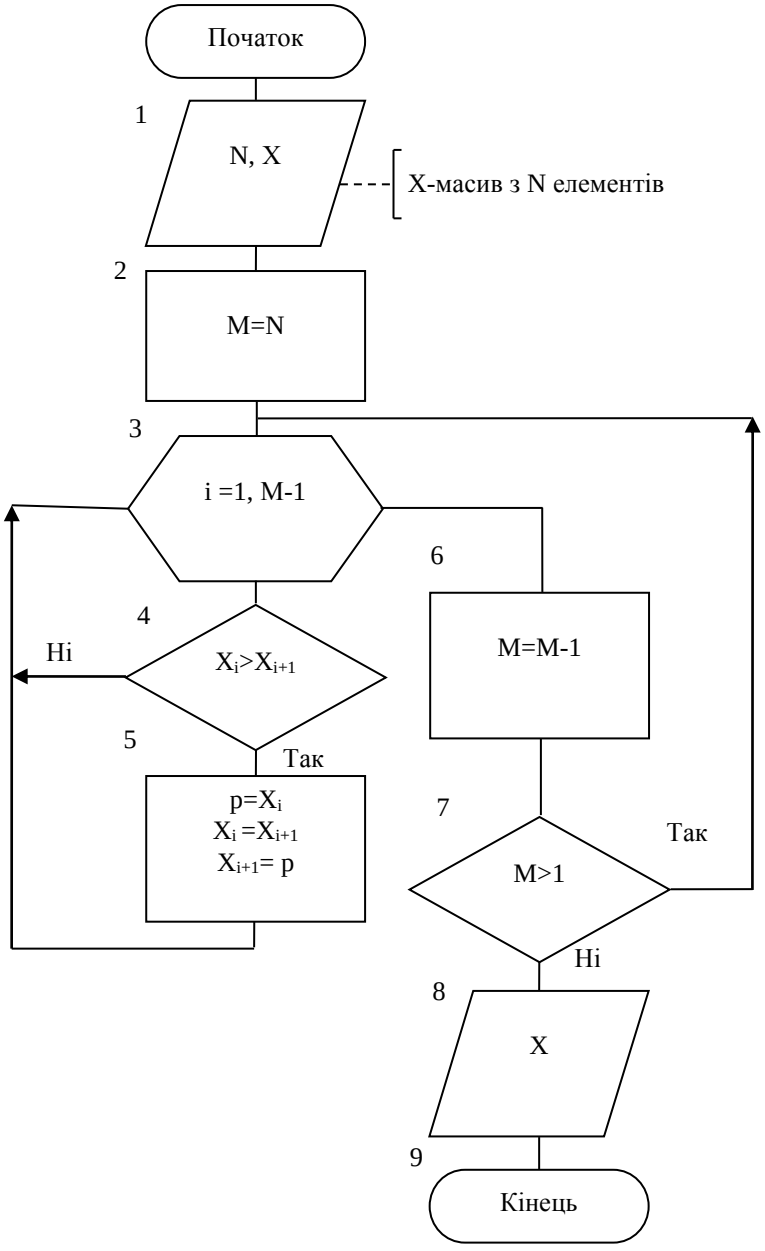
TP

```
Program p28;
Var
a: array [1..100] of real;
n, i: integer;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив a’);
For i:=1 to n do
Readln (a[i]);
j:=2;
Repeat
P:=a[n];
For n:= downto i+1 do
a[i]:= a[i-1];
a[j]:=p;
j:=j+2;
until j>n;
For i:=1 to n do
Writeln (‘a=’, a[i]:6:2);
End.
```

Сортування одномірного масиву

Задача 29.

Розташувати елементы масива *X* з *N* елементів за зростанням. Задачу вирішити методом «бульбашки».



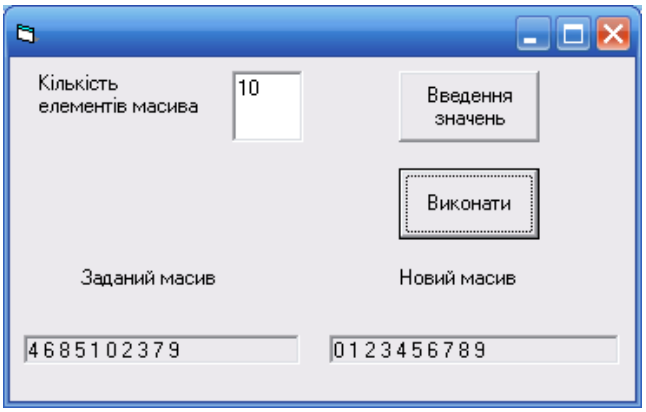
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i;
	double P;
	cout<<"N=";cin>>N;
double *X = new double [N+1];

for(i=1;i<=N;i++)
	{
		cout<<"X["<<i<<"]=";
		cin>>X[i];
	}

M=N;
do
{
  for(i=1;i<M;i++)
    if(X[i]>X[i+1])
      {
		P=X[i];
		X[i]=X[i+1];
		X[i+1]=P;
      }
M--;
}
while(M>1);

for(i=1;i<=N;i++)
		cout<<"X["<<i<<"]="<<X[i]<<endl;
```

```
delete []X;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim x() As Single, p As Single, i, m, n As Integer
Private Sub vvod_Click()
n = CInt(kolvo)
ReDim x(n)
For i = 1 To n
x(i) = CSng(InputBox("x(" & i & ")="))
Next i
End Sub

Private Sub rascet_Click()
mas = ""
For i = 1 To n
mas = mas + CStr(x(i)) + " "
Next i
For m = n To 1 Step -1
For i = 1 To m - 1
If x(i) > x(i + 1) Then p = x(i): x(i) = x(i + 1): x(i
+ 1) = p
Next i, m
mas1 = ""
For i = 1 To n
mas1 = mas1 + CStr(x(i)) + " "
Next i
End Sub
```

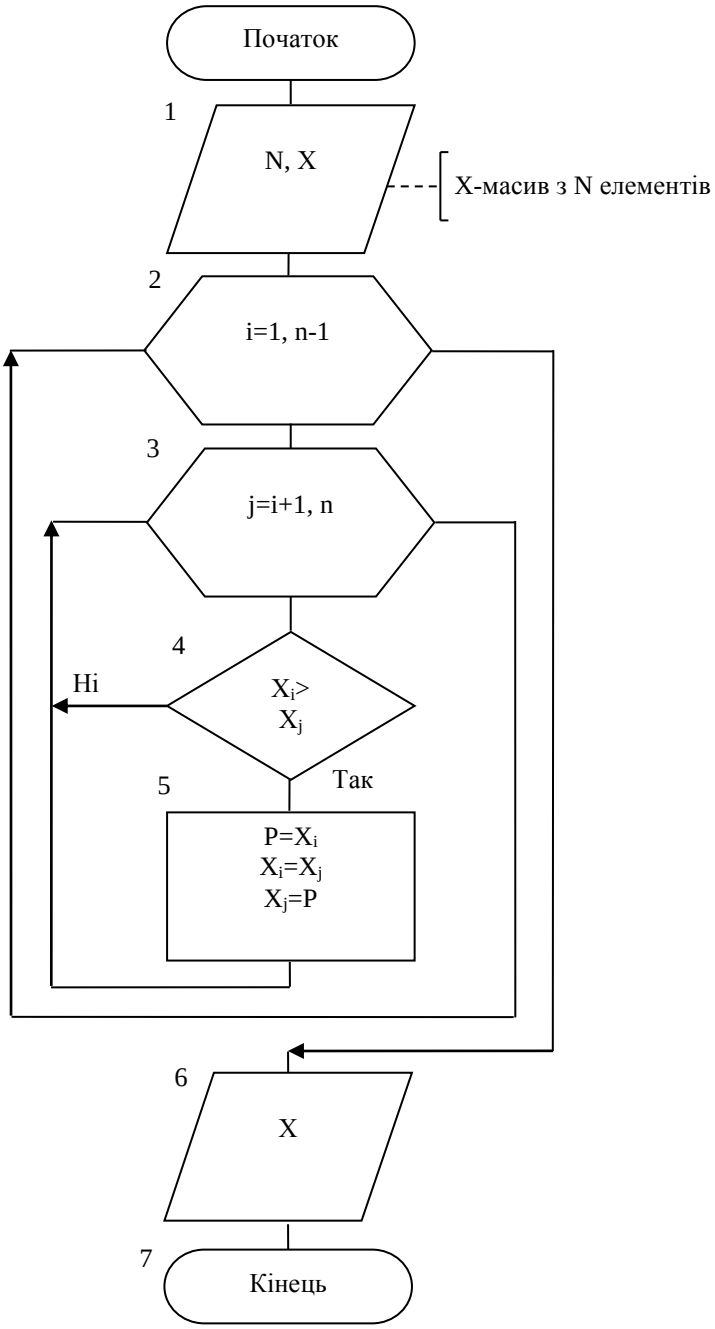


TP

```
Program p29;
Label
10;
Var
x: array [1..100] of real;
n, i: integer; p: real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив X’);
For i:=1 to n do
Readln (x[i]);
m:=n;
10:
For i:= 1 to m-1 do
If x[i]>x[i+1] then begin
p:= x[i];
x[i]:=x[i+1];
x[i+1]:=p;
end;
m:=m-1;
if m>1 then GOTO 10 else
For i:=1 to n do
Writeln (‘x=’, x[i]:6:2);
End.
```

Задача 30.

Розташувати елементы масива за зростанням. Задачу вирішити методом простого лінійного сортування.



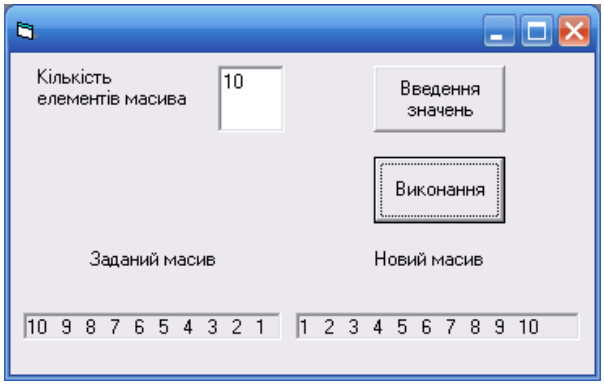
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
```

```
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int N,M,i,j;
        double P;
        cout<<"N=";cin>>N;
double *X = new double [N+1];

for(i=1;i<=N;i++)
        {
                cout<<"X["<<i<<"]=";
                cin>>X[i];
        }

 for(i=1;i<N;i++)
  for(j=i+1;j<=N;j++)
   if(X[i]>X[j])
    {
        P=X[i];
        X[i]=X[j];
        X[j]=P;
    }

for(i=1;i<=N;i++)
                cout<<"X["<<i<<"]="<<X[i]<<endl;

delete []X;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim x() As Single, p As Single, i, j, n As Integer
Private Sub vvod_Click()
n = CInt(kolvo)
```




```
ReDim x(n)
For i = 1 To n
x(i) = CSng(InputBox("x(" & i & ")="))
Next i
End Sub

Private Sub rascet_Click()
mas = ""
For i = 1 To n
mas = mas + CStr(x(i)) + "   "
Next i
For i = 1 To n - 1
For j = i + 1 To n
If x(i) > x(j) Then p = x(i): x(i) = x(j): x(j) = p
Next j, i
mas1 = ""
For i = 1 To n
mas1 = mas1 + CStr(x(i)) + "   "
Next i
End Sub
```

TP

```
Program p30;
Var
x: array [1..100] of real;
n, I, j: integer; p: real;
Begin
Writeln (‘Введіть n’);
Readln (n);
Writeln (‘Введіть масив x’);
For i:=1 to n do
Readln (x[i]);
For i:=1 to n-1 do
For j:=i+1 to n do
begin
If x[i]>x[j] then p:=(x[i]);
```

