Указатели

Указателят е променлива, чиято стойност е адрес от паметта.

Синтаксис:

име_на_тип *име_на_указател;

Дефинираният тип за указател трябва да съответства на типа на обекта, към който сочи указателя. Ако това изискване се наруши, компилаторът дава предупреждение. Възможно е чрез явно преобразуване на типовете да се укаже на компилатора, че това се налага и в такъв случай той няма да даде предупреждение.

Примери:

```
int  *p; float *p1; double z, *p2;
```

ВАЖНО! След като един указател е дефиниран той съдържа произволна стойност и може да бъде използват след като бъде инициализиран.

Примери:

```
p =  0; p = NULL;
p =  (int *) 1507;
```

Нулата (или макро NULL от stdio) означава нулев адрес. Няма част от паметта с нулев адрес, той се използва за указване, че указателят не сочи към адрес. Нулевият адрес има и други приложения – може да се използва като знак за край или признак за грешка и др. Изразът (int *) 1507 представлява абсолютен адрес в паметта с номер 1507.

Има две основни едноместни операции за указатели:

- **&операнд -** определя се адресът на операнда. Уточнение – всеки байт в паметта си има адрес. Ако операндът съдържа повече от един байт, тогава адресът му е адресът на най-младшия байт (този който има най-малък адрес). Операндът може да е променлива или елемент от масив.

- ***операнд -** в общият случай операндът е израз, чиято стойност е адрес. Тази операция определя стойността, която е записана на адреса, който е резултат от пресмятането на операнда. В случай, че операндът на * е адрес на променлива, тогава резултатът е променливата, записана в този адрес. И двете операции имат приоритет 2 и са дясно-асоциативни.

Примери:

```
float x, y, *px, *py;
x = 3.14; // x си присвоява 3.14
px = &x; // px си присвоява адреса на x;
y = *px + 10; // y си присвоява x + 10;
py = px; // py си присвоява адреса на x;
*py = 1; //x си присвоява 1;
*px – осигурява косвен достъп към x;
```

x – осигурява пряк достъп към x.

```
float x, *px;
unsigned z;
x = 3.14; px = &x;
z = px; //компилаторът дава предупреждение;
z = (unsigned) px; //компилаторът не дава предупреждение;
```

и в двата случая z ще си присвои стойността на адреса на променливата x.

```
float y, x, *px;
x = 3.14; px = &x;
y = ++*px; //и двете операции имат приоритет 2 и са дясно асоциативни;
```

след изпълнението: x = 4.14, y = 4.14.

```
float y, x, *px;
x = 3.14; px = &x;
y = (*px)--; //и двете операции имат приоритет 2 и са дясно-асоциативни,
затова поставяме скоби
```

след изпълнението: x = 2.14, y = 3.14.

```
float x, y;
int *px, *py;
px = &x; //предупреждение от компилатора, тъй като се разминават типовете на
         //указателя и на променливата
y = *px; //първите два байта на x, интерпретирани като int се присвояват на y
```

Можем да дефинираме указател към константа по следния начин:

```
const  int *ptr;
ptr = &initial;
```

Можем да дефинираме указател константа

```
int  *const ptr1 = &k;
```

в този случай можем да променяме *ptr, което е променливата k, но не можем да променяме адресът, към който сочи ptr.

Можем да дефинираме указател, който е константа и сочи към константа

```
const  int *const ptr3 = &initial;
```

тогава ptr3 не може да се променя адреса, към който сочи ptr3, нито може да се променя *ptr, което е константата initial.