CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

СРЕЩА 7 Указатели и Масиви

CODE ACADEMY

Указатели

Указателите в програмите на C се използват за достъп до паметта и манипулиране на адреса.
Първо да се запознаем с **адреса в C**.

Нека **var е променлива** в програмата.

Тогава **&var ще даде нейния адрес в паметта**.

Тук **& се нарича референтен оператор**.

Това означение се използва се във функцията scanf() за съхраняване на въведената от потребителя стойност в адреса на var:

```
scanf("%d", &var);
```

CODE ACADEMY

Указатели - пример

```
#include <stdio.h>
int main()
{
  int var = 5;

  printf("Value: %d\n", var);

  printf("Address: %u", &var);  // (&) е преди променливата.
}
```

Забележка При различните изпълнения се получава различна стойност на адреса

CODE ACADEMY

Променливи от тип указател – pointer

В С има специална променлива, която съхранява само адреса на друга променлива. Нарича се **указател или pointer**.

Деклариране на указател

data_type* pointer_variable_name;

За да декларираме **p** като променлива указател към тип int пишем

int* p;

CODE ACADEMY

Оператор референция и дереференция

Reference operator (&)

& се нарича reference оператор. Той ви дава адреса на променлива.

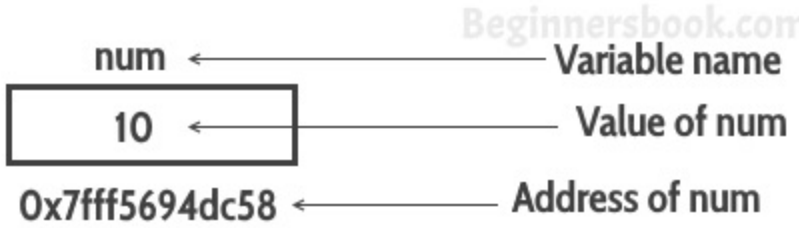
Dereference operator (*)

Операторът, който ви дава стойността по адреса се нарича dereference оператор (*).

Забележка:

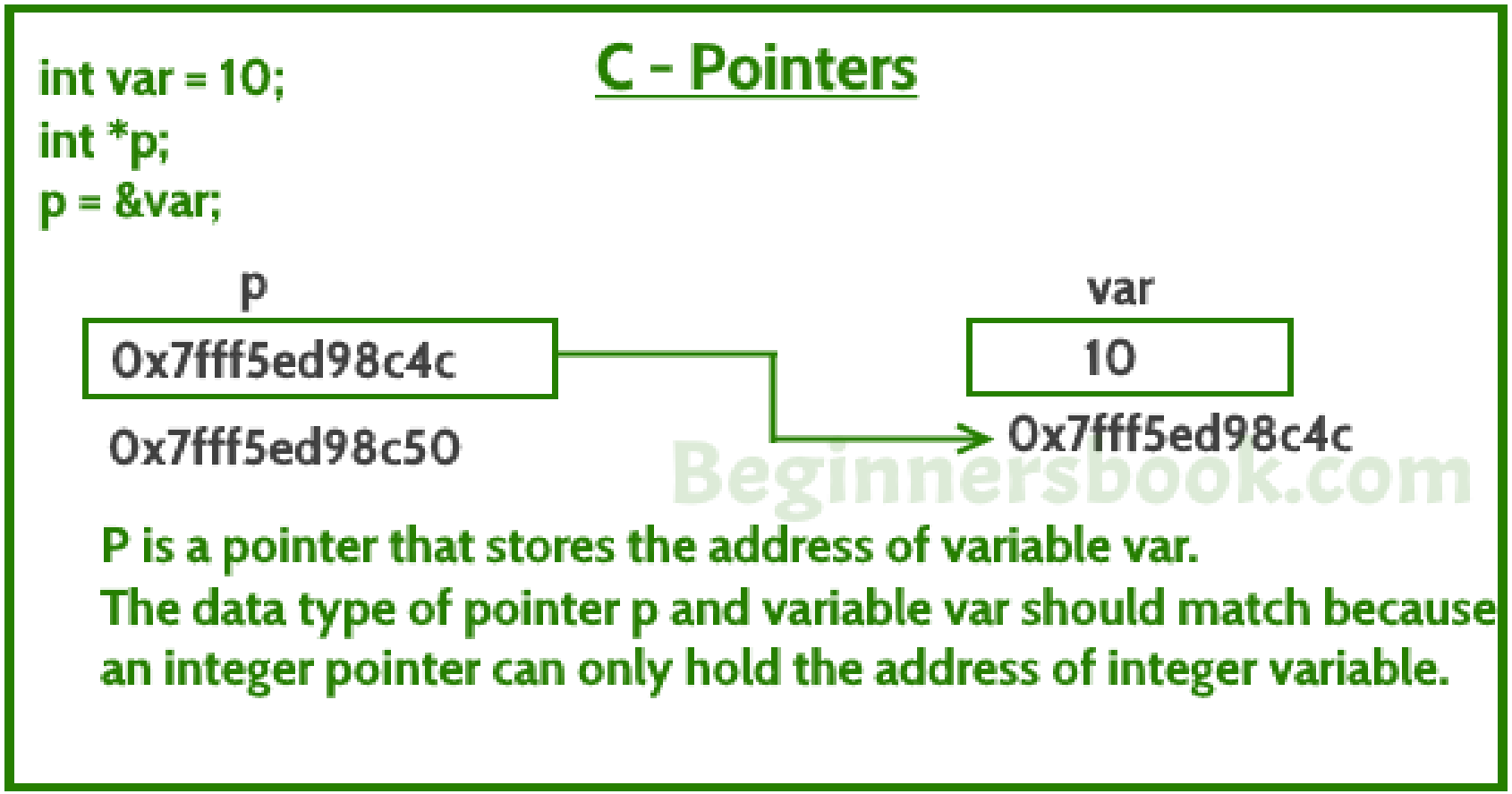
Знакът * при деклариране на указател не е оператор за дереферииране.

