Лекція 1
Вступ. Основні поняття

План

1. Преамбула
2. Алфавіт і лексика мови C++, ідентифікатори.
3. П’ять основних типів дних
4. Змінні-1
    4.1. Ініціалізація змінних
5. Операції в C++
6. Структура програми
    6.1. Стейтменти
    6.2. Вирази
    6.3. Функції
    6.4. Бібліотеки
    6.5. Приклад простої програми
7. Функції
    7.1. Значення, що повертаються
    7.2. Тип повернення void
    7.3. Повернення значень функцією main()
    7.4. Детальніше про значення, що повертаються
    7.5. Ікладені функції
    7.6. Локальні змінні
8. Фази компіляції
9. Замість висновків

1. Преамбула.

Мова програмування C++ (вимовляється як *«Сі плюс плюс»*) була розроблена Б'ярном Страуструпом в *Bell Telephone Laboratories* в 1979 році в якості доповнення до мови С. Вона додала безліч нових функціональних можливостей в мову С, однією з яких було об'єктно-орієнтованість даної мови. Щодо об'єктно-орієнтованого програмування (ООП) і його відмінностей від традиційних методів програмування ми поговоримо на відповідних заняттях.

Мова С++ була схвалена комітетом ISO в 1998 році і потім знову в 2003 році (під назвою **C++03**). Потім були наступні оновлення версій (раз в 3 роки), котрі додали ще більше функціональних можливостей:

- **C++11** в 2011 році;
- **C++14** в 2014 році;
- **C++17** в 2017 році;
- **C++20** в 2020 році.

Філософія мови програмування C++

Філософію мови програмування С++ можна визначити виразом "довіряти програмісту". Наприклад, компілятор не заважатиме вам зробити щось нове (що має сенс), але також не заважатиме вам зробити щось таке, що може призвести до збою. Це одна з головних причин, чому так важливо знати те, що можете робити в С/С++, так і те, що ви не повинні робити.

2. Алфавіт і лексика мови C++, ідентифікатори.

Алфавіт C++ складається з латинських великих і маленьких літер, арабських чисел та спеціальних символів:

+ - * / < > == | & ! \ ` ' @ # $ % ^ ? _ : ; , . ( ) [ ] { } " .

Лексика C++:

- ідентифікаторі відрізняються по першим 32 символам, причому великі і маленькі літери відрізняються, тобто name і Name різні змінні
- зарезервовані та ключові слова мови, подані в таблиці

Таблиця 1 Ключові ідентифікатори C++

asm	const	dynamic_cast	goto
auto	const_cast	else	if
break	continue	enum	inline
case	default	extern	int
catch	delete	float	long
char	do	for	new
class	double	friend	operator

private	struct	virtual	
protected	switch	void	
public	template	while	
register	this		
return	throw		
short	try		
signed	typedef		
sizeof	typeid		
static	union		
static_cast	unsigned		

Ідентифікатори

У мові C/C++ імена змінних, функцій, міток і інших об'єктів, створених користувачем, називаються *ідентифікаторами* (indentifiers). Ідентифікатори можуть складатися з одного або декількох символів. Перший символ ідентифікатора повинен бути буквою або символом підкреслення, а наступні символи повинні бути буквами, цифрами або символами підкреслення. Нижче наведені приклади правильних і неправильних ідентифікаторів.

Правильно	***Неправильно***
Count	lcount
test23	hi!there
high_balance	high...balance

Однак не всі символи, що утворять ідентифікатор, уважаються значущими. Якщо ідентифікатор використовується в процесі редагування зовнішніх зв'язків, то значущими вважаються, принаймні, шість його перших символів. Ці ідентифікатори називаються *зовнішніми іменами* (external names). До них ставляться імена функцій з глобальними змінними, використовуваними декількома файлами. У мові C++ немає обмежень на довжину ідентифікаторів і значущими вважаються, принаймні, 1024 перших символів.

Символи, набрані у верхньому й нижньому регістрі, різняться. Отже, count, Count і COUNT — це різні ідентифікатори.

Ключові слова не можна використовувати як ідентифікатори. Крім того, ідентифікатори не повинні збігатися з іменами функцій зі стандартних бібліотек.

3. П'ять основних типів даних

У C++ існують п'ять елементарних типів даних: символ, ціле число, число із плаваючою крапкою, число із плаваючою крапкою подвоєної точності й змінна,

що не має значень. Їм відповідають наступні ключові слова: **char, int, float, double** і **void.** Всі інші типи даних у мові С++ створюються на основі елементарних типів, зазначених вище. Розмір змінних і діапазон їхніх значень залежить від типу процесора й компілятора. Однак у всіх випадках розмір символу дорівнює 1 байт. Розмір цілочисельної змінної звичайно дорівнює довжині машинного слова, прийнятої в конкретній операційній системі. Однак, прагнучи до машино незалежності програм, варто уникати конкретних припущень про розмір цілочисельних змінних. Важливо чітко розуміти, що в мові С++ акцентується лише на *мінімальному діапазоні* у якому змінюються значення змінних кожного типу, а не їхній розмір у байтах.

Точне подання чисел із плаваючою крапкою залежить від їхньої конкретної реалізації. Розмір цілого числа звичайно дорівнює довжині машинного слова, прийнятої в операційній системі. Значення змінних типу **char,** як правило, використовуються для подання символів, передбачених у системі кодування ASCII. Значення, що виходять за межі припустимого діапазону, на різних комп'ютерах обробляються по-різному.

