Лекція 7

Змінні, оператори, модулі.

Дані, що обробляються компілятором, – це змінні і константи. Всі змінні в мові С ++ повинні бути описані явно. Це означає, що, по-перше, на початку кожної програми або функції Ви повинні привести список імен (ідентифікаторів) всіх використовуваних змінних, По-друге, вказати тип кожної з них. Оператор опису складається з специфікації типу і списку імен змінних, розділених комою. В кінці обов'язково повинна стояти крапка з комою. При описі можливе завдання початкового значення змінної. Ім'я змінної – ідентифікатор. Ім'я може бути довільної довжини, але значущими є тільки перші 32 символу; інші символи імені ігноруються.

- **Опис типу.**

Специфікація типу формується з ключових слів, що вказують на різні типи даних. Основні типи в С++ поділяються на дві групи:

- цілочисельні типии: **char**, **short**, **int**, **long, bool**;
- типи з плаваючою точкою: **float**, **double**, **long double.**

Імена цілочисельних типів можуть використовуватися в поєднанні з парою модифікаторів тіпу **signed** і **unsigned**. Ці модифікатори змінюють формат представлення даних, але не впливають на розміри виділених областей пам'яті. Модифікатор тіпу **signed** вказує, що змінна може приймати як позитивні, так і негативні значення. Модифікатор типу **unsigned** вказує, що змінна приймає тільки позитивні значення. Основні характеристики цілочисельних типів виглядають наступним чином (табл):

Таблиця 6.2

Тип данных	Байты	Биты	Min	Max
signed char	1	8	- 128	127
unsigned char	1	8	0	255
signed short	2	16	-32768	32767
unsigned short	2	16	0	65535
signed int	2	16	-32768	32767
unsigned int	2	16	0	65535
signed long	4	32	-2147483648	2147483647
unsigned long	4	32	0	4294967295

За замовчуванням вважається, що дані типів **char**, **int**, **short int**, **long** використовуються зі знаком. Тому ключове слово **signed** можна не вказувати.

Дані типу **char** використовуються для зберігання символів. Під символом мається на увазі одиночна буква, цифра або знак, який займає тільки один байт пам'яті. Змінні типу **char** можуть використовуватися як дані зі знаком (**signed char**) і як дані без знака (**unsigned char**). Якщо тип **char**

розглядається як **signed**, то старший байт його коду визначає знак. В цьому випадку діапазон значень типу **char** – від-128до + 127. У випадку **unsigned char** всі вісім біт розглядаються як код, а діапазон можливих значень – від 0 до 255. Використовуючи тип даних char можна відобразити будь-який з 256 символів. Всі закодовані символи представлені в таблиці ASCII.

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

Приклад. Представлення в пам'яті комп'ютера цілих чисел

Всі значення в пам'яті представлені у двійковому коді. Так число 135 у двійковій системі дорівнює 10000111. В памяті воно буде представлене наступним чином:

- для формату у вигляді 1 байта – 10000111 (відсутній знаковый разряд);
- для формату у вигляді 2 байтів – 0000000010000111;
- - для форматі у вигляді 4 байтів – 00000000000000000000000010000111.

Самий крайній лівий розряд відводиться під знак і для додатніх чисел =0. Для представлення відємних чисел використовується додатковий код.

Так у форматі 2 байти число -135: 1111111101111000 (нулі заміщуються 1, а 1 нулями (інверсія)).

Також в мові С++ передбачений логічний тип даних **bool**, який займає 1 байт і приймає значення 0-255. Тип **bool** є цілочиселним, який використовується виключно для зберігання результатів логічних виразів. У логічного виразу може бути один з двох результатів *true* або *false*. *true* – якщо логічний вираз істинний, *false* – якщо логічний вираз помилковий.

Але так як діапазон допустимих значень типу даних bool від 0 до 255, то необхідно було якось зіставити даний діапазон з логічними константами *true* і *false*. Таким чином, константі *true* еквівалентні все числа від 1 до 255 включно, тоді як константі *false* еквівалентно тільки одне ціле число – 0. Розглянемо програму з використанням типу даних **bool**.

