**Factorial using Recursion**

#include<stdio.h>

void main( )

{

int factorial(int);

int n,f;

printf("Enter the number: ");

scanf("%d",&n);

f=factorial(n);

printf("Factorial of the number is %d",f);

}

int factorial(int n)

{

int f;

if(n==0)

return 1;

else

f=n\*factorial(n-1);

return f;

}

**GCD using Recursion**

#include <stdio.h>

int hcf(int n1, int n2);

int main() {

int n1, n2;

printf("Enter two positive integers: ");

scanf("%d %d", &n1, &n2);

printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1, n2));

return 0;

}

int hcf(int n1, int n2) {

if (n2 != 0)

return hcf(n2, n1 % n2);

else

return n1;

}

**Storage Classes in C**

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

* Automatic
* External
* Static
* Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Storage Classes** | **Storage Place** | **Default Value** | **Scope** | **Lifetime** |
| auto | RAM | Garbage Value | Local | Within function |
| extern | RAM | Zero | Global | Till the end of the main program Maybe declared anywhere in the program |
| static | RAM | Zero | Local | Till the end of the main program, Retains value between multiple functions call |
| register | Register | Garbage Value | Local | Within the function |

## **Automatic**

* Automatic variables are allocated memory automatically at runtime.
* The visibility of the automatic variables is limited to the block in which they are defined.

The scope of the automatic variables is limited to the block in which they are defined.

* The automatic variables are initialized to garbage by default.
* The memory assigned to automatic variables gets freed upon exiting from the block.
* The keyword used for defining automatic variables is auto.
* Every local variable is automatic in C by default.

#include <stdio.h>

void fun()

{

auto int a=10;

++a;

printf(“\n%d”, a);

}

void main()

{

fun();

fun();

fun();

}

## **Static**

* The variables defined as static specifier can hold their value between the multiple function calls.
* Static local variables are visible only to the function or the block in which they are defined.
* A same static variable can be declared many times but can be assigned at only one time.
* Default initial value of the static integral variable is 0 otherwise null.
* The visibility of the static global variable is limited to the file in which it has declared.
* The keyword used to define static variable is static.

#include <stdio.h>

void fun()

{

static int a=10;

++a;

printf(“\n%d”, a);

}

void main()

{

fun();

fun();

fun();

}

## **Register**

* The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
* We can not dereference the register variables, i.e., we can not use &operator for the register variable.
* The access time of the register variables is faster than the automatic variables.
* The initial default value of the register local variables is 0.
* The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
* We can store pointers into the register, i.e., a register can store the address of a variable.
* Static variables cannot be stored into the register since we cannot use more than one storage specifier for the same variable.

#include<stdio.h>

Int main ()

{

int i, sum=0;

for(i=1;i<10;i++)

sum=sum+i;

i++;

printf(“%d”, sum);

}

## **External**

* The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
* The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
* The default initial value of external integral type is 0 otherwise null.
* We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
* An external variable can be declared many times but can be initialized at only once.
* If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

#include <stdio.h>

int main()

{

extern int a;

printf("%d",a);

}

#include <stdio.h>

int a;

int main()

{

extern int a; // variable a is defined globally, the memory will not be allocated to a

printf("%d",a);

}

#include <stdio.h>

int a;

int main()

{

extern int a = 0; // this will show a compiler error since we can not use extern and initializer at same time

printf("%d",a);

}

#include <stdio.h>

int main()

{

extern int a; // Compiler will search here for a variable a defined and initialized somewhere in the pogram or not.

printf("%d",a);

}

int a = 20;

**Program to print array in C**

#include <stdio.h>

int main() {

int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

int i;

//Print in original order

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

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

//Print in reverse order

for(i = 9; i >=0; i--)

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

return 0;

}

## **Iterative program to find the smallest and largest elements in an array**

#include<stdio.h>

int main()

{

int a[50],i,n,large,small;

printf(“\nEnter the number of elements : “);

scanf(“%d”,&n);

printf(“\nInput the array elements : “);

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

scanf(“%d”,&a[i]);

large=small=a[0];

for(i=1;i<n;++i)

{

if(a[i]>large)

large=a[i];

if(a[i]<small)

small=a[i];

}

printf(“\nThe smallest element is %d\n”,small);

printf(“\nThe largest element is %d\n”,large);

return 0;

}

ODD or EVEN

#include<stdio.h>

void odd\_or\_even(int a);

int main()

{

int my\_arr[] = {13,56,71,38,93}, i;

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

{

// passing one element at a time to odd\_or\_even() function

odd\_or\_even(my\_arr[i]);

}

// signal to operating system program ran fine

return 0;

}

void odd\_or\_even(int a)

{

if(a % 2 == 0)

{

printf("%d is even\n", a);

}

else

{

printf("%d is odd\n", a);

}

}

## **Sum of the array**

#include<stdio.h>

int main()

{

//let's assume the maximum array size as 100.