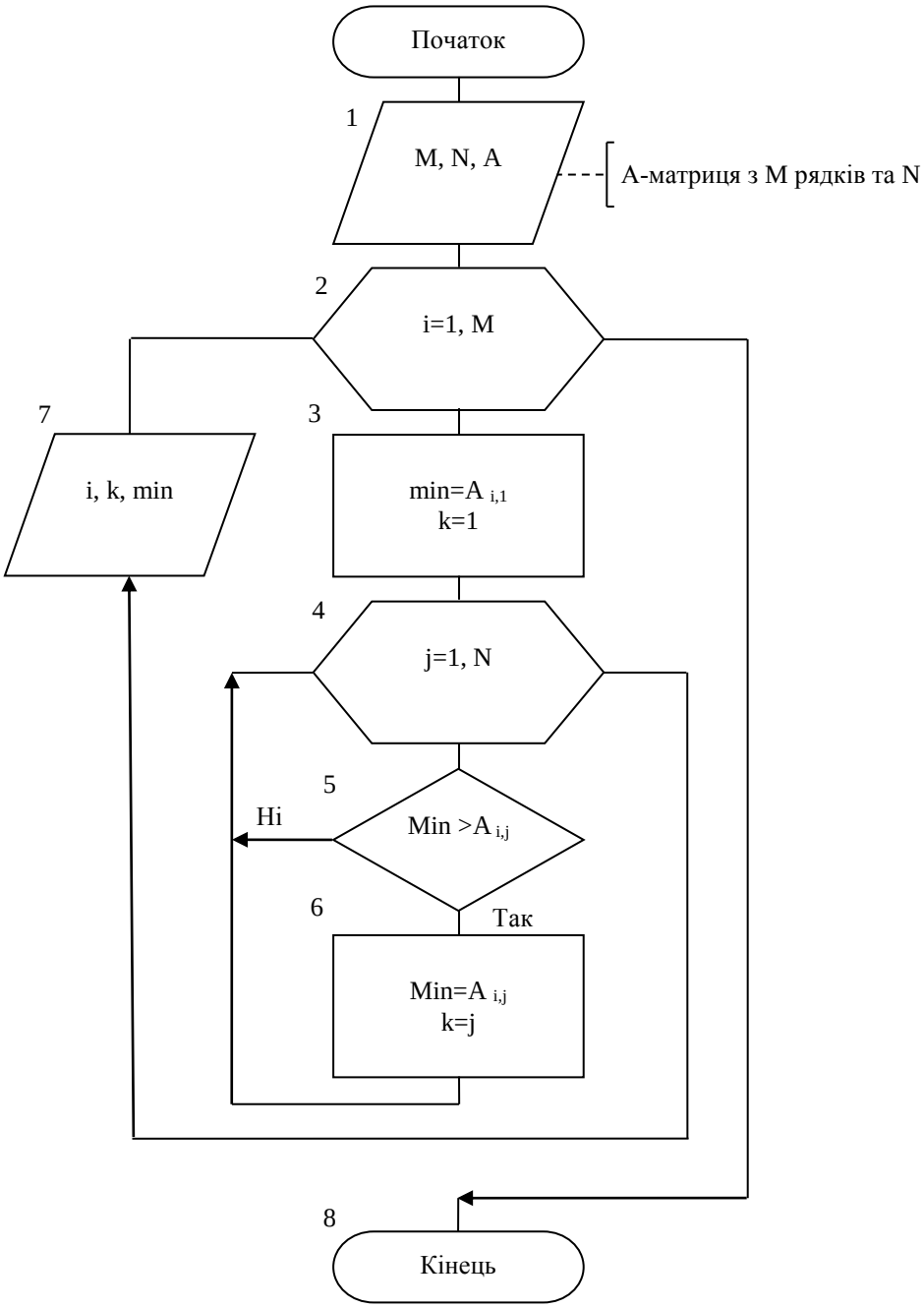
```
x[i]:= x[j];
x[i]:=p;
end;
For i:=1 to n do
Writeln (‘x=’,x[i]:6:2); End.
```

4.2.3 Обробка двувимірних масивів

Пошук екстремума в матриці

Задача 31.

Для кожного рядка матриці *A*(*M*,*N*) визначити мінімальний елемент та номер стовпця, в якому він знаходиться.



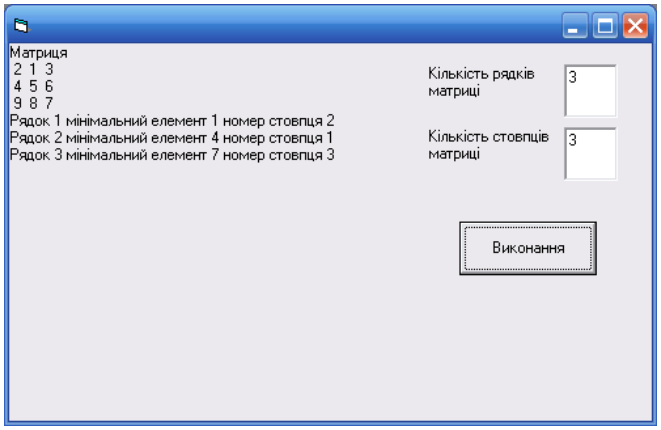
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i,j,k;
	double min;
	cout<<"M=";cin>>M;
	cout<<"N=";cin>>N;
double **A = new double* [M+1];
	for(i=1;i<=M;i++)
		A[i] = new double [N+1];

for(i=1;i<=M;i++)
 for(j=1;j<=N;j++)
	{
		cout<<"A["<<i<<"]["<<j<<"]=";
		cin>>A[i][j];
}
for(i=1;i<=M;i++)
{
 for(j=1;j<=N;j++)
		cout<<A[i][j]<<"\t";
 cout<<endl;
}
 for(i=1;i<=M;i++)
 {
	 min=A[i][1];
	 k=1;
  for(j=1;j<=N;j++)
    if(min>A[i][j])
	{
		min=A[i][j];
```

```
                k=j;
        }
        cout<<"Строка: "<<i<<" номер столбца: "<<k<<" min: "<<min<<endl;
 }
for(i=1;i<=M;i++)
        delete A[i];
delete []A;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a() As Single, min As Single, nmin, i, j, m, n As Integer
Private Sub rascet_Click()
m = CInt(text1)
n = CInt(text2)
ReDim a(m, n)
Print "Матриця"
For i = 1 To m
For j = 1 To n
a(i, j) = CSng(InputBox("a(" & i & "," & j & ")="))
Print a(i, j);
Next j
Print
Next i
For i = 1 To m
min = a(i, 1): nmin = 1
For j = 2 To n
If min > a(i, j) Then min = a(i, j): nmin = j
Next j
Print "Рядок"; i; "мінімальний елемент"; min; "номер стовпця"; nmin
Next i
End Sub
```



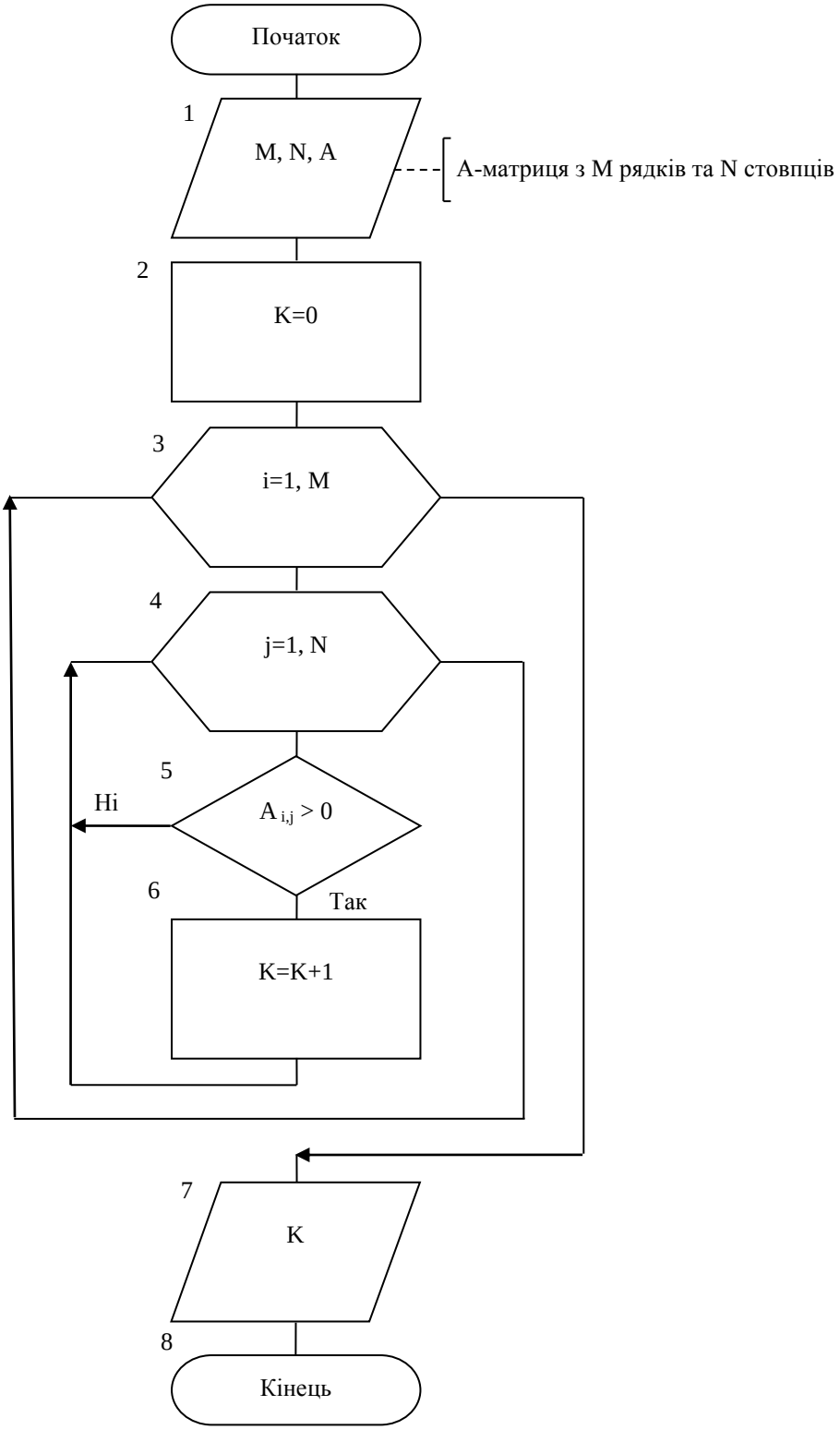
TP

