

ЛЕКЦІЯ 7

1. ПОКАЖЧИКИ В МОВІ С.
2. ОПЕРАЦІЇ НАД ПОКАЖЧИКАМИ
3. ЗВ'ЯЗОК МАСИВІВ ТА ПОКАЖЧИКІВ

Показчик - це змінна, яка містить адресу деякого об'єкту. Йде справа про адресу в пам'яті комп'ютера. Взагалі-то це просто ціле число. Не можна трактувати показчик як змінну або константу цілого типу.

Якщо змінна буде показчиком, то вона повинна бути відповідним чином оголошена. Показчик оголошується наступним чином:

*тип * <ім'я змінної>;*

В цьому оголошенні тип - деякий тип мови C, що визначає тип об'єкта, на який вказує показчик (адресу якого містить);

* - означає, що наступна за нею змінна є показчиком. наприклад:

*char * ch; int * temp, i, * j; float * pf, f;*

Тут оголошені показчики *ch*, *temp*, *j*, *pf*, змінна *i* типу *int* і змінна *f* типу *float*.

Операції над покажчиками

З покажчиками пов'язані дві спеціальні операції:

& та *.

Обидві ці операції є унарними, тобто мають один операнд, перед якими вони ставляться.

Операція & відповідає операції "взяти адресу".

Операція * визначає "значення, розташоване за вказаною адресою".

Особливість мови C полягає в тому, що знак * відповідає двом операціям, які не мають один до одного ніякого відношення: арифметичній операції множення і операції взяти значення. У той же час сплутати їх в контексті програми неможливо: одна з них унарна, інша - множення - бінарна.

Унарні операції & та * мають найвищий пріоритет нарівні з унарним мінусом.

В оголошенні змінної, що є покажчиком, дуже важливий базовий тип. Як при компіляції визначати, скільки байт пам'яті займає змінна, на яку вказує даний покажчик?

Відповідь проста: з базового типу покажчика. Якщо покажчик має базовий тип *int*, то змінна займає 4 байта, *char* - 1 байт й т.ін. Найпростіші дії з покажчиками ілюструються програмою Прикладу 1.

До покажчиків дозволено застосувати операцію присвоювання. Покажчики одного і того ж типу можуть використовуватися в цій операції, як і будь-які інші змінні. Приклад 2.

```
*main.cpp X
2  using namespace std;
3  int main()
4  {
5  float x=10.1, y; float *pf;
6  pf=&x;
7  y=*pf;
8  cout<<"x="<<x<<"\n";
9  cout<<"y="<<y<<"\n";
10 (*pf)++;
11 cout<<"x="<<x<<"\n";
12 cout<<"y="<<y<<"\n";
13 y=*pf;
14 cout<<"y="<<y<<"\n";
15 y=1+*pf*y;
16 cout<<"x="<<x<<"\n";
17 cout<<"y="<<y<<"\n";
18 return 0;
19 }
```

Приклад 1

```
C:\ "d:\CPP\C EDUCATION 1 sem 1 year"
x=10.1
y=10.1
x=11.1
y=10.1
y=11.1
x=11.1
y=124.21

Process returned 0 (0x0)
Press any key to continue.
```

Приклад 2

main.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x=10;
6      int *p, *g;
7      p=&x;
8      g=p;
9      cout<<"p:"<<p<<"\n";
10     cout<<"g:"<<g<<"\n";
11     cout<<"x="<<x<<"\n";
12     cout<<"*g="<<*g<<"\n";
13
14     return 0;
15 }
```

