Лабораторно упражнение № 7

Указатели и масиви(агрегатен тип данни) в езика С

времетраене: 4 уч. часа

7.1. Понятие за указател

Указателят е особен тип програмен обект, който се използува за описание на данни, служещи за съхраняване на адреси на други данни или програмни обекти. Описанието му в С-програмни модули се реализира във

формат:

тип * идентификатор

където: ***тип*** - тип на програмен обект;
идентификатор - име на указателя.

Забележки:⇩

- Достъпът до указателя, подобно на другите типове данни, се осъществява посредством името (идентификатора) му.
- Стойността, която се съхранява в паметта отделена за указателя, винаги е адрес на друга променлива (програмен обект).
- Типът, с който се дефинира даден указател, се определя от типа на обекта, чийто адрес той съхранява.
  - Вътрешното представяне на указателя зависи от модела на паметта и е цяло число без знак. Стойността, съхраняваща се в указателя е реален адрес от оперативната памет и може да заема стойности от 0 (нула) до някаква положителна стойност, зависеща от размера на наличната ОП.
- Особен тип указател е void. Той описва указател, несъдържащ информация за типа на данните (по-точно за размера им), чиито адреси съхранява. Указателите от тип **void** могат да приемат стойности от всеки друг тип, без да е необходимо явно преобразуване.

7.2. Адресна аритметика

В езика **С** са разрешени и аритметични операции с указатели: присвояване; събиране и изваждане с константа; разлика на два указателя.

Адресната аритметика в езика С има една важна особеност - при реализиране на аритметични операции с указатели, при които единият операнд е константа се наблюдава т.нар. ефект на мащабиране изразяващ се с формулата:

Операциите: ptr = ptr + *константа*; **или** ptr = ptr - *константа*
се изпълняват съответно като:
нова стойност на ptr := *стойност на* ptr + *константа* * sizeof (*тип ptr*) **или**
нова стойност на ptr := *стойност на* ptr - *константа* * sizeof (*тип ptr*)

където:	
константа	- аритметична константа;
***стойност на* ptr**	- съдържание на указателя;
тип ptr	- базов или производен тип, с който е дефиниран указателя.

- При задаване на такава операция компилаторът премества указателя напред или назад на такова разстояние, че в него да могат да се поместят зададеният от *константа* брой обекти от типа на указателя.
- Разликата между два указателя е цяло беззнаково число равно на броя обекти от типа на указателя, които могат да се разположат в паметта между адресите съхранявани от указателите.

7.3. Масиви

7.3.1 Понятие за масив

Масивите са *статични структури, които могат да се определят като подредено множество от еднотипни обекти, носещи общо име. Достъпът до обектите се осъществява чрез механизма на индексацията.*

Това определение характеризира масива със следните свойства:

- Той има краен брой еднотипни елементи, чиито тип е от типа на дефинирания масив. Например **int a [10]**; е масив от десет елемента, чиито тип е ***int*** и неговия размер не може да се променя в областта, в която е дефиниран;
- Името на всеки елемент се получава от името на масива с прилагане на операция индексация (**[ ]**). Това е бинарна операция. Първият операнд е името на масива, а вторият е израз от цял тип, наречен **индекс,** задаващ номера на елемента. Например: **a[0]**, **a[к+7], a[k*i]**;
- Масивът е подредено крайно множество, като първият елемент е с индекс 0 - т.е. **a[0]**, а след него са подредени елементите **a[1]**, **a[2]**, **a[3]** и т.н. до последния елемент **a[9]**).

7.3.2 Дефиниции на масиви

Съгласно синтаксиса на езика **C**, признак за описанието на масив е наличието на квадратни скоби **[ ]** след дефиницията на съответния идентификатор. Инициализцията на масивите може да се извърщи по два начина:

- Явна инициализация - непосредствено след декларацията на масива, след знак за присвояване (**=**), във фигурни скоби **{ }** се записва списък от стойности, присвоявани на съответните му елементи;
- Инициализация по премълчаване - ако не се зададе явна инициализация, компилаторът генерира код, с който занулява всички статични променливи.
- Останалите масиви (които не са статични или външни) не могат да се инициализират. Елементите им получават случайни стойности.

