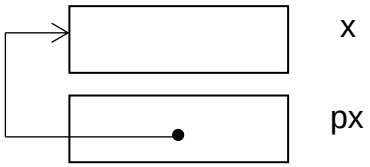
Тема 10
Указатели. Адресна аритметика. Псевдоними

1. Указатели

Всеки елемент от програма (константа, променлива) се съхранява в определени клетки от паметта, които се намират на на определено място в адресното пространство на паметта т.е. всяка данна се съхранява *на определен адрес от паметта*. Съответно, достъп до данната може да се осъществи като се използва нейният адрес. Като стойност, даден адрес може да се съхранява на друго място в паметта. Променлива, чиято стойност е адрес от паметта се нарича ***указател***. Стойността на указателя посочва (указва) местоположението на променливата. Така указателят косвено служи за достъп до променлива, за разлика от познатия, директен достъп, чрез нейното име. Като обобщение на казаното, ***указателят е променлива, чиято стойност е адрес на променлива*** (фиг. 10.1).

Използването на указатели е начин за писане на ефективни и компактни програми. От друга страна, ако не бъдат правилно използвани, указателите могат да доведат до разрушаване на данните.



Фиг. 10.1. Указателят px съдържа адреса на променлива x

Както всяка променлива, и указателят трябва да се дефинира. Общият вид на дефиницията на променлива-указател е:

тип *имеУказател [=стойност];

където:

тип определя типа на съдържанието на указателя т.е. това е типа на променливата, чийто адрес ще сочи указателя.

имеУказател е идентификатора на променливата-указател.

Символ * пред името на променливата определя, че променливата е указател. Обикновено, имената на променливите указатели започват с **p**, което се определя от английския термин **pointer** (указател).

Примери за дефиниране на указатели са:

```
int *px;          // px е указател към променлива от тип int
float *p1;        // p1 е указател към променлива от тип float
double *p2;       // p2 е указател към променлива от тип double
```

Както всяка друга променлива, така и указателят може да бъде инициализиран с дадена начална стойност. Тази стойност трябва да бъде адрес на променлива, както е в посочения пример:

```
int x;
int *px=&x;  // указателят px сочи адреса на променливата x
```

Ако указателят не е инициализиран, неговата стойност е ***NULL***. Тази стойност може да бъде присвоена на указател от всеки тип.

2. Операции с указатели. Адресна аритметика.

За работа с указатели са предвидени два оператора, които се наричат адресни:

- **оператор *&*** определя адреса на операнда;
- **оператор *** определя стойността от посочения адрес.

Форматът на оператор **&** е следния:

&променлива

Освен променлива, операндът може да е и елемент на масив. Като резултат операцията връща адреса на променливата.

Форматът на оператор ***** е следния:

****указател***

Резултат от действието на оператора е стойността на променливата, чийто адрес съдържа указателя.

На променливата-указател може да се присвои стойност на друг указател. Например:

```
int x, *pa;
int *px=&x;
…
pa=px;       // pa съдържа адреса на променливата x
```

Указателите, като операнди, могат да участват и в следните аритметични оператори и логически отношения:

+ събиране

- изваждане

++ инкрементиране

-- декрементиране

= присвояване

== равно

> по-голямо

>= по-голямо или равно

< по-малко

<= по-малко или равно

!= различно

Върху указателите могат да се прилагат по-малко на брой аритметични операции, отколкото върху обикновените променливи. Освен това, изпълнението на аритметичните оператори върху указатели е свързано с някои особености, поради което тяхното изпълнение е известно още като ***адресна аритметика***.

Особеност на адресната аритметика е, че при изпълнението на операторите за *събиране, изваждане, инкрементиране и декрементиране* автоматично се извършва ***мащабиране***, което отчита типа на обектите, към които сочат указателите.

Пример:

```
int x;
int *px=&x;
…
px++;
```

Указателят **px** вече сочи към следващ обект и това е следващата променлива от тип ***int***, чийто адрес е след адреса на **x**. Това означава, че като стойност, адресът **px** е увеличен не с един, а два (или четири) байта.

Общото правило, което отличава *адресната аритметика* от останалите операции е следното: *Ако **p** е указател от тип **type**, то **p+k** се изчислява чрез израза: **p+k*sizeof(type)**.* По този начин, новата стойност на указателя вече сочи следващ елемент от същия тип.

Адресната аритметика се прилага най-вече при работа с масиви, тъй като те представляват поредица от еднотипни елементи.

3. Указатели и масиви

В C/C++ съществува пряка връзка между указатели и масиви - ***имената на масивите се явяват указатели***, които съдържат адреса на първия елемент от масива т.е. на елемента с индекс 0. По този начин, достъпът до елемент на масив може да се осъществи и чрез указател.

