Лабораторно упражнение № 6
Указатели. Адресна аритметика

I. Теоретична обосновка

Всеки елемент от програма (константа, променлива) се съхранява в определени клетки от паметта или, както е прието да се казва: на определен адрес от паметта.

Променлива, чиято стойност е адрес от паметта се нарича ***указател***. Стойността на указателя посочва (указва) местоположението на дадена променлива и косвено служи за достъп до нея, за разлика от познатия, директен достъп до променливата, чрез името ѝ.

Като обобщение на казаното, ***указателят е променлива, чиято стойност е адрес на променлива.***

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. Обикновено, имената на променливите указатели започват с **p**, от английският термин **pointer** (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.

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.

4. 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. Например:

```
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++.

Пример за достъп до елементите на масив чрез указател е даден с програмната реализация на задачата за сумиране на елементите на едномерен масив mas от 10 целочислени елемента. Указателят p сочи текущият елемент от масива.

```
// намиране ма сумата от елементите на едномерен масив
#include <stdio.h>
main()
{ int mas[10],*p;
  int i,sum;
    p=mas;
   for(i=0;i<10;i++)
    { printf ("insert the elements of the masive=");
      scanf ("%d",p);          // достъпът де текущия елемент е чрез указател
```

```
      p++;
    }
   p=mas;
   sum=0;
     for (i=0;i<10;i++)
      { sum+=*p; // достъпът де текущия елемент е чрез указател
        p++;
       }
   printf ("sum=%d",sum);
   return 0;
}
```

II. Контролни въпроси

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. Напишете програмен ред, с който дефинирате 3 указателя към тип int.

4. Какво ще се изведе на екрана след изпълнението на програмен код:

```
int* px, py;
py=10;
px=&py;
++*px;
printf("%d\n", py);
```

5. Какво ще се изведе на екрана, ако програмен ред 4 се замени с програмен ред: `*++px;`

III. Задачи за изпълнение

1. Да се за размяна на две числа, като достъпът се осъществява чрез указатели.

2. 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 три начина: достъп чрез индекс посредством оператор [], достъп чрез индекс посредством адресен оператор *, достъп чрез указател

3. 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.

Лабораторно упражнение № 7
Програмни модули в C/C++. Функции

I. Теоретична обосновка

1. Видове програмни модули

В програмните езици, за видовете програмни модули се използва следната класификация:

- главна програма;
- подпрограми (процедури, функции).

Главна програма е програмен модул, на който операционната система предава управлението при стартиране на изпълнимия код. При изпълнението си, главната програма може да предаде управлението на други програмни модули, наречени ***подпрограми***. След завършване, главната програма връща управлението на операционната система.

Подпрограма е програмен модул, който получава управлението от друг програмен модул т.е. подпрограмата се извиква от друг модул. След приключване, подпрограмата връща управлението в програмния модул, от където е извикана.

За разлика от други програмни езици, където подпрограмите се разделят на процедури и функции, в C и C++ съществува само един тип програмни модули - ***функции***.

Функция е самостоятелен фрагмент от програма, съдържащ описания на променливи и набор оператори на езика. Фрагментът е затворен във скоби {…} и в крайна сметка се изпълнява като един обобщен оператор.

Всяка програма на C или C++ се състои от една или повече функции. Задължително трябва да съществува една главна функция с име ***main().*** Функция ***main()*** изпълнява роля на главна програма. Това е функцията, към която първоначално се предава управлението от операционната система. След приключване на изпълнението, функция ***main()*** връща управлението на операционната система.

Като самостоятелен програмен модул, функцията може да бъде извиквана многократното, като се стартира с различни входни данни (променливи). Входните променливи се наричат още ***параметри*** на функцията.

След завършването си функцията връща ***резултат***. Като програмен модул функцията връща чрез името си, един резултат.

Параметрите и ***резултатът*** са връзката на функцията с останалите програмни модули. Идея за използването на функции е те да могат да се извикват с различни входни данни.

2. Дефиниране на функции. Оператор return

Дефиниция на функция представлява цялостното описание на функцията. Описанието на функцията се състои две части:

- **заглавна част (*прототип*);**
- **тяло,** което съдържа декларации на локални променливи и оператори.

Общият вид на дефиницията на функция е следният:

```
тип  име (списък формални параметри)        // прототип на функция
{                                            // тяло на функция
   // описание на локални променливи
   // оператори;
```

}

Тип на функцията е типът на стойността, която функцията ще върне като резултат. По подразбиране функциите са от тип ***int*** т.е. ако не бъде записан тип, счита се, че функцията е от тип ***int***. Чрез името си, функцията връща един резултат.

Име на функцията е идентификатор, чрез който еднозначно се определя функцията. **Формални параметри** са списък входни променливи, чрез който се осъществява връзката между функцията и останалите функции. Те имат описателно значение.

Списък формални параметри се представя в следния вид:
(тип1 променлива1 , тип2 променлива2, ...)

Тялото на функцията включва множество декларации на локални променливи и оператори, реализиращи задачата на функцията.

3. Извикване на функция. Предаване на параметри

Извикването на функция става чрез името и. При извикване на функцията, в скобите след името се задават ***фактическите параметри*.** Извикването на функция се задава като:

име (списък фактически параметри);

Подобно на формалните, фактическите параметри са разделени със запетаи.

Фактическите или **действителни** параметри са константи, променливи или изрази, които при извикване на функцията предават своите стойности на формалните параметри, за да се изпълни функцията. Този процес се нарича ***свързване на формалните с фактическите параметри***.

Фактическите и формалните параметри си съответстват по брой и по тип.

4. Декларации на функции

В случай, че се наложи функцията да бъде извикана преди нейната дефиниция, необходимо е, функцията да бъде декларирана.

Декларацията на функцията е нейният прототип (заглавната част на функцията) последван от символ ***;***

5. Примери за реализация на функции

5.1. Намиране на N факториел

// намиране на N факториел чрез използване на функция

```
#include <stdio.h>
int Factoriel(int n);


main()
{
      int n,fact;
      printf("n=");
      scanf("%d",&n);
      fact = Factoriel(n);
      printf("N Factoriel = %d",fact);
      return 0;
```

```
}


int Factoriel(int n)
{
      int nf=1,i;
      for(i=1;i<=n;i++)
            nf*=i;
      return nf;
}
```

II. Задачи за изпълнение

1. Да се създаде конзолно приложение, което пресмята лицата на различни фигури: кръг, правоъгълник, триъгълник. Пресмятането на лицата на фигурите да се извършва с отделни функции. В проекта да се включи и функция, която проверява дали стойностите за страните на триъгълника са валидни.

2. Да се дефинира функция, която намира разстояние между две точки, координатите на който се предават като параметри. Да се създаде конзолно приложение, което намира разстояние между обща дължина на начупена линия.

Упътване: Координатите на точките да се дефинират като два отделни масива – първият да съдържа x- координатите, а втория – y-координатите.

3. Да се състави конзолно приложение за намиране на e^x в ред на Фурие. Натрупването на междинната сума да се прекрати, когато поредния член стане по-малък от 10^{-8} . Формулата за пресмятане е: $e^x=1+x/1+x^2/2!+x^3/3!+...+x^n/n!$

Да се сравни получената стойност със стойността, получена чрез използване на функция exp().

Упътване: Да се използва функция за намиране стойността на n!.