1. 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.
2. 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. low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.).

***features & key-points***

**Simple:** programs can be broken down into logical units and parts,

**Machine Independent but Platform Dependent**

* **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.)

**Rich library support**: rich library -STL 3rd party libraries (e.g. Boost libraries)

**Speed of execution**:as it is compiled(in-built default features such as garbage-collection, dynamic typing )

**Pointer and direct Memory-Access**: directly manipulate storage address, helps in doing low-level programming

Cross-platform Compatibility: C++ can be compiled and run on multiple platforms, including Windows, MacOS,

* Features: Object-Oriented Programming
* Machine Independent
* Simple
* High-Level Language- human-comprehensible English language
* Popular
* Case-sensitive
* Compiler Based
* Dynamic Memory Allocation
* Memory Management- In other programming languages such as [Java](https://www.geeksforgeeks.org/java/) and [Python](https://www.geeksforgeeks.org/python-programming-language/), the compiler automatically manages the memories allocated to variables not in c++, memory must be de-allocated dynamically allocated memory manually after it is of no use.
* Multi-threading-> [Multithreading](https://www.geeksforgeeks.org/multithreading-c-2/) is a specialized form of multitasking and multitasking is a feature that allows your system to execute two or more programs concurrently

A multithreaded program contains two or more parts that will run concurrently. Each part of such a program is named a thread, and every thread defines a separate path of execution.

Similarities bw c and cpp

Syntax,structure,completion of code, c++ extended grammer,same basic memeory model, Same notions

C keywords-59,cpp keyw-81

Procedural vs proced+oops

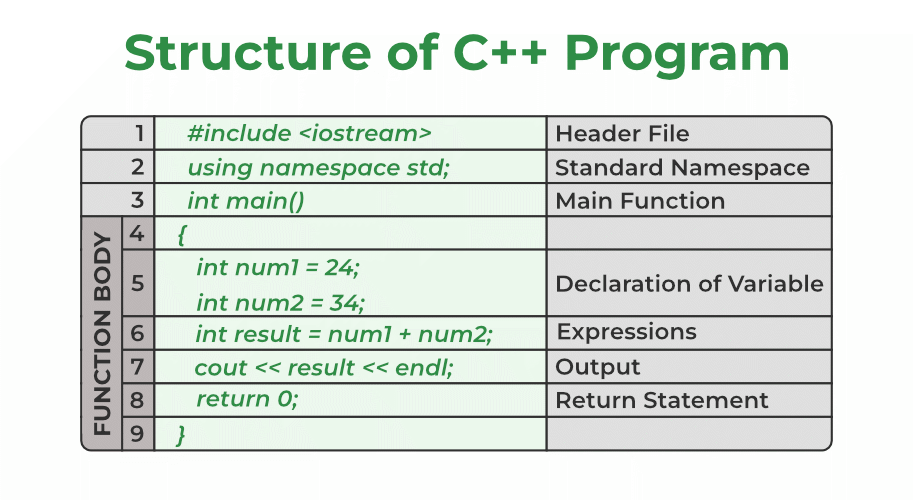
c-function driven

c vs.cpp,.c++,.cc,.cxx

cpp basics:

#include <iostream>, **using** **namespace** std;

//single,/\*\*/ multiline



Types of Tokens in C++

We have several types of tokens each of which serves a specific purpose in the syntax and semantics of C++. Below are the main types of tokens in C++:

1. Identifiers

2. Keywords

Constants- const data\_type variable\_name = value;

3.

