

ЕЛЕМЕНТИ АЛГОРИТМІЧНИХ МОВ. ТИПИ ДАНИХ. СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ

Загальна структура C++ програми

У загальному випадку програма мовою C++ має таку структуру:

1. *Коментар про призначення програми.*
2. *Область глобальних оголошень*
 - 2.1 Підключення бібліотек.
 - 2.2 Визначення препроцесора (констант, макросів тощо).
 - 2.3 Оголошення прототипів функцій.
 - 2.4. Оголошення глобальних змінних.
- 3 *Визначення головної функції.*
 - 3.1 Оголошення локальних змінних.
 - 3.2 Виклик функцій.
 - 3.3 Оператори.
- 4 *Визначення інших функцій.*

Основні функції мови C визначаються в системних бібліотеках і описуються в заголовних файлах. Список *заголовних файлів* визначений стандартом мови. Нижче наведені деякі з них

- math.h – математичні функції та макроси
- stdio.h – функції введення-виведення, а також типи та макроси
- string.h – функції для роботи з рядками символів
- time.h – типи і функції, пов'язані з датою та часом

Заголовні файли мають розширення .h і підключаються директивою #include. При підключенні заголовного файлу його текст, а з ним і текст відповідного файлу, автоматично вставляється у програму замість рядка з відповідною директивою #include. При компіляції програми усі підключені заголовні файли з файлами їхньої реалізації перекомпілюються, тому підключення надмірної кількості таких файлів сповільнює компіляцію програми. Хоча в заголовних файлах містяться всі описи стандартних функцій, у код програми включаються лише ті функції, які використовуються у програмі.

Розглянемо для прикладу наступну програму

```
//функції введення - виведення, а також типи та макроси
#include <stdio.h>
//директива препроцесора включення файлу
#include <iostream>
//використання проостору імен
using namespace std;
```

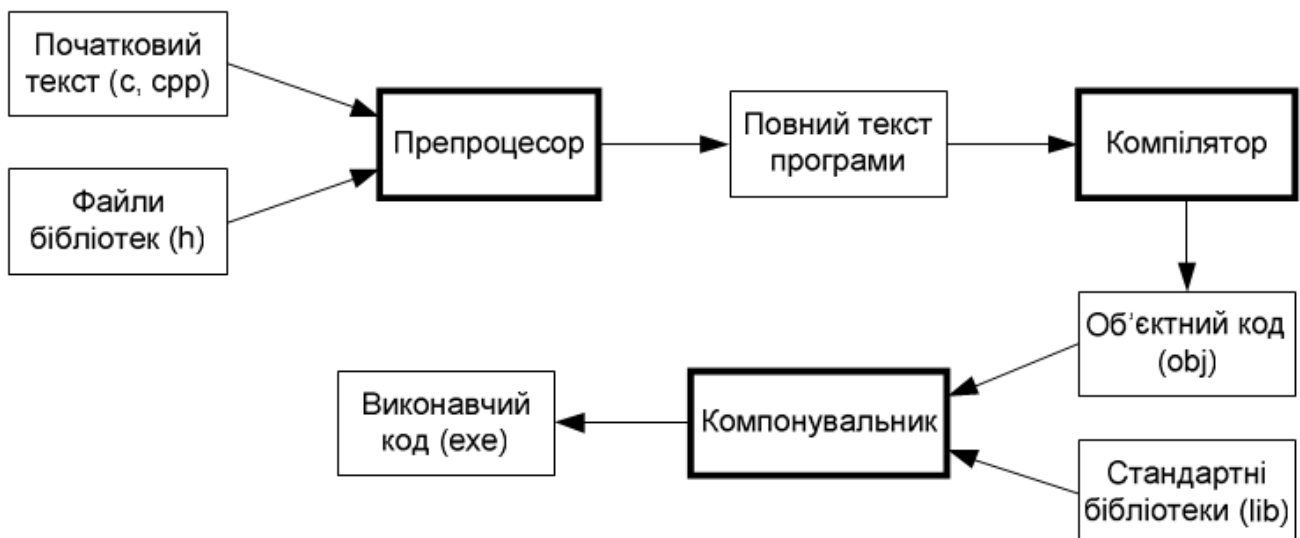
```

//Визначення головної функції
int main() {
    //Оголошення локальних змінних
    int X, Y, rs;
    //зчитування даних з переходом на новий рядок
    cout << "Enter X " << endl;
    cin >> X;
    cout << "Enter Y " << endl;
    cin >> Y;
    //обчислення
    rs = ((120 - X - Y - Z) / 5);
    //виведення результату
    cout << "Amount of programs " << rs;
    //завершення виконання функції, значення 0 - програма успішно завершила
    виконання
    return 0;
}
    
```

