Лекція 1. Основи програмування на С++

План

1. Класифікація мов програмування.
2. Основні алгоритмічні конструкції.
3. Етапи розв’язання обчислювальних задач на комп’ютері.
4. Алфавіт мови програмування.
5. Структура програми.
6. Умовні оператори.
7. Оператори циклу.
8. Оператори передачі управління.

1. Класифікація мов програмування

Класифікація за парадигмою програмування.

Імперативні мови програмування орієнтовані на оператори. Обчислення в них представляються як послідовність операторів (команд) (Fortran, Algo, Cobol, C\C++).

Функціональні мови програмування задають процес обчислень як послідовність функцій (Lisp, Haskell).

Логічні мови програмування описують обчислення за допомогою формальної логіки (Prolog).

Об’єктно-орієнтовані мови програмування – обчислення реалізуються у вигляді сукупності об’єктів (C#, Java).

Класифікація за видом трансляції.

Мови програмування, що **інтерпретуються**.

Мови програмування, що **компілюються**.

Скриптові мови програмування (Perl, JavaScript, PHP)

Мови програмування низького рівня – це мови програмування, кожна команда яких має відповідність до команд процесора (мова Асемблер, машині коди). Протилежністю мовам програмування низького рівня є мови програмування високого рівня.

Розглянемо, як із програми, написаної мовою високого рівня, утворюється інша – машинна. Вихідний текст програми за допомогою спеціальної програми (текстовий редактор) записують на диск у вигляді вихідного файла. Програма може складатися з кількох вихідних файлів – у великих програмах їх може нараховуватися десятки.

Під час роботи транслятора прочитується вихідний файл і створюється його машинний еквівалент – об’єктний код. Процес виконання програми-транслятора називається трансляцією, або компіляцією вихідного тексту.

Як правило, об’єктний код програми містить далеко не всі необхідні команди – програма може складатися з частин або включати підпрограми з бібліотек. Об'єктний код обробляється це однією програмою – редактором зв’язків, або компонувальником, яка «збирає» (компонує) повний код програми і записує (завантажує) його або в оперативну пам’ять, або на диск у вигляді готового до виконання файла, який можна завантажити пізніше.

Інтерпретатор на відміну від транслятора не створює машинну програму. Вхідні дані для інтерпретатора – це високорівнева програма й дані, що мають зчитуватися під час її виконання. Інтерпретація програми полягає в тому, що дії, задані програмою, відразу виконуються. Зазвичай інтерпретація вихідної програми відбувається повільніше, ніж виконання відповідної машинної програми.

Ще один спосіб обробки вихідної програми поєднує в собі трансляцію й інтерпретацію. Програма перекладається (транслюється) не в машинні команди, а в деяке проміжне зображення, що потім інтерпретується. Такий підхід реалізовано, зокрема, в мові Java.

Інтерпретація програми здійснюється за допомогою такого інструменту, як налагоджувач. Він забезпечує інтерпретацію вихідної програми невеликими порціями (кроками) і дає можливість побачити результати виконання кожного кроку. Це полегшує пошук помилки у вихідній програмі.

Описані засоби (текстовий редактор, транслятор та\або інтерпретатор, компонувальник, завантажувач та налагоджувач) разом утворюють систему програмування, або інтегроване середовище розробки.

2. Основні алгоритмічні конструкції

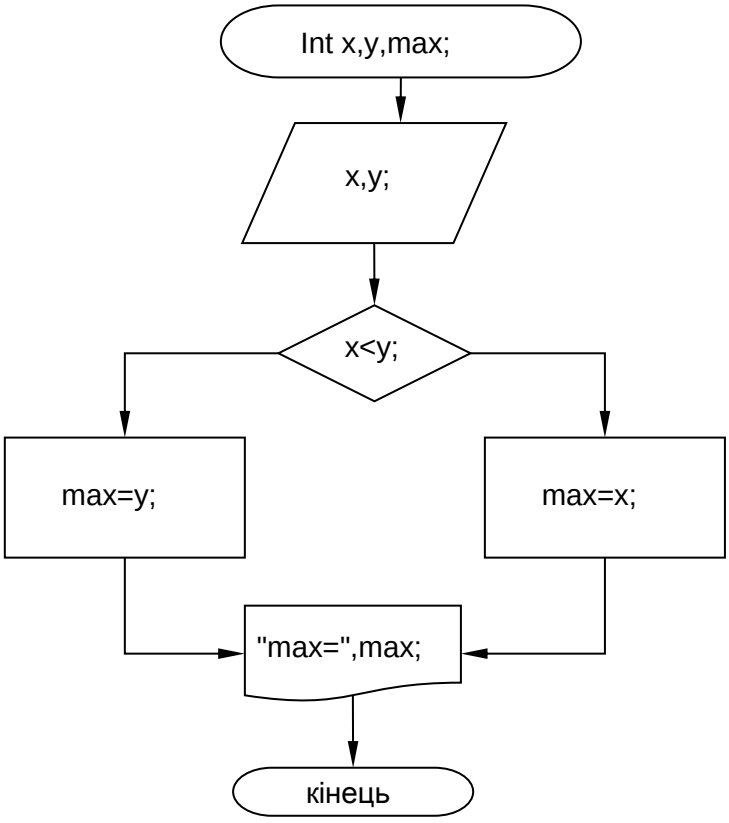
Способи запису алгоритмів:

- Словесний опис.
- Графічний опис (блок-схема).
- Псевдокод.

