







Storage Classes in C++ with Examples

C++ Storage Classes are used to describe the characteristics of a variable/function. It determines the lifetime, visibility, default value, and storage location which helps us to trace the existence of a particular variable during the runtime of a program. Storage class specifiers are used to specify the storage class for a variable.

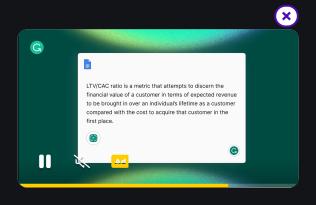
Syntax

To specify the storage class for a variable, the following syntax is to be followed:

storage_class var_data_type var_name;

C++ uses 6 storage classes, which are as follows:

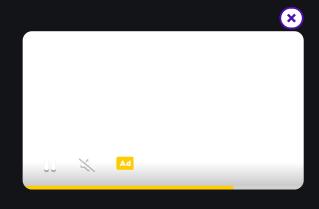
- 1. auto Storage Class
- 2. register Storage Class
- 3. extern Storage Class
- 4. static Storage Class
- 5. mutable Storage Class
- 6. thread_local Storage Class



C++ Storage Class

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extem	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage
Thread Local	thread_local	whole thread	Local or Global	Garbage

Below is a detailed explanation of each storage class:



1. auto Storage Class

The **auto storage class** is the default class of all the variables declared inside a block. The auto stands for automatic and all the local variables that are declared in a block automatically belong to this class.

Properties of auto Storage Class Objects

• Scope: Local

• **Default Value:** Garbage Value

• Memory Location: RAM

• Lifetime: Till the end of its scope

Example of auto Storage Class

```
C++
// variables
#include <iostream>
using namespace std;
void autoStorageClass()
    cout << "Demonstrating auto class\n";</pre>
    int a = 32;
    float b = 3.2;
    char* c = "GeeksforGeeks";
    char d = 'G';
    cout << a << " \n";
    cout << b << " \n";
    cout << c << " \n";
    cout << d << " \n";
int main()
    autoStorageClass();
```

```
return 0;
}
```

```
Demonstrating auto class
32
3.2
GeeksforGeeks
G
```

Note: Earlier in C++, we could use the **auto keyword** to declare the auto variables explicitly but after C++11, the meaning of **auto** keyword is changed and we could no longer use it to define the auto variables.

2. extern Storage Class

The **extern storage class** simply tells us that the variable is defined elsewhere and not within the same block where it is used (i.e. external linkage). Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. An extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere.

A normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block. The main purpose of using extern variables is that they ca different files which are part of a large program.

Properties of extern Storage Class Objects

• Scope: Global

 (\mathbf{x})

- **Default Value:** Zero
- Memory Location: RAM
- **Lifetime:** Till the end of the program.

Example of extern Storage Class

```
C++ Data Types C++ Input/Output C++ Arrays C++ Pointers C++ OOPs C++ STL C++ Interview Questions

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

```
int x;
void externStorageClass()
    cout << "Demonstrating extern class\n";</pre>
    extern int x;
    cout << "Value of the variable 'x'"</pre>
         << "declared, as extern: " << x << "\n";
    // value of extern variable x modified
    x = 2;
    // extern variables 'x'
    cout << "Modified value of the variable 'x'"</pre>
         << " declared as extern: \n"
         << x;
}
int main()
    externStorageClass();
```

```
return 0;
}
```

```
Demonstrating extern class

Value of the variable 'x'declared, as extern: 0

Modified value of the variable 'x' declared as extern:

2
```

For more information on how extern variables work, have a look at this <u>link</u>.

3. static Storage Class

The **static storage class** is used to declare static variables which are popularly used while writing programs in C++ language. Static variables have the property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope.

We can say that they are initialized only once and exist until the termination of the program. Thus, no new memory is allocated because they are not redeclared. Global static variables can be accessed anywhere in the program.

Properties of static Storage Class

Scope: Local

• **Default Value:** Zero

Memory Location: RAM

• **Lifetime:** Till the end of the program

Note: Global Static variables can be accessed

Example of static Storage Class