Діапазон зміни змінних типу **float** і **double** залежить від способу подання чисел із плаваючою крапкою. У кожному разі, цей діапазон досить широкий. Мінімальна кількість цифр, що визначають точність чисел із плаваючою крапкою, зазначене в табл. 2.

Таблиця 2. Характеристики основних типів даних С++

Ім’я типа	Байти	Діапазон значень
int	4	від -2 147 483 648 до 2 147 483 647
bool	1	false або true
char	1	від -128 до 127
double	8	1,7E +/- 308 (15 знаків)
float	4	3.4E+/- 38

Тип **Void** використовується для визначення функції, що не повертає ніяких значень, або для створення узагальненого покажчика (generic pointer).

4. Змінні-1

Змінна (variable) являє собою ім'я комірки пам'яті, яку можна використовувати для зберігання значення, що модифікується. Всі змінні повинні бути оголошені до свого використання. Нижче наведений загальний вид оголошення змінної

тип список_змінних;

Тут слово *тип* означає один із припустимих типів даних, включаючи модифікатори, а *список_змінних* може складатися з одного або декількох ідентифікаторів, розділених комами. Розглянемо кілька прикладів оголошень.

```
int  i, j, L;
 short   int   si;
 unsigned int ui;
 double balance, profit, loss;
```

Де оголошуються змінні

Як правило, змінні повідомляють у трьох місцях: усередині функцій, у визначенні параметрів функції й за межами всіх функцій. Відповідно такі змінні називаються локальними, формальними параметрами й глобальними.

4.1 Ініціалізація змінних

```
int a = 9;     int a=4*6+b;
```

Копіююча ініціалізація (або ще **“ініціалізація копіюванням”**) з допомогою знаку рівності (=).

```
int b(9);
```

Пряма ініціалізація з допомогою круглих дужок.

```
int c{ 6 };
```

uniform-ініціалізація

Пряма ініціалізація може працювати краще з деякими типами даних, копіююча ініціалізація — з іншими типами даних.

Пряма або копіююча ініціалізації працюють не з усіма типами даних (наприклад, ви не зможете використовувати ці форми для ініціалізації списку значень).

У спробі забезпечити єдиний механізм ініціалізації, який буде працювати з усіма типами даних, в C++11 додали нову форму ініціалізації, яка називається **uniform-ініціалізацією**:

Ініціалізація змінної з порожніми дужками вказує на ініціалізацію за замовчуванням (змінній присвоюється 0):

```
int  c{};  // ініціалізація змінної за замовчуванням - значенням 0
```

У uniform-ініціалізації є ще одна додаткова перевага: ви не можете присвоювати змінній значення, яке не підтримує її тип даних — компілятор видасть попередження або повідомлення про помилку, наприклад:

```
int value{ 4.5 }; // помилка: цілочисельна змінна не може містити не цілочисельні значення
```

```
int a;
cin >> a;
```

Введення значення з клавіатури

```
int g = rand();
```

За допомогою функції генерування випадкових чисел

5. Операції в С++

Арифметичні операції С++ наводяться нижче в таблиці. За винятком операції обчислення остачі від ділення, яка має сенс лише для символьного і цілого типу, арифметичні операції застосовуються до всіх арифметичних типів, в тому числі булевих.

Арифметичні операції

Символ операції	Назва	Спосіб використання
*	Множення	op1 * op2
/	Ділення	op1 / op2
%	Остача	op1 % op2
+	Додавання	op1 + op2
-	Віднімання	op1 - op2
,	Кома	op1, op2

Операція “кома” використовується тоді, коли на місці, синтаксично придатному для розміщення одного виразу, необхідно розмістити декілька виразів. Значенням виразу, побудованого за допомогою цієї операції, вважається значення її другого операнда. Перший операнд виявляється як би зайвим, на перший погляд його обчислення стає безрезультатним. Але це не так, оскільки в С++ існують операції, здатні змінювати значення змінних, зокрема, операцією є саме присвоєння, а також операції інкременту і декременту.

Операції порівняння

Символ операції	Назва	Спосіб використання
<	Менше	op1 < op2
<=	Менше або рівне	op1 <= op2
>	Більше	op1 > op2
>=	Більше або рівне	op1 >= op2
= =	Рівне	op1 == op2
!=	Не рівне	op1 != op2

Особливу увагу варто звернути на специфічне позначення рівності. Вживання знаку присвоєння ”=” замість знаку рівності ”= =” — одна з самих типових помилок в С/С++-програмах.

С++ успадкував від С два типи логічних операцій. Крім звичайних заперечення, кон'юнкції і диз'юнкції, є ще побітові логічні операції, які застосовуються до будь-яких інтегральних типів.

Складне присвоєння

До операторів складного присвоювання відносяться +=, -=, *=, /=.

Оператор x+=p; призначений для збільшення x на величину p. Оператор x-=p; призначений для зменшення x на величину p. Оператор x*=p; призначений для множення x на p. Оператор x/=p; призначений для розподілу x на p.

Операції цілочисельної арифметики

До операцій цілочисельної арифметики відносяться:
цілочисельне ділення /;
остача від ділення %.