Това е просто подобна нотация, която създава указател.



CODE ACADEMY

Оператор референция и дереференция



CODE ACADEMY

Пример

```
#include <stdio.h>

int main()
{
  int *ptr;
  int x;

  ptr = &x;
  *ptr = 0;

  printf(" x = %d\n", x);
  printf(" *ptr = %d\n", *ptr);

  *ptr += 5;
  printf(" x = %d\n", x);
  printf(" *ptr = %d\n", *ptr);

  (*ptr)++;
  printf(" x = %d\n", x);
  printf(" *ptr = %d\n", *ptr);
}
```

CODE ACADEMY

Указатели - видове

Има основно четири вида особени указатели:

Null Pointer

Void Pointer

Wild Pointer - див указател

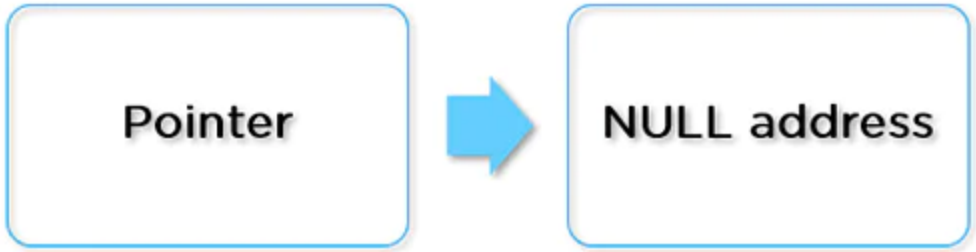
Dangling Pointer - висящ указатели

CODE ACADEMY

Null Pointer

Указател, на който се присвоява стойност NULL
по време на неговото деклариране.

```
int *var = NULL;

#include<stdio.h>

int main()
{
        int *var = NULL;
        printf("var=%d",*var);
}
```



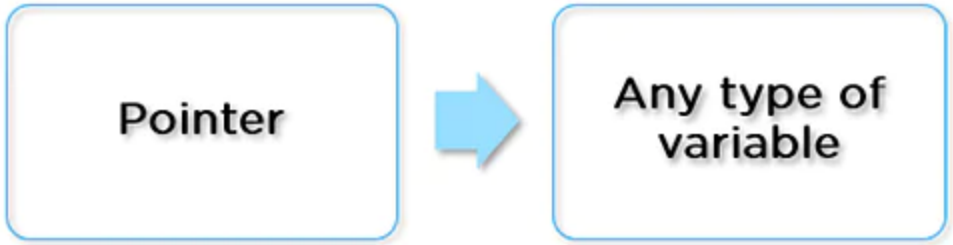
CODE ACADEMY

Void Pointer

void pointer-указател,деклариран с ключова дума void

```
void  *var;
```

```
#include<stdio.h>
int main()
{
int a=2;
void *ptr;
    ptr= &a;
printf("After Typecasting, a = %d", *(int *)ptr);

return 0;
}
```