Компіляція та виконання програми

Початкова програма, написана на C/C++ у вигляді текстового файла, проходить такі три етапи опрацювання:

- 1) *препроцесорне перетворення тексту* – попереднє опрацювання тексту програми;
- 2) *компіляція* тексту програми в об'єктний код;
- 3) *компонування* (редагування зв'язків або збірка) – зв'язування об'єктного коду з кодом функцій із стандартних бібліотек та генерування виконуваного коду – файлу програми.



Етапи опрацювання програми

Після цих трьох етапів формується виконуваний код програми.

Препроцесор – це комп'ютерна програма, яка приймає дані на вході і видає дані, призначені для входження іншої програми, наприклад, такої як компілятор.

У мові програмування C++ вбудовано підтримку препроцесора. Рядки в початковому коді, які мають бути опрацьовані препроцесором у вигляді `#define` та `#include`, називаються препроцесорними директивами.

Завдання препроцесора – перетворення тексту програми до її компіляції. Правила препроцесорного опрацювання визначає програміст за допомогою директив препроцесора.

Директиви препроцесора – команди, які керують перетворенням тексту програми до її трансляції. Вони дають змогу змінювати текст програми, наприклад, замінити певні лексеми у тексті, вставляти текст з іншого файлу, забороняти трансляцію частини тексту тощо. Всі директиви препроцесора починаються з символу `#`. Після директив препроцесора крапка з комою не ставиться.

У програмуванні термін “директива” (вказівка) за використанням схожий до терміну “команда”, оскільки також використовується для опису певних конструкцій мови програмування. Узагальнений формат директиви препроцесора має такий вигляд:

`# <ім'я_директиви> <лексеми_препроцесора>`

До `лексем_препроцесора` належать символні константи, імена файлів, які підключаються, ідентифікатори, знаки операцій, розділові знаки, рядкові константи (рядки), а також будь-які символи, відмінні від пробілу.

Залежно від препроцесора перед символом `#` і після нього в директиві можуть бути дозволені пробіли. Пробіли також дозволені перед лексемами препроцесора, між ними і після них.

У препроцесорах мови C/C++ визначені такі препроцесорні директиви:

- `#define` – визначення макроса чи препроцесорного ідентифікатора;
- `#include` – підключення тексту з файлу;
- `#undef` – скасування визначення макросу чи препроцесорного ідентифікатора;
- `#if` – перевірка умови-виразу;
- `#ifdef` – перевірка визначеності ідентифікатора;
- `#ifndef` – перевірка невизначеності ідентифікатора;
- `#else` – початок альтернативної гілки для `#if`;
- `#endif` – завершення умовної директиви `#if`;
- `#elif` – складена директива `#else/#if`;
- `#line` – ім'я файлу і бажаний початковий номер рядка;

- `#error` – формування тексту повідомлення про помилку при трансляції;
- `#pragma` – дії, передбачені реалізацією;
- `#` – порожня директива.

Складові частини системи програмування та їх призначення

- Вбудований текстовий редактор
- Препроцесор
- Компілятор
- Стандартні бібліотеки (модулі)
 - Редактор зв'язків – необхідність зв'язування об'єктного коду із функціями стандартних бібліотек, генерація виконуваного ехе-файлу
- Вбудований налагоджувач
- Підсистема допомоги

Коментарі

Коментарі використовуються лише для пояснень і жодних дій у програмі не виконують, тому що ігноруються компілятором.

