## C++ Storage Class –

- Storage class is used to define the lifetime and visibility of a variable and/or function within a C++ program.

- Lifetime refers to the period during which the variable remains active and visibility refers to the module of a program in which the variable is accessible.

- **Storage Classes** are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.
-
- There are five types of storage classes, which can be used in a C++ program –

| | | | | |
|---|---|---|---|---|
| **auto** | **register** | **extern** | **static** | **mutable** |

| **Storage Class** | **Keyword** | **Lifetime** | **Visibility** | **Initial Value** |
|---|---|---|---|---|
| Automatic | auto | Function Block | Local | Garbage |
| Register | register | Function Block | Local | Garbage |
| Mutable | mutable | Class | Local | Garbage |
| External | extern | Whole Program | Global | Zero |
| Static | static | Whole Program | Local | Zero |

## Automatic Storage Class –

- It is the default storage class for all local variables.

- The auto keyword is applied to all local variables automatically.

- The auto keyword provides type inference capabilities, using which automatic deduction of the data type of an expression in a programming language can be done.

- This consumes less time having to write out things the compiler already knows.

- This feature also extends to functions and non-type template parameters.

```
{
        auto int y;
        float y = 3.45;
}
```

```
#include<iostream.h>

void autoStorageClass()
{
        // Declaring an auto variable
        // No data-type declaration needed
        auto a = 32;
        auto b = 3.2;
        auto c = “hello world";
        auto d = 'G';

        // printing the auto variables
        cout << a << " \n";
        cout << b << " \n";
        cout << c << " \n";
        cout << d << " \n";
}
```

```
void main()
{
        // To demonstrate auto Storage Class
        autoStorageClass();
}
```

## Register Storage Class –

- The register variable allocates memory in register than RAM.
- Its size is same of register size. It has a faster access than other variables.
- It is recommended to use register variable only for quick access such as in counter.

***Note:*** ***We can't get the address of register variable.***

- This storage class declares register variables which have the same functionality as that of the auto variables.
- The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.
- If a free register is not available, these are then stored in the memory only.

```
register int counter=0;
```

```
#include<iostream.h>

void registerStorageClass()
{
        // declaring a register variable
        register char b = 'G';

        // printing the register variable 'b'
        cout << "Value of the variable b"<< b;
}
                                              void main()
                                              {

                                                      // To demonstrate register Storage Class
                                                      registerStorageClass();
                                              }
```

## Extern Storage Class –

- The extern variable is visible to all the programs.

- It is used if two or more files are sharing same variable or function.

- Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.

- Also, a normal global variable can be made extern as well by placing the ‘extern’ keyword before its declaration/definition in any function/block.

```
extern int counter=0;
```

```
#include <iostream>

int x;
void externStorageClass()
{
	extern int x;
	cout << “Value is: “<< x << "\n";

	x = 2;
	cout << “Modified value is: “<< x;
}

void main()
{
	externStorageClass();
}
```

## Static Storage Class –

- The static variable is initialized only once and exists till the end of a program.
- It retains its value between multiple functions call.
- The static variable has the default value 0 which is provided by compiler.
- Thus, no new memory is allocated because they are not re-declared.
- Their scope is local to the function to which they were defined.
- Global static variables can be accessed anywhere in the program.

```
static int counter=0;
```

```
#include<iostream.h>
#include<conio.h>

void func()
{
        static int i=0;                        //static variable
        int j=0;                               //local variable
        i++;
        j++;
        cout<<"i=" << i<<" and j=" <<j<<endl;
}

void main()
{
        func();
}
```

## Mutable Storage Class –

- Sometimes there is a requirement to modify one or more data members of class/struct through const function even though you don’t want the function to update other members of class/struct.

- This task can be easily performed by using the mutable keyword.

- The keyword mutable is mainly used to allow a particular data member of const object to be modified.

- When we declare a function as const, this pointer passed to function becomes const.

- Adding mutable to a variable allows a const pointer to change members.

```
mutable int counter=0;
```