При цілочисельному діленні операція / повертає цілу частина частки (дробова частина відкидається), а операція % - остача від ділення. Нижче наведені приклади цих операцій

11 % 4 = 3
11/4 = 2
7 % 3 = 1
7/3 = 2
26/5 = 5
26 % 5 = 1

Операції збільшення (інкримент) і зменшення (декремент)

У мові С++ є операції збільшення (++) і зменшення (--) на одиницю.
Оператор
p=p+1;
можна записати в префіксній формі
++p;
так і в постфіксній
p++;

Ці форми відрізняються при використанні їх у виразах. Якщо знак декремента (інкримента) передує операнду, то спочатку виконується збільшення (зменшення) значення операнда, а потім операнд бере участь у виразі.
Наприклад,
x=12;
y=++x;

У результаті в у буде зберігатися число 13.

Якщо знак декремента (інкремента) треба після операнда, то спочатку операнд бере участь у виразі, а потім виконується збільшення (зменшення) значення операнда.
Наприклад,
x=12;
y=x++;

У результаті в у буде зберігатися число 12.

Логічні операції

Символ операції	Назва	Спосіб використання
!	Заперечення	! op
&&	Кон'юнкція	op1 && op2
||	Диз'юнкція	op1 || op2
? :	Імплікація	(op1 ? op2: op3)

Аргументи і результати логічних операцій, крім імплікації, булеві. Імплікація служить для створення умовних виразів. Її перший операнд буллевий, а два інші довільні арифметичні, результат арифметичний. Визначають

імплікацію тотожності
(true ? x: y) == x
(false ? x: y) == y

Крім названих вище логічних операцій, в C++ є також побітові логічні операції. Вони застосовуються до арифметичних типів і самі дають результат арифметичного типу. Найпростіше ці операції вживати до типу unsigned int або unsigned long , що відповідатиме роботі з машинними словами. У випадку коротших типів або при наявності знаку результат може залежати від типу компілятора. Це значить, що одна і та ж програма даватиме після підготовки до виконання різними компіляторами даватиме різні результати. Але сучасні технології програмування віддають перевагу програмам, не залежним від компілятора і навіть платформи. Такі програми називають переносимими (portable).

Логічні побітові операції

Символ операції	Назва	Спосіб використання
~	Заперечення	~ op
<<	Зсування вліво	op1 << op2
>>	Зсування вправо	op1 >> op2
&	Кон'юнкція	op1 & op2
^	Виключна диз'юнкція	op1 ^ op2
|	Диз'юнкція	op1 | op2

Побітові логічні операції — це програмування на низькому, близькому до машинного, рівні. Програмування рівнем вище використовує стандартний клас bitset , визначений у стандартній бібліотеці.

В C++ існують прототипи стандартних математичних функцій. Перелік цих функцій наведено в табл.

Таблиця стандартних математичних функцій

abs (x)	Повертає абсолютне значення x
acos (x)	Повертає арккосинус x
asin (x)	Повертає арксинус x
atan (x)	Повертає арктангенс x
cbrt (x)	Повертає корінь кубічний з x
cos (x)	Повертає косинус x
cosh (x)	Повертає гіперболічний косинус x
exp (x)	Повертає значення E^x
expm1 (x)	Повертає e^x -1
fabs (x)	Повертає абсолютне значення плаваючого x
fdim (x, y)	Повертає додатну різницю між x та y
hypot (x, y)	Повертає sqrt ($x^2 + y^2$) без проміжного переповнення або недоливу
fma (x, y, z)	Повертає x * y + z без втрати точності

fmax (x, y)	Повертає найвище значення плаваючих x та y
fmin (x, y)	Повертає найнижче значення плаваючих x та y
fmod (x, y)	Повертає залишок з плаваючою комою від x / y
log(x)	натуральний логарифм від x
log10(x)	десятковий логарифм числа x
pow (x, y)	Повертає значення x у ступінь y
sin (x)	Повертає синус x (x в радіанах)
sinh (x)	Повертає гіперболічний синус подвійного значення
tan (x)	Повертає тангенс кута
tanh (x)	Повертає гіперболічний тангенс подвійного значення

Операції округлення:

round(), - математичне округлення
ceil(), - округлення до більшого
floor() - округлення в сторону меншого
trunc() – відкидання дробової частини

6. Структура програми

6.1 Стейтменти

Стейтмент (англ. ***"statement"***) — це найбільш поширений тип інструкцій у програмах. Це і є та сама найменша незалежна одиниця в **мові програмування C++**. Стейтмент в програмуванні — це те ж саме, що і "речення" в українській мові. Ми використовуємо речення для того, щоб виразити якусь думку/ідею. У C++ ми пишемо стейтменти, щоб виконати якесь завдання. **Всі стейтменти в C++ закінчуються крапкою з комою**.

Є дуже багато різних видів стейтментів в C++. Переглянемо найбільш поширені з них:

```
1  int x;
2  x = 5;
3  std::cout << x;
```

int x — це **стейтмент оголошення** (англ. *"statement declaration"*). Він повідомляє компілятору, що x є змінною.
x = 5 — це **стейтмент присвоювання** (англ. *"assignment statement"*). Тут ми присвоюємо значення 5 змінній x.
std::cout << x; — це **стейтмент виводу** (англ. *"output statement"*). Ми виводимо значення змінної x на екран.

