Unit -1

C++ is a general-purpose programming language that was developed by Bjarne Stroustrup at Bell Labs in Murray Hill, New Jersey, in the early 1980s. Stroustrup initially started working on C with Classes, which was an extension of the C programming language. The goal was to add object-oriented programming features to the C language, which was widely used for system programming at the time.

C++ was officially released in 1985 as a separate language, although it maintained a high degree of compatibility with C. The name "C++" reflects its evolution from C; the "++" operator in C++ is used to increment the value of a variable.

One of the key features of C++ is its support for object-oriented programming (OOP). This means that it allows developers to define classes, which are blueprints for creating objects, and use inheritance, polymorphism, and encapsulation to organize and structure code in a more modular and reusable way.

Over the years, C++ has evolved and gained popularity due to its efficiency, flexibility, and the extensive standard library that comes with it. It is widely used in a variety of applications, including system programming, game development, software infrastructure, and more.

Procedural Programming vs Object-Oriented Programming:

Procedural Oriented Programming	**Object-Oriented Programming**
In procedural programming, the program is divided into small parts called functions.	In object-oriented programming, the program is divided into small parts called objects.
Procedural programming follows a top-down approach.	Object-oriented programming follows a bottom-up approach.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way of hiding data so it is less secure.	Object-oriented programming provides data hiding so it is more secure.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.

In procedural programming, the function is more important than the data.	In object-oriented programming, the concept of data hiding and inheritance is used.
Procedural programming is based on the unreal world.	Object-oriented programming is based on the real world.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Code reusability absent in procedural programming,	Code reusability present in object-oriented programming.
Examples: C, FORTRAN, Pascal, Basic, etc.	Examples: C++, Java, Python, C#, etc.

Introduction to OOPS Languages:

Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that
function.

Characteristics and Concepts of oops

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Message Passing

1. Class:

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and

brands but all of them will share some common properties like all of them will have 4 wheels,
Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are
their properties.

2. Object:

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code it is sufficient to know the type of message accepted and type of response returned the objects.
For example "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark,
Sleep, and Eats

3. Data Abstraction

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the
implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

4. Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.
Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there

may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He
will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

5. Inheritance

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

6. Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

7. Message Passing

It is a form of communication used in object-oriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Application of OOP’s

Below are some applications of OOPs:

- **Real-Time System design:** Real-time system inherits complexities and makes it difficult to build them. OOP techniques make it easier to handle those complexities.

- **Hypertext and Hypermedia:** Hypertext is similar to regular text as it can be stored, searched, and edited easily. Hypermedia on the other hand is a superset of hypertext. OOP also helps in laying the framework for hypertext and hypermedia.
- **AI Expert System:** These are computer application that is developed to solve complex problems which are far beyond the human brain. OOP helps to develop such an AI expert System
- **Office automation System:** These include formal as well as informal electronic systems that primarily concerned with information sharing and communication to and from people inside and outside the organization. OOP also help in making office automation principle.
- **Neural networking and parallel programming**: It addresses the problem of prediction and approximation of complex-time varying systems. OOP simplifies the entire process by simplifying the approximation and prediction ability of the network.
- **Stimulation and modeling system:** It is difficult to model complex systems due to varying specifications of variables. Stimulating complex systems require modeling and understanding interaction explicitly. OOP provides an appropriate approach for simplifying these complex models.
- **Object-oriented database:** The databases try to maintain a direct correspondence between the real world and database object in order to let the object retain it identity and integrity.
- **Client-server system:** Object-oriented client-server system provides the IT infrastructure creating object-oriented server internet (OCSI) applications.

Advantages and Disadvantages of OOP

OOP stands for Object-Oriented Programming. As you can guess from it's name it breaks the program on the basis of the objects in it. It mainly works on Class, Object, Polymorphism, Abstraction, Encapsulation and Inheritance. Its aim is to bind together the data and functions to operate on them.

Some of the well-known object-oriented languages are Objective C, Perl, Java, Python, Modula, Ada, Simula, C++, Smalltalk and some Common Lisp Object Standard. Here we are discussing its benefits on C++.