Однорядкові коментарі позначаються символами `//` і записуються перед текстом коментаря:

```
// Текст після цього символу і до кінця рядка є коментарем
// Це також однорядковий коментар
```

Багаторядкові коментарі записуються між символами `/*` та `*/`.

```
/* Коментар до програми може займати кілька
   рядків */
```

Також коментарі часто використовуються під час відлагодження програми для тимчасового “вимкнення” певного її фрагмента.

Введення/виведення даних

Функція `cin`

Функція `cin` використовується для введення даних. Якщо функція введення має декілька параметрів, то дані потрібно вводити послідовно через пробіл, а після останнього введеного значення необхідно натиснути клавішу Enter. Якщо вводяться числові та символні значення відмінні від пропуску (пробілу), то пробіл як розділовий символ не використовується, оскільки він також є звичайним символом. Наприклад,

```
int n; double m;
// введення значень окремо
cin >> n;
cin >> m;
// введення значень через пробіл
cin >> n >> m;
```

Функція cout

Для виведення потоку використовується функція cout. У цій функції також можуть використовуватись маніпулятори, тобто функції, які з поточними об'єктами cin, cout записуються як звичайні змінні, проте виконують певні операції над потоком. При використанні маніпуляторів потрібно підключити бібліотеку iomanip:

`#include <iomanip>`

До основних маніпуляторів належать такі:

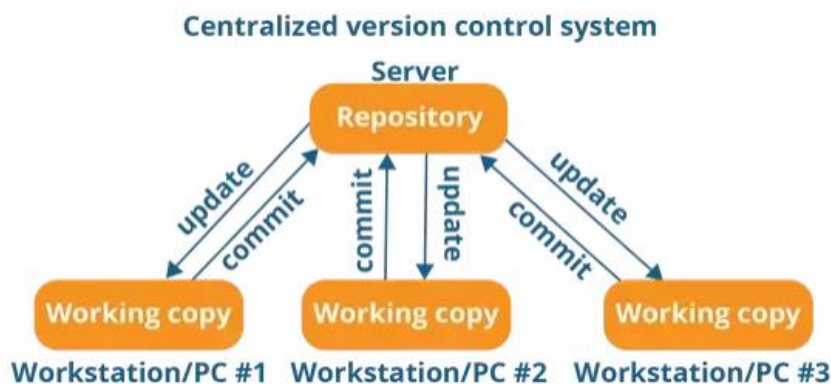
- endl – забезпечує переведення курсору на новий рядок;
- setw(n) – задає ширину поля, де n – це значення ширини, ціле число;
- setprecision(n) – задає точність значення, де n – це значення точності, ціле число;
- dec, hex, oct – задають системи числення (десяткову, шістнадцяткову та вісімкову відповідно).

Системи контролю версій.

Системи контролю версій – дуже важлива частина розробки програмного забезпечення. Це певна категорія програмних інструментів, які, з плином часу, допомагають розробникам керувати змінами у вихідному коді. Програмне забезпечення контролю версій відслідковує всі зміни внесені в код в спеціальній базі даних і при виявленні помилки розробники можуть повернутись назад і порівняти поточний код з його попередніми версіями, що допомагає виправити виявлені неполадки.

Централізовані системи контролю версій

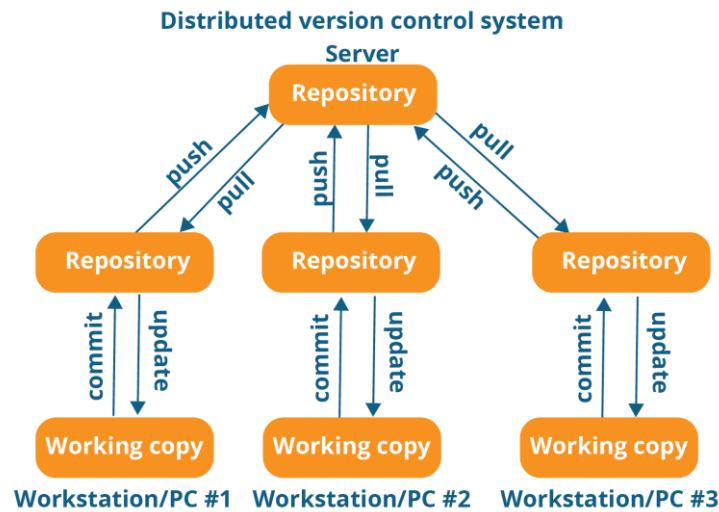
Для організації такої системи контролю використовується *центральный репозиторий*, що зберігається на сервері і який є єдиним хранилищем всіх версій файлів. Розробник працює з локальною копією і надсилає зміни в центральний репозиторий. Такі системи зберігають історію змін в вигляді ліній, на відміну від розподілених (у вигляді графа).