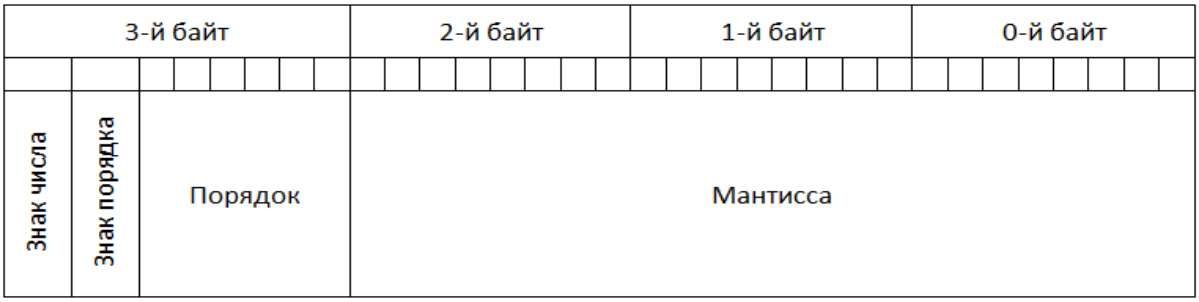
```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    bool boolean = 25; // змінна типу bool з іменем boolean
    // Значення 25 автоматично перетвориться на true,
    // оскільки будь-яке ненульове число інтерпретується як істина.

    if (boolean) // умова оператора if
        cout << "true = "  << boolean << endl;  // виконається, якщо умова істинна
    else
        cout << "false = " << boolean << endl; // виконається, якщо умова хибна

    system("pause"); // затримка перед завершенням програми (Windows)
    return 0;        // повернення коду завершення 0 (успішне виконання)
```

У рядку оголошена змінна типу bool, яка ініціалізована значенням 25. Теоретично після рядка 9, в змінній boolean повинно міститися число 25, але насправді в цій змінній міститься число 1. Результатом роботи програми буде: true =1.

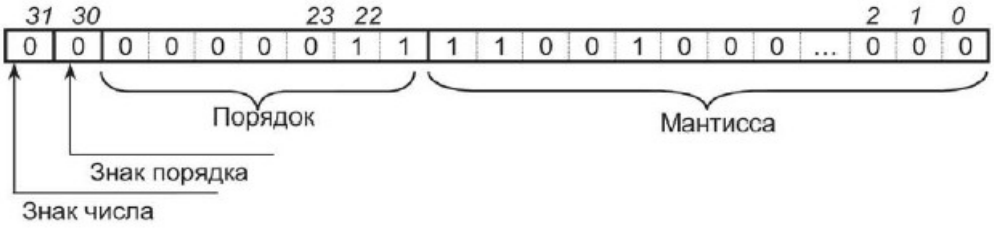
До плаваючих типів відносяться три типи, представлені наступними іменами типів, модифікаторів і їх поєднань: **float**, **double**, **long double**. Вони використовуються для роботи з дійсними числами, які подаються у формі запису з десятковою крапкою і в "наукової нотації". Різниця між нотаціями стає очевидною з простого прикладу, який демонструє запис одного і того ж реального числа в різних нотаціях:
297.7=2.977E2, 0.002355=2.355E-3.

У науковій нотації зліва від символу E (10) записується мантиса, праворуч – значення експоненти, яка завжди дорівнює показнику степеня 10.

Представимо це в загальному вигляді, як: $R = m \cdot E^n$, де m – називається мантисою, а порядок n – експонентою. У 4-байтовому форматі представлення дійсного числа перші три байта виділяються для розміщення мантиси, в четвертому байті розміщуються порядок числа, знаки числа і порядку.



Приклад запису числа $6{,}25_{10} = 110{,}01_2 = 0{,}11001 \cdot 2^{11}$, представленого в нормалізованому вигляді, в 4-х байтовому форматі з сімома розрядами для запису порядку.