Benefits of OOPs

- We can build the programs from standard working modules that communicate with one another, rather than having to start writing the code from scratch which leads to saving of development time and higher productivity,

- OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).
- The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.
- OOP systems can be easily upgraded from small to large systems.
- It is possible that multiple instances of objects co-exist without any interference,
- It is very easy to partition the work in a project based on objects.
- It is possible to map the objects in problem domain to those in the program.
- The principle of data hiding helps the programmer to build secure programs which cannot be invaded by the code in other parts of the program.
- By using inheritance, we can eliminate redundant code and extend the use of existing classes.
- Message passing techniques is used for communication between objects which makes the interface descriptions with external systems much simpler.
- The data-centered design approach enables us to capture more details of model in an implementable form.

  While it is possible to incorporate all these features in an OOP, their importance depends upon the type of project and preference of the programmer. These technology is still developing and current products may be superseded quickly.

  Developing a software is easy to use makes it hard to build.

Disadvantages of OOPs

- The length of the programmes developed using OOP language is much larger than the procedural approach. Since the programme becomes larger in size, it requires more time to be executed that leads to slower execution of the programme.
- We can not apply OOP everywhere as it is not a universal language. It is applied only when it is required. It is not suitable for all types of problems.
- Programmers need to have brilliant designing skill and programming skill along with proper planning because using OOP is little bit tricky.
- OOPs take time to get used to it. The thought process involved in object-oriented programming may not be natural for some people.
- Everything is treated as object in OOP so before applying it we need to have excellent thinking in terms of objects.

Difference between C and C++

C	C++
C was developed by Dennis Ritchie between the year 1969 and 1973 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979.
C does no support polymorphism, encapsulation, and inheritance which means that C does not support object oriented programming.	C++ supports polymorphism, encapsulation, and inheritance because it is an object oriented programming language.
Data and functions are separated in C because it is a procedural programming language.	Data and functions are encapsulated together in form of an object in C++.
C does not support information hiding.	Data is hidden by the Encapsulation to ensure that data structures and operators are used as intended.
Function and operator overloading is not supported in C.	Function and operator overloading is supported by C++.
Functions in C are not defined inside structures.	Functions can be used inside a structure in C++.
Standard IO header is stdio.h.	Standard IO header is iostream.h.
C structures don't have access modifiers.	C ++ structures have access modifiers.
C follows the top-down approach	C++ follows the Bottom-up approach

Introduction to C++ Programming Language

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language.

- C++ is a high-level, general-purpose programming language designed for system and application programming. It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language. C++ is an object-oriented, multi-paradigm language that supports procedural, functional, and generic programming styles.
- One of the key features of C++ is its ability to support low-level, system-level programming, making it suitable for developing operating systems, device drivers, and other system software. At the same time, C++ also provides a rich set of libraries and features for high-level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.
- C++ has a large, active community of developers and users, and a wealth of resources and tools available for learning and using the language. Some of the key features of C++ include:
- **Object-Oriented Programming:** C++ supports object-oriented programming, allowing developers to create classes and objects and to define methods and properties for these objects.
- **Templates:** C++ templates allow developers to write generic code that can work with any data type, making it easier to write reusable and flexible code.
- **Standard Template Library (STL):** The STL provides a wide range of containers and algorithms for working with data, making it easier to write efficient and effective code.
- **Exception Handling:** C++ provides robust exception handling capabilities, making it easier to write code that can handle errors and unexpected situations.

Overall, C++ is a powerful and versatile programming language that is widely used for a range of applications and is well-suited for both low-level system programming and high-level application development.

- C++ is a middle-level language rendering it the advantage of programming low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

Some of the features & key-points to note about the programming language are as follows:

1. **Simple:** It is a simple language in the sense that programs can be broken down into logical units and parts, has a rich library support and a variety of data-types.
2. **Machine Independent but Platform Dependent:** A C++ executable is not platform-independent (compiled programs on Linux won't run on Windows), however they are machine independent.
3. **Mid-level language:** It is a mid-level language as we can do both systems-programming (drivers, kernels, networking etc.) and build large-scale user applications (Media Players, Photoshop, Game Engines etc.)
4. **Rich library support:** Has a rich library support (Both standard ~ built-in data structures, algorithms etc.) as well 3rd party libraries (e.g. Boost libraries) for fast and rapid development.
5. **Speed of execution:** C++ programs excel in execution speed. Since, it is a compiled language, and also hugely procedural. Newer languages have extra in-built default features such as garbage-collection, dynamic typing etc. which slow the execution of the program overall. Since there is no additional processing overhead like this in C++, it is blazing fast.
6. **Pointer and direct Memory-Access:** C++ provides pointer support which aids users to directly manipulate storage address. This helps in doing low-level programming (where one might need to have explicit control on the storage of variables).
7. **Object-Oriented:** One of the strongest points of the language which sets it apart from C. Object-Oriented support helps C++ to make maintainable and extensible programs. i.e. Large-scale applications can be built. Procedural code becomes difficult to maintain as code-size grows.
8. **Compiled Language:** C++ is a compiled language, contributing to its speed.

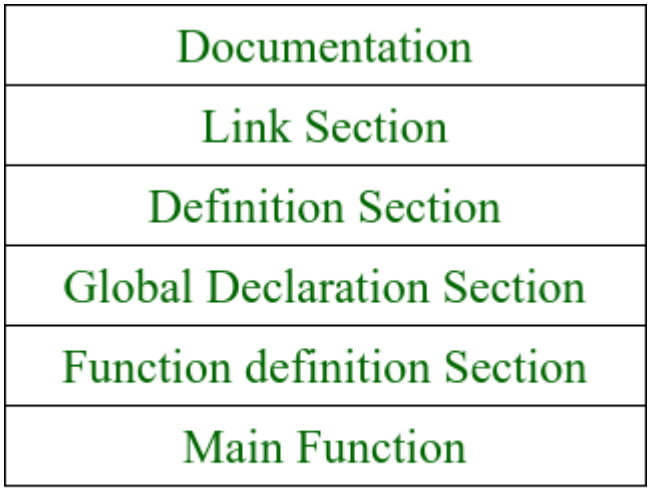
Applications of C++:

C++ finds varied usage in applications such as:

- Operating Systems & Systems Programming. e.g. Linux-based OS (Ubuntu etc.)
- Browsers (Chrome & Firefox)
- Graphics & Game engines (Photoshop, Blender, Unreal-Engine)
- Database Engines (MySQL, MongoDB, Redis etc.)
- Cloud/Distributed Systems

Structure of C++ Program:

The C++ program is written using a specific template structure. The structure of the program written in C++ language is as follows:



Skeleton of C Program

1. Documentation Section:

- This section comes first and is used to document the logic of the program that the programmer going to code.
- It can be also used to write for purpose of the program.
- Whatever written in the documentation section is the comment and is not compiled by the compiler.
- Documentation Section is optional since the program can execute without them. Below is the snippet of the same:

  Example:-

```
/*      This is a C++ program to find the
            factorial of a number

        The basic requirement for writing this
        program is to have knowledge of loops
```

To find the factorial of number
iterate over range from number to one
*/

2. Linking Section:

The linking section contains two parts:

1. Header Files:

Generally, a program includes various programming elements like built-in functions, classes, keywords, constants, operators, etc. that are already defined in the standard C++ library.
In order to use such pre-defined elements in a program, an appropriate header must be included in the program.
Standard headers are specified in a program through the preprocessor directive #include. In Figure, the iostream header is used. When the compiler processes the instruction #include<iostream>, it includes the contents of the stream in the program. This enables the programmer to use standard input, output, and error facilities that are provided only through the standard streams defined in <iostream>. These standard streams process data as a stream of characters, that is, data is read and displayed in a continuous flow. The standard streams defined in <iostream> are listed here.

2. Namespaces:

A namespace permits grouping of various entities like classes, objects, functions, and various C++ tokens, etc. under a single name.
Any user can create separate namespaces of its own and can use them in any other program.
In the below snippets, namespace std contains declarations for cout, cin, endl, etc. statements.

Syntax:

using namespace std;

3. Definition Section:

- It is used to declare some constants and assign them some value.
- In this section, anyone can define your own datatype using primitive data types.

In #define is a compiler directive which tells the compiler whenever the message is found to replace it with “Factorial\n”.
typedef int INTEGER; this statement tells the compiler that whenever you will encounter INTEGER replace it by int and as you have declared INTEGER as datatype you cannot use it as an identifier.

