

Циклова комісія Комп'ютерна інженерія

Курс лекцій з дисципліни «Технології (Програмування)»

Розробив: _____ О.В.Старосельцева
Розглянуто та схвалено
на засіданні предметної (циклової)
комісії комп'ютерної інженерії
Протокол № 1 від 01.09.2022 р.
Голова комісії _____ О.В. Старосельцева

Зміст

№	Тема лекції
1	Поняття та властивості алгоритму. Способи опису алгоритмів. Структура C++ програми. Директива препроцесора <code>include</code> . Призначення та принцип дії препроцесора, компілятора, лінкера.
2	Ідентифікатори та зарезервовані слова, змінні в мові C/C++. Числові типи даних. Операції над числовими типами даних, арифметичні вирази, введення-виведення даних.
3	Види управляючих структур. Блоки та складені оператори. Оператори розгалуження <code>if</code> та <code>if-else</code>
4	Види управляючих структур. Оператор-перемикач <code>switch</code> .
5	Ітераційний та регулярний цикли. Управління циклами за допомогою <code>break</code> , <code>continue</code> . Оператор <code>exit</code> .
6	Поняття масиву даних. Одномірні масиви даних. Алгоритми пошуку екстремумів одномірних масивів даних.
7	Алгоритми сортування даних в одномірних масивах даних.
8	Багатомірні масиви даних. Алгоритми обробки матриць.
9	Структури в C/C++. Масиви структур.
10	Динамічний розподіл пам'яті у C++. Вказівники. Формування та обробка динамічних масивів даних.
11	Основи створення та використання функцій в C++. Обмін даними між функціями. Передача параметрів «за значенням».
12	Обмін даними між функціями. Передача параметрів та «за посиланням».
13	Поняття рекурсії. Реалізація рекурсії. Рекурсивні математичні функції.
14	Файлова система в мові C++. Основні функції роботи з файлами.
15	Функції роботи з файлами послідовного доступу.
16	Організація файлового введення-виведення з використанням класів <code>ifstream</code> , <code>ofstream</code> та <code>fstream</code> .
17	Функції роботи з бінарними файлами.
18	Робота з файлами структур

Лекція № 1

з навчальної дисципліни Технології (Програмування)

Тема Поняття та властивості алгоритму. Способи опису алгоритмів.
Структура C++ програми. Директива препроцесора `include`.
Призначення та принцип дії препроцесора, компілятора, лінкера.
Структура C/C++ програми. Директива препроцесора `include`.
Призначення та принцип дії препроцесора, компілятора, лінкера.

Мета заняття Ввести поняття алгоритму. Дати характеристики властивостям алгоритму. Показати способи опису алгоритмів.
Ознайомити студентів із структурою програми на мові C/C++, охарактеризувати призначення та принцип дії препроцесора, компілятора, лінкера, визначити принципи дії директиви препроцесора `include`, дати характеристики функції `main()`.

Час – 2 години

План проведення лекції

- 1 Поняття алгоритму.
- 2 Властивості алгоритму.
- 3 Способи опису алгоритмів
- 4 Структура C++ програми
- 5 Призначення та принцип дії препроцесора, компілятора, лінкера (компонувальника).
- 6 Директива препроцесора `include`. Функція `main()`.

Навчальні матеріали лекції

1 Поняття алгоритму

Люди щоденно користуються різноманітними правилами, інструкціями, рецептами тощо, що складаються з певної послідовності команд (вказівок). Деякі з них настільки увійшли до нашого життя, що ми виконуємо їх майже не замислюючись, іноді кажуть, автоматично. Так для приготування яєчні потрібно виконати послідовність команд:

1. Поставити сковороду на плиту.
2. Покласти на сковороду шматочок вершкового масла.
3. Увімкнути конфорку.
4. Чекати, поки масло на сковороді розтане.
5. Розбити два яйця і вилити їх вміст на сковороду.

6. Посолити.
7. Чекати, поки загусне білок.
8. Вимкнути конфорку.

Розглянемо алгоритми розв'язування такої задачі.

Задача 1. Є посудина місткістю 8 л, яка заповнена рідиною, і дві порожні посудини місткістю 5 л і 3 л. Потрібно одержати в одній з посудин 1 літр рідини і повідомити в якій.

Розглянемо виконавця, який має таку систему команд:

1. Перелити рідину з однієї посудини в іншу.
2. Наповнити одну з посудин рідиною з іншої посудини.
3. Вивести повідомлення.

Для цього виконавця алгоритм розв'язування цієї задачі буде таким:

1. Наповнити 3-літрову посудину з 8-літрової.
2. Перелити з 3-літрової посудини в 5-літрову.
3. Наповнити 3-літрову посудину з 8-літрової.
4. Наповнити 5-літрову посудину з 3-літрової.
5. Вивести повідомлення: «1 л одержано в 3-літровій посудині».

Такі послідовності команд (вказівок) називають алгоритмами. **Алгоритм** – це скінченна послідовність команд (вказівок), що визначає, які дії та в якому порядку потрібно виконати, щоб досягти поставленої мети. Кожна команда алгоритму є спонукальним реченням, що вказує, яку дію має виконати виконавець алгоритму. Виконавцем алгоритму може бути людина, тварина, автоматичний пристрій, такий як робот, верстат з програмним керуванням, електронна обчислювальна машина тощо. Множину всіх команд, які може виконувати даний виконавець, називають **системою команд цього виконавця**. Наприклад, у систему команд виконавця, що виконуватиме другий з наведених вище алгоритмів, повинні входити такі команди:

Слово алгоритм походить від імені видатного вченого середньовічного Сходу Мухаммеда бен-Муси альХорезмі (783–850), який у своїх наукових працях з математики, астрономії та географії описав і використовував індійську позиційну систему числення, а також сформулював у загальному вигляді правила виконання чотирьох основних арифметичних дій: додавання, віднімання, множення та ділення. Європейські вчені ознайомилися з його працями завдяки перекладам їх на латину. Під час перекладу ім'я автора було подано як *Algorithmus*. Звідси й пішло слово алгоритм. А розроблені ним правила виконання арифметичних дій вважають першими алгоритмами.

Однак пізніше під словом алгоритм стали розуміти правила знаходження найбільшого спільного дільника, які були викладені ще в працях великого

давньогрецького математика Евкліда (III ст. до н.е.). За наших часів поняття алгоритму було узагальнено, і словом "алгоритм" стали позначати опис будь-якої послідовності дій. Поняття алгоритму є одним із фундаментальних у сучасній математиці й інформатиці.

2 Властивості алгоритму

Властивостями алгоритму є **дискретність, визначеність, виконуваність, скінченність, результативність і масовість**.

Дискретність (лат. discretus – розділений, розривний) алгоритму означає, що його виконання зводиться до виконання окремих дій (кроків) у певній послідовності. Причому, кожна команда алгоритму повинна виконуватися за скінченний інтервал часу.

Визначеність (або детермінованість (лат. determinans – визначальний)) алгоритму означає, що для заданого набору значень початкових (вхідних) даних алгоритм однозначно визначає порядок дій виконавця та результат цих дій. Алгоритм не повинен містити команди, які можуть сприйматися виконавцем неоднозначно, наприклад «Узяти 2–3 ложки цукру», «Трохи підігріти молоко», «Вимкнути світло через кілька хвилин», «Поділити число x на одне з двох даних чисел a або b » тощо. Крім того, в алгоритмах недопустимі ситуації, коли після виконання чергової команди виконавцю незрозуміло, яку команду він повинен виконувати наступною.

Виконуваність алгоритму означає, що алгоритм, призначений для певного виконавця, може містити тільки команди, які входять до системи команд цього виконавця. Так, наприклад, алгоритм для виконавця «Учень першого класу» не може містити команду «Побудуй бісектрису даного кута», хоча така команда може бути в алгоритмі, який призначений для виконавця «Учень восьмого класу». Зазначимо, що **виконавець повинен лише вміти виконувати кожну команду зі своєї системи команд, і не важливо, розуміє він її чи ні. Говорять, що виконання алгоритмів виконавцем носить формальний характер: виконавець може не розуміти жодну з команд, може не знати мети виконання алгоритму, і все одно отримає результат**. Так, наприклад, верстат з програмним керуванням не розуміє жодної з команд, які він виконує, але завдяки своїй конструкції успішно виготовляє деталі.

Скінченність алгоритму означає, що його виконання закінчиться після скінченної (можливо, досить великої) кількості кроків і за скінченний час при будь-яких допустимих значеннях початкових даних. Наведені вище послідовності команд є скінченними, а наступна послідовність команд – нескінченна:

1. Взяти число 2.
2. Помножити взяте число на 10.
3. Додати до одержаного числа 5.

4. Якщо одержане число додатне, то виконати команду 3, якщо ні, то припинити виконання алгоритму.

Результативність алгоритму означає, що після закінчення його виконання обов'язково одержуються результати, які відповідають поставленій меті. Результативними вважаються також алгоритми, які визначають, що дану задачу не можна розв'язати або дана задача не має розв'язків при заданому наборі початкових даних.

Масовість алгоритму означає, що алгоритм може бути застосований до цілого класу однотипних задач, для яких спільними є умова та хід розв'язування та які відрізняються тільки початковими даними. Таким, наприклад, є алгоритм розв'язування квадратного рівняння, який дає змогу знайти дійсні корені квадратного рівняння з довільними дійсними коефіцієнтами або визначити, що при певних значеннях коефіцієнтів рівняння не має дійсних коренів. Масовим також є, наприклад, алгоритм побудови бісектриси довільного кута з використанням циркуля та лінійки. Однак, крім масових алгоритмів, складаються й застосовуються алгоритми, які не є масовими. Таким, наприклад, є алгоритм розв'язування конкретного квадратного рівняння (наприклад, $2x^2 + 5x + 20$) або алгоритм приготування конкретного салату (наприклад, грецького) на конкретну кількість осіб.

3 Способи опису алгоритмів

Словесний запис алгоритмів

Для зображення алгоритмів можна користуватися різними способами запису, які відрізняються ступенем наочності і точності. Деякі способи орієнтовані на виконавця - людину, інші - на виконання комп'ютером, треті є допоміжними (використовуються для полегшення міркувань). Розглянемо три способи подання алгоритмів: за допомогою звичайної мови спілкування, із використанням блок-схем і за допомогою алгоритмічної мови.

Словесний спосіб запису заснований на тій чи іншій природній мові спілкування (див. алгоритм Евкліда). Однак словесний запис алгоритму відрізняється від звичайних мовних конструкцій ретельнішим доббором слів і фраз, який не допускає повторень або двозначного тлумачення. Крім того, у запису алгоритму можуть використовуватися математичні символи і вирази.

Розглянемо словесний спосіб запису ще на одному простому прикладі. Маємо знайти модуль величини x (тобто значення $|x|$) і надати цього значення змінній y . Під час побудови алгоритму скористаємося визначенням модуля: $|x| = x$ при $x \geq 0$ і $|x| = -x$ при $x < 0$. Алгоритм можна записати у такий спосіб.

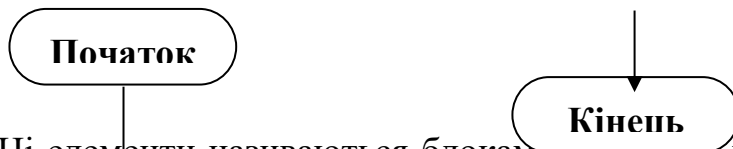
- 1 Початок.
- 2 Ввести числове значення величини x .

- 3 Якщо $x \geq 0$, то у надати значення x , інакше у надати значення $-x$.
- 4 Вивести значення у.
- 5 Кінець.

Словесний запис найчастіше застосовується на початковому етапі вивчення алгоритмів і призначається для використання алгоритму людиною. Однак ця форма запису алгоритму має два істотних недоліки. По-перше, вона недостатньо наочна і, по-друге, її важко безпосередньо перекласти мовою програми.

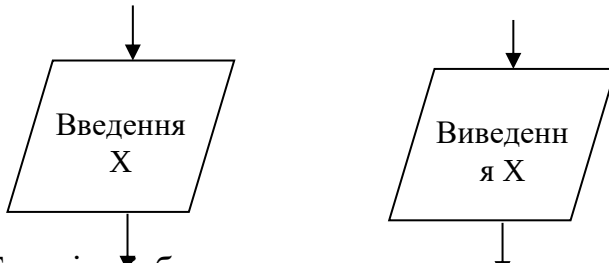
Блок-схеми алгоритмів

Наочною формою запису алгоритмів є блок-схеми, що складаються з геометричних фігур - блоків. Кожний блок відповідає певній дії. Наприклад, запис алгоритму починається і закінчується такими блоками:



Ці елементи називаються блоками початку і кінця алгоритму. Стрілки вказують напрямок виконання алгоритму. Блок **Початок** має одну вихідну стрілку, а блок **Кінець** - одну вхідну стрілку.

У алгоритмах часто використовуються команди введення і виведення значень. Цим командам відповідають блоки введення-виведення:



Тут лівий блок означає введення величини X , а правий блок - виведення X . За допомогою наведених вище блоків ви можете скласти найпростіший алгоритм введення-виведення величини X (див. рисунок 3.1).

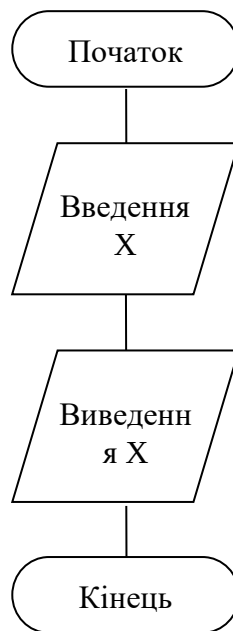


Рисунок 3.1. Блок-схема алгоритму введення виведення X.

Відповідно до цього алгоритму в програму вводиться значення величини X. Однак програма, що складається тільки з операції введення, навряд чи доцільна. Звичайно над введеною величиною виконуються певні дії, що позначаються прямокутними (операторними) блоками:



У середині прямокутників записані вирази, що виконуються над величинами. Лівий блок означає присвоювання змінній x значення суми $X+1$ (про операції присвоювання йтиметься далі). Правий блок відповідає за знаходження різниці $X-Y$ і надання значення різниці змінній Z. Операторні блоки можуть мати кілька входів і лише один вихід.

Зобразимо у вигляді блок-схеми найпростіший алгоритм обчислення квадрата якогось числа (див. рисунок 3.2):

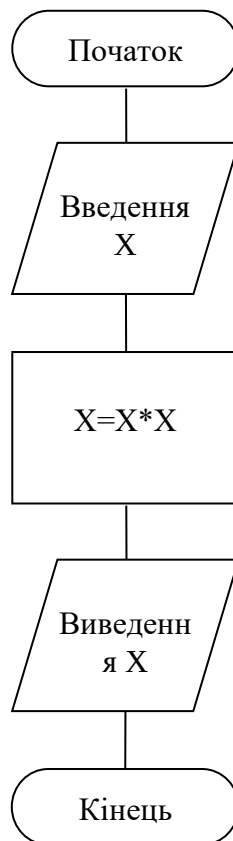
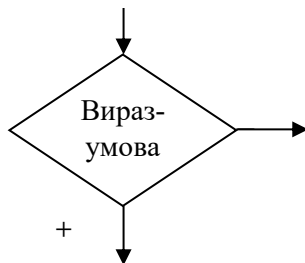


Рисунок 3.2. Блок-схема обчислення квадрата числа.

Відповідно до цього алгоритму вводиться величина X , потім обчислюється квадрат цієї величини (добуток $X \cdot X$) і виводиться отримане значення.

Усі наведені вище блоки дозволяють організувати послідовне виконання інструкцій алгоритму. Однак на практиці часто виникають ситуації, коли залежно від виконання будь-якої умови маємо змінити послідовний хід обчислень. Прикладом такої умови є нерівність $X \geq 0$ в алгоритмі знаходження модуля числа x . У схему алгоритму логічна умова вводиться за допомогою умовного блока. Цей блок прийнято зображати у вигляді ромба з одним входом і двома виходами:



Якщо умова, зазначена на зображенні блока, виконується (умова має значення **Істина**), то відбувається перехід по стрілці $+$, якщо не виконується (значення **Хибність**) - по стрілці $-$. Завдяки умовному блоку обчислювальний процес ніби розгалужується, тобто умовний блок використовується для організації розгалуження.

Наведемо приклад алгоритму обчислення модуля числа (див. рис. 3.3):

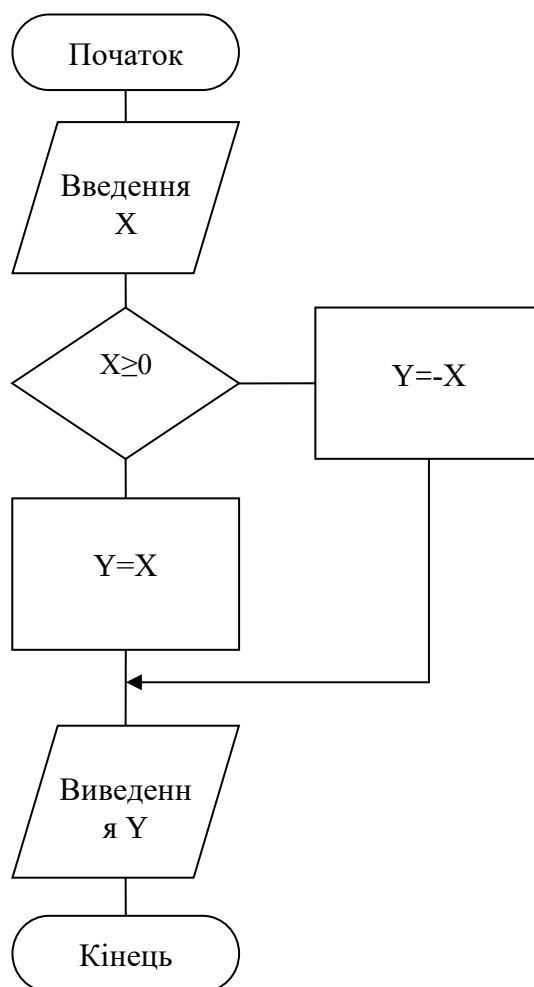


Рисунок 3.3. Блок-схема обчислення модуля числа.

Запис цього алгоритму обмежують блоки початку і кінця. За блоком початку алгоритму йде блок введення значень x , а за ним - умовний блок. В умовному блоці виконується перевірка умови $x \geq 0$ і в результаті перевірки здійснюється перехід по одній із гілок Так або Ні. На кожній із гілок розташований операторний блок надання значень змінній y . Після операції надання гілки алгоритму сходяться, і наступна інструкція алгоритму міститься в блоці виведення отриманого значення y .

Алгоритмічні мови

Блок-схеми є зручними для наочного подання алгоритмів, проте для складних алгоритмів вони стають громіздкими. Більш компактною формою подання алгоритму є його опис на алгоритмічній мові.

Алгоритмічною мовою називається штучна мова, призначена для подання алгоритмів. Алгоритмічні мови дозволяють подати алгоритм у вигляді тексту, який складається за певними правилами із застосуванням спеціальних слів, які називаються службовими. Кількість таких слів обмежена. Кожне службове слово має точно визначений смисл, призначення і спосіб застосування. Службові слова добираються так, щоб їх смисл в алгоритмічній мові перекликався із звичайним значенням слова.

Алгоритм, поданий алгоритмічною мовою, легко сприймається, тому що він читається як звичайний текст.

Алгоритмічна мова, конструкції якої однозначно перетворюються на команди комп'ютеру, називається **мовою програмування** (від грец. *programma* — розпорядження, припис, оголошення). Текст алгоритму, записаний мовою програмування, називається **програмою**.

Висновки

З алгоритмами ми зустрічаємося на кожному кроці. Вони вказують, яку послідовність дій необхідно виконати, щоб досягти потрібного результату. Кожний алгоритм розраховується на виконавця, який спроможний зрозуміти й виконати команди алгоритму. За алгоритмом процес розв'язання задачі постає як виконання скінченної послідовності окремих простих команд, кожна з яких точно визначає дії виконавця. Це розкриває шлях до створення технічних пристроїв, які сприймають і виконують алгоритми. Алгоритми подають у різних формах. Для наочного подання алгоритму використовуються блок-схеми. Подання алгоритму у вигляді тексту здійснюється за допомогою штучних мов. Навчальні алгоритмічні мови створюються для засвоєння основ складання алгоритмів. Мови програмування орієнтовані на їх «розуміння» комп'ютером, який у такому разі виступає виконавцем алгоритму.

4 Структура C / C++ програми

Сама по собі програма на мові C/C++ є текстовим файлом, в якому представлені конструкції і оператори даної мови в заданому програмістом порядку. Програма на мові C/C++ має структуру, наведену у лістингу 1.1.

Лістинг 1.1 – Загальна структура програми на мові C/C++

```
#директиви препроцесора
#директиви препроцесора
//описання функцій
функція a()
{
оператори; //будь-яка послідовність будь-яких операторів
}
функція z()
{
оператори; //будь-яка послідовність будь-яких операторів
}
int main () //функція, з якої починається виконання програми
{
оператори; //будь-яка послідовність будь-яких операторів
return 0;      //процедура завершення роботи програми
```

```
}
```

Перший крок при створенні програми – написання програмного коду у текстовому редакторі. У найпростішому випадку текстовий файл може містити інформацію, наведену у лістингу 1.2.

Лістинг 1.2 – Приклад простої програми

```
//це рядковий коментар, на екран не виводиться,  
//не впливає на роботу програми  
//приклад простої програми:  
#include <iostream.h> //директива препроцесора  
int main() //головна функція будь-якої програми, точка входу  
{  
cout<<"Hello"; //оператор виведення тексту на екран  
return 0; //процедура завершення роботи програми  
}
```

5 Призначення та принцип дії препроцесора, компілятора, лінкера (компонувальника).

Програма на мові С, так як і на більшості сучасних мов програмування, створюється в три етапи:

- препроцесорне перетворення тексту – перетворення тексту програми до її компіляції;
 - трансляція програми – переведення тексту вашої програми в машинні коди;
 - компоновка – збірка різних частин програми та підключення стандартних бібліотек.

Схема створення програми на С представлена на рисунку 2.1.

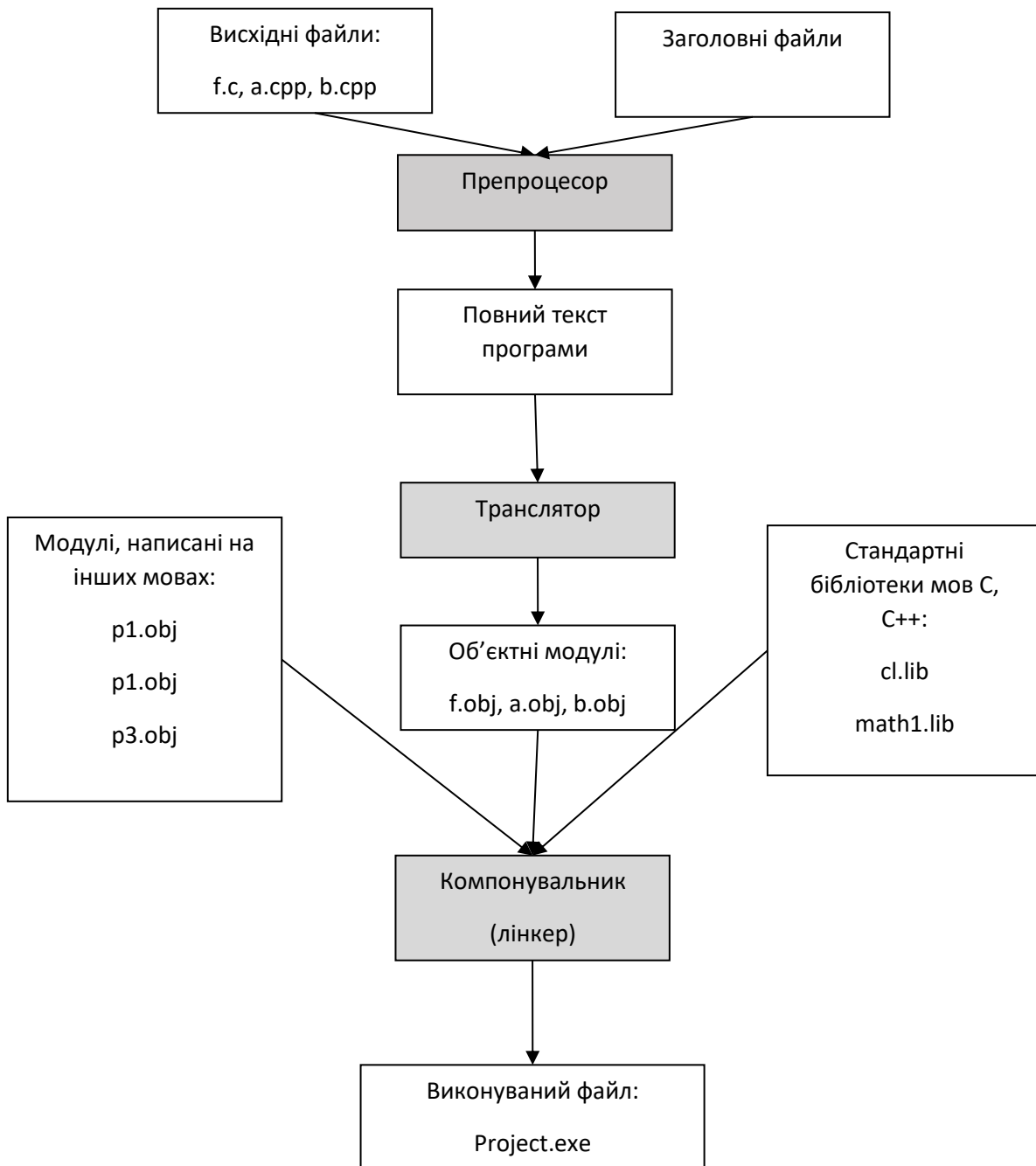


Рисунок 2.1. Послідовність етапів створення програми на С.

Завдання препроцесора – перетворення тексту програми до її компіляції. Правила препроцесорної обробки визначає програміст за допомогою директив препроцесора. Директиви препроцесора – управляють процесом перетворення тексту програми до її компіляції. Директива починається з символу "#".

Наприклад,

1) #define – указує правила заміни в тексті;

#define ZERO 0.0

Означає, що кожне використання в програмі імені ZERO замінюватиметься на 0.0.

2) `#include <ім'я заголовного файлу>` – призначена для включення в текст програми посилання на заголовний файл – стандартну бібліотеку (див. таблицю 2.1). Кожна бібліотечна функція C/C++ має відповідний опис в одному із заголовних файлів. Список заголовних файлів визначений стандартом мови. Вживання директиви `include` не підключає відповідну стандартну бібліотеку, а тільки дозволяють вставити в текст програми опис з вказаного заголовного файлу. Підключення коду бібліотеки здійснюється на останньому етапі – етапі компоновки, тобто після компіляції. Хоча в заголовних файлах містяться всі описи стандартних функцій, в код програми включаються тільки ті функції, які використовуються в програмі.

Після виконання препроцесорної обробки в тексті програми не залишається жодної препроцесорної директиви.

Наступний крок – це *компіляція початкового коду*. Під *компіляцією* розуміють процес, при якому вміст текстового файлу перетвориться у виконуваний машинний код, який розуміється процесором комп'ютера. Проте компілятор створює не готову до виконання програму, а тільки об'єктний код (файл з розширенням `*.obj`). Цей код є проміжним етапом при створенні готової програми. Справа у тому, що створювана програма може містити функції стандартних бібліотек мови C/C++, реалізації яких описані в об'єктних файлах бібліотек. Наприклад, в приведеній програмі використовується функція `cout` стандартної бібліотеки `«iostream.h»`. Це означає, що об'єктний файл `ex1.obj` міститиме лише інструкції по виклику даної функції, але код самої функції в ньому буде відсутній.

Для того, щоб підсумкова виконувана програма містила всі необхідні реалізації функцій, використовується компонувальник об'єктних кодів.

Компонувальник – це програма, яка об'єднує в єдиний виконуваний файл об'єктні коди створеної програми, об'єктні коди реалізацій бібліотечних функцій і стандартний код запуску для заданої операційної системи. У результаті і об'єктний файл, і виконуваний файл складаються з інструкцій машинної коди. Проте об'єктний файл містить тільки результат перекладу машинною мовою тексту програми, створеної програмістом, а виконуваний файл – також і машинний код для використовуваних стандартних бібліотечних підпрограм і для коди запуску.

6 Директива препроцесора `include`. Функція `main()`

Розглянемо детальніше приклад програми лістингу 2.2.

Перші три рядки задає коментарі, тобто зауваження, що допомагають краще зрозуміти програму. Вони призначені тільки для читання і ігноруються компілятором.

У другому рядку записана директива `#include`, яка дає команду препроцесору мови C/C++ вставити вміст файлу `"iostream.h"` на місце цього рядка при компіляції.

У третьому рядку визначена функція з ім'ям `main`, яка повертає ціле число (тип даних `int`) і не приймає ніяких аргументів (порожні дужки після імені `main` = тип даних `void`). Функція `main()` є обов'язковою функцією для всіх програм на мові C/C++ і без її наявності вже на етапі компіляції з'являється повідомлення про помилку, яке вказує на відсутність даної функції.

Обов'язковість даної функції обумовлюється тим, що вона є точкою входу в програму. В даному випадку під точкою входу розуміється функція, з якою починається і якою закінчується робота програми. Наприклад, при запуску exe-файлу відбувається активізація функції `main()`, виконання всіх операторів, що входять в неї і завершення програми. Таким чином, логіка всієї програми поміщена в цій функції. У приведеному прикладі при виклику функції `main()` відбувається виклик функції `cout`, яка виводить на екран монітора повідомлення "Hello", а потім виконується оператор `return`, який повертає нульове значення. Це число повертається самою функцією `main()` операційній системі і означає успішне завершення програми.

Фігурні дужки `{ }` служать для визначення почала і кінця тіла функції, тобто в них містяться всі можливі оператори, які описують роботу даної функції. Після кожного оператора в мові C/C++ ставиться символ `;`.

Таблиця 3.1 – Найпоширеніші заголовочні файли мови C/C++

№ з/п	Назва файлу	Призначення
1	<code><stdio.h></code>	Реалізує основні можливості вводу і виводу на мові C. Наприклад, даний файл включає функцію <code>printf()</code> для виведення даних на екран монітора та <code>scanf()</code> для зчитування даних з клавіатури
2	<code><iostream.h></code>	Реалізує основні можливості вводу і виводу на мові C++. Наприклад, даний файл включає функцію <code>cout</code> для виведення даних на екран монітора та <code>cin</code> для зчитування даних з клавіатури
3	<code><math.h></code>	Для обчислення основних математичних функцій (див. таблицю 2.2)
4	<code><iomanip.h></code>	Реалізує інструменти для роботи з форматованим виведенням
5	<code><string.h></code>	Для роботи з різними видами рядків
6	<code><vector.h></code>	Реалізує шаблон класу контейнерів <code>std::vector</code> – динамічний масив
7	<code><fstream.h></code>	Реалізує інструменти для файлового введення/виведення

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що таке алгоритм? Дайте визначення цього поняття.
- 2 Назвіть виконавців для таких алгоритмів:
 - а - спосіб розв'язання задачі, що записує на дошці вчитель;
 - б - інструкція про те, як завести автомобіль.
- 3 Назвіть відомі вам властивості алгоритмів.
- 4 Чи буде вважатися алгоритмом послідовність дій, що не приводить до будь-якого результату? Що таке результативність алгоритму?
- 5 Наведіть приклади властивості масовості алгоритму.
- 6 Назвіть відомі вам способи зображення алгоритмів.
- 7 Які переваги графічного зображення алгоритмів перед словесним записом?
- 8 Як властивість дискретності алгоритму пов'язана із зображенням алгоритму у вигляді блок-схеми?
- 9 Назвіть компоненти блок-схем алгоритмів.
- 10 Чи може умовний блок мати один вихід?
- 11 Що таке алгоритмічна мова?
- 12 З яких частин складається програма на C++?
- 13 Перерахуйте етапи створення виконуваної програми на мові C++.
- 14 Що являє собою етап обробки програм – препроцесор?
- 15 Що таке директива препроцесора?
- 16 Приведіть приклади директив препроцесора.
- 17 Як називається модуль і файл з яким розширенням отримується після роботи препроцесора?
- 18 Що являє собою етап обробки програм – компілятор?
- 19 Як називається модуль і файл з яким розширенням отримується після роботи компілятора?
- 20 Що являє собою етап обробки програм – компоновальник?
- 21 Як називається модуль і файл з яким розширенням отримується після роботи компоновальника?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Укажіть команди серед наведених речень:
 - а) Закрити вікно.
 - б) Не заважай читати.
 - с) Котра година?

d) Якщо йде дощ, візьми парасольку.

e) $+ 2 \square 5$.

f) Я живу в Києві.

2 Сформулюйте лінійні правила-алгоритми, які ви вивчали на уроках української мови, математики, інших предметів. Подайте ці алгоритми у вигляді блок-схем.

3 Виконайте алгоритм:

1. Накреслити відрізок АВ завдовжки 5 см.
2. Поставити вістря циркуля в точку А.
3. Побудувати коло, радіус якого дорівнює довжині відрізка АВ.
4. Поставити вістря циркуля в точку В.
5. Побудувати коло, радіус якого дорівнює довжині відрізка АВ.
6. Провести пряму через точки перетину побудованих кіл.

Яку б назву ви дали цьому алгоритму? Які геометричні задачі можна розв'язувати за цим алгоритмом? Складіть його блок-схему.

4 Складіть алгоритм приготування вашої улюбленої страви. Подайте його в словесній формі.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 2

з навчальної дисципліни Технології (Програмування)

Тема Ідентифікатори та зарезервовані слова, змінні в мові C/C++.
Числові типи даних: діапазони типів, константи, оголошення та
стартова ініціалізація змінних.
Операції над числовими типами даних, арифметичні вирази,
введення-виведення даних.

Мета заняття Ознайомити студентів із поняттями ідентифікатора та зарезервованих слів, поняттям змінної – правилами її опису та стартової ініціалізації. Ввести поняття типу в мовах програмування. Дати визначення числових типів даних.
Ознайомити студентів із різними типами операцій, які можуть застосовуватись у арифметичних виразах, пріоритетом операцій. Пояснити різні способи введення/виведення інформації в/з програми, вивчити принципи використання потоків введення/виведення cin, cout.

Час – 2 години

План проведення лекції

- 1 Склад мови C/C++
- 2 Типи даних, діапазони значень і займані обсяги пам'яті
- 3 Змінні.
- 4 Числові типи даних.
- 5 Операції в C/C++
 - 5.1 Унарні операції
 - 5.2 Бінарні операції.
 - 5.3 Операції порівняння
 - 5.4 Операції привласнення.
- 6 Вирази
- 7 Введення/виведення даних. Потоки введення / виведення cin та cout

Навчальні матеріали лекції

1 Склад мови C/C++

У тексті на будь-якій природній мові можна виділити чотири основні елементи: символи, слова, словосполучення та речення. Алгоритмічна мова також містить такі елементи, тільки слова називають лексемами (елементарними конструкціями), словосполучення – виразами, речення – операторами. Лексеми утворюються з символів, вирази з лексем і символів, оператори з символів виразів і лексем.

Таким чином, елементами алгоритмічної мови є:

1) *Алфавіт*, який для мови C/C++ включає:

- прописні і рядкові латинські букви і знак підкреслення;
- арабські цифри від 0 до 9;
- спеціальні знаки " { , | [] () + - / % * . \ : ; & ? < > = ! # ^;
- пробільні символи (пропуск, символ табуляції, символи переходу на новий рядок).

2) *Лексеми*. З символів формуються лексеми мови:

- ідентифікатори – імена об'єктів C/C++-програм. У ідентифікаторі можуть бути використані латинські букви, цифри і знак підкреслення, довжиною не більше 32 символів (проте значущі тільки перші 8 символів). Прописні і рядкові букви розрізняються, наприклад, PROG1, prog1 і Prog1 – три різні ідентифікатори. Першим символом повинні бути буква або знак підкреслення (але не цифра). Пропуски в ідентифікаторах не допускаються;

- ключові (зарезервовані) слова – це слова, які мають спеціальне значення для компілятора. Їх не можна використовувати як ідентифікатори;

- знаки операцій – це один або декілька символів, що визначають дію над операндами. Операції діляться на унарні, бінарні і тернарні по кількості операндів, що беруть участь в цій операції, відповідно, один операнд, 2 операнди, 3 операнди;

- константи – це незмінні величини. Існують цілі, дійсні, символьні і рядкові константи. Компілятор виділяє константу як лексему (елементарну конструкцію) і відносить її до одного з типів по її зовнішньому вигляду;

- роздільники – дужки, крапка, кома, пробільні символи.

3) *Константа* – це лексема, що представляє собою фіксоване числове, рядкове або символьне значення, яке не підлягає зміні.

Константи діляться на 5 груп:

- цілі;
- дійсні (з плаваючою крапкою);

- символні;
- рядкові;
- перерахування (enum).

Компілятор виділяє лексему і відносить її до тієї або іншої групи, а потім усередині групи до певного типу по її формі запису в тексті програми і по числовому значенню.

4) Використання коментарів у тексті програми

Коментар - це набір символів, які ігноруються компілятором. На цей набір символів, однак, накладаються наступні обмеження. Усередині набору символів, що представляє коментар, не може бути спеціальних символів, що визначають початок і кінець коментарів, відповідно (`/*` й `*/`). Коментарі в C позначаються `///
"/*...*/"`. Відзначимо, що коментарі можуть замінити як один рядок, так і трохи.