Структурна (блок) схема алгоритму – це графічне зображення алгоритму у вигляді схеми – пов’язаних між собою за допомогою стрілок (ліній переходу) блоків – графічних символів, кожний з яких відповідає одному кроку алгоритму. Усередині блоку дається опис відповідної дії.

Геометричне подання	Призначення оператора	Геометричне подання	Призначення оператора
	Процес (арифметичний оператор)		Розв'язок (умовний оператор)
	Дані (оператори введення-виведення)		Початок-кінець (оператор зупинки або початок блок-схеми)
	Документ (друк на бумагу)		Монітор (виведення на монітор)
	Сортування		Збереження даних
	Пам'ять з послідовним доступом		Пам'ять з прямим доступом

Наприклад, блок-схема знаходження максимального з двох значень, має такий вигляд:



Псевдокод – це неформальний запис алгоритму, що використовує структуру поширених мов програмування, але нехтує деталями коду, неістотними для розуміння алгоритму (опис типів, виклик підпрограм тощо). Мова програмування доповнюється природною мовою, компактними математичними позначеннями. Псевдокод є більш зрозумілою, ніж програми, формою запису алгоритмів.

```
Repeat
     Compute EMA(T)
     If no position opened
        If EMA(T) >= P
              If trend is going up
                    Open a long position
              Else if trend is going down
                    Open a short position
       Else if any position is opened
       If EMA(¬T) >= Q
              Close position
     If end of market
              Close all opened position
Until market close
```

2. Основні алгоритмічні конструкції

ЛІНІЙНІ АЛГОРИТМИ

Найпростішими для алгоритмізації є задачі, у яких перетворення інформації відбувається послідовно за певними формулами, що розкладаються на елементарні операції. Кожна дія виконується одна за одною, послідовно, у порядку розташування блоків, і при цьому жодна з дій не пропускається й не повторюється. Лінійні алгоритми не містять перевірок умов. Потрібно тільки визначити раціональну послідовність цих операцій і виконати їх за схемою, що наведена нижче.

Приклад 1: $Z=\sin^2(x^2+y^2) + \cos^2 \ln(x^2+y^2)$

Оскільки аргументом функцій sin та ln є вираз x^2+y^2, його треба обчислити його у першу чергу. Отже, алгоритм визначення Z матиме вигляд, поданий на рис:

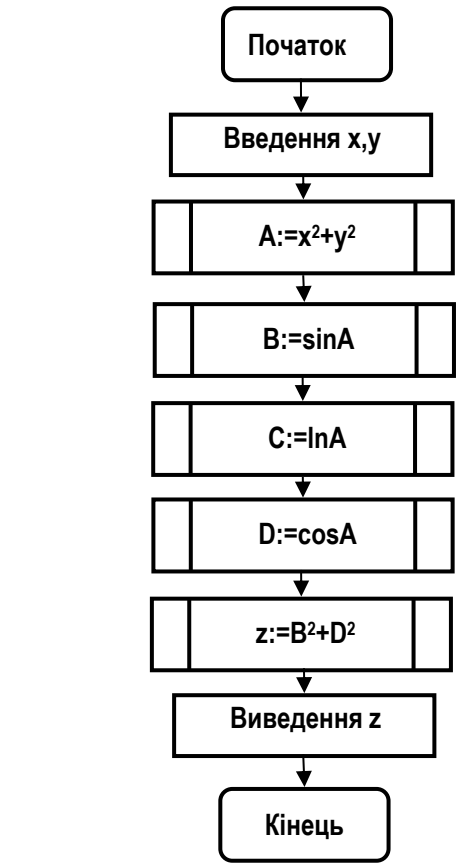
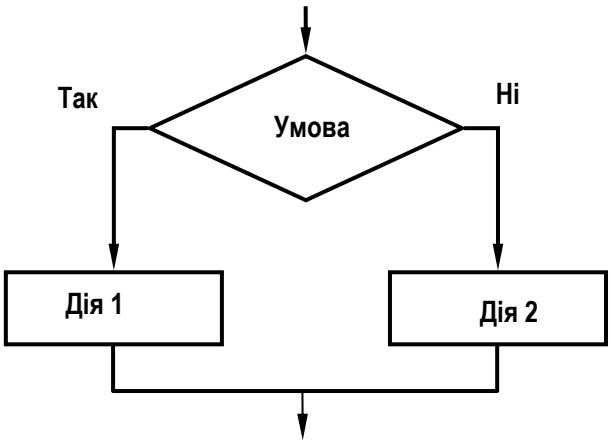


Рис. Приклад лінійного алгоритму

РОЗГАЛУЖЕНІ АЛГОРИТМИ

У тих випадках, коли перетворення інформації може здійснюватися за різними схемами, залежно від властивостей вхідних даних або проміжних результатів, використовуються розгалужені алгоритми.

Структура "розгалуження" передбачає виконання однієї з двох груп дій залежно від виконання умови у блоці розгалуження. На рис. знаком "Так" показано виконання умови, а знаком "Ні" – її невиконання. Часто використовується неповна команда розгалуження, коли один із блоків дії відсутній.



В алгоритмі розгалуження передбачаються усі можливі варіанти обробки інформації, кожний з яких розробляється як окрема гілка алгоритму, а вибір однієї з них для виконання здійснюється за допомогою перевірки певної умови, яка відображає властивості інформації, що використовуються у процесі перетворення. При цьому деякі дії можуть взагалі не виконуватися (пропускатися). В алгоритмічних системах для цього є спеціальні блоки-розпізнавачі.