4. Strings

5. Special Symbols  **Semicolon (;):** It is used to terminate the statement.

 **Square brackets []:** Theyare used to store array elements.

 **Curly Braces {}:**They are used to define blocks of code.

 **Scope resolution (::):**Scope resolution operator is used to access members of namespaces, classes, etc.

 **Dot (.):**Dot operator also called member access operator used to access class and struct members.

 **Assignment operator ‘=’:**This operator is used to assign values to variables.

 **Double-quote (“):** It is used to enclose string literals.

1.  **Single-quote (‘):** It is used to enclose character literals.
2. Operators
3. Unary Operators-> --,++
4. Binary Operators normal
5. Ternary Operators, -> ? : ;

**Local Variables**

**Instance Variables**

**Static Variables**

**How to Define Constants in C++?**

We can define the constants in C++ using three ways:

**Using const Keyword--** **const** DATATYPE *variable\_name* =value*;*

**Using constexpr Keyword---** **constexpr** DATATYPE variable\_name = value ;

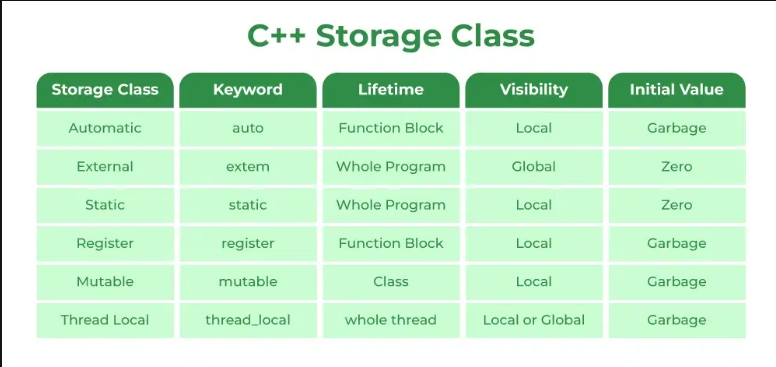
**Using #define Preprocessor--** **#define** MACRO\_NAME replacement\_value

// using #define to create a macro

#define Side 5

Scope of varible

1. Local Variables
2. Global Variables



**Properties of auto Storage Class Objects**

* **Scope:** Local
* **Default Value:**Garbage Value
* **Memory Location:** RAM
* **Lifetime:**Till the end of its scope
* **Properties of extern Storage Class Objects-extern storage class** simply tells us that the variable is defined elsewhere and not within the same block where it is used
* **Scope:** Global
* **Default Value:**Zero
* **Memory Location:** RAM
* **Lifetime:**Till the end of the program.

### static Storage Class

* **Scope:** Local
* **Default Value:**Zero
* **Memory Location:** RAM
* **Lifetime:**Till the end of the program

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

‘**register’ keyword** same functionality as that of the auto variables. compiler tries to store these variables in the register of the microprocessor

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.

#include <iostream>

#include <thread>

**using** **namespace** std;

// defining thread local variable

thread\_local **int** var = 10;

// driver code

**int** main()

{

    // thread 1

**thread** th1([]() {

        cout << "Thread 1 var Value: " << (var += 18) << '\n';

    });

    // 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;

}



Int,char,bool,float,double,void,wchar\_t,sizeof()

modifiers available in C++ are:

* **Signed**
* **Unsigned**
* **Short**
* **Long**

**Macro Constants**

CHAR\_MIN

CHAR\_MAX

SCHAR\_MIN

SCHAR\_MAX

UCHAR\_MAX

CHAR\_BIT

MB\_LEN\_MAX

SHRT\_MIN

SHRT\_MAX

USHRT\_MAX

INT\_MIN

INT\_MAX

UINT\_MAX

LONG\_MIN

LONG\_MAX

ULONG\_MAX

LLONG\_MIN

LLONG\_MAX

ULLONG\_MAX

**Common and Unique Functions in C++ STL Sequential Containers**

C++ Standard Template Library (STL) provides several sequential containers like vector, deque, list, array, and forward\_list. These containers share some common member functions, while others are unique to specific containers.