```
Program p31;
Var
a: array [1..20,1..20] of real;
n, m, i, j, k: integer; min: real;
Begin
Writeln (‘Введіть n, m’);
Readln (n, m);
Writeln (‘Введіть масив a’);
For i:=1 to m do
For j:=1 to n do
Readln (a[I,j]);
For i:=1 to m do
begin
Min:=a[I,j];
K:=1;
For j:= to n do
begin
If min>a[I,j] then
begin
min:=a[I,j];
k:=j;
end;
end;
writeln ('min=',min:6:2, i, k);
end;
End.
```

Підрахунок кількостей компонент, що задовільняють умові, в матриці

Задача 32.

Визначити кількість додатніх елементів двовимірного масиву $A_{m,n}$.



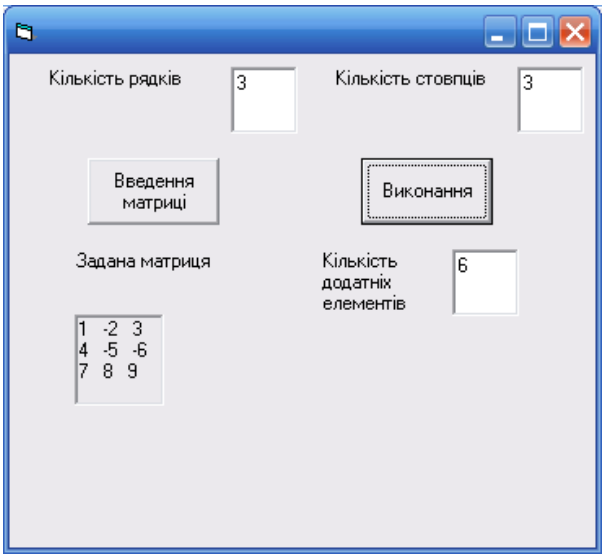
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i,j,k;
	double min;
	cout<<"M=";cin>>M;
	cout<<"N=";cin>>N;
double **A = new double* [M+1];
	for(i=1;i<=M;i++)
		A[i] = new double [N+1];

for(i=1;i<=M;i++)
 for(j=1;j<=N;j++)
	{
		cout<<"A["<<i<<"]["<<j<<"]=";_____
		cin>>A[i][j];
}

for(i=1;i<=M;i++)
{
 for(j=1;j<=N;j++)
		cout<<A[i][j]<<"\t";
 cout<<endl;
}

k=0;
 for(i=1;i<=M;i++)
  for(j=1;j<=N;j++)
    if(A[i][j]>0) k++;

 cout<<"Количество положтельных элементов = "<<k;
```

```
for(i=1;i<=M;i++)
        delete A[i];
delete []A;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a() As Single, k, i, j, n, m As Integer
Private Sub vvod_Click()
n = CInt(Text1)
m = CInt(Text2)
ReDim a(n, m)
For i = 1 To n
For j = 1 To m
a(i, j) = CSng(InputBox("a(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To n
For j = 1 To m
matr = matr + CStr(a(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
End Sub
Private Sub rascet_Click()
k = 0
For i = 1 To n
For j = 1 To m
If a(i, j) > 0 Then k = k + 1
Next j, i
Text3 = CStr(k)
End Sub
```

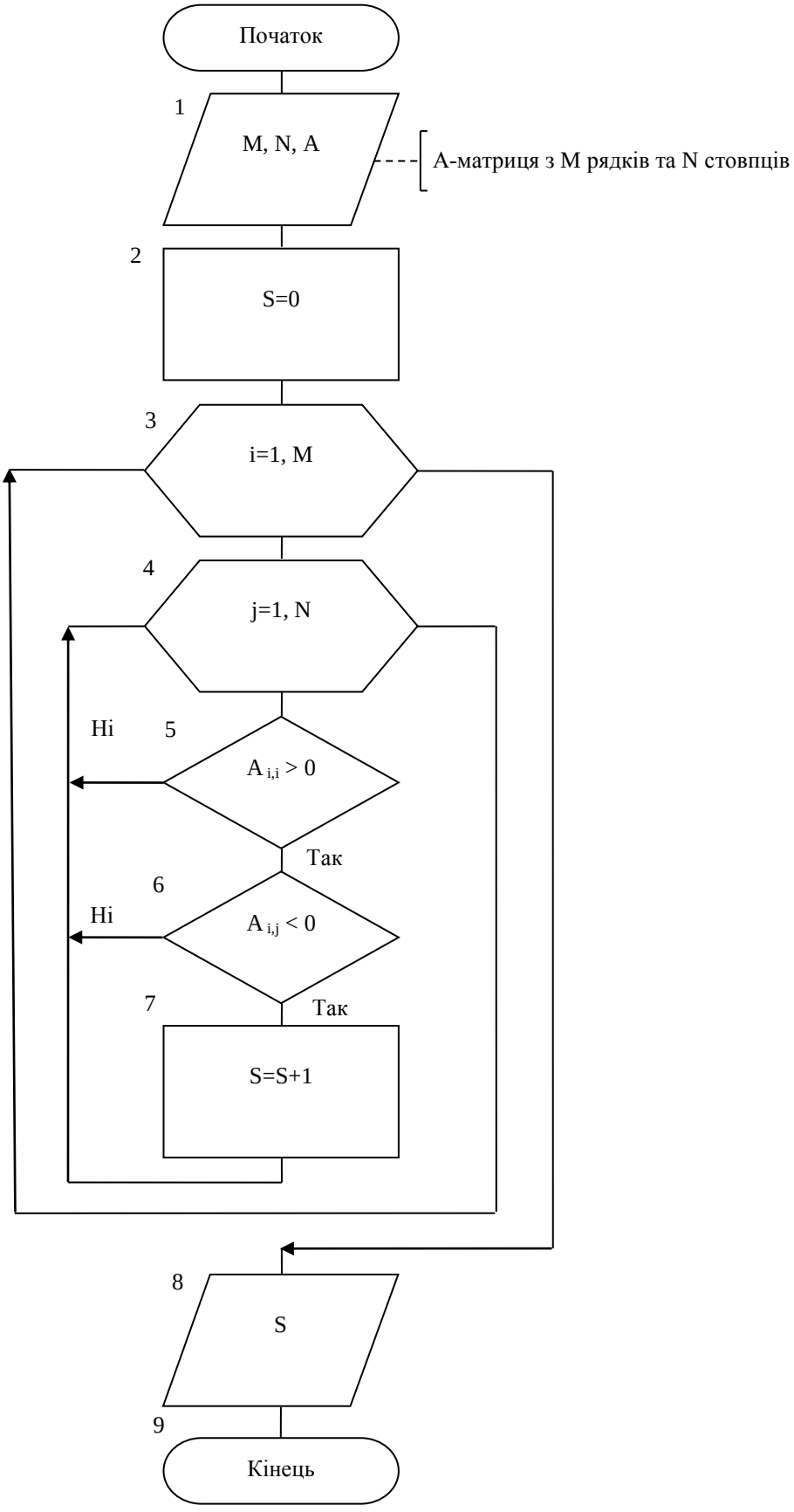


TP

```
Program p32;
Var
a: array [1..10,1..10] of real;
i, j, m, n, k: integer;
Begin
Readln (m, n);
for i:=1 to m do
for j:=1 to n do
readln (a[i,j]);
k:=0;
for i:=1 to m do
begin
for j:=1 to n do
if a[i,j]>0 then k:=k+1;
end;
writeln ('k=',k);
End.
```

Задача 33.

У квадратній матриці визначити кількість від’ємних елементів у рядках з додатними елементами на головній діагоналі.



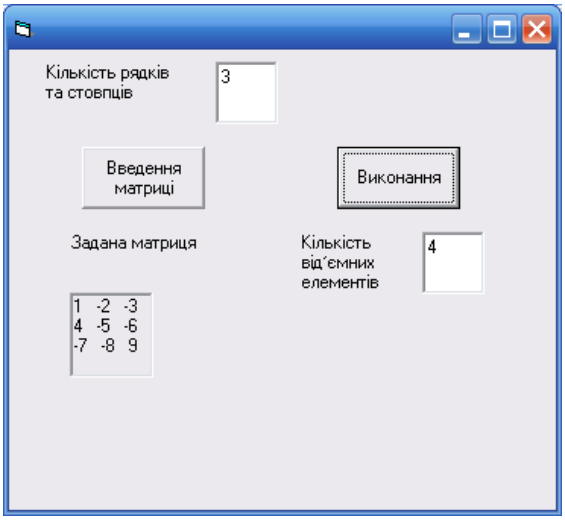
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,i,j,s;
	cout<<"N=";cin>>N;
double **A = new double* [N+1];
	for(i=1;i<=N;i++)
		A[i] = new double [N+1];

for(i=1;i<=N;i++)
 for(j=1;j<=N;j++)
	{
		cout<<"A["<<i<<"]["<<j<<"]=";
		cin>>A[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=N;j++)
		cout<<A[i][j]<<"\t";
 cout<<endl;
}
s=0;
 for(i=1;i<=N;i++)
	 if(A[i][i]>0)
	for(j=1;j<=N;j++)
	 if(A[i][j]<0) s++;
 cout<<"S = "<<s;

for(i=1;i<=N;i++)
	delete A[i];
```

```
delete []A;
system("pause");
        return 0;
}
```

VB

```
Option Explicit
Option Base 1
Dim a() As Single, k, i, j, n As Integer
Private Sub vvod_Click()
n = CInt(Text1)
ReDim a(n, n)
For i = 1 To n
For j = 1 To n
a(i, j) = CSng(InputBox("a(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To n
For j = 1 To n
matr = matr + CStr(a(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
End Sub

Private Sub rascet_Click()
k = 0
For i = 1 To n
For j = 1 To n
If a(i, i) > 0 Then
If a(i, j) < 0 Then k = k + 1
End If
Next j
Next i
Text3 = CStr(k)
End Sub
```



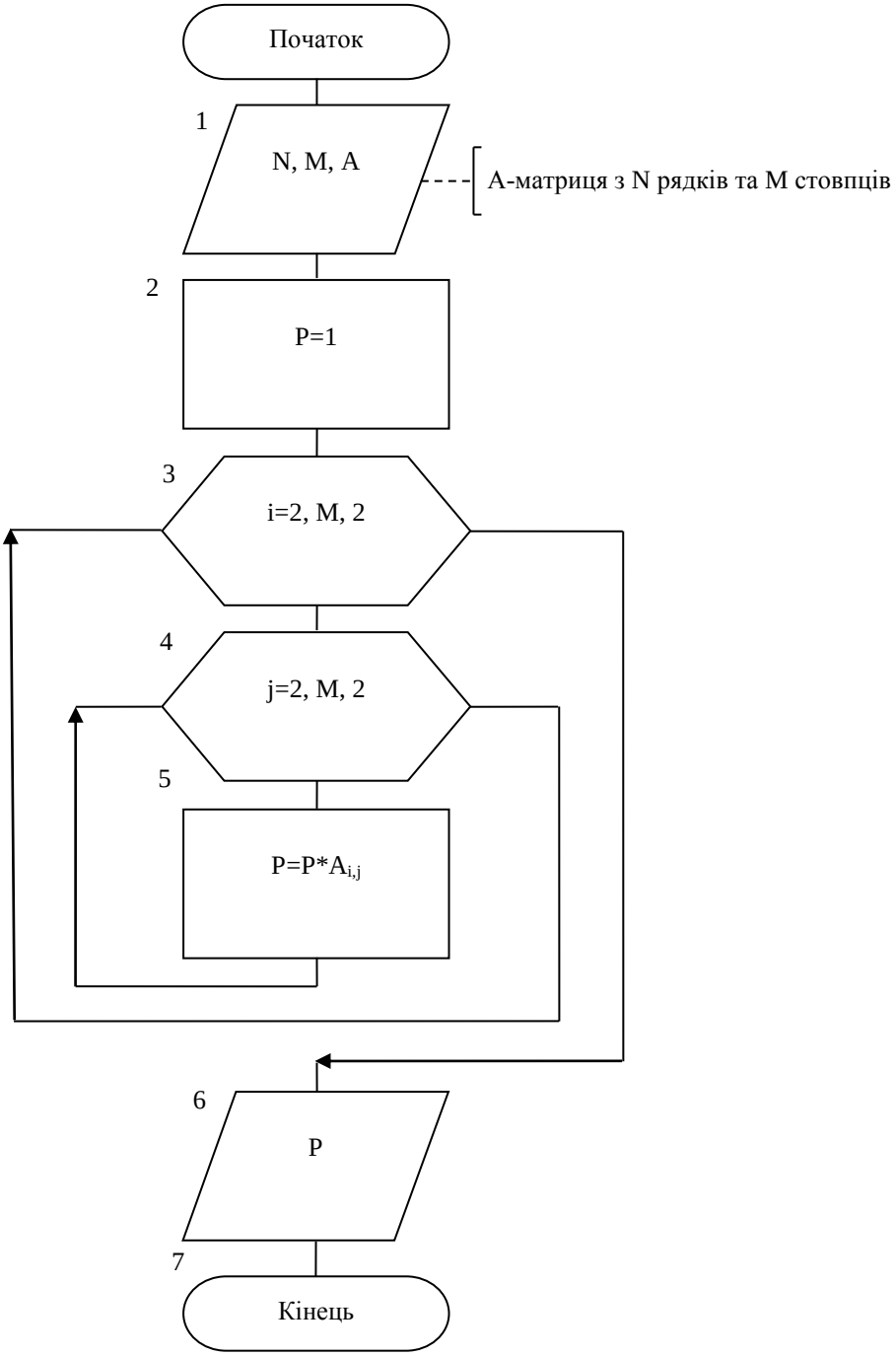
TP