Наприклад:

```
// коментарі до програми
```

```
// початок алгоритму
```

або

`/*` коментарі можна записати в наступному виді, однак, треба бути обережним, щоб усередині послідовності, що ігнорується компілятором, не зустрілися оператори програми, які також будуть ігноруватися `*/`

Неправильне визначення коментарів:

```
/* коментарі до алгоритму /* рішення крайового завдання */ */
```

2 Типи даних, діапазони значень і займані обсяги пам'яті

Важлива відмінність мови C від інших мов є відсутність принципу умовчання, що приводить до необхідності оголошення всіх змінних, використовуваних у програмі явно разом із вказівкою відповідних їм типів.

Тип даних визначає:

- 1 Внутрішнє представлення даних в пам'яті комп'ютера.
- 2 Діапазон значень, які можуть приймати величини цього типу.
- 3 Операції і функції, які можна застосовувати до даних цього типу.

Залежно від вимог завдання програміст вибирає тип для об'єктів програми. Типи C/C++ можна розділити на прості (скалярні) і складені (див. рисунок 2.1).



Рисунок 2.1 – Структурна схема типів даних в мові C/C++

До простих типів відносять типи, які характеризуються одним значенням.

3 Змінні.

Змінна в C/C++ – іменована область пам'яті, в якій зберігаються дані певного типу. У змінної є ім'я (ідентифікатор) і значення. Ім'я служить для звернення до області пам'яті, в якій зберігається значення. Перед використанням будь-яка змінна повинна бути описана. Приклади:

Лістинг 3.1 – Приклад описання змінних

```
int a; float x;
```

Загальний вид оператора опису:

[клас пам'яті][const] тип ідентифікатор [ініціалізатор];

Ідентифікатором називається послідовність цифр і букв, а також спеціальних символів, за умови, що першою коштує буква або спеціальний символ. Для утворення ідентифікаторів можуть бути використані рядкові або прописні букви латинського алфавіту. Як спеціальний символ може використатися символ підкреслення (_). Два ідентифікатори, для утворення яких використовуються співпадаючі рядкові й прописні букви, вважаються різними.

Наприклад: `abc`, `ABC`, `A128B`, `a128b` . // різні ідентифікатори

Важливою особливістю є те, що компілятор допускає будь-яку кількість символів в ідентифікаторі, хоча значимими є перші 31 символ. Ідентифікатор створюється на етапі оголошення змінної, функції, структури й т.п. після цього його можна використати в наступних операторах розроблювальної програми. Слід зазначити важливі особливості при виборі ідентифікатора.

По-перше, ідентифікатор не повинен збігатися із ключовими словами, із зарезервованими словами й іменами функцій бібліотеки компілятора мови C.

По-друге, варто звернути особливу увагу на використання символу `(_)` підкреслення як перший символ ідентифікатора, оскільки ідентифікатори, побудовані таким чином, що, з одного боку, можуть збігатися з іменами системних функцій й (або) змінних, а, з іншого боку, при використанні таких ідентифікаторів програми можуть виявитися недопустимими.

По-третє, на ідентифікатори, використовувані для визначення зовнішніх змінних, повинні бути накладені обмеження, формовані використанням редактором зв'язків (відзначимо, що використання різних версій редакторів зв'язків або різних редакторів накладає різні вимоги на імена зовнішніх змінних).

Клас пам'яті може приймати значення: `auto`, `extern`, `static`, `register`. Клас пам'яті визначає час життя і область видимості змінної. Якщо клас пам'яті не вказаний явно, то компілятор визначає його виходячи з контексту оголошення. Час життя може бути постійним – протягом виконання програми або тимчасовим – протягом блоку.

Класи пам'яті:

- `auto` – автоматична локальна змінна. Специфікатор `auto` може бути заданий тільки при визначенні об'єктів блоку, наприклад, в тілі функції. Цим змінним пам'ять виділяється при вході в блок і звільняється при виході з нього. Поза блоком такі змінні не існують;

- `extern` – глобальна змінна, вона знаходиться у іншому місці програми (у іншому файлі або далі по тексті). Використовується для створення змінних, які доступні у всіх файлах програми;

- `static` – статична змінна, вона існує тільки в межах того файлу, де визначена змінна;

- `register` – аналогічно `auto`, але пам'ять під них виділяється в регістрах процесора. Якщо такої можливості немає, то змінні обробляються як `auto`.

Область видимості – частина тексту програми, з якої допустимий звичайний доступ до змінної. Зазвичай область видимості співпадає з областю дії. Окрім того випадку, коли у внутрішньому блоці існує змінна з таким же ім'ям.

`Const` – показує, що цю змінну не можна змінювати (іменована константа).

При описі можна привласнити змінній початкове значення (ініціалізація). Якщо при визначенні початкове значення змінним не задається явним чином, то компілятор обнуляє глобальні (`extern`) і статичні (`static`) змінні. Автоматичні (`auto`) змінні не ініціалізуються та містять "мусор".

Лістинг 3.2 – Приклад описання змінних

```
int a; //глобальна змінна
int main()
{
    int b; //локальна змінна
    extern int x; //змінна x визначена у іншому місці
    static int c; //локальна статична змінна
    a=1; //привласнювання значення глобальній змінній
    int a; //локальна змінна a
    a=2; //привласнювання значення локальній змінній
    ::a=3; //привласнювання значення глобальній змінній
    int x=4; //визначення і ініціалізація x
    return 0;
}
```

У лістингу 3.2 змінна `a` визначена поза всіма блоками. Областю дії змінної `a` є вся програма, окрім тих рядків, де використовується локальна змінна `a`.

Змінні `b` і `c` – локальні, область їх видимості – блок. Час життя різний: пам'ять під `b` виділяється при вході в блок (оскільки за замовчуванням клас пам'яті `auto`), звільняється при виході з нього. Змінна `c` (`static`) існує, поки працює програма.

Ім'я змінної повинне бути унікальним в своїй області дії.

Опис змінної може бути виконаний або як оголошення, або як визначення. Оголошення містить інформацію про клас пам'яті і тип змінної, визначення разом з цією інформацією дає вказівку виділити пам'ять. У лістингу 3.2: `extern int x;` – оголошення, а інші – визначення.

4 Числові типи даних

В C++ визначено наступні прості числові типи даних:

Цілочисельні типи

`char`
`int`
`short`
`long`
`signed`
`unsigned`

Дійсні типи

`float`
`double`
`long double`

Існує 4 специфікатори типу, які уточнюють внутрішнє уявлення і діапазон стандартних типів, таких як `char`, `int` та `double`:

- `short` (короткий);
- `long` (довгий);
- `signed` (знаковий);
- `unsigned` (беззнаковий).

Модифікатори `signed`, `unsigned`, `long` та `short` можливо задавати до цілих базових типів даних. Окрім того, модифікатори `signed` та `unsigned` можливо використовувати з типом `char`, а модифікатор `long` – з типом `double`. Всі допустимі комбінації базових типів і модифікаторів для 16- та 32-розрядних систем наведені у таблицях 1.1 та 1.2. В цих таблицях також вказані типові розміри значень в байтах і діапазони допустимих значень для кожного типу.

Не дивлячись на дозвіл, використання модифікатору `signed` для цілих типів даних збиткове, оскільки оголошення за замовчуванням визначає значення із знаком.

Різницю між цілим значенням із знаком і без нього заключено у інтерпретації старшого розряду. Якщо задано ціле значення зі знаком, C/C++ компілятор згенерує код з врахування того, що старший розряд значення використовується в якості флагоу знаку. Якщо флаг знаку дорівнює 0, число вважається додатнім, а якщо він дорівнює 1 – від'ємним. Від'ємні числа майже завжди представляються у додатковому коді. Для отримання додаткового коду всі розряди числа беруться в зворотному коді, а потім отриманий результат збільшується на одиницю.

Тип `int`.

Значеннями цього типу є цілі числа.

Розмір типу `int` не визначається стандартом, а залежить від комп'ютера і компілятора. Для 16-розрядного процесора під нього відводиться 2 байти, для 32-розрядного – 4 байти.

Якщо перед `int` розташований специфікатор `short`, то під число відводиться 2 байти, а якщо специфікатор `long`, то 4 байти. Від кількості пам'яті, що відводиться під об'єкт, залежить діапазон допустимих значень, які може приймати об'єкт:

- `short int` – займає 2 байти, отже, має діапазон -32768 ... +32767;
- `long int` – займає 4 байти, отже, має діапазон -2 147 483 648 ... +2 147 483 647.

Тип `int` співпадає з типом `short int` на 16-розрядних ПК і з типом `long int` на 32-розрядних ПК.

Модифікатори `signed` і `unsigned` також впливають на діапазон допустимих значень, які може приймати об'єкт:

- `unsigned short int` – займає 2 байти, отже, має діапазон 0 ... 65536;

– `unsigned long int` – займає 4 байти, отже, має діапазон 0 ... +4 294 967 295.

Цілі константи можуть бути:

- десятковими – константа визначається як послідовність десяткових цифр, що починається не з 0, якщо це число не 0 (наприклад, 8, 0, 192345);
- восьмирічними – це константа, яка завжди починається з 0. За 0 слідує вісімкові цифри (наприклад, 016 – десяткове значення 14, 01 – десяткове значення 1);
- шістнадцятирічними – послідовність шістнадцятирічних цифр, яким передують символи 0x або 0X (наприклад, 0xA – десяткове значення 10, 0X00F – десяткове значення 15).

Залежно від значення цілої константи компілятор по-різному представить її в пам'яті комп'ютера (тобто компілятор припише константі відповідний тип даних).

Типи з плаваючою крапкою.

Внутрішнє представлення дійсного числа складається з 2 частин: мантиси і порядку. У IBM-сумісних ПК величини типу `float` займають 4 байти, з яких один розряд відводиться під знак мантиси, 8 розрядів під порядок і 23 – під мантису. Діапазон значень $3,4E-38 \dots 3,4E+38$.

Величини типу `double` займають 8 байт, один розряд відводиться під знак мантиси, під порядок і мантису відводяться 11 і 52 розряди відповідно. Діапазон значень $1,7E-308 \dots 1,7E+308$.

Довжина мантиси визначає точність числа, а довжина порядку його діапазон.

Дійсні константи мають іншу форму внутрішнього уявлення в пам'яті комп'ютера. Компілятор розпізнає такі константи по їх вигляду.

Дійсні константи складаються з наступних частин:

- ціла частина – послідовність цифр;
- десяткова крапка;
- дробова частина – послідовність цифр;
- символу експоненти `e` або `E`;
- експоненти у вигляді цілої константи (можливо зі знаком);

Будь-яка (але не обидві відразу) з нижченаведених можуть бути опущена:

- ціла або дробова частина;
- десяткова крапка або символ `e(E)` і експонента у вигляді цілої константи.

Дійсні константи можуть мати дві форми уявлення:

- з фіксованою крапкою. Вид константи з фіксованою крапкою: [цифри].[цифри] (наприклад, 5.7, .0001, 41.).
- з плаваючою крапкою. Вид константи з плаваючою крапкою: [цифри][.][цифри]E[e+|-][цифри] (наприклад, 0.5e5, .11e-5, 5E3).

Всі допустимі значення для всіх типів наведені у таблицях 1.1 та 1.2.

Таблиця 1.1 – Допустимі комбінації базових типів і модифікаторів для 16-разрядного середовища

Тип	Розмір в байтах	Діапазон значень
int	2	-32 768...32 767
unsigned int	2	0...65 535
signed int	2	-32 768...32 767
short int	2	-32 768...32 767
unsigned short int	2	0...65 535
signed short int	2	-32 768...32 767
long int	4	-2 147 483 648...2 147 483 647
signed long int	4	-2 147 483 648...2 147 483 647
unsigned long int	4	0...4 294 967 295
float	4	3,4E-38...3,4E+38
double	8	1,7E-308...1,7E+308
long double	10	3.4E-4932...1.1E+4932

Таблиця 1.2 – Допустимі комбінації базових типів і модифікаторів для 32-разрядного середовища

Тип	Розмір в байтах	Діапазон значень
int	4	-2 147 483 648...2 147 483 647
unsigned int	4	0...4 294 967 295
signed int	4	-2 147 483 648...2 147 483 647
short int	2	-32 768...32 767
unsigned short int	2	0...65 535
signed short int	2	-32 768...32 767
long int	4	-2 147 483 648...2 147 483 647
signed long int	4	-2 147 483 648...2 147 483 647
unsigned long int	4	0...4 294 967 295
float	4	3,4E-38...3,4E+38
double	8	1,7E-308...1,7E+308
long double	10	3.4E-4932...1.1E+4932

5 Операції в C++

Знаки операцій забезпечують формування виразів. Вирази складаються з операндів, знаків операцій і роздільників (дужок). Кожен операнд є, у свою чергу, виразом або окремим випадком виразу – константою або змінною.

За кількістю операндів, які приймають участь в операції вирази діляться:

- унарні – операція з одним операндом;
- бінарні – операції з двома операндами;
- тернарні – операції з трьома операндами.

По типу виконуваної операції вирази діляться на:

- арифметичні;
- операції зсуву;
- порозрядні логічні;
- логічні;
- операції відношення;
- операції привласнювання.

5.1 Унарні операції

Унарні операції наведені у таблиці 1.1.

Таблиця 1.1 – Знаки унарних операцій

Знак операції	Призначення
&	Отримання адреси операнду
*	Звертання за адресою (розіменування)
-	Унарний мінус, змінює знак арифметичного операнду
!	Логічне заперечення (НЕ). В якості логічних значень використовується 0 – <i>хибність</i> і не 0 – <i>істина</i> , запереченням 0 буде 1, запереченням будь-якого ненульового числа буде 0
++	Збільшення на одиницю: – префіксна операція – збільшує на одиницю операнд до його використання; – постфіксна операція – збільшує на одиницю операнд після його використання. <code>int m=1, n=2;</code> <code>int a=(++m)+n; //a=4, m=2, n=2</code> –префіксна <code>int a=(m++)+n; //a=3, m=2, n=2</code> –постфіксна

Знак операції	Призначення
--	Зменшення на одиницю: – префіксна операція – зменшує на одиницю операнд до його використання; – постфіксна операція – зменшує на одиницю операнд після його використання.
sizeof	Обчислення розміру (в байтах) для об'єкту того типу, який має операнд. Має 2 форми: – sizeof вираз; – sizeof (тип). Приклади: – sizeof (float) //4 – sizeof (1.0) /*8, оскільки дійсні константи за замовчуванням мають тип double*/

5.2 Бінарні операції.

Арифметичні бінарні операції наведені у таблицях 1.2 та 1.3.

Таблиця 1.2 – Знаки бінарних арифметичних адитивних операцій

Знак операції	Призначення
+	Бінарний плюс (складання арифметичних операндів)
-	Бінарний мінус (знаходження різниці арифметичних операндів)

Таблиця 1.3 – Знаки бінарних арифметичних мультиплікативних операцій

Знак операції	Призначення
*	Знаходження добутку операндів арифметичного типу
/	Ділення операндів арифметичного типу (якщо операнди цілі, виконується цілочислене ділення з відкиданням дробової частини). Наприклад, 10.4 / 2 = 5.2 //операція над дійсним числом. 5 / 2 = 2 //операція над цілим числом, дробова частина (0,5) відкинута.

Знак операції	Призначення
	$2 / 5 = 0$ //операція над цілим числом, дробова частина (0,4) відкинута.
%	Отримання залишку від ділення цілочислених операндів. <i>Формат $x\%y$</i> Результат дорівнює залишку від ділення x на y , і відповідно 0, якщо x ділиться на y без залишку. Наприклад, $8\%3 = 2$ $4\%2 = 0$ $5 \% 2 = 1$ $3\%8 = 3$ <i>Окремий випадок: $x\%2$ – перевірка на парність/непарність числа x. Якщо залишок дорівнює 0, число парне, якщо не 0 – непарне.</i>

5.3 Операції порівняння

Операції порівняння (відношення) наведені у таблиці 1.6.

Результатом усякої операції відношення є числове значення типу `int`, рівне одиниці, якщо порівняння істинно, і нулю в протилежному випадку. Таким чином, операції відношення у внутрішньому машинному представленні приводять до арифметичного результату.

Зауваження. Строго говорячи, логічне значення "істина" повинне відповідати будь-якому числовому значенню, відмінному від нуля. Надалі ми побачимо, що саме така угода прийнята в мові C. Це дає можливість об'єднати поняття арифметичного, умовного й логічного виразів у єдиному понятті "вираз", що дуже важливо з погляду гнучкості й "симетричності" мови. Вирази, сконструйовані за допомогою операцій відношення, прийнято називати **умовними** виразами. У процесі обчислення значення всякого умовного виразу, операції відношення обробляються зліва направо. Встановлений порядок може бути змінений шляхом внесення частини виразу в круглі дужки. Наприклад, запис

$x < y == z$

повністю еквівалентна запису

$(x < y) == z$

у той час як вираз виду

$x < (y == z)$

відрізняється від попереднього порядком виконання операцій $<$ й $==$.

Результатом цих операцій є значення, відмінне від нуля, якщо операція порівняння істинна, або нуль, якщо операція порівняння хибна.

Таблиця 1.6 – Знаки операцій відношення

Знак операції	Призначення
<	Дорівнює true, якщо лівий операнд менше правого операнду
>	Дорівнює true, якщо лівий операнд більше правого операнду
<=	Дорівнює true, якщо лівий операнд менше або дорівнює правому операнду
>=	Дорівнює true, якщо лівий операнд більше або дорівнює правому операнду
==	Дорівнює true, якщо лівий операнд дорівнює правому операнду
!=	Дорівнює true, якщо лівий операнд не дорівнює правому операнду

5.4 Операції привласнення.

Формат операції простого привласнення:

операнд1=операнд2

Особливості операції привласнення:

– окрім пересилення значення вона має також і значення самого результату привласнення, тому можливо в записі одного речення мови використовувати декілька операцій привласнення. Наприклад,

`a=b=c=d=4; //a=4, b=4, c=4, d=4;`

– допускаються комбіновані операції привласнення, такі як:

`+=, -=, *=, /=, %=, &=, ^=, |=, ~=, >>=, <<=`

Лівоприпустиме значення (L-значення) – вираз, який адресує деяку ділянку пам'яті, тобто в нього можна занести значення. Ця назва пішла від операції привласнення, оскільки саме ліва частина операції привласнення визначає, в яку область пам'яті буде занесений результат операції. Змінна – це окремий випадок ліводопустимого виразу.

6 Вирази

Під виразом в мові C/C++ розуміється послідовність операндів, знаків операцій і символів роздільників. Кожним виразом є правило обчислення нового значення. Якщо

вираз формує ціле або дійсне число, то воно називається арифметичним. Пара арифметичних виразів, об'єднана операцією порівняння, називається відношенням. Якщо відношення має ненульове значення, то воно – істинне, інакше – хибне.

Компілятор дотримується чіткого порядку інтерпретації виразів, який називається правилами передування. Цей порядок (наведений у таблиці 2.1) може бути змінений за допомогою круглих дужок.

Таблиця 2.1 – Пріоритет операцій у виразах

Ранг	Операції	Порядок виконання
1	() [] -> .	→
2	! ~ - ++ -- & * (унарні) (тип) sizeof (тип)	←
3	* / % (мультиплікативні бінарні)	→
4	+ - (адитивні бінарні)	→
5	<< >> (порозрядного зсуву)	→
6	< > <= >= (відношення)	→
7	== != (відношення)	→
8	& (порозрядна кон'юнкція "И")	→
9	^ (порозрядна сума за модулем 2 (виключне "ИЛИ"))	→
10	(порозрядна диз'юнкція "ИЛИ")	→
11	&& (кон'юнкція "И")	→
12	(диз'юнкція "ИЛИ")	→
13	?: (тернарна умовна операція)	←
14	= *= /= %= -= &= ^= = <<= >>= (операції привласнення)	←
15	, (операція кома)	→

У будь-яких арифметичних виразах можна використати стандартні математичні функції, які можна застосовувати до будь-яким числових операндам. При використанні цих функцій у програму необхідно включити файл <math.h>, тобто необхідно використати директиву #include <math.h> . При цьому будуть визначені наступні функції:

sin(x) - синус (аргумент у радіанах);
cos(x) - косинус (аргумент у радіанах);
tan(x) - тангенс (аргумент у радіанах);
asin(x) - арксинус (результат у радіанах);
acos(x) - арккосинус (результат у радіанах);

atan(x) - арктангенс (результат у радіанах);
sinh(x) - гіперболічний синус;
cosh(x) - гіперболічний косинус;
tanh(x) - гіперболічний тангенс;
log10(x) - десятковий логарифм;
pow10(x) - зведення числа 10 у ступінь x;
log(x) - натуральний логарифм;
exp(x) - експонента;
sqrt(x) - квадратний корінь;
pow(x,y) - зведення x у ступінь y;
fabs(x) - абсолютна величина для double;
abs(x) - абсолютна величина для int.

7 Введення/виведення даних. Потоки введення / виведення cin та cout

При використанні бібліотеки класів C++, використовується бібліотечний файл `iostream.h`, у якому визначені стандартні потоки введення даних з клавіатури `cin` і виведення даних на екран монітора `cout`, а також відповідні операції:

- << – операція запису даних в потік;
- >> – операція читання даних з потоку.

Загальна форма запису даних в потік виведення:

```
cout<<"Рядок_даних1" [<<"Рядок_даних2"...<<"Рядок_данихN"];
```

Рядок даних може містити:

- символи, які безпосередньо будуть виводитися на екран (проста текстова інформація);
- керуючі послідовності символів.

Аналогічно, як при використанні керуючих символів у функції виведення `printf()`, вони не виводяться на екран, а керують розташуванням символів, які виводяться на екран монітора. Ознакою керуючого символу є наявність символу `"\"`. У рядок формату можуть входити наступні основні керуючі послідовності символів:

- `'\n'` – новий рядок;
- `'\t'` – горизонтальна табуляція;
- `'\v'` – вертикальна табуляція;
- `'\b'` – повернення на символ;
- `'\r'` – повернення на початок рядка;
- `'\a'` – звуковий сигнал тощо.

Зазвичай, для переведення на новий рядок використовують вираз `endl`.

Наприклад,


```
cout<<"Привіт, мова Сі!"<<endl<<"Я – програміст.";
```

Результат роботи програми:

```
Привіт, мова Сі!  
Я – програміст.
```

Наприклад,

```
int a=5;  
cout<<"Значення змінної a"<<a;
```

Результат роботи програми:

```
Значення змінної a=5.
```

Наприклад,

```
float x=2.78;  
cout<<"Значення змінної x"<<x;
```

Результат роботи програми:

```
Значення змінної x=2.78.
```

Загальна форма запису читання даних з потоку введення:

```
cin>>змінна1[>>змінна2...>>зміннаN];
```

В якості змінних вказуються імена об'єктів, куди будуть записані дані, прочитані з потоку введення.

Наприклад,

```
int m;  
cin>>m;  
/* Ввести з клавіатури число  
і присвоїти його змінній m */  
  
float x, y;  
cin>>x>>y;  
/* Ввести з клавіатури два числа  
і присвоїти їх відповідним змінним x та y */
```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що включає в себе алфавіт мови C/C++?
- 2 Які базові лексеми включає в себе мова C/C++?
- 3 Що таке ідентифікатор і з чого він може складатися?
- 4 Дайте визначення поняття константи?
- 5 На які основні групи можливо розділити константи в мові C/C++?
- 6 Приведіть приклад цілої десяткової константи.
- 7 З яких частин складається дійсна константа?
- 8 Приведіть приклад дійсної константи з фіксованою крапкою.
- 9 Приведіть приклад дійсної константи з плаваючою крапкою.
- 10 Що таке операнд?
- 11 Які операції називаються унарними? Приведіть приклади унарних операцій.
- 12 Які операції називаються бінарними? Приведіть приклади бінарних операцій.
- 13 Які різниця між постфіксною і префіксною операціями інкремента (декремента)?
- 14 Яка операція називається операцією відношення?
- 15 В якому випадку відношення вважається брехнею, а в якому – істиною?
- 16 Перерахувати знаки операцій, які відносяться до операцій відношення?
- 17 Дайте визначення поняттю "вираз"?
- 18 З чого складається вираз?
- 19 Що таке пріоритет операцій в арифметичному вираз. Як його можна змінити?
- 20 Перерахувати операції привласнення, які існують в C/C++?
- 21 Що таке ліводопустиме значення?
- 22 Яка функція використовується в оригінальному Cі для виведення даних на екран монітора?
- 23 Що таке "рядок формату" у функції виведення даних printf()?
- 24 Як називається потік в C++ для виведення даних на екран монітора?
- 25 Наведіть приклад коду програми для введення з клавіатури одного цілого числа, використовуючи функції scanf() та потік cin.

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Які з наступних слів є ключовими словами мови C: `main`, `integer`, `function`, `char`, `=`?

- 2 Який діапазон значень у типів `int`, `float`?
- 3 Укажіть неприпустимий в C запис цілої константи:
- A. 01
 - B. 0x1
 - C. 0d1
- 4 Як записується десяткове число 13 у двійковій системі числення?
- A. 1100
 - B. 1011
 - C. 1101
- 5 Що таке ключове слово?
- A. Призначено для міток у програмі
 - B. Має фіксоване написання й зміст
 - C. Спеціальне слово для пароля
- 6 Що таке ідентифікатор?
- A. Адреса об'єкта
 - B. Показчик на підпрограму
 - C. Ім'я об'єкта
- 7 Чи вірні оголошення ідентифікаторів:
- Ab
 - Sum_1
 - y-x
 - r23
 - 2vf
- 8 У мові C рядкові й прописні букви
- A. Різняться скрізь
 - B. Різняться тільки в ідентифікаторах
 - C. Різняться тільки в зарезервованих словах

9 Яким буде значення змінної `t` після виконання наступних рядків коду:

```
...  
int t, z = 0;  
t = !z;  
...
```

10 Визначити істинність складеного висловлення: « $(2 \times 2 = 4 \text{ або } 3 \times 3 = 10)$ і $(2 \times 2 = 5 \text{ або } 3 \times 3 = 9)$ ».

11 Чому дорівнює значення змінної `x` після виконання команд:

```
int n=5;  
double x;  
x=n<=5?3.2:3.5;
```

12 Чому дорівнює значення змінної k після виконання команд:

а)

```
int n=2,m=3,k;  
k=n^m*4;
```

б)

```
int n=4,k;  
k=n+7;  
k/=n;
```

в)

```
int n=3,m=2,k;  
k=n++&m;
```

г)

```
int n=5,m=3;  
double k;  
k=n/m;
```

13 Напишіть фрагмент програми, що правильно виконує введення значення цілої змінної n:

14 Який специфікатор перетворення потрібно використовувати для виводу числа у вигляді 0x8a у поле із шириною 6?

15 Який рядок уводить ціле число й число із дробовою частиною?

- A. `scanf ("%x, %f", &a, &b);`
- B. `scanf ("%d, %f", &x, &y);`
- C. `scanf ("%u, %e", &celoe, &drob);`

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція №№ 3-4

з навчальної дисципліни Технології (Програмування)

Тема Види управляючих структур. Блоки та складені оператори. Оператори розгалуження if та if-else в мові програмування C/C++. Оператор-перемикач switch. Оператор break.

Мета заняття Ознайомити студентів із різними видами управляючих структур, поняттям порожнього оператора, блоку та складеного оператора, вивчити конструкції if та if-else, вивчити властивості та принципи використання оператора switch та оператора goto

Час – 4 години.

План проведення лекції

- 1 Базові конструкції структурного програмування
- 2 Оператор «вираз»
- 3 Складені оператори
- 4 Оператор вибору
- 5 Оператор перемикач (оператор switch)
- 6 Оператор безумовного переходу goto

Навчальні матеріали лекції

- 1 Базові конструкції структурного програмування

У теорії програмування доведено, що програму для вирішення завдання будь-якої складності можна скласти тільки з трьох структур: лінійної, розгалуженої і циклічної. Ці структури називаються базовими конструкціями структурного програмування.

Лінійною називається конструкція, що є послідовним з'єднанням двох або більшої кількості операторів.

Розгалуження – задає виконання одного з двох операторів, залежно від виконання умови.

Цикл – задає багатократне виконання оператора.

Метою використання базових конструкцій є отримання програми простої структури. Таку програму легко читати, налагоджувати і при необхідності вносити до неї зміни. Структурне програмування також називають програмуванням без goto,

оскільки часте використання операторів переходу утрудняє розуміння логіки роботи програми. Але іноді зустрічаються ситуації, в яких застосування операторів переходу, навпаки, спрощує структуру програми.

Оператори управління роботою програми називають управляючими конструкціями. До них відносять:

- складені оператори;
- оператори вибору;
- оператори циклів;
- оператори переходу.

2 Оператор «вираз»

Будь-який вираз, що закінчується крапкою з комою, розглядається як оператор, виконання якого полягає в обчисленні цього виразу. Окремим випадком виразу є порожній оператор ";".

Наприклад:

```
i++;  
a+=2;  
x=a+b;
```

3 Складені оператори

До складених операторів відносять власне складені оператори і блоки. В обох випадках це послідовність операторів, ув'язнена у фігурні дужки { }. Блок відрізняється від складеного оператора наявністю визначень в тілі блоку.

Наприклад:

```
//це складений оператор  
{  
n++;  
summa+=n;  
}  
//це блок  
{  
int n=0;  
n++;  
summa+=n;  
}
```

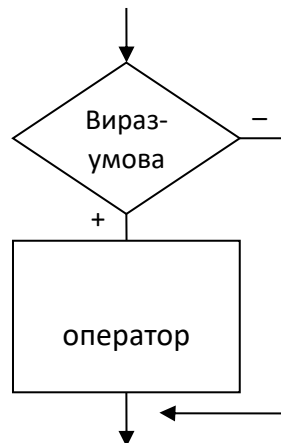
4 Оператор вибору

Одним з операторів вибору є умовний оператор.

Умовний оператор `if` має повну і скорочену форму.

Скорочена форма:

```
if (вираз-умова)
    оператор;
```

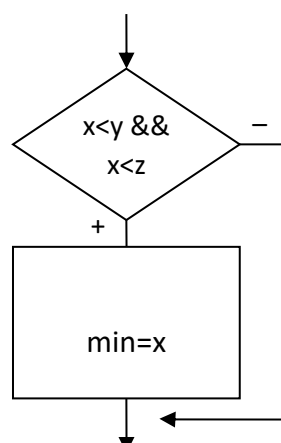


Як вираз-умова можуть використовуватися арифметичний вираз, відношення і логічний вираз. Якщо значення виразу-умови відмінне від нуля (тобто істинно), то виконується *"оператор"*.

Якщо *"оператор"* повинен складатися з декількох операторів, то необхідно використовувати складений оператор (блок).

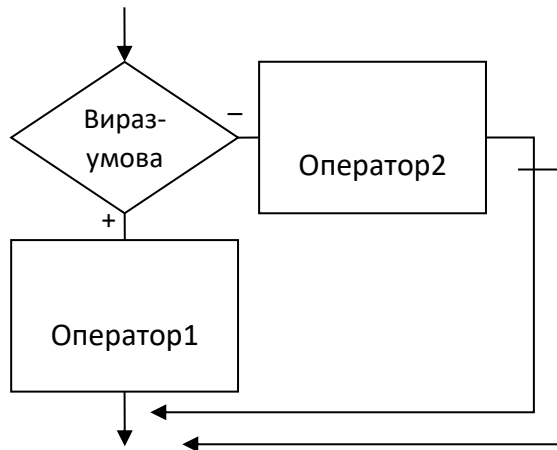
Наприклад:

```
if (x<y && x<z)
    min=x;
```



Повна форма

```
if (вираз-умова)
    оператор1;
else
    оператор2;
```



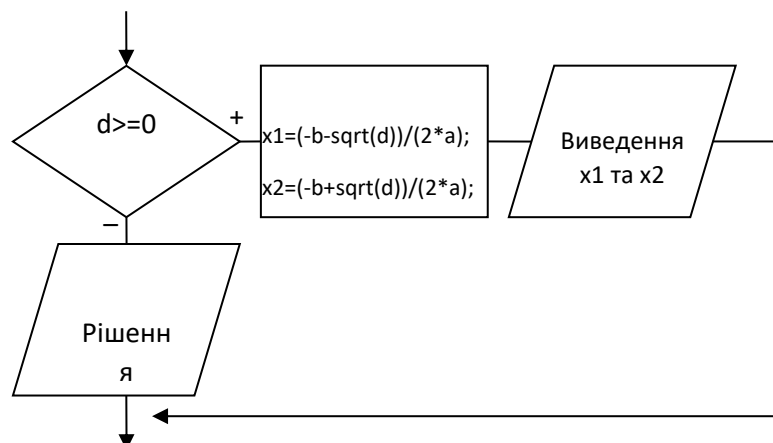
Якщо значення виразу-умови відмінне від нуля, то виконується *оператор1*, при нульовому значенні виразу-умови виконується *оператор2*.

Приклад 1. Фрагмент програми обчислення коренів квадратного рівняння.

```

if (d>=0)
{
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    cout<< "\nx1="<<x1<<"x2="<<x2;
}
else
    cout<<"\nРешения немає";
  
```

Блок-схема наведеного фрагменту програми:

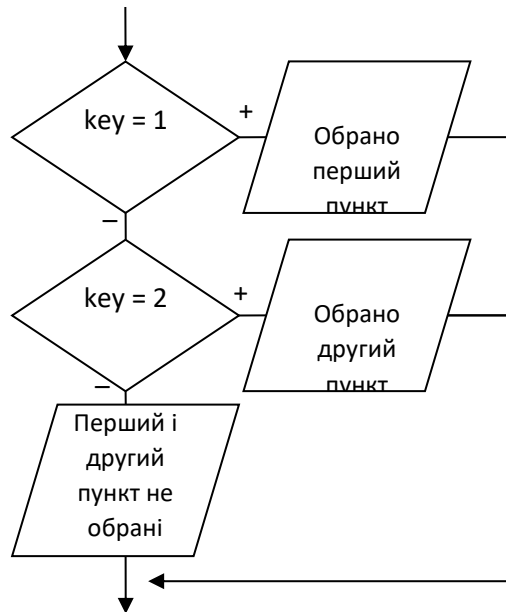


Приклад 2. Фрагмент програми реалізації вибору пунктів меню

```

if (key == 1)
    printf("\n Обрано перший пункт");
else
    if (key == 2)
        printf("\n Обрано другий пункт");
    else
        printf("\n Перший і другий пункти не обрані");
  
```


Блок-схема наведеного фрагменту програми:



5 Оператор перемикач (оператор switch)

Оператор if дозволяє здійснити вибір тільки між двома варіантами.

Для того, щоб проводити вибір один з декількох варіантів використовується оператор switch.

Загальна форма запису:

```
switch (константний_вираз)
{
case константне_значення1: оператор1; [оператори;] [break;]
case константне_значення2: оператор2; [оператори;] [break;]
. . .
case константне_значенняN: операторN; [оператори;] [break;]
[default: оператор;]
}
```

Оператор виконується таким чином:

- 1 Обчислюється вираз в дужках оператора switch;
- 2 Обчислене значення порівнюється з мітками (константними значеннями) в опціях case;
- 3 Порівняння проводиться до тих пір, поки не буде знайдена мітка, відповідна даному значенню, після цього виконується оператор відповідної гілки;
- 4 Якщо відповідна мітка не знайдена, то виконується оператор в опції default. Альтернатива default може бути відсутньою, тоді не буде проведено ніяких дій.

Оператор break здійснює вихід з оператора switch і перехід до наступного за ним оператора. За відсутності оператора break виконуватимуться всі оператори, починаючи з поміченого даною міткою і кінчаючи оператором в опції default.

Константний вираз (вираз, операнди якого константи) повинен бути цілого типу (включаючи char).

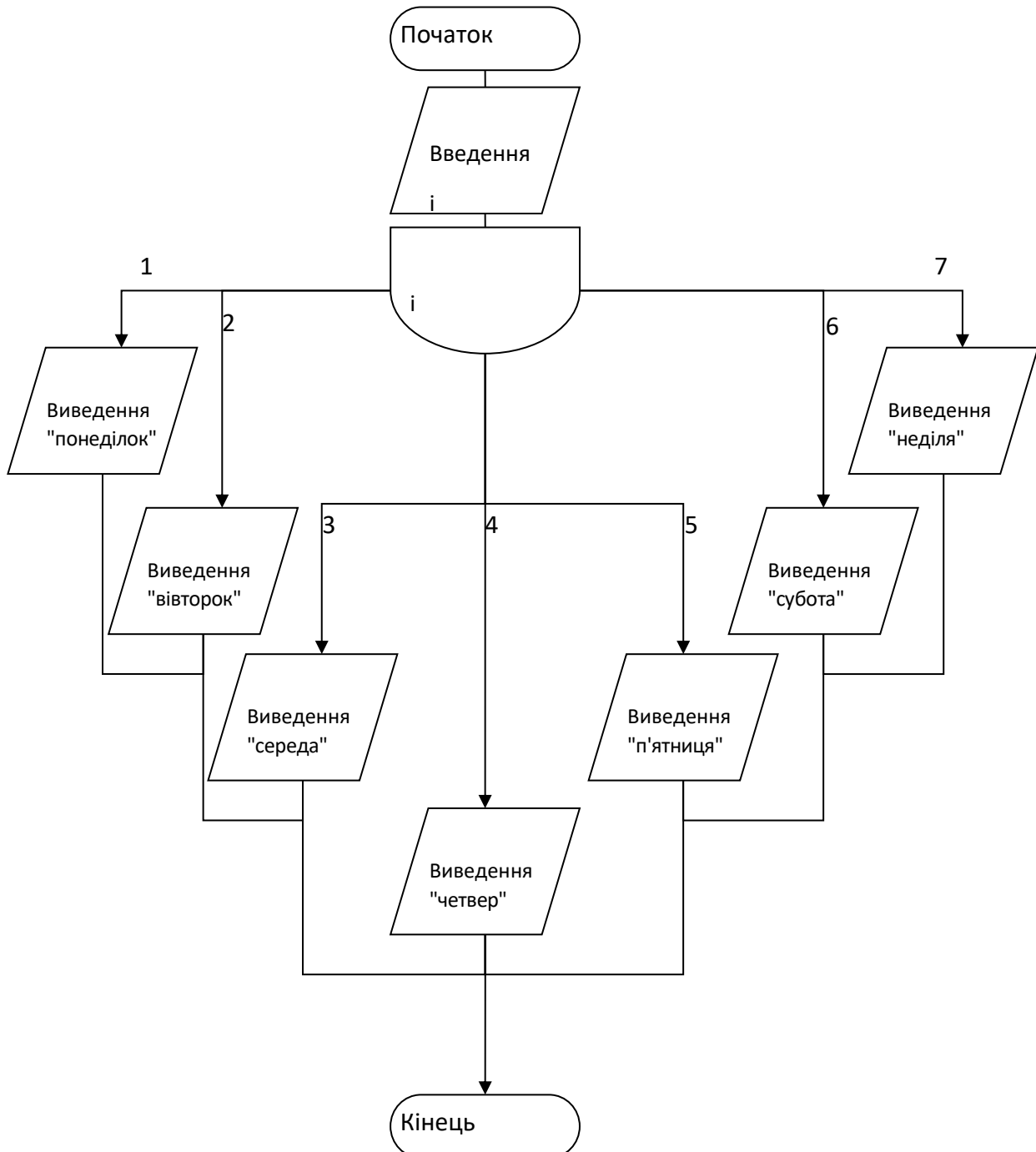
Наприклад. Розробити програму, що визначає день тижня по його введеному номеру. Програма повинна реагувати на невірний введений номер дня тижня.

