Тип указател

1. **Адрес на променлива** – с всяка променлива е свързана стойност – неопределена или константа от типа, от който е тя. Нарича се **rvalue**. Мястото в паметта, в което е записана rvalue се **нарича адрес на променливата** или **lvalue**. По-точно адресът е адреса на първия байт от множеството байтове, отделени за променливата.

Пример: Фрагментът int i = 1024;

дефинира променлива с име i и тип int. Стойността ѝ (rvalue) е 1024. i именува място от паметта (lvalue) с размери 4 байта, като ***lvalue е адреса на първия байт***.

Намирането на адреса на дефинирана променлива става чрез унарния префиксен дясноасоциативен оператор & (амперсанд). Приоритетът му е същия като на унарните оператори +, -, !, ++, -- и др.

1. ***Синтаксис***

&<променлива>

където <променлива> е вече дефинирана променлива.

2. ***Семантика***

Намира адреса на <променлива>

Пример: &i е адреса на променливата i и може да се изведе чрез оператора

cout << &i;

Операторът & не може да се прилага върху константи, изрази и променливи от тип масив, т.е. &100 и &(i+5) са недопустими обръщения. Не е възможно също прилагането му и върху променливи от тип масив, тъй като те имат и смисъла на константни указатели.

Адресите могат да се присвояват на специален тип променливи, наречени променливи от тип указател или само указатели.

2. Задаване на тип указател

Нека T е име или дефиниция на тип. За типа T, **T*** е тип, наречен указател към T.

T се нарича **указван тип** или **тип на указателя**. Всеки указател се свързва с определен тип. Типът на данните определя типа на обекта, който ще бъде адресиран чрез указателя. Например, указател от тип int ще сочи обект от тип int. Съответно, за да сочи обект от тип double указателят трябва да се дефинира от тип double. Паметта, отделена за един указател, има размер, необходим за записване на адрес в паметта. Това означава, че указатели от тип int и указатели от тип double имат обикновено еднакъв размер. Типа, асоцииран с указателя, определя как да бъде интерпретирано съдържанието и каква да е дължината на битовата последователност на този адрес от паметта.

Примери:

```
int *ip1, *ip2;
 char *ucp;
double *dp;
float  fpf, *fp2; //    fp е обект от тип float, а fp2 е указател към променлива от
                        тип float:
```

3. **Множество от стойности -** Състои се от адресите на данните от тип T, дефинирани в програмата, преди използването на T*. Те са константите на типа T*. Освен тях съществува специална константа с име ***NULL***, наречена **нулев указател**. Тя може да бъде свързвана с всеки указател независимо от неговия тип. Тази константа се интерпретира като “сочи към никъде”, а в позиция на предикат е false. **NULL е дефиниран във файла iostream.h.**

Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа T*, се нарича променлива от тип T* или променлива от тип указател към тип T. Дефинира се по стандартния начин.

4. Дефиниция на променлива от тип указател

T* <променлива> [<стойност>];

където

T е име или дефиниция на тип;

<променлива> ::= <идентификатор>

<стойност> е шестнадесетично цяло число, представляващо адрес на данна от тип T или NULL.

Дефиницията T* a, b; ⇔ { T* a; T b;

т.е. само променливата a е указател.

Примери: Дефиницията

```
int *p1, *p2;
```

задава два указателя към тип int, а

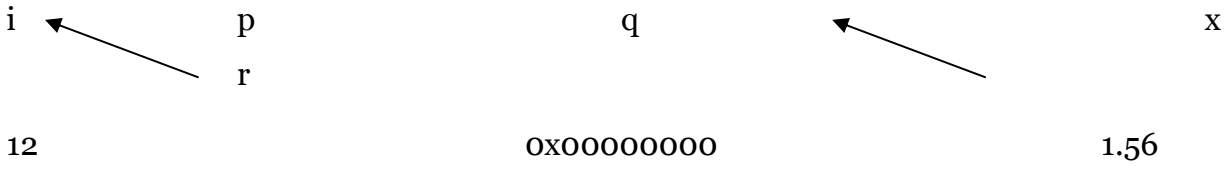
```
int *p1, p2;
```

- указател p1 към int и променлива p2 от тип int.

Пример: Дефинициите

```
int i = 12;
int* p = &i;            // p е инициализирано с адреса на i
double *q = NULL; // q е инициализирано с нулевия указтел
double x = 1.56;
double *r = &x;         // r е инициализирано с адреса на x
```

предизвикват следното разпределение на паметта



Съвет: Всеки указател, който не сочи към конкретен адрес, е добре да се свърже с константата NULL.