Нижче представлені основні характеристики типів даних з плаваючою точкою:

Тип данных	Байты	Биты	Min	Max
float	4	32	3.4E-38	3.4E+38
double	8	64	1.7E-308	1.7E+308
long double	10	80	3.4E-4932	3.4E+4932

При описі змінних їм можна присвоювати початкове значення (ініціалізувати). При ініціалізації значення типу **char** беруться у апострофи.

В залежності від призначення програми може виникати необхідність роботи з 8-річними або 16-річними значеннями (наприклад, при роботі з адресами). В таких випадках необхідно повідомити компілятору, що значення не є десятковими.

Якщо числовому значенню передує 0 (наприклад, 07), то це означає, що значення є 8-річним числом.

Якщо числовому значенню передує 0x (наприклад 0xFF0), то це означає, що значення є 16-річним числом.

Тип за перерахуванням enum

У С виділений окремий тип перерахування (***enum***), що задає набір всіх можливих цілочисельних значень змінної цього типу. синтаксис перерахування:

enum <имя> { <імя поля 1>, <імя поля 2>, … <імя поля N>};

Приклад

```c
#include <conio.h>
#include <stdio.h>

// Оголошуємо перелічуваний тип (enum) Gender з двома значеннями
enum Gender { MALE, FEMALE };

int main() {
    enum Gender a, b; // оголошуємо дві змінні типу Gender

    a = MALE;   // змінній a присвоюємо значення MALE (0)
    b = FEMALE; // змінній b присвоюємо значення FEMALE (1)

    // Виводимо значення змінних у числовій формі (%d)
    printf("a = %d\n", a);
    printf("b = %d\n", b);

    getch(); // очікуємо натискання клавіші перед завершенням (Windows)
    return 0; // коректне завершення програми
}
```

У цій програмі оголошено змінну за перерахуванням з ім'ям Gender. Змінна типу enum Gender може приймати лише два значення – це MALE і FEMALE.

За замовчуванням, перше поле структури приймає числове значення 0, наступне 1, наступне 2 і т.д.

Необхідно зауважити, що значення типу за перерахуванням не являються символьними або строковими змінними і їх значення не береться у апострофи або лапки.

Зазвичай перерахування використовуються в якості набору іменованих констант.

Приклад.

```
typedef enum Bool {
   FALSE,
   TRUE
      } Bool;
```

Пустий тип void

Множина значень цього типу порожня. Він використовується для визначення функцій, які не повертають значення, для вказівки порожнього списку аргументів функції, як базовий тип для покажчиків і в операції приведення типів.

Опис типу користувача

Для того щоб зробити програму більш зрозумілою, можна задати типу нове ім'я за допомогою ключового слова **typedef**:

typedef тип нове_ім’я [розмірність];

Приклади.

```
1) typedef unsigned int UINT;
2) typedef char Msg[100];
3) typedef struct{
            char f1o[30];
            int date, code;
            double salary;} Worker;
```

Опис змінних і констант

Як було зазначено раніше, типи даних визначаються на початку програми за описом їх типів. Таке визначення типів називається явним.

Для опису констант використовують ключове слово **const** за наступним шаблоном:

const [модифікатор**]** тип ідентифікатор=значення;

Приклад. Опис константи:

```
const double PI=3.14; // PI — дійсна константа подвійної точності
```

Змінні описують наступним чином:

[модифікатор**]** тип список змінних;

де змінні у списку перераховуються через кому.

Приклади. Опис змінних.

1) **unsigned int** a,b; **//** a,b – цілочисельні змінні без знаку

2) **float** x=3.14; **//** x – дійсна змінна з початковим значенням 3.14

3) **char** symbol=’f’ – змінній символьного типу symbol присвоюється початкове значення – літера f.

4) **int** octal_value=0227; – задання 8-річного числа.

5**) int** hex_value=0xFF0; – задання 16-річного числа.

Наряду з явним виначенням типів в мові С передбачено неявне приведення типу: **тип**(змінна).