Лістинг програми:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i;
    printf("\n Введіть номер дня тижня: ");
    scanf("%i",&i);
    switch(i)
    {
        case 1:
            printf("\n Понеділок.");
            break;
        case 2:
            printf("\n Вівторок.");
            break;
        case 3:
            printf("\n Середа.");
            break;
        case 4:
            printf("\n Четвер.");
            break;
        case 5:
            printf("\n П'ятниця.");
            break;
        case 6:
            printf("\n Субота.");
            break;
        case 7:
            printf("\n Неділя.");
            break;
        default:
            {
                printf("\n Невірно введений");
                printf("номер дня тижня.");
            }
    }
}
```

```
}  
}  
}
```

Блок-схема алгоритму рішення задачі:



Результат роботи програми:

Введіть номер дня тижня: 2

Вівторок.

6 Оператор безумовного переходу goto

Загальна форма запису

```
goto мітка;  
.  
.  
.  
мітка: <оператор>;
```

Виконання оператора goto викликає передачу управління в програмі операторові, поміченому міткою. У тілі тієї ж функції обов'язкова повинна бути присутня *мітка*.

Мітка – це звичайний ідентифікатор, областю видимості якого є функція. Для відділення мітки від оператора використовується двокрапка. Мітка з поміченим оператором може з'явитися в програмі як до оператора goto, так і після. Імена міток утворюються по тих же правилах, що і імена ідентифікаторів.

Наприклад,

```
if (size>12)  
goto a;  
else  
goto b;  
a: cost = cost*3;  
flag=1;  
b: s= const*flag;
```

Застосування goto порушує принципи структурного і модульного програмування, за якими всі блоки, з яких складається програма, повинні мати тільки один вхід і лише один вихід. Мова C/C++ реалізована таким чином, що можна програмувати без оператора goto.

Використання оператора goto виправдане, якщо необхідно виконати перехід з декількох вкладених циклів або перемикачів вниз по тексту програми або перейти в одне місце функції після виконання різних дій.

Не можна передавати управління всередину операторів if, switch і циклів. Не можна переходити всередину блоків, що містять ініціалізацію, на оператори, які стоять після ініціалізації.

Приклад помилкової роботи програми при неправильному використанні оператора безумовного переходу goto.

```
int k;  
goto m;  
.  
.  
.  
{int a=3, b=4;  
k=a+b;  
m: int c=k+1;  
.  
.  
.
```

}

В даному прикладі при переході на мітку *m* не виконуватиметься ініціалізація змінних *a*, *b* і *k*.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Назвіть базові конструкції програмування.
- 2 Що називається оператором?
- 3 Виконайте загальну блок-схему лінійного алгоритму.
- 4 Що таке складений оператор?
- 5 Що таке блок?
- 6 Виконайте загальну блок-схема розгалуженого алгоритму.
- 7 Наведіть повну форму коду умовного оператора *if*.
- 8 Виконайте загальний вигляд блок-схеми повного умовного оператора.
- 9 Наведіть скорочену форму коду умовного оператора *if*.
- 10 Виконайте загальний вигляд блок-схеми скороченого умовного оператора.
- 11 Що може включати "вираз-умова" в операторі *if*?
- 12 Що необхідно задіювати, якщо необхідне виконання не одного оператора, а деякого набору операторів в операторі *if*?
- 13 Опишіть призначення оператора *switch*.
- 14 Якого типу може бути константний вираз для порівняння з заданими варіантами у операторі *switch*?
- 15 Для чого призначене ключове слово *case* в операторі *switch*?
- 16 Для чого в операторі *switch* використовується оператор *break*?
- 17 Чи буде порушена логіка роботи програми в разі відсутності оператора *break* в гілках оператора *switch*?
- 18 Опишіть призначення гілки *default* в операторі *switch*.
- 19 Опишіть призначення оператора *goto*? З яких символів можуть складатися імена міток?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Написати програму розгалуженого алгоритму для підрахунку значення
$$g = \begin{cases} f - 2 * l, & s > 0 \\ \frac{h - k}{h + k}, & s \leq 0 \end{cases}$$

2 Написати програму обчислення площі кола. Програма повинна перевіряти правильність вхідних даних. Нижче наведено рекомендований вигляд екрану під час виконання програми:

Вычисление площади кольца.

Введите исходные данные:

Радиус кольца (см) -> **3.5**

Радиус отверстия (см) -> **7**

Ошибка! Радиус отверстия не может быть больше радиуса кольца.

3 Написати програму, що виводить завдання на знаходження різниці двох чисел, перевіряє його та виводить повідомлення: "Правильно!" або "Вы ошиблись" і правильний результат. Нижче наведено рекомендований вигляд екрану під час виконання програми:

Сколько будет 83-17?

Введите ответ и нажмите <Enter> ->**67**

Вы ошиблись. 83-17=66

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 5

з навчальної дисципліни Технології (Програмування)

Тема Ітераційні та регулярний цикли.

Управління циклами за допомогою операторів break і continue

Мета заняття Ознайомити студентів із різновидами циклічних структур, вивчити властивості та принципи використання операторів for, while та do-while, а також засвоїти правила використання управляючих конструкцій break та continue

Час – 2 години.

План проведення лекції

- 1 Регулярний цикл for
- 2 Ітераційні цикли while та do-while
- 3 Оператори break і continue

Навчальні матеріали лекції

Розрізняють наступні види циклів:

- регулярні цикли;
- ітераційні цикли.

Група дій, що повторюються в циклі, називається його *тілом*. Одноразове виконання циклу називається його *кроком*.

1 Регулярний цикл for

Цикл for – цикл з фіксованим числом повторень, яке заздалегідь відоме.

Загальна форма запису:

```
for(ініціалізація; перевірка_умови; корекція)  
<оператор>;
```

Зображення регулярного циклу наведено на рисунку 1.1.

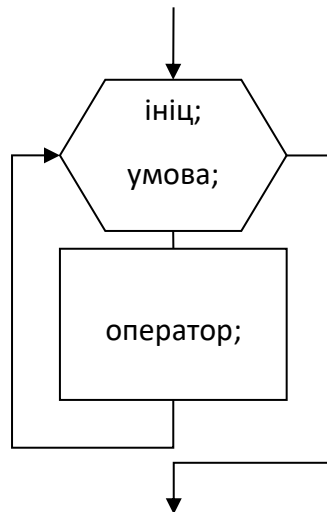


Рисунок 1.1. Регулярний цикл в блок-схемі.

Для організації такого циклу повинні розглядатися три операції:

- ініціалізація лічильника;
- порівняння його величини з деяким граничним значенням;
- зміна значення лічильника при кожному проходженні тіла циклу.

Ці три операції записуються в дужках і розділяються крапкою з комою (;).

Перший вираз (*ініціалізація*) служить для ініціалізації лічильника. Ініціалізація здійснюється тільки один раз – коли цикл `for` починає виконуватися.

Другий вираз (*перевірка умови*) служить для перевірки умови перед кожним можливим виконанням тіла циклу. Коли вираз стає брехнею (рівним нулю), цикл завершиться.

Третій вираз (*корекція*) обчислюється в кінці кожного виконання тіла циклу. Лічильник може як збільшуватися, так і зменшуватися.

Будь-який вираз може бути відсутнім, але символи крапки з комою (;), що їх розділяють, повинні бути обов'язково.

Наприклад.

Вивести на екран монітора числа від 1 до 5 та їх квадрати.

```

int num;
for(num=1; num<=5; num++)
printf("\n %d * %d = %d", num, num, num*num);
  
```

Блок-схема алгоритму наведеного фрагменту надана на рисунку 2.1.

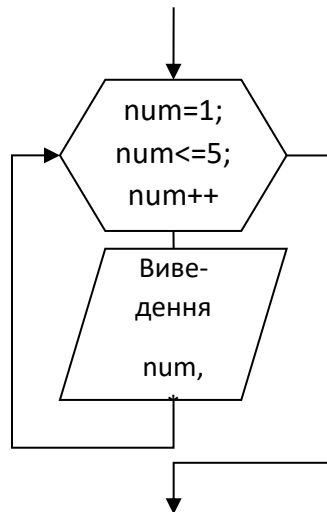


Рисунок 1.2. Блок- схема алгоритму.

Параметри, що знаходяться в заголовку циклу, можна змінити при виконанні операцій в тілі циклу.

Наприклад.

```
int k=2;
for(n=3; k<=25; )
{
  ...
  k = k*n;
}
```

У циклі `for` часто використовується операція «кома» (,) для розділення декількох виразів. Це дозволяє включити в специфікацію циклу декілька виразів, які будуть ініціалізуватися або коректуватися. Вирази, до яких застосовується операція «кома», обчислюватимуться зліва направо.

Наприклад.

```
for(i=0, j=1; i<n; i++, j=i);
```

У C/C++ допускаються вкладені цикли, тобто коли один цикл знаходиться усередині іншого.

Приклад

```
for( i=0; i<n; i++)
{
  for( j=0; j<n; j++)
  {
    ...
  }
  ...
}
```

Приклади використання регулярного циклу з параметром.

1) Збільшення лічильника;

```
for (n=0; n<10; n++)  
оператор;
```

2) Зменшення лічильника;

```
for (n=10; n>0; n--)  
оператор;
```

3) Довільна зміна кроку коректування;

```
for (n=2; n>60; n+=13)  
оператор;
```

4) Можливість перевіряти умову відмінну від умови, яка накладається на число ітерацій;

```
for (num=1; num*num*num<216; num++)  
оператор;
```

5) Корекція може здійснюватися не тільки за допомогою складання або віднімання.

```
for (d=100.0; d<150.0; d*=1.1)  
{  
<тіло циклу>;  
}  
for (x=1; y<=75; y=5*(x++)+10)  
{  
<тіло циклу>;  
}
```

2 Ітераційні цикли while та do-while

У ітераційних циклах заздалегідь невідома кількість разів виконання циклу.

Цикл while – ітераційний цикл з передумовою.

Загальна форма запису:

```
while (вираз-умова)  
оператор;
```

В якості <виразу-умови> найчастіше використовується відношення або логічний вираз. Якщо <вираз-умова> істинне (тобто не рівно нулю), то виконується оператор або група операторів, що входять в цикл while, та потім вираз перевіряється знову. Зображення на блок-схемі циклу з передумовою наведено на рисунку 2.1.

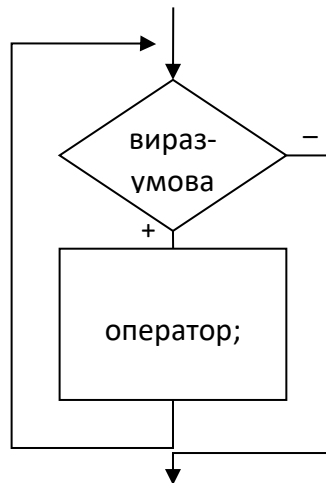


Рисунок 2.1. Зображення на блок-схемі циклу з передумовою

Послідовність дій, що складається з перевірки і виконання оператора, повторюється до тих пір, поки вираз не стане брехнею (тобто рівним нулю). Після цього відбувається вихід з циклу, і далі виконується оператор, що стоїть після оператора циклу. При побудові циклу `while`, в нього необхідно включити конструкції, що змінюють величину виразу, що перевіряється, так, щоб врешті-решт воно стало брехнею. Інакше виконання циклу ніколи не завершиться.

Цикл `while` – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу.

Наприклад.

```

k=5;
n=10;
while (k<n)
{
printf("k=%d n=%d\n", до, n);
k+=2;
k=k+2;
n++;
}
  
```

Do-while – ітераційний цикл з постумовою.

Загальна форма запису:

```

do
оператор;
while (вираз-умова);
  
```

Зображення на блок-схемі циклу з постумовою наведено на рисунку 2.2.

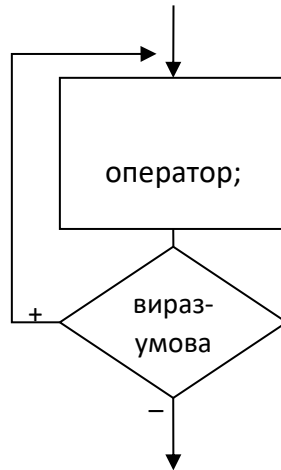


Рисунок 2.2. Зображення на блок-схемі циклу з постумовою

Цикл `do-while` — це цикл з постумовою, в якому істинність виразу-умови перевіряється після виконання всіх операторів, включених в тіло циклу. Тіло циклу виконується до тих пір, поки вираз не стане брехнею, тобто тіло циклу виконується хоч би один раз. Використовувати цикл краще всього в тих випадках, коли повинна бути виконана хоч би одна ітерація.

Наприклад.

```
do
{
printf(" Введіть n>0");
scanf("%d", &n);
}
while (n<0);
```

3 Оператори `break` і `continue`

У тілі будь-якого типу циклу можна використовувати оператора `break`, який дозволяє вийти з циклу, не завершуючи його. Оператор `continue` дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

1) `break` – оператор переривання циклу.

```
while (<вираз>)
{
...
break;
}
```

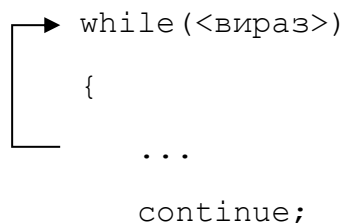
Тобто оператор `break` доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

Наприклад.

Написати програму, яка підраховує суму чисел що вводяться з клавіатури до тих пір, поки не буде введено 100 чисел або число 0

```
for (s=0, i=1; i<100; i++)
{
    cin>>x;
    if (x==0) break; // якщо ввели 0, то підсумовування закінчується
    s+=x;
}
```

2) `continue` – оператор переходу до наступної ітерації циклу. Він використовується, коли тіло циклу містить галуження.



```
while (<вираз>)
{
    ...
    continue;
}
```

The diagram shows a `while (<вираз>)` loop. Inside the loop body, there is an ellipsis `...` followed by the `continue;` statement. A curved arrow originates from the `continue;` statement and points back to the condition `<вираз>` of the `while` loop, illustrating that the loop body is skipped and the condition is re-evaluated immediately.

Наприклад.

Написати програму, яка підраховує кількість і суму додатних чисел

```
for ( k=0, s=0, x=1; x!=0; )
{
    cin>>x;
    if (x<=0) continue; //якщо ввели <=0, то перехід до наступної ітерації
    k++; s+=x;
}
```

При вкладених циклах дії операторів `break` і `continue` розповсюджується тільки на саму внутрішню структуру, в якій вони містяться.

Використання цих операторів можливе, але небажано, оскільки вони погіршують читаність програми, збільшують вірогідність помилок, утрудняють модифікацію.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Назвіть типи циклів.
- 2 Що називається тілом циклу?
- 3 Що називається кроком циклу?

- 4 Який оператор відповідає регулярному циклу?
- 5 Наведіть загальний вигляд оператора циклу з постумовою.
- 6 Який оператор відповідає циклу передумовою?
- 7 Наведіть загальний вигляд оператора регулярного циклу.
- 8 Який оператор відповідає циклу з постумовою?
- 9 Наведіть загальний вигляд оператора циклу з передумовою.
- 10 Які мінімальну кількість разів виконується цикл while?
- 11 Які мінімальну кількість разів виконується цикл do-while?
- 12 Охарактеризуйте призначення оператора break.
- 13 Охарактеризуйте призначення оператора continue?
- 14 Чи може в операторі регулярного циклу for бути відсутній будь-який з трьох виразів в заголовку?
- 15 Якщо мається декілька вкладених операторів циклів, то на який з них будуть мати вплив оператори break та continue?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Використовуючи оператор циклу з передумовою while або оператор циклу з постумовою do...while скласти програму рішення задачі:
 - 1.1 Підраховувати суму та добуток непарних чисел серед тих цілих чисел, що вводяться користувачем, поки не буде введено число 0.
 - 1.2 Підраховувати середнє арифметичне непарних чисел серед тих цілих чисел, що вводяться користувачем, поки не буде введено число з діапазону [-4;0].
- 2 Використовуючи оператор циклу for скласти програму рішення задачі
 - 2.1 Обчислити середнє арифметичне непарних чисел із 13-ти цілих чисел, що вводяться користувачем.
 - 2.1 Порахувати добуток парних чисел з діапазону [-3;16] серед 9-ти цілих чисел, що вводяться користувачем.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція №№ 6-7

з навчальної дисципліни Технології (Програмування)

Тема Одномірні масиви даних. Алгоритми обробки одномірних масивів. Алгоритми пошуку екстремумів та сортування одномірних масивів даних.

Мета заняття Пояснити студентам поняття масиву даних, представлення його в пам'яті ЕОМ, опис масиву, способи ініціалізації масиву. Розглянути алгоритми обробки лінійних масивів. Пояснити студентам ідеї алгоритмів пошуку екстремумів та ідеї алгоритмів сортування елементів одномірного масиву: метод вибору, сортування обмінами, сортування вставками. Розглянути практичні приклади застосування наведених алгоритмів. Пояснити поняття та організацію псевдо динамічного масиву.

Час – 4 години

План проведення лекції

- 1 Одномірні масиви. Опис масивів, уявлення в пам'яті.
- 2 Ініціалізація масиву.
3. Алгоритми обробки лінійних масивів.
- 4 Алгоритми пошуку екстремумів.
- 5 Формування псевдодинамічних масивів.
- 6 Алгоритми сортування лінійних масивів
 - 6.1 Сортування масиву методом вибору.
 - 6.2 Сортування обмінами («бульбашковий» метод сортування).
 - 6.3 Сортування простими вставками.

Навчальні матеріали лекції

- 1 Одномірні масиви. Опис масивів, уявлення в пам'яті

Більшістю об'єктів мови C/C++, з якими ми мали справу, були змінні. Кожна змінна при оголошенні отримувала тип і ім'я, з яким зв'язувався певний елемент пам'яті. Проте розташування значень змінних по адресах пам'яті ніяк не упорядковувалося. При вирішенні багатьох завдань, особливо з великою кількістю однотипних даних,

використання змінних з різними іменами, а значить не впорядкованих по адресах пам'яті, утрудняє або робить взагалі неможливим програмування. У подібних випадках в мові C/C++ використовують об'єкти, які називаються масивами.

Масив – це кінцева впорядкована послідовність однотипних даних. Кожному елементу масиву відводиться одна комірка пам'яті. Елементи одного масиву займають послідовно розташовані комірки пам'яті.

Всі елементи мають одне ім'я – ім'я масиву і відрізняються індексами – порядковими номерами в масиві. В C++ масиви можуть бути одномірними або багатомірними.

Одномірний (лінійний) масив – це список однотипних зв'язаних змінних. Доступ до окремого елементу лінійного масиву здійснюється за допомогою лише одного індексу. Кількість елементів в масиві називається його розміром.

Щоб відвести в пам'яті потрібну кількість комірок для розміщення масиву, треба заздалегідь знати його розмір. Резервування пам'яті для масиву виконується на етапі компіляції програми.

Загальна форма оголошення масиву

тип ім'я[розмір_масиву];

де тип – це тип елементу масиву, ім'я – ідентифікатор імені масиву, розмір_масиву – значення цілого типу, що відповідає кількості елементів масиву, під які необхідно відвести пам'ять.

Елементи масиву завжди нумеруються з 0.

Принцип розташування елементів масиву в пам'яті ЕОМ наведено на рисунку 1.1

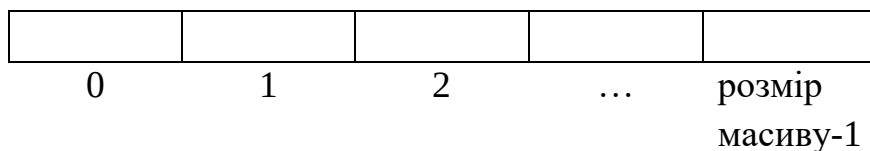


Рисунок 1.1 – Розташування елементів масиву у пам'яті

Наприклад.

```
int data[10]; //масив з 10 елементів цілого типу
```

Тут масив містить 10 елементів типу int:

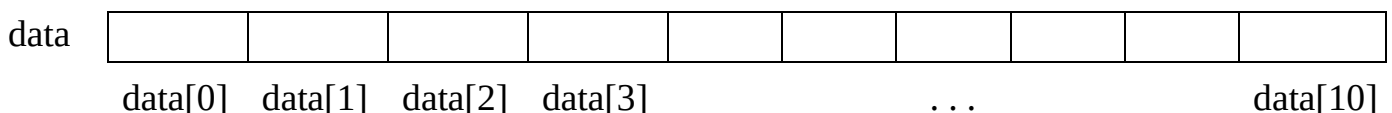


Рисунок 1.2 – Розташування елементів масиву data[10] у пам'яті

Щоб звернутися до елементу масиву, треба вказати ім'я масиву і номер елементу в масиві (індекс):

- $a[0]$ – індекс задається як константа;
- $a[8]$ – індекс задається як константа;
- $a[i]$ – індекс задається як змінна;
- $a[2*i]$ – індекс задається як вираз.

Така дія інакше називається адресацією або індексацією.

2 Ініціалізація масиву.

Існує два способи ініціалізації масиву:

- 1 Задання початкових значень при оголошенні масиву.
- 2 Організація циклу послідовного введення значень елементів масиву.

Загальна форма ініціалізації масиву при оголошенні

```
тип ім'я[n]={значення_0, значення_1, значення_2, ..., значення_n-1};
```

Початкові значення елементів масиву включають у фігурні дужки. Якщо програміст не вказав в квадратних дужках розмір масиву, то компілятор сам задасть розмір по кількості приведених значень. Якщо початкові значення у фігурних дужках не задані, то всі елементи масиву будуть нульовими.

Наприклад.

```
int a[10]={1,2,3,4,5,6,7,8,9,10}; //ініціалізація 10 елементів
int a[10]={1,2,3,4,5}; //5 з 10 елементів задані, решта нулі
int a[]={1,2,3,4,5}; //створення масиву на основі 5-ти елементів
```

Організація циклу послідовного введення значень елементів масиву.

Обробку масивів треба виконувати поелементно в циклі. Цикл послідовного введення значень елементів масиву зручно організувати за допомогою оператора for.

Наприклад. Розробити програму введення з клавіатури масиву data[] із 5 цілих чисел та виведення на екран отриманого масиву.

Блок – схема алгоритму рішення задачі наведена на рисунку 2.1

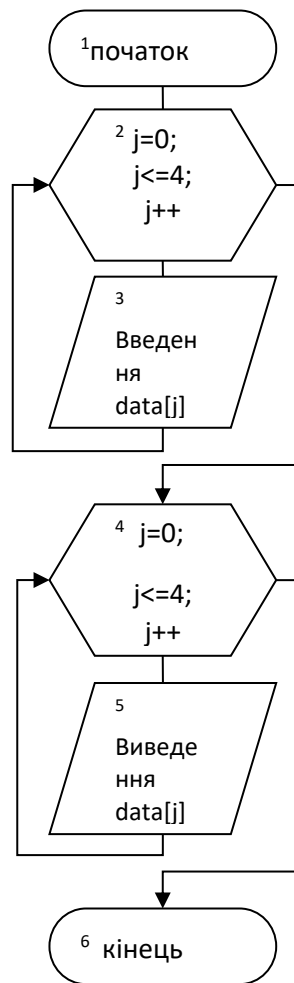


Рисунок 2.1 – Блок – схема введення-виведення одномірного масиву

Текст програми введення – виведення одномірного масиву із 5-ти цілих чисел наведено в лістингу 2.1

Лістинг 2.1

```

#include <stdio.h>
int main()
{
    int data[5];
    int j;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]", j);
        scanf("%i", &data[j]);
    }
    printf("\n\n Введений масив:");
    for (j=0; j<=4; j++)
    {
        printf("\n Data[%i]=%i", j, data[j]);
    }
    return 0;
}
  
```

3. Алгоритми обробки лінійних масивів

Приклад 1.

Для даного лінійного b масиву із 10 дійсних чисел знайти середньоарифметичне значення від'ємних елементів.

Програма буде містити три логічні блоки:

- блок введення значень елементів масиву;
- блок обчислення середньоарифметичного значення від'ємних елементів масиву;
- блок виведення результату. Враховувати, що від'ємні елементи в масиві можуть бути відсутні.

Блок – схема алгоритму рішення задачі приведена на рисунку 3.1.

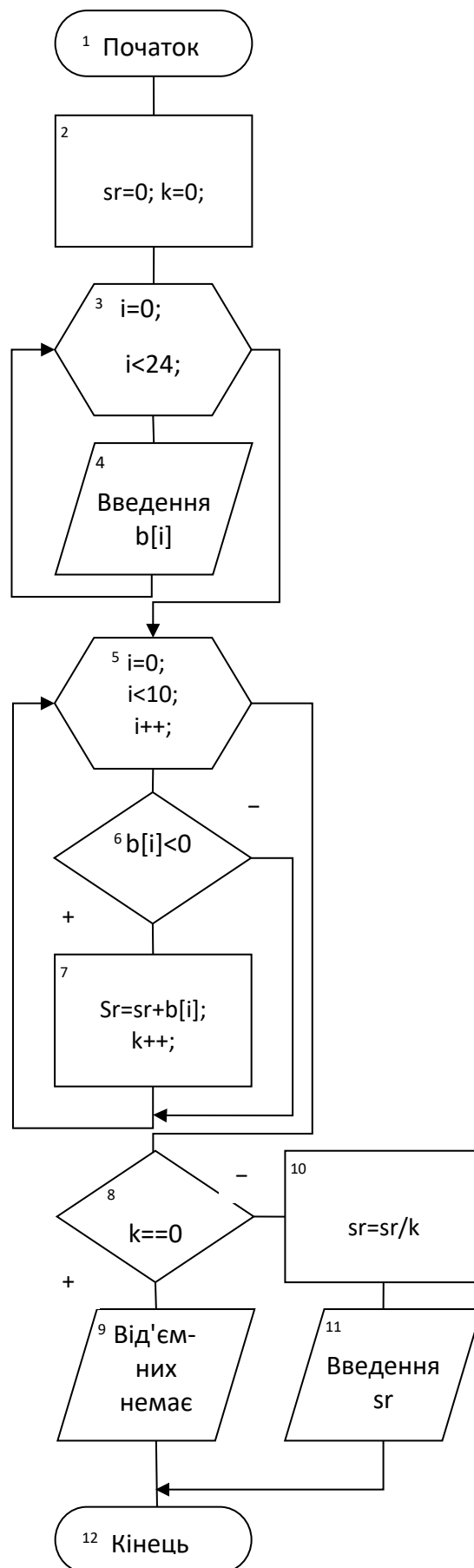


Рисунок 3.1. Блок – схема алгоритму пошуку середнього арифметичного значення від'ємних елементів лінійного масиву.

Програму рішення поставленої задачі наведено в лістингу 3.1

Лістинг 3.1

```
#include <iostream.h>
#include <conio.h>
int main (void)
{
float b[10]; //опис масиву
float sr=0;
int i;
int k=0; //кількість від'ємних елементів у введеному масиві
clrscr;
// блок введення елементів масиву
for (i=0; i<10; i++)
{ cout<<endl<<"Введите элемент массива b["<<i+1<<"]: ";
cin>>b[i];
}
//блок обчислення середньоарифметичного значення
for (i=0; i<10; i++)
if (b[i]<0) //перевіряємо, чи від'ємний елемент масиву
{ sr=sr+b[i];
k++; //збільшуємо кількість від'ємних
}
// блок виведення результату
if (k==0)
cout<<endl<<"В массиве нет отрицательных элементов";
else
{ sr=sr/k;
cout<<endl<<"Среднеарифметическое значение отрицательных
элементов= "<<sr;
}
cout<<endl<<"Для завершения работы нажмите Enter ";
getch();
return 0;
}
```

Приклад 2.

Дано лінійний масив цілих чисел `b[]`, що містить 12 елементів. Знайти парні елементи вхідного масиву, розташовані на непарних місцях та помістити їх в масив `rez[]`.

Програму рішення поставленої задачі наведено в лістингу 3.2

Лістинг 3.2

```
#include <iostream.h>
#include <conio.h>
int main (void)
{
int b[12]; //опис вхідного масиву
int rez[12]; //опис вихідного масиву
int i;
int k=0; //кількість шуканих елементів
```

```

    clrscr;

for (i=0;i<12;i++)
{ cout<<endl<<"Введите элемент массива b["<<i+1<<"]: ";
  cin>>b[i];
}
//i отримує тільки парні значення, що відповідають непарним місцям
for (i=0; i<12; i=i+2)
//якщо елемент масиву b, розташований на непарному місці, парний
if (b[i]%2==0) { rez[k]=b[i]; //заносимо його в масив rez
    k++; // збільшуємо k
}

if (k==0)
    cout<<endl<<"Нет искомых элементов";
else
{ cout<<endl<<"Результирующий массив";
  for (i=0; i<k; i++)
    cout<<endl<<rez[i];
}

cout<<endl<<"Для завершения рвботы нажмите Enter ";
getch();
return 0;

}

```

4 Алгоритми пошуку екстремумів

Знаходження екстремумів полягає у знаходженні максимального та/або мінімального елемента в масиві.

Одним з класичних алгоритмів знаходження екстремумів є наступний. Спочатку за максимальний (мінімальний) елемент береться будь-який (наприклад, нульовий) елемент масиву та заноситься в змінну «поточного» максимуму (мінімуму). Потім, за допомогою циклу від першого до останнього елемента масиву необхідно порівнювати змінну «поточного» максимуму (мінімуму) з елементами масиву i , якщо черговий елемент масиву більше (менше) «поточного» максимуму (мінімуму), необхідно зберігати цей елемент в відповідній змінній. Після проходження всього масиву в змінній «поточного» максимуму (мінімуму) буде міститись шуканий елемент.

Нехай

```

int data[5];    // вхідний масив з 5-ти цілих чисел
int max;       // змінна «поточного» максимуму
int min;       // змінна «поточного» мінімуму

```

Блок – схему алгоритму пошуку екстремумів масиву data[] наведено на рисунку

1.1

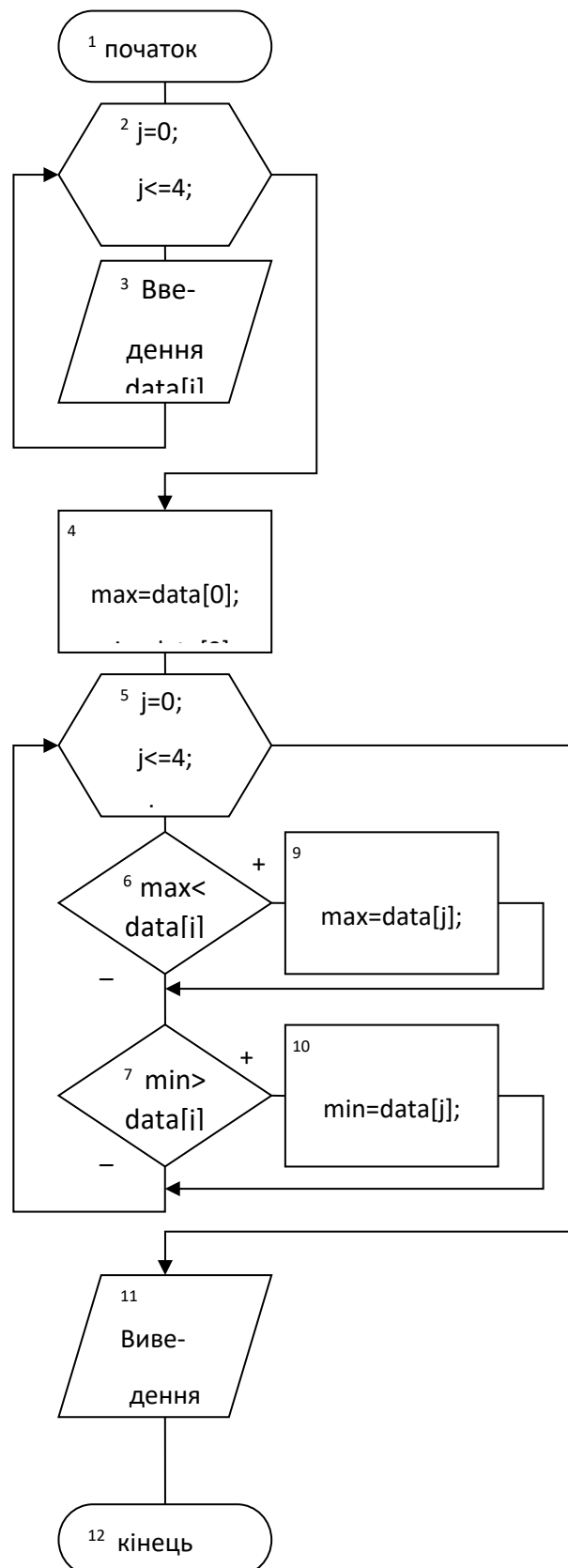


Рисунок 1.1 – Блок-схема алгоритму пошуку екстремумів.

Текст програми пошуку екстремумів лінійного масиву наведено у лістингу 1.1

Лістинг 1.1

```
#include <stdio.h>
int main()
{
    int data[5];
    int j,max,min;
    // Цикл введення масиву
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    // «Поточні» максимум та мінімум отримують стартові значення
    max=data[0];
    min=data[0];
    /* Цикл порівняння «поточних» екстремумів з кожним елементом
    i, в разі необхідності заміни їх значень */
    for (j=0; j<=4; j++)
    {
        if (max<data[j]) max=data[j];
        if (min>data[j]) min=data[j];
    }
    printf("\n Max=%i \n Min=%i", max, min);
    return 0;
}
```

Якщо масив містить декілька максимальних та/або мінімальних елементів, то приведена програма знайде перші з них за розташуванням в масиві.

Ускладнимо попередню задачу та знайдемо мінімальний непарний елемент лінійного масиву цілих чисел data[]. Для цього спочатку, в окремому циклі, знайдемо перший непарний елемент масиву і занесемо його в змінну «поточного» мінімуму. До речі, цей цикл допоможе нам з'ясувати, чи є у вхідному масиві непарні елементи.

Текст програми пошуку мінімуму серед непарних елементів лінійного масиву наведено у лістингу 1.2

Лістинг 1.2

```
#include <stdio.h>
int main()
{
    int data[5];
    int j,max,min;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    // пошук першого непарного елементу в масиві
    j=0;
    while (j<5 && data[j]%2==0) j++; //пропускаємо всі парні на
```



```

//початку масиву
if (j==5) //якщо кількість парних рівна розміру масиву
{
    printf("\n В масиві відсутні непарні елементи");
    return 0; //вихід на кінець
}
// пошук мінімуму, починаючи з першого непарного
min=data[j];
for (int i=j+1; i<=4; i++)
{
    //якщо черговий елемент непарний і менше «поточного» мінімуму
    if (data[i]%2!=0 && min>data[i])
        min=data[i]; //змінюємо значення «поточного» мінімуму
}
printf("\n Min=%i", min);
return 0;
}

```

5 Формування псевдодинамічних масивів.

При описі масиву в програмі треба обов'язково вказувати кількість елементів масиву для того, щоб компілятор виділив під цей масив потрібну кількість пам'яті. Це не завжди буває зручно, оскільки число елементів в масиві може змінюватися залежно від вирішуваного завдання. В цьому випадку в програмі формують псевдодинамічні масиви. Псевдодинамічні масиви реалізуються таким чином:

1) при визначенні масиву виділяється достатньо велика кількість пам'яті:

```

const int MAX_SIZE=100; //іменована константа
int mas[MAX_SIZE];

```

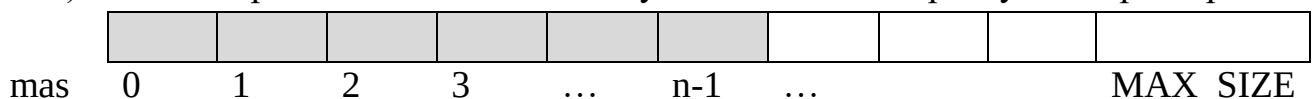
2) користувач вводить реальну кількість елементів масиву, але меншу за MAX_SIZE.

