CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Лекция 8 **Масиви и указатели. Двумерни масиви**

CODE ACADEMY

Промяна на стойностите на параметрите (скалари и масиви) при използване на функции

```
#include<stdio.h>

void film(int,char[ ]);

int main(){
int i = 5;
char x[ ] = "you and me";
printf("i = %d, x = %s\n", i, x);
film(i, x);
printf("i = %d, x = %s\n", i, x);
}

void film(int i, char z[ ])
{
printf("\nIn the film, i = %d, z = %s\n", i, z);
i = 0;
z[0] ='Y';
printf("In the film, i = %d, z = %s\n", i, z);
}
```

CODE ACADEMY

Промяна на адресите на параметрите (скалари и масиви) при използване на функции

```
#include<stdio.h>

void film(int,char[ ]);
int main()
{
  int i = 5;
  char x[ ] = "you and me";
  printf("i = %d, x = %s \n&i = %p, &x[0] = %p\n", i, x, &i, &x[0]);
  film(i, x);
  printf("i = %d, x = %s \n&i = %p, &x[0] = %p\n\n", i, x, &i, &x[0]);
}

void film(int i, char z[ ])
{
  printf("\nIn the film(), i = %d, z = %s \n&i = %p, &z[0] = %p\n\n", i, z, &i, &z[0]);
  i = 0;
  z[0] ='M';
  printf("In the film(), i = %d, z = %s \n&i = %p, &z[0] = %p\n\n", i, z, &i, &z[0]);
}
```

CODE ACADEMY

Промяна на параметрите при използване на функции

```
#include<stdio.h>
void multi(int arr[], int *i)
{
    int k;
    printf("array");
    for(k=0;k<3;k++ )
        printf("%d\t",arr[k]);

    int result=0;
    for( k=1;k<=3;k++){
        result=result + k*arr[k-1];
    }
    *i=result;
    printf("\n result=%d\n", *i);
}
int main(){
    int i=5;
    int *pi=&i;
    int arr[3]={1,2,3};
    printf("%d\n", pir, &i);
    printf("in main() i=%d\n", i);
```

CODE ACADEMY

Когато се предава масив на функция, промените, които функцията може да направи, ще засегнат масива, тъй като функцията има адреса на този масив.

```
#include<stdio.h>

void AddOne(int,int[ ]);
int main()
{
    int i, a[4] = {20, 40, 10, 60};
    printf("In main()\t a = %p\t&a[0] = %p\n", a, &a[0]);

    AddOne(a[2], a);
    for(i = 0; i <= 3; i = i + 1)
        printf("a[%d] = %d\n", i, a[i]);
}

void AddOne(int one, int x[ ])
{
    int i;
    printf("In AddOne()\t x = %p\t&x[0] = %p\n", x, &x[0]);
    for(i = 0; i <= 3; i = i + 1)
        x[i] = x[i] + one;
}
```

CODE ACADEMY

Използване на функция, която има параметър масив и функцията извежда масива от позиция, която се посочва на вход, до края.

```
#include<stdio.h>
#include<stdlib.h>
void res(char[ ]);
void main(void)
{
    res("abrakadabra");
    printf("\n");
}
void res(char x[ ])
{
    int j;
    printf("Enter an integer: ");
    scanf("%d", &j);
        printf("In res(), x = %s\n", x+j);
}
```

CODE ACADEMY

Извикване на функция с аргумент = елемент на масиви

```
#include<stdio.h>

void eha(int,int);

int main()
{
   int i = 50, a[3] = {90, 80, 60};
   eha(i, a[0]);
   printf("In main(), i = %d, a[0] = %d\n", i,a[0]);
}

void eha(int q, int w)
{
    printf("In eha(), q = %d, w = %d\n",q, w);
    q = 0;
    w = 0;
}
```

CODE ACADEMY

Stack пространство от паметта

Стека се разпределя на последователни блокове памет. Разпределението в стека се прави при извикване на функция. Размерът на паметта, която трябва да бъде разпределена, е известен на компилатора и всеки път, когато се извика функция, нейните **променливи получават памет, разпределена в стека**. Всеки път, когато извикването на функцията приключи, паметта, която е била отделена за променливите се преразпределя.