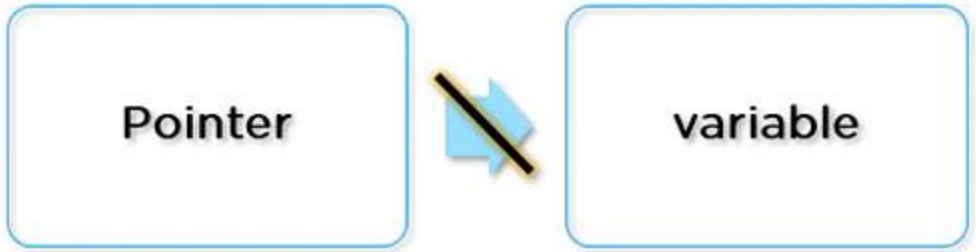


CODE ACADEMY

Wild Pointer

Wild Pointer: указател, който само се декларира, без да му се присвоява адрес на която и да е променлива.
Те са много трудни и могат да причинят грешки при сегментиране.

```
#include<stdio.h>
int main()
{
int *ptr;
        printf("ptr=%d",*ptr);

        return 0;
}
```

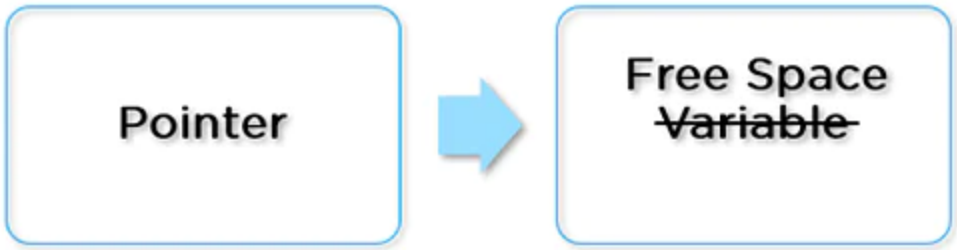


CODE ACADEMY

Dangling Pointer - висящ указател

Dangling Pointer: Да предположим, че има указател p, сочещ променлива в адрес от паметта 1004.
Ако тази памет се освободи с помощта на функция free() , тогава това p се нарича висящ указател.

