**ARRAYS**

Arrays are meant for a particular task.

[]---it indicates that we are using

It contain only homogeneous elements.(Similar data type elements)

Arrays are stored in concecutive memory location.(continuous manner)

**Syntax: datatype ArrayName[capacity/size];**

**EX: int arr[10];**

All elements should be work for only one purpose.

Struct Emp

{

int id;

char name[20];

};

Struct Emp e[10]

**Access Elements of array to store or evaluate:**

**Syntax: ArrName[indexvalue] = value;**

Index value starts from 0 to capacity-1.

The index values should always be integer values.

Base Address+(Index Value \* Size of(dt))

**Why index starts with 0?**

arr[2]=20;

There are 4 types of arrays:

1. Static array

2. Dynamic array

3. Stretchable array

4. Mutable array

**1. STATIC ARRAY:** The size of the array is known before to the compilation time.

**EX:** int arr[5];

**2. DYNAMIC ARRAY:** The size of the array is allocated or known at run time.

**EX:** malloc, calloc, realloc ---defined in stdlib.h

**3.STRETCHABLE ARRAY**: Sized of the array is increased or decreased depending on the need for dy.array.

**EX:** malloc, calloc, realloc

**4.MUTABLE ARRAY:** The size of the array is known/alloated at the time of linking and before execution.

**EX:**

#include<stdio.h>

int main()

{

int intArr[5];

printf(“\NBA of array =%u”,&intArr[0]);

printf(“\nAddress of intArr[0]=%u and Value=”%d”, &intArr[0],intArr[0]);

printf(“\nAddress of intArr[1]=%u and Value=”%d”, &intArr[1],intArr[1]);

printf(“\nAddress of intArr[2]=%u and Value=”%d”, &intArr[2],intArr[2]);

printf(“\nAddress of intArr[3]=%u and Value=”%d”, &intArr[3],intArr[3]);

printf(“\nAddress of intArr[4]=%u and Value=”%d”, &intArr[4],intArr[5]);

intArr[0]=10;

intArr[1]=11;

intArr[2]=12;

intArr[3]=13;

intArr[4]=14;

printf(“\NBA of array =%u”,&intArr[0]);

printf(“\nAddress of intArr[0]=%u and Value=”%d”, &intArr[0],intArr[0]);

printf(“\nAddress of intArr[1]=%u and Value=”%d”, &intArr[1],intArr[1]);

printf(“\nAddress of intArr[2]=%u and Value=”%d”, &intArr[2],intArr[2]);

printf(“\nAddress of intArr[3]=%u and Value=”%d”, &intArr[3],intArr[3]);

printf(“\nAddress of intArr[4]=%u and Value=”%d”, &intArr[4],intArr[5]);

}

**Write a program to store odd numbers in an array between n and m?**

**#include <stdio.h>**

int main() {

int n, m;

printf("Enter the value of n: ");

scanf("%d", &n);

printf("Enter the value of m: ");

scanf("%d", &m);

int count=0;

int arr[50];

for(int i=n;i<m;i++)

{

if(i%2==1)

{

arr[count]=i;

count++;

}

}

printf("count: %d\n", count);

for(int j=0;j<count;j++)

{

printf("%d ", arr[j]);

}

return 0;

}

**2dimentional array---- dt arrName[ROW][COL]**

**int a[2][3];**

**2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012**

**1 2 3 4 5 6**

**a[0][0] a[0][1] a[0][2] a[1][0] a[1][1] a[1][2]**

**printf (“\n%d”,a[1][2]);**

**EX:**

/\* 2D ARRAY \*/

#include<stdio.h>

#define ROW 2

#define COL 3

int main()

{

int a[ROW][COL] = {{1,2,3}, {4,5,6}};

printf (“\n%d”,a[1][2]);

printf(“\n\n”);

return 0; }

**EX:**

int main()

{

int a1[ROW][COL] = {{1,2,3}, {4,5,6}};

int a2[ROW][COL] = {1,2,3,4,5,6};

printf (“\n%d”,a2[0][2]);

printf(“\n\n”);

return 0; }

**EX:**

#include <stdio.h>

#define ROW 2

#define COL 3

int main() {

int a1[ROW][COL] = {{1,2,3}, {4,5,6}};

int a2[ROW][COL] = {1,2,3,4,5,6};

int riv,civ;

// printf ("\n%d",a2[0][2]);

for(riv=0;riv<ROW;riv++)

{

for(civ=0;civ<COL;civ++)

printf("%d ",a2[riv][civ]);

printf("\n");

}

return 0;

}

**EX:SCANING THE ELEMENTS**

int main()

{

int a1[ROW][COL] = {{1,2,3}, {4,5,6}};

int a2[ROW][COL] = {1,2,3,4,5,6};

int riv,civ;

printf (“\n%d”,a2[0][2]);

for(riv=0;riv<ROW;riv++)

{

for(civ=0;civ<COL;civ++)

scanf(“%d ”,a2[riv][civ]);

}

for(riv=0;riv<ROW;riv++)

{ for(civ=0;civ<COL;civ++)

printf(“%d ”,a2[riv][civ]);

printf(“\n”);

}

printf(“\n\n”);

return 0; }

**EX:**

#include <stdio.h>

#define CAP 100

Int main()

{

int a[CAP], i;

int mid, countOdd, countEven;

int m=51, n=103;

mid = (n-m)/2;

printf(“\nmid=%d”,mid);

for(i=m,countOdd=0,countWven=mid;i<=n;i++)