Всичко това се прави от някои предварително дефинирани процедури в компилатора. Програмистът не трябва да се притеснява за разпределението на паметта и освобождаването на променливи от стека. Такова разпределение на паметта, се нарича временно разпределение на паметта, тъй като веднага след като методът завърши изпълнението си, всички данни, принадлежащи на този метод, се изхвърлят от стека автоматично.

РАЗПРЕДЕЛЕНИЕТО НА ПАМЕТТА НА СТЕКА И ОСВОБОЖДАВАНЕТО НА СТЕКА СТАВА АВТОМАТИЧНО.
Ако паметта на стека е запълнена напълно, се получава съобщение за грешка.

CODE ACADEMY

Stack пространство от паметта

```c
#include<stdio.h>
#include<stdlib.h>
int* readArr(int n){
     int* result=malloc(n*sizeof(int));
     for(int i=0;i<n;i++)
     {
          scanf("%d",result+i);
     }
     return result;
}
void swap(int *p, int *q){
   int tmp=*p;
   *p=*q;
   *q=tmp;
}
void bubbleSort(int x[], int n)
{
     int br=0;
     while(br<n-1)
     {
        for(int i=0; i<n-1; i++){
             if(x[i+1]<x[i])
                  swap(&x[i],&x[i+1]);
        }
      br++;
     }
}
```

CODE ACADEMY

Stack пространство от паметта

Всяка работеща програма има свое собствено оформление на паметта, което се състои от много сегменти:

stack: съхранява локални променливи

heap: динамична памет за разпределяне на програмиста

data: съхранява глобални променливи, разделени на инициализирани и неинициализирани

text: съхранява кода, който се изпълнява

За да определим точно всяко място в паметта на програмата, на всеки байт памет се присвоява **„адрес“**. **Адресите вървят от 0 чак до най-големия възможен адрес,** в зависимост от машината.

Сегментите с текст, данни и heap имат ниски адресни номера.

Паметта на стека има по-високи адреси.

CODE ACADEMY

Stack пространство от паметта

```
void printArr(int w[], int n){
   for(int ind=0;ind<n; ind++)
        printf("%d\t", *(w+ind));
   printf("\n");
}



int main()
{
       int* myArr=readArr(6);
       bubbleSort(myArr,6);
       printArr(myArr,6);

       int number;
       scanf("%d",&number);

       int *copyArr=malloc(7*sizeof(int));
       int k=0;

       int i;
```

CODE ACADEMY

Heap пространство от паметта

Паметта в **Heap се разпределя по време на изпълнение на инструкции,** написани от програмисти.
Всеки път, когато правим обект, той винаги създава в Heap-пространство.
Разпределението на паметта в Heap-пространството не е толкова безопасно, колкото разпределението на паметта на стека.

В Heap не е предоставена функция за автоматично освобождаване.
За да използваме паметта ефективно трябва да използваме колектор за боклук, за да премахнем старите неизползвани обекти.

Времето за достъп на Heap паметта е доста бавно в сравнение със стековата памет.

Heap-паметта също **не е нишкова безопасна** като Stack-memory, тъй като данните, съхранявани в Heap-memory, са видими за всички нишки.

Размерът на паметта на Heap е доста по-голям в сравнение със стековата памет.

Heap-паметта е достъпна или съществува, докато работи цялото приложение.

CODE ACADEMY

Heap пространство от паметта

Паметта в **Heap се разпределя по време на изпълнение на инструкции,** написани от програмисти.
Всеки път, когато правим обект, той винаги създава в Heap-пространство.
Разпределението на паметта в Heap-пространството не е толкова безопасно, колкото разпределението на паметта на стека.

В Heap не е предоставена функция за автоматично освобождаване.
За да използваме паметта ефективно трябва да използваме колектор за боклук, за да премахнем старите неизползвани обекти.

Времето за достъп на Heap паметта е доста бавно в сравнение със стековата памет.

Heap-паметта също **не е нишкова безопасна** като Stack-memory, тъй като данните, съхранявани в Heap-memory, са видими за всички нишки.

Размерът на паметта на Heap е доста по-голям в сравнение със стековата памет.

Heap-паметта е достъпна или съществува, докато работи цялото приложение.

