Лекція 3. Типи даних С/С++, використання змінних і констант.

1. Внутрішнє представлення цілочисельних даних.

2. Внутрішнє представлення даних дійсного типу.

3. Оголошення змінних та констант числових типів.

4. Форматований ввід-вивід даних.

5. Оператор безумовного переходу.

6. Оператор вибору (перемикач).

1. Внутрішнє представлення цілочисельних даних

Для зберігання цілочисельних даних зі знаком в IBM PC використовується так званий доповнювальний двійковий код. Запис додатних чисел відповідає їхньому запису в двійковій системі числення. Для отримання доповнювального коду від'ємного числа потрібно інвертувати (замінити 0 на 1, а 1 на 0) всі двійкові розряди відповідного додатного числа і додати одиницю в перший розряд. наприклад:

+5

0	0	0	0	0	**1**	0	1

−5

1	1	1	1	1	0	1	0

Доповнювальний код дозволяє приблизно на 25% прискорити виконання арифметичних операцій, таких як додавання та віднімання.

Найкоротші цілочисельні дані зі знаком представлені в пам'яті IBM-сумісних ПК одним байтом, в якому може розміститися будь-яке число з діапазону від -128 до 127, записане в доповнювальному коді. У мовах С/С++ для опису змінних такого типу використовується специфікатор ***char***. В новіших стандартах С++ запроваджено тип ***byte***, оскільки ***char*** хоча і є числовим типом, але традиційно використовується для зберігання символьних даних і «поводиться як текст», наприклад, при виводі потоком ***cout***. В змінну типу ***byte*** можна записати найкоротше ціле число без знака, за термінологією С таким числам відповідає специфікатор ***unsigned char***. Діапазон допустимих даних при цьому дорівнює [0, 255].

Друга категорія цілочисельних даних в системах програмування на IBM PC, представлена двобайтовими цілими числами. У варіанті зі знаком вони підтримують діапазон від -32768 до 32767, у варіанті без знака - від 0 до 65535. В ранніх компіляторах для позначення типу таких даних використовували ***int*** (коротке ціле зі знаком) та ***unsigned int*** (коротке ціле без знаку). В нових версіях компіляторів C++ цей цілочисельний тип даних отримав позначення ***short*** та ***unsigned short*** відповідно.

Третю категорію цілих чисел за класифікацією IBM PC представлено чотирьохбайтовими даними. У варіанті зі знаком вони покривають діапазон від -2147483648 до +2147483647, у варіанті без знака – від 0 до 4294967295. Для опису чотирьохбайтові даних цілого типу в C/C++ використовуються специфікатори ***long*** та ***unsigned long***. У середовищі візуального програмування C + + Builder та MS Visual Studio такі діапазони покриваються типами ***int*** та ***unsigned int***. Взагалі кажучи, кількість "потужність" типу даних залежить від системних можливостей, і в деяких системах типи ***int*** та ***long*** можуть виявитися еквівалентними за діапазоном допустимих даних.

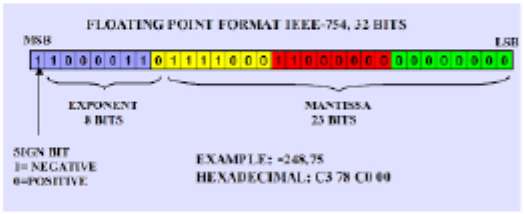
Існують і інші типи для оголошення цілочисельних даних, основні з них подано в таблиці нижче (за документацією з C++ від компанії Microsoft).

Назва типу	К-сть байтів	Інші назви	Діапазон значень
int	4	signed	-2147483648 2147483647
unsigned int	4	unsigned	0..4294967295
__int8	1	char	-128..127
unsigned__int8	1	unsigned char	0..255
__int16	2	short, shortint, signed shortint	-32768..32767
unsigned __int16	2	unsigned short, unsigned short int	0..65535
__int32	4	signed, signed int, int	-2147483648 2147483647
unsigned__int32	4	unsigned, unsigned int	0..4294967295
__int64	8	long long, signed long long	-9223372036854775808 9223372036854775807
unsigned__int64	8	unsigned long long	0 18446744073709551615
short	2	shortint, signed shortint	-32768..32767
unsigned short	2	unsigned shortint	0..65535
long	4	long int, signed long int	Від 2147483648 До 2147483647
unsigned long	4	unsigned long int	0..4294967295
long long	8	Еквівалентно _int64	- 9223372036854775808 9223372036854775807