**1. Common Functions Across All Sequential Containers**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| begin() | Returns an iterator to the first element. | auto it = container.begin(); |
| end() | Returns an iterator to the past-the-end element. | auto it = container.end(); |
| cbegin() | Returns a const iterator to the first element. | auto it = container.cbegin(); |
| cend() | Returns a const iterator to the past-the-end element. | auto it = container.cend(); |
| rbegin() | Returns a reverse iterator to the first element. | auto rit = container.rbegin(); |
| rend() | Returns a reverse iterator to the past-the-end element. | auto rit = container.rend(); |
| crbegin() | Returns a const reverse iterator to the first element. | auto rit = container.crbegin(); |
| crend() | Returns a const reverse iterator to the past-the-end element. | auto rit = container.crend(); |
| empty() | Checks if the container is empty. | bool isEmpty = container.empty(); |
| size() | Returns the number of elements. | size\_t s = container.size(); |
| max\_size() | Returns the maximum possible number of elements. | size\_t max\_s = container.max\_size(); |
| front() | Accesses the first element. | auto elem = container.front(); |
| back() | Accesses the last element. | auto elem = container.back(); |
| assign() | Assigns new values to the container. | container.assign(first, last); |
| insert() | Inserts elements. | container.insert(pos, value); |
| erase() | Erases elements. | container.erase(pos); |
| clear() | Clears all elements. | container.clear(); |
| swap() | Swaps contents with another container of the same type. | container.swap(other); |
| emplace() | Constructs and inserts element in place. | container.emplace(pos, args...); |
| emplace\_back() | Constructs and inserts element at the end. | container.emplace\_back(args...); |
| push\_back() | Adds an element at the end. | container.push\_back(value); |
| pop\_back() | Removes the last element. | container.pop\_back(); |

**2. Unique Functions for Each Sequential Container**

|  |  |  |
| --- | --- | --- |
| **Container** | **Unique Functions** | **Syntax** |
| **vector** | reserve(), capacity(), shrink\_to\_fit() | container.reserve(n); size\_t cap = container.capacity(); container.shrink\_to\_fit(); |
| **deque** | push\_front(), pop\_front() | container.push\_front(value); container.pop\_front(); |
| **list** | push\_front(), pop\_front(), merge(), splice(), remove(), remove\_if(), unique(), reverse(), sort() | container.push\_front(value); container.pop\_front(); container.merge(other); container.splice(pos, other); container.remove(value); container.remove\_if(pred); container.unique(); container.reverse(); container.sort(); |
| **forward\_list** | push\_front(), pop\_front(), merge(), splice\_after(), remove(), remove\_if(), unique(), reverse(), sort(), before\_begin(), cbefore\_begin() | container.push\_front(value); container.pop\_front(); container.merge(other); container.splice\_after(pos, other); container.remove(value); container.remove\_if(pred); container.unique(); container.reverse(); container.sort(); auto it = container.before\_begin(); auto it = container.cbefore\_begin(); |
| **array** | fill(), data() | container.fill(value); auto ptr = container.data(); |