Приклад.

float(a) – змінна а приводиться до дійсного типу.

double(15) – число 15 приводиться до дійсного типу подвійної точності.

В С++ також передбачена унарна операція приведення типу:

static_cast</*тип даних*/>(/*змінна або число*/),

яка виконує перетворення за звичайними правилами.

Для динамічного перетворення використовують:

dynamic_cast<> ()

і для перетворення вказівників:

reinterpret_cast<> ()

Правила перетворення типів

Якщо операнди оператора належать до різних типів, то вони приводяться до деякого загального типу. Приведення виконується відповідно до невеликого числа правил. Зазвичай автоматично виконуються лише ті перетворення, які без будь-якої втрати інформації перетворюють операнди з меншим діапазоном значень в операнди з великим діапазоном:

1.Всі змінні типу **char** і **short int** перетворюються до типу **int**. Всі змінні типу **float** перетворюються до типу **double**.

2. Якщо один з пари операндів має тип **long double**, інший операнд також перетвориться до **long double**.

- інакше якщо один з операндів має тип **double**, інший операнд також перетвориться до **double**.

- інакше якщо один з операндів має тип **long**, інший операнд також перетвориться до **long**,

- інакше якщо один з операндів має тип **unsigned**, інший операнд також перетвориться до **unsigned**.

В результаті застосування цих правил перетворення кожна пара операндів матиме однаковий тип і результат кожної операції буде збігатися за типом з операндами. Зауважимо, що друге правило має кілька умов, які повинні застосовуватися послідовно. Наприклад, розглянемо перетворення типів, показане нижче.

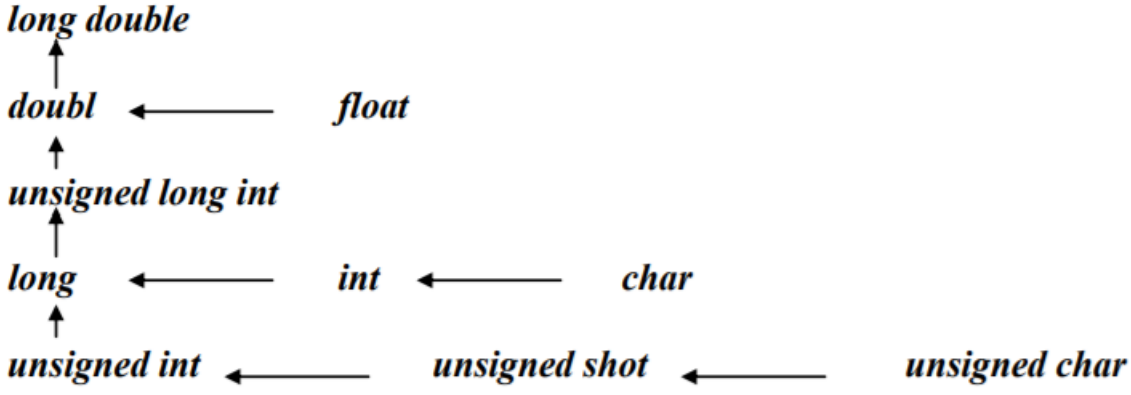
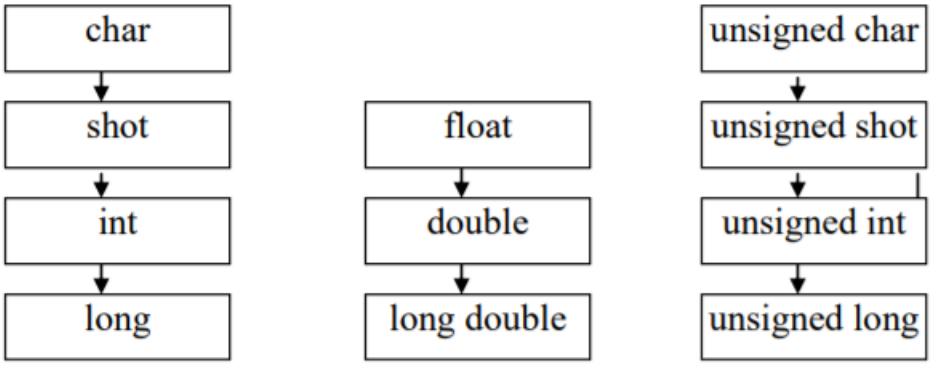