6.2 Вирази

Компілятор також здатний опрацьовувати вирази. **Вираз** (англ. *"expression"*) — це математичний об'єкт, який генерує певне значення. Наприклад, в математиці вираз 2 + 3 генерує значення 5.

Виразами можуть бути:

- значення (наприклад, 2, 4);
- змінні (наприклад, x, y);
- оператори (наприклад, +, -);
- функції.

Вони можуть бути як одним значенням (наприклад, 2 чи x), так і комбінацією значень (наприклад, 2 + 3, 2 + x, x + y чи (2 + x) * (y - 3)). Наприклад, x = 2 + 3; — це коректний стейтмент присвоювання. Вираз 2 + 3 генерує результат: значення 5, яке потім присвоюється змінній x.

6.3 Функції

В C++ стейтменти об'єднуються в блоки — функції. **Функція** — це послідовність стейтментів. Кожна програма в C++ повинна містити **головну функцію main()**. Саме з першого стейтмента в main() і починається виконання програми. Функції, як правило, виконують конкретне завдання. Наприклад, функція max() може містити стейтменти, які визначають максимальне число з двох переданих їй, а функція calculateGrade() може обчислювати оцінку студента.

6.4 Бібліотеки

Бібліотека — це набір скомпільованого коду (наприклад, функцій), який був "упакований" для повторного використання в інших програмах. За допомогою бібліотек можна розширити функціонал програм. Наприклад, якщо ви пишете гру, то вам доведеться підключити бібліотеку звуку або графіки (якщо ви самостійно не хочете їх писати з нуля).

Мова C++ не така вже й велика, як ви могли б подумати. Проте, вона поставляється в комплекті зі **Стандартною бібліотекою C++**, яка надає додатковий функціонал. Однією з найбільш часто використовуваних частин Стандартної бібліотеки C++ є **бібліотека iostream**, яка дозволяє виводити інформацію на екран і опрацьовувати дані, які вводить користувач.

6.5 Приклад простої програми в C++

Тепер, коли у вас є загальне уявлення про те, що таке стейтменти, функції та бібліотеки, давайте розглянемо програму "Hello, world!":

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello, world!";
6     return 0;
7 }
```

Рядок №1: Спеціальний тип інструкції, який називається **директивою препроцесора**. Директиви препроцесора повідомляють компілятору, що йому потрібно виконати певне завдання. В цьому випадку ми повідомляємо компілятору, що хотіли б підключити вміст заголовкового файлу iostream до нашої програми. Заголовковий файл iostream дозволяє нам отримати доступ

до функціоналу бібліотеки iostream для можливості виведення інформації на екран.
Рядок №2: Порожній простір, який ігнорується компілятором.
Рядок №3: Оголошення головної функції main().
Рядок №4 і №7: Вказуємо компілятору область функції main(). Все, що знаходиться між відкриваючою фігурною дужкою в рядку №4 і закриваючою фігурною дужкою в рядку №7, — вважається частиною функції main().
Рядок №5: Наш перший стейтмент (який закінчується крапкою з комою) — стейтмент виводу. std::cout — це спеціальний об'єкт за допомогою якого ми можемо виводити дані на екран. << — це оператор виводу. Все, що ми відправляємо в std::cout, — виводиться на екран. Тут ми виводимо текст "Hello, world!".
Рядок №6: **Оператор повернення return**. Коли програма завершує своє виконання, функція main() передає значення, яка вказує на результат виконання програми (успішно чи ні), назад в операційну систему. Якщо оператор return повертає значення 0, то це означає, що все добре! Будь-які ненульові числа використовуються для того, щоб вказати, що щось пішло не так.

Більш функціональніший шаблон функції **main()** описується наступним кодом:

```
1   #include <iostream>
2   #include "windows.h"
3   using namespace std;
4
5   int main()
6   {
7       SetConsoleCP(1251);
8       SetConsoleOutputCP(1251);
9   // все що потрібно тут!!!
10      // і тут також
11      cout << endl;
12      system("pause");
13      return 0;
14  }
```

З нового тут появилось:

1. **Кирилиця в консолі**

```
#include <Windows.h>
…
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
```

2. **Затримка консолі**

```
system("pause");
```

також можна використовувати:
cin.clear();
cin.ignore(32767, '\n');
cin.get();
system("pause"); - можуть працювати не на всіх типах операційних систем

3. підключення простору імен std **рядок 3** – дане підключення дозволяє не писати постійно std::cout
4. **в рядку 11** ми переводимо курсор виведення на рядок нище команда **endl**, тим самим створюємо порожній рядок в консолі виведення, що дозволяє візуально відділити різну інформацію при виведенні на екра.

7. Функції

Функція — це послідовність стейтментів для виконання певного завдання. Часто програми перериваються виконанням одних функцій заради виконання інших. Ви це постійно робите в реальному житті, наприклад, ви читаєте книгу і згадали, що повинні були зробити телефонний дзвінок. Ви залишаєте *закладку* в своїй книзі, берете телефон і набираєте номер. Після того, як ви вже поговорили, ви повертаєтеся до тієї сторінки в книзі, на якій ви зупинилися.

Програми в C++ працюють схожим чином. Іноді, коли програма виконує код, вона може зіткнутися з викликом функції. **Виклик функції** — це **вираз**, який вказує процесору перервати виконання поточної функції і приступити до виконання іншої функції. Процесор “залишає закладку” в поточній точці виконання, а потім виконує функцію, що викликається. Коли виконання функції, що викликається, — завершено, то процесор повертається до “закладки” і відновлює виконання перерваної функції.

Функція, в якій знаходиться виклик, називається **викликаючою функцією** (англ. ***“caller”***). Наприклад:

```
1  #include <iostream> // для std::cout і std::endl
2
3  // Оголошення функції doPrint(), яку ми будемо викликати
4  void doPrint() {
5      std::cout << "In doPrint()" << std::endl;
6  }
7
8  // Оголошення функції main()
9  int main()
10 {
11     std::cout << "Starting main()" << std::endl;
12     doPrint(); // перериваємо виконання main() викликом
   функції doPrint(). Функція main() в даному випадку є
   викликаючою функцією
13     std::cout << "Ending main()" << std::endl;
14     return 0;
15 }
```

Результат виконання програми:

```
Starting main()
In doPrint()
Ending main()
```

Ця програма починає своє виконання з першого рядка функції main(), в якому виводиться на екран Starting main(). Другий рядок функції main() викликає функцію doPrint(). На цьому етапі виконання стейтментів у функції main() призупиняється і процесор переходить до виконання стейтментів всередині функції doPrint(). Перший (і єдиний) рядок в doPrint() виводить текст In doPrint(). Коли процесор завершує виконання doPrint(), він повертається назад в функцію main() до тієї точки, на якій зупинився. Отже, наступним стейтментом є вивід рядка Ending main() на екран.

Зверніть увагу, для виклику функції потрібно вказати її ім'я і список параметрів в круглих дужках (). У вищенаведеному прикладі параметри не використовуються, тому круглі дужки порожні.

Правило: Не забувайте вказувати круглі дужки () при виклику функцій.

7.1 Значення, що повертаються

Коли функція main() завершує своє виконання, вона повертає цілочисельне значення назад в операційну систему, використовуючи **оператор return**.

Функції, які ми пишемо, також можуть повертати значення. Для цього потрібно вказати **тип повернення значення**. Він вказується при оголошенні функції, перед її ім'ям. Зверніть увагу, що тип повернення не вказує, яке саме значення повертатиметься. Він вказує тільки тип цього значення.

Після цього всередині функції, що викликається, ми використовуємо оператор return, щоб вказати фактичне **значення, що повертається**.

Розглянемо просту функцію, яка повертає цілочисельне значення:

```
1  #include <iostream>
2
3  // int означає, що функція повертає цілочисельне значення у
   викликаючу функцію
4  int return7()
5  {
6        // Ця функція повертає цілочисельне значення, тому ми повинні
   використовувати оператор return
7        return 7; // повертаємо число 7 у викликаючу функцію
8  }
9
10 int main()
11 {
12       std::cout << return7() << std::endl; // на екран виведеться 7
13       std::cout << return7() + 3 << std::endl; // на екран виведеться
   10
14
15       return7(); // значення 7, що повертається, - ігнорується, тому
   що main() з ним нічого не робить
16
17       return 0;
18 }
```

Результат виконання програми:

```
7
10
```

Тепер давайте розберемося детально:

Перший виклик функції return7() **рядок 12** повертає 7 у викликаючу функцію, де воно потім передається в **std::cout** для виводу на екран.

Другий виклик функції return7() **рядок 13** знову повертає 7 у викликаючу функцію. Вираз 7 + 3 генерує результат 10, який виводиться на екран.

Третій виклик функції return7() **рядок 15** знову повертає 7 у викликаючу функцію. Проте main() нічого з ним не робить, тому нічого і не відбувається (значення, що повертається, просто ігнорується).

> **Примітка**: Значення, що повертаються, не виводяться на екран, якщо їх не передавати об'єкту std::cout. В останньому виклику функції return7() значення не надсилається в std::cout, тому нічого і не відбувається.

7.2 Тип повернення void

Функції можуть і не повертати значення. Щоб повідомити компілятору, що функція не повертає значення, потрібно використати **тип повернення void**. Погляньмо ще раз на функцію doPrint() з вищенаведеного прикладу:

```
1 void doPrint() // void - це тип повернення
2 {
3     std::cout << "In doPrint()" << std::endl;
4     // Ця функція не повертає значення, тому і оператор return
  тут не потрібен
5 }
```

Ця функція має тип повернення void, який означає, що функція не повертає значення. Оскільки значення не повертається, то і оператор return тут не потрібен.

Ось ще один приклад використання функції типу void:

```
1  #include <iostream>
2
3  // void означає, що функція не повертає значення
4  void returnNothing()
5  {
6      std::cout << "Hi!" << std::endl;
7      // Ця функція не повертає значення, тому і оператор return
   тут не потрібен
8  }
9
10 int main()
11 {
12     returnNothing(); // функція returnNothing() викликається,
   але в main() нічого не повертається
13
14     std::cout << returnNothing(); // помилка: цей рядок не
   скомпілюється (вам потрібно буде його закоментувати)
15     return 0;
16 }
```

У першому виклику функції returnNothing(), **рядок 12,** виводиться Hi!, але нічого не повертається назад у викликаючу функцію. Точка виконання повертається назад в main(), де програма продовжує своє виконання.

Другий виклик функції returnNothing(), **рядок 14.** навіть не скомпілюється. Функція returnNothing() має тип повернення void, який означає, що ця функція не повертає значення. Однак main() намагається відправити це значення (яке не повертається) в std::cout для виведення на екран. std::cout не може опрацювати цей випадок, оскільки не було надано значення для виводу. Отже, компілятор видасть помилку. Вам потрібно буде **закоментувати** цей рядок, щоб компіляція пройшла успішно.

7.3 Повернення значень функцією main()

Тепер у вас є розуміння того, як працює функція main(). Коли програма виконується, операційна система викликає функцію main() і починається її виконання. Стейтменти в main() виконуються послідовно. В кінці функція main() повертає цілочисельне значення (зазвичай 0) назад в операційну систему. Тому main() оголошується як int main().

Чому потрібно повертати значення назад в операційну систему? Справа в тому, що значення, що повертається функцією main(), є **кодом стану**, який повідомляє операційній системі про те, чи успішно було виконання програми. Зазвичай, значення 0 (нуль) означає що все пройшло успішно, тоді як будь-яке інше значення означає невдачу/помилку.

Зверніть увагу, за стандартами мови C++ функція main() повинна повертати цілочисельне значення. Однак, якщо ви не вкажете return в кінці функції main(), то компілятор поверне 0 автоматично, якщо ніяких помилок не буде. Проте все ж рекомендується вказувати оператор return в кінці функції main() і використовувати тип повернення int для функції main().

7.4 Детальніше про значення, що повертаються

По-перше, якщо тип повернення функції не є void, то вона повинна повертати значення зазначеного типу (використовуючи оператор return). Єдиний виняток — це функція main(), яка повертає 0, якщо не надано інше значення.

По-друге, коли процесор зустрічає в функції оператор return, він негайно виконує повернення значення назад у викликаючу функцію і точка виконання також переходить у викликаючу функцію. Будь-який код, який знаходиться за оператором return у функції — ігнорується.

Функція може повертати тільки одне значення через оператор return у викликаючу функцію. Це може бути або число (наприклад, 7), або значення змінної, або вираз (який генерує результат), або певне значення з набору можливих значень.

Є способи обійти правило повернення одного значення, але про це трішки пізніше.

Нарешті, автор функції вирішує, що означатиме значення, яке повертає функція. Деякі функції використовують повернені значення в якості кодів стану для надання інформації щодо результату виконання функції (успішне виконання чи ні). Інші функції повертають певне значення з набору можливих значень, ще інші функції взагалі нічого не повертають.

Повторне використання функцій

Одну і ту ж функцію можна викликати декілька разів, навіть в різних програмах, що дуже корисно:

```
1   #include <iostream>
2
3   // Функція getValueFromUser() отримує значення від користувача,
    а потім повертає його назад у викликаючу функцію
4   int getValueFromUser()
5   {
6       std::cout << "Enter an integer: ";
7       int x;
8       std::cin >> x;
9       return x;
10  }
11
12  int main()
13  {
14      int a = getValueFromUser(); // перший виклик функції
    getValueFromUser()
15      int b = getValueFromUser(); // другий виклик функції
    getValueFromUser()
16
17      std::cout << a << " + " << b << " = " << a + b <<
    std::endl;
18
19      return 0;
20  }
```

Результат виконання програми:

```
Enter an integer: 4
Enter an integer: 9
4 + 9 = 13
```

Тут виконання функції main() переривається 2 рази. Зверніть увагу, в обох випадках отримане значення від користувача зберігається в змінній x, а потім передається назад в функцію main() за допомогою оператора return, де присвоюється змінній a або b!

Також main() не є єдиною функцією, яка може викликати інші функції. Будь-яка функція може викликати будь-яку іншу функцію!

```
1   #include <iostream>
2
3   void printO()
4   {
5       std::cout << "O" << std::endl;
6   }
7
8   void printK()
9   {
10      std::cout << "K" << std::endl;
11  }
12
```

```
13  // Функція printOK() викликає як printO(), так і
14  printK()
    void printOK()
15  {
16      printO();
17      printK();
18  }
19
20  // Оголошення функції main()
21  int main()
22  {
23      std::cout << "Starting main()" << std::endl;
24      printOK();
25      std::cout << "Ending main()" << std::endl;
26      return 0;
27  }
```

Результат виконання програми:

```
Starting main()
O
K
Ending main()
```

7.5 Вкладені функції

В мові C++ одні функції можуть бути оголошені всередині інших функцій (тобто бути вкладеними). Наступний код викличе помилку компіляції:

```
1  #include <iostream>
2
3  int main()
4  {
       int boo() // ця функція знаходиться всередині
5  функції main(), що є заборонено
6      {
7          std::cout << "boo!";
8          return 0;
9      }
10
11     boo();
12     return 0;
13 }
```

Правильно ось так:

```
1  #include <iostream>
2
3  int boo() // тепер вже не в функції main()
4  {
5      std::cout << "boo!";
6      return 0;
```

```
7  }
8
9  int main()
10 {
11      boo();
12      return 0;
13 }
```

7.6 Локальні змінні

Змінні, оголошені усередині функції, називаються *локальними* (local variables). Локальні змінні можна використовувати тільки в операторах, розташованих усередині блоку, де вони оголошені. Інакше кажучи, локальні змінні невидимі зовні їхнього блоку. Нагадаємо, що блок обмежений відкриваючою й закриваючою фігурною дужками.

Найчастіше локальні змінні оголошуються усередині функцій. Розглянемо два приклади.