```
Program p33;
Var
a: array [1..10,1..10] of real;
i, j, n, s: integer;
Begin
Readln (n);
for i:=1 to n do
for j:=1 to n do
readln (a[i,j]);
s:=0;
for i:=1 to n do
begin
for j:=1 to n do
if a[i,i]>0 then begin
if a[i,j]<0 then s:=s+1;
end; end;
writeln ('s=',s);
End.
```

Накопичення в матриці

Задача 34.

Визначити добуток елементів матриці, що знаходяться на перетині парних рядків та парних стовпців.



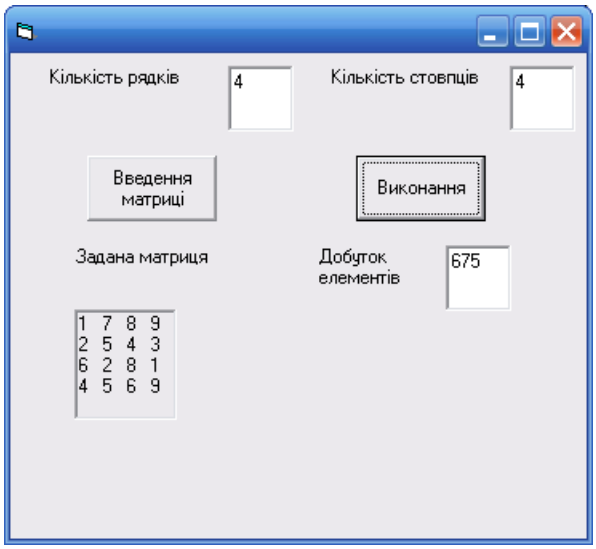
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
```

```
   SetConsoleOutputCP(1251);
        int N,M,i,j,p;
        cout<<"N=";cin>>N;
        cout<<"M=";cin>>M;
double **A = new double* [N+1];
        for(i=1;i<=N;i++)
                A[i] = new double [M+1];

for(i=1;i<=N;i++)
 for(j=1;j<=M;j++)
        {
                cout<<"A["<<i<<"]["<<j<<"]=";
                cin>>A[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=M;j++)
                cout<<A[i][j]<<"\t";
 cout<<endl;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a(), p As Single, i, j, n, m As Integer
Private Sub vvod_Click()
n = CInt(Text1)
m = CInt(Text2)
ReDim a(n, m)
For i = 1 To n
For j = 1 To m
a(i, j) = CSng(InputBox("a(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To n
```



```
For j = 1 To m
matr = matr + CStr(a(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
End Sub

Private Sub rascet_Click()
p = 1
For i = 2 To n Step 2
For j = 2 To m Step 2
p = p * a(i, j)
Next j, i
Text3 = CStr(p)
End Sub
```

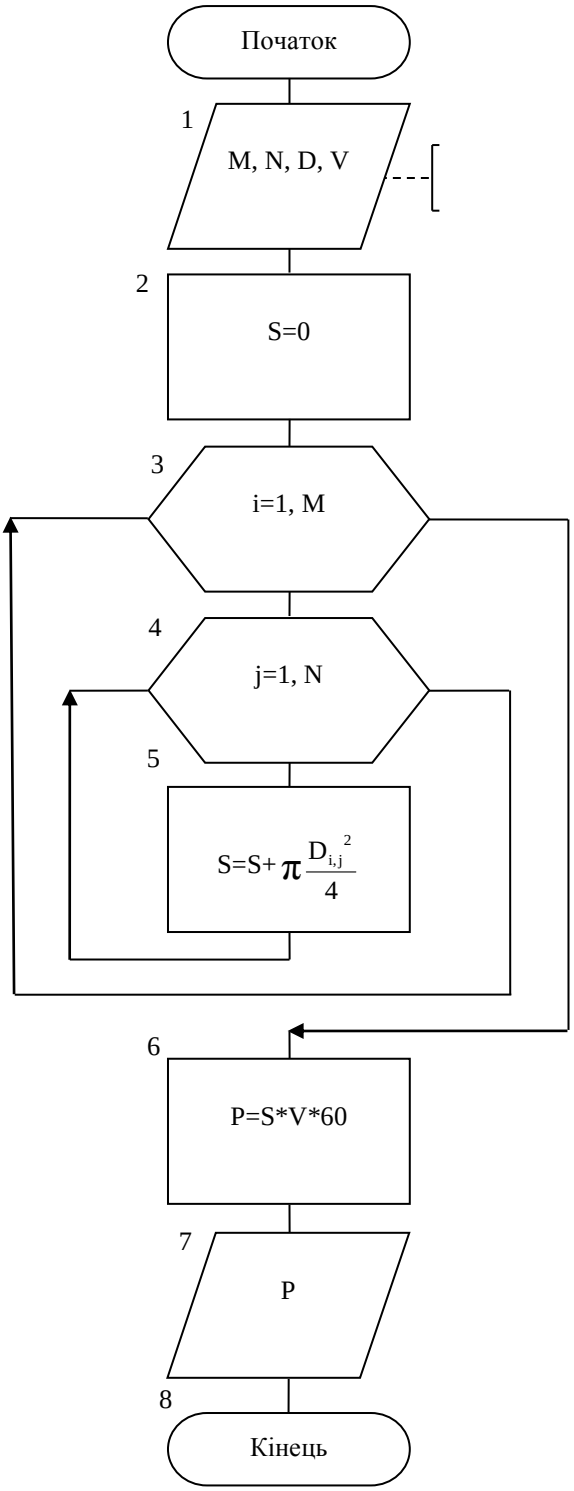
TP

```
Program p34;
Var
a: array [1..10,1..10] of real;
i, j, m, n: integer; p: real;
Begin
Readln (m, n);
for i:=1 to n do
for j:=1 to m do
readln (a[i,j]);
p:=1;
i:=2;
repeat
j:=2;
repeat
p:=p*a[I,j];
j:=j+2;
until j>m;
i:=i+2;
until i>n;
```

```
writeln (‘p=’,p:6:2);
End.
```

Задача 35.

Решітка водозабірної станції містить круглі отвори, розташовані у вигляді прямокутної матриці з кількістю отворів *M*x*N*. Внаслідок запливання і корозії діаметри отворів змінилися до значень D_{11}, D_{12}, D_{13},.., D_{mn}. Визначити продуктивність водозабірної станції протягом однієї хвилини, якщо середня швидкість руху водяного потоку через решітку дорівнює *V* м/сек.



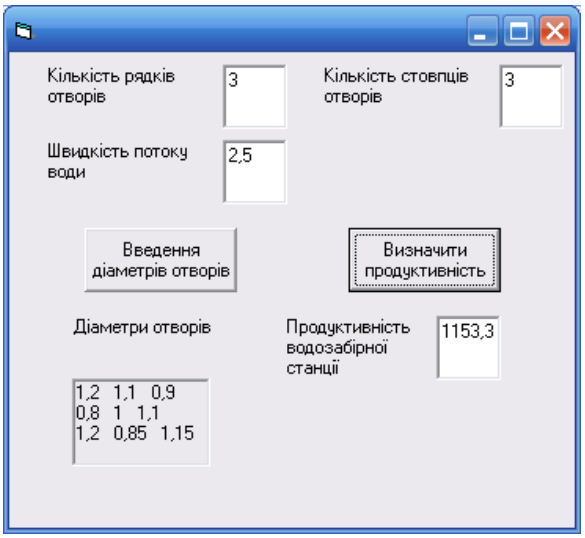
C++

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i,j;
	double V,S,P;
	cout<<"N=";cin>>N;
	cout<<"M=";cin>>M;
double **D = new double* [N+1];
	for(i=1;i<=N;i++)
		D[i] = new double [M+1];

	cout<<"V=";cin>>V;


for(i=1;i<=N;i++)
 for(j=1;j<=M;j++)
	{
		cout<<"D["<<i<<"]["<<j<<"]=";
		cin>>D[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=M;j++)
		cout<<D[i][j]<<"\t";
 cout<<endl;
}
```

```
S=0;
 for(i=1;i<=N;i++)
       for(j=1;j<=M;j++)
         S+=M_PI*pow(D[i][j],2)/4;
 P=S*V*60;

 cout<<"P = "<<P;

for(i=1;i<=N;i++)
           delete D[i];
delete []D;
system("pause");
           return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim d(), v, s, p As Single, i, j, n, m As Integer

Private Sub vvod_Click()
n = CInt(Text1)
m = CInt(Text2)
v = CSng(Text3)
ReDim d(n, m)
For i = 1 To n
For j = 1 To m
d(i, j) = CSng(InputBox("d(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To n
For j = 1 To m
matr = matr + CStr(d(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
```



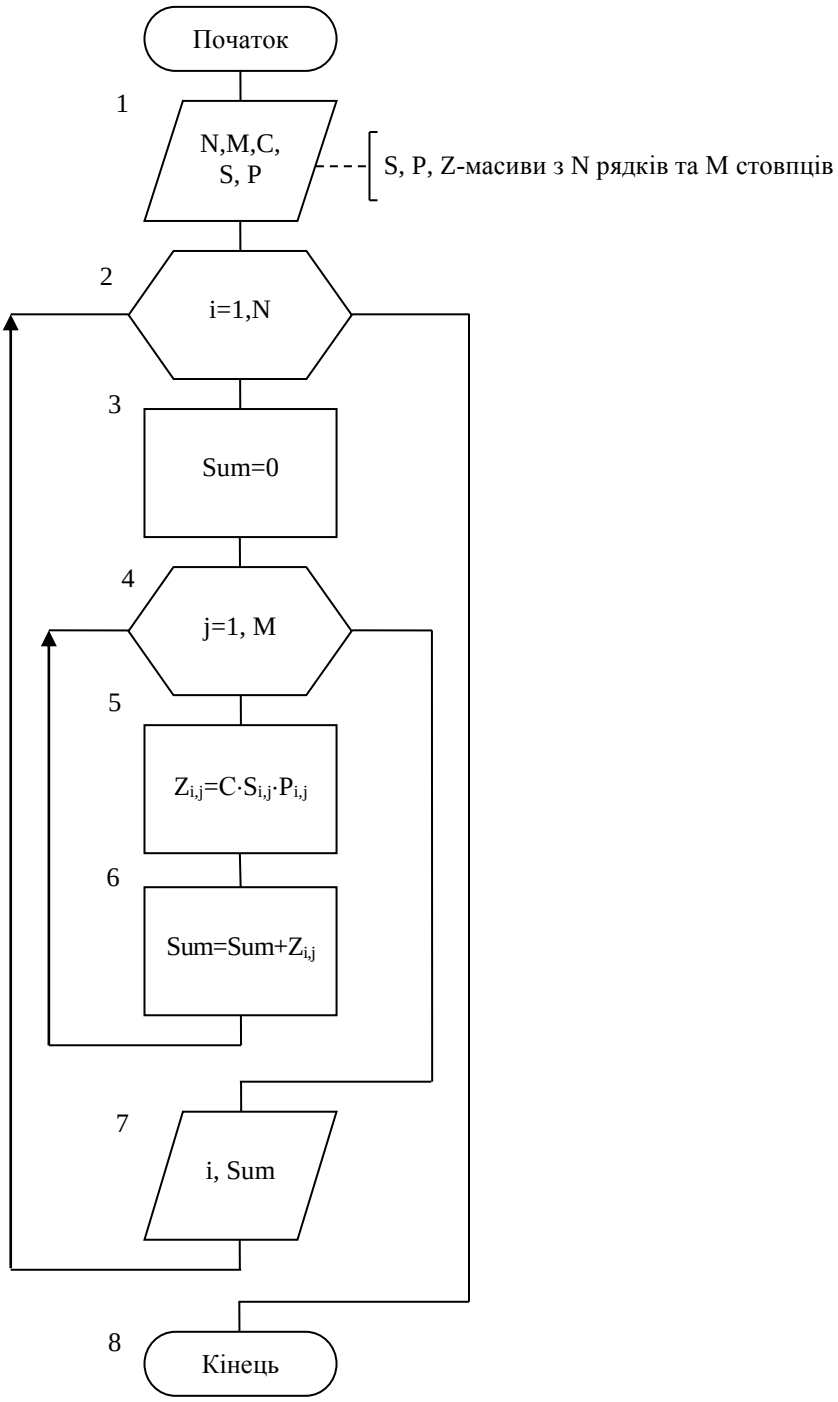
```
End Sub
Private Sub rascet_Click()
s = 0
For i = 1 To n
For j = 1 To m
s = s + 3.14 * d(i, j) ^ 2 / 4
Next j, i
p = s * v * 60
Text4 = CStr(p)
End Sub
```

TP