CODE ACADEMY

Сравнение между Stack и Heap

Разпределението на паметта на стека се счита за по-безопасно и по-бързо в сравнение с разпределението на паметта на Heap.

Стековата памет има по-малко място за съхранение в сравнение с паметта Heap.

Достъпът до рамката на стека е по-лесен от рамката на Heap, тъй като стекът има малък регион на паметта и е удобен за кеширане.

Стекът не е гъвкав, размерът на разпределената памет не може да бъде променен, докато Heap е гъвкава и разпределената памет може да бъде променена.

За разлика от паметта на стека, паметта на heap се разпределя изрично от програмистите и няма да бъде освободена, докато не бъде изрично освободена.

Времето за достъп до Heap е повече от времето на достъп до Stack.

Проблемът с недостиг на памет е по-вероятно да се случи в стека, докато основният проблем в Heap паметта е фрагментацията.

CODE ACADEMY

Функция, която въвежда елементите в масив от цели числа. Програмата чете цяло число n и извиква функцията да прочете масив от n числа. Накрая програмата извежда един от елементите на масива.

```
#include<stdio.h>
int* readArr(int n){
    int* result=malloc(n*sizeof(int));
    for(int i=0;i<n;i++)
    {
        scanf("%d",result+i);
    }
    return result;
}

int main()
{
    int* myArr;
    myArr=readArr(6);
    printf("%d", myArr[3]) ;
}
```

Тук, вградената стандартна функция malloc(k) заделя памет от k байта от Heap паметта и тези k байта са презназначени за масива с резултата от четенето.

https://en.cppreference.com/w/c/memory/malloc

Внимание! Заделената с malloc() памет трябва да се освободи

```
int *p1 = malloc(4*sizeof(int));
```

Разглеждане на rand() и RAND_MAX от библиотеката <stdlib.h>

CODE ACADEMY

Напишете програма, която създава и инициализира динамичен едномерен масив от 100 двуцифрени цели числа. За целта използвайте функцията rand(), генерираща случайно число в диапазона от 0 до RAND_MAX, включително.

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

int main()
{
   printf("%d\n", RAND_MAX);
   int *arr= malloc(100*sizeof(int));

   for(int i=0; i <100; i++)
      *(arr+i)=rand()%100;

   for(int i=0; i <100; i++)
      printf("%d\t", *(arr+i));
}
```

CODE ACADEMY

Задачи

1. Напишете програма, която да обедини два масива от цели числа, като ги сортира в низходящ ред и ги принтира.

```
    for( i=0; i<6; i++)
    {
        if(number<myArr[i])
        {
            *(copyArr+k)=number;
             k++;
            break;
        }
        else
        {
           if(number>myArr[i])
           {
                *(copyArr+k)=*(myArr+i);
                k++;
           }
        }
    }

while(i<6)
   {
    *(copyArr+k)=*(myArr+i);
    i++;
    k++;
}
printArr(copyArr,7);
}
```

```
#include<stdio.h>
#include<stdlib.h>
int* readArr(int n){
    int* result=malloc(n*sizeof(int));
    for(int i=0;i<n;i++)
    {
        scanf("%d",result+i);
    }
    return result;
}
void swap(int *p, int *q){
  int tmp=*p;
  *p=*q;
  *q=tmp;
}
void bubbleSort(int x[], int n)
{
    int br=0;
    while(br<n-1)
    {
       for(int i=0; i<n-1; i++){
            if(x[i+1]<x[i])
                swap(&x[i],&x[i+1]);
       }
     br++;
    }
}
```

```
void printArr(int w[], int n){
   for(int ind=0;ind<n; ind++)
       printf("%d\t", *(w+ind));
   printf("\n");
}


int main()
{
     int* myArr=readArr(6);
     bubbleSort(myArr,6);
     printArr(myArr,6);

     int number;
     scanf("%d",&number);

     int *copyArr=malloc(7*sizeof(int));
     int k=0;

     int i;
```

Задача 1 - коментар по решението

```
/// a --> 8,5,1
// b-->34, 25, 18, 10, 2
// c[0]=34   j=1 k=1
// c[1]=25   j=2  k=2
// c[2]=18   j=3   k=3
// c[3] =10   j=4   k=4
// a[0]>b[4]
// 8>2        c[4]=a[0]  i=1  k=5
//
// 5>2        c[5]=a[1]   i=2  k=6
//
// 1<2        c[6]=b[5]   j=5  k=7
//
// while(a[2]>b[5]) --> не влиза
//
//    -------
//    b --->c
//    остава  a[2]];
```