пример P_7.3.2 ⇨ дефиниции и инициализация на едномерни масиви в C ⇩

```
/* ------------------------------------------------------------------------------------ */
```

```
int a[10];      /*  int масив с 10 елемента от a[0] до a[9] */
char t1[15]={'П' ,'Р', 'И ' ,'М', 'Е' , 'Р', ' -', '>', '6',  '.',  '2' };
        /*  символният масив t1 се инициализира като t1[0]:='П'; t1[1]:='Р'
        /*  ...t1[10]:='2'; а  елементите  от  t1[11] до t1[14]  имат неустановени     */
        /*  стойности                */
static float m[5];  /*  всички елементи се инициализират с 0.0        */
int k[5]={1, 1, 1};
        /* елементите k[0] = k[1] = k[2] =1,  а k[3] и  k[4]  са с неопределени        */
        /* стойности        */
/* ---------------------------------------------------------------------------------          */
```

7.3.3 Размерност на масиви

В езика **C** се поддържат и многомерни масиви, като **размерността** (***dimension***) се определя от броя на двойките скоби следващи идентификатора на масива.

✍пример P_7.3.3 ⇨ дефиниции и инициализация на многомерни масиви в C ⇩

```
/* ---------------------------------------------------------------------------------          */
int d[3][4];       /* двумерен масив - три реда по 4 елемента в ред  */
char z[3][3]={ 1, 2, 3, 4, 5, 6,  7, 8, 9};
/* дефиниция с инициализация на двумерен масив от тип char    */
/*  z[0][0]:=1;  z[0][2]:=3  .... z[2][2]:=9 */
char text[3][21]={"текст ред първи", "текст ред втори", "текст ред трети" };
int u[ ][3] ={ {0},
{1,2},
{4,5,6}
};
/* всеки ред се инициализира  частично ( вътрешните скоби)       */
/* u[0][0]:=0;   u[1][0]:=1; u[1][1]:=2;  u[2][0]:=4; u[2][1]:=5 и      */
/* u [2][2]:=6; останалите елементи са неопределени; */
/* ---------------------------------------------------------------------------------          */
```

🔔 Забележки:⇩

- В ОП масивите съгласно конвенциите на C се съхраняват в нарастващ порядък на най-десния им индекс (този начин често се нарича съхранение на масивите по редове).
- При инициализация на двумерни символни масиви – в примера масива **text** е дефиниран като двумерен състоящ се от 3 реда с по 21 символа във всеки ред. Същият се инициализира със записаните литерални константи, но в края на всеки ред се поставя символа NULL (ASCII код 0) - т.е. редовете се съхраняват като ASCIIZ последователност, т.е. **text[0][15]= text[1][15]= text[2][15]=NULL**;
- Разрешена е частична инициализация на многомерните масиви – (P_7.3.3 - **int u[ ][3]**), като не се задават всички елементи. Размерите на всички индекси без първия задължително се задават. Размерът на първия може да се пропусне - компилатора го определя по броя на вътрешните скоби.
- В езика C с цел повишаване производителността на генерирания изпълним код, не се осъществява **КОНТРОЛ ЗА ДОПУСТИМОСТ НА ИНДЕКСНИТЕ СТОЙНОСТИ** в границите на дефинирания масив - това е грижа на **ПРОГРАМИСТА**. Ако в масив **int t[5];** се реализира достъп от вида t**[10]**, грешката няма да се открие от компилатора.

7.4. Взаимовръзка между масиви и указатели

7.4.1. Едномерни масиви и указатели

Наред с т.нар. ***класически начин*** за достъп до елементите на масива - индексирането, в езика **C** се реализира и достъп до структурните данни, каквито са масивите, и чрез използуване механизма на указателите.

Поради факта, че масивът е подредена съвкупност от еднотипни елементи, то достъпът може да се осъществи и чрез указател, който е инициализиран с адреса на първия елемент на масива. Използва се операцията * (извличане на стойност по съхранявания в указателя адрес). В **C** за едномерните масиви важи правилото, че идентификаторът на едномерния масив е указател към първия му елемент. С други думи за едномерния масив **ar[N]** са в сила тъждествата:

ar==&ar[0] **и** **ar+i==&ar[i]**

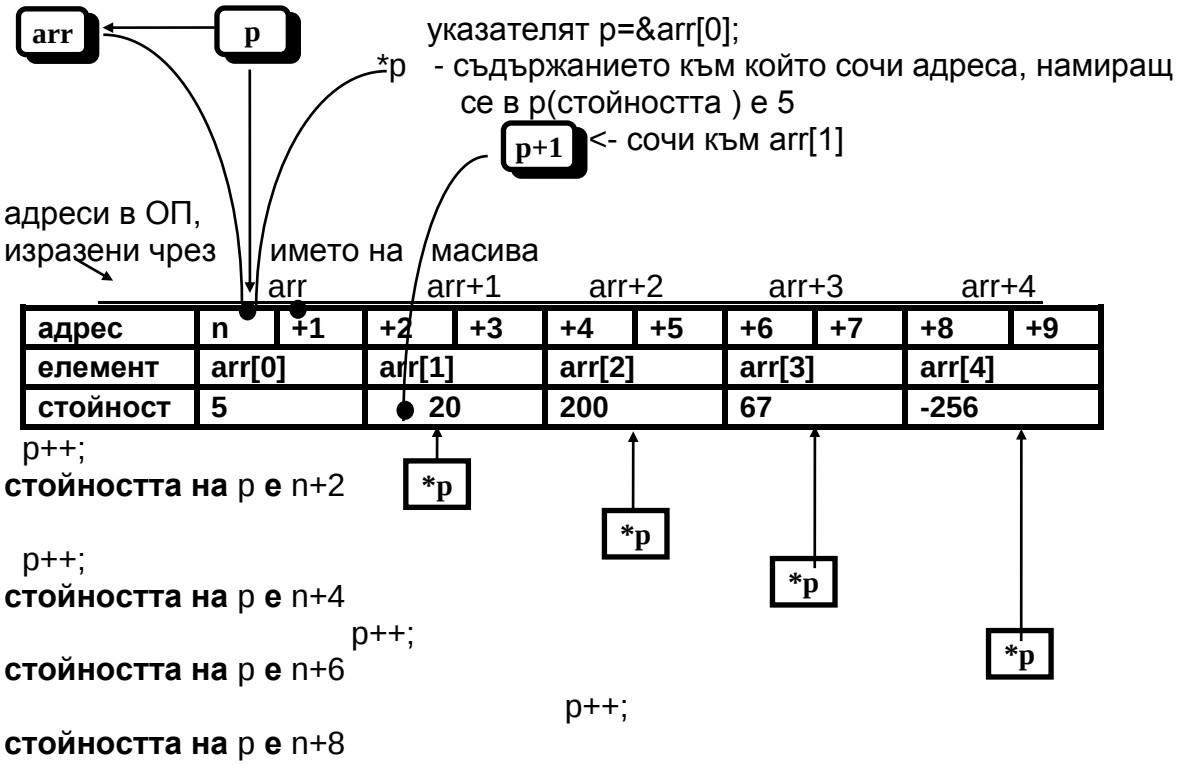
Обръщенията към стойностите, съхранявани в масива **ar** могат да се запишат по двата способа съответно:

ar[i] **и** ***(ar+i),**

които са напълно равностойни.

На ***фиг. 6-1*** схематично и текстово е описано съответствието между ***int*** едномерен масив и ***int*** указател. Нека са дефинирани:

```
int arr[5]={5,20,200,67,-256};
int *p=arr;
/*дефинира се указател p,  които се инициализира (зарежда) с адреса на първия елемент*/
/* на масива arr, което е равносилно на p=&arr[0]      */
```



фиг. 7.2.1-1

След инициализацията на указателя **p=arr** той сочи към първия елемент на масива (**arr[0]**) и операцията ***p** дава резултат **5**. При инкрементиране на указателя (**p++**) той сочи вече към втория елемент на масива (**arr[1]**), така че сега ***p** дава стойността **20** и т.н. Втори начин за достъп до стойностите на елементите на масива е възможен, чрез използуване на адресни операции, например ***(arr+2)** дава резултат **200**, т.е. стойността на третия елемент на масива (**arr[2]**).

☝ 🔔 Забележки:⇩

- В езика С операцията индексиране exp1[exp2] се дефинира и реализира като
- *((exp1) + (exp2))
- където exp1 е от тип указател, а exp2 е от цял тип.
- Тъй като операцията индексиране *[ ]* се реализира чрез механизма на указателите, то не е възможна ефективната проверка за излизане извън рамките на дефинираните масиви.
- Не е възможно използуването на операцията arr++ защото името на масива е константен указател към първия елемент и компилаторът не позволява промяна на съдържанието му (изместване на началото на масива), тъй като вече е заделил точно определен сегмент от паметта за него при дефиницията му.
- Поради същата причина опитът за сравнение на два масива чрез техните имена, например:

  int a[10], b[10];
  ...
  if(a == b) ...