4. Global Declaration Section:

Here, the variables and the class definitions which are going to be used in the program are declared to make them global.
The scope of the variable declared in this section lasts until the entire program terminates.
These variables are accessible within the user-defined functions also.
Function Declaration Section:
It contains all the functions which our main functions need.
Usually, this section contains the User-defined functions.
This part of the program can be written after the main function but for this, write the function prototype in this section for the function which for you are going to write code after the main function.

5. Main Function:

The main function tells the compiler where to start the execution of the program. The execution of the program starts with the main function.
All the statements that are to be executed are written in the main function.
The compiler executes all the instructions which are written in the curly braces {} which encloses the body of the main function.
Once all instructions from the main function are executed, control comes out of the main function and the program terminates and no further execution occur.

Storage Classes in C++:

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

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

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 can be accessed between two different files which are part of a large program.

Properties of extern Storage Class Objects

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

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 re-declared. 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

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

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

When we declare a function as const, this pointer passed to the function becomes const. Adding a mutable to a variable allows a const pointer to 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.

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

Reference Variable in C++:

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting '&' in the declaration.

Also, we can define a reference variable as a type of variable that can act as a reference to another variable. '&' is used for signifying the address of a variable or any memory. Variables associated with reference variables can be accessed either by its name or by the reference variable associated with it.
Syntax:

data_type &ref = variable;

Applications of Reference in C++:

There are multiple applications for references in C++, a few of them are mentioned below:

1. Modify the passed parameters in a function
2. Avoiding a copy of large structures
3. In For Each Loop to modify all objects
4. For Each Loop to avoid the copy of objects

1. Modify the passed parameters in a function:

If a function receives a reference to a variable, it can modify the value of the variable. For example, the following program variables are swapped using references.

2. Avoiding a copy of large structures:

Imagine a function that has to receive a large object. If we pass it without reference, a new copy of it is created which causes a waste of CPU time and memory. We can use references to avoid this.

3. In For Each Loop to modify all objects:

We can use references for each loop to modify all elements.

```
// C++ Program for changing the
// values of elements while traversing
```

```
// using references
#include <iostream>
#include <vector>

using namespace std;

// Driver code
int main()
{
	vector<int> vect{ 10, 20, 30, 40 };

	// We can modify elements if we
	// use reference
	for (int& x : vect) {
		x = x + 5;
	}

	// Printing elements
	for (int x : vect) {
		cout << x << " ";
	}
	cout << '\n';
	return 0;
}
```

Output

15 5 35 45

4. For Each Loop to avoid the copy of objects:

We can use references in each loop to avoid a copy of individual objects when objects are large.

```
// C++ Program to use references
// For Each Loop to avoid the
// copy of objects
#include <iostream>
#include <vector>

using namespace std;
```

```
// Driver code
int main()
{
	// Declaring vector
	vector<string> vect{ "geeksforgeeks practice",
	                                        "geeksforgeeks write",
	                                        "geeksforgeeks ide" };

	// We avoid copy of the whole string
	// object by using reference.
	for (const auto& x : vect) {
		cout << x << '\n';
	}

	return 0;
}
```

Output

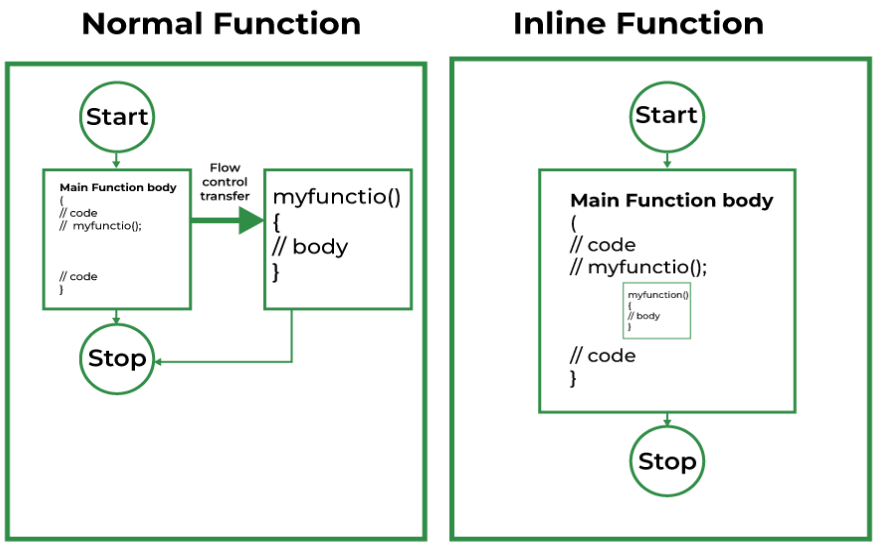
```
geeksforgeeks practice
geeksforgeeks write
geeksforgeeks ide
```