```
Program p35;
Var
d: array [1..10,1..10] of real;
i, j, m, n: integer; s, v, p: real;
Begin
Writeln (‘Введіть v’);
Readln (v);
Writeln (‘Введіть n, m’);
Readln (m,n);
Writeln (‘Введіть масив d’);
for i:=1 to n do
for j:=1 to m do
readln (d[i,j]);
s:=0;
for i:=1 to n do
begin
for i:=1 to m do
s:=s+(pi*sqr(d[i,j])/4);
end;
p:=s*v*60;
writeln ('p=', p:6:2);
End.
```

Задача 36.

Скласти відомість зарплати для N шоферів, належну за M днів роботи. Задано масив S, в якому для кожного водія зазначено відстань, пройдену машиною протягом кожного з M днів і масив $P_{i,j}$, в якому аналогічним чином вказаний вантаж, перевезений за j-тий день i-тим шофером. Зарплата розраховується за формулою: $Z_{i,j}=C \cdot S_{i,j} \cdot P_{i,j}$, де C - тариф за 1т/км.



C++

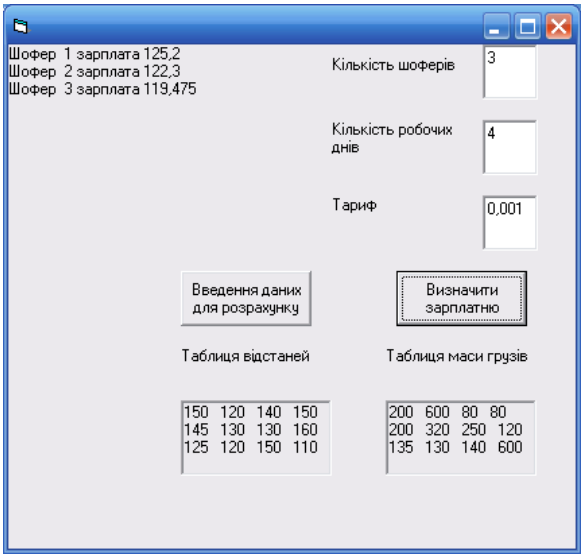
```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i,j;
	double C,s;
	cout<<"N=";cin>>N;
	cout<<"M=";cin>>M;
	cout<<"C=";cin>>C;
double **S = new double* [N+1];
	for(i=1;i<=N;i++)
		S[i] = new double [M+1];

for(i=1;i<=N;i++)
 for(j=1;j<=M;j++)
	{
		cout<<"S["<<i<<"]["<<j<<"]=";
		cin>>S[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=M;j++)
		cout<<S[i][j]<<"\t";
 cout<<endl;
}

double **P = new double* [N+1];
	for(i=1;i<=N;i++)
		P[i] = new double [M+1];

for(i=1;i<=N;i++)
```

```
 for(j=1;j<=M;j++)
          {
                    cout<<"P["<<i<<"]["<<j<<"]=";
                    cin>>P[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=M;j++)
                    cout<<P[i][j]<<"\t";
 cout<<endl;
}

double **Z = new double* [N+1];
          for(i=1;i<=N;i++)
                    Z[i] = new double [M+1];

 for(i=1;i<=N;i++)
 {
           s=0;
       for(j=1;j<=M;j++)
                    {
         Z[i][j]=C*S[i][j]*P[i][j];
                      s+=Z[i][j];
                    }
                    cout<<"Номер шофера "<<i<<" сумма "<<s<<endl;
 }


for(i=1;i<=N;i++)
          delete S[i];
delete []S;

for(i=1;i<=N;i++)
          delete P[i];
delete []P;
```

```
for(i=1;i<=N;i++)
        delete Z[i];
delete []Z;
system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim s(), p(), z(), sum, c As Single, i, j, n, m As Integer
Private Sub vvod_Click()
n = CInt(Text1)
m = CInt(Text2)
c = CSng(Text3)
ReDim s(n, m), p(n, m), z(n, m)
For i = 1 To n
For j = 1 To m
s(i, j) = CSng(InputBox("Відстань " & i & " шофера за " & j & " день"))
p(i, j) = CSng(InputBox("Вантаж " & i & " шофера за " & j & " день"))
Next j, i
matr = "": matr1 = ""
For i = 1 To n
For j = 1 To m
matr = matr + CStr(s(i, j)) + "   "
matr1 = matr1 + CStr(p(i, j)) + "   "
Next j
matr = matr + vbCrLf
matr1 = matr1 + vbCrLf
Next i
End Sub
Private Sub rascet_Click()
For i = 1 To n
sum = 0
```



```
For j = 1 To m
z(i, j) = c * s(i, j) * p(i, j)
sum = sum + z(i, j)
Next j
Print "Шофер "; i; "зарплата"; sum
Next i
End Sub
```

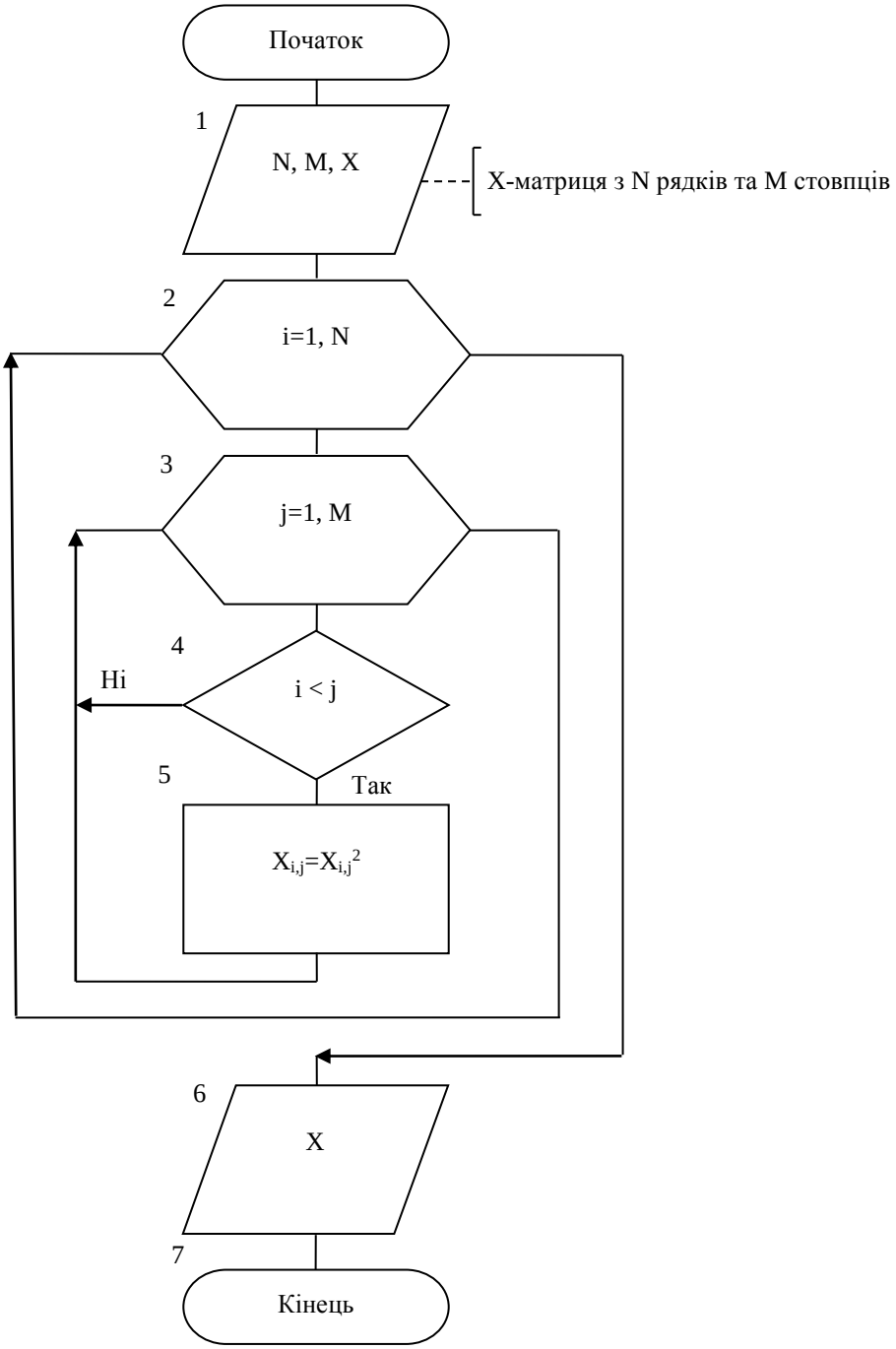
TP

```
Program p36;
Var
Z, p, s: array [1..10,1..10] of real;
i, j, m, n: integer; k, c: real;
Begin
Writeln (‘Введіть с’);
Readln (c);
Readln (m, n);
Writeln (‘Введіть масиви p та s’);
for i:=1 to n do
for j:=1 to m do
readln (p[i,j], s[i,j]);
for i:=1 to n do
begin
k:=0;
for j:=1 to m do
z[I,j]:=c*s[I,j]*p[I,j];
k:=k+z[i,j];
end;
writeln (I, 'k=',k:6:2);
End.
```

Обробка матриць з умовами по осям симетрії

Задача 37.

Заданий двовимірний масив. Всі елементи, розташовані вище головної діагоналі, піднести в квадрат.

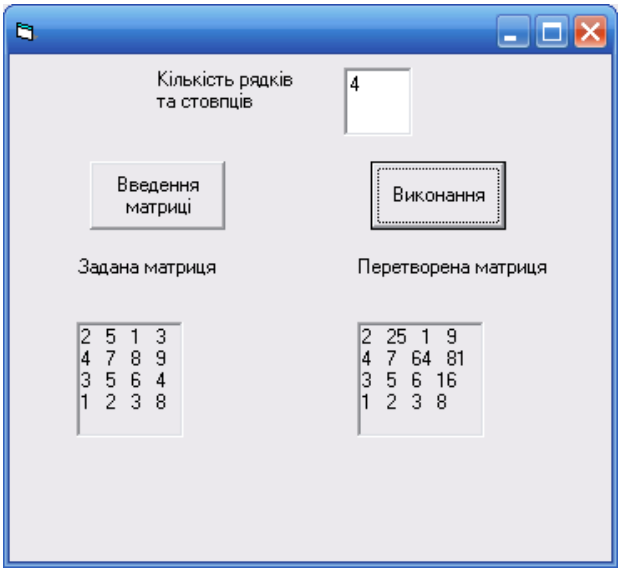


C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int N,M,i,j;
	cout<<"N=";cin>>N;
	cout<<"M=";cin>>M;
double **X = new double* [N+1];
	for(i=1;i<=N;i++)
		X[i] = new double [M+1];

for(i=1;i<=N;i++)
 for(j=1;j<=M;j++)
	{
		cout<<"X["<<i<<"]["<<j<<"]=";
		cin>>X[i][j];
}

for(i=1;i<=N;i++)
{
 for(j=1;j<=M;j++)
		cout<<X[i][j]<<"\t";
 cout<<endl;
}
```