```

int n;
cout<<"\n Введіть розмір масиву, меншу за "<<MAX_SIZE<<": ";
cin>>n;

```

3) подальша робота з масивом обмежується заданою користувачем розмірністю n.



Як бачимо, під масив mas[] виділяється блок пам'яті фіксованого розміру, що вміщує MAX_SIZE цілих чисел. А кожного разу масив обробляється до елемента з індексом n-1. Тому цей масив і називається псевдодинамічним.

6 Алгоритми сортування лінійних масивів

Сортування масивів полягає у розстановці елементів масиву у певному порядку – за зростанням або за спаданням. В залежності від способу порівняння і обміну елементів розрізняють різні типи алгоритмів сортування.

Основними алгоритмами сортування є:

- сортування масиву методом вибору;
- сортування обмінами («бульбашковий» метод сортування);
- сортування простими вставками.

6.1 Сортування масиву методом вибору.

Одномірний масив із n цілих чисел відсортуємо за зростанням значень елементів методом вибору. Алгоритм полягає в тому, що вибирається найменший елемент і міняється місцями з першим елементом масиву, потім розглядаються елементи масиву, починаючи із другого, і найменший з них міняється місцями із другим елементом, і так далі $n-1$ раз (при останньому проході циклу при необхідності міняються місцями передостанній й останній елементи масиву).

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуємо його за зростанням елементів.

Блок – схема алгоритму сортування за зростанням одномірного масиву методом вибору представлена на рисунку 3.1

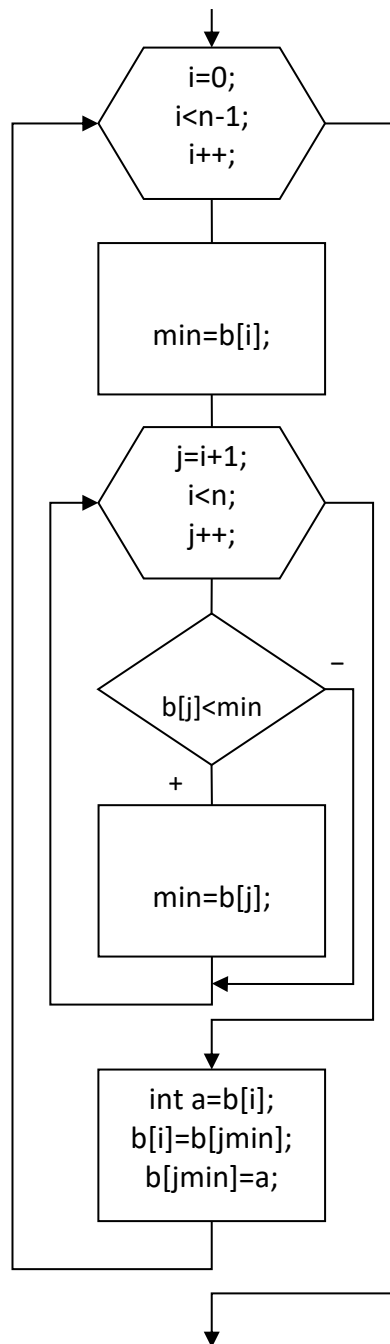


Рисунок 3.1. Блок-схема алгоритму сортування вибором.

Текст програми сортування за зростанням одномірного масиву методом вибору представлено в лістингу 3.1

Лістинг 1.1

```

#include <iostream.h>
#include <conio.h>
int main()
{
    //Організація псевдо динамічного масиву
    int b[100];          //виділення пам'яті під масив цілих
                        //максимальної довжини

```

```

int n;    // фактична кількість елементів масиву
int min, jmin; // мінімальний елемент та його номер
int i, j;
cout<<endl<<"Введіть кількість елементів в масиві";
    cin >> n;
// Введення масиву із клавіатури
for (i = 0; i<n; i++)
{
    cout<<endl<<"Введіть елемент масива b["<<i+1<<"] ";
    cin >> b[i];
}

for (i=0; i<n-1; i++) // n-1 раз шукаємо найменший елемент
{
    // приймаємо за найменший перший з розглянутих елементів
    min = b[i]; jmin=i; // та запам'ятовуємо його номер
    // пошук мінімального елемента з неупорядкованих:
    for (j = i + 1; j<n; j++)
        if (b[j] < min)
            // якщо знайшли менший елемент, запам'ятовуємо його та його номер
            { min=b[j]; jmin = j;}
    // обмін елементів з номерами i та jmin
    int a = b[i];
    b[i] = b[jmin];
    b[jmin] = a;
}
// Виведення впорядкованого масиву
cout<<endl<<"Упорядоченный массив";
for (i = 0; i<n; i++)
    cout<< endl<<b[i];
cout<<endl<<"Для завершения программы нажмите Enter";
getch();
return 0;
}

```

6.2 Сортвання обмінами («бульбашковий» метод сортвання)

Одномірний масив із n цілих чисел відсортуюмо за зростанням значень елементів. Алгоритм полягає в тому, що при перегляді вхідного масиву попарно порівнюються сусідні елементи. Якщо порядок їхнього проходження не відповідає заданому критерію впорядкованості, то елементи міняються місцями. В результаті одного такого перегляду при сортванні за збільшенням елементів елемент з найбільшим значенням переміститься на останнє місце в масиві. При наступному проході на своє місце переміститься другий за величиною елемент і т.д. Для постановки на свої місця n елементів слід зробити $n-1$ проходів. При кожному черговому проході кількість елементів, що порівнюються, треба зменшувати на одиницю.

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуюмо його за зростанням елементів.

Блок – схема сортування за зростанням одномірного масиву обмінами наведено на рисунку 3.2

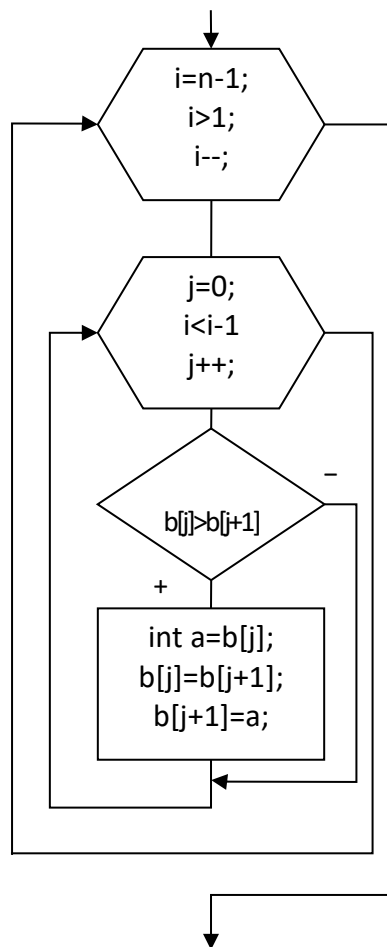


Рисунок 3.2 Блок – схема алгоритму сортування обмінами.

Текст програми сортування за зростанням одномірного масиву обмінами представлено в лістингу 3.2

Лістинг 3.2

```

#include <iostream.h>
#include <conio.h>
int main()
{
    //Організація псевдо динамічного масиву
    //виділення пам'яті під масив цілих максимальної довжини
    int b[100];
    int n;                // фактична кількість елементів масиву
    int min, jmin;        //мінімальний елемент та його номер
    int i, j;
    cout<<endl<<"Введіть кількість елементів в масиві";
    cin >> n;
    // Введення масиву із клавіатури
    for (i = 0; i<n; i++)
    {

```

```

    cout<<endl<<"Введіть елемент масива b["<<i+1<<" ] ";
    cin >> b[i];
}
//Цикл проходів по масиву
for (i=n-1; i>1; i--)
{ // цикл попарного порівняння сусідніх елементів
    for (j =0; j<=i-1; j++)
        if (b[j]>b[j+1])
        { // обмін елементів з номерами j та j+1
            int a = b[j];
            b[j] = b[j+1];
            b[j+1] = a;
        }
    }
// Виведення впорядкованого масиву
cout<<endl<<"упорядоченный массив";
for (i = 0; i<n; i++)
    cout<< endl<<b[i];
cout<<endl<<"Для завершения программы нажмите Enter";
getch();
return 0;
}

```

6.3 Сортування простими вставками

Одномірний масив $b[]$ із n цілих чисел відсортуємо за зростанням значень елементів методом вибору. Ідея алгоритму: для кожного елемента масиву $b[i]$ вважаємо, що попередні елементи вже відсортовані. Елемент $b[i]$ потрібно вставити на відповідне місце в вже відсортовану послідовність $b[0] \dots b[i-1]$. Це місце визначається послідовним порівнянням $b[i]$ з вже впорядкованими елементами i , якщо необхідно - елементи міняються місцями.

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуємо його за зростанням елементів.

Блок – схема сортування за зростанням одномірного масиву вставками наведена на рисунку 3.3

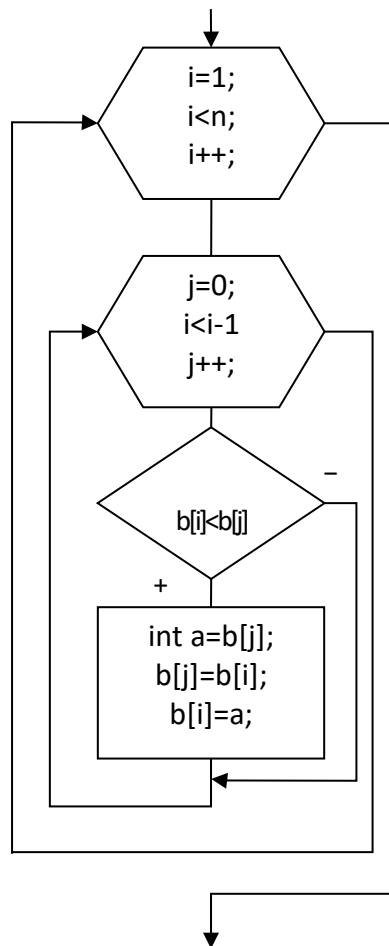


Рисунок 3.3 Блок – схема алгоритму сортування вставками.

Текст програми сортування за зростанням одномірного масиву вставками представлено в лістингу 3.3

Лістинг 3.3

```

#include <iostream.h>
#include <conio.h>
int main()
{
//Організація псевдо динамічного масиву
//виділення пам'яті під масив цілих максимальної довжини
int b[100];
int n; // фактична кількість елементів масиву
int i, j;
cout<<endl<<"Введіть кількість елементів в масиві";
    cin >> n;
// Введення масиву із клавіатури
for (i = 0; i<n; i++)
{
    cout<<endl<<"Введіть елемент масива b["<<i+1<<"] ";
    cin >> b[i];
}
// цикл порівняння b[i]-го елемента з усіма попередніми
  
```

```

for (i=1; i<n; i++)
{
    for (j =0; j<=i-1; j++)
        if (b[i]<b[j])
        {
            // обмін місцями елементу b[j]-го з b[i]-м
            int a = b[j];
            b[j] = b[i];
            b[i] = a;
        }
}
// Виведення впорядкованого масиву
cout<<endl<<"Упорядоченный массив";
for (i = 0; i<n; i++)
cout<< endl<<b[i];
cout<<endl<<"Для завершения программы нажмите Enter";
getch();
return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дайте визначення поняття "масив".
- 2 Що називається розміром масиву?
- 3 Як розташовані у пам'яті елементи масиву?
- 4 Що спільного мають всі елементи масиву?
- 5 Чим відрізняються всі елементи масиву?
- 6 Наведіть загальну форму оголошення масиву.
- 7 Що називається індексом елемента масиву?
- 8 З якого номера починається нумерація елементів в масиві?
- 9 Назвіть способи ініціалізації масиву.
- 10 Чи можна змінювати розмір масиву у пам'яті під час роботи програми?
- 11 Що необхідно вказати, щоб звернутися до елемента масиву?
- 12 Опишіть алгоритм пошуку максимального (мінімального) елемента масиву.
- 13 Що треба змінити в наведеній програмі пошуку екстремумів (див. Лістинг 1.1), щоб отримати останні за розташуванням в масиві екстремальні елементи (тобто останній мінімум та останній максимум)?
- 14 В якому випадку максимальний елемент масиву дорівнюватиме мінімальному?
- 15 Як сформулювати псевдодинамічний масив?
- 16 Який фактичний розмір псевдо динамічного масиву?
- 17 Чому масив називається псевдодинамічним?
- 18 Яка ідея полягає в основі алгоритму сортування обмінами?

- 19 Опишіть алгоритм сортування методом вибору?
- 20 Опишіть алгоритм сортування вставками?
- 21 Скільки проходів по масиву відбувається при сортуванні обмінами? вибором? вставками?
- 22 На вхід подається відсортований масив. Як вдосконалити метод обмінами так, щоб сортування припинялось вже після першого проходу по масиву?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Написати програму, що вводить із клавіатури одномірний масив з 5 цілих чисел, після чого виводить кількість ненульових елементів.

2 Дано лінійний масив із 10 дійсних чисел. Написати програму, що перевіряє, чи перебуває уведене із клавіатури число в масиві.

3 Написати програму, що обчислює середню (за тиждень) температуру повітря. Вихідні дані повинні вводитися під час роботи програми. Рекомендований вид екрану наведено нижче (дані, введені користувачем, виділені напівжирним шрифтом).

Введіть температуру воздуха за неделю.

Понедельник **12**

Вторник **10**

Среда **16**

Четверг **18**

Пятница **17**

Суббота **16**

Воскресенье **14**

Средняя температура за неделю: **14.71 град.**

4 Написати програму, що перевіряє, чи представляють елементи уведеного із клавіатури масиву зростаючу послідовність.

5 Написати програму, що обчислює, скільки разів уведене із клавіатури число зустрічається в масиві.

6 Написати програму, що перевіряє, є чи в уведеному із клавіатури масиві елементи з однаковим значенням.

7 Для заданого одномірного масиву дійсних чисел знайти останній мінімальний елемент та вивести його номер.

8 В заданому одномірному масиві дійсних чисел поміняти місцями мінімальний та максимальний елементи.

9 Для заданого одномірного масиву дійсних чисел знайти мінімальний серед від'ємних елементів масиву та вивести його номер.

- 10 В веденому масиві цілих чисел знайти мінімум парних елементів масиву та максимум непарних елементів. Вивести знайдені екстремуми або повідомити, що відповідні елементи в масиві відсутні.
 - 11 В заданому одномірному масиві дійсних чисел відсортувати за зростанням тільки непарні елементи використовуючи метод вставками.
 - 12 Написати програму, що методом прямого вибору сортує за спаданням введений із клавіатури одномірний масив.
 - 13 Написати програму, що методом обміну («бульбашковим» методом) сортує за спаданням введений із клавіатури одномірний масив.
- Введені з клавіатури два масиви дійсних чисел впорядкувати за зростанням. Об'єднати два впорядковані масиви в один, теж впорядкований

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 8

з навчальної дисципліни Технології (Програмування)

Тема Багатомірні масиви даних. Використання конструкцій C/C++ для опису, ініціалізації та обробки матриць.

Мета заняття Пояснити студентам поняття багатомірного масиву даних, представлення його в пам'яті ЕОМ, опис масиву, способи ініціалізації масиву. Розглянути алгоритми обробки матриць.

Час – 2 години

План проведення лекції

- 1 Багатомірні масиви. Матриці. Опис, стартова ініціалізація.
- 2 Використання конструкцій мови C/C++ для обробки матриць.

Навчальні матеріали лекції

- 1 Багатомірні масиви. Матриці. Опис, стартова ініціалізація

У мові C/C++ масиви бувають не тільки одновимірні, але і багатовимірні. Відмінність полягає в тому, що в одновимірному масиві положення елемента визначається одним індексом, а в багатовимірному – декількома. Теоретично на число розмірностей масиву ніяких обмежень не накладається. При визначенні багатомірних масивів задається тип компонентів масиву, ім'я масиву, а потім, в окремих квадратних дужках, число елементів у першому "вимірі", другому, третьому й т.д.

Загальна форма оголошення багатовимірного масиву:

тип ім'я_масиву[індекс_1][індекс_2]...[індекс_n];

де тип – визначає тип даних елементів, з яких буде складатися масив,

ім'я_масиву – ідентифікатор масиву,

індекс_n – кількість елементів масиву у кожній його розмірності.

Елементи багатовимірного масиву розташовуються в послідовних елементах оперативної пам'яті за збільшенням адрес. Між елементами немає розривів. У пам'яті ЕОМ елементи розташовуються рядковим чином так, що найшвидше міняється останній індекс.

Приклад розташування в пам'яті ЕОМ тривимірного масиву `int d[2][2][2]`

`d[0][0][0] d[0][0][1]`

`d[0][1][0] d[0][1][1]`

`d[1][0][0] d[1][0][1]`

`d[1][1][0] d[1][1][1]`

Простіший багатовимірний масив – двовірний. Двовірний масив являє собою список одноірних масивів. Для того щоб оголосити двовірний масив необхідно задати лише дві розмірності.

```
тип ім'я_масиву[індекс_1][індекс_2];
```

де тип – визначає тип даних елементів, з яких буде складатися масив,
ім'я_масиву – ідентифікатор масиву,
індекс_1 – кількість рядків матриці,
індекс_2 – кількість стовпців матриці.

Наприклад, для оголошення двовірного масиву цілих чисел розмірністю 10 рядків та 20 стовпців, необхідно написати:

```
int twod [10][20];
```

В двовірному масиву позиція кожного елемента визначається двома індексами. Якщо уявити двовірний масив у вигляді таблиці даних, то перший індекс відповідає рядку, а другий – стовпцю. Якщо доступ до елементів масиву уявити в порядку, в якому вони реально зберігаються у пам'яті, то другий індекс буде змінюватись швидше, ніж перший.

Приклад розташування в пам'яті ЕОМ двовірного масиву `int d[3][4]` наведений у таблиці 1.1.

Таблиця 1.1 – Приклад розташування в пам'яті ЕОМ двовірного масиву

стовпці рядки	0	1	2	3
0	d[0][0]	d[0][1]	d[0][2]	d[0][3]
1	d[1][0]	d[1][1]	d[1][2]	d[1][3]
2	d[2][0]	d[2][1]	d[2][2]	d[2][3]

Для того щоб отримати доступ до елемента матриці необхідно вказати ім'я масиву та два індекси – індекс рядка та індекс стовпця. Наприклад, для доступу до елемента масиву `twod` з координатами 3,5, необхідно записати:

```
twod[3][5];
```

Необхідно пам'ятати, що місце зберігання всіх елементів масиву визначається під час компіляції. Окрім того, пам'ять, виділена для зберігання масиву, використовується в продовж всього терміну існування масиву. Для визначення кількості байт пам'яті, яку займає двовірний масив, необхідно використовувати наступну формулу:

*число байт = кількість рядків * кількість стовпців * розмір типу в байтах*

Відповідно, двовірний цілочисельний масив, описаний як

```
int mas [10][5];
```

займає в пам'яті $10 \times 5 \times 4 = 200$ байт.

При стартовій ініціалізації багатовимірного масиву список ініціалізаторів кожної розмірності (підгрупу ініціалізаторів) можна заключити у фігурні дужки. Наприклад, для ініціалізації масиву `mas[3][5]` можна використати вираз:

```
int mas[3][5]={
    {1, 5, -5, 0, 7},
    {0, -5, 100, 20, 19},
    {0, -2, -3, -4, 10}
};
```

Початкових значень, як й в одномірних масивах, може бути менше, ніж число елементів масиву. Іншим елементам буде привласнюватися нульове значення. Наприклад:

```
int m[2][3]={ { 10, 20 }, { 30.40 } };
```

Це еквівалентно наступному явному визначенню:

```
int m[2][3]={ 10, 20, 0, 30, 40, 0 };
```

Подібно одномірним масивам, число початкових значень не повинне перевищувати числа елементів у рядку:

```
int m[2][3]={ {10,20,30,40}, {50,60} }; // помилка!
```

Метод визначення розміру масиву шляхом специфікації початкових значень може використатися й для багатомірних масивів, але тільки частково. Дозволяється опускати число рядків, але потрібно задавати число стовпців. Компілятор буде підраховувати число початкових значень і визначати число рядків.

```
int b[ ][ 3 ]={{10,20,30},{ 0,50,60}};
```

Незалежно від того, указується число рядків, чи ні, не можна опускати число стовпців, це помилка:

```
int m[2][ ]={{10,20,30},{40,50,60}} // помилка!
```

Теоретично компілятор може підраховувати групи рядків і розуміти структуру матриці, однак він цього не робить. Можливо, краще задавати розмірність масивів явно.

2 Використання конструкцій мови C/C++ для обробки матриць

При переборі елементів багатомірного масиву варто використати вкладені цикли. У вкладених циклах багатомірних масивів внутрішній цикл завершує всі свої операції спочатку для однієї ітерації зовнішнього циклу, потім для наступної ітерації зовнішнього циклу й т.д.

Для обробки матриць, зазвичай, використовують два регулярних цикли – зовнішній для переходів по рядкам, внутрішній для переходів по стовпцям.

Розглянемо приклад коду для введення елементів матриці цілих чисел `a[][]`, що складається з 3-х рядків та 4-х стовпців, та виведення її у загально прийнятому вигляді

(див. лістинг 2.1). для введення та виведення матриці використовуються регулярні цикли for:

```
for (i=0; i<3; i++)  
for (j=0; j<4; j++)  
...
```

Внутрішній цикл змінює індекс j від 0 до 3 для кожного значення зовнішнього циклу по індексу i (який міняється від 0 до 2).

Лістинг 2.1 – Приклад коду введення елементів матриці та виведення їх на екран

```
#include<iostream.h>  
#include <conio.h>  
int main()  
{  
    int a[3][4];  
    int i,j;  
    for(i=0;i<3;i++)  
    {  
        for(j=0;j<4;j++)  
        {  
            cout<<"\n Введіть A["<<i+1<<" "<<j+1<<"]: ";  
            cin>>a[i][j];  
        }  
    }  
    system("cls"); // очищення екрану  
    for(i=0;i<3;i++)  
    {  
        for(j=0;j<4;j++)  
        {  
            cout.width(3); /*встановлення ширини  
                               виводу в 3 позиції*/  
            cout<<a[i][j];  
        }  
        cout<<endl;  
    }  
    getch();  
    return 0;  
}
```

Блок – схему алгоритму рішення даної задачі наведено на рисунку 2.1

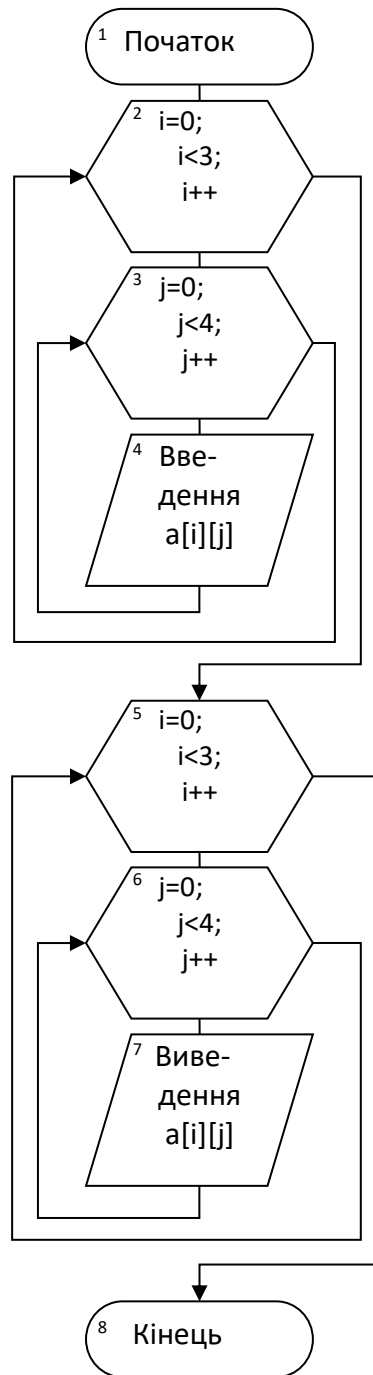


Рисунок 2.1. Блок – схема алгоритму введення – виведення матриці.

Приклад.

1 Постановка задачі.

Дано матрицю $A[3][4]$. Обчислити добуток від’ємних парних елементів у кожному з рядків.

2 Блок – схема алгоритму обчислення добутку від’ємних парних елементів у кожному з рядків матриці (див. рисунок 2.2).

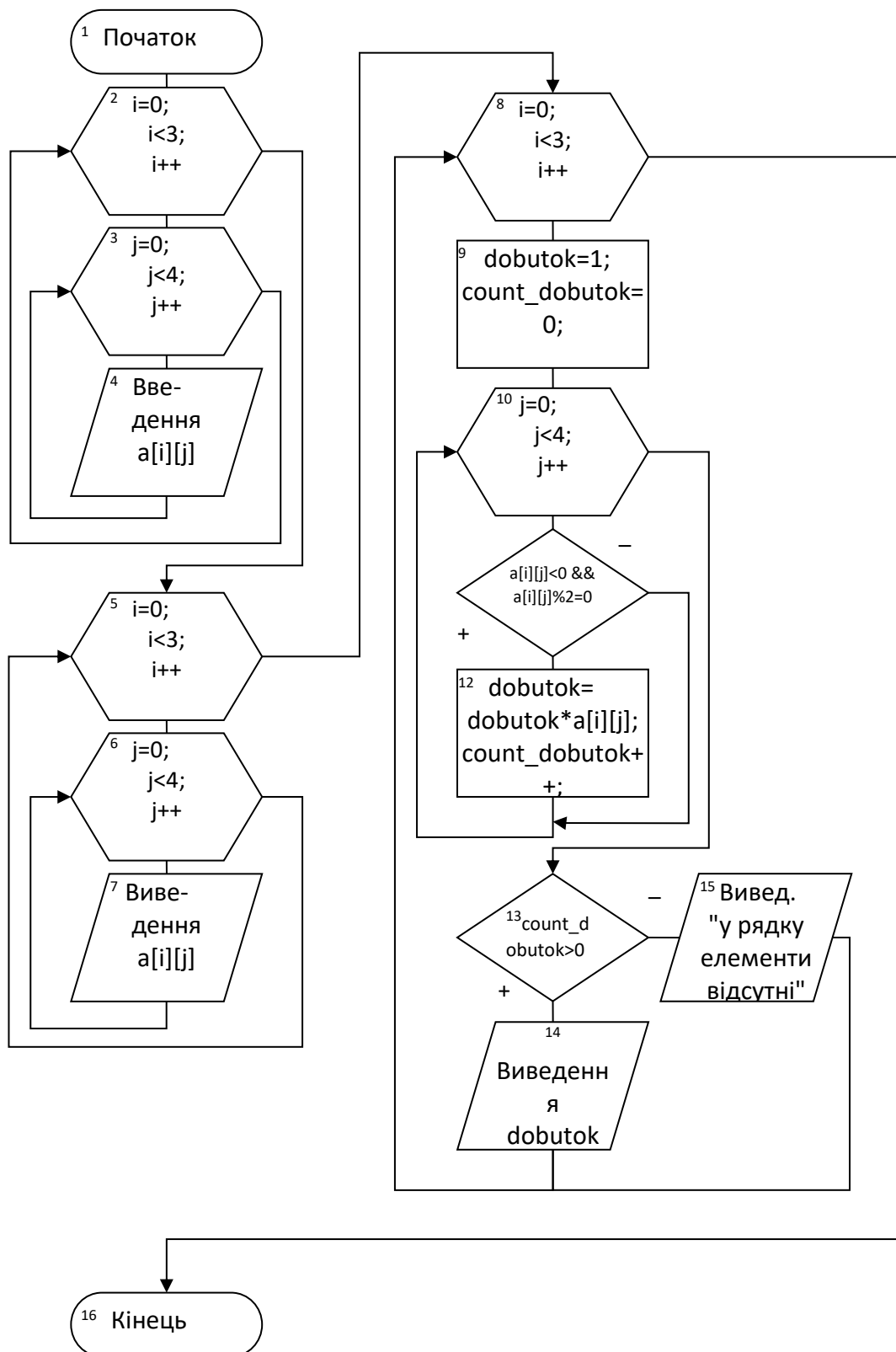


Рисунок 2.2. Блок – схема алгоритму обчислення добутку від’ємних парних елементів у кожному з рядків матриці

Лістинг 2.2 – Приклад коду програми для обчислення добутку від’ємних парних елементів у кожному з рядків матриці


```

#include <iostream.h>
#include <conio.h>
int main()
{
    int a[3][4];
    int dobutok=1;
    int count_dobutok=0;
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<<endl<<"Введіть A["<<i+1<<","<<j+1<<"]: ";
            cin>>a[i][j];
        }
    }
    system("cls");
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout.width(3);
            cout<<a[i][j];
        }
        cout<<endl;
    }
    for(i=0;i<3;i++)
    {
        dobutok=1;
        count_dobutok=0;
        for(j=0;j<4;j++)
        {
            if (a[i][j]<0 && a[i][j]%2==0)
            {
                dobutok=dobutok*a[i][j];
                count_dobutok++;
            }
        }
        if (count_dobutok>0)
            cout<<endl<<"В "<<i+1<<"-у рядку добуток від'ємних парних= "
                <<dobutok;
        else
            cout<<endl<<"В "<<i+1<<"-у рядку від'ємні парні відсутні";
    }
    getch();
    return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для актуалізації опорних знань та перевірки попереднього матеріалу.

- 23 Наведіть загальну форму оголошення масиву.
- 24 Що називається індексом елемента масиву?
- 25 З якого номера починається нумерація елементів в масиві?
- 26 Назвіть способи ініціалізації масиву.
- 27 Чи можна змінювати розмір масиву у пам'яті під час роботи програми?
- 28 Що необхідно вказати, щоб звернутися до елемента масиву?

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що називається багатовимірним масивом?
- 2 Наведіть загальну форму оголошення багатовимірного масиву.
- 3 З якого числа починається нумерація індексів багатовимірного масиву?
- 4 Як розташовується в пам'яті багатовимірний масив?
- 5 Наведіть загальну форму оголошення двовимірного масиву.
- 6 Наведіть приклад ініціалізації багатовимірного масиву.
- 7 Скільки і яку назву мають індекси, необхідні для звернення до елемента матриці?
- 8 Як визначити загальний розмір пам'яті, який займе матриця?
- 9 Які ініціалізатори використовує компілятор у випадку, коли у підгрупах ініціалізаторів не вистає членів підгруп?
- 10 Які типи циклів і в якому порядку зазвичай використовують для виконання будь-яких дій над матрицями?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Дана матриця $B(3,4)$. Знайти і роздрукувати мінімальний елемент і середнє арифметичне кожного стовпчика.
- 2 Дана матриця $A(4, 4)$. Знайти добуток та суму ненульових елементів спочатку головної діагоналі, а потім – останнього стовпчика.
- 3 Дана матриця $T(4,4)$. Знайти мінімальний елемент серед максимальних елементів стовпців.
- 4 Дані дві матриці: $A(5, 5)$ та $B(5, 5)$. Підрахувати добуток елементів головної діагоналі кожної матриці. Роздрукувати ту матрицю, у якої добуток більший.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 9

з навчальної дисципліни Технології (Програмування)

Тема Рядки символів.

Структури в C/C++. Масиви структур.

Мета заняття Ознайомити студентів поняттям рядка символів, способами-ми опису рядків символів, із різними способами введення / виведення рядків символів. Розглянути приклад обробки рядка символів.
Ввести поняття структури як складеного типу даних, що визначається програмістом. Розглянути формати визначення структурного шаблону, приклади ініціалізації структур та операцій над структурами.

Час – 2 години

План проведення лекції

- 1 Опис рядків символів. Функції введення – виведення рядків символів.
 - 1.1 Опис та стартова ініціалізація рядків символів.
 - 1.2 Введення рядків символів
 - 1.3 Виведення рядків символів
- 2 Програма обробки рядка символів
- 3 Поняття структури
- 4 Ініціалізація структур
- 5 Операції над структурами
- 6 Приклад програми обробки масиву структур

Навчальні матеріали лекції

У C ++ підтримується два типи рядків - вбудований тип, що дістався від C, і клас string стандартної бібліотеки C++. Клас string надає більше можливостей і тому зручніше в застосуванні, однак на практиці нерідкі ситуації, коли необхідно користуватися вбудованим типом.

- 1 Опис рядків символів. Функції введення – виведення рядків символів.
 - 1.1 Опис та стартова ініціалізація рядків символів.

Рядок являє собою масив символів, що закінчується нуль-символом. Нуль-символ - це символ з кодом, рівним 0, що записується у вигляді керуючої послідовності '\0'. По положенню нуль-символу визначається фактична довжина рядка.

Пам'ять під рядки, як і під інші масиви, може виділятися як компілятором, так і безпосередньо при виконанні програми. Довжина динамічного рядка може задаватися виразом, довжина нединамічного рядка повинна бути тільки константним виразом.

```
const int n = 10;  
char str[n];
```

При завданні довжини необхідно враховувати завершальний нуль-символ. Наприклад, у наведеному вище рядку можна зберігати не 10, а 9 символів.

Рядок можна ініціалізувати строковою константою, при цьому нуль-символ формується автоматично:

```
char str[10] = "Vasia";
```

В цьому прикладі виділяється масив з 10 елементів з номерами від 0 до 9 і всі символи рядка поміщаються в масив. Під цей рядок виділяється 10 байт, 5 з яких зайняті під символи рядка, а шостий - під нуль-символ. Якщо рядок при визначенні ініціалізується, то його розмірність можна опускати (компілятор сам виділить потрібну кількість байт):

```
char str[] = "Я - студент"; //виділено й заповнено 6 байт  
Як і для інших масивів, str є покажчиком на перший елемент масиву. А тому  
str == &str[0]  
*str == 'Я'  
*(str + 4) == 'с'
```

Для створення рядка можна використовувати вказівник. Наприклад:

```
char *m4 = "31 грудня - свято Нового року";
```

m4 є вказівником на рядок, і розмір масиву визначаються компілятором. В нашому прикладі в пам'яті створюється 30 елементів для запам'ятовування рядка. Але, крім того, виділяється ще один елемент пам'яті для змінної m4, що є вказівником. Спочатку ця змінна указує на початок рядка, але її значення може змінюватися. Тому допустимо використовувати операцію приросту на одиницю: ++m4 указуватиме на другий символ рядка — '1'.

Інструкція

```
Const char *str = "Vasia";
```

створює не строкову змінну, а покажчик на строкову константу, змінювати яку неможливо.

Операція присвоєння одного рядка іншому не визначена (оскільки рядок є масивом) і може виконуватися за допомогою циклу або за допомогою функцій стандартної бібліотеки. Посимвольна обробка рядків теж здійснюється в циклі.

Для розміщення рядка в динамічній пам'яті треба описати вказівник на `char`, а потім виділити пам'ять за допомогою `new`:

```
char *p = new char [m];
```

Природно, що в цьому випадку довжина рядка може бути змінною й задаватися на етапі виконання програми. Динамічні рядки, як і динамічні масиви, не можна ініціалізувати при створенні.

Доступ до рядка може здійснюватися через покажчик типу `char*`. У прикладі

```
char *st = "Ціна знання\n";
```

компілятор поміщає всі символи рядка в масив і привласнює змінній `st` адресу першого елемента масиву.

Звичайно для перебору символів рядка застосовується адресна арифметика. Оскільки рядок завжди закінчується нуль-символом, можна збільшувати покажчик на 1, поки черговим символом не стане нуль. Наприклад:

```
while (*st++)      // поки не кінець рядка символів
{ ... }
```

рядок `st` розкривається, і отримане значення перевіряється на істинність. Будь-яке відмінне від нуля значення вважається істинним і, отже, цикл закінчується тоді, коли буде досягнутий символ з кодом 0. Операція інкременту додає 1 до вказівника `st` й у такий спосіб зрушує його до наступного символу.

1.2 Введення рядків символів

Нехай

```
#include <stdio.h>
#include <conio.h>
char s[16];
```

Функція `gets()`

Функція `gets(s)` читає символи із клавіатури до появи символу `'\n'` переведення рядка й поміщає їх у рядок `s` (сам символ `'\n'` в рядок не включається, замість нього в рядок уставляється нуль-символ `'\0'`). У C++ рядки також можна вводити за допомогою методів `get()` і `getline()`.

```
#include <iostream.h>
int main() {
    const int n=10;
    char s[n];
    cin.getline(s,n); cout <<s<<endl;
    cin.get(s,n); cout <<s<<endl;
```

```
return 0;
}
```

Метод *getline(s, n)* зчитує із вхідного потоку *n-1* символів або менше (якщо символ '\n' зустрінеться раніше) і записує в рядкову змінну *s*. Символ '\n' також зчитується (видаляється) із вхідного потоку, але не записується в строкову змінну, замість нього розміщується завершальний '\0'. Якщо в рядку вихідних даних більше *n-1* символів, наступне введення буде виконуватися з того ж рядка, починаючи з першого нечитаного символу.

Метод *get()* працює аналогічно, але залишає в потоці символ '\n'. У строкову змінну додається завершальний '\0'.

Не можна звертатися до методу *get* із двома аргументами два рази підряд, не видаливши '\n' із вхідного потоку. Наприклад:

```
cin.get(s,n);      //1-зчитування рядка
cout<<s<<endl;    //2-виведення рядка
cin.get(s,n);      //3-зчитування рядка
cout<<s<<endl;    //4-виведення рядка
cin.get(s,n);      //5-зчитування рядка
cout<<s<<endl;    //6-виведення рядка
cout<<"Кінець - справі вінець"<<endl;
```

При виконанні цього фрагмента ви побачите на екрані перший рядок, виведений оператором 2, а потім завершальне повідомлення, виведене оператором

Метод *get* у цьому випадку "уткнеться" у символ '\n', залишений у вхідному потоці від першого виклику цього методу (оператор 1). У результаті будуть зчитані й, відповідно, виведені на екран порожні рядки. Можливе рішення цієї проблеми - видалити символ '\n' із вхідного потоку шляхом виклику методу *get* без параметрів, тобто після операторів 1 й 3 потрібно вставити

```
cin.get();
```

У випадку, якщо потрібно ввести кілька рядків, переважніше користуватися *getline*, наприклад:

```
#include <iostream.h>
int main(){
const int n=10;
char s[n];
while (cin.getline(s,n)){
cout <<s<<endl;
//обробка рядка
. . .
}
return 0;
}
```

1.3 Виведення рядків символів

Приклади:

```
1. char s[20];  
cin>>s; //введення рядка із стандартного потоку  
cout<<s; //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до першого пропуску, тобто в рядок s буде введено тільки перше слово рядка "123\0", отже, виведеться: "123".

```
2. char s[20];  
gets(s); //введення рядка із стандартного потоку  
puts(s); //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до символу '\n', тобто в s занесеться рядок "123 456 789\n\0", при виведенні рядка функція puts повертає ще один символ '\n', отже, буде виведений рядок "123 456 789\n\n".