- ще дава винаги резултат **0** независимо от стойностите на елементите им.

Варианти за изчисляване на средноаритметична стойност на данни, съхранявани в едномерен масив, посредством индекси и указатели:

✍ *пример P_7.4.1-1 ⇨ достъп до едномерен масив посредством индекси* ⇩

вариант А

```
# include<stdio.h>
# include<conio.h>
# define SIZE 1000          /* максимален размер на масива  */

int main(void)
{
  int n,Sum=0,i;     /*n - въвеждан от потребителя размер  */
  int ar[SIZE];              /*масива с елементите  */
  float sr;
  do
  {
    printf("\nВъведете нова стойност за n=");
    scanf ("%d",&n);        /* въвеждане на размера на масива  */
    if(n>1000||n<0)
   {
     printf("\n\n некоректен размер на масива !!! ");
     printf("\n\n за продължение - произволен клавиш ");
    getch();
   }
```

```
  } while(n>1000||n<=0);  /*повторение до правилно въвеждане */
  for(i=0;i<n;i++)    /*въвеждане на стойности на елементите*/
 {
    printf("\nВъведете стойност за ar[%d]=",i);
//-------- фрагмент за зареждане елементите на масива, чрез явна индексация ---------
    scanf ("%d",&ar[i]);
    Sum+=ar[i];  /* сумиране на елементите като достъпа е чрез явна индексация     */
//-------------------------------------------------------------------------------------------------------------------------
  }
  sr= (float) Sum/n;        /* намиране на средната стойност   */
  printf("Средна стойност sr=%8.2f\n",sr);
  return 0;
}
```

пример P_7.4.1-2 ⇨ фрагмент за достъп до едномерен масив посредством променлива - указател ⇩

вариант Б

```
int main(void)
{
  int n,Sum=0,i,*pa;        /*p - работен указател   */
  int ar[SIZE];        /*масива с елементите  */
  float sr;

// ---- фрагмент за избор на броя на елементите – същия като в пример P_7.2.1-1

  for(i=0,pa=ar;i<n;i++)    /*насочване на указателя към адреса на първия елемент от
                            масива началото */
  {
    printf("\nВъведете стойност за ar[%d]=",i);
    scanf ("%d",pa);        /*въвеждане на стойности на елементите*/
    Sum+=*pa;               /* сумиране на елементите     */
    pa++;                   /* указателят сочи следващ елемент        */
  }
// ……….……
```

пример P_7.4.1-3 ⇨ фрагмент за достъп до едномерен масив посредством константен указател, с използувайки адресна аритметика - указател ⇩

вариант В

```
int main(void)
{
  int n,Sum=0,i,*pa;        /*p - работен указател   */
  int ar[SIZE];        /*масива с елементите  */
  float sr;
```

// ---- фрагмент за избор на броя на елементите – същия като в пример P_*7.2.1-1*

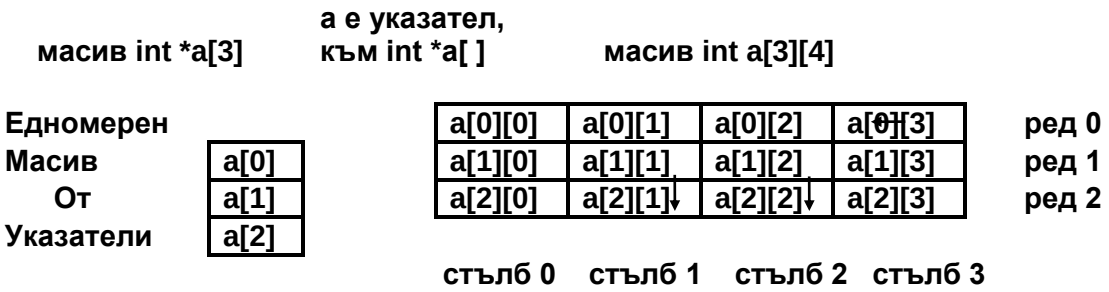
```
 for(i=0;i<n;i++)
 {
   printf("\nВъведете стойност за ar[%d]=",i);
          scanf ("%d",ar+i);      /* името на масива е константен указател към първия
                                     му  елемент*/
   Sum+=*(ar+i);           /* извличане на съдържанието на i-тия елемент */
 }
// ..........................
```

7.4.2. Многомерни масиви и указатели

Връзката между двумерните масиви и указателите в **C** може да се формулира по следния начин:

Идентификаторът на двумерен масив е **указател-константа** към едномерен масив от указатели-константи. Всеки елемент на едномерния масив от указатели е указател-константа към първия елемент на съответния ред от двумерния масив.