5. Операции и вградени функции

a. Извличане на съдържанието на указател-

- *Синтаксис*

*<променлива_от_тип_указател>

- *Семантика*

Извлича стойността на адреса, записан в <променлива_от_тип_ указател>, т.е. съдържанието на <променлива_от_тип_ указател>.

Като използваме дефинициите от примера по-горе, имаме:

*p е 12 // 12 е съдържанието на p

*r е 1.56 // 1.56 е съдържанието на r

Освен, че намира съдържанието на променлива от тип указател, обръщението

*<променлива_от_тип_указател>

е данна от тип T (променлива или константа). Всички операции, допустими за типа T, са допустими и за нея.

Като използваме дефинициите от примера по-горе, *p и *r са цяла и реална променливи, съответно. След изпълнение на операторите за присвояване

*p = 20;

*r = 2.18;

стойността на i се променя на 20, а тази на r – на 2.18.

b. Аритметични и логически операции

Променливите от тип указател могат да участват като операнди в следните аритметични и логически операции +, -, ++, --, ==, !=, >, >=, < и <=. Изпълнението на аритметични операции върху указатели е свързано с някои особености, заради което

аритметиката с указатели се нарича още адресна аритметика. Особеността се изразява в т. нар. мащабиране. Ще го изясним чрез пример.

Да разгледаме фрагмента

```
int *p;
double *q;
...
p = p + 1;
q = q + 1;
```

Операторът p = p + 1; свързва p не със стойността на p, увеличена с 1, а с p + 1*4, където 4 е броя на байтовете, необходими за записване на данна от тип int (p е указател към int). Аналогично, q = q + 1; увеличава стойността на q не с 1, а с 8, тъй като q е указател към double (8 байта са необходими за записване на данна от този тип).

Общото правило е следното: **p + i*sizeof(T)**

c. **Въвеждане -** Не е възможно въвеждане на данни от тип указател чрез оператора cin.

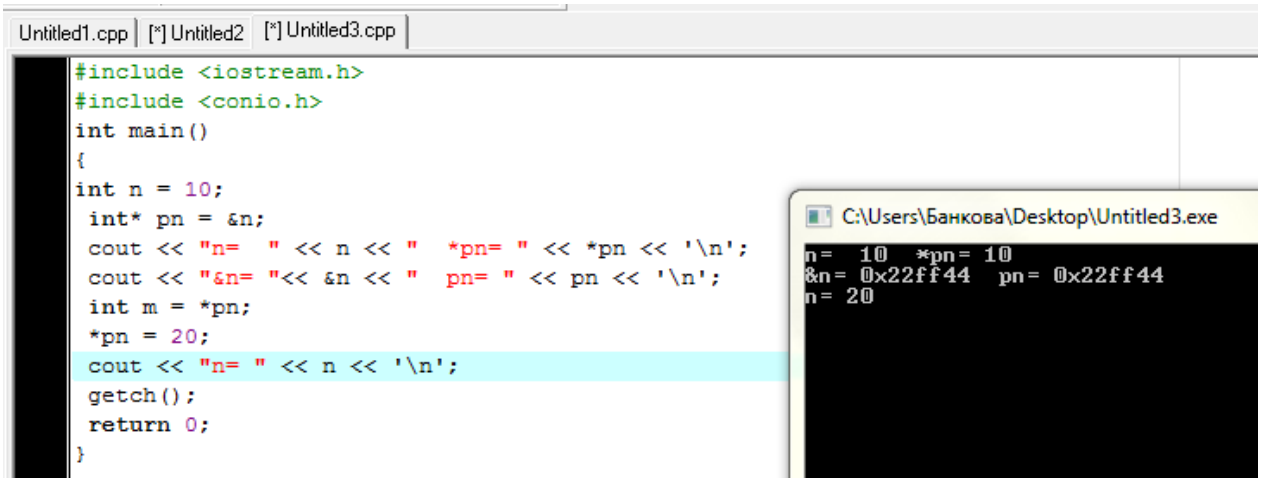
d. **Извеждане -** Осъществява се по стандартния начин - чрез оператора cout.

e. **Допълнение - т**ипът, който се задава в дефиницията на променлива от тип указател, е информация за компилатора относно начина, по който да се интерпретира съдържанието на указателя. В контекста на горния пример *p са четири байта, които ще се интерпретират като цяло число от тип int. Аналогично, *q са осем байта, които ще се интерпретират като реално число от тип double.

