Itr=1

LoopLabel:

Printf(“\nitr=%d”,itr);

itr++;

if(itr >10)

goto exitLabel;

else

goto LoopLabel;

exitLabel:

printf(“\n\n”);

Goto is like a jump statement

The name of label should be at starting of line.

**Arrays**

Derivated basic datatype

Arrays should be on a single/particular purpose.

Represented by [ ]

It can have only homogeneous data.

Consecutive memory

//Declaration: Datatype Arrname[Size];

int arr[10];

float arr1[10];

//Access Elements: ArrName[indexValue]= value;

Why index starts from 0?

Array works on a formula**: BaseAddress+(IndexValue\*sizeof(datatype))**

So, it should start from 0 else it will go back which is wrong

Index values should always be whole numbers(integers)

Int a=10;

Int b[2]= {1,2};

Int c=20;

Printf(“%d%d”,b[0],b[1]);

Printf(“%d%d”,b[2],b[-1]); // output is 10 20

The output is having value bcoz c is not checking out of bound.

Here the memory allocation is bottom to top.

For eg: c-🡪2000, b[0] 🡪2002, b[1]🡪2004, a🡪2006

So as mentioned above the values b[2] and b[-1] follows the formula and assigns and access the values as all are of same type. If a and c are different data types, we get garbage value.

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]; 🡪stack memory

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

Ex: malloc() , calloc(), realloc() 🡪 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} 🡪compiler automatically does the arrangement

We need to define macros for rows and columns

A screenshot of a computer program

Description automatically generated

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

1. User defined Functions

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

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

Rdt fName(input datatypes); 🡪semicolon 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]);**

A screenshot of a computer program

Description automatically generated

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