Розподілені системи контролю версій

Дозволяють зберігати репозиторий (його копію) у кожного розробника, що працює з даною системою. При цьому можна умовно виділити центральний репозиторий, в який будуть надсилатись зміни з локальних і з ним ці локальні репозиторії будуть синхронізуватись. При роботі з такою системою користувачі періодично синхронізують свої локальні репозиторії з центральним і працюють

безпосередньо зі своєю локальною копією і після внесення змін до неї, вони (зміни) відправляються на сервер. Великою перевагою такого підходу є автономія розробника при роботі з проектом, завдяки тому, що кожен має локальну копію центрального репозиторія.



Git – найпопулярніша розподілена система контролю версій з відкритим вихідним кодом.

▪ *Система контролю:* Git відстежує і зберігає контент. Найчастіше він використовується для програмного коду.

▪ *Система контролю версій:* код, який зберігається в Git може змінюватися і додаватися. Часто над одним проектом паралельно працюють кілька програмістів. Система контролю версій зберігає історію змін і дозволяє зручно з нею працювати. Крім того в Git є такі зручні речі як гілки (branch) і злиття (flow).

▪ *Розподілена система контролю версій:* в Git є віддалений репозиторій, який розташовується на сервері, і локальний репозиторій на машині кожного розробника. Це означає, що код не тільки зберігається на центральному сервері. На кожному комп'ютері є його повна копія.

Основною відмінністю від інших систем є те, як Git сприймає дані. Більшість СКВ зберігають інформацію як список змін до вмісту файлів. Такі системи розглядають інформацію як список файлів та змін кожного з них протягом деякого часу.

Замість цього, Git сприймає свої дані як низку знімків мініатюрної файлової системи. Щоразу, як ми зберігаємо стан нашого проекту (створюємо commit – «коміт», збереження), Git запам'ятовує, як виглядають всі наші файли в цей момент і зберігає посилання на цей знімок. Для ефективності, якщо файли не змінилися, Git не зберігає файли знову, просто робить посилання на попередній ідентичний файл, котрий вже зберігається. Тобто, Git розглядає свої дані як потік знімків