Програмістові слід враховувати діапазон допустимих значень цілочисельних типів, зокрема, при виконанні арифметичних операцій. Компілятори C/C++ не контролюють виходу значень змінних за межі, дозволені для відповідного типу даних. Результат дій з цілими числами в двійковому представленні заноситься в пам'ять, відведену для змінної справа наліво, якщо розрядів (бітів) для зберігання числа недостатньо, зайві старші розряди відкидаються. Навіть, якщо усі біти числа поміщаються у відведену пам'ять, то для signed-типів це все одно може призводити до помилок через використання доповнювального коду. Наприклад, максимальне число двобайтного типу short 32767, що має двійкове представлення 01111111 11111111 після збільшення на 1 перетворюється в число 10000000 00000000, що є двійковим записом від'ємного числа -32768.

2. Внутрішнє представлення даних дійсного типу

Запис дійсних чисел, які можуть мати як цілу, так і дробову частину у вигляді окремого двійкового коду обидвох частин є нераціональним, бо дозволяє вмістити лише числа значно менші, аніж цілі типи аналогічного розміру. Для внутрішнього представлення дійсних чисел їх записують у канонічному вигляді і подають у так званому форматі з плаваючою крапкою. Цей формат передбачає зберігання у виділеній області пам'яті відразу двох компонентів числа – мантиса m і порядку p. Обидва компоненти зберігаються в своїй ділянці пам'яті у вигляді доповнювального двійкового коду. Мантиса m представлена дійсним числом з інтервалу [0; 1), тому для неї зберігається лише дробова частина у двійковій формі. Порядок p задає степінь числа 2 (додатний чи від'ємний), на який потрібно помножити мантису, щоб отримати саме число.



Такий спосіб представлення числових даних дозволяє при однаковій кількості двійкових розрядів, відведених для зберігання чисел істотно розширити діапазон допустимих даних. Так цілочисельні значення зі знаком в цьому в 32-бітному форматі (тип **`int`**) дозволяють використовувати числа, модуль яких належить інтервалу $[0, 2*10^9]$. Якщо для числа з плаваючою комою в 32-бітному записі виділити 1 двійковий розряд для знаку числа, 8 двійкових розрядів для зберігання порядку (степеня 2), та 23 розряди для мантиси, тоді найменше допустиме число рівне добутку мінімальної мантиси (2^{-1}) на мінімальний порядок (2^{-128}), тобто 2^{-129}, що приблизно відповідає 10^{-39}. Найбільше за модулем число є добутком $(1-2^{-23}) * 2^{127}$, що приблизно відповідає 10^{38}. Таким чином, якщо цілі 32-бітні числа покривають діапазон в 9 десяткових розрядів, то формат з плаваючою комою при тій ж розрядності слова перекриває діапазон в 77 розрядів. Однак при цьому 23-бітові мантиси дійсних даних дозволяють працювати з 7-8 значущими цифрами.

Найбільш вживані типи дійсних даних в C/C++ та похідних мовах програмування представлені типами з плаваючою крапкою ***float*** (4 байти) і ***double*** (8 байт). Тип ***float*** забезпечує зберігання додатних та від'ємних чисел модуль яких лежить в діапазоні від 10^{-38} до 10^{+38}. Для 8-байтового формату діапазон істотно розширюється – від 10^{-308} до 10^{+308}, а кількість значущих цифр збільшується до 15-16.

Взагалі кажучи, дати вичерпний опис типів даних доволі складно, оскільки C/C++ передбачена можливість оголошення власних типів за допомогою оператора typedef. Наприклад, розмістивши, на початку файлу програми інструкцію:

```
typedef double real64;
```

отримаємо можливість в усьому коді користуватися типом ***real64*** в якості восьмибітного дійсного типу.

3. Оголошення змінних та констант числових типів

Змінні числового типу, використовувані в програмі, обов'язково повинні бути оголошені до їх використання в тих чи інших виконуваних операторах. На відміну від мови Pascal алгоритмічні мови C/C + + дозволяють вводити такі описи не тільки на початку програмних одиниць (функцій), але і будь-де в коді програми, де це необхідно. Головне правило, – оголошення чого-небудь в програмі повинно передувати його використанню. Неоголошені попередньо змінні (як, наприклад, в Visual Basic) використовувати не можна. Спроба виконати команду

```
sum = 0;
```

без попереднього оголошення змінної ***sum*** приведе до помилки компіляції, що відслідковується синтаксичним аналізатором IDE ще на етапі написання коду.