```
C:\ "d:\CPP\C EDUCATION 1 sem 1 year"
p:0x22ff24
g:0x22ff24
x=10
*g=10
Process returned 0 (0x0)
```

Як і над іншими типами змінних, над покажчиками можливо виконувати арифметичні операції: додавання і віднімання. (Операції ++ і -- є окремими випадками операцій додавання і віднімання.)

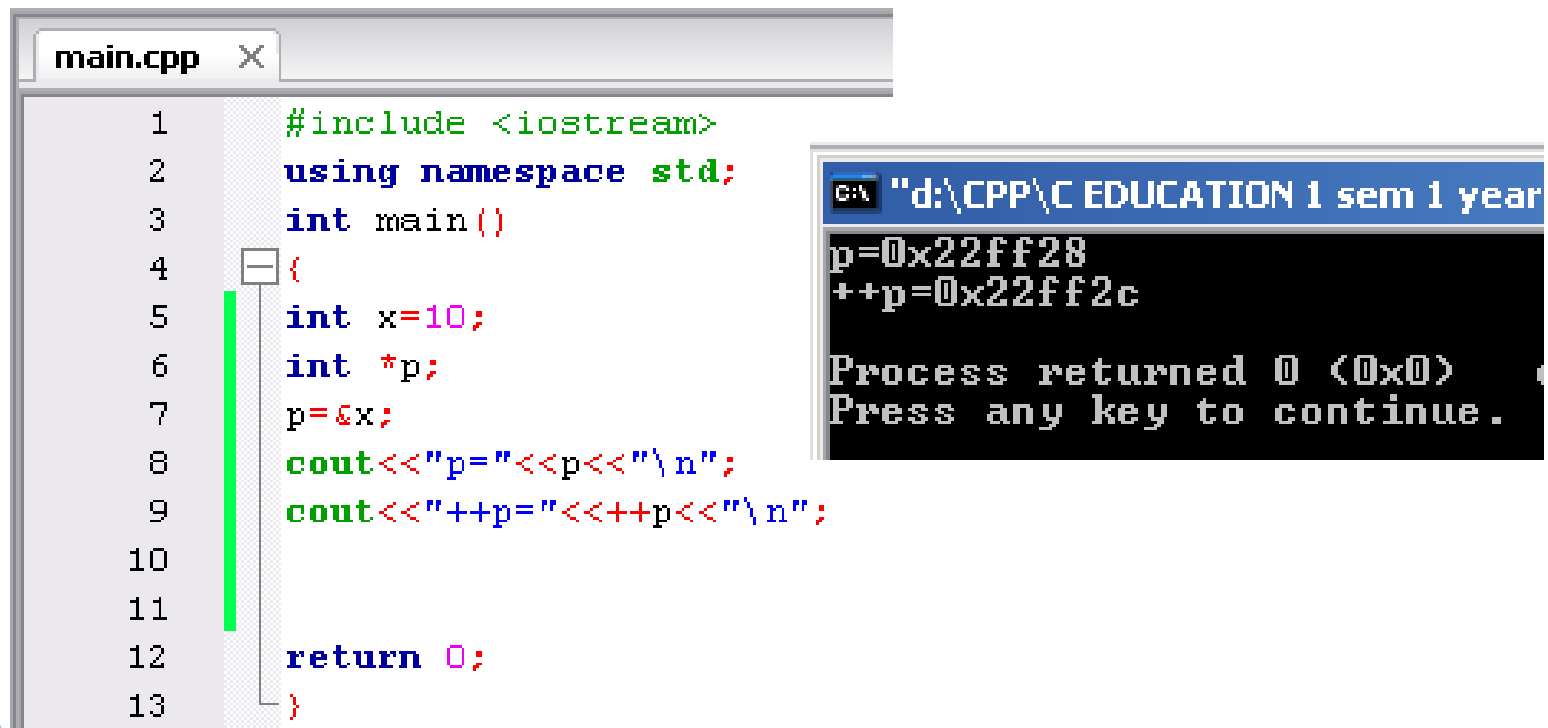
Арифметичні дії над покажчиками мають свої особливості. Вони полягають у змінюванні значень покажчика на величину, пропорційну кількості байтів, що відведено під змінну відповідного базового типу.

Наприклад, при операції ++ значення покажчика буде збільшуватись на кількість байт, займаних змінною базового типу покажчика. Якщо покажчик описаний як *double*, то значення збільшиться на 8, якщо *int* - на 4.

Загальна формула для обчислення значення покажчика після виконання операції $p = p + n$; матиме вигляд

$$\langle P \rangle = \langle p \rangle + n * \langle \text{кільк. байт пам'яті базового типу покажчика} \rangle$$

Приклад 3



```
main.cpp X
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x=10;
6      int *p;
7      p=&x;
8      cout<<"p="<<p<<"\n";
9      cout<<"++p="<<++p<<"\n";
10
11
12      return 0;
13  }
```