```
2 4 87 2 5 7
4 12
---------------
87 7 5 4 2 2
12 4
-----------
7 5 4 2 2
12 4
87
-----
7 5 4 2 2
4
87  12
---
5 4 2 2
4
87 12 7
----
4 2 2
4
87 12 7 5
--
2 2
нищо
```

Задачи

CODE ACADEMY

2. Напишете програма, която въвежда масив от цели числа от клавиатурата. Сортирайте го във възходящ ред. След това от клавиатурата въведете цяло число и го поставете на правилната му позиция в масива, така че да не се нарушава реда. Принтирайте масива.

Задача 2 - решение

CODE ACADEMY

```c
#include<stdio.h>
#include<stdlib.h>
int* readArr(int n){
    int* result=malloc(n*sizeof(int));
    for(int i=0;i<n;i++)
    {
        scanf("%d",result+i);
    }
    return result;
}
void swap(int *p, int *q){
  int tmp=*p;
  *p=*q;
  *q=tmp;
}
void bubbleSort(int x[], int n)
{
    int br=0;
    while(br<n-1)
    {
       for(int i=0; i<n-1; i++){
            if(x[i+1]<x[i])
                swap(&x[i],&x[i+1]);
       }
     br++;
    }
}
```

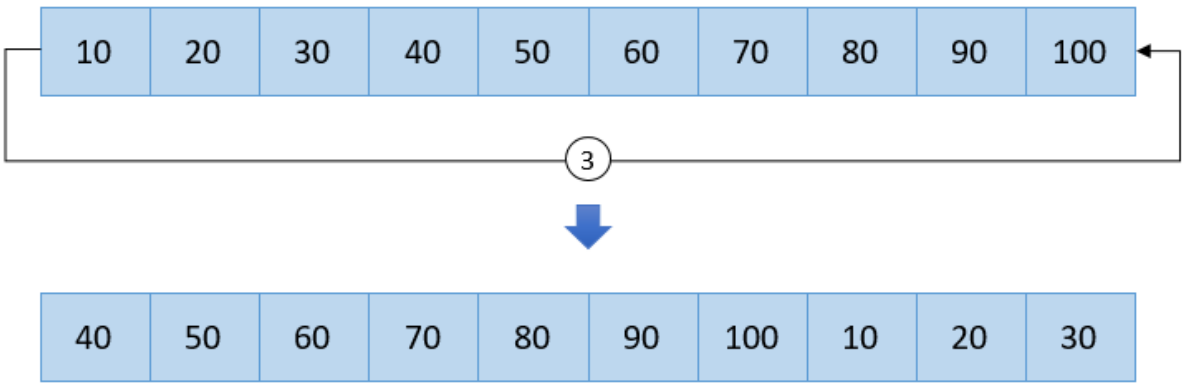
```c
void printArr(int w[], int n){
  for(int ind=0;ind<n; ind++)
      printf("%d\t", *(w+ind));
  printf("\n");
}


int main()
{
    int* myArr=readArr(6);
    bubbleSort(myArr,6);
    printArr(myArr,6);

    int number;
    scanf("%d",&number);

    int *copyArr=malloc(7*sizeof(int));
    int k=0;

    int i;
```

```c
    for( i=0; i<6; i++)
    {
        if(number<myArr[i])
        {
            *(copyArr+k)=number;
             k++;
            break;
        }
        else
        {
           if(number>myArr[i])
           {
                *(copyArr+k)=*(myArr+i);
                k++;
           }
        }
    }

