Storage Class:

Every variable in C++ has two features: type and storage class.

Type specifies the type of data that can be stored in a variable. For example: int, float, char etc.

And, storage class controls two different properties of a variable: lifetime (determines how long a variable can exist) and scope (determines which part of the program can access it).

Depending upon the storage class of a variable, it can be divided into 4 major types:

- Local variable
- Global variable
- Static local variable
- Register Variable

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

C++ uses 5 storage classes, namely:

1. auto
2. register
3. extern
4. static
5. mutable

Local Variable / Auto:

It is the default storage class for all local variables. The auto keyword is applied to all local variables automatically. auto can only be used within functions.

A variable defined inside a function (defined inside function body between braces) is called a local variable or automatic variable.
Its scope is only limited to the function where it is defined. In simple terms, local variable exists and can be accessed only inside a function.The life of a local variable ends (It is destroyed) when the function exits.

```
#include <iostream>
using namespace std;
void test();
int main()
{
   // local variable to main()
   int var = 5;
   test();
   // illegal: var1 not declared inside main()
   var1 = 9;
}

void test()
{
   // local variable to test()
   int var1;
   var1 = 6;
   // illegal: var not declared inside test()
   cout << var;
}
```

```
#include <iostream>
using namespace std;
void autoStorageClass()
{
   cout << "Demonstrating auto class\n";

   // Declaring an auto variable
   // No data-type declaration needed
```

```
    auto a = 32;
    auto b = 3.2;
    auto c = "GeeksforGeeks";
    auto d = 'G';

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

int main()
{
    // To demonstrate auto Storage Class
    autoStorageClass();

    return 0;
}
```

Output:
Demonstrating auto class
32
3.2
GeeksforGeeks
G

Global Variable / extern

If a variable is defined outside all functions, then it is called a global variable.

The scope of a global variable is the whole program. This means, It can be used and changed at any part of the program after its declaration.

Likewise, its life ends only when the program ends.

Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to

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

```
#include <iostream>
using namespace std;
// Global variable declaration
int c = 12;
void test();
int main()
{
        ++c;
        // Outputs 13
        cout << c <<endl;
        test();
        return 0;
}

void test()
{
        ++c;
        // Outputs 14
        cout << c;
}
```

```
#include <iostream>
using namespace std;

// declaring the variable which is to
// be made extern an initial value can
// also be initialized to x
int x;
void externStorageClass()
{

    cout << "Demonstrating extern class\n";

    // telling the compiler that the variable
    // x is an extern variable and has been
    // defined elsewhere (above the main
```

```
    // function)
    extern int x;

    // printing the extern variables 'x'
    cout << "Value of the variable 'x'"
         << "declared, as extern: " << x << "\n";

    // value of extern variable x modified
    x = 2;

    // printing the modified values of
    // extern variables 'x'
    cout
         << "Modified value of the variable 'x'"
         << " declared as extern: \n"
         << x;
}

int main()
{

    // To demonstrate extern Storage Class
    externStorageClass();

    return 0;
}
```

OUTPUT:
Demonstrating extern class
Value of the variable 'x'declared, as extern: 0
Modified value of the variable 'x' declared as extern:
2

Static Local Variable:

A static local variable exists only inside a function where it is declared (similar to a local variable) but its lifetime starts when the function is called and ends only when the program ends.

The main difference between local variable and static variable is that, the value of static variable persists the end of the program.

```
#include <iostream>
using namespace std;

void test()
{
        // var is a static variable
        static int var = 0;
        ++var;
        cout << var << endl;
}

int main()
{

        test();
        test();
        return 0;
}
```

Output:

1

2

```
#include <iostream>
using namespace std;

// Function containing static variables
// memory is retained during execution
int staticFun()
{
   cout << "For static variables: ";
   static int count = 0;
   count++;
   return count;
}

// Function containing non-static variables
// memory is destroyed
int nonStaticFun()
{
   cout << "For Non-Static variables: ";

   int count = 0;
```

```
    count++;
    return count;
}

int main()
{

    // Calling the static parts
    cout << staticFun() << "\n";
    cout << staticFun() << "\n";

    // Calling the non-static parts

    cout << nonStaticFun() << "\n";
    cout << nonStaticFun() << "\n";
    return 0;
}
```

OUTPUT:
For static variables: 1
For static variables: 2
For Non-Static variables: 1
For Non-Static variables: 1

Register Variable (Deprecated in C++11)

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.

Keyword register is used for specifying register variables.

Register variables are similar to automatic variables and exists inside a particular function only. It is supposed to be faster than the local variables.

If a program encounters a register variable, it stores the variable in processor's register rather than memory if available. This makes it faster than the local variables.

However, this keyword was deprecated in C++11 and should not be used.

```
#include <iostream>
using namespace std;
void registerStorageClass()
{
    cout << "Demonstrating register class\n";
    // declaring a register variable
    register char b = 'G';
    // printing the register variable 'b'
    cout << "Value of the variable 'b'"
         << " declared as register: " << b;
}
int main()
{
    // To demonstrate register Storage Class
    registerStorageClass();
    return 0;
}
```

OUTPUT:

```
Demonstrating register class
Value of the variable 'b' declared as register: G
```