Залежно від того, на яку кількість гілок розгалужується алгоритм, він може бути простим або складним. Для простого розгалуженого процесу перевіряється одна умова (один розпізнавач), для складного – дві чи більше умов, кожна з яких виокремлює одну гілку.

Просте розгалуження відбувається за схемою, поданою на рис. 3.5.

Умова формулюється так, щоб відповідь перевірки була "так" чи "ні".

Проста умова містить два вирази (значення), що поєднуються знаком операції відношення:

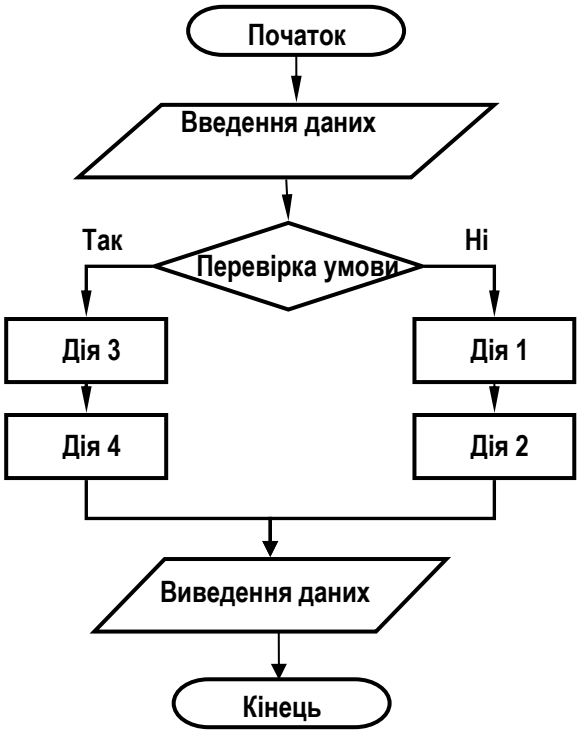
> більше за ...;

< менше за ...;

≥ більше або дорівнює...;

≤ менше або дорівнює...;

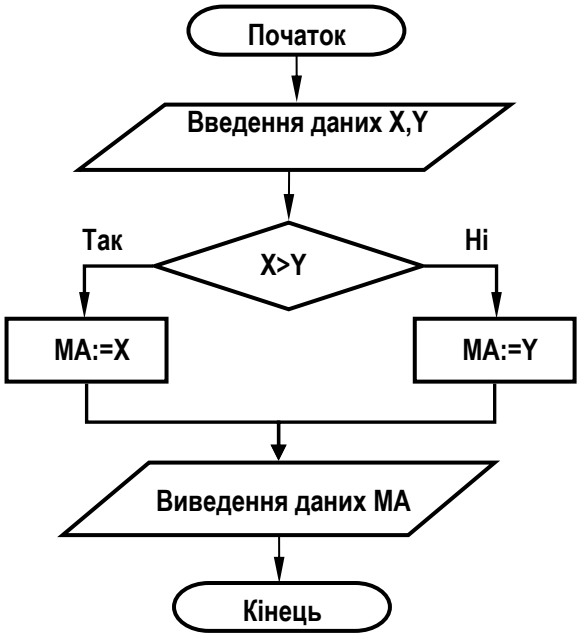
≠ не дорівнює...



Результатом перевірки умови є логічний вираз **ІСТИНА**, якщо умова виконується, або **ХИБНІСТЬ**, якщо умова не виконується.

Приклад 2: Дано дійсні числа x та y. Обчислити max(x, y).

Алгоритм розв’язання представлений на рис. 3.6



Складна умова містить дві або більше простих умов, поєднаних знаками логічних операцій:

І – усі вказані умови мають виконуватися одночасно, тобто результатом усіх включених простих умов має бути **ІСТИНА**. При цьому операція **І** дасть результат **ІСТИНА,** а якщо хоча б одна з перелічених умов має результат **ХИБНІСТЬ**, операція **І** дасть результат **ХИБНІСТЬ.**

АБО – деякі з указаних умов можуть виконуватися, а деякі – ні. Якщо жодна з перелічених умов не виконується, то результатом операції **АБО** буде **ХИБНІСТЬ**, у всіх інших випадках – результат **ІСТИНА.**

НЕ – заперечення для умови.

Складне розгалуження відбувається послідовно, з відокремленням гілок.

Розгалужень може бути скільки завгодно. Збільшення кількості умов робить алгоритм більш складним і заплутаним, він втрачає наочність, перевірити його правильність досить складно. У таких випадках необхідно перехід до будь-якої гілки розгалуженого алгоритму пов'язати з деякою змінною, кожне значення якої відповідатиме одній з гілок розгалуження, тобто одному з варіантів обробки інформації. Це можуть бути не тільки окремі значення, а й проміжки значень, до яких належатиме конкретне значення змінної. Тоді всі логічні блоки алгоритму об'єднуються в один блок аналізу цієї змінної, який матиме не два входи, а стільки, скільки існує варіантів обробки.

ЦИКЛІЧНІ АЛГОРИТМИ

Циклом називають повторення послідовної кількості кроків алгоритму. Обчислювальний процес, що містить цикл, називається циклічним. Керування повторенням циклу здійснюється за допомогою змінної, яка називається параметром циклу. Спочатку цьому параметру надають певне початкове значення. Потім цикл виконується зміною параметра. При кожному повторенні від початкового до кінцевого значення збільшується на величину, що називається кроком циклу. Крок циклу може бути додатним або від'ємним. Залежно від цього параметр циклу зростає або зменшується. Цикл припиняється, якщо значення параметра вийде за межі діапазону між початковими і кінцевим значеннями.