VB

```
Option Explicit
Option Base 1
Dim x() As Single, i, j, n As Integer
Private Sub vvod_Click()
n = CInt(Text1)
ReDim x(n, n)
For i = 1 To n
For j = 1 To n
x(i, j) = CSng(InputBox("x(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To n
For j = 1 To n
matr = matr + CStr(x(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
End Sub
Private Sub rascet_Click()
For i = 1 To n
For j = 1 To n
If i < j Then x(i, j) = x(i, j) ^ 2
Next j
Next i
matr1 = ""
For i = 1 To n
For j = 1 To n
matr1 = matr1 + CStr(x(i, j)) + "   "
Next j
matr1 = matr1 + vbCrLf
Next i
End Sub
```

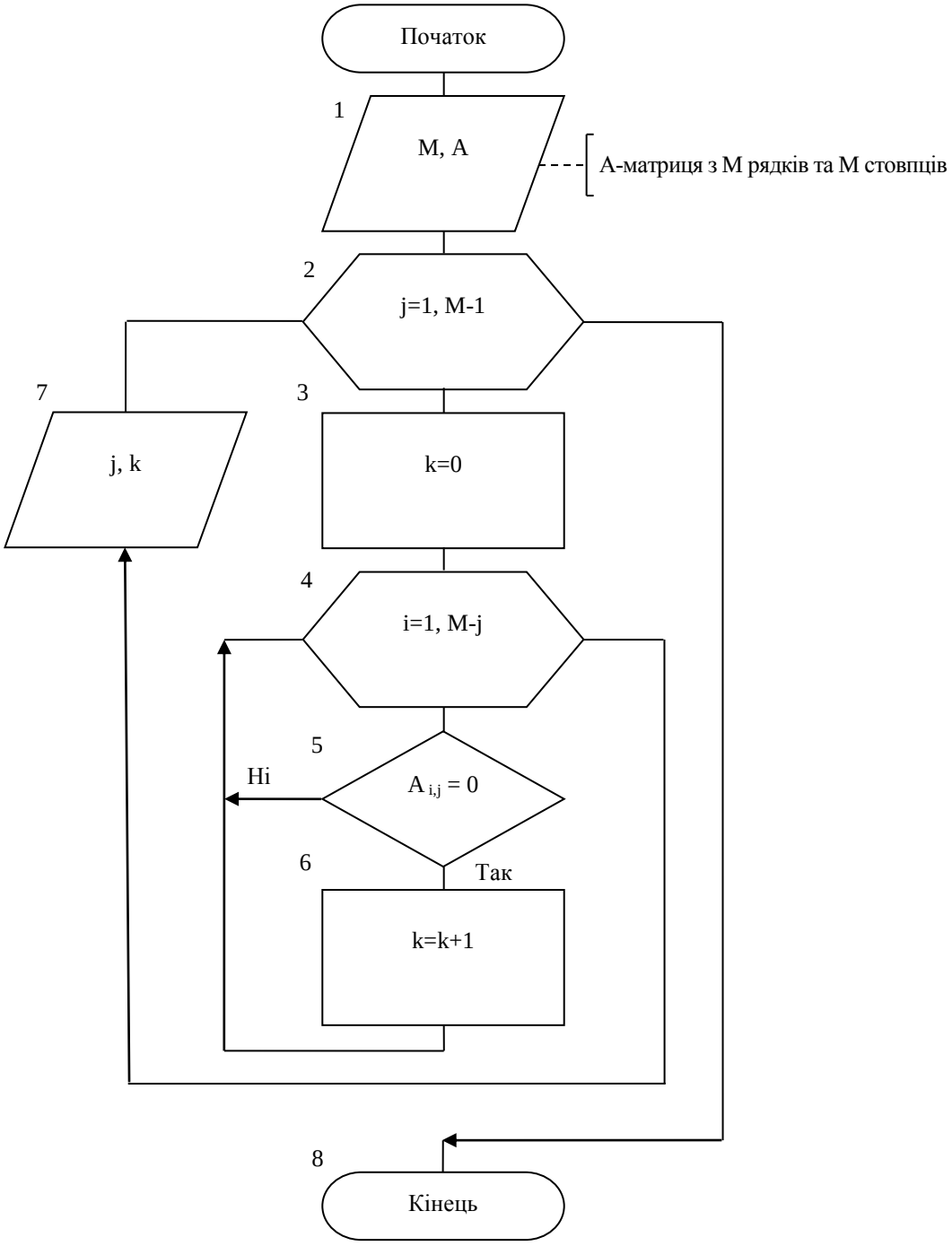


TP

```
Program p37;
Var
x: array [1..10,1..10] of real;
i, j, n, m: integer;
Begin
Writeln (‘Введіть n, m’);
readln (n, m);
Writeln (‘Введіть масив x’);
for i:=1 to n do
for j:=1 to m do
readln (x[i,j]);
for j:=1 to n do
Begin
for i:=1 to n do
if i>j then begin x[I,j]:=x[I,j]*x[i,j];
writeln ('x[I,j]',x[I,j]:6:2); end;
end;
End.
```

Задача 38.

Підрахувати кількість нульових елементів у кожному стовпці матриці $A(M,M)$ серед елементів, що лежать вище побічної діагоналі.



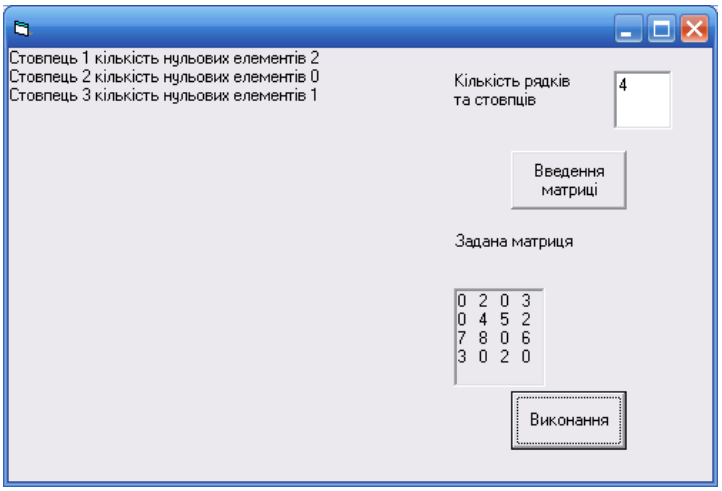
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int M,i,j,k;
	cout<<"M=";cin>>M;
double **A = new double* [M+1];
	for(i=1;i<=M;i++)
		A[i] = new double [M+1];

for(i=1;i<=M;i++)
 for(j=1;j<=M;j++)
	{
		cout<<"A["<<i<<"]["<<j<<"]=";
		cin>>A[i][j];
}

for(i=1;i<=M;i++)
{
 for(j=1;j<=M;j++)
		cout<<A[i][j]<<"\t";
 cout<<endl;
}


 for(j=1;j<M;j++)
 {
	 k=0;
   for(i=1;i<=M-j;i++)
     if(A[i][j]==0) k++;
	cout<<"Столбец: "<<j<<" количество: "<<k<<endl;
 }
```

```
for(i=1;i<=M;i++)
        delete A[i];
delete []A;

system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a() As Single, k, i, j, m As Integer
Private Sub vvod_Click()
m = CInt(Text1)
ReDim a(m, m)
For i = 1 To m
For j = 1 To m
a(i, j) = CSng(InputBox("a(" & i & "," & j & ")="))
Next j, i
matr = ""
For i = 1 To m
For j = 1 To m
matr = matr + CStr(a(i, j)) + "   "
Next j
matr = matr + vbCrLf
Next i
End Sub

Private Sub rascet_Click()
For j = 1 To m - 1
k = 0
For i = 1 To m - j
If a(i, j) = 0 Then k = k + 1
```



```
Next i
Print "Стовпець"; j; "кількість нульових
елементів"; k
Next j
End Sub
```

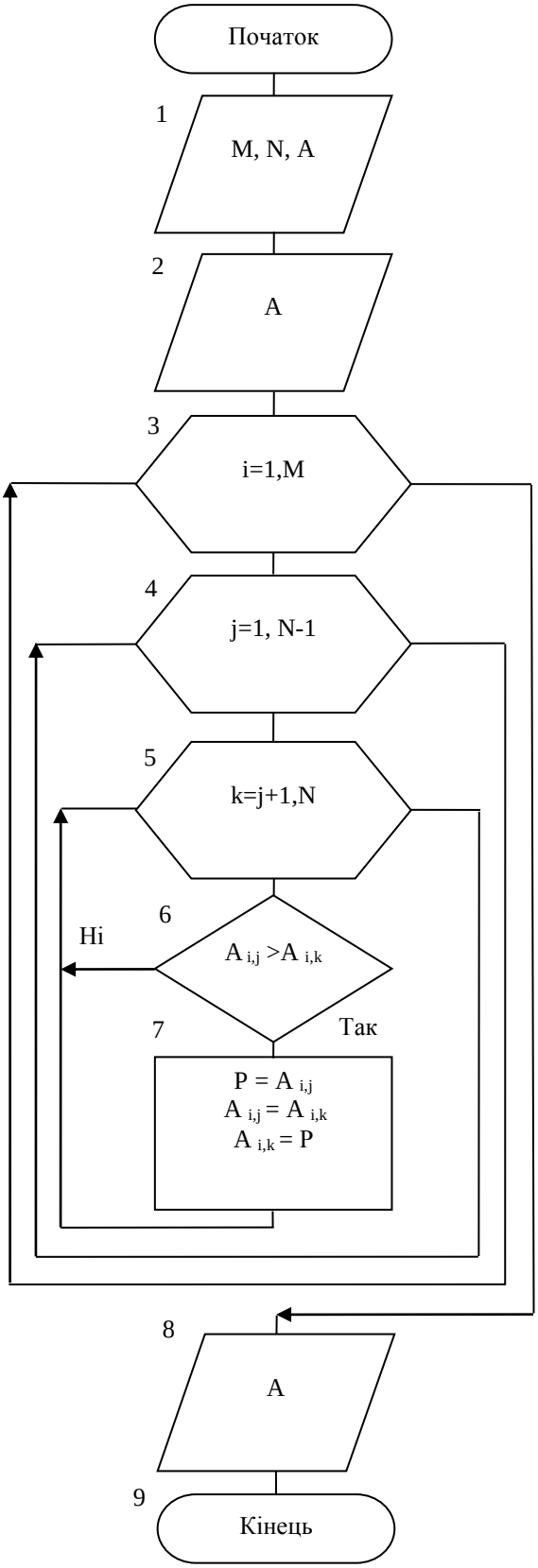
TP

```
Program p38;
Var
a: array [1..10,1..10] of real;
i, j, m, k: integer;
Begin
Readln (m);
for i:=1 to m do
for j:=1 to m do
readln (a[i,j]);
for j:=1 to m-1 do
begin
k:=0;
for i:=1 to m-j do
if a[I,j]=0 then begin k:=k+1;
writeln (j, 'k=', k);
end; end;
End.
```

Сортування матриць

Задача 39.

Розташувати елементи кожного рядка матриці $A(M,N)$ за зростанням (із застосуванням методу лінійного сортування).



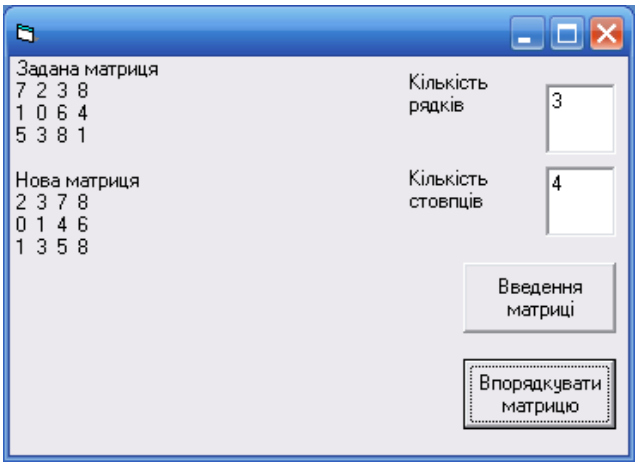
C++