```
3. char s[20];  
scanf("%s",s); //введення рядка із стандартного потоку  
printf("%s",s); //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до першого пропуску, тобто в рядок s занесеться тільки перше слово рядка "123\0", отже, виведеться: "123". Оскільки s – ім'я масиву, тобто адреса його першого елемента, операція & у функції scanf не використовується.

2 Програма обробки рядка символів

Постановка задачі:

Для введеного користувачем рядка символів підрахувати кількість слів в цьому рядку. Зауваження: словом будемо називати підпоследовність будь яких символів в рядку, що не містить пробілу.

Текст програми

```
#include <iostream.h>
```



```

#include <string.h>
#include <conio.h>
#include <stdio.h>
int main (void)
{
int i=0, j, k=0;    // k - кількість слів в рядку
char * str;
clrscr;
cout<<endl<<"Введіть строку " <<endl;
gets(str);
j=strlen(str);    // обчислення довжини введеного рядка
//цикл пропуску можливих пробілів на початку рядка символів
while ( (*(str+i)==' ') && (i<j)) i++;
// обробка рядка
while (i<j)    //цикл поки не кінець рядка символів
{
    //цикл пропуску чергового слова
    while ( (*(str+i)!=' ') && (i<j)) i++;
    k++;
    //цикл пропуску пробілів
    while ( (*(str+i)==' ') && (i<j)) i++;
}

if (k==0)
    cout<<endl<<"В строке нет слов"<<endl;
else
    cout <<endl<<"Строка содержит " <<k<< " слов"<<endl;
getch();
return 0;
}

```

3 Поняття структури

Структура – це об'єднання одного або декількох об'єктів, можливо різного типу, під одним ім'ям, яке є типом структури. В якості об'єктів можуть виступати змінні, масиви, вказівники і інші структури. Елементи структури, які називаються полями, обов'язково повинні мати різні імена. Всі поля однієї структури у сукупності називаються записом.

Структури дозволяють трактувати групи пов'язаних між собою об'єктів не як множину окремих елементів, а як єдине ціле.

Елементи структури розташовуються в пам'яті ЕОМ у тому ж порядку у якому вони оголошуються.

Формати визначення структурного шаблону наступні:

1) struct ім'я_типу // *спосіб 1*

```

{
тип1 поле1;

```



всі поля у сукупності називаються

записом

```
тип2 поле2;
. . .
типN полеN;
};
```

В даному способі оголошенні структури після закриваючої фігурної дужки "}" обов'язково ставиться крапка з комою ";".

Приклад.

```
struct Date //визначення структури
{
    int day;
    int month;
    int year;
};
Date birthday; //змінна типу Date
```

2) struct //спосіб 2

```
{
    тип1 поле1;
    тип2 поле2;
    . . .
} список ідентифікаторів;
```

Приклад.

```
struct
{
    int min;
    int sec;
    int msec;
} time_beg, time_end;
```

У першому випадку опис структури визначає новий тип, ім'я якого можна використовувати разом із стандартними типами.

У другому випадку опис структури служить визначенням змінних.

3) Структурний тип можна також задати за допомогою ключового слова typedef.

```
typedef struct //спосіб 3
{
    тип1 поле1;
    тип2 поле2;
    . . .
} тип_структури;
```

Приклад.

```
typedef struct {
```

```

float re;
float im;
} complex;
complex a[100]; //масив з 100 комплексних чисел.

```

4 Ініціалізація структур

Ініціалізація структур може бути здійснена двома способами:

- привласнення значень елементам структури в процесі оголошення змінної, яка відноситься до типу структури;
- привласнення початкових значень елементам структури з використанням потоків введення / виведення `cin`, `cout` та функцій – `scanf()`, `printf()`.

Для ініціалізації структур в процесі її оголошення значення її полів перераховують у фігурних дужках.

Загальний вигляд ініціалізації структури в процесі оголошення.

```

struct тип_структури ім'я_змінної=
{значення_поля_1,
 значення_поля_2,
 . . . ,
 значення_поля_n};

```

Приклади:

```

1 struct Student
{
char name[20];
int kurs;
float rating;
};
Student s={"Іванов",1,3.5};

2 struct
{ char name[20];
char title[30];
float rate;
} employee = {"Петров", "директор",10000};

```

5 Операції над структурами

Операція привласнення.

Для змінних одного і того ж структурного типу визначена операція привласнення. При цьому відбувається по елементне копіювання.

Приклад.

```
Student s;  
...  
Student ss=s;
```

Доступ до елементів структур.

Доступ до елементів структур забезпечується за допомогою оператора "." та уточнених імен.

Загальний вигляд доступу до елементів структури.

```
Ім'я_структури.ім'я_поля
```

Приклад.

```
struct employee  
{ char name[20];  
  char title[30];  
  float rate;  
};  
employee emp1 = {"Петров", "директор", 10000};
```

- emp1.name - доступ до поля name (рядок "Петров");
- emp1.rate - доступ до поля rate. Змінна цілого типу із значенням 10000.

6 Приклад програми обробки масиву структур

Постановка задачі:

Написати програму, яка буде вводити дані про 5-х студентів та визначати студента з максимальним рейтингом. Дані зберігати у масиві структур. Кожний запис повинен містити: П.І.Б. студента, групу, середній бал успішності (рейтинг).

Блок-схема алгоритму рішення задачі (дивись рисунок 3.1):

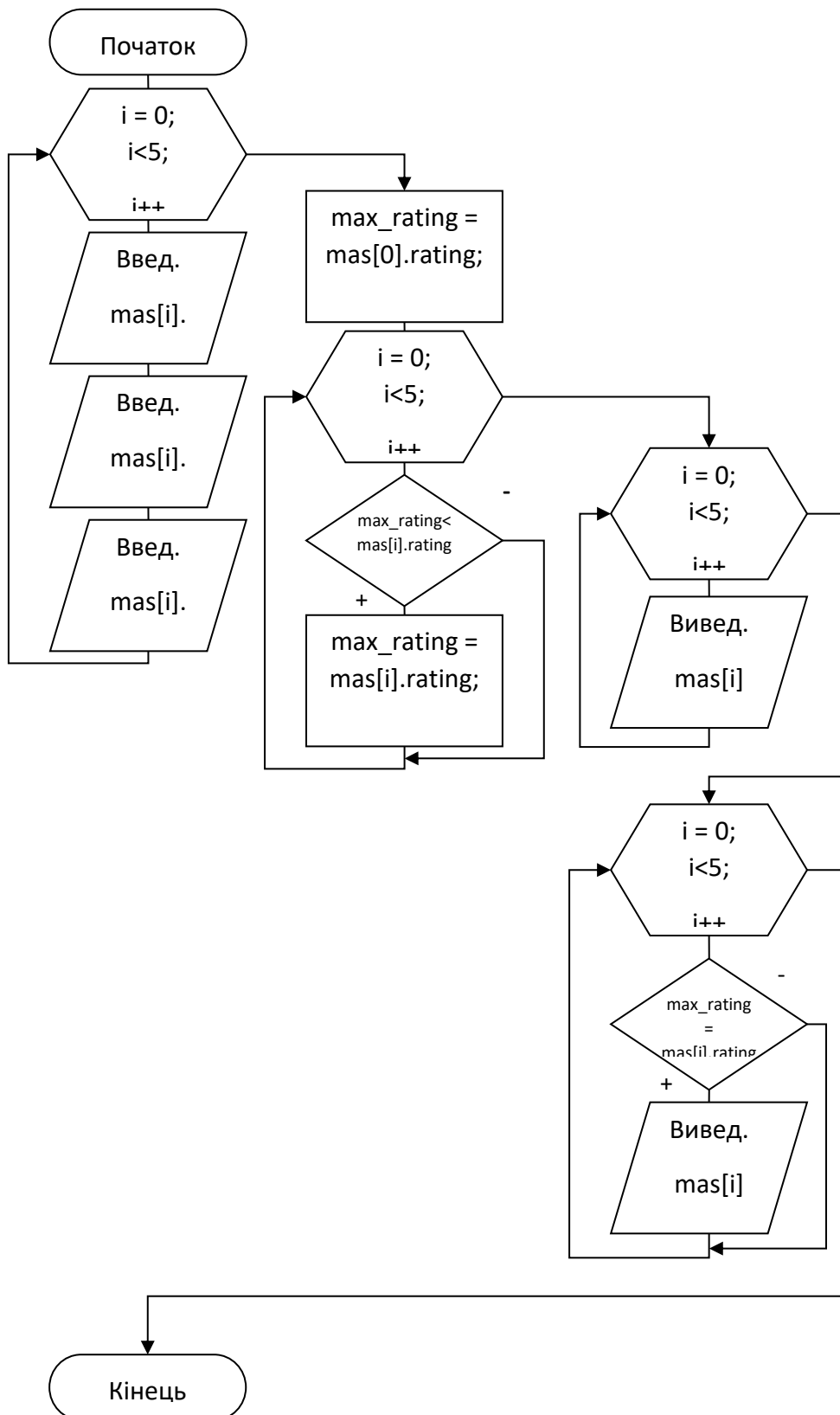


Рисунок 3.1. Блок-схема алгоритму рішення задачі:

Текст програми обробки масиву структур наведено у лістингу 3.1

Лістинг 3.1

```
#include <iostream.h>
#include <conio.h>
struct Student
{char name[30];
char group[10];
float rating;
};
int main()
{
Student mas[5];
float max_rating;
clrscr();
//введення значень масиву
for(int i=0;i<5;i++)
{
cout<<endl<<"Enter name:";
cin>>mas[i].name;
cout<<endl<<"Enter group:";
cin>>mas[i].group;
cout<<endl<<"Enter rating:";
cin>>mas[i].rating;
}
//визначення максимального рейтингу студента
max_rating=mas[0].rating;
for(int i=0;i<5;i++)
{
if(max_rating<mas[i].rating)
{
max_rating=mas[i].rating;
}
}

//виведення введених студентів на екран
clrscr();
for(int i=0;i<5;i++)
{
cout<<endl<<endl<<"Student # "<<i+1;
cout<<endl<<"\t Name:"<<mas[i].name;
cout<<endl<<"\t Group:"<<mas[i].group;
cout<<endl<<"\t Rating:"<<mas[i].rating;
}

//виведення студента (-ів) з максимальним рейтингом
cout<<"Students have a max rating:";
for(int i=0;i<5;i++)
{
if(max_rating==mas[i].rating)
{
cout<<endl<<endl<<"Student # "<<i+1;
cout<<endl<<"\t Name:"<<mas[i].name;
```

```

cout<<endl<<"\t Group:"<<mas[i].group;
cout<<endl<<"\t Rating:"<<mas[i].rating;
}
}
getch();
return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Який тип даних визначає ключове слово char?
- 2 Яку довжину має тип даних char?
- 3 Який діапазон значень може приймати символьний тип даних?
- 4 Як задаються константи типу char?
- 5 Які операції застосовуються до типу char?
- 6 Опишіть призначення функції getchar().
- 7 Опишіть призначення функції putchar().
- 8 Дати визначення структури як типу даних.
- 9 Які формати визначення структурного шаблону ви знаєте?
- 10 Як здійснюється ініціалізація структур?
- 11 Як виконується операція привласнення для структур?
- 12 Як здійснюється доступ до елементів структури?

Завдання для самоперевірки засвоєного матеріалу на лекції

Розробити програми обробки рядка символів за вказаним алгоритмом.

- 1 Зауваження: словом в рядку символів будемо вважати підпоследовність символів, що не містить пробілу всередині себе.
- 2 Видалити із рядка символів найдовше слово (найдовші слова, якщо їх кілька).
- 3 Продублювати в рядку символів найдовше слово (останнє з найдовших слів, якщо їх декілька).
- 4 Поміняти місцями у реченні перше та найдовше слова (перше та останнє з найдовших, якщо їх декілька).
- 5 Записати в реченні символи найдовшого слова (усіх найдовших, якщо їх декілька) у зворотному порядку.
- 6 Поміняти місцями в реченні слова: перше із другим, третє з четвертим, тощо. Якщо слів непарна кількість, то спочатку останнє слово із рядка символів видалити.

- 7 Найдовше слово (перше з найдовших, якщо їх декілька) розбити на два слова навпіл. Якщо у цьому слові непарна кількість символів, то середній символ слова видалити.

Розробити конструкцію структури, що дозволить містити наступну інформацію про банківських клієнтів: Номер, Прізвище, Баланс.

8 Описати тип Tdate – структуру з полями цілого типу: Day (день), Month (місяць), Year (рік) і функцію логічного типу з параметром типу Tdate, що повертає значення true, якщо рік в даті високосний, і false – якщо ні.

9 Скласти програму, яка забезпечить введення з клавіатури даних мінімум про 5 студентів коледжу (ПІБ, рік народження, адреса, курс, група) та виведення на екран даних про студентів, які народились у 1997 році. В програмі повинні бути присутні такі поля запису: `pib`, `rik_n`, `adres`, `curs`, `grup`.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 10

з навчальної дисципліни Технології (Програмування)

Тема Вказівники: визначення, опис, операції над вказівниками.
Обробка статичних масивів даних з використанням вказівників

Мета заняття Пояснити студентам поняття та призначення вказівника.
Навчити описувати вказівники. Пояснити, які операції виконуються вказівниками та правила виконання цих операцій. Пояснити студентам зв'язок між ім'ям статичного масиву і вказівником. Навчити використовувати вказівники для звернення до елементів лінійних масивів та матриць.

Час – 2 години

План проведення лекції

- 1 Вказівники: визначення, опис.
- 2 Операції над вказівниками.
- 3 Масиви і вказівники.

Навчальні матеріали лекції

- 1 Вказівники: визначення, опис.

Вказівник – змінна, що містить адресу об'єкту, наприклад такого, як змінна. Наприклад, якщо змінна *x* містить адресу змінної *y*, то про змінну *x* кажуть, що вона «вказує» на змінну *y*.

Поняття «вказівник» можна пояснити, використовуючи спрощену схему організації пам'яті ЕОМ (див. рисунок 1.1). Як правило, пам'ять ЕОМ можна представити у вигляді послідовності пронумерованих одnobайтових комірок, з якими можна працювати окремо або блоками. Вказівник – це теж змінна, яка розміщується в пам'яті. Зазвичай вказівники займають 2 або 4 байти – залежно від моделі пам'яті. На рисунку 1.1 – змінна "с" має тип `char`, вказівник "р" містить адресу змінної "с". Взаємозв'язок змінних "р" і "с" показана стрілкою.

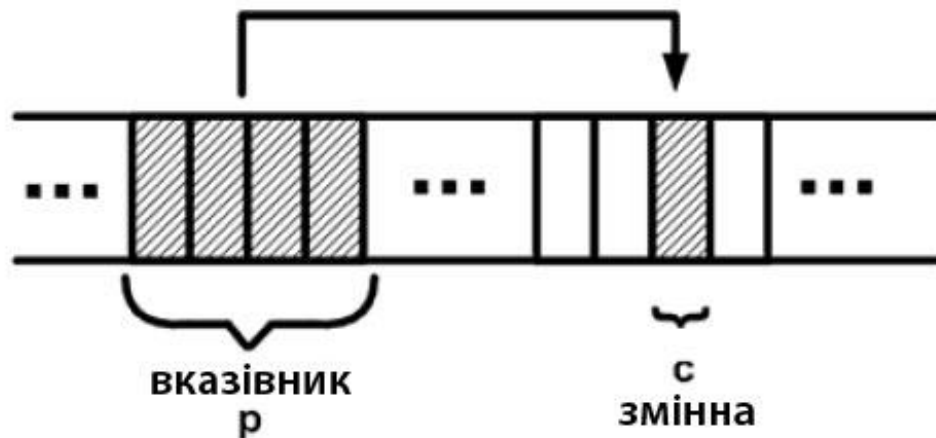


Рисунок 1.1 – Схематичне розміщення взаємодії вказівників та змінних

Вказівники застосовуються:

- для доступу до елементів оперативної пам'яті і створення нових об'єктів в ході виконання програми;
- для доступу до складних елементів даних;
- для виконання різних операцій з елементами масиву.

□ Вказівник не є самостійним типом, він завжди пов'язаний з яким-небудь іншим конкретним типом.

Вказівники оголошуються так:

тип *ідентифікатор;

де тип може бути будь-яким, окрім посилання, ідентифікатор – формується як будь-який ідентифікатор.

При оголошенні вказівника зірочка(*) дає компіляторові зрозуміти, що це саме вказівник. Наприклад:

```
int *a; //a - вказівник на змінну цілого типу
```

```
double *things; //вказівник на змінну дійсного типу
                //подвійній точності
```

```
const int *pci; //pci - вказівник на константу цілого типу
```

Зірочка ставиться безпосередньо до імені, тому для того, щоб об'явити кілька вказівників, потрібно ставити її перед ім'ям кожного з них. Наприклад:

```
int *a, b, *c;
```

(а й с - вказівники на цілую змінну, b - цілочисельна змінна).

Вказівник може вказувати на дані з атрибутом const і сам мати атрибут const. Наприклад:

```
constchar *p1; /*опис вказівника на константний об'єкт типу char,
сам вказівник не є константою;*/
```

```
int *constp2; /*опис константного вказівника на неконстантний
об'єкту типу int. Значення, що записане в комірку, може змінюватись, але
перевести вказівник на іншу комірку не можна;*/
constfloat *constp3; /*опис константного вказівника на
константний об'єкт типу float.*/
```

2 Операції над вказівниками

Для роботи з вказівниками в C/C++ визначено наступні операції:

1) Операція присвоєння (=)

Змінній – вказівнику можна присвоювати тільки адресу.

2) Операція отримання адреси (&) – дозволяє визначити адресу об'єкту.

Операція & видає адресу об'єкту. Нехай маємо:

```
int c; int *p;
```

тоді запис `p=&c`; привласнює вказівнику `p` адресу комірки пам'яті, в якій розміщується значення змінної `c`.

3) Операція розіменування (*) – дозволяє отримати значення об'єкту за його адресою.

Операція `*`, застосована до вказівника, видає значення об'єкту, на який даний вказівник посилається. Нехай маємо:

```
int c=10; int *p;
p=&c;
```

тоді запис `cout<<*p`; призведе до виведення на екран значення за вказівником `p` (тобто значення змінної `c`).

Якщо вказівник указує на змінну, то це значення можна змінювати, також використовуючи операцію розіменування (*).

Приклади:

```
int a, b; //змінні типу int
int *pb=&b; //вказівник на змінну b
*pb=10; //привласнити змінній b значення 10 через вказівник
a=*pb+1; //змінній a привласнити значення на одиницю більше значення b
```

Щоб інкрементувати (декрементувати) значення, розміщене за вказівником, необхідно використовувати конструкцію:

```
(*pb)++; (*pb)--
```

Враховуючи наведений вище приклад, змінна `b` збільшиться (зменшиться) на 1. Дужки обов'язкові, оскільки оператор `*` має нижчий пріоритет, ніж оператор `++`

Для ініціалізації вказівників існують наступні способи:

1 За допомогою операції отримання адреси.

```
int a=5;
int *p=&a; //або int p(&a);
```

2 За допомогою іншого вказівника, який проініціалізований.

```
int *r=p;
```

3 Адреса привласнюється в явному вигляді.

```
char *cp=(char*) 0xB8000000;
```

де 0xB8000000 – шістнадцятирічна константа, (char*) – операція приведення типу.

4 Привласнення порожнього значення:

```
int*N=NULL;
```

```
int *R=0;
```

4) Арифметичні операції над вказівниками (++, --, +, -) [1]

З вказівниками можна використовувати тільки чотири арифметичні операції: інкремента (++), декремента (--), додавання (+) та віднімання (-). Почнемо з прикладу. Нехай `int *p1` має значення 2000 (тобто `p1` містить адресу 2000). Після виконання (в 32-розрядному середовищі) виразу

```
p1++;
```

вміст змінної-вказівника `p1` стане рівним 2004, а не 2001. При кожному інкрементуванні вказівник `p1` буде вказувати на *наступне* значення цілого типу тому що `sizeof(int)==4`. Для операції декрементування справедливим буде зворотне твердження, тобто при виконанні виразу

```
p1--;
```

значення `p1` буде зменшуватись на 4. Отже, при інкрементуванні або декрементуванні значення змінної – вказівника буде збільшуватись або зменшуватись на величину, що дорівнює розміру того типу, на який вказує вказівник (базового типу вказівника).

Наприклад,

```
char *pc;
```

```
int *pi;
```

```
float *pf;
```

```
. . . . .
```

```
pc++; //значення збільшиться на 1
```

```
pi++; //значення збільшиться на 4
```

```
pf++; //значення збільшиться на 4
```

Зі значеннями вказівників можна виконувати операцію додавання, але тільки якщо другим операндом є ціле число (тобто до вказівника можна додавати ціле число). Вираз

```
p1=p1+9;
```

примусить p1 вказувати на дев'ятий елемент базового типу вказівника p1 відносно елемента, на який вказував p1 до виконання цієї інструкції.

Додавати вказівники не можна!

Можна відняти один вказівник від іншого, але якщо вони мають один і той же базовий тип. Різниця покаже кількість елементів базового типу, які вміщаються між цими двома вказівниками.

Наприклад:

```
int a=123, b=456, c=789;
int *pi1=&a;
int *pi2=&b;
int *pi3=&c;
printf("\n%x", pi1-pi2);
printf("\n%x", pi1-pi3);
```

На рисунку 2.1 представлено варіант розміщення описаних та ініціалізованих змінних в оперативній пам'яті.

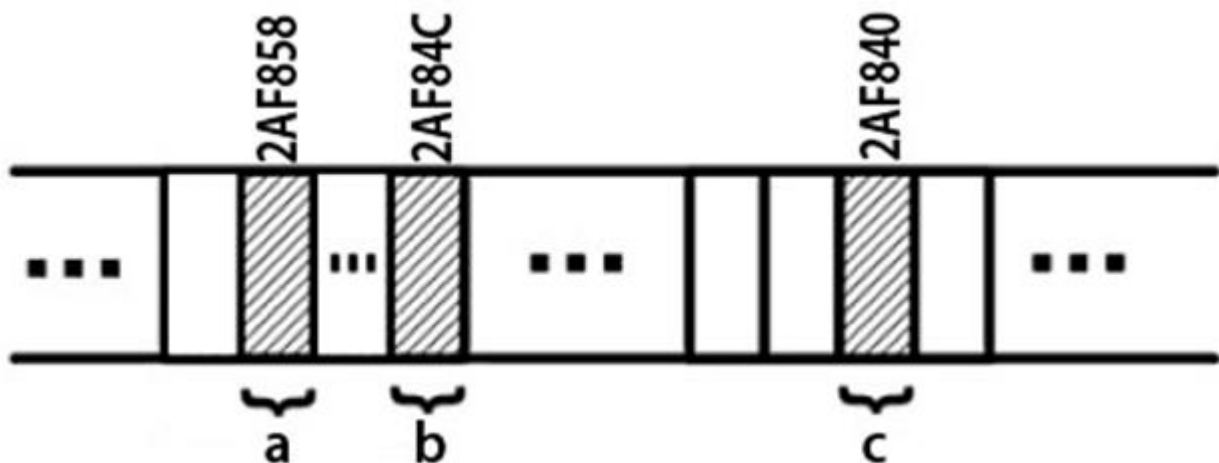


Рисунок 2.1. Представлення змінних для підрахування різниці двох вказівників
Тоді результат роботи фрагменту програми:

3	//2AF85816-2AF84C16=281608810-281607610=12 / 4 (sizeof(int)) = 3
6	//2AF85816-2AF84016=281608810-281606410=24 / 4 (sizeof(int)) = 6

5) Порівняння вказівників[1]

Вказівники можна порівнювати, використовуючи операції відношення. При цьому порівнюються адреси відповідних блоків пам'яті. Однак порівняння вказівників має сенс тільки тоді, коли вказівники будь-яким чином пов'язані між собою. Нехай, наприклад, вказівники вказують на елементи одного й того ж масиву. Тоді результат операцій порівняння вказівників буде залежати від взаємного положення елементів один щодо іншого:

```
int a[5]={7, 1, 3, 5, 2};
int *p1=&a[0], *p2=&a[2];
int x=p1==p2; //хотримає значення 0
int y=p1!=p2; //у отримає значення 1
int z=p1<=p2; //z отримає значення 1
int d=p1>=p2; //d отримає значення 0
```

6) Приведення типів.

На одну і ту ж область пам'яті можуть посилатися вказівники різного типу. Якщо застосувати до них операцію розіменування (*), то вийдуть різні результати.

Наприклад,

```
int a=123;
int *pi=&a;
char *pc=(char*) &a;
float *pf=(float*) &a;
printf("\n%x\t%i", pi, *pi); //fff4 123
printf("\n%x\t%c", pc, *pc); //fff4 {
printf("\n%x\t%f", pf, *pf); //fff4 0.000000
```

Тобто адреса у трьох вказівників одна і та ж, але при розіменуванні виходять різні значення залежно від типу вказівника.

У прикладі при ініціалізації вказівника була використана операція приведення типів – (char*) та (float*). При використанні в одному виразі вказівників різних типів, явне перетворення потрібне для всіх типів, окрім void*. Вказівник може неявно перетворюватися в значення типу bool, при цьому ненульовий вказівник перетвориться в true, а нульовий в false.

Приклад програми для роботи з вказівниками на змінні.

```
#include <stdio.h>
#include <conio.h>
int main()
{
int a;
int *b;
a=134;
b=&a;
printf("\n Значення змінної a = %i", a);
printf("\n Адреса змінної a = %p", &a);
printf("\n Дані за адресою вказівника b= %i", *b);
printf("\n Значення вказівника b = %p", b);
printf("\n Адреса розташування вказівника b=%p", &b);
getch();
return 0;
}
```

Результат роботи програми:

Значення змінної `a = 134`.

Адреса змінної `a = 0012FED4`.

Дані за адресою вказівника `b = 134`.

Значення вказівника `b = 0012FED4`.

Адреса розташування вказівника `b` рівна `0012FEC8`.

3Масиви і вказівники

В мові програмування C++ між вказівниками та масивами існує тісний зв'язок. Коли оголошується масив в вигляді `int mas[10]`, то це означає не тільки виділення пам'яті для десяти цілочисельних елементів масиву. При цьому також виділяється пам'ять під вказівник з іменем `mas`, значення якого дорівнює адресі нульового елемента масиву. З точки зору синтаксису мови C++ вказівник `mas` є константним вказівником, значення якого можна використовувати, але змінити це значення не можна.

Отже, *вказівником на масив* є ім'я даного масиву а також вказівник на його нульовий елемент. Наприклад:

```
int mas[10], *pmas;
```

```
pmas=mas; //або pmas=&mas[0];
```

Зміна вказівника переводить на інший елемент масиву:

```
pmas++; // вказівник переведено на наступний елемент масиву
```

```
pmas--; //вказівник переведено на попередній елемент масиву
```

Операція `&ім'я_масиву` якості результуючого значення повертає адресу нульового елемента масиву, визначеним іменем масиву.

Отже `ім'я_масиву==&ім'я_масиву[0]==&ім'я_масиву`

Оскільки ім'я масиву є константним вказівником, то його неможливо змінити, отже, запис `*(mas++)` буде помилковою, а `*(mas+1)` – ні.

Операція `sizeof (ім'я_масиву)` в якості результуючого значення повертає розмір всього масиву. Тоді:

```
int k=sizeof(mas); //результатом буде 4*10=40 (байтів)
```

```
int n=sizeof(mas)/sizeof(mas[0]); /*кількість елементів масиву*/
```

Для доступу до елементів лінійного масиву ми використовували операцію індексування `ім'я_масиву[індекс]` – це вираз з двома операндами, де `ім'я_масиву` – це вказівник константа, а `індекс` визначає зсув від початку масиву. Використовуючи вказівники, звернення по індексу можна записати таким чином:

```
* (ім'я_масиву+індекс);
```

Графічне представлення масиву в пам'яті ЕОМ представлено на рисунку 3.1, де `data` – адреса початку масиву; `sizeof(data)` – розмір масиву `data` в байтах; `sizeof(float)` — розмір пам'яті під один елемент масиву в байтах; `p1` і `p2` – вказівники для роботи з масивом.

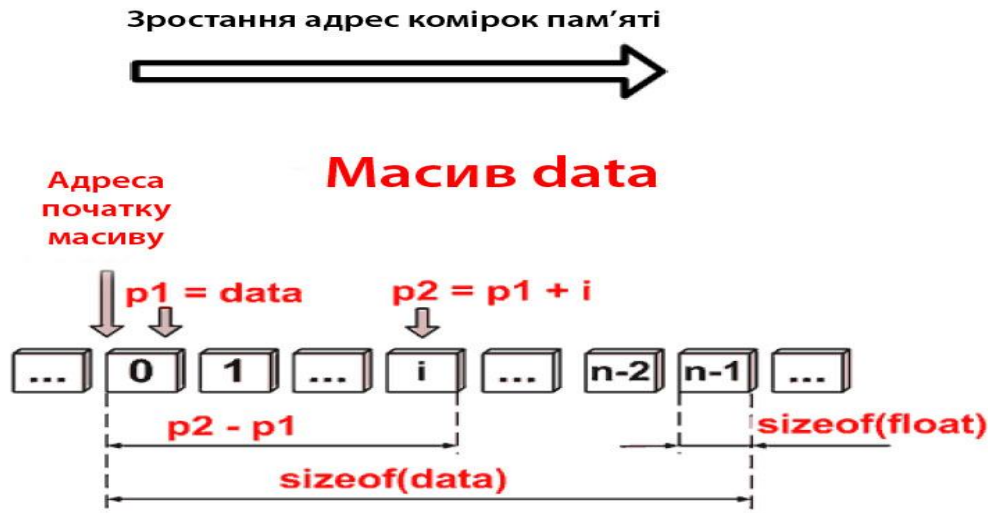


Рисунок 1.1 – Представлення масиву у пам'яті

Приклад звернення до елементів масиву за допомогою вказівників:

```
data[0] == *p1;
data[1] == *(p1 + 1);
data[2] == *(p1 + 2);
data[i] == *(p1 + i);
data[n-1] == *(p1 + n - 1).
```

У наступному прикладі виділяється пам'ять для двох коротких символьних масивів `buf[]` й `data[]`, визначаються два символьних покажчики `p` і `q`. Покажчики ініціалізуються за допомогою адреси початкового елемента масиву. Даний приклад показує, що це можна зробити явно через адресу

(`p = &buf [0];`) або неявно (за допомогою імені масиву (`q = data;`)):

```
char buf[6], data[6], *p, *q; // масиви й покажчики
int i;
p=&buf[0]; // явний синтаксис для адреси першого елемента
q=data; // неявний синтаксис для адреси першого елемента
for (i=0; i<6; i++) // присвоєння компонентів масиву
{
    *(p+i)='A'+i; // букви у верхньому регістрі "ABCDEF"
    *(q+i)='a'+i; // букви в нижньому регістрі
}
```

Єдина різниця між покажчиком й ім'ям масиву в тім, що покажчик можна перепризначити, і він буде посилатися на іншу комірку пам'яті (за допомогою операції

одержання адреси & або присвоювання покажчика), а ім'ямасиву - це константа. Їй не можна привласнити інша адреса. У наступному прикладі перша частина масиву `data[]` (символи в нижньому регістрі) копіюється вдругу частину масиву `buf[]`, після чого він буде містити букви "ABCabc", а не "ABCDEF".

```
p=&buf[3]; // указує на другу половину, масиву
for (i=0; i<3; i++) // заміна перших трьох компонентів
    *(p+i)=*(q+i);
```

Зверніть увагу, що операція розіменнування має більш високий пріоритет, чим арифметичні операції, тому `*p + i` тут використовувати не слід, тому, що це буде означати `p[0]+i` а не `p[i]`.

Двомірний масив (матрицю) `float b[4][6]` можна розглядати як одномірний масив із 4-х елементів (рядків), в якому кожен елемент (рядок) є одномірним масивом із 6-ти елементів. Тоді ім'я масиву `b` є константним вказівником, значення якого дорівнює адресі масиву (або адресі його елементу `b[0][0]`). Тобто `b==&b==&b[0][0]`. Оскільки перший індекс матриці означає номер рядка, то вираз `b+i` означає зсув від початку масиву на `i`-тий рядок, а вираз `*(b+i)` має значення адреси початку `i`-того рядка. Для того, щоб отримати доступ до `j`-того елементу цього рядка (тобто до елемента матриці, розташованого в `i`-тому рядку та `j`-тому стовбці) необхідно змістити адресу `*(b+i)` на значення `j`: `*(b+i)+j`. Отже, значення відповідного елементу масиву `b` можна отримати за допомогою виразу: `*(*(b+i)+j)`.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дайте визначення поняття "вказівник".
- 2 Яку інформацію зберігає вказівник?
- 3 Який обсяг пам'яті займає змінна-вказівник
- 4 Для чого найчастіше використовується вказівники?
- 5 На які категорії (групи) діляться вказівники?
- 6 Наведіть загальну форму оголошення вказівника.
- 7 Для чого призначення операція "&" по відношенню до вказівника?
- 8 Для чого призначення операція "*" по відношенню до вказівника?
- 9 Які існують способи ініціалізації вказівників?
- 10 Яку операцію необхідно використовувати для отримання значення об'єкта на який посилається вказівник?
- 11 Яку операцію необхідно використовувати для отримання адреси об'єкту для ініціалізації вказівника?

- 12 Для чого призначена операція розіменування вказівника?
- 13 При використанні операції явного приведення типів при визначенні адреси змінної одного типу чи буде результат розіменування вказівника однаковий?
- 14 На яке значення збільшується значення вказівника при використанні операції інкремент?
- 15 На яке значення зменшується значення вказівника при використанні операції декремент?
- 16 Що являє собою різниця двох вказівників на елементи одного масиву?
- 17 Чи допускається складання вказівників?
- 18 Чи вірне твердження `ім'я_масиву=&ім'я_масиву[0]=&ім'я_масиву`?
- 19 Чи вірне твердження `ім'я_масиву=&ім'я_масиву[1]`?
- 20 Чи вірне твердження `*(a++)`, якщо `a` – ім'я масиву?
- 21 Чи вірне твердження `*(a+1)`, якщо `a` – ім'я масиву?
- 22 Яка загальна форма звернення до елементу лінійного масиву за допомогою вказівника?
- 23 Яка загальна форма звернення до елементу матриці за допомогою вказівника?
- 24 Що ми отримаємо в результаті інструкції `*(a+3)`, якщо `a` – ім'я лінійного масиву розміром 10 елементів?
- 25 Що ми отримаємо в результаті інструкції `*a+3`, якщо `a` – ім'я лінійного масиву розміром 10 елементів?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Як оголосити покажчик з ім'ям `valPtr`, що має базовий тип `long int`?
- 2 На яку величину зростає значення покажчика на тип `double` при його інкрементуванні, якщо припустити, що тип `double` займає 8 байт?
- 3 Опишіть покажчик на цілий тип і проініціалізуйте його адресою змінної `a`.
- 4 Нехай:

```
int x, y;  
int *px, *py;
```

 - Як привласнити змінним `x` та `y` значення через вказівник?
 - Як змінній `x` привласнити через вказівник значення, на одиницю більше значення `y`?
- 5 Нехай:

```
int b[5]={5, 1, 3, 5, 2};  
int *p1=&b[0], *p2=&b[3];  
int x, y, z, d;
```

Яке значення отримають:

- змінна `x` після виконання інструкції `p1==p2;`
- змінна `u` після виконання інструкції `p1!=p2;`
- змінна `z` після виконання інструкції `*p1<=*p2;`
- змінна `d` після виконання інструкції `p1>=p2;`
- змінна `d` після виконання інструкції `*p1>*p2;`

6 Написати мовою програмування C++ програми обробки масивів з використанням вказівників.

5 Дано $B(4)$ – масив дійсних чисел. Знайти і роздрукувати мінімальний елемент і середнє арифметичне елементів масиву.

6 Дана матриця $A(4, 4)$. Знайти добуток та суму ненульових елементів спочатку головної діагоналі, а потім – останнього стовпчика.

7 Дана матриця $T(4,4)$. Знайти мінімальний елемент серед максимальних елементів стовпців.

8 Дані дві матриці: $A(5, 5)$ та $B(5, 5)$. Підрахувати добуток елементів головної діагоналі кожної матриці. Роздрукувати ту матрицю, у якої добуток більший.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. C++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування C. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція №№ 11-12

з навчальної дисципліни Програмування

Тема Основи створення та використання функцій в C++. Обмін даними між функціями.

Передача параметрів за значенням та за посиланням.

Мета заняття Пояснити правила визначення функції в C++ програмах, поняття формальних та фактичних параметрів, виклик функції, повернення функцією значення, опис прототипу функції.
Пояснити принципи передачі параметрів з значенням та за посиланням. Розглянути приклади передачі масивів даних в якості параметрів.

Час – 4 години

План проведення лекції

- 1 Визначення функції в C++ програмах.
- 2 Прототип функцій.
- 3 Виклик функцій. Повернення значення в точку виклику функції.
- 4 Правила дії областей видимості функцій.
- 5 Способи передачі даних у функцію та повернення значень.
- 6 Передача масиву як параметра функції.

Навчальні матеріали лекції

1 Визначення функції в C++ програмах

Із збільшенням об'єму програми стає неможливо утримувати в пам'яті всі деталі. Щоб зменшити складність програми, її розбивають на частини. У C++ задача може бути розділена на простіші під задачі за допомогою функцій. Розрізняють системні (у складі систем програмування) і власні функції.

Розбиття програм на функції дає наступні переваги:

- функцію можна викликати з різних місць програми, що дозволяє уникнути повторного програмування;
- одну і ту ж функцію можна використовувати в різних програмах;
- функції підвищують рівень модульності програми і полегшують її проектування;
- використання функцій полегшує читання і розуміння програми, прискорює пошук і виправлення помилок.

Функція – це іменована послідовність описів і операторів, що виконує закінчену дію, наприклад, пошук максимального значення серед двох чисел, пошук модулю числа тощо.

Функція, по-перше, є одним з похідних типів C++, а, по-друге, мінімальним виконуваним модулем програми.

Будь-яка функція повинна бути оголошена і визначена. Оголошення функції (прототип) задає ім'я функції, тип значення, яке повертається, і список параметрів, які передаються. Визначення функції містить, окрім прототипу (оголошення), тіло функції, яке є послідовністю описів і операторів.

Загальний вигляд визначення функції:

```
тип_значення ім'я_функції([список_формальних_параметрів])
{
    тіло_функції;
    return [вираз];
}
```

Приклад визначення функції знаходження суми двох чисел.

Функція з ім'ям `sum()`, в яку передаються 2 параметри – дійсні змінні `x`, `y`. Функція повертає дійсне значення */

```
float sum(float x, float y)
{
    float z; //оголошення дійсної змінної z
    // знаходження суми змінних x та y та запис результату у змінну z //
    z=x+y;
    return z; // повернення з функції значення змінної z
}
```

Тип значення, яке повертається, може бути будь-яким, окрім масиву.

Список формальних параметрів – це ті величини, які потрібно передати у функцію. Елементи списку розділяються комами. Для кожного параметра вказується тип і ім'я. У оголошенні імена можна не вказувати.

Тіло функції – це блок або складений оператор. У середині функції **не можна визначити іншу функцію**.

У тілі функції може бути оператор, який повертає значення функції в точку її виклику. Він може мати 2 форми:

- `return вираз;`
- `return.`

Перша форма використовується для повернення результату, тому "вираз" повинен мати той же тип, що і тип функції у визначенні.

Друга форма використовується, якщо функція не повертає значення, тобто має тип `void`.

Для того, щоб виконувалися оператори, записані в тілі функції, функцію необхідно викликати. При виклику указуються: ім'я функції і фактичні параметри, які будуть передані функції.

Фактичні параметри замінюють формальні параметри при виконанні операторів тіла функції. Фактичні і формальні параметри повинні співпадати по кількості і типу.

Оголошення функції повинне знаходитися в тексті раніше виклику функції, щоб компілятор міг здійснити перевірку правильності виклику. Якщо функція має тип не `void`, то її виклик може бути операндом виразу.

У лістингу 1.1 наведено приклад програми обчислення факторіалу введеного числа.

Лістинг1.1.