Комбінуючи базові структури між собою, можна відтворювати алгоритм, що реалізує складний обчислювальний процес.

Структурна побудова алгоритму включає:

– використання методу покрокової деталізації;

– використання на кожному із зазначених кроків трьох перелічених базових структур;

– аналіз створеного алгоритму – методу ручної "прокрутки" – перевірка правильності функціонування створеного алгоритму шляхом підстановки вхідних значень і перегляду роботи алгоритму вручну.

Розрізняють такі види циклів:

– цикл із передумовою (спочатку умова, потім дія);

– цикл із постумовою (спочатку дія, потім умова);

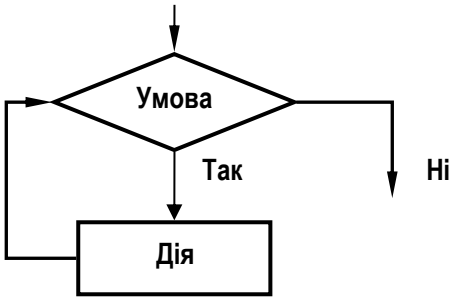
– цикл із параметром.

Перші два види циклів використовуються у випадках, коли кількість повторень заздалегідь невідома.

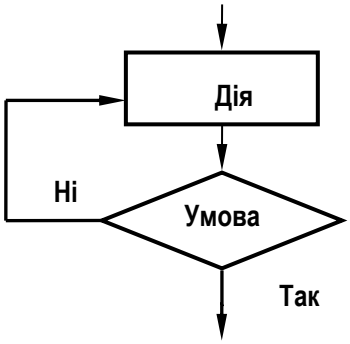
При кожному черговому виконанні циклу перевіряється умова на продовження роботи, і якщо умова набуває результату **ІСТИНА**, цикл виконується, а якщо умова набуває результату **ХИБНІСТЬ** – цикл не виконується.

Цикл з передумовою характеризується тим, що спочатку перевіряється умова (звідси й назва – цикл з передумовою). Якщо умова виконується, то виконується дія. Потім знову перевіряється умова і т. д. Виконання циклу

припиняється, коли умова перестає виконуватися. Для цього необхідно, щоб дія в циклі впливала на зміну умови. В іншому разі відбудеться "зациклювання" – нескінченне виконання циклу.

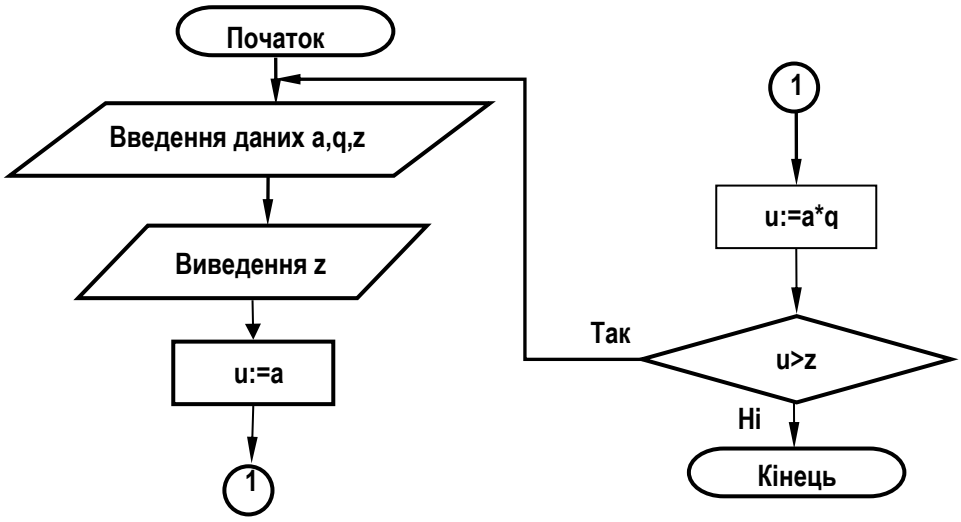


Цикл з постумовою характеризується тим, що спочатку виконується дія, а потім перевіряється умова. Повторення дії відбувається в тому випадку, якщо умова не виконується.



Дія в циклі з постумовою виконується завжди хоча б один раз, а з передумовою вона може не виконуватися жодного разу, якщо із самого початку умова не виконується.

Приклад. Задана спадна геометрична прогресія із першим членом a і знаменником b. Визначити всі члени цієї прогресії, значення яких більше за z. Блок-схема алгоритму обчислення членів спадної прогресії зображена на рис. 3.10.