```
void  func1(void)
{
      int x;
      x = 10
}

void func2(void)
{
      int x;
      x = -199;
     }
```

Змінна x оголошена двічі: спочатку – у функції **find ()**, а потім — у функції **func2().** Змінна x з функції **fund** () не має ніякого відношення до змінного x, оголошеної усередині функції **func2 ()**. Кожна із цих змінних існує тільки усередині блоку, де вона була оголошена.

З міркувань зручності й за традицією більшість програмістів оголошують всі змінні, використовувані у функції, відразу після відкриваючої фігурної дужки й перед всіма іншими операторами. Однак варто мати на увазі, що локальні змінні можна повідомляти в будь-якому місці блоку. Розглянемо наступний приклад.

```
void f(void)
{
      int   t;
      cin >> t;
      if (t = = l) {
           char s[80];   /* Ця змінна створюється тільки при
                                          вході в даний блок
*/
           cout << "Уведіть ім'я: ";
           gets(s); /*  Якісь операції   ...   */
      }
     }
```

У цьому фрагменті локальна змінна **s** створюється при вході в блок if і руйнується відразу після виходу з нього. Крім того, змінна **s** є видимою тільки

усередині блоку **if**, і до неї неможливо звернутися ззовні, навіть із інших частин функції, що містить даний блок.

Оголошення змінних усередині блоків дозволяє уникнути побічних ефектів. Оскільки локальна змінна поза блоком не існує, її значення неможливо змінити ненавмисно.

Оскільки локальна змінна створюється при вході в блок, де вона оголошена, і руйнується при виході з нього, її значення губиться. Це особливо важливо враховувати при виклику функції. Локальні змінні функції створюються при її виклику й знищуються при поверненні керування в зухвалий модуль. Це значить, що між двома викликами функції значення її локальних змінних не зберігаються. (Змусити їх зберігати свої значення можна за допомогою модифікатора **static**.)

За замовчуванням локальні змінні зберігаються в стеці. Оскільки стік є динамічною структурою, і обсяг займаної їм пам'яті постійно змінюється, стає зрозумілим, чому локальні змінні в принципі не можуть зберігати свої значення між двома викликами функції, усередині якої вони оголошені.

Формальні параметри

Якщо функція має аргументи, варто оголосити змінні, які будуть приймати їхні значення. Ці змінні називаються *формальними параметрами* (formal parameters). Усередині функції вони нічим не відрізняються від інших локальних змінних. Як показано в наведеному нижче фрагменті програми, оголошення таких змінних повинне розміщатися відразу після ім'я функції й полягати в дужки.

Приклад:

```
/*  Функція  повертає 1, якщо символ з є частиною рядка s; у
протилежному випадку вона повертає 0 */
int  is_in(char* s, char c)
{
     while (*s)
          if (*s == c)  return 1;
          else  s++;
     return 0;
    }
```

Функція **is_in()** має два параметри: **s** і **c.** Вона повертає 1, якщо символ з є частиною рядка **s**, у противному випадку функція повертає 0.

Тип формальних параметрів задається при їхньому оголошенні. Після цього їх можна використовувати як звичайні локальні змінні. Урахуйте, що, як і локальні змінні, формальні параметри є динамічними й руйнуються при виході з функції.

Формальні параметри можна використовувати в будь-яких конструкціях і вираженнях. Незважаючи на те що свої значення вони одержують від аргументів, переданих ззовні функції, у всім іншому вони нічим не відрізняються від локальних змінних.

Глобальні змінні

На відміну від локальних, *глобальні змінні* (global variables) доступні з будь-якої частини програми й можуть бути використані де завгодно. Крім того, вони зберігають свої значення на всім протязі виконання програми. Оголошення глобальних змінних повинні розміщатися поза всіма функціями. Ці змінні можна використовувати в будь-якому вираженні, у якому би блоці воно не перебувало.

У наведеному нижче фрагменті програми змінна **count** оголошена поза всіма функціями. Незважаючи на те, що її оголошення розташоване до функції **main (),** цю змінну можна було б з таким же успіхом оголосити в іншому місці, але поза функціями й до її першого використання. І все-таки найкраще розміщати оголошення глобальних змінних на самому початку програми.

```
#include  <iostream>
int   count;   /*   Змінна   count   є   глобальною   */
void  func1(void);
void func2(void);

int main(void)
{
     count = 100;
     func1();

     return  0;
}

void  func1(void)
{
     int  temp;
     temp = count;
     func2();
     std::cout<<"count   = "<< count;   /*   Виведе   число   100.
*/
}