```
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
	SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
	int M,N,i,j,k;
	double P;
	cout<<"M=";cin>>M;
	cout<<"N=";cin>>N;
double **A = new double* [M+1];
	for(i=1;i<=M;i++)
		A[i] = new double [N+1];

for(i=1;i<=M;i++)
 for(j=1;j<=N;j++)
	{
		cout<<"A["<<i<<"]["<<j<<"]=";
		cin>>A[i][j];
}

for(i=1;i<=M;i++)
{
 for(j=1;j<=N;j++)
		cout<<A[i][j]<<"\t";
 cout<<endl;
}



 for(i=1;i<=M;i++)
   for(j=1;j<N;j++)
		for(k=j+1;k<=N;k++)
	if(A[i][j]>A[i][k])
		  {
```

```
                        P=A[i][j];
            A[i][j]=A[i][k];
            A[i][k]=P;
                    }

cout<<endl<<endl;
for(i=1;i<=M;i++)
{
 for(j=1;j<=N;j++)
                cout<<A[i][j]<<"\t";
 cout<<endl;
}

for(i=1;i<=M;i++)
        delete A[i];
delete []A;

system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Option Base 1
Dim a() As Single, p As Single, i, j, k, m, n As
Integer
Private Sub vvod_Click()
m = CInt(text1)
n = CInt(text2)
ReDim a(m, n)
For i = 1 To m
For j = 1 To n
a(i, j) = CSng(InputBox("a(" & i & "," & j &
")="))
Next j
Next i
End Sub
```



```
Private Sub rascet_Click()
Print " Задана матриця"
For i = 1 To m
For j = 1 To n
Print a(i, j);
Next j
Print
Next i
Print
For i = 1 To m
For j = 1 To n - 1
For k = j + 1 To n
If a(i, k) < a(i, j) Then p = a(i, k): a(i, k) = a(i, j):
a(i, j) = p
Next k, j, i
Print " Нова матриця"
For i = 1 To m
For j = 1 To n
Print a(i, j);
Next j
Print
Next i
End Sub
```

TP

```
Program p39;
Var
a: array [1..10,1..10] of real;
i, j, m, n, k: integer; p: real;
Begin
Readln (m, n);
for i:=1 to m do
for j:=1 to n do
readln (a[i,j]);
for i:=1 to m do
begin
```

```
for j:=1 to n-1 do
for k:=j+1 to n do
if a[I,j]> a[I,k] then begin
p:=a[I,j];
a[I,j]:= a[I,k];
a[I,k]:= p;
end; end;
for i:=1 to m do
begin
for j:= 1 to n do
write (a[I,j]:6:2);
writeln;
end;
End.
```

4.2.4 Ітераційні цикли

Накопичення суми ряда

Задача 40.

Обчислити суму членів ряду.

$$\tilde{o} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \ldots + (-1)^{n-1}\frac{x^n}{n!} + \ldots$$

Обчислення вести до тих пір, поки абсолютна величина різниці між двома сусідніми членами ряду не стане менше деякої малої величини ε ($10^{-3}>\varepsilon>10^{-4}$).

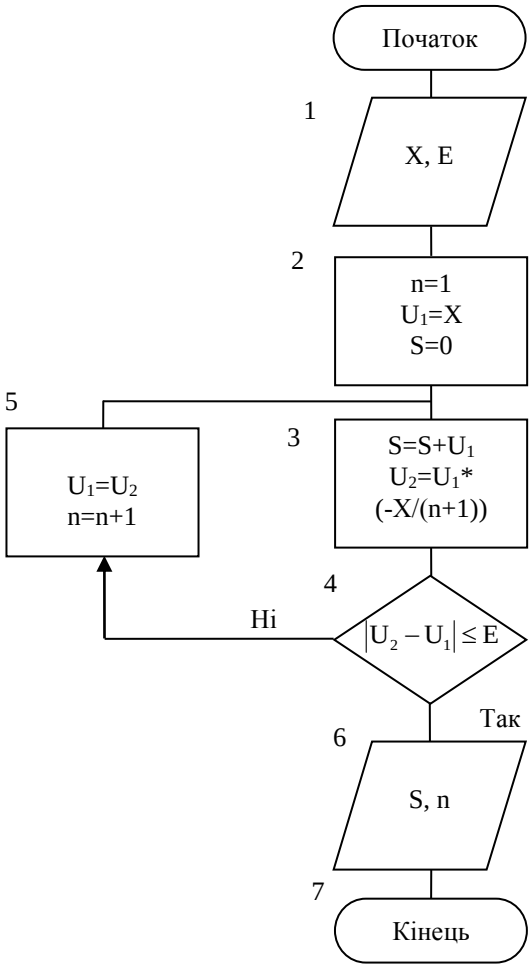
X – аргумент.

E – точність.

U1, U2 – попередній та наступний члени ряда (сусіди).

S – сума членів ряда.

n – номер члена ряда.



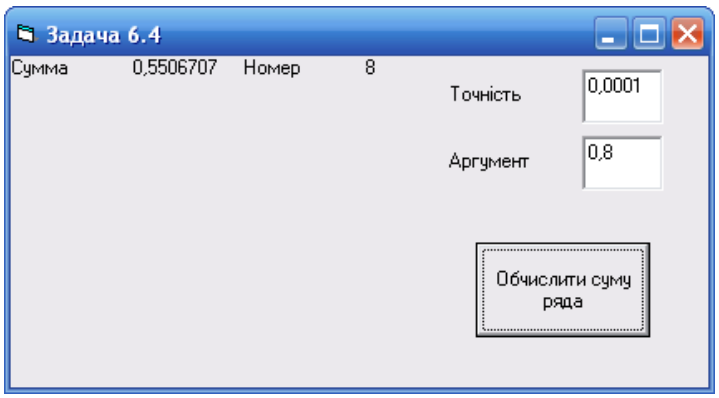
C++

```
#include<cmath>
#include <iostream>
#include<Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
        int n,S;
        double X,E,U1,U2;
        cout<<"X=";cin>>X;
        cout<<"E=";cin>>E;


n=1;
U1=X;
```

```
S=0;

do
{
        S+=U1;
        U2=U1*(-X/(n+1));
        U1=U2;
        n++;
}
 while(fabs(U2-U1)<=E);

 cout<<"S="<<S<<endl;
 cout<<"n="<<n<<endl;

system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Dim n As Integer, x, e, s, u1, u2, u As Single
Private Sub rascet1_Click()
e = CSng(Text1)
x = CSng(Text2)
u2 = x: n = 1: s = u2
Do
u1 = u2
u2 = -u1 * x / (n + 1)
s = s + u2
n = n + 1
Loop Until Abs(u1 - u2) < e
Print "Сумма", s, "Номер", n
End Sub
```



TP

```
Program p40;
Var
X, e, u1, u2, s: real;
N: integer;
Begin
Writeln (‘Введіть x, e’);
Readln (x, e);
N:=1;
U1:=x;
S:=0;
Repeat
S:=s+u1;
U2:=u1*(-x/(n+1));
U1:=u2;
N:=n+1;
Until abs(u2-u1)<=e;
Writeln (‘s=’, s:8:3,’n=’,n);
End.
```

Знаходження коренів алгебраїчних рівнянь методом ітерацій

Задача 41.

Знайти корені алгебраїчного рівняння $X^5 - 3X + 1 = 0$ використовуючи методи:

а) простої ітерації;

б) метод половинного ділення;

в) метод Рибакова.

В алгоритмах розв’язання задачі використані наступні змінні:

x_1 – ліва межа інтервалу знаходження кореня рівняння;

x_2 – права межа інтервалу знаходження кореня рівняння;

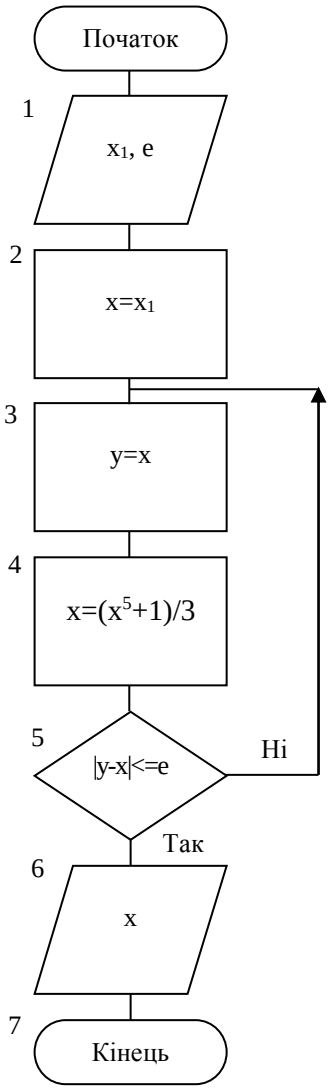
e – бажана точність визначення кореня;

f1, f2 – значення функції у спеціально підібраних точках;

d – крок збільшення аргументу;

M – деяке число, що замінює похідну функції $f(x_i)$.

Рішення а)



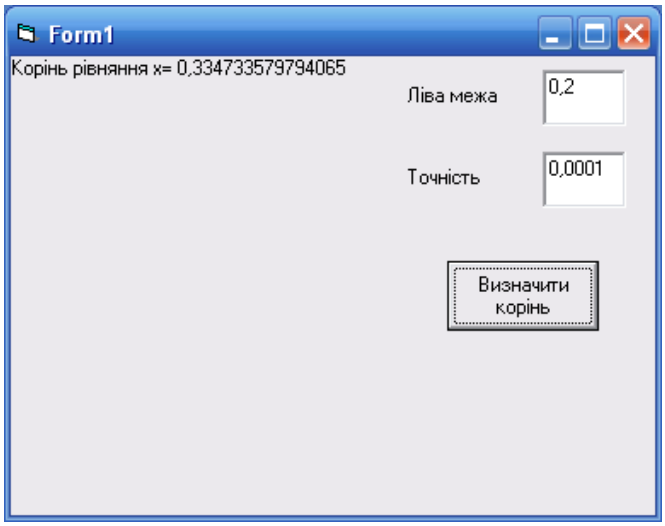
C++

```
#include<iostream>
#include<clocale>
#include<cmath>
using namespace std;
int main()
{
        double y,x,x1,e;
        cout<<"Введіть ліву межу інтервалу";
        cin>>x1;
   cout<<"Введіть точність";
        cin>>e;
        x=x1;
        do
```