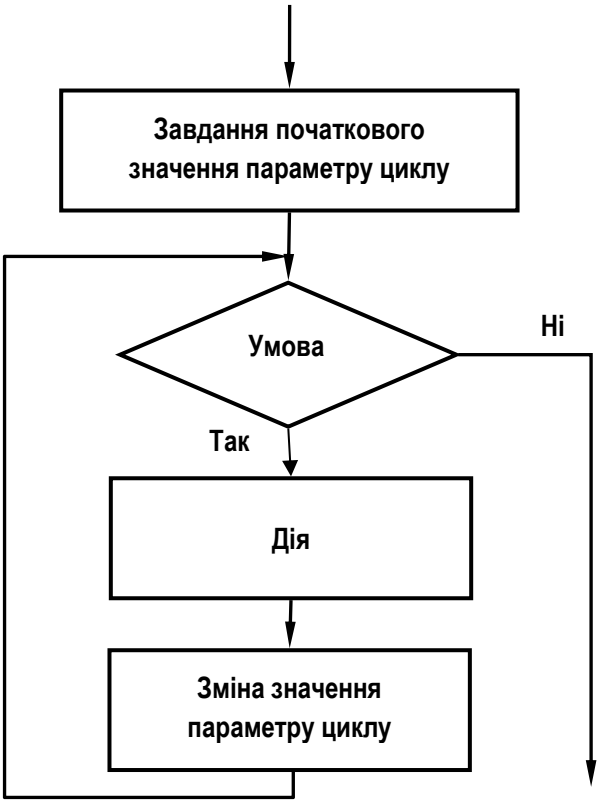


<u>Цикл із параметром.</u> Особливістю цього циклу є те, що кількість повторень циклу заздалегідь відома. У зв'язку з цим цикл буде виконуватися, доки значення змінної, яка називається параметром циклу, не перевищить зазначену кількість повторень.

Спочатку параметру циклу присвоюється певне початкове значення. Потім цикл виконується зі змінною параметра при кожному повторенні від початкового до кінцевого значень на величину, яка називається кроком циклу. Цикл припиняється, якщо параметр циклу здобуває значення, яке знаходиться поза межами діапазону між початковим і кінцевим значеннями.

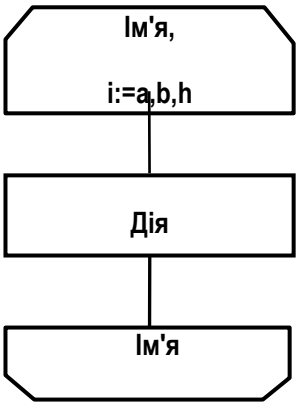
Крок циклу може бути додатним або від'ємним. Залежно від цього параметр циклу зростає або зменшується.

Цикл із параметром будується на основі одного з перших двох циклів. У більшості випадків він використовує цикл з передумовою:



У зв’язку з широким використанням циклів для них використовується спеціальний символ "**Межа циклу**", який складається з двох частин: початку і кінця циклу.

У цих символах вказують ім’я циклу з параметром – початкове й кінцеве значення, а також крок циклу.



тут Ім’я – назва циклу;

i – параметр циклу;

a – початкове значення параметра циклу;

b – кінцеве значення параметру циклу;

h – крок циклу.

Якщо h = 1, то його можна не вказувати.

При організації циклічних обчислювальних процесів часто виникає необхідність перебирати значення не однієї, а декількох змінних. У цьому випадку йде мова про зовнішній і внутрішні цикли. Для кожного значення параметра в зовнішньому циклі відбувається багаторазове виконання дій у внутрішньому циклі, який називається вкладеним.

3. Етапи розв'язання обчислювальних задач на комп'ютері

Хоча існують і використовуються сотні тисяч різноманітних прикладних програм, користувачі самі мають можливість розробляти власні програми для розв'язування тих чи інших задач на комп'ютері. Процес розробляння програми може бути подано як послідовність таких кроків:

1) постановка задачі – окреслення вимог щодо програми (приміром, визначення початкових даних, їхнього формату, параметрів введення). Результатом цього етапу розробляння є докладний словесний чи математичний опис алгоритму;

2) побудова математичної моделі задачі – опис задачі за допомогою математичних формул, визначення переліку початкових даних та шуканих результатів, вихідні умови, точність обчислень;

3) вибір методу розв'язування, оскільки одну й ту саму задачу може бути розв'язано за допомогою різних методів. Вибір методу має визначатися багатьма чинниками, основними з яких є точність результатів, час розв'язування, обсяг займаної оперативної пам'яті. У кожному конкретному випадку за критерій вибору методу беруть один зі згаданих критеріїв;

4) розробляння схеми алгоритму розв'язування задачі, тобто оптимальної логічної послідовності дій з урахуванням обраного методу розв'язування для здобуття певних результатів;

5) написання та введення розробленої програми алгоритмічною мовою програмування до комп'ютера;

6) налагодження програми, тобто пошук і виправлення можливих синтаксичних та алгоритмічних помилок у програмі;

7) тестування – перевірка правильності роботи програми.

4. Алфавіт і словник мови програмування

У будь-якій мові програмування програма – це набір зрозумілих компілятору команд. Для створення програм треба знати синтаксис мови, тобто правила запису команд і використання лексичних одиниць мови.

Алфавіт мови програмування – це скінченний набір символів. За допомогою цих символів можуть бути записані ідентифікатори, вирази та оператори мови.

Символи алфавіту мови C\C++ можна поділити на такі категорії:

- символи, що використовуються для складання ідентифікаторів (малі латинські символи, великі латинські символи, десяткові цифри від 0 до 9, символ підкреслення);
- розділовий символ пробілу;

- спеціальні символи, які використовуються в процесі побудови конструкцій мови (+ - * / = > < . , ; : ‘ ( ) [ ] { } ^ @ # $ & | %).

Із символів алфавіту складаються лексичні одиниці мови, або **лексеми** – мінімальні значущі одиниці в текстах програм. Множина всіх допустимих лексем називається **словником** мови програмування.

У мові C\C++ розрізняють такі **види лексем**: спеціальні символи, зарезервовані (ключові) слова, ідентифікатори, неіменовані константи, коментарі та директиви препроцесора.