Три стани Git

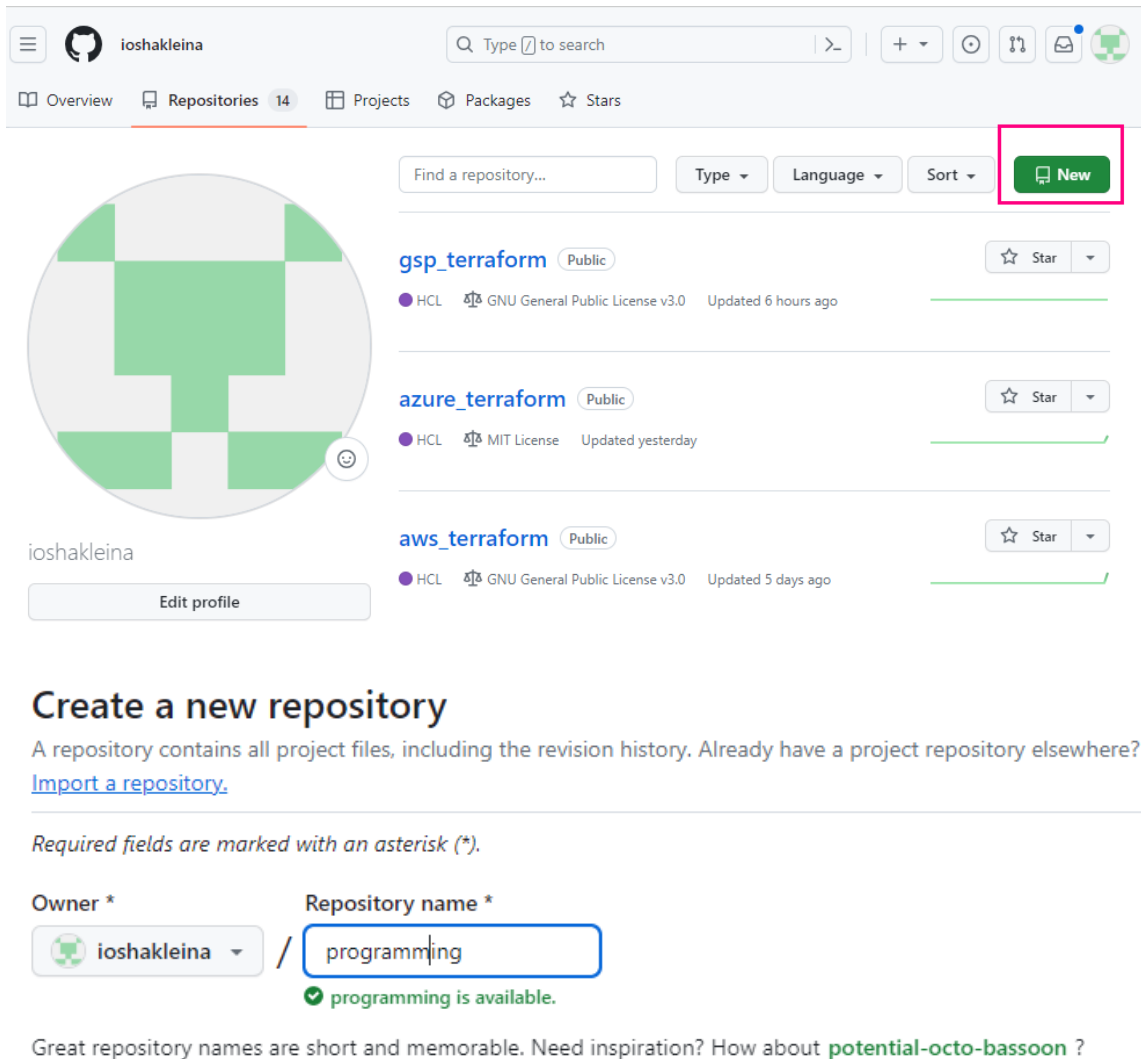
Найважливіша річ, яку слід запам'ятати: Git має три основних стани, в яких можуть перебувати файли: збережений у коміті (committed), змінений (modified) та індексований (staged):

- *Збережений у коміті* означає, що дані безпечно збережено в локальній базі даних.

- **Змінений** означає, що у файл внесено зміни, які ще не збережено в базі даних.
- **Індексований стан** виникає тоді, коли позначили змінений файл у поточній версії, щоб ці зміни увійшли до наступного знімку коміту.

Щоб *імпортувати проект* в свій git-репозиторій на GitHub, потрібно:

1. Створити репозиторій



The screenshot shows the GitHub profile page for 'ioshackleina'. The 'New' button to create a repository is highlighted with a pink box. Below the repository list, the 'Create a new repository' form is shown. The 'Owner' is 'ioshackleina' and the 'Repository name' is 'programming', which is marked as available with a green checkmark. The form also includes a note about required fields and a suggestion for repository names.

2. **Ініціалізувати репозиторій** – відкрити консоль git (для папки з проектом вибрати пункт контекстного меню Git Bash Here) виконати:

```
$ git init
```

3. **Вибрати файли, які потрібно перенести:**

```
$ git add .
```

4. **Створити збереження (commit)** і дати назву

```
$ git commit -m "назва"
```

5. **Вказати власний репозиторій** за допомогою команди

```
$ git remote add origin <шлях_до_репозиторію>
```

6. **Відправити файл(и),** вказавши гілку (branch)

```
$ git push -u origin master
```

Команди останніх двох пунктів можна скопіювати з GitHub:

...or push an existing repository from the command line

```
git remote add origin https://github.com/ioshackleina/programming.git  
git branch -M main  
git push -u origin main
```

Якщо все зроблено правильно, то з'явиться вікно авторизації, після якої файли проєкту буде додано до відповідної гілки.