```

d:\CPP\C EDUCATION 1 sem 1 year
p=0x22ff28
++p=0x22ff2c

Process returned 0 (0x0)
Press any key to continue.
```


Інші арифметичні операції над покажчиками заборонені!

Наприклад, не можливо скласти два покажчика, помножити покажчик на число й т. ін.

Покажчики можна порівнювати. Можливо застосувати всі 6 операцій: $<$, $>$, $<=$, $>=$, $=$, $!=$.

Порівняння $p < g$ означає, що адреса, що знаходиться в p , менше адреси, що знаходиться в g . Якщо p і g вказують на елементи одного масиву, то індекс елемента, на який вказує p , менше індексу масиву, на який вказує g .

Зв'язок покажчиків і масивів

У мові C існує важливий зв'язок між масивами і покажчиками. Прийнято, що ім'я масиву - це адреса пам'яті, починаючи з якої розташований масив, тобто адреса його першого елементу.

Таким чином, якщо був оголошений масив *int plus [10]*, то *plus* є покажчиком на масив, точніше, на перший елемент масиву. Оператори

pl = plus; i pl = & plus [0];

приведуть до однакового результату. Для того, щоб отримати значення 6-го елемента масиву *plus*, можливо написати

*plus [5] або *(pl + 5).*

Результат буде один і той же. Перевага використання другого варіанту полягає в тому, що арифметичні операції над покажчиками виконуються швидше, якщо працюють з елементами масиву, які йдуть один за одним.

Масиви покажчиків

Покажчики, як і змінні будь-якого іншого типу, можуть об'єднуватися в масиви. Оголошення масиву покажчиків на 10 цілих чисел має вигляд

*int * x [10];*

Кожен з елементів масиву може отримати значення адреси.

Наприклад, для того, щоб третьому елементу призначити адресу цілої змінної *y*:

x [2] = & y;

щоб знайти значення змінної *y*, можливо написати

** x [2].*

Дуже часто масив покажчиків використовується, якщо необхідно мати посилання на стандартний набір рядків.

Наприклад, якщо потрібно зберігати повідомлення про можливі помилки, це зручно зробити так:

```
char * errors [] = { "Cannot open file",  
                    "Cannot close file",  
                    "Allocation error",  
                    "System error" } ;
```

При такому оголошенні рядкові константи будуть занесені в розділ констант в пам'яті, масив покажчиків буде складатися з чотирьох елементів, під які буде виділена пам'ять, і ці елементи будуть ініційовані адресами, що вказують на початок цих малих констант.

Взагалі рядкова константа в мові C асоціюється з адресою початку рядка в пам'яті, тип рядка - *char ** (покажчик на тип *char*). Тому можливо і активно використовується наступне присвоювання:

```
char * pc; pc = "Hello, World!"; cout<<pc; cout<<pc[7];
```

Ініціалізація покажчиків

Після того, як покажчик був оголошений, але до того, як йому було присвоєно якесь значення, покажчик містить невідоме значення. Спроба використовувати покажчик до присвоєння йому якогось значення є помилкою. Вона може порушити роботу не тільки програми, але і операційної системи!

Навіть якщо цього не відбулося, результат роботи програми буде неправильним, знайти помилку буде складно. Прийнято вважати, що покажчик, який вказує в "нікуди", повинен мати значення null, однак і це не робить його "безпечним". Після того, як він потрапить в праву або ліву частину оператора присвоювання, він знову може стати "небезпечним". З іншого боку, нульовий покажчик можна використовувати, наприклад, для позначення кінця масиву покажчиків. Якщо була спроба задати якесь значення тому, на що вказує покажчик з нульовим значенням, система видає попередження *"Null pointer assignment"*.

Поява цього повідомлення є приводом для пошуку використання неініціалізованих покажчиків в програмі.

У мові С можлива також ситуація, коли **показчик вказує на показчик**. В цьому випадку опис буде мати наступний вигляд:

```
int **point;
```

Показчик		Показчик		Змінна
адреса	→	адреса	→	Значення

```
main.cpp X
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int i, *pi, **ppi;
6      i=7;
7      pi=&i;
8      ppi=&pi;
9      cout<<"pi="<<pi<<"\n";
10     cout<<"ppi="<<ppi<<"\n";
11     **ppi=12;
12     cout<<"i="<<i;
13     return 0;
14 }
```

```
C:\ "d:\CPP\C EDUCATIO
pi=0x22ff28
ppi=0x22ff24
i=12
Process returned
Press any key to
```

ДЯКУЮ ЗА УВАГУ!