//initialize sum as 0. Otherwise, it will take some garbage value.

int arr[100], size, i, sum = 0;

//Get size input from user

printf("Enter array size\n");

scanf("%d",&size);

//Get all elements using for loop and store it in array

printf("Enter array elements\n");

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

scanf("%d",&arr[i]);

//add all elements to the variable sum.

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

sum = sum + arr[i]; // same as sum += arr[i];

//print the result

printf("Sum of the array = %d\n",sum);

return 0;

}

Delete element from array

#include <stdio.h>

int main()

{

int array[100], position, c, n;

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d elements\n", n);

for ( c = 0 ; c < n ; c++ )

scanf("%d", &array[c]);

printf("Enter the location where you wish to delete element\n");

scanf("%d", &position);

if ( position >= n+1 )

printf("Deletion not possible.\n");

else

{

for ( c = position - 1 ; c < n - 1 ; c++ )

array[c] = array[c+1];

printf("Resultant array is\n");

for( c = 0 ; c < n - 1 ; c++ )

printf("%d\n", array[c]);

}

return 0;

}

Coping Array to array

#include <stdio.h>

int main()

{

int a[5] = { 3, 6, 9, 2, 5 }, n = 5;

int b[n], i;

// copying elements from one array to another

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

b[i] = a[i];

}

// displaying first array before

// copy the elements from

// one array to other

printf("The first array is :");

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

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

}

// displaying array after copy

// the elements from one

// array to other

printf("\nThe second array is :");

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

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

}

return 0;

}

2-d 3\*3 array print in c

#include <stdio.h>

void main()

{

int arr1[3][3],i,j;

/\* Stored values into the array\*/

printf("Input elements in the matrix :\n");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

printf("element - [%d],[%d] : ",i,j);

scanf("%d",&arr1[i][j]);

}

}

printf("\nThe matrix is : \n");

for(i=0;i<3;i++)

{

printf("\n");

for(j=0;j<3;j++)

printf("%d\t",arr1[i][j]);

}

printf("\n\n");

}

**Addition of two 3\*3 matrix**

#include <stdio.h>

int main()

{

int a[3][3], b[3][3], result[3][3];

// Taking input using nested for loop

printf("Enter elements of 1st matrix\n");

for (int i = 0; i < 3; ++i)

for (int j = 0; j < 3; ++j)

{

scanf("%d", &a[i][j]);

}

// Taking input using nested for loop

printf("Enter elements of 2nd matrix\n");

for (int i = 0; i < 3; ++i)

for (int j = 0; j < 3; ++j)

{

scanf("%d", &b[i][j]);

}

// adding corresponding elements of two matrices

for (int i = 0; i < 3; ++i)

for (int j = 0; j < 3; ++j)

{

result[i][j] = a[i][j] + b[i][j];

}

// Displaying the sum

printf("Sum Of Matrix:\n");

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 3; ++j)

{

printf("%d\t", result[i][j]);

}

printf("\n");

}

return 0;

}

Develop, implement and execute a C program that reads two matrices A (m x n) and B (p x q) and Compute product of matrices A and B. Read matrix A and matrix B in row major order and in column major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only.

#**include**<stdio.h>

**int** main()

{

**int** i,j,k;

**float** a[3][3], b[3][3], mul[3][3];

printf("Enter elements of first matrix:\n");

**for**(i=0;i< 3;i++)

{

**for**(j=0;j< 3;j++)

{

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

scanf("%f", &a[i][j]);

}

}

printf("Enter elements of second matrix:\n");

**for**(i=0;i< 3;i++)

{

**for**(j=0;j< 3;j++)

{

printf("b[%d][%d]=",i,j);

scanf("%f", &b[i][j]);

}

}

**for**(i=0;i< 3;i++)

{

**for**(j=0;j< 3;j++)

{

mul[i][j] = 0;

**for**(k=0;k< 3;k++)

{

mul[i][j] = mul[i][j] + a[i][k]\*b[k][j];

}

}

}

printf("Multiplied matrix is:\n");

**for**(i=0;i< 3;i++)

{

**for**(j=0;j< 3;j++)

{

printf("%f\t", mul[i][j]);

}

printf("\n");

}

**return** 0;

}