```
#include <iostream.h>
#include <conio.h>
/* визначення функції з ім'ям factorial(), якій передається одне ціле
значення - змінна "i". Функція повертає дійсне значення. */
float factorial(int i)
{
    int j;
    float k;
    k=1;
    for (j=2; j<i+1; j++) k=k*j;
// повернення в програму обчисленого значення
return k;
}
// головна функція програми
int main()
{
    clrscr();
    int i;
    cout<<endl<<"Введіть ціле число: ";
    cin>>i;
// виклик функції factorial() з фактичним параметром i
cout<<endl<<i<<"!="<<factorial(i);
getch();
return 0;
}
```

Блок – схема алгоритму рішення задачі наведена на рисунку 1.1.

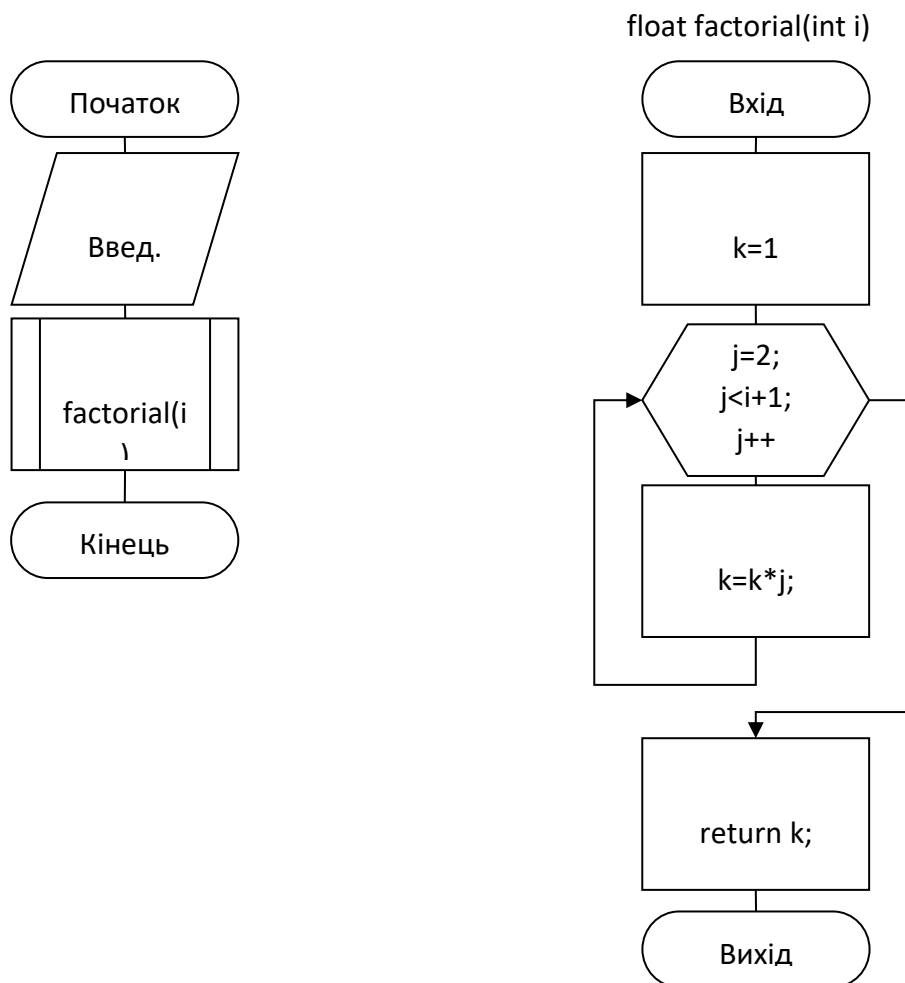


Рисунок 1.1. Блок – схема алгоритму пошуку факторіалу цілого числа.

Результат роботи програми:

Введіть ціле число: 7
7.000000!=5040.000000

2 Прототип функцій

У практиці програмування бувають випадки, коли тіло функції розташовується в програмі нижче за місцем, де виконується її виклик, або функція взагалі компілюється окремо. В цьому випадку до використання функції повинен бути вказаний прототип функції (або оголошення функції), що містить:

- тип функції;
- ім'я функції;
- інформацію про параметри.

Прототип необхідний для того, щоб компілятор зміг здійснити перевірку відповідності типів і кількості фактичних параметрів, які передаються, типам і кількості формальних параметрів. Оголошення функції має той же вигляд, що і визначення

функції, проте тіло функції відсутнє, і імена формальних параметрів також можуть бути відсутніми.

Загальний вигляд визначення прототипу функції.

```
тип_значення ім'я_функції([список_формальних_параметрів]);
```

Приклад визначення прототипу (оголошення) функції модуля числа:

```
int abs(int i);
```

При програмуванні на мові C/C++ широко використовуються бібліотечні функції. Ці функції були заздалегідь розроблені і записані до складу системи програмування. Прототипи бібліотечних функцій знаходяться в спеціальних заголовних файлах з розширенням *.h, які необхідно підключати за допомогою директиви #include.

Не лістинг 2.1 наведено приклад програми генерації таблиці чисел 2^n , де n – числа від 1 до 10.

Лістинг 2.1.

```
#include <iostream.h>
#include <conio.h>
```

```
/* Опис прототипу функції power() з 2-ма цілими параметрами - змінні
base, index. Функція повертає ціле значення. */
```

```
int power(int base, int index);
```

```
// головна функція програми
```

```
int main()
```

```
{
```

```
clrscr();
```

```
int i;
```

```
for (i=0; i <=10; i++ )
```

```
{
```

```
cout<<power(2,i)<<"    ";
```

```
}
```

```
getch();
```

```
return 0;
```

```
}
```

```
// Безпосередньо функція power()
```

```
int power(int base, int index)
```

```
{
```

```
    int i, p;
```

```
    p = 1;
```

```
    for (i = 0; i <= index; i++)
```

```
    {
```

```
        p = p*base;
```

```
    }
```

```
    return p;
```


}

Блок – схема алгоритму рішення задачі наведена на рисунку 2.1.

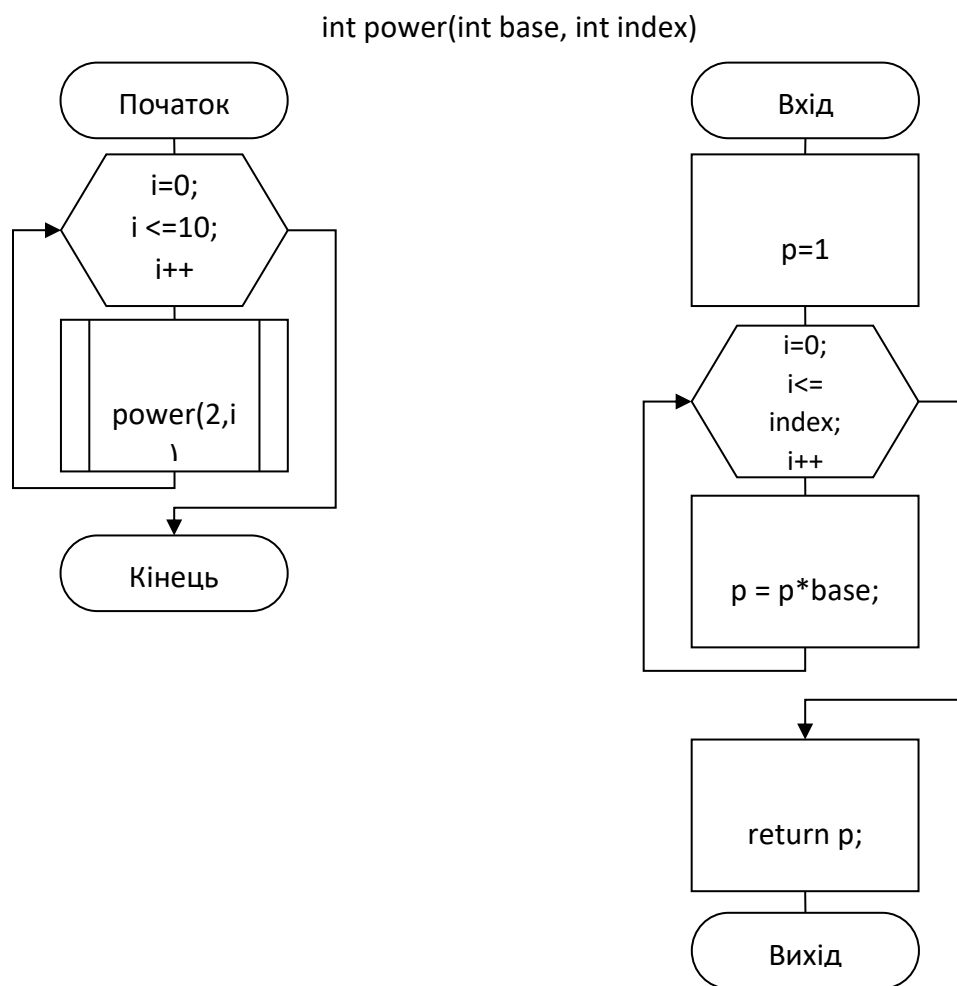


Рисунок 2.1. Блок – схема алгоритму обчислення цілого ступеня числа 2

Результат роботи програми:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048

3 Виклик функцій. Повернення значення в точку виклику функції

Загальний вигляд виклику функції:

```
ім'я_функції([список_фактичних_параметрів]);
```

Фактичний параметр – це величина, яка привласнюється формальному параметру при виклику функції. Таким чином, формальний параметр – це змінна у функції, що викликається, а фактичний параметр – це конкретне значення, привласнене цій змінній. Фактичний параметр може бути константою, змінною або виразом. Фактичний

параметр спочатку обчислюється, а потім його значення передається у функцію. Якщо у функцію потрібно передати декілька параметрів, то вони записуються через кому.

Функція може передати в точку виклику програми тільки одне значення, для цього використовується оператор `return`.

```
return [вираз];
```

Працює оператор так. Обчислюється значення виразу, вказаного після `return` і перетворюється до типу значення, що повертається функцією. Виконання функції завершується, а обчислене значення передається в функцію, яка її викликала. Будь-які оператори, наступні у функції за оператором `return`, ігноруються. Програма продовжує свою роботу з оператора наступного за оператором виклику даної функції.

Якщо функція має тип `void`, тобто не повертає в точку виклику ніякого значення, в операторі `return [вираз]` можна опустити, якщо повернення відбувається перед фігурною дужкою, що закривається, і у функції `main`.

Приклад використання `return` в середині функції.

```
void function()
{
    ...
    return;
    ...
}
```

У прикладі оператор `return` завершує виконання функції `function()` і передає управління в функцію, яка здійснила виклик. Ніяке значення при цьому не передається.

Також функція може містити декілька операторів `return`, якщо це визначено потребами алгоритму. Наприклад, в наступній програмі функція `f` порівнює значення змінної з нулем. Якщо число додатне, то в програму передається значення рівне 1, якщо від'ємне, то -1, а якщо нульове, то 0.

```
#include <iostream>
int f(int a)
{
    if (a>0) return 1;
    else if (a<0) return -1;
    else return 0;
}
//функцію можна переписати так
//int f(int a)
//{ int r;
//if (a>0) r=1;
//else if (a<0) r=-1;
//else r=0;
```

```

//return(r);
//}
int main ()
{
int b;
cout<<"b=";
cin>>b;
if (f(b)==1) cout<<"positive value \n";
else if (f(b)==-1) cout<<"negative value \n";
else cout<<"zero value \n";
system ("pause");
} //у функції main відсутній оператор return

```

Виклик функції залежить від того, повертає вона значення чи ні. Якщо функція повертає значення, то її виклик проводиться з математичних і логічних виразів.

У лістингу 3.1 наведено приклад функції визначення максимуму серед двох чисел.

Лістинг 3.1.

```

#include <iostream.h>
#include <conio.h>

int max(int a, int b)
{
if (a>b)
return a;
else
return b;
}

int main()
{
clrscr();
int x, y;
cout<<endl<<"Введіть 2 числа: ";
cin>>x>>y;
cout<<endl<<"Максимум з них = "<<max(x,y);
getch();
return 0;
}

```

Блок – схема алгоритму рішення задачі наведена на рисунку 3.1.

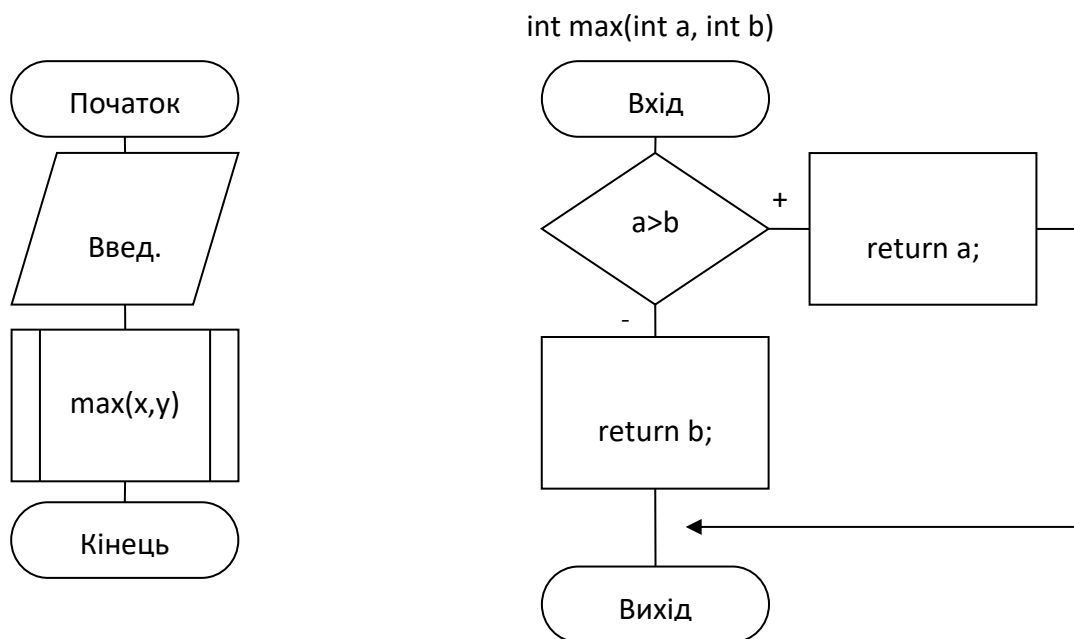


Рисунок 3.1. Блок – схема алгоритму пошуку максимуму двох чисел

Результат роботи програми:

Введіть 2 числа: 10 7
Максимум з них = 10

Функції можуть і не повертати значення, а просто виконувати деякі обчислення або вивід на екран. У такому разі вказується порожній тип даних `void`, і оператор `return` не використовується.

На лістингу 3.2 наведено приклад програми, в якій функція `star(int a)` не повертає значення в точку виклику, тому що призначення цієї функції - виведення на екран заданої користувачем кількості «зірочок».

Лістинг 3.2.

```
#include <iostream.h>
#include <conio.h>

void star (int a);

int main()
{
    clrscr();
    int k;
    cout<<endl<<"Скільки символів надрукувати?";
    cin>>k;
    star(k);
    getch();
    return 0;
}
```

```

}

void star (int a)
{
    int i;
    for(i=1; i<=a;i++)
    {
        cout<<"*";
    }
}

```

Блок – схема алгоритму рішення задачі наведена на рисунку 3.2.

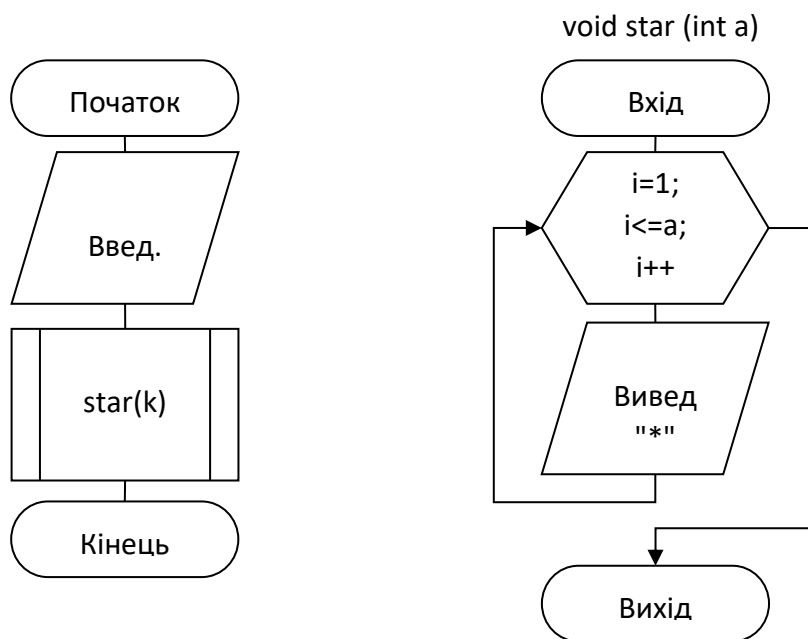


Рисунок 3.2. Блок – схема програми виведення заданої кількості символу «*»

Функції використовують пам'ять зі стека. Деяка область стека приділяється функції й залишається пов'язаною з нею до закінчення її роботи, по завершенню якої відведена їй пам'ять звільняється й може бути зайнята іншою функцією.

Всі величини, описані усередині функції, а також її параметри, є локальними. Областю їхньої дії є функція. При виклику функції, як і при вході в будь-який блок, у стеці виділяється пам'ять під локальні автоматичні змінні. При виході з функції відповідна ділянка стека звільнюється, тому значення локальних змінних між викликами однієї й тієї ж функції не зберігаються. Якщо цього потрібно уникнути, при оголошенні локальних змінних використовується модифікатор `static`:

```

#include <iostream.h>
void f(int a)
{
    int m=0;

```

```

    cout<<"n m p\n";
    while (a--)
    {
        static int n=0;
        int p=0;
        cout<<n++<<' ' <<m++<<' ' <<p++<<'\n';
    }
}

int main()
{
    f(3); f(2);
    return 0;
}

```

Статична змінна *n* розміщується в сегменті даних й ініціалізується один раз при першому виконанні оператора, що містить її визначення. Автоматична змінна *m* ініціалізується при кожному вході у функцію. Автоматична змінна *p* ініціалізується при кожному вході в блок циклу. Програма виведе на екран:

```

n m p
0 0 0
1 1 0
2 2 0
n m p
3 0 0
4 1 0

```

4 Правила дії областей видимості функцій

Правила дії областей видимості будь-якої мови програмування - це правила, які дозволяють управляти доступом до об'єкта з різних частин програми [1]. Інакше кажучи, правила дії областей видимості визначають, який код має доступ до тієї або іншої змінної. Ці правила також визначають тривалість "життя" змінної В С++ визначено дві основні області видимості: *локальна* й *глобальна*. У кожній з них можна оголошувати змінні. Розглянемо, чим змінні, оголошені в локальній області, відрізняються від змінних, оголошених у глобальній області.

Локальна область видимості.

Локальна область видимості створюється блоком. (Згадайте, що блок починається з відкриваючої фігурної дужки й завершується закриваючою). Таким чином, щораз при створенні нового блоку програміст створює нову область видимості. Змінна, оголошена усередині будь-якого блоку, називається *локальною* стосовно цього блоку.

Локальну змінну можуть використати лише інструкції, включені в блок, у якому ця змінна оголошена. Інакше кажучи, локальна змінна невідома за межами власного блоку. Важливо розуміти, що локальні змінні існують тільки під час виконання програмного блоку, у якому вони оголошені. Це означає, що локальна змінна створюється при вході у свій блок і руйнується при виході з нього. А оскільки локальна змінна руйнується при виході з "свого" блоку, її значення губиться.

Найпоширенішим програмним блоком є функція. У С++ кожна функція визначає блок програми, що починається з відкриваючої фігурної дужки цієї функції й завершується її закриваючою фігурною дужкою. Код функції - це її "приватна власність", і до неї не може одержати доступ жодна інструкція з будь-якої іншої функції, за винятком інструкції її виклику. Інакше кажучи, код і дані, певні в одній функції, не можуть взаємодіяти з кодом і даними, певними в іншій, оскільки дві функції мають різні області видимості.

Розглянемо, наприклад, програму [1].

```
#include <iostream.h>
void f1();
int main ()
{
    int val=10; // змінна val локальна стосовно функції main()
    cout<<"Значение val в функции main():"<<val<<endl;
    f1();
    cout<<"Значение val в функции main():"<<val<<endl;
    return 0;
}
void f1()
{
    int val=88; // ця змінна val локальна стосовно функції f1()
    cout<<"Значение val в функции f1():"<<val<<endl;
}
```

Результат виконання цієї програми:

Значение val в функции main(): 10
Значение val в функции f1(): 88
Значение val в функции main(): 10

Цілочисельна змінна `val` оголошується в цій програмі двічі: перший раз в функції `main()`, другий раз – в функції `f1()`. При цьому змінна `val`, оголошена в функції `main()`, не має ніякого відношення до однойменної змінної із функції `f1()`.

Оскільки локальні змінні створюються з кожним входом і руйнуються з кожним виходом із програмного блоку, в якому вони оголошені, вони не зберігають своїх значень між активаціями блоків. Це особливо важливо пам'ятати до відношення функцій. При виклику функції її локальні змінні (в тому числі і параметри) створюються, а при виході із неї – руйнуються. Це означає, що локальні змінні не зберігають своїх значень між викликами функцій.

Глобальна область видимості.

Це декларативна область, яка заповнює простір поза всякими функціями [1]. При оголошенні змінної в глобальній області видимості створюється *глобальна змінна*.

Глобальні змінні відомі на протязі всієї програми, їх можна використовувати в будь-якому місці програми, і вони зберігають свої значення під час виконання всієї програми. Отже, доступ до цих змінних можна отримати із будь-якого виразу, незалежно від функції, в якій цей вираз знаходиться.

У мові програмування C++ глобальні (зовнішні) змінні оголошуються поза якою-небудь функцією. З їх допомогою зручно організовувати обмін даними між функціями, проте це вважається поганим тоном, оскільки легко заплутує програму. Локальні змінні в мові програмування C++ називають автоматичними. Область дії автоматичних змінних розповсюджується тільки на ту функцію, в якій вони були оголошені. Параметри функції також є локальними змінними.

Структурна організація файлу програми на мові C++, що містить декілька функцій, може виглядати трохи по-різному. Оскільки виконання починається з `main()`, то їй повинні бути відомі специфікації (імена, кількість і тип параметрів, тип значення, що буде повернуто) всіх функцій, які з неї викликаються. Звідси витікає, що оголошуватися функції повинні до того, як будуть викликані. А ось визначення функції вже може слідувати і до, і після `main()`.

Способи передачі даних у функцію, що викликається:

- ☐ шляхом передачі аргументів функції;
- ☐ з використанням глобальних змінних;
- ☐ через файли на зовнішніх пристроях, що запам'ятовують;

Способи передачі даних з функції, що викликається:

- ☐ через значення, що повертається;
- ☐ через формальні параметри, що викликаються по посиланню;
- ☐ шляхом зміни значень глобальних змінних;
- ☐ через файли на зовнішніх пристроях, що запам'ятовують;

Крім того, функція може виконувати яку-небудь дію, що не вимагає передачі з неї даних в функцію, яка її викликала.

5 Способи передачі даних у функцію та повернення значень

Існує два способи передачі параметрів у функцію: за значенням та за посиланням (за адресою).

Передача параметрів за значенням.

Розглянемо спочатку приклад програми, в якій використовується передача параметрів за значенням.

Наступна програма (див. Лістинг 2.1) передає два параметри з іменами `big` і `small` у функцію `display_values`. Функція `display_values`, у свою чергу, привласнює обом параметрам число 1001 і потім виводить значення кожного параметра. Коли функція завершується, програма поновлюється і виводить значення цих же параметрів:

Лістинг 2.1.

```
#include <iostream.h>

void display_values(int a, int b)
{
    a = 1001;
    b = 1001;
    cout << "Значення у функції display_values рівні " << a << " і "
<< b << endl;
}

void main(void)

{
    int big = 2002, small = 0;
    cout << "Значення до функції " << big << " і " << small << endl;
    display_values(big, small);
    cout << "Значення після функції " << big << " і " << small << endl;
}
```

Коли буде відкомпільована і запущена ця програма, на екрані з'явиться наступний вивід:

Значення до функції 2002 і 0

Значення у функції `display_values` рівні 1001 і 1001

Як видно, значення параметрів `a` та `b` у функції `display_values` були змінені на 1001. Проте після завершення функції значення змінних `big` і `small` в `main` залишилися колишніми. Щоб зрозуміти, чому зміна параметрів не вплинула на змінні `big` і `small` в `main`, необхідно зрозуміти, як C++ передає параметри у функції.

Коли передаються параметри у функцію, C++ розміщує копії значень параметрів в тимчасовій ділянці пам'яті, званій стеком. Будь-які зміни, що виконуються функцією над параметрами, виявляються тільки усередині стека. Коли функція завершується, C++ скидає вміст стека разом з будь-якими змінами, які функція провела в параметрах. Оскільки функція не знає адресу пам'яті параметра, вона не може змінити його значення.

Отже, при передачі за значенням у стек заносяться копії фактичних параметрів, і оператори функції працюють із цими копіями. Доступу до вихідних значень параметрів у функції немає, а, отже, немає й можливості їх змінити.

Передача параметрів за посиланням.

Щоб повідомити функції адресу параметра, програми повинні використовувати оператор адреси C++ (&). Наступний виклик функції ілюструє, як програма використовуватиме оператор адреси для передачі адрес змінних `big` і `small` у функцію `change_values`:

```
change_values (&big, &small); //Передача параметрів за адресою
```

Усередині функції необхідно повідомити C++, що програма передаватиме параметри за допомогою адреси. Для цього оголошуються змінні-вказівники, що позначаються "*" перед іменем кожної змінної, як показано нижче:

```
void change_values (int *big, int *small) //Вказівник на тип int
```

Змінна-вказівник є змінною, яка містить адресу пам'яті. Усередині функції необхідно повідомити C++, що функція працює з адресою параметра. Для цього ви передуете імені параметра "*", як показано нижче (тобто розіменовуєте змінну-вказівник):

```
*big = 1001;
```

```
*small = 1001;
```

Наступна програма (див. Лістинг 2.2) використовує оператор адреси для передачі адрес параметрів `big` і `small` у функцію `change_values`. Функція, у свою чергу, використовує вказівники ділянок пам'яті параметрів. Отже, зміни параметрів, зроблені функцією, залишаються і після завершення функції:

Лістинг 2.2.

```
#include <iostream.h>
```

```

void change_values (int *a, int *b)
{
    *a = 1001;
    *b = 1001;
    cout << "Значення у функції display_values" << " рівні " << *a <<
" i " << *b << endl;
}

void main(void)
{
    int big = 2002, small = 0;
    cout << "Значення перед функцією " << big << " i " << small <<
endl;
    change_values(&big &small);
    cout << "Значення після функції " << big << " i " << small << endl;
}

```

Коли буде відкомпільовано і запущено цю програму, на екрані з'явиться наступний вивід:

Значення перед функцією 2002 і 0

Значення у функції change_values рівні 1001 і 1001

Значення після функції 1001 і 1001

Як видно, значення, які функція `change_values` привласнює параметрам `big` і `small` нові значення, які залишаються і після завершення функції. Щоб зрозуміти, чому зміни, які функція виконала над змінними, залишилися після її завершення, необхідно пригадати, що функція має доступ до елементу пам'яті кожної змінної. Якщо передаються параметри за адресою, C++ поміщає адресу кожної змінної в стек. Використовуючи вказівник (адреси пам'яті) усередині функції, `change_values` може звернутися до пам'яті за адресою кожного параметра, змінюючи значення параметрів, що і потрібне.

6 Передача масиву як параметра функції

При використанні масиву в якості параметра, у функцію передається покажчик на його перший елемент, іншими словами, масив завжди передається за адресою (тобто по

посиланню). В якості прикладу розглянемо програму, в якій використовується функція `sum` обчислення суми елементів лінійного масиву цілих чисел (див. Лістинг 3.1).

Лістинг 3.1.

```
#include <iostream.h>
int sum( int* mas,  int n); //Прототип функції
int const n=10;
int main()
{
    int marks[n]={5, 4, 5, 4, 4};
    cout << "Сумма элементов массива: "<< sum(marks, n); /*Виклик функції
*/
    return 0;
}
// Визначення функції
int sum ( int* mas,  int n)
{
    int s = 0;
    for (int i = 0; i < n; i++) s +=mas[i];
    return s;
}
```

Як бачимо, перший параметр функції `sum` є вказівником на тип елементу масиву `int* mas`. Як ми знаємо, ім'я масиву містить адресу цього масиву, а тому при виклику функції `sum(marks, n)`; немає необхідності використовувати операцію адреси (&). При цьому внутрішній код функції працює з реальним вмістом цього масиву і, тому, в тілі функції при звертанні до елементу масиву немає необхідності використовувати операцію розіменування (*).

При передачі масиву по посиланню інформація про кількість елементів губиться, і варто передавати його розмірність через окремий параметр. В нашому прикладі – це другий параметр `int n` функції `sum`.

У випадку масиву символів, тобто рядка, також використовується передача параметра – рядка по посиланню, але при цьому фактичну довжину рядка можна визначити по положенню нуль-символу і передавати її як параметр не потрібно. Проаналізуємо, наприклад, функцію `print_upper()`, яка друкує свій строковий аргумент на верхньому регістрі (див. Лістинг 3.2):

Лістинг 3.2.

```
#include <iostream.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

```

void print_upper(char *str); // Прототип функції

int main(void)
{
    char s[80];
    cout<<"Введіть рядок символів: ";
    gets(s);
    print_upper(s); //Виклик функції
    cout<<endl<<"s тепер на верхньому регістрі: "<<s;
    return 0;
}
/* Опис функції */
/* Друкувати рядок на верхньому регістрі. */
void print_upper(char *str)
{
    register int t;
    for(t=0; t<strlen(str); t++) {
        str[t]= toupper(str[t]);
        putchar(str[t]);
    }
}

```

Ось що буде виведене у разі фрази "This is a test." (це тест):

Введіть рядок символів: This is a test.
 THIS IS A TEST.
 s тепер у верхньому регістрі: THIS IS A TEST.

При передачі в якості параметра двовимірного масиву (матриці) теж використовується передача по посиланню. Але треба враховувати, що ім'я матриці – це вказівник на вказівник і, тому форма опису параметру – матриці має вид:

Тип_елемента ** ім'я_масиву

Розглянемо програму обробки динамічної матриці (див. Лістинг 3.3). Ця програма заміняє всі від'ємні елементи введеної матриці на додатні та виводить оновлену матрицю на екран. В програмі використовуються три функції:

void in_put(int** mas, int n, int m) – для введення елементів матриці;

void chang_ma(int** mas, int n, int m) – для заміни від'ємних елементів на додатні;

void out_put(int** mas, int n, int m) – для виведення матриці;

Лістинг 3.3.

```
#include<iostream.h>
#include <conio.h>
#include <stdio.h>
// Прототипи функцій
void in_put(int** mas, int n, int m);
void out_put(int** mas, int n, int m);
void chang_ma(int** mas, int n, int m);
int main()
{
    int str, stb;
    cout<<endl<<"Введіть кількість строк ";
    cin>>str;
    cout<<endl<<"Введіть кількість стовбців ";
    cin>>stb;
    int i,j;
    // Формування динамічної матриці
    int **a=new int*[str];
    for (i=0;i<str;i++) a[i]=new int[stb];
    // Виклик функції введення матриці
    in_put(a, str, stb);
    // Виклик функції оновлення матриці
    chang_ma(a, str, stb);
    // Виклик функції виведення матриці
    out_put(a, str, stb);
    getch();
    return 0;
}
// Опис функції введення елементів матриці
void in_put(int** mas, int n, int m)
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        {
            cout<<"\n Введіть M["<<i+1<<","<<j+1<<"]: ";
            cin>>mas[i][j];
        }
}
// Опис функції виведення матриці
void out_put(int** mas, int n, int m)
{
    int i,j;
    for(i=0;i<n;i++)
    {
```

```

        for(j=0;j<m;j++)
        {
            cout.width(3); /*встановлення ширини виводу в 3 позиції*/
            cout<<mas[i][j];
        }
        cout<<endl;
    }
}
// Опис функції заміни від'ємних елементів на додатні
void chang_ma (int** mas, int n, int m)
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        {
            if (mas[i][j]<0) mas[i][j]=mas[i][j]*-1;
        }
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дайте визначення поняття "функція"?
- 2 Дайте визначення поняття "прототип (оголошення) функції"?
- 3 Дайте визначення поняття "виклик функції"?
- 4 Дайте визначення поняття "визначення функції"?
- 5 Дайте визначення поняття "формальний параметр"?
- 6 Дайте визначення поняття "фактичний параметр"?
- 7 Яким чином повертаються та передаються значення у функцію?
- 8 Яка роль оператора return при поверненні значень функцією?
- 9 Які можуть бути типи даних, що повертаються функцією?
- 10 Навіщо використовувати прототип функції до моменту її використання?
- 11 Що таке формальні та фактичні параметри?
- 12 Яка область видимості локальних та глобальних змінних?
- 13 Пояснити поняття передачі параметрів за значенням.
- 14 Пояснити поняття передачі параметрів по посиланню.
- 15 Яка область видимості локальних та глобальних змінних?

16 Як компілятор «відрізняє» параметр, що передається по значенню та параметр, що передається по посиланню?

17 Чим відрізняється передача параметрів за значенням та за посиланням?

18 Коли виникає необхідність передачі параметрів по посиланню?

19 При виклику функції яку операцію слід застосовувати до фактичних параметрів, що передаються по посиланню? Коли без цієї операції можна обійтись?

20 В тілі функції яку операцію слід застосовувати до фактичних параметрів, що передаються по посиланню? Коли без цієї операції можна обійтись?

21 Яка особливість передачі параметру – масиву?

22 В чому відмінність передачі як параметра лінійного масиву від матриці?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Скласти програму, в якій для обчислення значення функції $Y(X)=\sqrt{1-x}$ організована відповідна функція. Організувати обчислення значень функції для різних значень аргументу, що вводяться користувачем і циклі, поки користувач не припинить цей процес.

2 Скласти програму, яка містить дві, обумовлені користувачем функції, та виводить на екран наступний текст:

Three blind mice

Three blind mice

See how they run

See how they run

Одна функція, що викликається двічі, виводить на екран перші два рядки, інша функція, яка також викликається двічі, виводить на екран наступні рядки.

3 Написати функцію, що виводить на екран послідовність цілих чисел, які належать заданому користувачем діапазону дійсних чисел $[a, b]$.

4 Описати функцію SortDec, яка міняє значення дійсних чисел a, b, c таким чином, щоб вони виявилися впорядкованими за зростанням.

5 Описати функцію SK, яка виконує праве циклічне зрушення дійсних чисел a, b, c : значення a переходить в b , значення b – в c , значення c – в a .

6 Описати функцію, що виконує згладжування масиву m дійсних чисел із k елементів наступним чином: кожен елемент $m[k]$ масиву (окрім першого) замінюється на середнє арифметична попередніх елементів масиву.

7 Описати функцію, що змінює порядок елементів масиву mas дійсних чисел із n елементів наступним чином: найменший елемент масиву розміщує на першому місці, найменший з тих, що залишились – на останньому, наступний по величині розміщує на другому місці, наступний – на передостанньому і т.д. В результаті графік елементів масиву повинен нагадувати пагорб.

8 Для заданих користувачем значень a та k у написати функцію для

обчислення значення $z = \begin{cases} k + 2, a < \frac{7}{5} \\ 3, a = \frac{7}{5} \\ \sqrt[4]{\ln(a^2) - k}, a > \frac{7}{5} \end{cases}$

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / ІО.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 13

з навчальної дисципліни Технології (Програмування)

Тема Поняття рекурсії. Реалізація рекурсії. Рекурсивні математичні функції.

Мета заняття Сформувати у студентів вміння реалізації рекурсивного звернення до процедур і функцій. Дати уявлення про алгоритмічну стратегію «поділяй та володарюй».

Час – 2 години

План проведення лекції

- 1 Поняття рекурсії.
- 2 Приклади завдань рекурсивного рішення.
- 3 Алгоритмічна стратегія "розділяй і володарюй".

Навчальні матеріали лекції

1 Поняття рекурсії

Рекурсія (від латинського *recursio* - повернення) - це такий спосіб організації обчислювального процесу, при якому процедура або функція в ході виконання складових її операторів звертається сама до себе.

Для того, щоб таке звернення не було нескінченним, в тексті підпрограми має бути умова (термінальна умова), по досягненню якої подальшого звернення не відбувається. Таким чином, рекурсивне звернення може включатися тільки в одну з гілок підпрограми.

Немає ніяких обмежень на рекурсивні виклики підпрограм, необхідно тільки розуміти, що кожен черговий рекурсивний виклик приводить до утворення нової копії локальних об'єктів підпрограми і усі ці копії, відповідні ланцюжку активізованих і не завершених рекурсивних викликів, існують незалежно один від одного.

Рекурсія досить широко застосовується в програмуванні, що засновано на рекурсивній природі багатьох математичних алгоритмів. А також Ви повинні знати, що будь-який рекурсивний алгоритм можна перетворити в еквівалентний ітеративний (тобто що використовує циклічні конструкції).

У великих і складних програмах іноді доводиться замінювати рекурсію на ітерацію. Річ у тому, що рекурсія пов'язана з багатократними викликами процедур, а це декілька менш ефективно при виконанні в порівнянні з використанням циклів. Проте рекурсивні версії програм, як правило, набагато коротше і наочніше.

2. Приклади завдань рекурсивного рішення

Хорошою ілюстрацією механізму рекурсії є функція для обчислення факторіалу натурального числа. Згадаємо, що факторіалом числа називається добуток усіх натуральних чисел від 1 до цього числа включно:

$$N! = 1 * 2 * 3 * (N - 1) * N$$

$$1! = 1$$

$$0! = 1$$

Наведена функція використовує рекурсивні звернення і заснована на очевидному рекурентному співвідношенні:

$$N! = (N - 1)! * N$$

Іншими словами, щоб набути значення факторіалу від числа N, досить помножити на N значення факторіалу від попереднього числа:

```
int fact( int n)
{
    if (n<=1) //термінальна умова
        return 1;
    else
        return n*fact(n-1); //рекурентне співвідношення
}
```

Повністю програма, що обчислює факторіал числа, виглядатиме так (див. Лістинг 2.1):

Лістинг 2.1.

