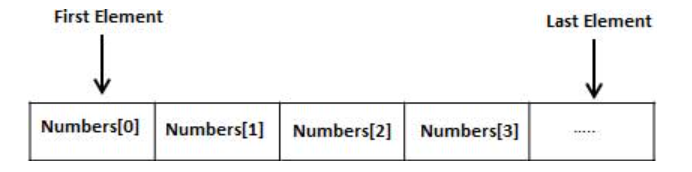
# Обработка на масиви. Работа с масиви и указатели. Обмен на данни (базови типове, масиви) между функции. Разработване на програми с масиви

# Масиви

Масивите са структура за съхранение на данни, която може да пази последователно разположени елементи, еднакви по размер, от един и същи тип. Масива попада към структурите от данни, но по – често се разглежда като съвкупност от променливи от един и същ тип. Вместо да се декларират отделни променливи една по една събирането им в една идея позволява динамичност при решаване на стандартни задачи и възможност за разглеждане на по – сложни проблеми. Конкретен елемент от масива се достъпва чрез така нареченият индекс – поредния номер от положението му спрямо началото на данните. Масива съдържа последователни адреси от паметта, като най – малкият адрес е началото на масива, а всеки следващ се променя чрез отместването с един елемент.



# Деклариране на масив

Декларацията на масив изисква да посочим типа данни и броят елементи, които да се съдържат в масива.

***<тип> име [брой елементи]***

Така обявеният масив се нарича едномерен. Съществуват и многомерни масиви, които ще разгледаме в следващото упражнение. Броя елементи е константен размер, като може да бъде всеки целочислен тип данни, стига да е **положително** число. Типа данни може да бъде всеки валиден тип в езика.

Пример:

Деклариране на масив от реални числа с 10 елемента: 😀

double balance[10];

По този начин вече имаме променлива която се казва balance и в себе си съдържа не една а 10 променливи, които могат да съдържат double числа.

# Инициализиране на масив

Можем да инициализираме масив, като даваме стойности една по една или наведнъж.

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Броя елементи в скобите {} не може да надвишава подадените като размер на масива, случая 5.

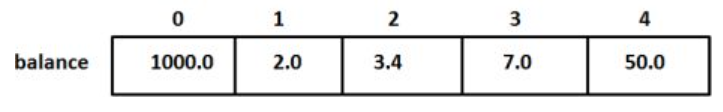
При инициализация с подаване на елементи може да си позволим да пропуснем броя елементи в [], като ще приеме броя на подадените стойности в {}.

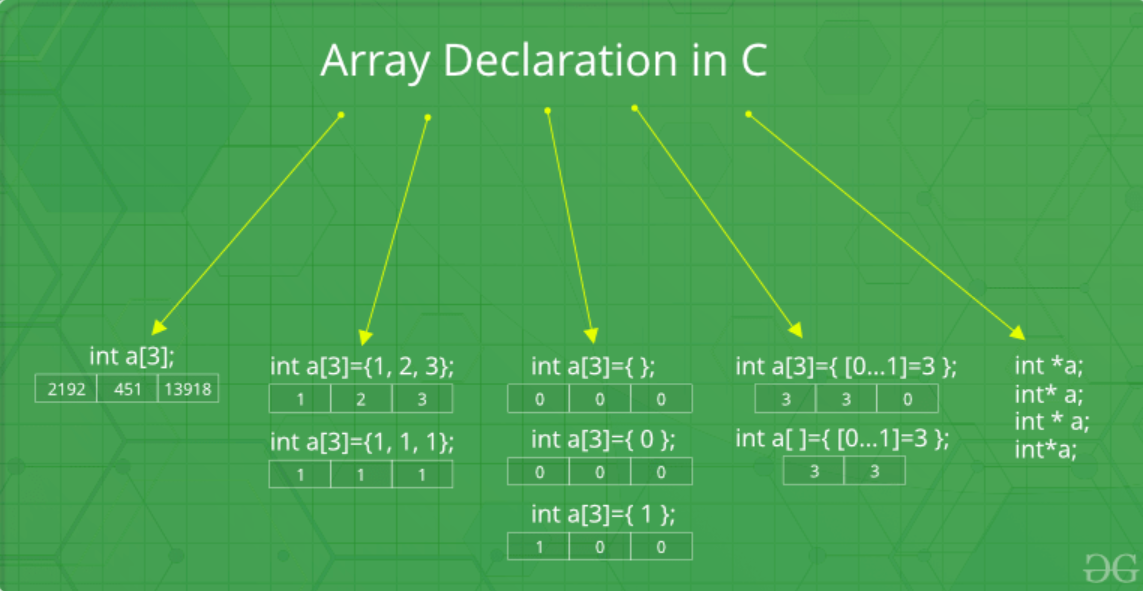
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Задаването на един елемент на масива се получава като се обърнем към индекса на масива и му присвоим стойност.

balance[4] = 50.0;

Тук даваме на 5-ят елемент стойност 50. Всеки масив започва своето индексиране от 0 и съдържа елементи до индекс N-1. Нулевият индекс се нарича базов индекс, а последният достъпен индекс е със стойност броя елементи минус 1.