Оголошенню будь-якої змінної передує службове слово, що визначає діапазон допустимих значень – тип змінної.

```
<Ім'я типу> <Список змінних>;
```

Іменем типу може бути один із згаданих вище типів числових даних або тип, визначений користувачем, за допомогою інструкції ***typedef***, опису структури, переліку чи класу.

```
int m, n, sum;

double x, y;
```

Оголошення змінної можна поєднати з присвоєнням їй початкового значення (ініціалізацією):

```
short p = 333;

int a = p, b = a;
```

Для оголошення іменованих констант зазвичай використовують наступну конструкцію:

```
const <type> <const_name> = <value>;
```

Значення, присвоєне константі при ініціалізації змінювати в подальшому коді заборонено.

Часто використання іменованих констант заміняють, вдаючись до механізму найпростішої макропідстановки:

```
#define Nmax 100

#define eps 1e-6
```

В цьому випадку перед трансляцією програми компілятор (точніше, прекомпілятор) прогляне її текст і всюди, де буде знайдена послідовність символів ***Nmax***, її замінять на число 100, а послідовність символів ***eps*** на число 1e-6. Використання макропідстановки пов'язане з більш помітними витратами часу на етапі компіляції програми, але, на відміну від іменованих констант, не використовує місця в пам'яті, відведеній для роботи програми.

Подібним до механізму використання іменованих констант в C/C++ є оголошення так званих перечислень (enum):

```
enum _digit{ one, two, three };

_digit sh = two;

cout << "Sh = " << sh << endl;
```

В поданому вище фрагменті коду оголошено три послідовні значення ***one=1, two=2, three=3,*** які в подальшому можуть використовуватися як значення для змінних оголошеного типу ***_digit***. Якщо не обумовлюється інше, то значення елементам перечислення присвоюються послідовно, починаючи з 1. Можна задати початок відліку значень:

```
enum _digit{ one = 55, two, three };

_digit sh = two;

cout << "Sh = " << sh << endl;
```

або, за потреби присвоїти конкретні значення:

```
enum _digit{ one = 55, two = 34, three =21 };

_digit sh = two, ph = one;

cout << "Sh = " << sh << endl;
```

При цьому елементи переліків можуть набувати лише цілих значень.

4. Форматований ввід-вивід даних

Форматний вивід числових результатів на стандартний пристрій виводу (stdout), здійснюється за допомогою функції ***printf***. наприклад:

```
# include <stdio.h>
 int main ()
{Int i;
float f;
double d;
printf ("% d% f% lf", i +1, f, f * d);
```

Її аргументи нагадують за формою звернення до функції scanf з тією лише різницею, що список виведення становлять не адреси змінних, а арифметичні вирази, значення яких попередньо будуть підраховані, а потім виведені у відповідності з використаними форматними вказівниками.

Для виводу числових даних використовуються форматні вказівники, загальний вигляд яких містить розширений набір ознак:

%[модифікатори][ширина][.точність][{l|h|L}]{d|i|u|o|x|X|f|e|E|g|G}

Поле ***ширина*** визначає кількість знакомісць на екрані, де буде розміщено виведене значення. Як правило, його задають з деяким запасом, щоб сусідні значення при виводі не накладалися. Коротке число притискається молодшим розрядом до правої межі вказаного поля. Це забезпечує загальноприйнятий спосіб виведення числових таблиць, коли в стовпцях однойменні розряди чисел розташовуються один під одним. Якщо задана ширина менше, ніж це потрібно для виведеного значення, то воно все одно буде виведено повністю.

Поле ***модифікаторів*** може містити до чотирьох керуючих символів з набору [-, +, пробіл, 0, #]. Символ "мінус" встановлює притискання виведеного значення до лівої межі відведеного поля (за замовчуванням до правої). Символ "плюс" встановлює обов'язковий режим виводу знаку числа (навіть якщо воно додатне). Символ "пробіл" встановлює такий режим, при якому замість знака "+" виводиться пробіл. Символ "нуль" встановлює режим виводу чисел, при якому старші незначущі позиції поля виводу заповнюються нулями. Символ # впливає на формат виводу вісімкових, шістнадцяткових та дійсних чисел. При його використанні перед вісімковими числами виводиться початковий нуль, перед шістнадцятковими числами - префікс 0x або 0X.

Різниця між форматами% 0x і% 0X полягає в тому, що в першому випадку шістнадцятковий запис числа формується з малих букв [a, b, c, d, e, f], а в другому випадку - з великих літер [A, B, C, D, E, F].