Например, нека са дефинирани масив от 5 целочислени елемента и указател, който в последствие да получи адреса на масива:

```
int array[5];
int *p;
int x,y,i;
...
// променливите x, y осъществяват достъп до един и същ елемент
x=array[0];
y=*array;
```

```
...
// променливите x, y осъществяват достъп до елемент с индекс i
x=array[i];
y=*(array+i);
...
p=array+i;   // също, достъп до елемент с индекс i
y=*p
```

Основната разлика между името на масива и променливата-указател е, че *името на масива се явява константа* и стойността му не може да бъде променена. Тази особеност се илюстрира чрез следните примери:

```
p=array;            // изразите са допустими
p++;


array=p;            // изразите са недопустими
array++;
array=&x;
```

Достъпът до елементите на масив може да се осъществи както чрез индекс, така и чрез указател. Достъпът чрез указател е много по-бърз, отколкото този чрез индекс и поради тази причина често е предпочитан в програмите на C/C++.

Пример за достъп до елементи на масив чрез указател е програмния код, който намира сумата на елементите на масив в задачата за намиране на средноаритметична стойност от елементи на едномерен масив:

```
#include <stdio.h>
main()
{
      double mB[10];
      int i;
      double suma, average;
      for(i=0;i<10;i++)
      {
            printf("Item[%d]=",i);
            scanf("%lf",&mB[i]);
      }
      suma = 0;
      double *p;
      for(p=mB; p<(mB+10); p++)
            suma += *p; // достъп до елемент на масив чрез указател
      average = suma/10;
      printf("\nArerage = %lf",average);

      return 0;
}
```

4. Указатели към тип void

Типът на указателя, определен в дефиницията, дава информация на компилатора относно начина, по който се обработва съдържанието на указателя. Например, ако **p** е указател към тип ***int***, то съдържанието на **p** е 2 (или 4) байта. Ако, **p** е указател към тип ***float***, то неговото съдържание е 4 байта.

Когато типът на данните, към които сочи даден указател е без значение, указателят се дефинира от тип **void**. В случая е важно, каква е стойността на указателя, а не какво е неговото съдържание.

Дефиницията на указател от тип ***void*** се извършва по обичайния начин:

void *pv; // в случая pv е указател към тип void

Указателите от тип ***void*** са предвидени с цел едни и същи указатели да сочат към данни от различен тип.

Съдържанието на указател от тип ***void*** не може да се извлече пряко. При опит за пряко извличане, компилаторът издава съобщение за грешка. За да се изведе съдържанието на указател от тип ***void***, се извършва явно или неявно преобразуване на тип.

Неявното преобразуване е присвояване на указателя от тип ***void*** на указател от друг тип. Например:

```
int num;
int *p=&num;
void *pv;

pv=p;        // типът на pv се преобразува към (int *)
cout <<*pv   // извежда се стойността на променливата num
```

Явното преобразуване е чрез оператор ***typecast***. Например:

```
int num;
void *pv;

pv=&num;
cout <<*((int *)pv);  // типът на pv се преобразува явно към (int *)
```

5. Указатели към указатели

Тъй като указателят също е променлива, която се разполага на определен адрес, възможно е, друг указател да сочи неговият адрес. Такъв тип променлива се нарича ***двоен указател*** или ***указател към указател***.

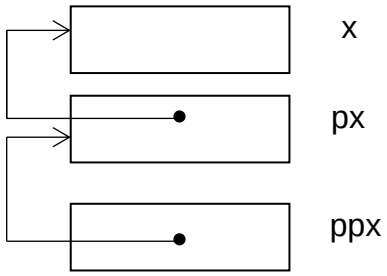
Дефинирането на двоен указател се извършва по следният начин:

тип **име[=стойност];

Например, нека **x** е променлива; **px** - указател към променливата и нека **ppx** да е указател, съдържащ адреса на **px**. Дефинирането на тези променливи е по следният начин:

```
int x;
int *px=&x;
int **ppx=&px;
```

Примерът е илюстриран на фиг. 10.2.



*Фиг. 10.2. Указателят **px** сочи променлива **x***

Дефиниран по този начин, указателят **ppx** съдържа адреса на указателя **px** и се явява ***двоен указател***. Типът пред двойният указател е типът на променливата x, която се сочи от указател px.

Приложението на двойните указатели е при работата с различни структури от данни, като дървета, списъци, графи, както и при работа с многомерни масиви.

6. Псевдоними

В език C++, за разлика от програмен език C, е въведен тип псевдоним (**reference**). Псевдонимът се явява друго име на вече дефинирана променлива. Общият вид на дефиницията на псевдоним е следната:

тип &имеПсевдоним = променлива;

В случая инициализацията е задължителна. При дефинирането на псевдоним трябва да се посочи на коя променлива той ще бъде свързан, тъй като псевдоним се асоциира с определена памет.

Дефинирането на псевдоними може да се илюстрира чрез следния пример:

```
int x;
int &y=x;
```

В този случай променливата **x** има второ име и това е псевдонима **y** т.е. **x** и **y** са две имена на една и съща променлива, тъй като и псевдонимът, и инициализаторът са от един и същи тип. Всяка операция, която се приложи на променливата **x** ще се отрази и на псевдонима **y** и обратно. Възможно е, обаче псевдонима и инициализатора да са от различни типове. В този случай компилаторът създава две отделни променливи и се

стреми да конвертира типа на инициализатора. Съответно, операциите върху инициализатора не се отразяват на псевдонима и обратно.

Относно въпроси по темата на адрес: ln_zh_st@yahoo.com