{ if(i%2!=0)

{ a[counted] = i;

countOdd++;

}

else

{ printf(“\n%d”,i);

a[countEven]=i;

countEven++;

}

}

for(int i=0;i<countEven

}

**EX: REVERSE THE ARRAY**

#include <stdio.h>

#define CAP 10

int main() {

int i=0;

int mid;

int t;

int a[CAP]={1,2,3,4,5,6,7,8,9,10};

for(i=0;i<CAP;i++)

printf(" %d ",a[i]);

for(i=0,mid=CAP/2;i<mid;i++)

{

t=a[i];

a[i]=a[(CAP-1)-i];

a[(CAP-1)-i] = t;

}

printf("\n");

for(int i=0;i<CAP;i++)

{

printf(" %d ",a[i]);

}

return 0;

}

**EX: printf("\n%d\n",a[a[2]]);**

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

**FUNCTIONS:**

Certain task to be performed for a particular task.

**1.Standard library functions------**printf, sqrt, abs, pow----bulild-in functions

**2.Userdedined----------**User is defining his/her own task to be performed.

**SYNTAX:**

rdt(return datatype) fName(input args)

{

Sts;

return rdt;

}

No need of declare the name of a variable while we declaring the prototype.

Implementation is in .c file.

Interface is in .h file

**EX:**

**int add(int val1, int val2)**

**{ int result =val1+val2;**

**return result;**

**}**

**EX:**

**ISPRIME.H**

#ifndef ISPRIME\_H

#define ISPRIME\_H

#define True 1

#define False 0

int isPrime(int);

#endif

**ISPRIME.C**

#include<stdio.h>

#include <isPrime.h>

int isPrime(int val)

{

int it, flag=0;

if(val <=1)

return False;

for(it=2;i<=val/2;it++)

{

if(val%it == 0)

{

flag=1;

break;

}

}

if(flag ==1 )

return False;

else

return True;

}

**MAIN.C**

#include <stdio.h>

#include <isPrime.h>

int main() {

int num;

scanf(“%d “,&num);

if(isPrime(num) == True)

printf("\n%d is a prime number:", num);

else

printf("\n%d is not a prime number:",num);

printf("\n\n");

return 0;

}

**EX:**

#include <stdio.h>

#include <stdlib.h>

int changeValue(int [], int);

void disp(int [], int);

int main()

{

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

printf("\nBA of array(main): %u",&a[0]);

disp(a,5);

return 0;

}

void disp(int arr[3], int n)

{

int i;

printf("\nBA of arr (disp): %u\n",&arr[0]);

for(int i=0;i<n;i++)

printf("%d ",arr[i]);

printf("\n\n");

}

**EX:**

#include <stdio.h>

#include <stdlib.h>

int changeValue(int \*, int);

void disp(int [], int);

int main()

{

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

printf("\nBA of array(main): %u",&a[0]);

disp(a,5);

return 0;

}

void disp(int arr[], int n)

{

int i;

printf("\nBA of arr (disp): %u\n",&arr[0]);

for(int i=0;i<n;i++)

printf("%d ",arr[i]);

printf("\n\n");

}

**EX:**

int main()

{

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

int CAP= sizeof(a)/sizeof(a[0]);

printf("\nBA of array(main): %u",&a[0]);

printf("\nCAP:%d",CAP);

disp(a,CAP);

return 0; }

**EX: CHANGE THE VALUE**

#include <stdio.h>

#include <stdlib.h>

int changeValue(int [], int);

void disp(int \*, int);

int main()

{

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

int CAP= sizeof(a)/sizeof(a[0]);

// printf("\nBA of array(main): %u",&a[0]);

printf("\nCAP:%d\n",CAP);

disp(a,CAP);

changeValue(a,3);

disp(a,CAP);

return 0;

}

void disp(int arr[], int n)

{

int i;

//printf("\nBA of arr (disp): %u\n",&arr[0]);

for(int i=0;i<n;i++)

printf("%d ",arr[i]);

printf("\n\n");

}

int changeValue(int \*arr,int key)

{

int i;

int flag=0;

int CAP=0;

printf("\nCAP=%d\n",sizeof(arr)/sizeof(arr[0]));

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

{

if(arr[i] == key)

{

arr[i] =40;

flag=1;

}

}

if(flag == 0)

return 1;

else

return 0;

}

Nesting of the function is not similar to the recursive function.

In recursive we have the stack overflow because is calls multiple times.

**EX: RECURSIVE FUNCTION**

#include <stdio.h>

in f(int);

int main() {

int res=f(5);

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

return 0;

}

int f(int v)

{

f(v);

}

There is not end for this program so it leads to stack overflow.If we terminate the code we will kill it.

**TO OVERCOME THIS OVERFLOW**

#include <stdio.h>

int f(int);

int main() {

int res=f(5);

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

return 0;

}

int f(int v)

{

//printf("%d ",v);

if (v == 0)

return 1;

v--;

f(v);

printf("\n V value in the funt: %d\n",v);

return v;

}

USE THIS CODE FOR FEBINCICC AND FACTORIAL

**OUTPUT:** 5 4 3 2 1 0

V value in the funt: 0

V value in the funt: 1

V value in the funt: 2

V value in the funt: 3

V value in the funt: 4

Res=4

In recurive it does not destroy the address it only destroy the value. The address is same for all values.