void func2(void)
{
     int   count;
     for (count = 1; count < 10; count++)
          putchar('.');
    }
```

Придивіться до цієї програми уважніше. Помітьте, що, хоча ні функція **main ()**, ні функція **fund ()** не містять оголошення змінної **count,** обидві ці функції успішно неї використовують. Однак усередині функції **func2()** оголошена локальна змінна **count.** Коли функція **func2 ()** посилається на змінну **count,** вона використовує лише локальну змінну, а не глобальну. Якщо глобальна і локальна змінні мають однакові імена, то всі посилання на ім'я змінної усередині блоку будуть ставитися до локальної змінної й не впливати на її глобальну тезку. Можливо, це зручно, однак про цю особливість легко забути, і тоді дії програми можуть стати непередбаченими.

Глобальні змінні зберігаються в спеціально відведеному для них місці. Вони виявляються досить корисними, якщо різні функції в програмі використовують ті самі дані. Однак, якщо в них немає особою необхідності, глобальних змінних варто уникати. Справа в тому, що вони займають пам'ять під час усього виконання програми, навіть коли вже не потрібні. Крім того, використання глобальних змінних там, де можна було б обійтися локальними, послабляє незалежність функцій, оскільки вони змушені залежати від сутності, певної в іншому місці. Отже, застосування великої кількості глобальних змінних підвищує ймовірність помилок внаслідок непередбачених побічних ефектів.

Більшість проблем, що виникають при розробці більших програм, є наслідком непередбаченої зміни значень змінних, які використовуються в будь-якому місці програми. Це може трапитися, якщо в програмі оголошено занадто багато глобальних змінних.

8. Фази компіляції

Після створення вихідний текст компілюється. Етап компіляції і редагування зв'язків розпадається на кілька фаз.

1. Вихідний файл кодується основним набором текстових символів, причому наприкінці кожного рядка програми розставляються символи переходу на новий рядок. Кожен символ, що не входить в основний набір текстових символів, замінюється відповідним універсальним символом.

2. З проміжного тексту видаляються символи переходу на новий рядок, і фізичні рядки вихідного коду стають логічними. Якщо в процесі трансформації тексту випадково виникнуть універсальні символи, поводження компілятора стандартом не регламентується. Це стосується і ситуації, коли непорожній текстовий файл не завершується символом переходу на новий рядок.

3. Вихідний файл розкладається на лексеми і послідовності роздільників, включаючи коментарі. При цьому текст програми не повинний завершуватися незакінченою лексемою чи незавершеним коментарем (наприклад, <include чи /* неповний коментар). Кожен коментар замінюється пробілом.

Символи переходу на новий рядок зберігаються. Обробка інших роздільників залежить від конкретного компілятора. Інтерпретація символів, що утворюють текст програми, залежить від контексту (наприклад, символ < у директиві #include <ім'я файлу> і символ < в умовному вираженні обробляються по-різному).

4. Виконуються директиви препроцесора і макровизначення. Якщо в процесі конкатенації лексем випадково утвориться універсальний символ, поводження компілятора не регламентується. У результаті виконання директиви #include у вихідний текст програми вставляється текст відповідного заголовного файлу, до якого рекурсивно застосовуються фази 1–4.

5. Кожен символ з основного набору текстових символів, escape-послідовність чи універсальний символ, що є частиною символьного чи рядкового літерала, конвертується у відповідний елемент із набору керуючих символів.

6. Суміжні рядкові і розширені рядкові літерали конкатенуються.

7. Роздільники між лексемами ігноруються. Кожна лексема препроцесора перетворюється в звичайну лексему. Отримані лексеми піддаються синтаксичному і семантичному аналізу. Поняття “вихідний файл” і “одиниці трансляції” є абстрактними — їм не обов'язково відповідають фізичні файли. Інакше кажучи, щоб скомпілювати програму, її не обов'язково зберігати у файлі. (Це зауваження стосується, скоріше, до інтегрованих систем програмування, що поєднують текстовий редактор, компілятор і редактор зв'язків єдиним інтерфейсом.)

8. Розпізнаються всі зв'язки програми з зовнішніми об'єктами і функціями. Для зв'язування функцій і об'єктів, не визначених у процесі трансляції,

підключаються бібліотечні компоненти. У результаті виникає образ програми, що містить всю інформацію, необхідну для виконання програми (exe-модуль).

9. Замість висновків

Середовища для роботи

Веб-компілятори підходять для написання простих та невеликих програм, ідеально підходять для навчальних цілей. Їх функціонал обмежений: ви не зможете створювати виконувані файли або ефективно проводити відлагодження програм, тому краще скачати повноцінну IDE. Веб-компілятори більше підійдуть для швидкого запуску невеликих програм.

Популярні веб-компілятори:

1. **C++ Shell** http://cpp.sh/
2. **OnlineGDB** https://www.onlinegdb.com/
3. **TutorialsPoint** https://www.tutorialspoint.com/
    a. https://wandbox.org/
4. **Repl.it** - є можливість працювати із файлами.
   Для прикладу можна запустити наступну програму:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
 cout << "Hello KH!\n";

ofstream g;
g.open("my.txt");
g<<"Hello word";
g.close();
return 0;
}
```

Популярні середовища.

1. **Visual Studio 2019 Community edition** - https://visualstudio.microsoft.com/
2. **Code::Blocks** - https://www.codeblocks.org/
3. **Jet Brains** https://www.jetbrains.com/resharper-cpp/
4. **Qt creator** - https://www.qt.io/download

Використана література

1. Г. Шилдт, Полный справочник по C++, 4-е видання, в-во «Вильямс», 2006
2. Х.М.Дейтел, Как програмировать на C++, 4-е видання, в-во «Бином-Пресс» 2009.
3. Р. Лафоре, Объектно-ориентированное программирование в C++, в-во «Питер», 2004, с 924.