**Указатели**

Променлива , която съдържа адреса на друга променлива(или обект). Всяка променлива (или друг обект) имат адрес в оперативната памет, където се съхраняват техните стойности, указателите извършват действия само с адресите на клетките от паметта. Когато декларираме указател е необходимо да се определи към какъв тип данни сочи.

Тип \* Име на указател ;

Int \*p;

**Тип** - към какъв тип променлива ще сочи указателя (int,double, float и т.н.) и в зависимост от това колко байта от паметта ще заеме.

Звездичката **\*** ( в литературата я наричат оператор за дереференция) се използва.

Операторът **&** връща адреса на променливата стояща след него.

foo = &myvar;

Това би присвоило адреса на променлива myvar на foo; като предхождаме името на променливата myvar с адрес-на оператора ( &), вече не присвояваме съдържанието на самата променлива foo, а нейния адрес.

1. myvar = 25;
2. foo = &myvar;
3. bar = myvar;

Първо, ние сме присвоили стойността 25 на myvar.

Второто изявление присвоява foo адреса на myvar.

И накрая, се присвоява стойността, съдържаща се в myvar на bar. Това е стандартна операция за присвояване.

1. **Деклариране на указатели**

Поради способността на указателя да се отнася директно към стойността, към която сочи, указателят има различни свойства, когато сочи към char, отколкото когато сочи към int или float. Декларацията на указател трябва да включва типа данни, към който ще сочи указателят.

Декларацията за указатели следва този синтаксис:

**type \* name;**

къде type е типът данни, към който сочи указателят. Този тип не е типът на самия указател, а типът на данните, към които сочи указателят. Например:

1. int \* number;
2. char \* character;
3. double \* decimals;

Това са три декларации на указатели. Всеки от тях е предназначен да сочи към различен тип данни, но всъщност всички те са указатели и всички те вероятно ще заемат еднакво пространство в паметта (размерът в паметта на указател зависи от платформата където програмата работи). Независимо от това данните, към които те сочат, не заемат еднакво пространство, нито са от един и същи тип: първият сочи към int, вторият към char, а последният към double. Следователно, въпреки че тези три примерни променливи са всички указатели, те всъщност имат различни типове:, int\*и char\*, double\*съответно, в зависимост от типа, към който сочат.

Имайте предвид, че звездичката (\*), използвана при деклариране на указател, означава само, че той е указател (той е част от спецификатора на неговия тип).

**Пример:**

1. // my first pointer
2. #include <iostream>
3. using namespace std;
4. int main ()
5. {
6. int firstvalue, secondvalue;
7. int \* mypointer;
8. mypointer = &firstvalue;
9. \*mypointer = 10;
10. mypointer = &secondvalue;
11. \*mypointer = 20;
12. cout << "firstvalue is " << firstvalue << '\n';
13. cout << "secondvalue is " << secondvalue << '\n';
14. return 0;
15. }

Забележете, че firstvalue и secondvaluе, не са директно зададени някакви стойности в програмата, и двете завършват със стойност, зададена индиректно чрез използването на mypointer. Mypointer се присвоява адресът на firstvalue с помощта на &. След това на стойността, посочена от, mypointer се присвоява стойност от 10. Тъй като в този момент mypointer сочи към местоположението на паметта на firstvalue, това всъщност променя стойността на firstvalue.

**Пример:**

1. #include<iostream>
2. using namespace std;
3. int main()
4. {
5. double \*p, r;
6. r = 3.456;
7. p = & r; // p взема адреса на r
8. cout<<\*p<< endl; // извежда 3.456
9. cout<< p<< endl; // извежда адреса на r
10. cout<<& p<< endl; // извежда адреса на указателя p
11. cout<<++p<< endl; // извежда адреса на r увеличен с 8
12. cout<<--p<< endl; // извежда адреса на r
13. cout<<\*p; // извежда 3.456
14. return 0;
15. }

**Пример:**

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. int a=5,\*p;
6. p=&a;
7. cout<< p<< endl;
8. cout<< \*p<< endl;
9. return 0;
10. }

int a=5, \*p; - декларираме променлива а=5 и указател сочещ към целочислена променлива;

p=&a; - това означава, че в указателя р се съдържа адреса на променливата а (нещо от рода 0х22ff14– в шестнадесетична бройна система 0х означава шестн. бр. система ).

cout<< p<< endl; - извежда на екрана адреса на променливата а ;

cout< <\*p; - ще изведе на екрана 5 т.е. компютъра ще отиде на адреса, който се съдържа

в указателя р и от него ще извлече стойността ( на променливата а, която е 5).

1. **Указатели и масиви**

Понятието масиви е свързано с това на указателите. Например, разгледайте тези две декларации:

int myarray [20];

int \* mypointer;

Следната операция за присвояване ще бъде валидна:

mypointer = myarray;

След това mypointer и myarray биха били еквивалентни и биха имали много сходни свойства. Основната разлика е, че на mypointer може да бъде присвоен различен адрес, докато myarray винаги ще представлява един и същ блок от 20 елемента от тип int. Следователно следното задание не би било валидно:

myarray = mypointer;

**Пример:**

1. // more pointers
2. #include <iostream>
3. using namespace std;
4. int main ()
5. {
6. int numbers[5];
7. int \* p;
8. p = numbers; \*p = 10;
9. p++; \*p = 20;
10. p = &numbers[2]; \*p = 30;
11. p = numbers + 3; \*p = 40;
12. p = numbers; \*(p+4) = 50;
13. for (int n=0; n<5; n++)
14. cout << numbers[n] << ", ";
15. return 0;
16. }

Указателите и масивите поддържат един и същ набор от операции, със същото значение и за двете. Основната разлика е, че на указателите могат да се присвояват нови адреси, докато масивите не могат.

**Пример:** Промяна на стойността, посочена от указатели

1. #include <iostream>
2. using namespace std;
3. int main() {
4. int var = 5;
5. int\* pointVar; // store address of var
6. pointVar = &var; // print var
7. cout << "var = " << var << endl; //print \*pointVar
8. cout << "\*pointVar = " << \*pointVar << endl;
9. cout << "Changing value of var to 7:" << endl; //change value of var to 7
10. var = 7; // print var
11. cout << "var = " << var << endl; // print \*pointVar
12. cout << "\*pointVar = " << \*pointVar << endl;
13. cout << "Changing value of \*pointVar to 16:" << endl; // change value of var to 16
14. \*pointVar = 16; // print var
15. cout << "var = " << var << endl; // print \*pointVar
16. cout << "\*pointVar = " << \*pointVar << endl;
17. return 0;
18. }

Задача 1

Напишете програма, която въвежда променлива и покажете същата стойност чрез указател.

Задача 2

Напишете програма, която иска от потребителя да въведе цели числа като входни данни, които да се съхраняват съответно в променливите 'a' и 'b'. Има също и два указателя наречени ptrA и ptrB. Задайте стойностите на 'a' и 'b' съответно на ptrA и ptrB и ги покажете.

Задача 3

Напишете програма, която намира максимума на поредица от N на брой цели числа. Програмата да отпечатва на екрана указател, който сочи към максималната стойност.