Лексеми спеціальних символів, окрім спеціальних символів з алфавіту мови, містять ще складені спеціальні символи, що сприймаються компілятором як єдине ціле (<=, >=, +=, *=, ++, ==, -- і т.д.).

Зарезервовані (ключові) слова мають строго визначений зміст. Їх призначення не може змінюватися. Зарезервовані слова використовуються для позначення алгоритмічних конструкцій (циклів, умов, операторів), типів змінних тощо.

Ідентифікатор – це ім’я, значення якого може варіюватися від програми до програми або навіть у межах однієї програми. У мові C\C++ розрізняють стандартні ідентифікатори та ідентифікатори користувача.

Стандартними ідентифікаторами є імена вбудованих у мову функцій (cin, cout, sin, cos тощо), типів даних (int, float, char тощо) і специфікаторів (inline, static, virtual тощо).

Ідентифікатор користувача – це ім’я, яке обирає програміст для позначення (ідентифікації) елементів програми (констант, змінних, типів, функцій). Існують **правила вибору ідентифікаторів**:

- ідентифікатор починається буквою або символом підкреслення;
- ідентифікатор може складатися з букв, цифр, символу підкреслення;
- ідентифікатор може мати довільну довжину, але значущими є тільки перші 63 символи;
- в ідентифікаторі неприпустимо використовувати символи пробілу, крапки та інші символи пунктуації;
- малі та великі літери в ідентифікаторах розрізняються;
- зарезервовані слова не можуть використовуватись як ідентифікатори.

Існують також **рекомендації щодо використання імен у програмах**:

- використовуйте мнемоничні ідентифікатори (такі, що легко запам’ятовуються);
- використовуйте довгі ідентифікатори, що складаються з декількох слів, кожне з яких починається з великої літери, наприклад: CompGraphics, MnemonicCode$;
- замість транслітерації українських і російських слів бажано використовувати їх переклади англійською мовою.

Дотримання цих рекомендацій, нарівні із застосуванням коментарів, зробить текст програми більш зрозумілим.

Коментар – це фрагмент тексту програми, який записується між лексемами /* та */ або після лексеми //. Коментарі ігноруються компілятором та не впливають на роботу програми, але суттєво полегшують її розуміння.

Директива препроцесора – це рядок, що починається символом #. Директиви препроцесора визначають режими компіляції і можуть істотно впливати на зміст згенерованого компілятором машинного коду.

5. Структура програми

Записана мовами C\C++ програма складається з трьох частин:

- директив препроцесора;
- декларативної (оголошення ідентифікаторів, що використовуються у програмі);
- функцій (іменованих частин програми, які містять запис дій, що виконуються).

Декларативна частина програми, що містить глобальні оголошення, передує функціям, кількість яких визначає розробник.

Самі функції теж містять декларації та оператори мов C\C++. Одна з функцій обов'язково має ім'я main() і є точкою входу до програми. Після імені функції обов'язково записують круглі дужки, оскільки в них можуть оголошуватися параметри, що їй передаються. Механізм передачі параметрів у функцію розглядатиметься у подальших лабораторних. Декларації змінних можуть розміщуватися в будь-якому місці програми, де це не заперечено синтаксисом мови. Головне, що слід урахувати розробнику програм, декларації повинні передувати операторам, у яких вони застосовуються. Програма не має оператора, що позначає її ім'я, проте гарним стилем програмування вважається використання коментарів, що описують назву та мету програми.

Приклад 1. Введення двох чисел та обчислення їх суми.

```
// example 1.cpp ввести два числа та визначити їх суму
#include "stdafx.h"                    // Включення заголовних файлів
#include <iostream>
using namespace std;                   // Визначення простору імен
int a;                                 // Оголошення змінних
float b;

void input()                           // Визначення функції
{
    cout << "Enter int a, float b" << endl;    // Вивід строки з запрошенням вводу даних
    cin >> a >> b;                             // Ввід даних

}

int main()
{
```

```
	cout << "Hello, world"<<endl;			// Вивести повідомлення

	input();						// Виклик функції
	cout << "a+b= " << a + b << endl;	// Вивід строки з результатом обчислень
	return 0;
}
```

Пояснімо код програми прикладу 1. На початку програми міститься коментар з назвою та метою створення програми. Він не є обов'язковим, його можна опустити. Він використовується з метою швидкої ідентифікації потрібної програми поміж інших.

Далі записана **директива препроцесора**. Директиви визначають режими роботи компілятора на стадії препроцесорної обробки. У мовах C\C++ означені різні директиви. В прикладах, що розглядатимуться далі, найчастіше застосовується директива #include. Директива включення вказує препроцесору, що потрібно додати вміст іншого файла у вихідний файл в точку програми, де директива згадується. Заголовні файли, що використовуються у програмах, написаних мовою C++, не мають розширення, наприклад, файл iostream. У директиві #include розширення .h вказується для тих заголовних файлів, які успадковані від мови C, наприклад, stdio.h.

Синтаксис оголошення директиви включення заголовних файлів має вигляд:

#include <ім'я заголовного файла>

#include "ім'я заголовного файла"

Імена заголовних файлів, що входять до стандартних бібліотек мов C та C++ середовища Visual Studio, записуються у кутових дужках < >, імена заголовних файлів, які створює користувач, записуються у лапках.