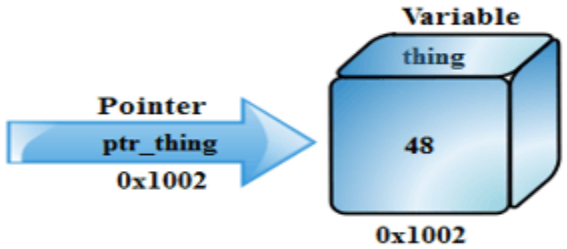
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *ptr=(int *)malloc(sizeof(int));
        int a=5;
        ptr=&a;
        free(ptr);
        //now this ptr is known as dangling pointer.
        printf("After deallocating its memory *ptr=%d",*ptr);
        return 0;
}
```



CODE ACADEMY

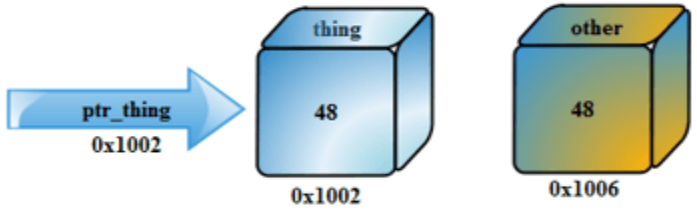
С указателите могат да се извършат следните действия

A) ptr_thing = &thing;



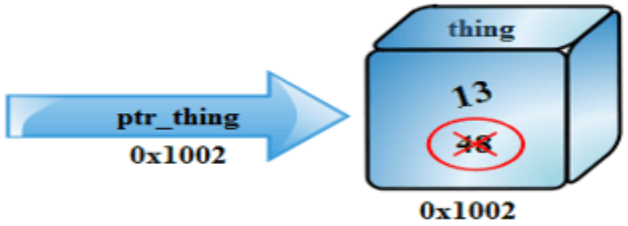
Assigns thing's address to ptr_thing

B) other = *ptr_thing;



Assigns to other the value pointed by ptr_thing

C) *ptr_thing=13;



Assigns new value to the pointed variable

Приоритет на операциите с указатели

Операторите * и & имат един и същ приоритет като унарните оператори отрицание !, incrementation++, decrement--

В един и същ израз унарните оператори *, &,!, ++, – се изчисляват от дясно наляво.

Ако указателят P сочи променливата X, тогава * P може да се използва вместо X .
int X =10
int *P = &X;
За този код следните изрази имат стойност true

Expression	Equivalent Expression
Y=*P+1	Y=X+1
*P=*P+10	X=X+10
*P+=2	X+=2
++*P	++X
(*P)++	X++

В последния случай са [illegible] яват отдясно наляво, без скобите указателят P ще се увеличи, а не обектът, върху който сочи P.

CODE ACADEMY

Аритметичните и основните операции при работа с указатели

Operation	Explanation

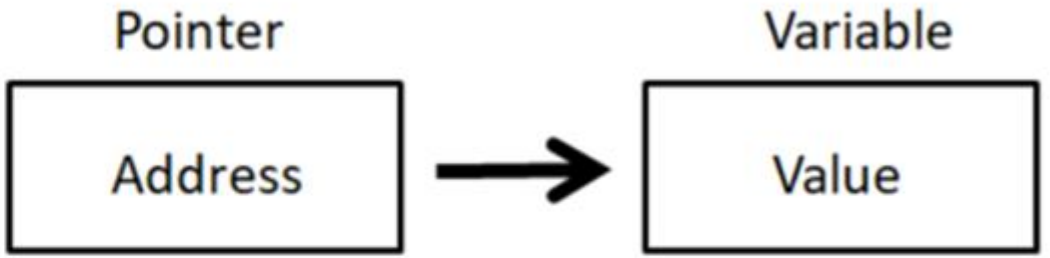
Присвояване	int *P1,*P2
	Int x=6;
	P2=&x;
	P1=P2; // P1=&x;
	P1 и P2 сочат една и съща целочислена променлива

Incrementation и decrementation	Int *P1; Int x=8; P1=&x; P1++; P1-- ;
Добавяне на отместване (постоянно)	Това позволява на указателя да се премести N елемента в таблица. Указателят ще бъде увеличен или намален с N пъти броя на байтовете от типа на променливата. P1+5;

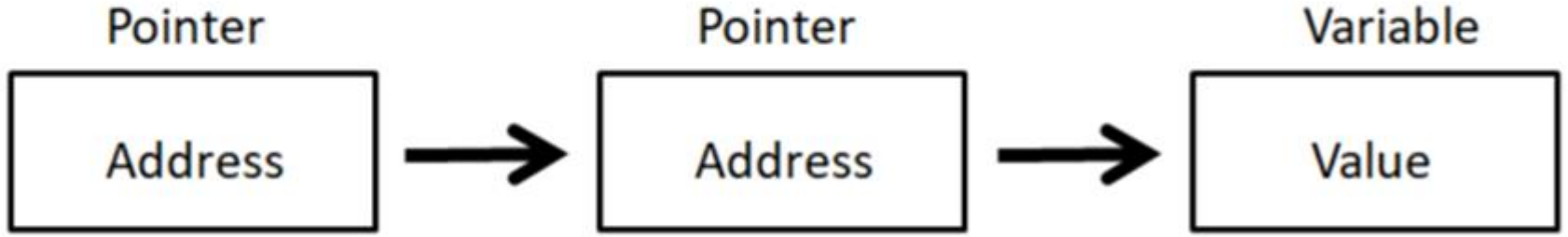
CODE ACADEMY

Двойни указатели

```
int *p;
```




```
int **p;
```



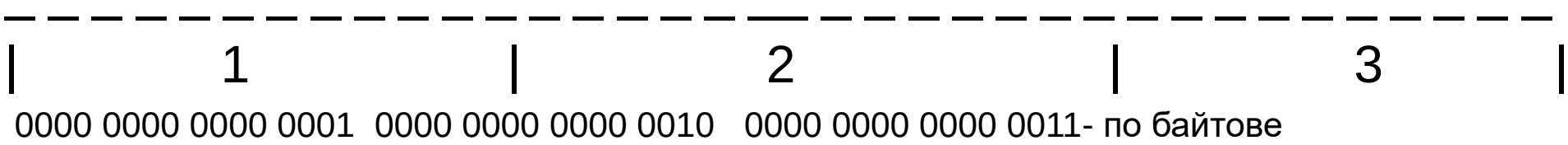
CODE ACADEMY

Масиви

- Масивите съхраняват колекция от елементи от един тип
  (например: пакет бисквити)
- Елементите на масива са разположени последователно в паметта
- Достъпът до елемент се прави посредством неговата позиция (индекс)

Разполагане в паметта

short int arrNum [3]; /* едномерен масив */
arrNum[0] = 1;
arrNum[1] = 2;
arrNum[2] = 3;
sizeof(arrNum) -> 6



CODE ACADEMY

Дефиниция и декларация

- дефиниция