На ***фиг.7.2.2-1*** е показан **int a[3][4]** масив - три реда и четири стълба. За него компилаторът ще задели **12 x 2** (***int*** променлива заема 2 байта) - общо **24** байта (например от адрес **adr** до **adr+24**). За така дефинирания масив **a,** записът **a[i]**, където **i** се променя в интервала **[0,2]**, може да се разглежда като елемент от едномерен масив от указатели, всеки от които съдържа адреса на първия елемент от съответния ред на двумерния масив. Записът **int *a[3];** се интерпретира от компилатора като едномерен масив от указатели (операцията **[ ]** е с по-висш приоритет от *****). Тогава за двумерния масив са верни следните отношения:



фиг.7.2.2-1

a[0]==&a[0][0]; a[1]==&a[1][0]; a[2]==&a[2][0];

(à+1)[0] == a[1]; a[1] +1 == &a[1][1];

В пример ***P_6-6*** е показана проста програма, инициализираща масива **a[3][4]** с конкретни стойности **a[i][j]:=i*10+j** и извеждането им в таблична форма по редове, заедно с адресите от **ОП**, заделена за всеки елемент. Адресите са изведени в шестнадесетична и десетична форма.

пример P_7.2.2-1 ⇨ достъп до двумерен масив посредством указател ⇩

```
#include <conio.h>
#include <stdio.h>

int main(void) {

int a[3][4];
int i,j;
  clrscr();
  for (i=0;i<3;i++)
    for (j=0;j<4;j++)
      a[i][j]=i*10+j;
  for (i=0;i<3;i++) {
    printf("\n+----------+------------+------------+------------+------------+");
    printf("\n|   адрес  |");
    for (j=0;j<4;j++)
      printf("    %4p / %5u   |",&a[i][j], &a[i][j]);
    printf("\n+----------+------------+------------+------------+------------+");
    printf("\n| стойност |");
    for (j=0;j<4;j++)
      printf(" [%1d][%1d]=%3d |",i,j,a[i][j]);
   }
  printf("\n+----------+------------+------------+------------+------------+");
  return 0;
}
```

изход за пример P*_7.2.2-1* ⇩

Адреси	FFDE / 65502	FFE0 / 65504	FFE2 / 65506	FFE4 / 65508
Стойност	a[0][0]= 0	a[0][1]= 1	a[0][2]= 2	a[0][3]= 3
Адреси	FFE6 / 65510	FFE8 / 65512	FFEA / 65514	FFEC / 65516
Стойност	a[1][0]= 10	a[1][1]= 11	a[1][2]= 12	a[1][3]= 13
Адреси	FFEE / 65518	FFF0 / 65520	FFF2 / 65522	FFF4 / 65524
Стойност	a[2][0]= 20	a[2][1]= 21	a[2][2]= 22	a[2][3]= 23

7.5. Масиви, символни низове и указател към указател

Както вече беше отбелязано, в **C** съхраняването на символни последователности се осъществява само в съставни (агрегатни) типове променливи каквито са масивите, тъй като простият тип ***char*** е с размер 1 байт и с него може да се обработва само един символ.

Съществена особеност при обработка на **символни последователности** (**низове**) е очакването всеки низ да завършва със специалния символ **NULL**. Затова при дефиниция на низ е необходимо увеличаване размера на масива с 1 байт за съхраняването му (за об-

работка на символен низ от **80** символа трябва да се дефинира масив например **char text[81];**).