Програма має багато ідентифікаторів, які визначені в різних областях дії. Іноді змінна однієї області дії конфліктує із змінною з тим самим ім'ям в іншій області дії. Для запобігання конфлікту імен оголошують простір імен, який визначає область дії імен. У межах свого простору імен ідентифікатори можуть викликатися саме так, як вони оголошені. Поза свого простору імен ідентифікатори невідомі. Використання імен поза простору імен, в якому вони визначені, можливо із застосуванням імені простору імен і оператора :: розширення області дії.

Для розширення області видимості імен у програмі оголошують директиву using. Синтаксис директиви оголошення простору імен такий:

using namespace <ім'я простору імен>;

Багато імен належать простору std, наприклад cout, cin. В програмах на C++ застосування директиви

using namespace <std>;

звільняє розробника від дописування імені std до імені відповідного оператора, наприклад,

std::cout<<”Hello, world”;

6. Типи даних

Тип даних визначає:

- множину допустимих значень, яких може набувати змінна або константа зазначеного типу;
- множину допустимих операцій, що застосовуються до даних певного типу;
- спосіб зображення даних у пам’яті комп’ютера.

Цілочислові типи

Цілочислові типи – це типи даних, множини допустимих значень яких є множинами цілих чисел. Ідентифікатори цілочислових типів, множини допустимих значень цих типів та об’єми пам’яті, що потрібні для збереження відповідних даних, наведені у табл. 1.

Табл. 1 – Цілочислові типи

Ідентифікатор типу	Кількість байтів	Діапазон значень
char	1	-128..+127 (2^7-1)
unsigned char	1	0..255 (2^8-1)
short	2	-32768..32768 (2^{15}-1)
unsigned short	2	0..65535 (2^{16}-1)
int	4	-2147483648.. 2147483647 (2^{31}-1)
unsigned int	4	0..4294967295 (2^{32}-1)
long	4	-2147483648.. 2147483647 (2^{31}-1)
unsigned long	4	0..4294967295 (2^{32}-1)

Більш детально з типами, що використовуються в системі Visual Studio можна ознайомитися за посиланням

https://msdn.microsoft.com/ru-ru/library/s3f49ktz.aspx

У мовах C\C++ визначені знакові та беззнакові типи даних. У разі, коли нема потреби зберігати від’ємні значення, використовують беззнакові типи даних. Їх позначають з ключовим словом unsigned.

Над усіма цілочисловими типами визначено однаковий набір операцій (табл. 2). Ці операції поділяються на унарні (з одним операндом) та бінарні (з двома операндами).

Табл. 2 – Арифметичні операції та операції порівняння

Знак операції	Зміст операції
+	Додавання
-	Віднімання
*	Множення
/	Визначення цілої частини від ділення
%	Визначення остачі від ділення

- (унарний)	Зміна знаку числа
++	Префіксний та постфіксний інкремент (збільшення на 1)
--	Префіксний та постфіксний декремент (зменшення на 1)
<<	Зсув бітів ліворуч (2<<1=4, 1<<5=32)
>>	Зсув бітів праворуч (2>>1=1, 8>>2=2)
==	Дорівнює
!=	Не дорівнює
>	Більше
<	Менше
>=	Більше дорівнює (не менше)
<=	Менше дорівнює (не більше)

Значення типів char, short, int, long є знаковими, а типів з ключовим словом unsigned – беззнаковими. При додаванні, відніманні та множенні знакових цілих чисел можливе перенесення одиниці зі старшого цифрового розряду в знаковий розряд. Така ситуація називається **переповненням**. Саме це відбувається при додаванні, наприклад, 1 і 32767 для типу short. При переповненні комірки оперативної пам'яті формується результат у межах множини значень цілочислового типу, але не обов'язково правильний.

Дійсні типи

Для запису дійсних чисел в оперативній пам'яті використовується формат з **плаваючою комою.**

Для оперування з досить великими числами у прикладних задачах виникає необхідність представити дійсне число у наступному вигляді:

$$Y = \pm M \cdot S^{\pm p}.$$

Тут Y – значення дійсного числа, M – мантиса числа, S – основа системи числення, p – порядок числа.

Числами в формі з плаваючою комою є, наприклад, маса Сонця $2x10^{30}$ кг, або маса електрона $9x10^{-28}$ г.

Кількість цифр у мантисі характеризує точність числа. Чим більше цифр у мантисі, тим вище точність. Порядок визначає місцезнаходження десяткової точки в числі.

У мовах C\C++ означено три дійсних типи (табл. 3): дійсний (float), дійсний з подвійною точністю (double), дійсний з підвищеною точністю (long double). Серед усіх дійсних типів має найширший діапазон і найвищу точність, але потребує при цьому найбільших витрат пам'яті.

Табл. 3 – Дійсні типи даних

Назва	Кількість байтів	Найменше за модулем число	Найбільше за модулем число
float	4	$3,4x10^{-38}$	$3,4x10^{38}$
double	8	$1,7x10^{-308}$	$1,7x10^{308}$
long double	10	$3,4x10^{-4932}$	$1,1x10^{4932}$

Над дійсними числами можна виконувати ті ж самі операції, що і для цілих чисел.

Булів тип

Множина допустимих значень булевого, або логічного, типу містить дві константи: false (хибність) і true (істина). Назва цього типу даних походить від приізвища видатного англійського математика Джорджа Буля, засновника математичної логіки. Ідентифікатор логічного типу bool означено тільки в мові C++. У C\C++ усі значення, що відрізняються від нуля, вважаються істинними.

До булевих значень застосовуються операції «і», «або», «ні», що називаються логічним множенням (кон'юнкцією), логічним додаванням (диз'юнкцією) і запереченням відповідно. Ці операції позначаються лексемами &&, ||, та ! відповідно. Результати застосування цих операцій до булевих значень наведено в табл. 4.