# Достъпване на елементите

Достъп до елемент от масива се получава като изберем индекс от него, поставяйки го в [] комбинирани с името на масива.

double salary = balance[9];

Тук достъпваме 10 тия елемент от масива balance и го подаваме на променливата salary.

Примерен код за използване на масиви:

#include <stdio.h>

int main () {

int n[ 10 ]; /\* n is an array of 10 integers \*/

int i,j;

/\* initialize elements of array n to 0 \*/

for ( i = 0; i < 10; i++ ) {

n[ i ] = i + 100; /\* set element at location i to i + 100 \*/

}

/\* output each array element's value \*/

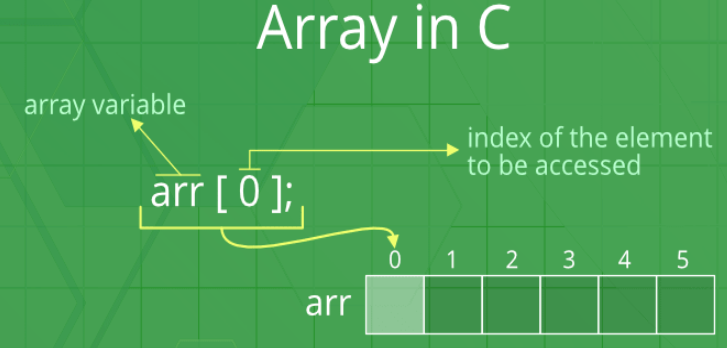
for (j = 0; j < 10; j++ ) {

printf("Element[%d] = %d\n", j, n[j] );

}

return 0;

}



Името на масива всъщност е и указател към началото му. Променянето на индекса на масива се изразява в промяна мястото от началното положение – отстояване на елемента от началото на масива. В този случай може да фигурират и различни записи за използване на маси ва – чрез добавяне на стойност към указателя сочещ началото.

Arr[5] -> \*(arr+5) -> 5[arr]

Разглеждане адресите на масива и доказателство за последователността на елементите:

**int** arr[5], i;

**printf**("Size of integer in this compiler is %lu\n",

**sizeof**(**int**));

**for** (i = 0; i < 5; i++)

        // The use of '&' before a variable name, yields

        // address of variable.

**printf**("Address arr[%d] is %p\n", i, &arr[i]);

Когато използваме масиви трябва да внимаваме за размерността им. Ако излезем извън масива няма проверка, която да ни предупреди, а изкарва произволна според паметта стойност(неочаквана). Когато пробваме да променим такава памет или паметта е защитена приложението спира да работи с неочакван изход.

# Масиви и указатели

Указателите се ползват за заделянето на динамични масиви и подаването на масиви към функции. Има разлика между указател и името на масива. В най-общата си част за разлика от указателя името на масива е елемент, който не е редно да се измества или променя, защото е дефиниран с идеята да определя конкретна памет.

Пример, за разликата между двете неща:

#include <stdio.h>

**int** main()

{

**int** arr[] = { 10, 20, 30, 40, 50, 60 };

**int**\* ptr = arr;

    // sizof(int) \* (number of element in arr[]) is printed

**printf**("Size of arr[] %ld\n", **sizeof**(arr));

    // sizeof a pointer is printed which is same for all

    // type of pointers (char \*, void \*, etc)

**printf**("Size of ptr %ld", **sizeof**(ptr));

**return** 0;

}

Изход:

Size of arr[] 24

Size of ptr 8

Смяната на адреса към който сочи началото на масива не е позволена операция:

#include <stdio.h>

**int** main()

{

**int** arr[] = {10, 20}, x = 10;

**int** \*ptr = &x; // This is fine

   arr = &x;  // Compiler Error

**return** 0;

}

Изход:

Compiler Error: incompatible types when assigning to

type 'int[2]' from type 'int \*'

Чрез указател можем да изкарваме стойности от самият масив:

#include <stdio.h>

**int** main()

{

**int** arr[] = { 10, 20, 30, 40, 50, 60 };

    // Assigns address of array to ptr

**int**\* ptr = arr;

**printf**("Value of first element is %d", \*ptr);

**return** 0;

}

Изход:

Value of first element is 10

Както беше упоменато по – горе може да използваме аритметиката позволена за указателите за да изведем стойности от масив:

#include <stdio.h>

**int** main()

{

**int** arr[] = {10, 20, 30, 40, 50, 60};

**int** \*ptr = arr;

**printf**("arr[2] = %d\n", arr[2]);

**printf**("\*(arr + 2) = %d\n", \*(arr + 2));

**printf**("ptr[2] = %d\n", ptr[2]);

**printf**("\*(ptr + 2) = %d\n", \*(ptr + 2));

**return** 0;

}

Изход:

arr[2] = 30

\*(arr + 2) = 30

ptr[2] = 30

\*(ptr + 2) = 30

Елементите на масив се подават винаги като указател, независимо дали ползваме указател или [].

#include <stdio.h>

**int** fun(**int** ptr[])

{

**int** x = 10;

    // size of a pointer is printed

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

    // This allowed because ptr is a pointer, not array

    ptr = &x;

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

**return** 0;

}

// Driver code

**int** main()

{

**int** arr[] = { 10, 20, 30, 40, 50, 60 };

    // size of a array is printed

**printf**("sizeof(arr) = %d\n", (**int**)**sizeof**(arr));

    fun(arr);

**return** 0;

}

Изход:

sizeof(arr) = 24

sizeof(ptr) = 4

\*ptr = 10

# Работа с функции

Ако искаме да подадем масив като аргумент на функция можем да го направим по един от следните начини:

Подаване като указател:

void myFunction(int \*ptr) {

.

.

.

}

Подаване на масив с размерността му:

void myFunction(int param[10]) {

.

.

.

}

Подаване на масив без размера му:

void myFunction(int param[]) {

.

.

.

}

Пример:

double getAverage(int arr[], int size) {

int i;

double avg;

double sum = 0;

for (i = 0; i < size; ++i) {

sum += arr[i];

}

avg = sum / size;

return avg;

}

#include <stdio.h>

/\* function declaration \*/

double getAverage(int arr[], int size);

int main () {

/\* an int array with 5 elements \*/

int balance[5] = {1000, 2, 3, 17, 50};

double avg;

/\* pass pointer to the array as an argument \*/

avg = getAverage( balance, 5 ) ;

/\* output the returned value \*/

printf( "Average value is: %f ", avg );

return 0;

}

Както виждаме от кода размера на масива не е от значение, защото работим с референция към памет.

Връщане на указател от функция:

В езика не можем да върнем всички стойности от масив, но можем да върнем указател към тези стойности в паметта:

int \* myFunction() {

.

.

.

}

Пример:

#include <stdio.h>

/\* function to generate and return random numbers \*/

int \* getRandom( ) {

static int r[10];

int i;

/\* set the seed \*/

srand( (unsigned)time( NULL ) );

for ( i = 0; i < 10; ++i) {

r[i] = rand();

printf( "r[%d] = %d\n", i, r[i]);

}

return r;

}

/\* main function to call above defined function \*/

int main () {

/\* a pointer to an int \*/

int \*p;

int i;

p = getRandom();

for ( i = 0; i < 10; i++ ) {

40 printf( "\*(p + %d) : %d\n", i, \*(p + i));

}

return 0;

}

В С не можем да върнем стойност, която да е от локалната памет на функцията!

Задачи:

Да се състави програма, чрез която се въвеждат 7 цели числа от интервала [-5000..5000] в едномерен масив. Програмата да изведе общата сума на въведените числа.

Пример: 89, -123, 45, 196, 1204, 0, 112  
Изход: 1523

Да се състави програма, чрез която се въвеждат 7 цели числа от интервала [-5000..5000] в едномерен масив. Програмата да изведе най-голямото от въведените числа.

Пример: 89, -123, 45, 196, 1204, 0, 112  
Изход: 1204

Да се състави програма, чрез която се въвеждат 7 цели числа от интервала [-5000..5000] в едномерен масив. Програмата да изведе средната стойност на въведените числа.

Пример: 89, -123, 45, 196, 1204, 0, 112  
Изход: 217.571

Да се състави програма, чрез която се въвеждат 7 цели числа от интервала [-5000..5000] в едномерен масив. Програмата да изведе числото, което е най-близко до средната стойност на въведените числа и къде се намира.

Пример: 1,2,3,4,5,6,7  
Изход: средна стойност 4, най-близка стойност 4 на 4-то място