**Arrays**

* Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
* To store homogeneous data elements for particular elements.
* To create an array, define the data type (like int) and specify the name of the array followed by **square brackets []**.
* **int myNumbers[] = {25, 50, 75, 100};**

To access the element of an array :

* To access an array element, refer to its index number.
* Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Int a[N]; (wrong ) we should not write char

**Compile time initialization :**

Int a[5] ={1,-2,11,5,7}

Int a[] ={0,1,2,3,4}

Int a[5] = {1,2,3}

Int a[5] ={0}

Int a[5] ={} (error)

Char b[4] ={‘v’,’f’,’t’,’u’}

**Runtime :**

Int a[5];

Printf(“”enter number of array: “);

For(i=0;i>5;i++)

{

Scanf(“%d”,&a[i]);

}

**Memory representation and accessing of array :**

Formula for accessing element = (**Base address +(index\*size of array)**)

**2D Arrays in C :**

**Needs :** Datatype nameOfArray [row size][coloum size]

int marks [3] [5];

At compile time : int a[2][3]={0,0,1,1,1};

int a[2][3]={

{0,0,1},

{1,1,1};

}

**Types of Arrays**

1. Static Array

2. Dynamic Array

3. Stretchable Array

4. Mutable Array

Static Array: The size of array is known before compilation time.

Ex: int arr[5]; àstack memory

Dynamic Array: The size of array is given at run time.

Ex: malloc() , calloc(), realloc() à heap memory

Stretchable Array: The array size can be increased/decreased depending on need for dynamic.

Ex: malloc(), calloc(), realloc()

Mutable Array: The size of array is known at the time of linking before execution.

Ex: The declaration is in another file and execution in another file

Q. WAP to store odd numbers in an array btw n an m

Q. Multiplication of 2 numbers

Int a[2] ={1,2}

Int a[2][3]={ {1,2,3}, {4,5,6}};

Int a[2][3]= {1,2,3,4,5,6} àcompiler automatically does the arrangement

We need to define macros for rows and columns

Y[y[x]] is acceptable

We can specify the index of array before square brackets : 2[a] or a[2]

Functions

1. Standard lib Functions

Printf, sqrt, abs, pow

2. User defined Functions

Function name starts with small letters and for special functions starts with \_

// Declaration/Prototyping : (in .h files)

Rdt fName(input datatypes); àsemicolon indicates no body

// Definition (in .c files)

Rdt fName(input args)

{

Sts;

return rdt;

}

In header files we can use “” if the header file is in same directory and no -I is required at compilation

When a function is invoked, it returns to either next line or same line

Generally, If there’s a return value then same line and if it has no return value then next line

So, in the same way for the return address is very important in stack frames

Arrays and pointers are almost same

· Functions2.c (observe it by changing the [] with \*)

To know the size of array/length of array

· Int CAP= sizeof(a)/sizeof(a[0]);

When we are passing the array we need to pass the array size(CAP)

Nesting function:

Main()

{

Fun1();

}

Fun1()

{ fun2(); }

Fun2()

{

}

Recursive function

Leads to stack overflow

// Recursive function example

#include <stdio.h>

int main() {

int res= f(5);

printf("\nRes=%d\n\n",res);

return 0;

}

int f(int v){

if(v==0)

return 1;

v--;

f(v);

printf("Value in func:%d\n",v);

return v;

}

Output:

V value in func: 0

V value in func: 1

V value in func: 2

V value in func: 3

V value in func: 4

Res=4

The difference between function stack frames and recursive func stack frames is the variables will have local address in func stack frames whereas no multiple local address for recursive func stack frames.

In recursive func the address is not destroyed it is the same address throughout the function