Табл. 4 – Булеві операції

a	b	a&&b	a||b	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

7. Оператори умовного переходу

Оператори if та if-else

Узагальнена форма оператора if має вигляд:

if (вираз)

оператор

Якщо значення виразу відрізняється від нуля (true), тоді оператор виконується, якщо вираз дорівнює нулю (false) – оператор пропускається.

Оператор if-else має узагальнену форму:

if (вираз)

оператор 1

else

оператор 2

Якщо вираз відмінний від нуля, тоді виконується оператор 1, а оператор 2 пропускається; якщо вираз – нуль, тоді пропускається оператор 1 і виконується оператор 2.

8. Оператори циклу

Оператор while

Узагальнена форма оператора while:

while (вираз)
    оператор

Спочатку обчислюється вираз. Якщо результат відмінний від нуля (true), тоді виконується оператор і управління переходить до початку циклу while. Тіло цикла, тобто оператор виконується до тих пір, поки вираз не стане рівним нулю (false).

Оператор for

Оператор for – ітераційний оператор, який використовується зі змінною – лічильником циклу. Узагальнена форма оператора for має вигляд:

for (вираз 1; вираз 2; вираз 3)
    оператор

Вираз 1 представляє собою визначення та ініціалізація лічильника циклу. Вираз 2 – умова продовження циклу, якщо вираз 2 відмінно від нуля (true) – виконується оператор та обчислюється вираз 3. Вираз 3 – вираз за яким змінюється лічильник циклу (збільшується або зменшується).

Приклад 2. Програма обчислення суми цілих чисел від 0 до 10.

```
sum = 0;
for (i = 1; i <= 0; ++i)
    sum += i;
```

Оператор do

Оператор do може розглядатися як варіант оператора while, однак, на відміну від while, цикл do виконує обчислення виразу після виконання оператора. Узагальнена форма має вигляд:

do
    оператор
while (вираз)

Таким чином, оператор виконається принаймні один раз.

Спочатку виконується оператор, потім обчислюється вираз. Якщо його результат відмінний від нуля (true), тоді управління переходить на початок циклу do. Якщо значення виразу нуль (false), тоді управління переходить до наступного після do оператора.

9. Оператори передачі управління

Оператори break і continue

Щоб перервати потік управління в циклі використовуються два спеціальних оператори **break** і **continue.**

Оператор **break** виходить з самого вкладеного циклу. На відміну від нього, оператор **continue** завершує поточну ітерацію циклу та починає наступну.

Оператор switch

Оператор switch – умовний оператор, який узагальнює оператор **if-else**. Стандартна форма оператора:

```
switch (вираз)
        case вираз 1: оператор 1; break;
        case вираз 2: оператор 2; break;
…
default:
        оператор за замовченням;
```

Резюме лекції

Таким чином, на цій лекції були розглянуті такі питання:

1. Інтуїтивне поняття алгоритму.
2. Формалізоване поняття алгоритму.
- Машина Поста.
- Машина Тюрінга.
- Нормальний алгоритм Маркова.
3. Приклад алгоритмів пошуку НСД.
4. Розглянуто основні типи задач.

ЛІТЕРАТУРА

1. Ахо А., Хопкрофт В., Ульман Д. Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. – М. : «Вильямс», 2010. – 384 с.
2. Вирт Н. Алгоритмы и структуры данных. – «Невский Диалект», 2008. – 352 с.
3. Каррано Ф.М., Причард Д.Д. Абстракция данных и решение задач на C++. - М. : «Вильямс», 2003. – 848 с.
4. Ковалюк Т.В. Алгоритмізація та програмування. Львів: «Магнолія 2006», 2013. – 400 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. – М. : «Вильямс», 2007. – 1296 с.
6. Логинов В.И., Шемагина Л.Н. Основы алгоритмизации. – Н. Новгород: «ВГАВТ», 2010. – 81 с.
7. Левитин А. Алгоритмы: введение в разработку и анализ. : Пер. с англ. – М. : «Вильямс», 2006. – 576 с.
8. Потопахин В.В. Современный самоучитель по алгоритмам. – М.: ДМК Пресс, 2012. – 320 с.
9. Скиена С. Алгоритмы. Руководство по разработке. – СПб.: БХВ-Петербург, 2011. – 720 с.
10. Сэджвик Р. Функдаментальные алгоритмы на С. – К.: «ДиаСофт», 2001. – 688 с.

Додаткова:

1. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. 3-е изд.: Пер. с англ.: Уч. пос. -М.: «Вильямс», 2000. – 720 с.

2. Кнут Д. Искусство программирования, том 3. Сортировка и поиск. 2-е изд.: Пер. с англ.: Уч. пос. -М.: «Вильямс», 2000. – 824 с.

3. Мартинюк Тетяна Борисівна. Рекурсивні алгоритми багатоопераднної обробки інформації.- Вінниця: Універсум, 2000.- 216с.

4. Лекції курсу "Розробка та аналіз алгоритмів".- Запоріжжя: ЗДУ, 2000.- 54с.

5. Караванова, Тетяна Петрівна Основи алгоритмізації та програмування: 750 задач з рекомендаціями та прикладами: Посіб.- К.: ФОРУМ, 2002.- 287 с.

6. Кулик А.Я. Адаптивні алгоритми передавання інформації: Монографія. - Вінниця: Універсум, 2003.- 214с.