```
Ctype arrVector[ N ]; /* array , N elements */
Ctype arrMatrix[ N ] [ M ]; /* array, N * M elements */
```

- декларация

```
extern Ctype arrVector[ ];
extern Ctype arrMatrix[ ] [ M ];
```

CODE ACADEMY

Масиви - инициализация

- инициализация при дефиниране

```
char szName [] = "Ivan Ivanov";
char szJob [16] = { 0 };
char szSex [16] = {'W', 'o', 'm', 'a', 'n', '\0'};
int szWorkDays [31] = { 0 }; /* all elements are 0 */
int i_arr[] = {1, 2, 3};
double d_arr[] = {1.2, 2.3, 3.4};
```

- достъп до елементите с индекси
- индексите са нула базирани
- възможно е масивите да не са инициализирани при дефиниране

CODE ACADEMY

Инициализация

Има още един начин за инициализиране на масив още при неговото дефиниране:

```
int arrNum [ 3 ] = { 1, 2, 3 };

for(int i = 0; i < 3; i++){
printf("%d",arrNum[i]);
}
```

Също така можем да пропуснем размерността му.

```
 int arrNum [ ] = { 1, 2, 3 };
/* sizeof(arrNum) / sizeof(arrNum[0]) -> 3 */
```

CODE ACADEMY

Инициализация

В горния случай, **всеки int се разполага на 4 байта**, а **масивът** се състои от 3 цели числа и следователно **заема 12 байта**.

Трябва да се отбележи, че **индексите са нула базирани**
- първият елемент е с индекс 0
- вторият елемент е с индекс 1 и т.н.

Често срещана грешка е да се опитаме да достъпим последния елемент от масива, като забравим, че индексите започват от нула

Последният елемент има индекс размерността на масива минус единица.

CODE ACADEMY

Масиви и указатели

В езика **C**, **указателите и масивите са много тясно взаимосвързани**.

Масивът е структура от данни, която съдържа в себе си последователна колекция от определен тип. За да достигнем определен елемент от масива използваме индекс. Индексът започва от **0** и завършва с **N-1**, където N е броят на елементите в масива.

В **C**, когато се обърнем директно към **името на масива**, то се държи като **указател към нулевия елемент на масива**.

Това означава, че следните два записа са еквивалентни:

```
int * ptr = &arr[0];
```

```
int * ptr = arr;
```

CODE ACADEMY

Пример

```
#include<stdio.h>
        int main()
        {
                short int a[3]={1,2,3};

                printf("%d\t",  *a);
                printf("%d\t",  a[0]);
                printf("0x%x\t",  a);  // адреса на началото на масива= адреса на
a[0]
                printf("0x%x\n",  &a[0]);

                printf("%d\t",  a[1]);
                printf("%d\t",  *(a+1));
                printf("0x%x\t",  (a+1));
                printf("0x%x\n",  &a[1]);

                printf("%d\t",  a[2]);
                printf("%d\t",  *(a+2));
                printf("0x%x\t",  (a+2));
                printf("0x%x\n",  &a[2]);

                printf("%d\t",  a[3]);
                printf("%d\t",  *(a+3));
                printf("0x%x\t",  (a+3));
                printf("0x%x\n",  &a[3]);
}
```

CODE ACADEMY

Масиви и указатели

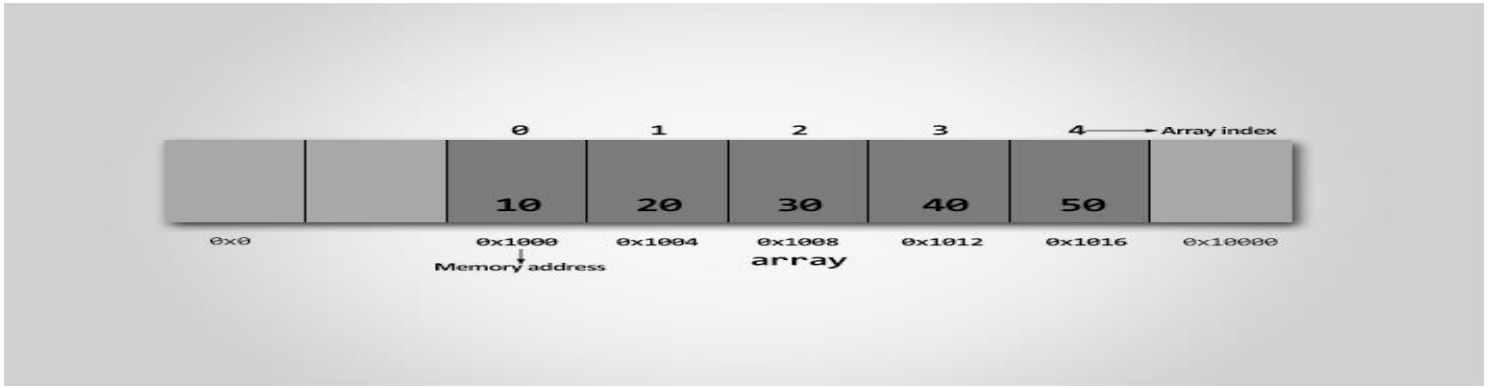
Това специално поведение на масивите и указателите ни позволява да взаимозаменяме масивите с указатели в програмите си. В контекста на масиви и указатели, всички следващи записи са валидни:

```
int arr[5];
int * ptr = arr;

arr[0] = 10;     // Stores 10 at 0th element of array
ptr[1] = 20;     // Stores 20 at 1st element of array

ptr    = arr;    // ptr and arr both points to 0th element of array
*ptr   = 100;    // Stores 100 at 0th element of array (Since ptr points at arr[0])
*arr   = 100;    // Stores 100 at 0th element of array
```

Масиви и указатели

Елементите на един масив се запазват в паметта последователно.

```
int arr[] = {10, 20, 30, 40, 50};
```




```
#include<stdio.h>
        int main()
        {
                short int
a[3]={1,2,3};
    int l=sizeof(a)/sizeof(a[0]);
    printf("%d", l);
```

CODE ACADEMY

Масиви и указатели

В горния пример, първия елемент на масива `arr[0]` се намира на адрес `0x1000`. В този пример се има предвид, че размерът на int е 4 байта. Следователно, следващият елемент в масива `arr[1]` ще се намира на адрес `0x1004` и така до петия елемент, който започва от `0x1016`. И така, след като елементите на масива са разположени последователно в паметта, ние можем да използваме pointer arithmetic, за да итерираме елементите на масива. Можете да използвате следните операции:

```
int arr[5];

int * ptr = arr;          // Integer pointer pointing at arr[0]

ptr[0] = 10;              // Assigns 10 to arr[0]

ptr++;                    // ptr now points at arr[1]

ptr--;                    // ptr now points back at arr[0]

*(ptr + 4) = 100;         // Assigns 100 to arr[4].

                          // Note, ptr currently pointing at arr[0] not arr[1].

                          // Hence (ptr + 4) will point at arr[4]

*(arr + 0) = 10;          // Assigns 10 to arr[0]
```

CODE ACADEMY

Масиви и указатели

```c
#include <stdio.h>
#define SIZE 10     // Maximum array size

int main()
{
    int arr[SIZE];  // Declare an array of size 10
    int *ptr = arr; // Pointer to first element of integer array
    int i = 0;
    /* Input number from user in array */
    printf("Enter %d array elements: ", SIZE);
    while(ptr < &arr[SIZE])
    {
        // Input in array using pointer
        scanf("%d", ptr);

        // Move pointer to next array element
        ptr++;
    }
    // Make sure pointer point back to 0th element
    ptr = arr;

    // Print all elements using pointer
    printf("Elements in array are: ");
    for(i=0; i < SIZE; i++)
    {
        printf("%d, ", *(ptr + i));
    }

    return 0;
}
```

CODE ACADEMY

Масиви и указатели

Както вече казахме, можете да използвате **името на масива** директно, като компилаторът го интерпретира като указател към нулевия елемент от масива. Този указател не може да бъде модифициран, т.е. можем да разглеждаме указателя като **const** указател. Константният указател е специален указател, чиято стойност не може да бъде променяна. С други думи, ако такъв указател сочи към определено място в паметта, той не може да бъде “пренасочен” към друг адрес по-късно в програмата.