```
C++
#include <iostream>
using namespace std;
int staticFun()
     cout << "For static variables: ";</pre>
     static int count = 0;
     count++;
     return count;
}
int nonStaticFun()
     cout << "For Non-Static variables: ";</pre>
     int count = 0;
     count++;
     return count;
}
int main()
{
     cout << staticFun() << "\n";</pre>
     cout << staticFun() << "\n";</pre>
     cout << nonStaticFun() << "\n";</pre>
     cout << nonStaticFun() << "\n";</pre>
     return 0;
```

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

4. register Storage Class

The **register storage class** declares register variables using the '**register**' **keyword** which has 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.

An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

Properties of register Storage Class Objects

• Scope: Local

• **Default Value:** Garbage Value

• Memory Location: Register in CPU or RAM

• Lifetime: Till the end of its scope

Example of register Storage Class

```
C++

// C++ Program to illustrate the use of register
#include <iostream>
using namespace std;

void registerStorageClass()
{
```

```
Demonstrating register class

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

Note: The **register keyword** is deprecated in C++17 onwards.

5. mutable Storage Class

Sometimes there is a requirement to modify one or more data members of class/struct through the 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 k to allow a particular data member of a const obj

When we declare a function as const, this point becomes const. Adding a mutable to a variable a

change members.

Properties of mutable Storage Class

The mutable specifier does not affect the linkage or lifetime of the object. It will be the same as the normal object declared in that place.

Example of mutable Storage Class

```
C++
#include <iostream>
using std::cout;
class Test {
public:
    int x;
    mutable int y;
    Test()
         x = 4;
         y = 10;
};
int main()
    const Test t1;
    t1.y = 20;
    cout << t1.y;</pre>
    return 0;
```

20

6. thread_local Storage Class

The thread_local Storage Class is the new storage class that was added in C++11. We can use the **thread_local** storage class specifier to define the object as thread_local. The thread_local variable can be combined with other storage specifiers like static or extern and the properties of the thread_local object changes accordingly.

Properties of thread_local Storage Class

- Memory Location: RAM
- Lifetime: Till the end of its thread

Example of thread_local Storage Class

```
// thread 2
thread th2([]() {
    cout << "Thread 2 var Value: " << (var += 7) << '\n';
});

// thread 3
thread th3([]() {
    cout << "Thread 3 var Value: " << (var += 13) << '\n';
});

th1.join();
th2.join();
th3.join();

return 0;
}</pre>
```

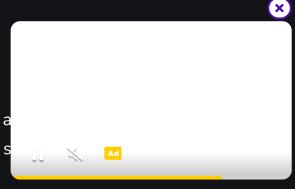
```
Thread 1 var Value: 28
Thread 2 var Value: 17
Thread 3 var Value: 23
```

As we can see, each thread got its own copy of the thread_local variable and was only assigned the value that was specified in its callable.

Your options after clearing GATE Examinations can be:

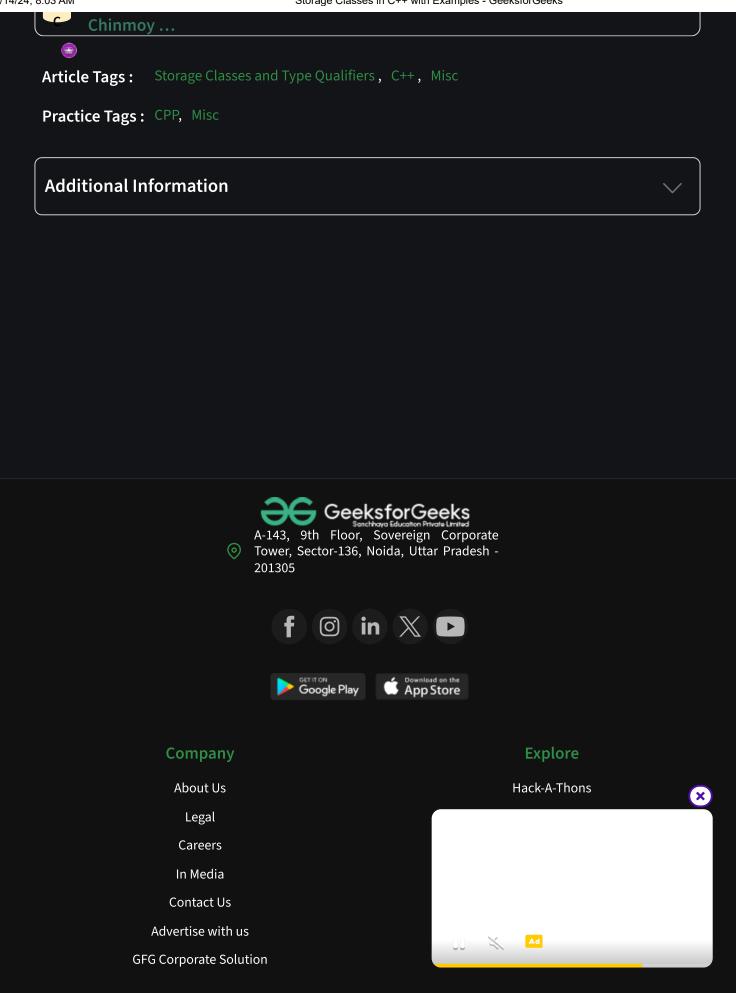
- 1. Go for higher studies in prestigious IITs
- 2. Apply for job roles in PSUs or MNCs
- 3. Specialize further in AI or Cybersecurity
- 4. Take up teaching roles
- 5. And much more!