**Explanation of Unique Functions**

* **vector**:
  + **reserve()**: Reserves storage for at least n elements.
  + **capacity()**: Returns the size of the storage space currently allocated for the vector, expressed in terms of elements.
  + **shrink\_to\_fit()**: Requests the container to reduce its capacity to fit its size.
* **deque**:
  + **push\_front()**: Adds an element to the front of the deque.
  + **pop\_front()**: Removes the first element.
* **list**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| * **Feature** | **C++** | **Java** | **Python** | **PHP** | **JavaScript** |
| **Type** | Compiled | Compiled (JVM) | Interpreted | Interpreted | Interpreted |
| **Syntax** | Complex, manual memory management | Simpler than C++, managed memory | Very simple, dynamic typing | Simple, dynamic typing | Simple, dynamic typing |
| **Performance** | High, close to hardware | High, but slower than C++ | Moderate, slower than C++ & Java | Moderate, generally used for web | Moderate, fast in V8 engine |
| **Memory Management** | Manual | Automatic (Garbage Collection) | Automatic (Garbage Collection) | Automatic (Garbage Collection) | Automatic (Garbage Collection) |
| **Concurrency** | Threads with manual control | Built-in concurrency with threads | GIL restricts true parallelism | Limited, but extensions available | Event-driven, asynchronous |
| **Platform Dependency** | Platform-specific (requires recompile) | Platform-independent (JVM) | Platform-independent | Platform-independent | Platform-independent |
| **Use Cases** | System software, games, drivers | Enterprise apps, Android dev | Web, scripting, data science, AI | Web development | Web development, front-end scripting |
| **Library Support** | Extensive, low-level libraries | Extensive, high-level libraries | Extensive, high-level libraries | Extensive, web-related libraries | Extensive, web-related libraries |
| **OOP Support** | Full, multi-paradigm | Full, strictly OOP | Full, multi-paradigm | Partial, multi-paradigm | Full, prototype-based OOP |
| **Error Handling** | Exception handling | Robust exception handling | Exception handling | Exception handling | Exception handling |
| **Community Support** | Large, mature | Large, mature | Large, rapidly growing | Large, web-focused | Large, web-focused |
| **Popularity** | High among system programmers | High in enterprise and Android dev | High in general-purpose scripting | High in web development | High in web development |
| **Learning Curve** | Steep due to complexity | Moderate | Gentle | Gentle | Gentle |

* + **merge()**: Merges two sorted lists into one.
  + **splice()**: Transfers elements from one list to another at a specified position.
  + **remove()**: Removes all elements equal to a specified value.
  + **remove\_if()**: Removes elements that satisfy a specified condition.
  + **unique()**: Removes consecutive duplicate elements.
  + **reverse()**: Reverses the order of elements.
  + **sort()**: Sorts the elements.
* **forward\_list**:
  + **splice\_after()**: Transfers elements from one forward\_list to another after a specified position.
  + **before\_begin()**: Returns an iterator to the element before the first element (used for insertion at the beginning).
  + **cbefore\_begin()**: Returns a const iterator to the element before the first element.
* Object-Oriented Programming
* Machine Independent
* Simple
* High-Level Language
* Popular
* Case-sensitive
* Compiler Based
* Dynamic Memory Allocation
* Memory Management
* Multi-threading

1. [Simple](https://www.javatpoint.com/features-of-java#Simple)
2. [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
3. [Portable](https://www.javatpoint.com/features-of-java#Portable)
4. [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
5. [Secured](https://www.javatpoint.com/features-of-java#Secured)
6. [Robust](https://www.javatpoint.com/features-of-java#Robust)
7. [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)
8. [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)
9. [High Performance](https://www.javatpoint.com/features-of-java#High-Performance)
10. [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)
11. [Distributed](https://www.javatpoint.com/features-of-java#Distributed)
12. [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)

 **Simple and Easy to Learn**

 **Interpreted Language**

 **Dynamically Typed**

 **High-Level Language**

 **Extensive Standard Library**

 **Cross-Platform Compatibility**

 **Supports Multiple Programming Paradigms**

 **Automatic Memory Management**

 **Large Community and Ecosystem**

 **Highly Readable Code**

 **Robust Integration Capabilities**

* **array**:
  + **fill()**: Assigns the given value to all elements.
  + **data()**: Returns a pointer to the first element.

// Single line comment

 /\*

  Multiline Comment

  .

  .

  .

 \*/

* ***Shortcut for single line:****The shortcut to make a single line into a comment is****‘ctrl + /’.***

***(‘ctrl + /’)****.*

**here are 3 aspects of defining a variable:**

1. Variable Declaration
2. Variable Definition
3. Variable Initialization



**C Variable Types**

The C variables can be classified into the following types:

1. **Local Variables**
2. **Global Variables**
3. **Static Variables**
4. **Automatic Variables**
5. **Extern Variables**
6. **Register Variables**