```
int main()
{
    int arr[] = {10, 20, 30, 40, 50};  // Integer array
    int *ptr = arr;  // Pointer to 0th element of array
    /*
     * If arr behaves as a constant pointer then compiler
     * must complain about arr++. Since arr++ is equivalent to
     * arr = arr + 1 which is not permitted.
     */
     arr++; // Error
     // No error
     ptr++;

    return 0;
}
```

CODE ACADEMY

Примери:

```
int arr[] = {10, 20, 30, 40, 50};

arr[0];  // Equivalent to *(arr + 0)
arr[4];  // Equivalent to *(arr + 4)

arr[0]          =>  *(arr + 0)
*(arr + 0)      =>  *(0 + arr)  // Since additions are associative

*(arr + 0)      =>  arr[0]
```

CODE ACADEMY

Задачи

1. Напишете програма, която въвежда и принтира елементите на масив, използвайки указател.

2. Напишете програма, която копира един масив в друг, използвайки указатели.

3. Напишете програма, която разменя елементите на два еднакви по размер масива, използвайки указатели.

CODE ACADEMY

```
#include <stdio.h>
#define SIZE 3     // Maximum array size
                                  /* from b to a
*/
int main()
{
    int a[SIZE]={1,2,3};
    int b[SIZE]={10,20,30};

    int i;

   for(i=0; i < SIZE; i++)
   {
       *(a+i)=*(b+i);
   }
   for(i=0; i < SIZE; i++)
   {
       printf("%d\t",*(a+i));
   }

    return 0;
}
```

CODE ACADEMY

Задача 3 – решение

```
void swap(int *x,  int *y)
{
        int tmp;
        tmp = *x;
        *x = *y;
        *y = tmp;
}

int main()
{
        int a=3, b=4;

        swap( &a,  &b );

        printf("a is %d\n", a);
        printf("b is %d\n", b);
}
```

CODE ACADEMY

Задачи

4. Напишете програма, която обръща местата на елементите в един масив, използвайки указатели.

5. Напишете програма за търсене на определен елемент в даден масив, използвайки указатели. Дали дадено число се среща в елементите на масива.

6. **Напишете програма, която приема 10 цели числа от клавиатурата, запазва ги в масив и ги принтира в обратен ред.**

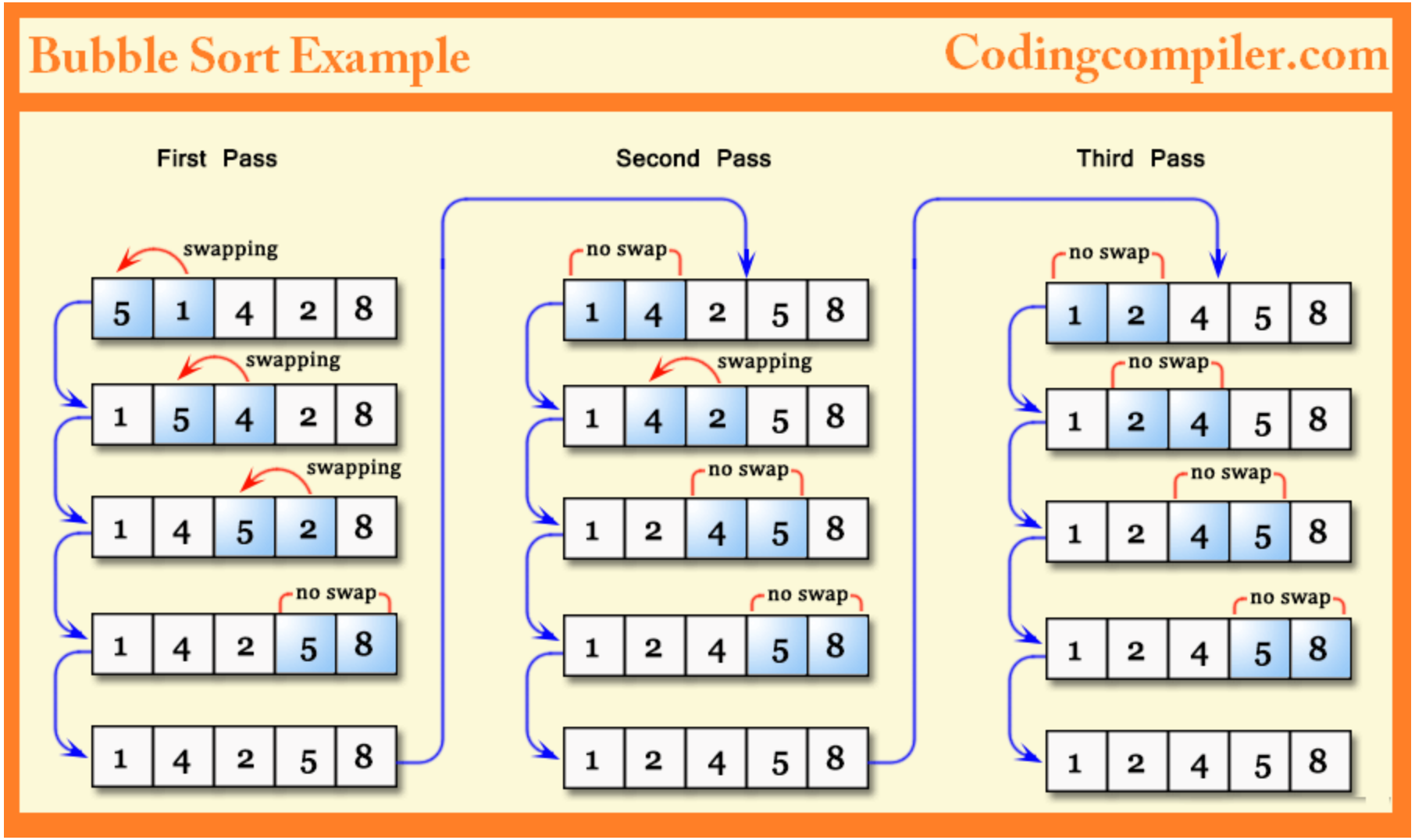
CODE ACADEMY

Задачи

7. **Напишете програма, която събира две числа, използвайки указатели.**

8. **Напишете програма, която сортира масив, въведен от клавиатурата, използвайки указатели. За целта използвайте метода на мехурчето, Bubble sort**

CODE ACADEMY



CODE ACADEMY