```
// include <iostream.h>

int fact( int n)
{
    if (n<=1) //термінальна умова
        return 1;
    else
        return n*fact(n-1); //рекурентне співвідношення
}

void main()
{
    int m;
    cout<<"Введіть натуральне число ";
    cin>> m;
    cout<<"Для заданного числа "<<m<<"значення факториала равно "<<
fact(m);
}
```

Після запуску програми на екран виводиться запит " Введіть натуральне число ", потім з клавіатури прочитується введене значення m. У вираженні fact(m)

викликається функція `fact` з параметром-значенням `m`. У підпрограмі-функції перевіряється умова $p \leq 1$. Якщо вона виконується, то функція `fact` повертає значення 1 і на цьому виконання функції завершується. Якщо умова $p \leq 1$ не дотримується, то виконується обчислення добутку $p * fact(n-1)$. Обчислення добутку носить рекурсивний характер, оскільки при цьому здійснюється виклик функції `fact(n-1)`, значення якої обчислюється, у свою чергу, через виклик функції `fact`, параметром якої також буде функція `fact`, і так далі, до тих пір, поки значення формального параметра `p` не дорівнюватиме 1. Отже, при виконанні рекурсивної підпрограми здійснюється багатократний перехід від деякого поточного рівня організації алгоритму до нижнього рівня послідовно до тих пір, поки не буде отримано тривіальне рішення поставленої задачі.

При організації рекурентних викликів функцій необхідно строге дотримання двох умов:

1 У тілі рекурсивної підпрограми обов'язково повинне бути умова, при якому припиняється рекурсивний виклик підпрограми - термінальна умова.

2 В основі рекурсивних рішень обчислювальних процесів лежать рекурентні співвідношення.

Проілюструємо це на прикладі обчислення значення рекурсивної математичної функції:

Постановка задачі: Одержати m -й член послідовності: $\frac{\cos^{m-1}(3x)}{(x-5)^{m+2}}$. Значення x й m вводяться користувачем.

Розробка рекурентного співвідношення й термінальної умови:

Термінальна умова: $y_1 = \frac{1}{(x-5)^2}$;

Рекурентне співвідношення: $y_m = y_{m-1} * \frac{\cos(3x)}{(x-5)^2}$;

Текст програми:

```
// include <iostream.h>

float P( float y,int n)
{
    if (n==1) //термінальна умова
        return 1/((y-5)*(y-5));
    else
        return          cos(3*y)/((y-5)*(y-5))*P(y,n-1); //рекурентне
співвідношення
}

void main()
```

```

{
    int m;
    float x;
    cout<<"Введіть аргумент функції ";
    cin>> x;
    cout<<"Введіть номер члена послідовальності ";
    cin>> m;
    if (x==5)
        cout<<"Для заданного x="<<x<<"послідовальність не определена ";
    else
        cout<<m<<"-й член послідовальності равен "<< P(x,m);
}

```

3 Алгоритмічна стратегія "розділяй і володарюй"

Багато алгоритмів використовують два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. Така рекурсивна схема, мабуть, є найбільш важливий випадок добре відомого методу "розділяй і володарюй" (divide and conquer) розробки алгоритмів.

Як приклад розглянемо завдання відшукування максимального з N елементів, збережених в масиві $a[1], \dots, a[N]$ з елементами типу float. Це завдання легко може бути вирішене за один прохід масиву :

```

max=a[1];
for (i=1; i<n; i++)
    if (a[i] > max) max :=a[i];

```

Рекурсивне рішення типу "розділяй і володарюй" - ще один простий (хоча абсолютно інший) спосіб рішення тієї ж задачі (див. Лістинг 3.2):

Лістинг 3.2.

```

float max (float* a, int first, int last);
{
    int m;
    m=(first+last)/2;
    if (first == last)
        return a[first]
    else
    {
        if ( max(a, first, m)> max(a, m+1, last))
            return max(a, first, m);
        else
            return max(a, m+1, last);
    }
}

```

Найчастіше підхід "розділяй і володарюй" використовують через те, що він забезпечує швидші рішення, ніж ітераційні алгоритми.

Основним недоліком алгоритмів типу "розділяй і володарюй" являється те, що ділять завдання на незалежні під задачі. Коли під задачі незалежні, це часто призводить до неприпустимо великих витрат часу, оскільки одні і ті ж під задачі починають вирішуватися по декілька разів.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Яка функція називається рекурсивною?
- 2 Які умови є необхідними для правильної організації рекурсивних викликів функцій?
- 3 Що таке термінальна умова?
- 4 Навіщо використовується рекурентне співвідношення?
- 5 В чому полягає ідея алгоритмічної стратегії «розділяй та володарюй»?

Завдання для самоперевірки засвоєного матеріалу на лекції

Напишіть програми, що демонструють виконання рекурсивного алгоритму для завдань:

1 На друк виводиться казка "Про попа і його собаку" певне число разів. ("У попа був собака, він її любив. Вона з'їла шматок м'яса - він її убив. У землю закопав, напис написав ..)

2 Напишіть рекурсивний алгоритм знаходження:

- а) n -го члена геометричної прогресії
- б) знаходження суми n членів арифметичної прогресії
- в) знаходження суми n членів геометричної прогресії
- г) знаходження n -го члена ряду Фібоначчі.

3 Сформулюйте термінальну умову та рекурентне співвідношення для обчислення n -го члена наступної послідовності: 2, 4, 16, 265, ...

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.

2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання)
[Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 14

з навчальної дисципліни Технології (Програмування)

Тема Файлова система в мові C. Основні функції роботи з файлами.
Мета заняття Ввести поняття моделі потоків введення-виведення в стилі C програм. Охарактеризувати роботу стандартних потоків бібліотеки stdio. Пояснити призначення основних функцій роботи з файлами.
Час – 2 години

План проведення лекції

- 1 Модель потоку введення – виведення
- 2 Робота з файлами

Навчальні матеріали лекції

- 1 Модель потоку введення – виведення

Більшість сучасних операційних систем доручають керування пристроями введення – виведення спеціалізованим драйверам, а потім програми звертаються до них за допомогою засобів бібліотеки введення - виведення, що забезпечують максимально однаковий зв'язок з різними джерелами й адресатами даних. Загалом , драйвери пристроїв глибоко впроваджені в операційну систему й недоступні для більшості користувачів, а бібліотечні засоби введення - виведення забезпечують абстракцію введення – виведення, так що програміст не повинен думати про пристрої і їхні драйвери.

Коли використовується така модель, вся вхідна й вихідна інформація може розглядатися як **потоки** байтів (символи), оброблювані засобами бібліотеки введення - виведення . Наша робота як програмістів зводиться до наступного: (див. рис. 1.1)

- 1 Налаштувати потоки введення - виведення на відповідні джерела й адресати даних.
- 2 Прочитати й записати їхні потоки.

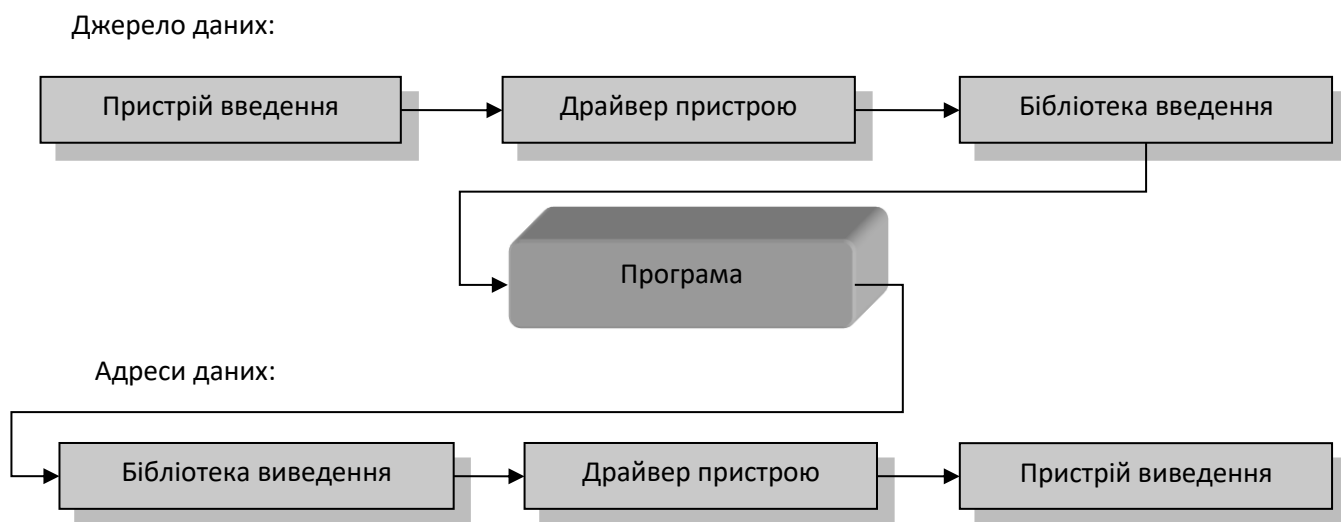


Рисунок 1.1. Модель потоків введення – виведення.

Як й інші сучасні мови, C++ не має вбудованих операцій введення - виведення. Вони винесені із самої мови в бібліотеку. Програми C++ можуть використати дві бібліотеки: стандартну бібліотеку введення – виведення `stdio`, успадковану з мови C, і більше нову бібліотеку `iostream`, розроблену спеціально для C++.

Обидві бібліотеки підтримують файлове введення й виведення. Бібліотека C складна й породжує помилки. Це важливо знати програмістам, що використовують успадковані програми на C.

Бібліотека C++ сприяють зниженню кількості помилок у програмах, але більше складна й громіздка. Часто одна й та ж дія може виконуватись двома способами. Щоб зрозуміти, як працює бібліотека C++ , потрібно знати про використання класів C++ , про спадкування, множинне спадкування й інші концепції, які поки не обговорювалися. От чому в даній лекції описуються лише базові засоби, що дозволяють зчитувати дані з файлів на диску й записувати їх у файл.

Стандартну бібліотеку введення – виведення `stdio`

Стандартна бібліотека мови C містить визначення типів `stdin` – стандартний потік введення, `stdout` – стандартний потік виведення та `stderr` – стандартний потік помилок.

Потік `stdin` робить наступне:

- 1 Одержує послідовність символів "звідкись" (наприклад, з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера).
- 2 Перетворює послідовності символів у значення різних типів.

Для взаємодії з операційною системою потік `stdin` використовує буфер. Буфер - це структура даних, яку потік `stdin` використає для зберігання інформації, отриманої від вас у ході взаємодії з операційною системою. При цьому буферизація може виявитися

візуально помітною для користувача. Коли ви використаєте потік `stdin` пов'язаний із клавіатурою, усе, що ви введете, залишиться в буфері, поки ви не натиснете клавішу `Enter` (увести й перейти на новий рядок), і якщо ви передумали, то можете видалити символи за допомогою клавіші `<Backspace>` (поки не нажали клавішу `<Enter>`).

Потік `stdin` можна зобразити наступним чином (див. рис. 1.2):

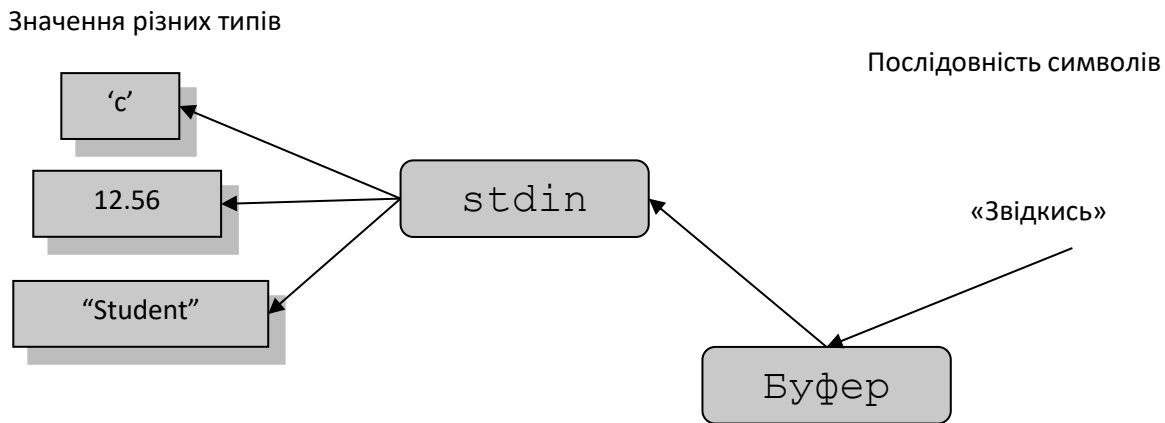


Рисунок 1.2. Модель потоку `stdin`.

Потік `stdout` робить наступне:

- 1 Перетворює значення різних типів у послідовності символів.
- 2 Посилає ці символи "кудись" (наприклад, на консоль, у файл, основну пам'ять або на інший комп'ютер).

Потік `stdout` можна зобразити наступним чином (див. рис. 1.3):

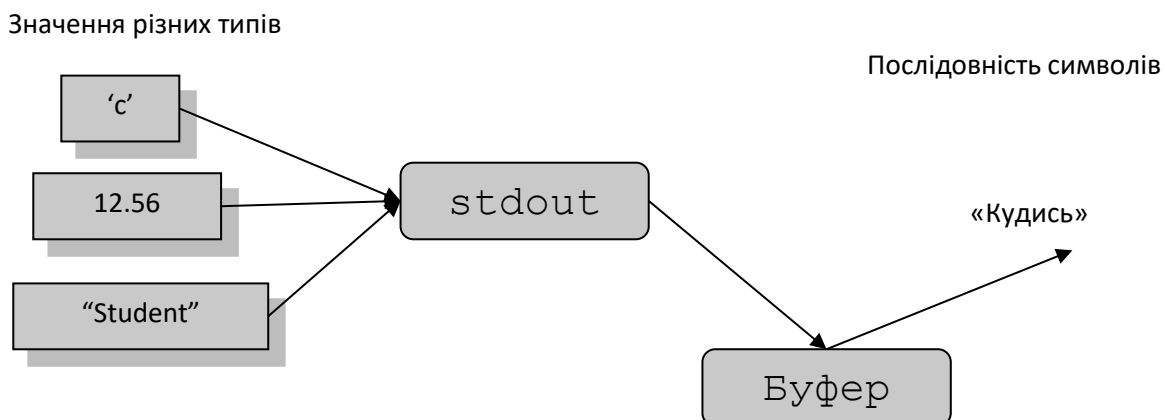


Рисунок 1.3. Модель потоку `stdout`.

2 Робота з файлами

Звичайно ми маємо набагато більше даних, чим здатна вмістити основна пам'ять нашого комп'ютера, тому більша частина інформації зберігається на дисках або інших

засобах зберігання даних високої ємності. Такі пристрої також запобігають зникнення даних при вимиканні комп'ютера – такі дані є персистентними.

На самому нижньому рівні файл просто являє собою послідовність байтів, пронумерованих починаючи з нуля.

Файл має формат, інакше кажучи, набір правил, що визначають зміст байтів. Наприклад, якщо файл є текстовим, то перші чотири байти являють собою чотири символи. З іншого боку, якщо файл зберігає бінарне представлення цілих чисел, то перші чотири байти використовуються для бінарного представлення одного цілого числа. Формат стосовно файлів на диску грає ту ж роль, що й типи стосовно об'єктів в основній пам'яті. Ми можемо приписати бітам, записаним у файлі, певний зміст тоді й тільки тоді, коли відомий його формат.

При роботі з файлами потік `stdout` перетворить об'єкти, що зберігаються в основній пам'яті, у потоки байтів і запише їх на диск. Потік `stdin` діє навпаки; інакше кажучи, він зчитує потік байтів з диска й становить із них об'єкт (див. рис. 2.1).

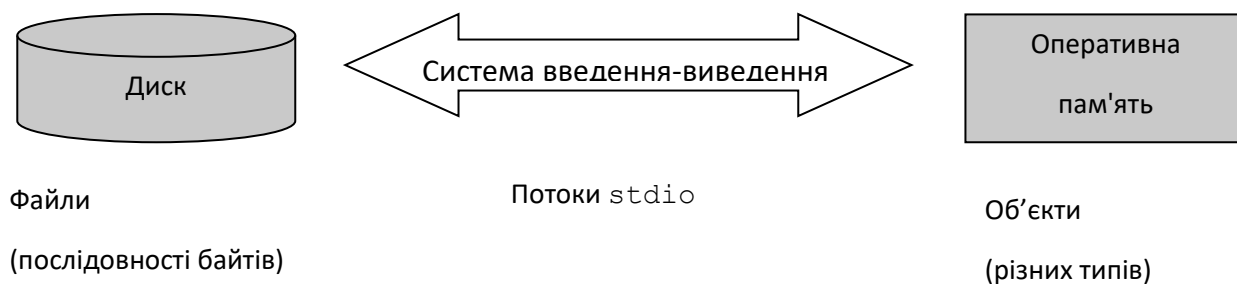
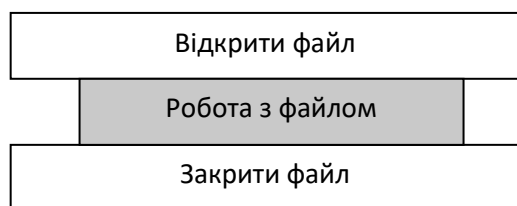


Рисунок 2.1. Файловий обмін даними.

Найчастіше ми припускаємо, що байти на диску є символами з звичайного набору символів. Це не завжди так, але, оскільки інші подання обробити нескладно, ми, як правило, будемо дотримуватися цього припущення. Крім того, будемо вважати, що всі файли перебувають на дисках (тобто на магнітних пристроях зберігання даних). І знов-таки це не завжди так (згадаєте про флеш-пам'ять), але на даному рівні програмування фактичний пристрій зберігання не має значення. Це одне з головних перевага абстракцій файлу й потоку.

Робота з файлами будується за принципом сандвіча:



Поняття "відкрити файл" означає "почати з ним роботу", зробити його активним і заблокувати обіг інших програм до цього файлу. При закритті файлу він звільняється (тепер з ним можуть працювати інші програми) і всі ваші зміни вносяться на диск.

Доступ до файлу здійснюється за допомогою покажчика. Покажчик на файл описується в такий спосіб:

```
FILE *fp;
```

Тип `FILE` - це структура, певна в `<stdio.h>` й утримує деяку інформацію про файл: наприклад, прапори стану файлу, розмір буфера, покажчик на буфер й ін. Описаний покажчик можна зв'язати з певним файлом у момент відкриття даного файлу. Це здійснюється за допомогою функції:

```
fopen ("шлях до файлу", "тип доступу") ,
```

що повертає покажчик на файл або `NULL` у випадку помилки.

Наприклад, у результаті виконання оператора

```
fp = fopen ("ex1.txt", "w") ,
```

файл `ex1.txt` відкритий для запису, а в програмі на цей файл можна послатися за допомогою покажчика `fp` (тобто функція `fopen()` бере зовнішнє представлення файлу – його фізичне ім'я – і ставить йому у відповідність внутрішнє логічне ім'я, що далі й буде використатися в програмі).

Як тип доступу можуть бути зазначені наступні параметри:

"w" - існуючий файл відкривається для запису, при цьому його старий вміст затирається. Маркер файлу вказує на його початок. Якщо файл ще не існував, то він створюється (якщо це можливо);

"r" - існуючий файл відкривається для читання (з початку). Якщо файл не існує - це помилка;

"a" - файл відкривається для дозапису в кінець. Якщо файл ще не існує, він буде створений;

"w+" - відкривається порожній файл для читання й запису, при цьому його старий вміст затирається. Якщо файл ще не існував, то він створюється;

"r+" - існуючий файл відкривається для читання й запису. Якщо файл не існує - це помилка;

"a+" - файл відкривається для читання й додавання інформації в кінець.

Файл можна відкрити у двійковому (b) або текстовому (t) режимі. Ці символи записують у другому параметрі, наприклад, "rb" або "rt". Двійковий режим означає, що символи перекладу рядка й повернення каретки (0x13 й 0x10) обробляються точно так само, як й інші. У текстовому режимі ці символи перетворюються в одиночний символ перекладу рядка. За замовчуванням файли відкриваються в текстовому режимі.

Приклад:

```
FILE *f = fopen ("d:\\cpp\\data", "rb+");
```

(відкривається файл `d:\\cpp\\data` для читання й запису у двійковому режимі й пов'язує з покажчиком `f`).

Показчик `f` використовується в подальших операціях з потоком (файлом).

По закінченні роботи з файлом він повинен бути закритий. Для цього використовується функція

```
fclose (показчик_файлу)
```

При закритті файлу очищаються буфери.

`fclose(f)` повертає 0 при успішному закритті й EOF в інших випадках:

```
if (fclose(f)!=0) printf("Ошибка при закрытии файла %s\n");
```

EOF - константа, що повідомляє про кінець файлу (ціле негативне) або про помилку при роботі з функціями читання.

Іноді програма не може відкрити файл. Якщо файл відкривається на читання, це можливо в наступних випадках:

- невірно задане ім'я файлу або файлу немає на диску;
- файл використовується іншою програмою й заблокований.

Якщо файл відкривається на запис, операція може закінчитися невдало, якщо:

- на диску немає місця;
- файл захищений від запису;
- невірно задане ім'я файлу (наприклад, воно містить дві крапки, знак питання й т.п.).

Якщо файл не вдалося відкрити, функція `fopen` повертає спеціальне нульове значення (нульовий показчик), що позначається `NULL`. Тому треба завжди перевірити правильність відкриття файлу, особливо в режимі читання. Якщо файл не є відкритий, треба вивести повідомлення про помилку й вийти із програми.

```
if ( fp == NULL )
{
printf("Ошибка открытия файла");
return;
}
```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Охарактеризуйте модель потоку введення – виведення, що використовується в мові C/C++.
- 2 Опишіть призначення та принцип дії основних функцій стандартної бібліотеки введення – виведення `stdio`
- 3 Назвати два види файлових потоків. Пояснити різницю між ними.
- 4 Сформулювати порядок роботи з файлами в C програмі.
- 5 Що таке файловий показчик? Як описується файловий показчик?

- 6 Сформулювати призначення та описати прототип функції `open()`.
- 7 Перерахувати та пояснити відмінність між основними типами доступу до файлів.
- 8 Сформулювати призначення та описати прототип функції `fclose()`.

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Обрати правильну відповідь:
 - 1.1 Стандартний потік введення `stdin` бібліотеки `stdio.h` робить наступне:
 - А. одержує послідовність символів з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера та перетворює послідовності символів у значення різних типів;
 - В. тільки одержує послідовність символів з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера;
 - С. тільки перетворює послідовності символів у значення різних типів.
 - 1.2 Стандартний потік введення `stdout` бібліо-теки `stdio.h` робить наступне:
 - А. перетворює значення різних типів у послідовності символів;
 - В. посилає символи вихідного потоку на консоль, у файл, в основну пам'ять або на інший комп'ютер;
 - С. перетворює значення різних типів у послідовності символів та посилає ці сим-воли на консоль, у файл, в основну па-м'ять або на інший комп'ютер.
- 2 Написати функцію відкриття файлу `data.txt`:
 - на читання в текстовому режим;
 - на запис в текстовому режимі;
 - як порожній на читання й запис в текстовому режимі;
 - на запис в кінець в текстовому режимі.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 15

з навчальної дисципліни Програмування

Тема	Функції роботи з файлами послідовного доступу.
Мета заняття	Ознайомити студентів з правилами використання стандартних функцій введення-виведення при роботі з текстовими файлами.
Час	– 2 години

План проведення лекції

- 1 Функції посимвольного читання-запису
- 2 Функції `fgets()` та `fputs()`
- 3 Функції `fprintf()`, `fscanf()`.

Навчальні матеріали лекції

Функції послідовного текстового введення-виведення дозволяють:

- вводити символи або рядки символів із текстового потоку (файлу);
- виводити в потік (файл) символи або рядки символів.

1 Функції посимвольного читання-запису

Функція `getc()`.

Функція `getc()` зчитує символ з файлу. Вона повертає цілочисельну змінну, молодший байт якої містить уведений символ. Якщо виникла помилка, то старший байт цієї змінної дорівнює 0.

Якщо при читанні виявлений кінець файлу, функція `getc()` повертає константу EOF. Таким чином у C кінець файлу виявляється після зчитування.

Розглянемо програму, яка в текстовому файлі "file1.txt" підраховує кількість символів, відмінних від пробілу та символу ознаки кінця рядка:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
    char ch;
    int k=0;
    FILE *f;
    clrscr();
```

```

        f=fopen("d:\\file1.txt","r"); // відкриття текстового файлу для
читання
        if (f==NULL) // обробка помилки відкриття файлу
        {
            cout<<endl<<"Ошибка открытия файла. Для выхода из программы
нажмите Enter"; getch(); return;
        }
        ch = getc(f); // прочитати 1 символи з файлу
        while (ch!=EOF) // поки не кінець файлу
        {
            if ((ch!=' ') && (ch!='\n'))
            {
                k++;
            }
            ch = getc(f); // прочитати черговий символи з файлу
        }
        cout<<endl<<"k="<<k<<endl;
        fclose(f); // закриття файлу
        getch();
        return;
    }

```

Функція putc() .

Функція putc() записує символ у файл, відкритий за допомогою функції fopen(). Прототип цієї функції виглядає так:

```
int putc( int ch, FILE* fp)
```

Тут параметр fp являє собою покажчик файлу, а аргумент ch є символом, що підлягає виведенню. Покажчик файлу повідомляє функції putc() у який саме файл варто зробити запис. Незважаючи на те, що параметр ch має тип int, у файл записується лише молодший байт. Якщо функція putc() виконана успішно, вона повертає символ, записаний нею у файл. У протилежному випадку вона повертає константу EOF.

Розглянемо програму, яка по вхідному текстовому файлу формує новий файл, записуючи в нього лише кожен 3-й символ вхідного файлу.

```

#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
    char ch;
    int k=0;
    FILE *in;
    FILE *out;

```



```

clrscr();
in=fopen("d:\\file2.txt","r"); // відкриття вхідного текстового
файлу для читання
if (in==NULL) // обробка помилки відкриття файлу
{
    cout<<endl<<"Error1"; getch(); return;
}
out=fopen("d:\\file3.txt","w"); // відкриття вихідного текстового
файлу для запису
if (out==NULL) // обробка помилки відкриття файлу
{
    cout<<endl<<"Error2"; getch(); return;
}
while ((ch = getc(in))!=EOF) //поки не кінець вхідного файлу
{
    if ((k%3)==0)
    {
        putc(ch, out); // вивести символ в вихідний файл
    }
    k++;
}
fclose(in); fclose(out);
return;
}

```

2 Функції fgets() та fputs()

Для рядкового введення-виведення використовуються наступні функції:

1) `char* fgets(char* s, int n, FILE* f)`, де
`char* s` – адреса, за якою розміщуються зчитані байти (символи),
`int n` – кількість зчитаних байтів (символів),
`FILE* f` – покажчик на файл, з якого відбувається зчитування.

Зчитування байтів закінчується після отримання `n-1` байтів або при досягненні керуючого символу ‘`\n`’ – ознаки кінця рядка. Керуючий символ теж передається в рядок `s`. Рядок в будь-якому випадку закінчується символом ‘`\0`’. При успішному завершенні зчитування функція повертає покажчик на зчитаний рядок. При помилці зчитування повертається `0 (NULL)`.

2) `int puts(char* s, FILE* f)`, де
`char* s` – адреса, за якою отримуються байти, які записуються в файл.
`FILE* f` – покажчик на файл, в який відбувається запис.

Символ кінця рядка ('\0') в файл не записується. Функція повертає EOF, якщо при запису в файл виникла помилка. При успішному записі функція повертає додатне число. Розглянемо фрагмент програми (див. Лістинг 2.1) копіювання файлу in в файл out

Лістинг 2.1.

```
int MAXLINE=255;      //максимальна довжина рядка
FILE *in,              //висхідний файл
*out;                  //отриманий (результуючий) файл
char* buf[MAXLINE]; /*рядок символів, за допомогою якого відбувається
копіювання*/
in=fopen("f1.txt","r"); //відкриття висхідного файлу для читання
out=fopen("f2.txt","w"); //відкриття результуючого файлу для запису
while(fgets(buf,MAXLINE,in)!=0) /* поки не кінець файлу, читати
байти з файлу in в рядок buf */
fputs(buf,out);        //записати байти з рядка buf в файл out
fclose(in);fclose(out); // закрити обидва файли
```

3 Функції fprintf(), fscanf().

Ці функції відрізняються від відповідних printf(), scanf() тільки тим, що додається один параметр - покажчик на FILE створений при виклику функції fopen().

fprintf() – форматоване виведення файл, пов'язаний з потоком. При цьому файл повинен бути відкритий як текстовий, у режимі, що допускає запис. Синтаксис:

```
int fprintf (FILE * Потік, Формат, Список_змінних);
```

fscanf() – форматоване читання значень змінних з файлу, пов'язаного з потоком. Файл повинен бути відкритий як текстовий у режимі, що допускає читання. Синтаксис:

```
int fscanf (FILE * Потік, const char* Формат, Список_адрес);
```

Як приклад, розглянемо програму (див. Лістинг 3.1), яка зчитує з файлу input.txt дійсні числа й поміщає їх в динамічний масив дійсних чисел, подвоює ці числа та записує їх в файл output.txt

Лістинг 3.1.

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
void main()
{
int i,N,m;
float* A;
cout<<"Enter N"<<endl; cin>>N;
A=new float [N];
```

```

FILE *fp;
fp = fopen( "d:\\input.txt", "r" );
if ( fp == NULL )
{
    printf("Нет файла данных");
    return;
}
for ( i = 0; i < N; i ++ )
{
    m=fscanf(fp, "%f", &A[i] );
    if ( EOF == m)
    { printf("Не хватает данных в файле ");
      break;
    }
    if ( 0 == m )
    { printf("Некорректные данные ");
      break;
    }
}
fclose ( fp );
for ( i = 0; i < N; i ++ )
A[i] = A[i] * 2;
fp = fopen( "d:\\output.txt", "w" );
for ( i = 0; i < N; i ++ )
fprintf ( fp, "%f\n", A[i] );
fclose ( fp );
return;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Пояснити роботу функції `getc()` посимвольного зчитування з текстового файлу.
- 2 Пояснити роботу функції `putc()` посимвольного запису в файл.
- 3 Пояснити роботу функції `fgets()` рядкового зчитування з текстового файлу.
- 4 Пояснити роботу функції `fputs()` рядкового запису в файл.
- 5 Пояснити особливості роботи функцій `fprintf()` та `fscanf()` при роботі з текстовими файлами.

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Записати в текстовий файл K рядків, кожен з яких містить N символів '*’.

- 2 Дано рядок символів S та текстовий файл. Додати рядок S:
 - в кінець файлу;
 - в початок файлу.
- 3 В даному текстовому файлі підрахувати кількість рядків та символів.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 16

з навчальної дисципліни Технології (Програмування)

Тема Функції роботи з бінарними файлами.

Мета заняття Пояснити роботу функцій введення-виведення `fread()`, `fwrite()`, функції довільного доступу `fseek()`. Показати приклад роботи з файлом структур.

Час – 2 години

План проведення лекції

- 1 Читання / запис двійкових даних. Функції `fread()`, `fwrite()`
- 2 Довільний доступ до фалів. Функції `fseek()`, `ftell()`
- 3 Приклад програми роботи з файлом структур

Навчальні матеріали лекції

- 1 Читання / запис двійкових даних. Функції `fread()`, `fwrite()`

Функція

```
size_t fread (void *buffer, size_t size, size_t cout, FILE *stream);
```

зчитує `count` елементів довжиною `size` байтів в область, задану покажчиком `buffer` з потоку `stream`;

Функція

```
size_t fwrite (const void buffer, size_t size, size_t n, file stream);
```

записує `n` елементів довжиною `size` байтів з буфера, заданого покажчиком `buffer` у потік `stream`. Повертає число успішно записаних елементів звичайно воно `= size`, але при помилці воно може бути менше.

Тип `size_t` визначений у файлі `stdio.h`, означає «будь-який цілий». У байт, зайнятих цим «будь-яким цілим» можна одержати за допомогою функції `sizeof()`. Наприклад `sizeof(int)` - скільки байтів займає тип `int`.

Приклад:

```
char buf[256]; // масив символів
fwrite(buf, 256, 1, fp); // перед buf не ставиться &, тому що buf = &buf[0]
int k[10]; // масив цілих
fwrite(k, sizeof(int), 10, fp);
```

У прототипах цих функцій використовується `void *buffer`. Він діє у якості «пастки» для всіх типів даних.

2 Довільний доступ до фалів. Функції `fseek()`, `ftell()`

Функція

```
int fseek (file *f, long off, int org);
```

переміщає поточну позицію у файлі, пов'язаному з потоком `f` (відкритим за допомогою `fopen()`) на позицію `off`, відлічувану від значення `org`, що повинне бути дорівнює однієї з 3х констант, певних в `stdio.h`:

- `seek_cur` - від поточної позиції,
- `seek_end` - від кінця файлу,
- `seek_set` - від початку файлу.

Функція повертає значення 0 при успішному завершенні та 1 – якщо помилка.

Функція

```
Long int ftell(file *f);
```

визначає поточну позицію у файлі `f` і повертає, як довге ціле позитивне число байтів, відлічуване від початку файлу. Якщо виникла помилка, функція поверне значення -1.