Inline Functions in C++:

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

```
inline return-type function-name(parameters)
{
  // function code
}
```



Inline functions Advantages:

Function call overhead doesn't occur.

It also saves the overhead of push/pop variables on the stack when a function is called. It also saves the overhead of a return call from a function.

When you inline a function, you may enable the compiler to perform context-specific optimization on the body of the function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of the calling context and the called context.

An inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function called preamble and return.

Inline function Disadvantages:

The added variables from the inlined function consume additional registers, After the in-lining function if the variable number which is going to use the register increases then they may create overhead on register variable resource utilization. This means that when the inline function body is substituted at the point of the function call, the total number of variables used by the function also gets inserted. So the number of registers going to be used for the variables will also get increased. So if after function inlining variable numbers increase drastically then it would surely cause overhead on register utilization.

If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of the same code.

Too much inlining can also reduce your instruction cache hit rate, thus reducing the speed of instruction fetch from that of cache memory to that of primary memory.

The inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled because the compiler would be required to replace all the code once again to reflect the changes, otherwise it will continue with old functionality.

Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.

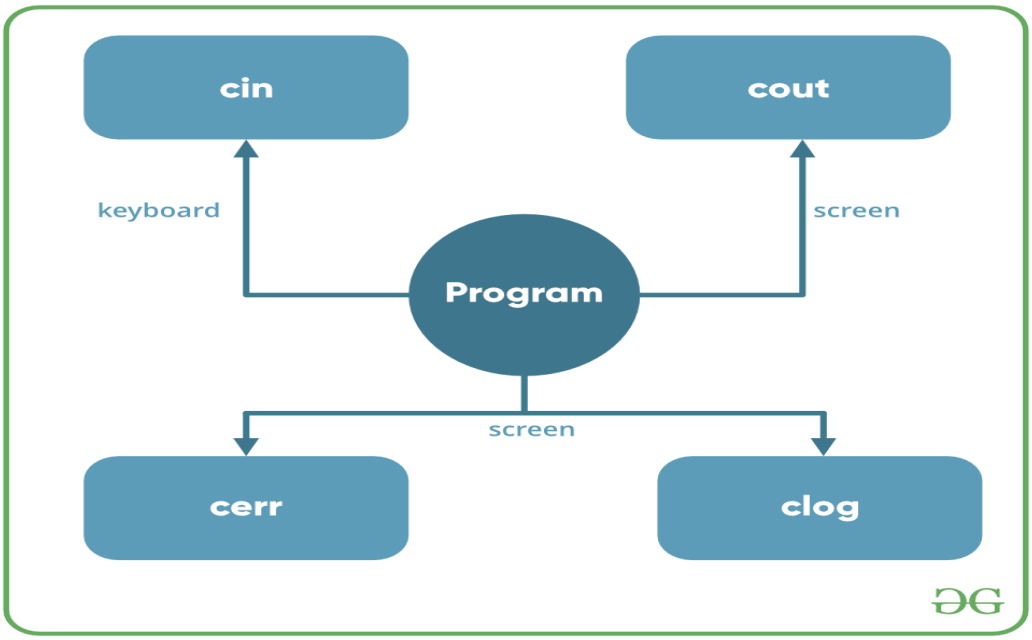
Inline functions might cause thrashing because inlining might increase the size of the binary executable file. Thrashing in memory causes the performance of the computer to degrade. The following program demonstrates the use of the inline function.

Basic Input / Output in C++:

C++ comes with libraries that provide us with many ways for performing input and output. In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams.