Следващата програма илюстрира дефинирането и операциите за работа с указатели.

```
#include <iostream.h>
int main()
{
```

```
int n = 10;          // дефинира и инициализира цяла променлива
 int* pn = &n;       /*дефинира и инициализира указател pn към n показва, че указателят
                     сочи към n*/
 cout << "n=  " << n << "  *pn= " << *pn << '\n';  // показва, че адресът на n е равен на
                                                    стойността на pn */
 cout << "&n= "<< &n << "  pn= " << pn << '\n';  // намиране на стойността на n чрез pn
 int m = *pn;      //  == 10
// промяна на стойността на n чрез pn
 *pn = 20;
 // извеждане на стойността на n
 cout << "n= " << n << '\n'; // n == 20
 return 0;
}
```



В някои случаи е важна стойността на променливата от тип указател (адресът), а не нейното съдържание. Тогава тя се дефинира като указател към тип *void*. Този тип указатели са предвидени с цел една и съща променлива - указател да може в различни моменти да

сочи към данни от различен тип. В този случай, при опит да се използва съдържанието на променливата от тип указател, ще се предизвика грешка. Съдържанието на променлива - указател към тип void може да се извлече само след привеждане на типа на указателя (void*) до типа на съдържанието. Това може да се осъществи чрез операторите за преобразуване на типове.

Пример:

```
int a = 100;
void* p;                    // дефинира указател от тип void
p = &a;                     // инициализира p
cout << *p;                 // грешка
cout << *((int*) p);        // преобразува p в указател към int
                            // и тогава извлича съдържанието му.
```

6. **Указатели и масиви -** имената на масивите са указатели към техните "първи" елементи

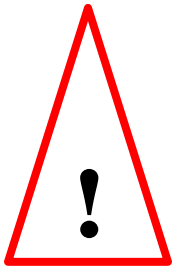
   a. *Указатели и едномерни масиви*

Нека a е масив, дефиниран по следния начин:

```
int a[100];
```

a[0] = = *a

Тогава *(a+i) е друг запис на a[i]



Има обаче една особеност. Имената на масивите са ***константни указатели***. Заради това, някои от аритметичните операции, приложими над указатели, не могат да се приложат над масиви. Такива са ++, -- и присвояването на стойност.

Пример:

```
#include <iostream.h>
int main()
{int a[] = {1, 2, 3, 4, 5, 6};
 for (int i = 0; i <=  5; i++)
     cout << a[i] << '\n';
 for (int i = 0; i <=  5; i++)
     cout << *(a+i) << '\n';
 return 0;
}
```

I начин

II начин

Фрагментът

```
for (i = 0; i <= 5; i++)
{cout << *a << '\n';
 a++;
}
```

a++ дава грешка, защото a е константен указател и не може да бъде променян

Решението е да се ползва помощна променлива - указател

```
int* p = a;
for (i = 0; i <= 5; i++)
{cout << *p << '\n';
 p++;
}
```

Това е възможно, поради факта, че ++ е допустима операция за указател,

Използването на указатели е по-бърз начин за достъп до елементите на масива и заради това се предпочита. Индексираните променливи правят кода по-ясен и разбираем. В процеса на компилация всички конструкции от вида a[i] се преобразуват в *(a+i), т.е. операторът за индексиране [...] се обработва от компилатора чрез адресна аритметика.

Полезно е да отбележим, че операторът [] е лявоасоциативен и с по-висок приоритет от унарните оператори (в частност от оператора за извличане на съдържание *).

b. *Указатели и двумерни масиви* - Името на двумерен масив е константен указател към първия елемент на едномерен масив от константни указатели

Пример:

int a[10][20];

Променливата a е константен указател към първия елемент на едномерния масив a[0], a[1], ..., a[9], като всяко a[i] е константен указател към a[i][0] (i = 0,1, ..., 9), т.е.

a

a[0] ⟶ a[0][0] a[0][1] ... a[0][19]

a[1] ⟶ a[1][0] a[1][1] ... a[1][19] ...

a[9] ⟶ a[9][0] a[9][1] ... a[9][19]

Тогава

****a == a[0][0]**

a[0] == *a a[1] == a[0]+1 ... a[9] == a[0]+9,

т.е.

a[i] == a[0] + i == *a + i

Като използваме, че операторът за индексиране е лявоасоциативен, получаваме:

a[i][j] == (*a + i) [j] == *((*a + i) + j)

3. Указатели и низове

Низовете са масиви от символи. Името на променлива от тип низ е константен указател, както и името на всеки друг масив. Така всичко, което казахме за връзката между масив и указател е в сила и за низ – указател.

Пример:

```
#include <iostream.h>
int main()
{
char str[] = "C++Language"; // str е константен указател
 char* p = str;
 while (*p)
 {
cout << *p << '\n';
  p++;
 }
 return 0;
}
```

```
#include <iostream.h>
 int main()
 {
 char* str = "C++Language";  // str е променлива
  while (*str)
  {
 cout << *str << '\n';
   str++;
  }
  return 0;
 }
```