Sounds like something you wish to learn more a <u>Counselling session</u> with our experts and they s direction.



Also get assured GfG T-Shirts if you attend the counselling. Register Now. NOTE: This service is exclusively for the students in Delhi/NCR Region as the classroom program for GATE is in our Noida Center only. We are soon coming up with more all across India!

Last Updated: 18 Aug, 2023 **n**凸 56 Д Previous Next Scope of Variables in C++ Static Keyword in C++ Add Your Comment **Similar Reads** C | Storage Classes and Type Qualifiers | Object Storage VS Block Storage in Cloud Question 1 C | Storage Classes and Type Qualifiers | C | Storage Classes and Type Qualifiers | Question 19 Question 3 C | Storage Classes and Type Qualifiers | C | Storage Classes and Type Qualifiers | Question 19 Question 19 C | Storage Classes and Type Qualifiers | C | Storage Classes and Type Qualifiers | Question 6 Question 7 C | Storage Classes and Type Qualifiers | C | Sto Question 8 Quest



Placement Training Program

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

Tutorials Archive

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

ML Maths

Data Visualisation Tutorial

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Python

Python Programming Examples

Python Projects

Python Tkinter

Web Scraping

OpenCV Python Tutorial

Python Interview Question

DevOps

Git

DSA

Data Structures

Algorithms

DSA for Beginners

Basic DSA Problems

DSA Roadmap

Top 100 DSA Interview Problems

DSA Roadmap by Sandeep Jain

All Cheat Sheets

HTML & CSS

HTML

CSS

Web Templates

CSS Frameworks

Bootstrap

Tailwind CSS

SASS

LESS

Web Design

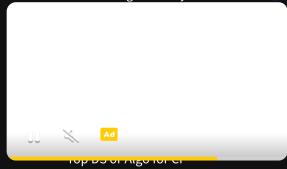
Django Tutorial

Computer Science

Operating Systems

Computer Network

Database Management System



 (\mathbf{x})

AWS Top 50 Tree Docker Top 50 Graph Kubernetes Top 50 Array Azure Top 50 String GCP Top 50 DP DevOps Roadmap Top 15 Websites for CP **System Design JavaScript** High Level Design JavaScript Examples Low Level Design TypeScript **UML Diagrams** ReactJS Interview Guide NextJS **Design Patterns AngularJS** OOAD NodeJS System Design Bootcamp Lodash **Interview Questions** Web Browser **School Subjects Preparation Corner** Company-Wise Recruitment Process Mathematics **Resume Templates Physics** Aptitude Preparation Chemistry Puzzles **Biology** Company-Wise Preparation **Social Science English Grammar** World GK **Free Online Tools Management & Finance Typing Test** Management (\mathbf{x}) **Image Editor HR Management Finance** Income Tax Organisational Behaviour Marketing

More Tutorials	GeeksforGeeks Videos			
Software Development	DSA			
Software Testing	Python			
Product Management	Java			
SAP	C++			
SEO - Search Engine Optimization	Data Science			
Linux	CS Subjects			
Excel				
@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved				