Як правило, довільний доступ потрібен лише при роботі з бінарними файлами. Оскільки текстові файли можуть вміщати керуючі символи, кількість байтів, на яке слід виконати переміщення файлового курсору, може не відповідати очікуваному символу. Функція `fseek()` коректно працює з текстовим файлом лише тоді, коли позиція курсора була заздалегідь визначена функцією `ftell()` та макросом `seek_set`.

Приклад програми посимвольного виводу текстового файлу відкритого в бінарному режимі у зворотному порядку (див. лістинг 2.1).

Лістинг 2.1. Посимвольна обробка файлу

```
#include <stdio.h>
#include <stdlib.h>
#define ctrl_z '0_32' // це ознака eof у текстовому файлі MS-DOS
#define n[50]

int main()
{char file[n]; // для введення імені оброблюваного файлу
char ch;
file *fp;
long c,k;
puts("Уведіть ім'я оброблюваного файлу");
```

```

gets(file);
if ((fp=fopen(file,"rb"))==null)
/* якщо помилка при відкритті файлу (тільки для читання в бінарному
режимі) */
{printf("Не можу відкрити файл %s\n",file);
exit(1);
}
fseek(fp,0L,seek_end); /* перехід до кінця файлу. 0L - це довга ціла
константа «нуль» */
r=tell(fp); // у цьому випадку це в байтів у файлі
for (c=1L; c<=r; c++)
{fseek(fp,-c,seek_end); //рух назад
ch=getc(fp); // читання чергового символу
if ((ch!=ctrl_z)&&(ch!='\r'))
/* для MS-DOS якщо це не символ кінця файлу (^Z) і не символ '\r' (див.
самий початок теми) */
putchar(ch); // вивід на екран поточного символу
}
putchar('\n');
fclose(fp);
return 0;
}

```

3 Приклад програми роботи з файлом структур

Нехай файл містить наступну інформацію про співробітників патрульної служби ДАІ: ПІБ співробітника, стаж роботи, табельний номер зброї, стать, номер ланки.

Програма дозволяє:

- виконувати коректне відкриття файлу;
- додавання нових записів в файл;
- перегляд вмісту файлу;
- редагування запису.

Програма містить користувацьке меню.

Текст програми наведено в лістингу 3.1.

Лістинг 3.1. програма обробки файлу структур.

```

#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

// описание структуры данных

struct Record
{char fio[25]; //ФИО сотрудника
int staz;

```

```

        int tab; // табельный номер оружия
        char gen; //пол: F - женский, M - мужской
        int zveno; //номер звена
    };

// Прототипы функций

void Create_F(); //корректное открытие  файла
void Intake(); //добавление данных
void View(); //просмотр файла
void Edit(); //редактирование записи


void main()
{
    int punkt;
    char c = 'y';

// Организация меню программы

    while(c=='y')
    { clrscr();
        puts("Программа учета сотрудников патрульной службы ГАИ Жовтневого
р-на");
        cout<<endl;
        puts("Главное меню:");
        puts("1. Корректное открытие файла");
        puts("2. Добавление сотрудников в файл");
        puts("3. Просмотр данных о сотрудниках");
        puts("4. Редактирование записей");
        puts("5. Выход из программы");
        puts("Выберите пункт меню: ");
        cin>>punkt;
        // cin.sync();
        switch(punkt)
        {
            case 1: Create_F(); break;
            case 2: Intake(); break;
            case 3: clrscr();View(); break;
            case 4: Edit(); break;
            case 5: c='n'; puts("Нажмите Enter завершить работу"); getch();
break;

        }

    }

    return;
}


//функция корректного открытия файла

void Create_F ()
{

```



```

int i;
FILE *fp;
if((fp = fopen("d:\\pat.dan", "r+b"))==NULL)
{ puts("Файл не существует. Хотите создать? (1 - да):");
  cin>>i;
  if(i==1)
  {
    fp = fopen("d:\\pat.dan", "wb"); //создание нового файла взамен старого
    fclose(fp);
    puts("Создан новый файл. Для возврата в главное меню нажмите Enter");
    getch();
  }
  else
  {
    puts("Вы отказались создавать новый файл. Для возврата в главное
меню нажмите Enter ");
    getch();
  }
}
else
{
  puts("Файл существует! Вы хотите удалить старый файл и создать новый?
(1 - да): ");
  cin>>i;
  if(i==1)
  {
    fclose(fp);
    fp = fopen("d:\\pat.dat", "wb"); //создание нового файла взамен старого
    fclose(fp);
    puts("Создан новый файл. Для возврата в главное меню нажмите Enter
");
    getch();
  }
  else
  {
    puts("Вы отказались создавать новый файл. Для возврата в главное меню
нажмите Enter ");
    fclose(fp);
    getch();
  }
}
return;
}

```

//функция добавления записей в файл

```

void Intake()
{
  clrscr();
  FILE * fp;
  Record PATR;
  char c='y';

```

```

if((fp = fopen("d:\pat.dan", "a+b"))==NULL)
{
    fputs("Ошибка открытия файла pat.dan", stderr);
    puts("Для возврата в главное меню нажмите Enter");
    getch();
    return;
}
else
{
    while (c=='y')
    {
        cout<<endl<<"Введите данные о   сотруднике\n";
        cout<<endl<<"ФИО сотрудника:   ";
        cin>>PATR.fio;
        cout<<endl<<"Стаж работы:   ";
        cin>>PATR.staz;
        cout<<endl<<"Табельный номер:   ";
        cin>>PATR.tab;
        cout<<endl<<"Пол сотрудника:   ";
        cin>>PATR.gen;
        cout<<endl<<"Номер звена:   ";
        cin>>PATR.zveno;

        c=fwrite(&PATR, sizeof(PATR), 1, fp);
        cout<<endl<<"Продолжить ввод? (y/n):   ";
        cin.sync();
        cin>>c;
        getch();
    }
    cout<<endl<<"Для возврата в главное меню нажмите Enter   ";
    getch();
    fclose(fp);
    return;
}
}

//функция просмотра содержимого файла

void View()
{
    int i;
    FILE * fp;
    Record PATR;
    if((fp = fopen("pat.dan", "r+b"))==NULL)
    {
        fputs("Ошибка открытия файла patrol.dat", stderr);
        puts("Для возврата в главное меню нажмите любую клавишу.Enter");
        getch(); return;
    }
    else
    {
        rewind(fp); //установка маркера в начало файла
        i = 0;

```

```

while(fread(&PATR, sizeof(PATR), 1, fp)==1) i++;
if(i==0)
{ puts("Файл pat.dan пуст!");
  fclose(fp);
  puts("Для возврата в главное меню нажмите любую клавишу.Enter..");
  getch(); return;
}
else
{
  puts("Текущее содержимое файла pat.dan");
  cout<<"Сотрудников в файле: "<<i<<endl;
  rewind(fp);
  while (!(feof(fp)))
  { int n=1;
    while(!(feof(fp)) && (n<=18))
    {
      if (fread(&PATR, sizeof(PATR), 1, fp)==1)
      {
        cout<<endl<<n<<"    "<<PATR.fio<<"    "<<PATR.staz<<"    "<<PATR.tab<<"
"<< PATR.gen<<"    "<< PATR.zveno<<endl;
        n++;
      }
    }

    getch();
  }
  cout<<endl<<"Для возврата  нажмите Enter "<<endl;

  getch();
  fclose(fp); return ;
}
}

//функция редактирования записи

void Edit()
{
  clrscr();
  int i, punkt;
  FILE * fp;
  Record PATR;
  View();
  cout<<"Выберите номер записи"<<endl;
  cin>>i;
  clrscr();
  if((fp = fopen("d:\pat.dan", "r+b"))==NULL)
  { fputs("Ошибка открытия файла patrol.dat", stderr);
    puts("Для возврата в главное меню нажмите Enter");
    getch();
    return;
  }
}

```

```

    }
else
{
    fseek(fp, (i-1)*sizeof(PATR), SEEK_SET);
    if (fread(&PATR, sizeof(PATR), 1, fp)==1)
    {
        cout<<endl<<PATR.fio<<"      "<<PATR.staz<<"      "<<PATR.tab<<"      "<<
PATR.gen<<"      "<< PATR.zveno<<endl;
        cout<<endl;
        cout<<"Виберите поле редактирования"<<endl;
        cout<<"1. ФИО сотрудника"<<endl;
        cout<<"2. Стаж работы"<<endl;
        cout<<"3. Табельный номер"<<endl;
        cout<<"4. Пол"<<endl;
        cout<<"5. Номер звена"<<endl;
        puts("Виберите пункт меню: ");
        cin>>punkt;
        cout<<endl;
        switch(punkt)
        {
            case 1: cin>>PATR.fio; break;
            case 2: cin>>PATR.staz; break;
            case 3: cin>>PATR.tab; break;
            case 4: cin>>PATR.gen; break;
            case 5: cin>>PATR.zveno; break;
        }
        fseek(fp, (i-1)*sizeof(PATR), SEEK_SET);
        fwrite(&PATR, sizeof(PATR), 1, fp);
    }
else
    cout<<"Ошибка выбора номера записи"<<endl;
}
cout<<endl<<"Для возврата нажмите Enter "<<endl;

getch();
fclose(fp); return ;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Пояснити роботу функції fread() блокового зчитування з бінарного файлу.
- 2 Пояснити роботу функції fwrite() блокового запису в файл.
- 3 Пояснити роботу функції fseek() довільного доступу до даних бінарного файлу.
- 4 Пояснити роботу функції ftell().

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Дані про книги зберегти у файлі `book.dat`. Якщо файл уже існує, то програма відображає його вміст, а потім дозволяє додати до нього дані про нові книги. Уведіть дані про нові книги вводяться в масив структур `bibl`, а потім весь масив (потрібне в записів без обліку того, що вже в ньому було) записується у файл. Вивід повного переліку книг походить із масиву структур, а не з файлу.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 17

з навчальної дисципліни Технології (Програмування)

Тема	Організація файлового введення-виведення з використанням класів ifstream, ofstream та fstream.
Мета заняття	Пояснити роботу основних функцій класів ifstream, ofstream та fstream. Розглянути приклади обробки текстових файлів.

Час – 2 години

План проведення лекції

- 1 Основи файлового введення/виведення в C++.
- 2 Приклади організації роботи з текстовими файлами

Навчальні матеріали лекції

- 1 Основи файлового введення/виведення в C++

Файлове введення/виведення підтримується тією же ієрархією класів, що й консольне введення/виведення. Для реалізації файлового введення/виведення, необхідно включити в програму заголовок <fstream>. У ньому визначено кілька класів, включаючи класи ifstream, ofstream й fstream. Ці класи є похідними від класів istream й ostream. Класи istream й ostream, у свою чергу, є похідними від класу ios, тому класи ifstream, ofstream й fstream також мають доступ до всіх операцій, обумовлених класом ios. В C++ файл відкривається за допомогою його зв'язування з потоком. Є три типи потоків: введення, виведення й введення/виведення. Перед тим як відкрити файл, потрібно, по-перше, створити потік. Для створення потоку введення необхідно оголосити об'єкт типу ifstream. Для створення потоку виведення - об'єкт типу ofstream. Потоки, які реалізують одночасно введення й виведення, повинні оголошуватися як об'єкти типу fstream. Наприклад, у наступному фрагменті створюється один потік для введення, один потік для виведення й ще один потік одночасно для введення й для виведення:

```
ifstream in; // введення
ofstream out; // виведення
fstream io; // введення й виведення
```

Після створення потоку, одним зі способів зв'язати його з файлом є функція open(). Ця функція є членом кожного із трьох поточкових класів. Тут показані її прототипи для кожного класу:

```
void ifstream::open(const char *ім'я_файлу, openmode режим = ios::in) ;
void ofstream::open(const char *ім'я_файлу, openmode режим = ios::out || ios::trunc);
```

```
void fstream::open(const char *ім'я_файлу, openmode режим = ios::in || ios::out);
```

Тут ім'я_файлу - ім'я файлу, у яке може входити й специфікатор шляху. Значення режим задає режим відкриття файлу. Воно повинне бути значенням типу openmode, що є перерахуванням, певним у класі ios. Значення режим може бути одним з наступних:

```
ios:: app
ios ate
ios:: binary
ios:: in
ios:: out
ios:: trunc
```

Ви можете об'єднати два або більше цих значення за допомогою оператора OR (. Розглянемо, що означає кожне із цих значень.

Значення ios::app викликає відкриття файлу в режимі додавання в кінець файлу. Це значення може застосовуватися тільки до файлів, що відкриваються для висновку.

Значення ios::ate задає режим пошуку кінця файлу при його відкритті. Хоча значення ios::ate викликає пошук кінця файлу, проте , операції уведення/висновку можуть бути виконані в будь-якому місці файлу.

Значення ios::in задає режим відкриття файлу для уведення.

Значення ios::out задає режим відкриття файлу для висновку.

Значення ios::binary викликає відкриття файлу у двійковому режимі. За замовчуванням всі файли відкриваються в текстовому режимі. У текстовому режимі має місце перетворення деяких символів, наприклад, послідовність символів "повернення каретки/переклад рядка" перетворюється в символ нового рядка. Якщо ж файл відкривається у двійковому режимі, такого перетворення не виконується. Запам'ятаєте, що будь-який файл, незалежно від того, що в ньому втримується - відформатований текст або неопрацьовані дані - може бути відкритий як у текстовому, так й у двійковому режимі. Відмінність між ними тільки у відсутності або наявності згаданого символного перетворення.

Значення ios::trunc приводить до видалення вмісту раніше існуючого файлу з тією же назвою й усіканню його до нульової довжини. При створенні потоку висновку за допомогою ключового слова ofstream кожної раніше існуючий файл із тим же ім'ям автоматично усикається до нульової довжини. У наступному фрагменті для висновку відкривається файл test:

```
ofstream mystream;
mystream.open("test");
```

У цьому прикладі параметр режим функції `open()` за замовчуванням встановлюється в значення, що відповідає типу потоку, що відкриває, тому немає необхідності вказувати його явно. Якщо виконання функції `open()` завершилося з помилкою, у булевому вираженні потік буде дорівнює значенню `false`. Цей факт можна використати для перевірки правильності відкриття файлу за допомогою, наприклад, такої інструкції:

```
// програма обробки помилки відкриття файлу
if (Imystream)
    cout << "Файл відкрити неможливо\n";
```

Як правило, перед тим як намагатися одержати доступ до файлу, варто перевірити результат виконання функції `open()`. Перевірити правильність відкриття файлу можна також за допомогою функції `is_open()`, що є членом класів `ifstream`, `ofstream` й `fstream`. Нижче показаний прототип цієї функції:

```
bool is_open();
```

Функція повертає істину, якщо потік удалося зв'язати з відкритим файлом, у противному випадку функція повертає неправду. Наприклад, у наступному фрагменті перевіряється, чи відкритий файл, пов'язаний з потоком `mystream`:

```
if ( !mystream.is_open() )
{
    cout << "Файл не відкритий\n";
    //

}
```

Хоча використати функцію `open()` для відкриття файлу в цілому правильно, часто ви цього робити не будете, оскільки в класів `ifstream`, `ofstream` й `fstream` є конструктори, які відкривають файл автоматично. Конструктори мають ті ж параметри, у тому числі й задають по умовчання, що й функція `open()`. Тому частіше ви будете користуватися таким способом відкриття файлу:

```
ifstream mystream ("myfile") ; // відкриття файлу для введення
```

Як уже встановлено, якщо з якихось причин файл не відкривається, змінному, відповідному потоку, в умовній інструкції буде дорівнює значенню `false`. Тому, незалежно від того, чи використаєте ви конструктор або явно викликаєте функцію `open()`, вам буде потрібно переконатися в успішному відкритті файлу шляхом зі значення потоку.

Для закриття файлу використайте функцію-член `close()`. Наприклад, щоб закрити файл, пов'язаний з потоком `mystream`, необхідна наступна інструкція:

```
mystream.close ( ) ;
```

Функція `close()` не має параметрів і значення, що повертає.

За допомогою функції eof(), що є членом класу ios, можна визначити, чи був досягнутий кінець файлу уведення. Нижче показаний прототип цієї функції:

```
bool eof ( ) ;
```

Функція повертає істину, якщо був досягнутий кінець файлу; у протилежному випадку функція повертає неправду.

Після того як файл відкритий, дуже легко вважати з нього або записати в нього текстові дані. Просто використайте оператори << й >> так само, як це робилося для консольного уведення/висновку, тільки замініте потік cin або cout тим потоком, що пов'язаний з файлом. Так само, як й оператори <<й >> для читання з файлу й запису у файл годяться функції 3 - fprintf() і fscanf(). Вся інформація у файлі зберігається в тім же форматі, як якби вона перебувала на екрані. Отже, файл, створений за допомогою оператора <<, представляє із себе файл із відформатованим текстом, і навпаки, будь-який файл, уміст якого зчитується за допомогою оператора >> повинен бути файлом з відформатованим текстом. Тобто, як правило, файли з відформатованим текстом, які ви будете обробляти, використовуючи оператори <<й >> варто відкривати в текстовому, а не у двійковому режимі. Двійковий режим більше підходить для невідформатованих файлів.

2 Приклади організації роботи з текстовими файлами

1. У представленій нижче програмі створюється файл для висновку, туди записується інформація, і файл закривається. Потім файл знову відкривається вже як файл для введення, і записана раніше інформація відтіля зчитується:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream fout ("test") ;           // создание файла для вывода
    if (! fout) { cout <<"Файл открыть невозможно\n"; return 1; }
    cout <<"Привет! \n";
    cout << 100 << ' ' << 64 << endl;
    fout.close () ;
    ifstream fin("test"); // открытие файла для ввода
    if(!fin) { cout << "Файл открыть невозможно\n"; return 1;}
    char str[80] ; int i;
    cin >>str >> i;
    cout << str << ' ' << i << endl;
    fin.close () ;
    return 0;
}
```

Після того як програма завершиться, проаналізуйте вміст файлу test. Воно буде наступним:

Привіт ! 100 64

Як уже встановлено, при використанні операторів << й >> для реалізації файлового введення/висновку, інформація форматується так само, як якби вона перебувала на екрані.

2. Розглянемо інший приклад файлового введення/виведення. У цій програмі введені із клавіатури рядки зчитуються й записуються у файл. Програма завершується при введенні знаку долара \$ як перший символ рядка. Для використання програми в командному рядку задайте ім'я файлу для виведення.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
    if(argc!=2) { cout << "Введіть <ім'я_файла>\n" ; return 1;}
    ofstream out (argv[1] ) ;          // файл для вивода
    if (!out) { cout << "Файл открыть невозможно\n"; return 1;}
    char str[80] ;
    cout << "Вводите строки; для окончания ввода введите $\n";
    do {
        cout << ": ";
        cin >> str;
        cout <<str << endl;
    } while (*str != '$' ) ;
    out. close () ;
    return 0 ;
}
```

3. У наступній програмі копіюється текстовий файл і при цьому пробіли перетворюються в символи |. Зверніть увагу, як для контролю кінця файлу для введення використовується функція eof(). Також зверніть увагу, як потік введення fin зі скидання прапора skipws. Це запобігає пропуску пробілів на початку рядків.

```
// Превращение пробелов в вертикальные линии |
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
    if(argc!=3)
    { cout << "Преобразование <файл_ввода> <файл_вывода>\n";
```

```

return 1;}
ifstream fin(argv[1] ) ; // открытие файла для ввода
ofstream fout (argv[2] ) ; // создание файла для вывода
if (!out)
{ cout << "Файл открыть невозможно\n"; return 1; }
if (! fin)
{ cout << "Файл открыть невозможно\n"; return 1 ; }
char ch;
fin. unsetf (ios:: skipws) ; // не пропускать пробелы
while (!fin.eof () )
{
cin >> ch;
if (ch==' ') ch = | ' ;
if (!fin.eof () ) cout << ch;
}
fin. close () ; fout . close ( ) ;
return 0;
}

```

4. Між вихідною бібліотекою введення/виведення C++ і бібліотекою введення/виведення сучасного стандарту Standard C++ є деякі відмінності, які необхідно враховувати при модернізації старих програм. По-перше, у вихідній бібліотеці введення/виведення C++ у функції `open()` є третій параметр, що задає режим захисту файлу. За замовчуванням це режим звичайного файлу. У сучасній бібліотеці C++ зазначений параметр не підтримується. По-друге, при роботі зі старою бібліотекою для відкриття потоку введення/виведення `fstream` необхідно явно вказати значення режиму відкриття файлу `ios::in` й `ios::out`. Значення режиму відкриття файлу за замовчуванням не підтримуються. Це ставиться як до конструктора класу `fstream`, так і до функції `open()`. Наприклад, при роботі зі старою бібліотекою введення/виведення C++, щоб відкрити файл для уведення й висновку за допомогою функції `open()`, необхідно використати наступну конструкцію:

```

fstream mystream;
mystream.open("test", ios::in | ios::out);

```

У сучасній бібліотеці введення/виведення C++, якщо режим відкриття файлу не зазначений, будь-який об'єкт типу `fstream` автоматично відкриває файл для уведення й висновку.

І останнє. При роботі зі старою бібліотекою введення/виведення до помилки виконання функції `open()` веде значення режиму відкриття файлу, рівне `ios::noncreate`, якщо зазначений файл не існує, або рівне `ios::noreplace`, якщо, навпаки, зазначений файл уже існує. У стандарті Standard C++ даного значення режиму відкриття файлу не підтримуються.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

1. Пояснити особливості команд відкриття потоків введення, виведення та введення/виведення.
2. Виділити відмінності функції відкриття файлу для трьох потокових класів.
3. Як організувати обробку помилок відкриття файлів?
4. Особливості функції закриття файлу.
5. Функція визначення кінця файлу.
6. Як організувати виведення при роботі з текстовими файлами?

Завдання для самоперевірки засвоєного матеріалу на лекції

1. Напишіть програму для копіювання текстового файлу. Ця програма повинна підраховувати число символів і виводити на екран отриманий результат. Чому це число відрізняється від того, котре виводиться при перегляді списку файлів каталогу?
2. Напишіть програму для заповнення інформацією наступної таблиці у файлі phone.
Исаак Ньютон, 415 555-3423
Роберт Годдард, 213 555-2312
Энрико Ферми, 202 555-1111
3. Напишіть програму для підрахунку числа слів у файлі. Для простоти вважайте, що словом є всі, що має із двох сторін пробіли.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / ІО.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Лекція № 18

з навчальної дисципліни Програмування

Тема Функції роботи з бінарними файлами в стилі C++.
Мета заняття Розглянути роботу та приклади використання функцій введення/виведення, заданих в класі ios.

Час – 2 години

План проведення лекції

- 1 Неформатоване двійкове введення/виведення
- 2 Додаткова інформація про функції двійкового введення/виведення
- 3 Довільний доступ

Навчальні матеріали лекції

- 1 Неформатоване двійкове введення/виведення

Хоча текстові файли (тобто файли, інформація в які представлена в кодах ASCII) корисні в багатьох ситуаціях, у них немає гнучкості неформатованих двійкових файлів. Неформатовані файли містять ті самі вихідні або "сирі" двійкові дані, які безпосередньо використовуються вашою програмою, а не зручний для сприйняття людини текст, дані для якого транслюються операторами << й >>. В C++ для двійкових файлів підтримується широкий діапазон функцій введення/виведення. Ці функції дають можливість точно контролювати процеси зчитування з файлів і запису у файли. На нижньому рівні двійкового введення/виведення перебувають функції `get()` і `put()`. За допомогою функції-члена `put()` можна записати байт; а за допомогою функції-члена `get()` - ввести. Ці функції є членами всіх потокових класів відповідно для введення й для виведення. Функції `get()` і `put()` мають безліч форм. Нижче наведені їх версії, що найчастіше зустрічаються:

```
istream &get(char &символ);  
ostream &put(char символ);
```

Функція `get()` зчитує один символ з пов'язаного з нею потоку й передає його значення аргументу `символ`. Повертає посилання на потік. При зчитуванні символу кінця файлу функція поверне потоку значення `false`. Функція `put()` записує символ у потік і повертає посилання на потік. Для зчитування й запису блоків двійкових даних використовуються функції `read()` і `write()`, які також є членами потокових класів відповідно для введення й для виведення. Тут показані їхні прототипи:

```
istream &read(char *буфер, streamsize число_байт);
```

```
ostream &write(const char *буфер, streamsize число_байт);
```

Функція `read()` зчитує із потоку стільки байтів, скільки задане в аргументі `число_байт` і передає їх у буфер, певний покажчиком `буфер`. Функція `write()` записує у відповідний потік з буфера, що визначений покажчиком `буфер`, задане в `число_байт` число байтів. Значення типу `streamsize` являють собою деяку форму цілого.

Якщо кінець файлу досягнут до того, як було лічене `число_байт` символів, виконання функції `read()` просто припиняється, а в буфері виявляється стільки символів, скільки їх було у файлі. Довідатися, скільки символів було лічено, можна за допомогою іншої функції-члена `gcount()`, прототип якої наведений нижче:

```
streamsize gcount;
```

Функція повертає кількість символів, лічених під час останньої операції двійкового уведення.

Природно, що при використанні функцій, призначених для роботи із двійковими файлами, файли звичайно відкривають у двійковому, а не в текстовому режимі. Зміст цього зрозуміти легко, значення режиму відкриття файлу `ios::binary` запобігає як б не було перетворення символів. Це важливо, коли у файлі зберігаються двійкові дані, наприклад, цілі, дійсні або покажчики. Проте, для файлу, відкритого в текстовому режимі, хоча в ньому втримується тільки текст, двійкові функції також цілком доступні, але при цьому пам'ятайте про можливість небажаного перетворення символів.

Приклади.

1 У наведеній нижче програмі для запису рядка й числа типу `double` у файл `test` використовується функція `write()`:

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main ( )
{
    ofstream out ("test", ios::out | ios::binary) ;
    if (!out) { cout << "Файл открытъ невозможно \n"; return 1;}
    double num = 100.45;
    char str[] = "Это проверка";
    out.write((char *) &num, sizeof(double));
    out.write(str, strlen(str));
    out.close();
    return 0;
}
```

Приведення типу до (char*) при виклику функції write() необхідно, якщо буфер виведення не визначений як символьний масив. Оскільки в C++ здійснюється строгий контроль типів, покажчик на один тип не перетвориться автоматично в покажчик на інший тип.

2. У наступній програмі для зчитування з файлу, створеного в програмі прикладу 1, використається функція read():

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("test", ios::in | ios::binary);
    if(!in) { cout << "Файл открытъ невозможно\n"; return 1;}
    double num;
    char str[80];
    in.read((char *) &num, sizeof(double));
    in.read(str, 15) ;
    str[14] = '\0';
    cout << num << ' ' << str;
    in.close();
    return 0;
}
```

Як й у програмі з попереднього прикладу, приведення типів усередині функції read() необхідно, оскільки в C++ покажчик одного типу автоматично не перетвориться в покажчик іншого типу.

2 Додаткова інформація про функції двійкового введення/виведення

Крім представленої раніше форми, функцію get() можна перевантажити ще декількома способами. Тут показані прототипи трьох форм, що найчастіше перевантажують:

```
istream &get (char *буфер, streamsize число_байт) ;
istream &get( char *буфер, streamsize число_байт, char обмежник) ;
int get() ;
```

Перша функція get() зчитує символи в масив, заданий покажчиком буфер, доти, поки або не зчитано стільки символів, скільки задано параметром число_байт-1, або не зустрівся символ кінця файлу. Наприкінці масиву, заданого покажчиком буфер, функція get() поміщає нуль. Якщо в потоці введення зустрінеться символ нового рядка, він не витягається, а залишається в потоці до наступної операції введення.

Друга функція `get()` зчитує символи в масив, заданий покажчиком буфер, доти, поки або не зчитано стільки символів, скільки задано параметром `число_байт-1`, або не зустрівся символ, заданий параметром обмежник, або не зустрівся символ кінця файлу. Наприкінці масиву, заданого покажчиком буфер, функція `get()` поміщає нуль. Якщо в потоці введення зустрінеться символ обмежник, він не витягається, а залишається в потоці до наступної операції введення.

Третя функція `get()` повертає з потоку наступний символ. Вона повертає символ EOF, якщо досягнуто кінець файлу. Ця форма функції `get()` нагадує функцію `getc()` мови C.

Іншою функцією для реалізації введення є функція `getline()`. Ця функція - член всіх потокових класів введення. Нижче показані її прототипи:

```
istream &getline(char *буфер, streamsize число_байт);  
istream &getline(char *буфер, streamsize число_байт, char обмежник);
```

Перша функція зчитує символи в масив, позначений покажчиком буфер, доти, поки або не зчитано стільки символів, скільки задано параметром `число_байт - 1`, або не зустрівся символ нового рядка, або не зустрівся символ кінця файлу. Наприкінці масиву, заданого покажчиком буфер, функція `getline()` поміщає нуль. Якщо в потоці введення зустрінеться символ нового рядка, він витягається, але не міститься в масив.

Друга функція зчитує символи в масив, позначений покажчиком буфер, доти, поки або не зчитано стільки символів, скільки задано параметром `число_байт - 1`, або не зустрівся символ обмежник, або не зустрівся символ кінця файлу. Наприкінці масиву, заданого покажчиком буфер, функція `getline()` поміщає нуль. Якщо в потоці введення зустрінеться символ обмежник, він витягається, але не міститься в масив.

Як можна помітити, обидві версії функції `getline()` фактично тотожні версіям `get(буфер, число_байт)` і `get(.(буфер, число_байт, обмежник)` функції `get()`. Обидві зчитують символи з потоку введення й поміщають їх у масив, позначений покажчиком буфер доти, поки або зчитано `число_байт-1` символів, або не зустрівся символ обмежник або символ кінця файлу. Відмінність між функціями `get()` і `getline()` у тім, що функція `getline()` зчитує й видаляє з потоку введення символ обмежник, а функція `get()` - ні.

Використовуючи функцію `peek()`, можна одержати наступний символ з потоку введення без його видалення з потоку. Функція є членом потокових класів введення й має наступний прототип:

```
int peek ();
```

Функція повертає наступний символ з потоку, якщо досягнуто кінець файлу або символ EOF.

За допомогою функції `putback()`, що є членом потокових класів введення, можна повернути останній зчитаний з потоку символ назад у потік. Нижче показаний прототип цієї функції:

```
istream fiputback (char c);
```

Тут `c` - це останній зчитаний з потоку символ.

При виконанні виведення дані не відразу записуються на пов'язаний з потоком фізичний пристрій, а інформація тимчасово зберігається у внутрішньому буфері. Тільки після заповнення буфера його вміст пишеться на диск. Однак виклик функції `flush()` викликає фізичний запис інформації на диск до заповнення буфера. Нижче показаний прототип функції `flush()`, що є членом потокових класів виводу:

```
ostream &flush();
```

Виклики функції `flush()` виправдані при роботі в несприятливих умовах (наприклад, у ситуаціях, коли часто трапляються збої електропостачання).

Приклади:

1 Як ви знаєте, при використанні для зчитування рядка оператора `>>`, зчитування припиняється при зустрічі першого розділового символу. При наявності в рядку пробілів таке зчитування стає неможливим. Однак, як показано в програмі, за допомогою функції `getline()` можна вирішити цю проблему:

```
// Использование функции getline()
//для считывания строки с пробелами
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char str[80] ;
    cout <<"Введите ваше имя: "; cin.getline (str, 79);
    cout << str << '\n' ;
    return 0;
}
```

У цьому випадку обмежником для функції `getline()` є символ нового рядка. Це робить виконання функції `getline()` дуже схожим на виконання стандартної функції `gets()`.

2. У реальному програмуванні особливо корисні функції `peek()` і `putback()`. Вони дозволяють спростити керування, коли невідомий тип вводимі інформації. Наступна програма ілюструє це. У ній з файлу зчитуються рядки або цілі. Рядки й цілі можуть бути розміщені в будь-якому порядку.

```
// Демонстрация работы функции peek()
#include <iostream>
```

```

#include <fstream>
# include <cctype>
using namespace std;
int main ( )
{
char ch;
of stream out ("test", ios::out | ios:: binary );
if (!out) { cout << "Файл открыть невозможно\n"; return 1;}
char str [80] , *p;
cout << 123 << "this is a test" << 23;
cout << "Hello there!" << 99 << "sdf" << endl;
out. close() ;
ifstream in ("test", ios:: in | ios :: binary );
if (!in) { cout << "Файл открыть невозможно\n"; return 1;}
do
{
p = str;
ch = in. peek ();
// выяснение типа следующего символа
if (isdigit(ch) )
{ while (isdigit (*p=in.get ( ) ) ) p++; // считывание целого
in.putback (*p) ; // возврат символа в поток
*p = '\0'; // заканчиваем строку нулем
cout << "Целое: " <<atoi(str);
}
else
if (isalpha(ch) )
{ while (isalpha (*p=in.get ( ) ) ) p++;// считывание строки
in.putback(*p) ; // возврат символа в поток
*p = '\0'; // заканчиваем строку нулем
cout << "Строка: " << str;
}
else in. get (); // пропуск
cout << ' \n ' ;
} while ( lin.eof ( ) ) ;
in. close ( ) ; return 0;
}

```

3 Довільний доступ

У системі введення/виведення C++ *довільний доступ (random access)* реалізується за допомогою функцій `seekg()` і `seekr()`, що є відповідно потоковими функціями введення й виведення. Тут показані їхні основні форми:

```
istream &seekg(off_type зсув, seekdir завдання);
```

```
ostream &seekp(off_type зсув, seekdir завдання);
```

Тут `off_type` - це цілий тип даних, певний у класі `ios` і сумісний з максимальним правильним значенням, що здатний зберігати параметр зсув. Тип `seekdir` - це перерахування, певне в класі `ios` й утримуючі наступні значення:

<code>ios::beg</code>	Пошук з початку файлу
<code>ios::cur</code>	Пошук від поточної позиції у файлі
<code>ios::end</code>	Пошук з кінця файлу

Система введення/виведення C++ управляє двома покажчиками, пов'язаними з файлом. Перший - це *покажчик зчитування* (*get pointer*), що задає наступне місце у файлі, звідки буде вводитися інформація. Другий - це *покажчик запису* (*put pointer*), що задає наступне місце у файлі, куди буде виводитися інформація. При кожному введенні або виведенні відповідний покажчик послідовно просувається далі. Однак за допомогою функцій `seekg()` і `seekp()` можливий непослідовний доступ до файлу. Функція `seekg()` установлює покажчик зчитування відповідного файлу в позицію, що відстоїть на величину зсув від заданого місця завдання. Функція `seekp()` установлює покажчик запису відповідного файлу в позицію, що відстоїть на величину зсув від заданого місця завдання.

Як правило, файли, доступні для функцій `seekg()` і `seekp()`, повинні відкриватися в режимі операцій для двійкових файлів. У такий спосіб запобігає можливе несподіване перетворення символів усередині файлу. Визначити поточну позицію кожного із двох покажчиків можна за допомогою функцій:

```
pos_type tellg();  
pos_type tellp();
```

Тут `pos_type` - це цілий тип даних, певний у класі `ios` і здатний зберігати найбільше можливе значення покажчика. Для переміщення файлових покажчиків зчитування й записи на позицію, задану повертають значеннями, що, функцій `tellg()` і `tellp()`, використовуються перевантажені версії функцій `seekg()` і `seekp()`. Прототипи цих функцій представлені нижче:

```
istream fseekg(pos_type позиція);  
ostream fseekp(pos_type позиція);
```

Приклади

1. У наступній програмі показана робота функції `seekp()`. Вона дозволяє замінити у файлі заданий символ. Укажіть у командному рядку ім'я файлу, потім номер байту у файлі, що ви хочете змінити, і, нарешті, новий символ для заміни. Зверніть увагу: файл відкривається для операцій читання й запису.

```
#include <iostream>  
#include <fstream>
```

```

#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
if (argc! =4) { cout << "Замена: <файл> <байт> <символ>\n"; return
1;}
fstream out(argv[1], ios:: in | ios:: out | ios :: binary );
if (!out) { cout << "Файл открыть невозможно\n"; return 1 ;}
out.seekp(atoi(argv[2]), ios::beg);
out.put(*argv[3]);
out.close();
return 0;
}

```

2. У наступній програмі функція seekg() використається для установки покажчика зчитування в задану позицію усередині файлу й для висновку вмісту файлу, починаючи із цієї позиції. Ім'я файлу й позиція початку зчитування задаються в командному рядку.

```

// Демонстрация работы функции seekg()
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
char ch;
. if (argc! =3) { cout << "Поиск: <файл> <позиция>\n"; return 1; }
ifstream in(argv[1], ios::in | ios : binary );
if(!in) { cout << "Файл открыть невозможно\n"; return 1; }
in.seekg(atoi (argv[2] ) , ios: :beg) ;
while(!in.eof () ) { in. get (ch) ; cout « ch;}
in. close () ;
return 0;
}

```

Питання та завдання до контролю знань студентів

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Дано наступний клас:

```

class account {
int custnum;
char name[80]; double balance;
public:
account(int c, char *n, double b)

```

```
{ custnum = c; strcpy(name, n); balance = b; }  
// здесь нужна пользовательская функция вывода  
}
```

2 Напишіть програму для висновку вмісту класу у файл. Для цієї мети створіть користувальницьку функцію висновку.

3 Напишіть програму для порядкового зчитування текстового файлу й висновку кожного ліченого рядка на екран. Використайте функцію `getline()`. Подумайте про ситуації, у яких може виявитися корисним виклик функції `flush()`.

4 Напишіть програму для висновку на екран умісту текстового файлу у зворотному порядку. (Підказка: Обміркуйте завдання перед початком програмування. Рішення простіше, ніж може здатися на перший погляд.)

5 Напишіть програму, що попарно міняє місцями символи в текстовому файлі. Наприклад, якщо у файлі втримується "1234", те після виконання програми там повинне втримуватися "2143". (Для простоти вважайте, що у файлі втримується парне число символів.)

ЛІТЕРАТУРА

1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.

2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>