while(i<6)
   {
    *(copyArr+k)=*(myArr+i);
    i++;
    k++;
}
printArr(copyArr,7);
}
```

CODE ACADEMY

Задачи:

3. Напишете програма, която да завърти даден масив от цели числа с **N** позиции наляво. Масива и числото **N** трябва да бъдат въведени от клавиатурата. Принтирайте оригиналния и резултатния масиви. Пример:



CODE ACADEMY

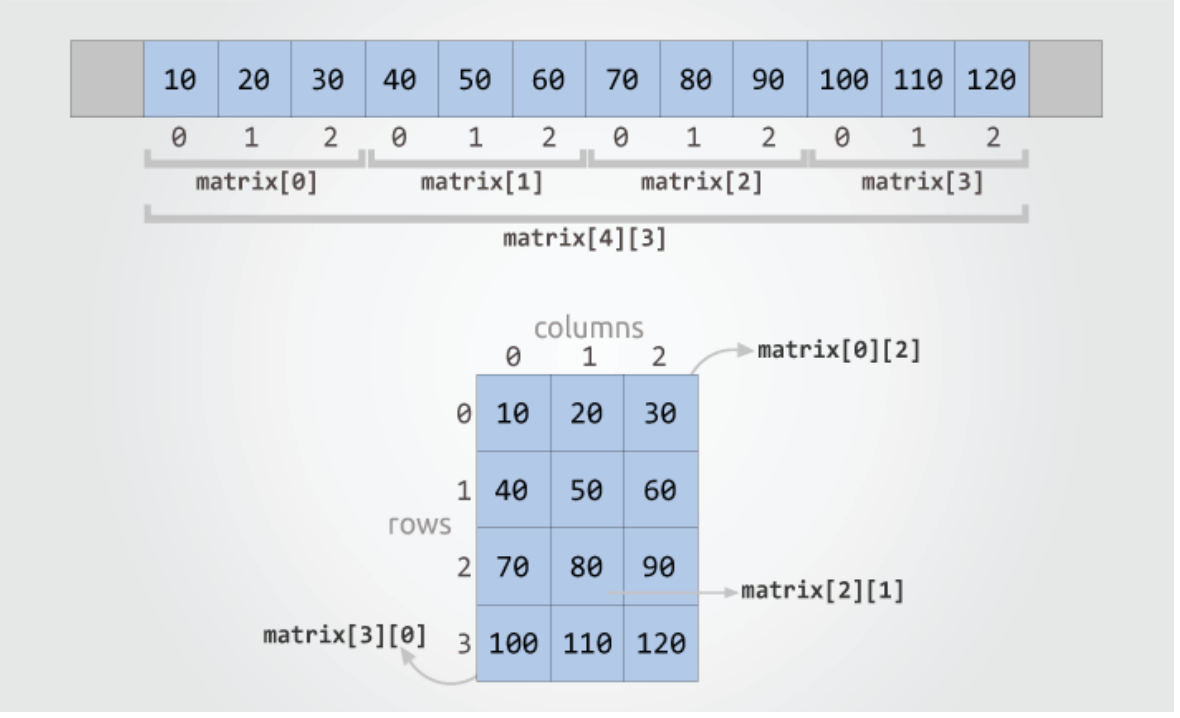
Задача 3 - решение

```
#include<stdio.h>

 int main()
 {
     int arr[]={1,2,3,4,5,6,7,8,9,10};
     int b[10];
     int len=10;
     int N;
     scanf("%d",&N);

     for(int i=0;i<len;i++)
     {
         int newIndex= (i-N+len)%len;
         b[newIndex]=arr[i];
     }

     for(int i=0;i<len;i++)
        printf("%d\t",b[i]);
 }
```

CODE ACADEMY

Двумерни масиви

Двумерните масиви могат да бъдат разглеждани като колекция от едномерни такива. Логически можем да си ги представим като матрица. Всяка математическа задача, касаеща матрици, може лесно да бъде представена като двумерен масив.



CODE ACADEMY

Двумерни масиви

Декларация:

```
type array_name[row-size][col-size];
```

- `type` - валиден **C** тип.
- `array_name` - идентификатор.
- `row-size` - константа, специфицираща броя редове на матрицата.
- `col-size` - константа, специфицираща броя колони в матрицата.

Пример:

```
int matrix[3][4];
```

CODE ACADEMY

Двумерни масиви

Инициализация:

```
int matrix[4][3] = {
                        {10, 20, 30},    // Initializes matrix[0]
                        {40, 50, 60},    // Initializes matrix[1]
                        {70, 80, 90},    // Initializes matrix[2]
                        {100, 110, 120}  // Initializes matrix[3]
                    };
```

Възможен е и друг запис:

```
int matrix[4][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110,
120};
```

Двумерни масиви

При инициализацията е възможно да пропуснем броя редове в матрицата:

```
int matrix[][3] = {
                        {10, 20, 30},    // Initializes
matrix[0]
                        {40, 50, 60},    // Initializes
matrix[1]
                        {70, 80, 90},    // Initializes
matrix[2]
                        {100, 110, 120}  // Initializes
matrix[3]
                    };
```

CODE ACADEMY

Двумерни масиви

Достъпът до елементите на всеки двумерен масив, както и при едномерните, става чрез индекс. В този случай обаче, указваме както реда, така и колоната на елемента. Пример:

```
matrix[0][0] = 10;  // Assign 10 to first element of first row
matrix[0][1] = 20;  // Assign 20 to second element of first
row
matrix[1][2] = 60;  // Assign 60 to third element of second
row
matrix[3][0] = 100; // Assign 100 to first element of fourth
row
```

CODE ACADEMY

Двумерни масиви

Тъй като индексите са цели числа, можем да напишем цикъл за достъп до отделните елементи на матрицата. Но в този случай ще се наложи да използваме и вложен цикъл, за да итерираме както по редовете, така и по колоните на матрицата.
Пример:

```
// Runs 4 times iterating through each row
for(i = 0; i < 4; i++)
{
    // Runs 3 times for each column.
    for(j = 0; j < 3; j++)
    {
        scanf("%d", &matrix[i][j]);
    }
}
```

CODE ACADEMY

Задачи с двумерни масиви

Задача 1. Декларирайте двумерен масив с по 5 елемента (5 x 5). След като сте готови, направете въвеждане на данните в масива, като четете от потребителя със scanf. Отпечатайте масива в конзолата, спазвайки броя редове и колони.

Задача 2. Дом Напишете програма, която събира две матрици (4x4) и записва резултата в трета такава със същия размер. Събирането на матриците става като намерим сумата на съответстващите елементи в тях. Данните за изходните матрици трябва да бъдат въведени от потребителя. Разпечатайте трите матрици в конзолата.

CODE ACADEMY

Задачи с двумерни масиви

Задача 3. Напишете програма, която проверява дали две дадени матрици са еднакви, сравнявайки съответните елементи в тях. Разпечатайте в конзолата двете изходни матрици и резултата от сравнението.

Задача 4* Дом. Напишете програма, която проверява дали дадена матрица е **identity matrix** - матрица с размер (N x N), където само елементите в главния диагонал са единици, а всички останали елементи са нули.

Задача 5*. Напишете програма, която търси дадено число в сортирана по редове матрица. Генерирайте елементите на матрицата с rand(). Принтирайте матрицата и въведете от клавиатурата търсеното число. Покажете позицията му.

Задачи с двумерни масиви-решение на зад 5

```
void swap(int *p, int *q){


void bubbleSort(int x[], int n)
{

int find(int x[], int n, int number)
{
    for(int elem=0;elem<n;elem++)
        if(x[elem]==number)
          return elem;
    return -1;
}

int main(){
    int a[3][4];

    for(int row=0;row<3;row++)
        for(int col=0;col<4;col++)
            a[row][col]=rand()%50;

    for(int row=0;row<3;row++) {
        int *pRow=a[row];
        bubbleSort(pRow,4);
    }
```

```
    for(int row=0;row<3;row++)
     {
       for(int col=0;col<4;col++)
         {
           printf("%d\t", a[row][col]);
         }
         printf("\n");
     }

      int num ;
      scanf("%d",&num);

      int p,q;
      for(int row=0;row<3;row++)
      {    p=row;
           q=find(a[row],4, num);
           if(q>=0)
              printf("(%d,%d)\t", p,q);
      }

}
```

Задачи с двумерни масиви

Задача 6*. **Дом** Напишете програма, която печата броя на уникалните редове в бинарна матрица, съставена само от единици и нули.

Задача 7*.(да се помисли-решението следващо занятие) Напишете програма, която намира под-матрица с максимален размер в дадена бинарна матрица, съставена само от нули и единици. Пример:

The given array in matrix form is :
0 1 0 1 1
1 1 1 1 0
1 1 1 1 0
1 1 1 1 0
1 1 1 1 1
0 1 0 1 0
The maximum size sub-matrix is:
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1