```
        {
                y=x;
                x=(pow(x,5)+1)/3;
        }
        while (fabs(y-x)<=e);

        cout<<"x="<<x<<endl;

        system("pause");
        return 0;
}
```

VB

```
 Option Explicit
Dim x, x1, e, y As Single
Private Sub Command1_Click()
x1 = CSng(Text1)
e = CSng(Text2)
x = x1
Do
y = x
x = (x ^ 5 + 1) / 3
Loop Until Abs(y - x) <= e
Print "Корінь рівняння x="; x
End Sub
```

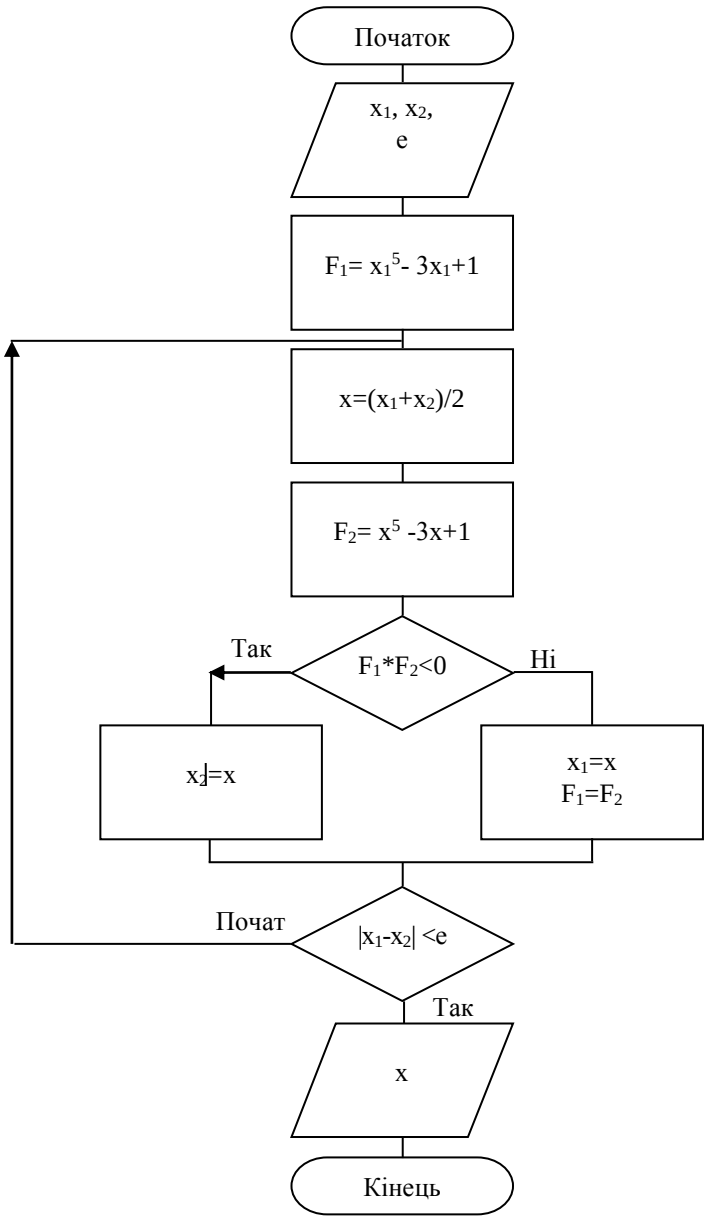


TP

```
 Program z_41;
    Var  x,x1,y,e:real;
      Begin
        writeln('Vvedite x1 i E') ;
        readln(x1,e);
       x:=x1;
```

```
 repeat
   y:=x;
   x:=(exp(5*ln(x)+1))/3;
 until abs(y-x)<=e;
writeln('x=  ',x:3:3);readln;
end.
```

Рішення б)

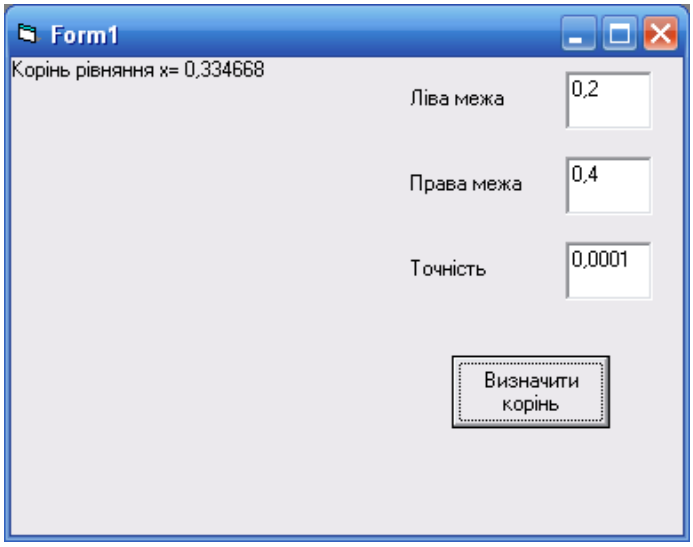


C++

```
#include<iostream>
#include<clocale>
#include<cmath>
using namespace std;
int main()
{
	double x,x1,x2,e,f1,f2;
	cout<<"Введіть ліву межу інтервалу";
	cin>>x1;
	cout<<"Введіть праву межу інтервалу";
	cin>>x2;
   cout<<"Введіть точність";
	cin>>e;
	f1=pow(x1,5)-3*x1+1;
	do
	{
		x=(x1+x2)/2;
		f2=pow(x,5)-3*x+1;
		 if(f1*f2<0) x2=x;
		  else
		   {
			x1=x;
		     f1=f2;
		    }

	}
	while (fabs(x1-x2)<e);

	cout<<"x="<<x<<endl;

	system("pause");
	return 0;
}
```

VB

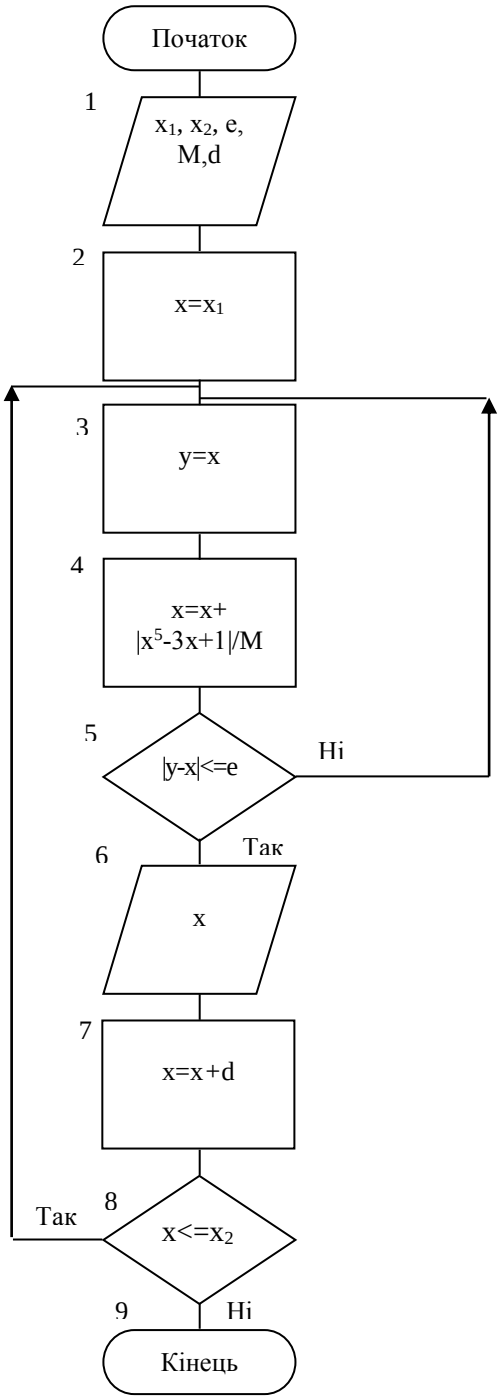
```
Option Explicit
Dim x, x1, x2, f1, f2, e As Single
Private Sub Command1_Click()
x1 = CSng(Text1)
x2 = CSng(Text2)
e = CSng(Text3)
f1 = x1 ^ 5 - 3 * x1 + 1
Do
x = (x1 + x2) / 2
f2 = x ^ 5 - 3 * x + 1
If f1 * f2 < 0 Then
x2 = x
Else
x1 = x: f1 = f2
End If
Loop Until Abs(x1 - x2) <= e
Print "Корінь рівняння x="; x
End Sub
```



TP

```
Program z_41b;
  Var  x,x1,x2,f1,f2,e:real;
   Begin
    writeln('Vvedite x1,x2 i E') ;
    readln(x1,x2,e);
   f1:=exp(5*ln(x1))-3*x1+1;
    repeat
      x:=(x1+x2)/2;
      f2:=exp(5*ln(x))-3*x+1;
        if (f1*f2)<0 then x2:=x else begin x1:=x;f1:=f1;end;
   until abs(x1-x2)<e;
  writeln('x=  ',x);readln;
 end.
```

Рішення в)



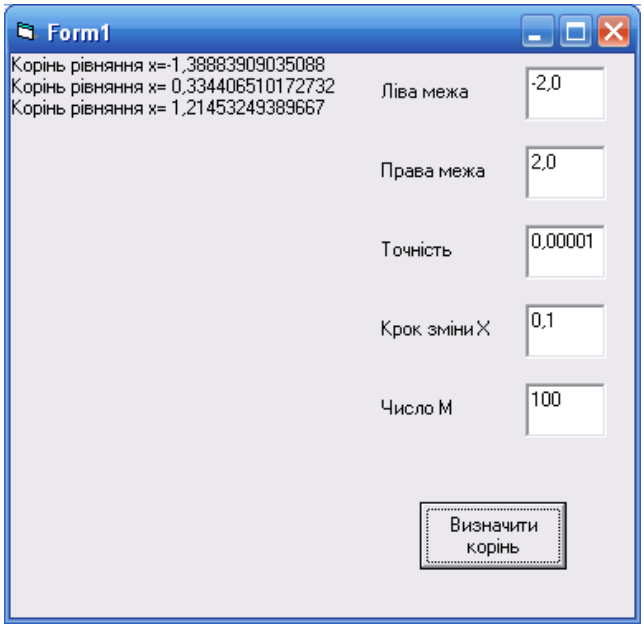
C++

```
#include<iostream>
#include<clocale>
#include<cmath>
using namespace std;
int main()
{
```

```
        double y,x,x1,x2,e,f1,f2,M,d;
        cout<<"Введіть ліву межу інтервалу";
        cin>>x1;
        cout<<"Введіть праву межу інтервалу";
        cin>>x2;
   cout<<"Введіть точність";
        cin>>e;
        cout<<"Введіть крок зміни X";
        cin>>d;
        cout<<"Введіть число M";
        cin>>M;
        x=x1;
        do
        {
run:
                y=x;
                x+=fabs(pow(x,5)-3*x+1)/M;

        if (fabs(y-x)<=e || x>=x2)
        {
        cout<<"x="<<x<<endl;
        x+=d;
        }
        else goto run;
        }
        while(x<=x2);



        system("pause");
        return 0;
}
```

VB

```
Option Explicit
Dim x, x1, x2, e, y, d As Single, M As Integer
Private Sub Command1_Click()
x1 = CSng(Text1)
x2 = CSng(Text2)
e = CSng(Text3)
d = CSng(Text4)
M = CSng(Text5)
x = x1
Do
Do
y = x
x = x + Abs(x ^ 5 - 3 * x + 1) / M
Loop Until Abs(y - x) <= e Or x >= x2
If x <= x2 Then Print "Корінь рівняння x="; x
x = x + d
Loop Until x > x2
End Sub
```



TP

```
Program z_41b;
  Var  x,x1,x2,f1,f2,e:real;
    Begin
      writeln('Vvedite x1,x2 i E') ;
      readln(x1,x2,e);
     f1:=exp(5*ln(x1))-3*x1+1;
      repeat
        x:=(x1+x2)/2;
        f2:=exp(5*ln(x))-3*x+1;
          if (f1*f2)<0 then x2:=x else begin x1:=x;f1:=f1;end;
     until abs(x1-x2)<e;
    writeln('x=  ',x);readln;
   end.
```

Навчальне видання

Азарян Альберт Арамаісович, **Карабут** Надія Олександрівна, **Козикова** Таїсія Павлівна,
Трачук Аннаіт Альбертівна, **Рибальченко** Олена Геннадієвна, **Шаповалова** Нонна Наілєвна

ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГАМУВАННЯ МОВАМИ C++, VISUAL BASIC, TURBO PASCAL

Навчальний посібник

Дизайн обкладинки Н.Н. Шаповалова
Редактор
Комп’ютерне верстання

Підп. до друку.
Формат . Папір офс. Гарнітура Times New Roman Cyr. Друк офс.
Ум. друк. арк. Обл.-вид. арк.
Тираж пр. Вид. №
Зам. №

Видавництво
Свідоцтво про внесення суб’єкта видавничої справи до Державного реєстру ДК № від.
Надруковано у друкарні ОктанПринт
.