Для виведення однобайтових цілочисельних даних зі знаком (типу char) можна користуватися одним із наступних форматів -% o,% 0x,% 0X,% i,% d,% ho,% hx,% hX,% hi,% hd. Для виведення однобайтових цілих без знака поряд з перерахованими вказівниками можна використовувати формат% u. Однак слід мати на увазі, що однобайтові значення розширюються до багатобайтових, зберігаючи знак числа. Тому спроба вивести однобайтове значення 127 за форматом % u призведе до правильного результату. Але якщо ми за таким же

форматом виведемо число -5, то результатом буде число 65531 (доповнення до максимального двобайтного числа).

Виведення числових результатів дійсного типу передбачає дві форми відображення - з фіксованою комою (форматні вказівники %f і%lf) або з плаваючою комою (форматні вказівники% е і% Е). Форматні вказівники% g і% G пропонують системі самій вибрати один з цих форматів, який виявиться більш компактним для виведеного значення. Велика чи маленька буква в форматних вказівниках з плаваючою комою призводить до того, що порядку числа передує велика чи маленька буква "е".

При форматному виведенні числових результатів часто супроводять його пояснювальними підписами. Зазвичай текст такого пояснення включають в форматну стрічку. Всі символи, які не є форматними вказівниками виводяться без змін. Наприклад, при виконанні наступних операторів:

```
x1 = 127; x2 = -350;

printf ("x1 =% d x2 =% d", x1, x2);
```

на екрані дисплея з'явиться рядок:

```
x1 = 127 x2 = -350
```

Якщо ми хочемо розташувати виведену інформацію на екрані дисплея з початку наступного рядка, то на початку форматних вказівників зазвичай вставляють керуючий символ "\ n":

```
printf ("\nx1 =% d x2 =% d", x1, x2);
```

Такий же керуючий символ може бути включений в середину або в кінець форматного рядка. За допомогою символу "\" в форматний рядок можуть бути включені й інші керуючі символи (так звані Escape-послідовності), список яких наведено в таблиці.

Символ	Призначення	Символ	Призначення
\a	Звуковий сигнал	\\	Вивід символа \
\b	Витирання попереднього символа	\'	Вивід символа '
\f	Перехід на нову сторінку	\"	Вивід символа "
\n	Перехід на наступний рядок	\?	Вивід символа ?
\r	Повернення до початку стрічки	\0xxx	Вивід символа з вісімковим кодом xxx
\t	Горизонтальна табуляція	\xhh	Вивід символа з шістнадцятковим кодом hh
\v	Вертикальна табуляція	\xHH	Вивід символа з шістнадцятковим кодом HH

Набагато більш складним для початківців програмістів є введення числових даних, за допомогою функції scanf, що використовує аналогічний підхід (за допомогою форматної стрічки):

```
# include <stdio.h>

int main ()

{Int i;

float f;

double d;

.....scanf ("%d %f %lf", & i, & f, & d);
```

Рядок даних, що вводяться надходить зі стандартного пристрою введення (stdin), яким за замовчуванням вважається клавіатура. Завершення набору рядка введення - натискання клавіші Enter.

Перший аргумент функції scanf представляє форматну стрічку, що керує процесом перетворення числових даних, набраних користувачем в рядку введення, в машинний формат, відповідний типам змінних, адреси яких вказані слідом в наступних аргументах. Числові значення в рядку введення рекомендується розділяти одним або декількома пропусками.

У наведеному прикладі змінній ***i*** (в списку введення вказано її адресу - & i), оголошеній за допомогою специфікатора типу int, відповідає форматний вказівник% d. Це означає, що першим числовим значенням в рядку введення може бути тільки ціле десяткове число зі знаком (d - від decimal, десятковий). Вводу дійсної змінної f типу float в форматній стрічці відповідає вказівник %f. Це означає, що друге числове значення в рядку вводу повинно належати діапазону, передбаченому для коротких дійсних даних. Для змінної d типу double використаний форматний вказівник %lf (l - від long).

Як правило, кількість форматних вказівників, перерахованих у першому аргументі функції scanf, має збігатися з кількістю адрес змінних, наступних за форматної рядком. Виняток становить випадок, коли форматний вказівник наказує програмі пропустити чергове значення з введеного рядка. У цьому випадку кількість адрес у списку введення зменшується відповідним чином. Наприклад:

```
scanf ("% d% * l% lf", & i, & d);
```

При виконанні такого оператора введення програма проігнорує друге числове значення, набране користувачем. Звичайно, при ручному наборі значень, безглуздо примушувати користувача набирати дані, які програмі не знадобляться. Але така можливість може виявитися корисною, коли рядок надходить не з клавіатури, а з інших джерел (читається з файла на диска, сформована іншою програмою в оперативній пам'яті).