Input Stream: If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.

Output Stream: If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.



cin in C++:

The cin object in C++ is an object of class iostream. It is used to accept the input from the standard input device i.e. keyboard. It is associated with the standard C input stream stdin. The extraction operator(>>) is used along with the object cin for reading inputs. The extraction operator extracts the data from the object cin which is entered using the keyboard.

```
// C++ program to demonstrate the

// cin object

#include <iostream>
```

```
using namespace std;


// Driver Code
int main()
{
string s;
// Take input using cin
        cin >> s;
// Print output
cout << s;
return 0;
}
```

Input:

Practice

Output:

Practice

cout in C++:

The cout object in C++ is an object of class iostream. It is defined in iostream header file. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

```
// C++ program to illustrate the use

// of cout object

#include <iostream>

using namespace std;


// Driver Code

int main()

{

	// Print standard output

	// on the screen

	cout << "Welcome to GFG";


	return 0;

}
```

Output:

```
Welcome to GFG
```

Scope resolution operator in C++:

In C++, the scope resolution operator, ::, is used to specify the scope to which a particular identifier belongs. It allows you to access members (variables, functions, or types) defined in a class, namespace, or global scope.To define a function outside a class.

Here are a few common uses of the scope resolution operator:

1. Accessing class members: You can use :: to access static members of a class or to qualify members that have the same name as other entities in a different scope.

```
class MyClass {
public:
   static int myStaticVar;
};

int MyClass::myStaticVar = 42;

int main() {
   int myStaticVar = 10;
   std::cout << "Local variable: " << myStaticVar << std::endl;
   std::cout << "Class static variable: " << MyClass::myStaticVar << std::endl;
   return 0;
}
```

2. **Accessing global variables:** You can use :: to access global variables inside a function or block that has a local variable with the same name.

```
int x = 10;

int main() {
   int x = 20;
   std::cout << "Local variable x: " << x << std::endl;
```

```
    std::cout << "Global variable x: " << ::x << std::endl;
    return 0;
}
```

3. **Accessing nested namespaces:** You can use :: to access entities in nested namespaces.

```
namespace Outer {
  int x = 5;
namespace Inner {
    int x = 10;
  }
}
int main() {
  std::cout << "Outer x: " << Outer::x << std::endl;
  std::cout << "Inner x: " << Outer::Inner::x << std::endl;
  return 0;
}
```

Member Dereferencing Operator:

In C++, the member dereferencing operator -> is used to access members of a class or structure through a pointer to that class or structure. It is often used when you have a pointer to an object and need to access its members.

```
#include <iostream>

using namespace std;

class MyClass {

public:

   int myVar;

   void myMethod() {

      cout << "Hello from myMethod!" << endl;

   }

};


int main() {

   MyClass obj;

   obj.myVar = 42;

 MyClass* ptr = &obj;

   cout << "Using pointer: " << ptr->myVar << endl;

   ptr->myMethod();

return 0;

}
```

In this example, ptr->myVar accesses the myVar member of the MyClass object pointed to by ptr, and ptr->myMethod() calls the myMethod member function of the same object.

The member dereferencing operator -> is essentially a shorthand for (*ptr)., making it more convenient to work with pointers to objects.

Default Arguments in C++:

In C++, default arguments allow you to specify default values for function parameters. When a function is called without providing a value for a parameter that has a default argument, the default value is used instead. Default arguments are specified in the function declaration or definition.

```
#include <iostream>

using namespace std;


// Function with default arguments

void greet(string name = "Guest") {

   cout << "Hello, " << name << "!" << endl;

}


int main() {

   greet();         // Outputs: Hello, Guest!

   greet("Alice");   // Outputs: Hello, Alice!


   return 0;

}
```

In this example, the greet function has a default argument of "Guest" for the name parameter. When greet is called without any arguments, it uses the default argument and prints "Hello, Guest!". When called with an argument, such as "Alice", it uses the provided argument and prints "Hello, Alice!".

Default arguments are specified in the function declaration or definition, typically in the form type name = default_value. It's important to note that once you provide a default argument

for a parameter, all subsequent parameters must also have default arguments. Default arguments are resolved at compile time based on the function call, so they cannot depend on runtime values.