Типични примери за дефиниция и инициализация на програмни обекти - низове са:

```
.....
char niz[21];
    /* заделят се 21 байта - от 0 до 20 за елементите на символен низ,        */
    /* без да се инициализира с конкретни стойности                           */

char niz1[41]={”Инициализация на низ”};
    /* заделят се 41 байта - от 0 до 40 за елементите на символен низ,        */
    /* като се инициализира с текста в скобите niz1[0]='И',..niz1[19]=’з’,    */
    /* niz1[20]=NULL, а останалите елементи остават със случайни              */
    /* стойности, зависещи от моментното състояние на ОП                      */

char niz2[]={”ТЕКСТ”};
    /* компилаторът прави оценка (преброява символите) на литералната        */  /*
константа, добавя 1 символ за NULL и заделя място в ОП от 6 байта,        */
    /* което е идентично с int niz2[6];                                       */
.....
char  text[3][21];
    /* компилаторът заделя 63 байта за масива без инициализация               */
char  text1[3][10]={”низ 01”, ”низ 02”, ”низ 03” };
    /* компилаторът заделя 30 байта за масива и инициализира всеки ред        */
char  text2[][10]={”низ 01”, ”низ 02”, ”низ 03” };
    /* компилаторът заделя памет като за  text1 30 байта                      */
.....
char  *mess[3]={”1aaaa”, ”2bbbb”, ”3cccc”};
    /* масив от три указателя, които се инициализират с началото на           */
    /* всеки ред ->   *mess[0]=’1’ ; *mess[1]=’2’ ; *mess[2]=’3’ ;            */
.....
char  *t[ ]={”1aaaa”, ”2bbbb”, ”3cccc”};
    /* същото, но компилаторът сам определя броя на елементите на            */
    /* масива от указатели                                                    */
.....
```

пример P_*7.5-1* ⇨ обработка на текст ⇩

```
#include <conio.h>
#include <stdio.h>

int main(void) {

char text[256], ch;
int i,j;
 printf(”въведете текст не повече от 256 символа, последван от <Enter>”);
for (i=0;i<256;i++)
{
  ch=getche();
  if(ch==’\r’) break;
  text[i]=ch;
}
text[i]=NULL;
```

```
for(j=0;text[j];j++) ;
printf("текстът => %s => е s дължина %d символа",text,j);
return 0;
}
```

Съгласно синтаксиса на **C** е възможно дефинирането на променлива указател към указател (в практиката обикновено се нарича "двоен указател"). Признак за описание на двоен указател е наличието на записа ****** пред идентификатора на обекта. По-общо казано, броят на повтарящите се ***** определя и броя на вложените "указател към указател", но фактически при дефиниция на двоен указател компилаторът заделя памет за такава променлива, в която ще съхранява адреса на указател т.е. (2 или 4 байта в зависимост от модела на паметта).

Примерни дефиниции:

```
.......
int t=5;            /* int променлива с инициализация                    */
int *ptr_t=&t;      /* указател към int - зарежда се с адреса на  t      */
int **ptr_ptr_t=&ptr_t;
            /* указател към указател - зарежда се с адреса на ptr_t  */
int ***ptr_ptr_ptr_t=&ptr_ptr_t;
            /* указател към указател  "троен указател"               */
.......
t=*ptr_t-1;                 /* достъп до t, чрез ptr_t  t:=4;        */
t=**ptr_ptr_t-1   /* достъп до t, чрез ptr_ptr_t  t:=3;              */
t=***ptr_ptr_t-1  /* достъп до t, чрез ptr_ptr_ptr_t  t:=2;          */
.......
```

Достъпът до данните при работа с указатели към указатели е възможен и чрез индекси. За горните дефиниции двойките записи са еквивалентни:

```
*ptr_t          ------------>   ptr_t[0];
**ptr_ptr_t     ------------>   ptr_ptr_t[0][0];
***ptr_ptr_ptr_t  ---------->ptr_ptr_ptr_t[0][0][0];
```

<u>задания за самостоятелна работа върху масиви (протокол – 3)</u>

(обработка на данни в масиви – достъп до елементи от масива с указатели)

Да се създадат **C**-програми за:

- въвеждане на конкретни стойности в целочислен масив от клавиатурата (въвеждането в елементите да се реализира по трите варианта за достъп до елементите на масива – явна индексация, променлива – указател към началото на масива и константен указател – името на масива и променлива указваща отместването спрямо началото на масива).
- след въвеждането да се изведе на екрана съдържанието на масива.
- да се направят следните обработки:
  - да се намери максималната (минималната) стойност на елемента от масива, след което всички елементи чиято стойност е по-малка от половината на максималната (по-голяма два пъти от минималната) стойност да се занулят (0), а в останалите да се запише 1.

- да се разменят съдържанието на елементите на масива като 1-виа елемент се премести в последния, 2-рия в предпоследния и т.н. т.е масива да се пренареди в обратен ред.

- да се изведе на екрана съдържанието на масива след обработката му.

Да се създаде **C**-програма за въвеждане на конкретни стойности в целочислен масив от клавиатурата след което да се намери максималния/минималния елемент от всички стойности, като се изведе стойността му и мястото (индекса) му в масива.