5. Оператор безумовного переходу

Перед будь-яким виконуваним оператором програми може знаходитися символьна мітка, відокремлена від нього двокрапкою:

```
m5: printf("\nx=%f",x);
```

На позначений таким чином оператор може бути передано управління за допомогою оператора ***goto m5***. Такі переходи допустимі тільки всередині функції. Перехід з однієї функції в іншу оператором goto неприпустимий.

Поширена думка про те, що користуватися цим оператором не слід. Однак це не так. Оператором goto користуватися можна, а от зловживати ним справді не рекомендовано. Бездумне і надмірне використання операторів goto призводить до появи програм важких для розуміння, що мають заплутану логіку і непридатні до модифікації.

Одним з випадків, коли застосування оператора goto призводить до простішого та ефективнішого програмного коду є необхідність у виході з внутрішнього циклу за межі зовнішнього:

```
int i,j;

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

  for(j=0; j<20; j++) {...

    if(умова_виходу) goto mm;

  }

}

mm:
```

Цей фрагмент коду можна переписати без використання goto, але він тоді стане більш заплутаним:

```
int i,j,k=1;

for(i=0; i<n && k; i++) {...

  for(j=0; j<20 && k; j++) {...

    if(умова_виходу) {k=0; break;}

  }

}
```

Застосовуючи оператор goto, варто дотримуватися таких рекомендацій:

- не входити всередину блоку ззовні, особливо якщо на початку цього блоку присутні оголошення локальних змінних з ініціалізацією;
- не входити в цикл ззовні, бо це призведе до неправильної роботи циклу;
- не передавати керування всередину умовного оператора;
- не передавати керування всередину перемикача.

6. Оператор вибору (перемикач)

Оператор вибору розширює можливості умовного оператора if, за допомогою якого можна було організувати розгалуження по двох напрямках: одне в разі істинності перевіряється умови, інше - у випадку хибності. Кілька операторів if дозволяють зробити більшу кількість розгалужень. Наприклад, найпростіший калькулятор можна було б змоделювати таким чином:

```
  cin >>x>>y; //ввід операндів
  cin >> ch;  //ввід символа операції
  if(ch=='+')
{
    z=x+y;
    goto m;
}
  if(ch=='-')
{
    z=x-y;
    goto m;
}

  if(ch=='*')
{
    z=x*y; goto m;
}
  if(ch=='/' && y!=0)
{
    z=x/y; goto m;
}
  cout << "Ця операція не визначена"<<endl;
  z=0;
m:
  cout << "Result="<<z<< endl;
```

Приблизно таку ж функціональність забезпечує фрагмент програми з перемикачем switch:

```
cin >>x>>y; //ввід операндів
cin >> ch;  //ввід символа операції
switch(ch)
{
  case '+': z=x+y; break;
  case '-': z=x-y; break;
  case '*': z=x*y; break;
  case '/': if(y!=0) {z=x/y; break;}
  default : cout << "Ця операція не визначена"<<endl;
    z=0;
}
cout << "Result="<<z<< endl;
```

Після службового слова switch (від англ. - перемикач) в круглих дужках записується вираз, що може бути цілочисельним або символьним. Тіло перемикача завжди є складеним оператором і поміщається у фігурні дужки. За службовим словом case (від англ. - випадок) записується константа, з якою порівнюється значення виразу перемикача. Якщо значення співпадають, то управління передається оператору, що слідує за константою через двокрапку. На його місці може виявитися або один оператор, або ланцюжок операторів. І в тому, і в іншому випадку ці дії повинні завершуватися оператором break, що передає керування оператору, розташованому одразу за перемикачем.

Якщо значення перемикача не співпадає із зазначеною константою, то з тіла перемикача вибирається наступний рядок, що починається з службового слова case, і перевірка продовжується. Завершувати конструкцію оператора може рядок, що починається з службового слова default. На нього управління потрапляє тільки в тому випадку, якщо значення виразу не співпадає з жодною з передбачених констант. У цьому випадку виконуються оператори після слова ***default***, відокремлені від нього двокрапкою. У перемикачі може і не бути частини з позначенням ***default***, тоді в разі неспівпадання значення виразу з жодною з заданих констант перемикач нічого не робить, тобто спрацьовує як порожній оператор.

Відсутність оператора break в якості завершального дії якої-небудь групи приведе до того, що почнуть виконуватися оператори, що належать наступній групі (в цьому випадку ніякої перевірки на збіг з наступною константою вже не відбувається - частина наступного рядка "case c:" просто ігнорується).