```
#include <stdio.h>
#define SIZE 10
void swap(int *x,  int *y)
                {
                                int tmp;
                                tmp = *x;
                                *x = *y;
                                *y = tmp;
                }
int main()
{
   int a[SIZE]={10,22,3, 56, 4,56, 6, 7,23, 1};

   int i,j;

   int number=SIZE;
   while(number>1)
   {
      for(i=0; i < number-1; i++)
      {

         if( *(a+i) > *(a+i+1) )
            swap(a+i, a+i+1);
      }
      number--;
   }
  for(i=0; i < SIZE; i++)
  {
     printf("%d\t",*(a+i));
  }
 }
```

CODE ACADEMY

Задачи

1.9. Напишете програма, която намира максималния и минималния елементи в масив, използвайки указатели.

10. Напишете програма, която да изчисли факториел на дадено число, използвайки указатели.

CODE ACADEMY

```
#include <stdio.h>
#define SIZE 10

int maxElement(int a[], int n)
{
    int i;
    int max=*a;
    for(i=1; i< SIZE; i++)
    {
        if(max< *(a+i))
            max= *(a+i);
    }
    return max;
}

int main()
{
    int a[SIZE]={10,22,3, 56, 4,56, 6, 7,23, 1};

    printf("Max=%d", maxElement(a,SIZE));

}
```

CODE ACADEMY

```
#include <stdio.h>

long fact(int n)
{
    long result=1;
    int i;
    for(i=1;i<=n;i++)
    {
        result=result*i;
    }
    return result;
}

int main()
{
  int number;

  int *p=&number;

  scanf("%d", p);

  long factoriel = fact(*p);

  printf("%ld",factoriel);

}
```

CODE ACADEMY

Задачи за самостоятелна работа

1. Прочетете масив от цели числа от клавиатурата.
   Копирайте въведения масив във втори масив, като умножите стойността на всеки елемент по 2. Принтирайте двата масива.

2. Напишете програма, която запълва масив от 20 елемента с произволно избрани цели числа. За целта използвайте функцията `rand()`.
   Въведете едно цяло число от клавиатурата. Проверете дали въведеното число от клавиатурата е намерено в масива. Принтирайте масива и отговора дали числото е намерено в масива.

CODE ACADEMY

Задачи за самостоятелна работа

3. Напишете програма, която принтира уникалните елементи от масив с цели числа, въведени от клавиатурата. За целта, някои от числата трябва да се повтарят.

4. Напишете програма, която да брои колко пъти се среща едно число в даден масив от цели числа. Масивът трябва да бъде въведен от клавиатурата. Принтирайте резултатите.

CODE ACADEMY

Задачи за самостоятелна работа

Задача 5.Дефинирайте едномерен масив int с 10 елемента = {100,90,80,70,60,50,40,30,20,10}.
Дефинирайте пойнтер, който ще сочи към масива.

Достъпете 3 тия елемент от масива и му задайте стойност 5.
Достъпете 4 тия елемент и му задайте стойност 33.
Достъпете 5 тия елемент и го намалете със 7.
Достъпете 7 мия елемент и го увеличете с 10.
Достъпете 10 тия елемент и го умножете по 3.
Изпишете го по двата възможни начина и закоментирайте единия.
Принтирайте на екрана всички елементи на масива.