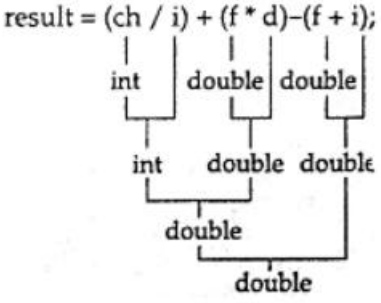
Схема послідовного перетворення типів



Перетворення, що гарантують збереження значення

Приклад перетворення типів

```
char ch;
int i;
float f;
double d;

result = (ch / i) + (f * d)-(f + i);
```



Але існують ситуації, де необхідно бути дуже уважним до перетворення типів. Наприклад, при діленні 2-х цілих чисел результат також буде цілим числом, що в загальному випадку є невірним.

Приклад.

7/2 буде дорівнювати 3, так як виконується операція над цілими числами, то і результат буде приведений до цілого числа.

Щоб отримати результат 3,5 необхідно хоча б один операнд привести до дійсного типу, що можна зробити двома способами:

7./3 – представити число 7 як дійсне, поставивши після нього десяткову крапку або за допомогою явного приведення типів float(7)/3.

Другий випадок, де необхідно бути обережним, це присвоєння даних різних типів. У цьому випадку керуються правилом, що змінній старшого типу можна присвоювати змінні молодшого типу, а не навпаки, інакше це може призвести до втрати інформації.

Приклад.

1)
```
int a=2;
float x;
x=a; // x прийме значення дійсного типу 2.
```

2)
```
int a
float x =2.5;
a=x; // a прийме цілочисельне значення 2.
```

3) **static_cast**<**float**>(15)/2– результат дорівнює 7.5

або

```
int ret=15;
static_cast<float>(ret)/2 //результат дорівнює 7.5
```

У разі зі змінною треба розуміти, що змінна ret не перетворюються на тип даних float, а всього лише на всього створюється тимчасова копія змінної ret з типом даних float.

Операція sizeof()

За допомогою операції **sizeof** можна визначити розмір пам'яті, яка відповідає ідентифікатору або типу. Вона має такий вигляд:

sizeof (вираз).

У якості виразу може бути використаний будь-який ідентифікатор (крім імені функції), або ім'я типу. Якщо в якості виразу вказано ім'я масиву, то результатом є розмір всього масиву (тобто добуток числа елементів на довжину типу). Результатом операції sizeof є розмір в байтах типу або оголошеної змінної, а стосовно до масивів операція повертає число байтів, необхідне для розміщення всіх елементів масиву. Якщо визначається розмір змінної, то ім'я змінної можна також вказувати через пропуск після sizeof без круглих дужок.

Приклад.

```
#include <iostream> // стандартна бібліотека вводу-виводу
using namespace std;

int main() {
    int i;
    char c;
    long double ff;

    cout << "Дані типу char займають " << sizeof(char) << " байт\n";
    cout << "Дані типу int займають " << sizeof(int) << " байт\n";
    cout << "Дані типу long займають " << sizeof(long) << " байт\n";

    cout << "Змінна i займає " << sizeof(i) << " байт\n";
    cout << "Змінна c займає " << sizeof(c) << " байт\n";
    cout << "Змінна ff займає " << sizeof(ff) << " байт\n";

    return 0; // коректне завершення програми
}
```

В результаті виконання програми отримаємо:

Дані типу char займають 1 байт
Дані типу int займають 2 байти
Дані типу long займають 4 байти
Змінна i займає 2 байти
Змінна c займає 1 байт

Змінна ff займає 10 байт