КАТЕДРА: КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ
ДИСЦИПЛИНА: АЛГОРИТМИ СТРУКТУРИ ОТ ДАННИ
ДИСЦИПЛИНА: СИНТЕЗ И АНАЛИЗ НА АЛГОРИТМИ

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 2

ТЕМА: Указатели. Работа с указатели. (Преговор)

ЦЕЛ:

Целта на упражнението е студентите да преговорят и упражнят своите познания при работата с указатели. След упражнението студентите би следвало да нямат проблеми да боравят с тях.

I. ТЕОРЕТИЧНА ЧАСТ

1. Въведение.

Всеки елемент от програма (константа, променлива) се съхранява в определени клетки от паметта, които се намират на определено място в адресното пространство на паметта т.е. всяка една от данните се съхраняват на определен адрес от паметта. Съответно, достъп до тях 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.1).

Фиг.1. Указателя px съдържа адреса на променлива x.

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

тип *име_указател [=стойност];

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

Име_указател е идентификатора на променливата указател. Символ * пред името на променливата определя, че променливата е указател.

Както всяка друга променлива, така и указателят може да бъде инициализиран с дадена начална стойност. Тази стойност трябва да бъде адрес на променлива.

Примери:

```
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
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

Съществуват две специални операции с указатели, които позволяват тяхното ефективно използване:

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

Форматът на операциите & и * е следния:

&променлива

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

****указател***

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

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

Пример:

```
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 операции върху указатели е свързано с някои особености, поради което тези операции са известни още като адресна аритметика.

Особеност на адресната аритметика е, че при изпълнението на операциите 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 елемент

3. Указатели и масиви

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

Пример:

```
int array[5];
int *p;
int x,y,i;

x=array[0];   // променливите x, y осъществяват достъп до един и същ елемент
y=*array;

x=array[i];    // променливите x, y осъществяват достъп до елемент с индекс 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.

Пример:

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

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

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++

II. ПРАКТИЧЕСКА ЧАСТ

В практическата част са показани примери за използването на указатели и масиви като структури от данни.

ЗАДАЧА1: *Отговорете на следните контролни въпроси. (Коментирайте въпросите с преподавателя)*

1. Какво се дефинира със следния програмен ред?
   int px, py, pz;*
2. Има ли разлика ако дефиницията е записана *int *px, py, pz;* ?
3. Какво ще се изведе на екрана след изпълнението на програмен код:
   int px, py;*
   py=10;
   px=&py;
   *++*px;*
   Cout<<py;
4. Какво ще се изведе на екрана, ако програмен ред 4 се замени с програмен ред: **++px*

ЗАДАЧА2: *Да се напише сорс код който сумира елементите на целочислен едномерен масив от 10 елемента. Достъпа до елементите на масива да се реализира чрез указатели.*

КОД:

```
#include main()
{
        int mas[10],*p;
        int i,sum;
        p=mas;
        for(i=0;i<10;i++)
        {       cout<<"Insert the elements of the massive=";
                cin>>*p;                // достъпът де текущия елемент е чрез указател
                                        //scanf ("%d",p);
                p++;
        }
        p=mas;
        sum=0;
        for (i=0;i<10;i++)
        {       sum+=*p; // достъпът де текущия елемент е чрез указател
                p++;
        }
        Cout<<"sum="<<sum;
        return 0;
}
```

ПОЯСНЕНИЕ

Чрез първия for цикъл, потребителя вмъква елементите на масива. Като достъпа до всеки елемент се осъществява чрез указател. Чрез втория for цикъл се изчислява сумата на елементите на масива, като отново елементите се извличат чрез указателя.

III. Задача за самостоятелна работа.

1. Тествайте задача 2 от практическата част.
2. Да се осъществи размяна на две целочислени променливи, като достъпът се осъществява чрез указатели.

3. Да се създаде конзолно приложение, с което се дефинира едномерен масив от 10 целочислени елемента. Да се въведат стойностите на елементите и да се изведат като се използват два начина: достъп чрез индекс посредством оператор [], достъп чрез указател
4. Да се състави конзолно приложение за търсене на най-голям и най-малък елемент в едномерен масив от 10 реални елемента. Достъпът до елементите на масива да се осъществи чрез указатели.