CPP101

Coding Club, IIT Guwahati

Coding Club
IIT Guwahati

```cpp
#include <iostream>
#include <functional>

int main()
{
    int C;
    std :: cin >> C;

    std::function<void(int)> Advanced = [&](int C) {
        C++;
    };

    Advanced(C++);

    return 0;
}
```

Week 1

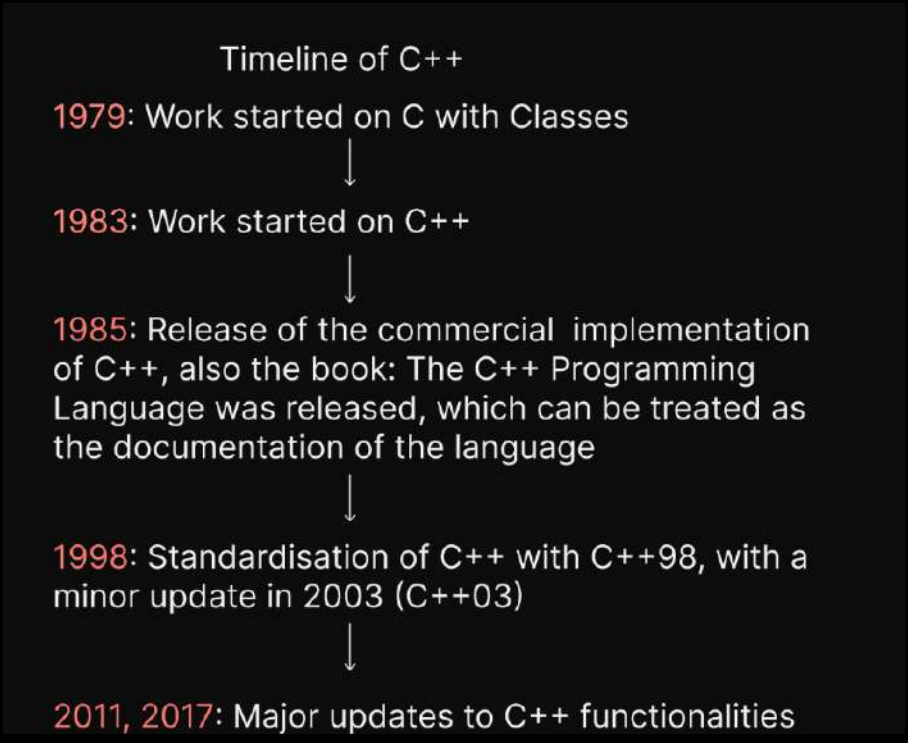
Day 1

What is C++?

C++ is a general purpose programming language with a slight bias towards systems programming developed by Danish computer scientist Bjarne Stroustrup as an extension to the C programming language.

History and Need of C++

In 1979, when the initial work on C++ began, large software development was quite a difficult task and there was no particular language which supported large software development, some were quite slow (for example, Simula67) or some were too low level of a language (for example, C and BCPL). This prompted Stroustrup to create a new language which combined the functionalities of Simula but maintained the performance and the freedom to manipulate memory at low level offered by C and BCPL.



The language was first known as ‘C with Classes’ but in 1983, was renamed as ‘C++’ implying one more than C, or signifying the evolutionary changes from C.

You can read about the different versions of C++ from here.

IDE’s for C++

Integrated development environment (IDE) is a software which helps one to write programs easily by providing the necessary additional software required to run the program in the IDE itself. An IDE normally consists of at least a source code editor, build automation tools and a debugger.
IDEs made specifically for C++:
You can read:
(1) Article 1
(2) Article 2

Alternatives to C++

There are many languages which are considered as an extension or alternative to C++, for example, Rust, Go and more recently Carbon. Actually, Rust and C++ are quite similar syntactically, but Rust has not been fully ‘developed’ as it’s documentation and frameworks aren’t on the same level as C++, even though Rust provides better speed and memory management than C++, something C++ is also known for. In the end, if the community backs Rust, it might well replace C++ in the near future. Go was a language developed to combine the simple syntax of Python with the speed of C++, interestingly, it was motivated by the shared dislike towards C++ of the designing team. Even though Go has a better efficiency than Python, it is somewhat slower than C++ and has memory overflow issues. Carbon by Google is seen as a successor to C++ and not a replacement . Carbon attempts to overcome problems in C++ by starting over with solid language foundations such as modern generics, a simple syntax, and modular code organisation. It is to be noted that Carbon has currently no compiler of it’s own so it can’t be used by developers as of now, but it can become a super set of C++ just like C++ became C’s super set in the future.

You can read the following articles to read about the potential alternatives to C and C++:

1. Article 1
2. Article 2
3. Discussion

Preprocessor Directives in C++

What's the preprocessor? Well, its name reflects that it processes the source code before the "main" stages of compilation. It's simply there to process the textual source code, modifying it in various ways. The preprocessor doesn't even understand the tokens it operates on - it has no notion of types or variables, classes or functions - it's all just quoted- and/or parentheses- grouped, comma- and/or whitespace separated text to be manhandled. This extra process gives more flexibility in selecting, combining and even generating parts of the program.
Some types of Preprocessors:

- Macros: A macro is a label defined in the source code that is replaced by its value by the preprocessor before compilation. Macros are initialized with the #define preprocessor command and can be undefined with the #undef command.
  You can read about macros from this article or this article.

- Conditional Inclusions: By using conditional inclusion statements different code can be included in the program depending on the situation at the time of compilation. Conditional inclusion statements are used to prevent multiple inclusion of a header file. The C++ preprocessor supports conditional compilation of parts of source file. C++ preprocessor directives control conditional inclusion. C programming language provides #if, #else, #elif, #ifdef, #ifndef, and #endif conditional preprocessor directives.
  You can read more about conditional inclusions from this article

- Source-file inclusions (#include): Tells the preprocessor to include the contents of a specified file at the point where the directive appears.
  You can read more about #include from this article or this article .

- Pragma Directive (#pragma): The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.
  You can read more about #pragma from this article or this article .

- Typedef: It is used as an alias to some typename in our cpp file.
  You can read more about typedef from this article or this article.

You can read more on this from this article or this article.

Namespace in C++

A namespace is a declarative region that provides a scope to the identifiers inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

You can have a look at this video to learn more about namespaces here

Also you can learn here about the benefits and disadvantages of using 'using namespace std; ' here

The Main function in C++

The main function is the driver function in the C++. The main function is a special function. Every C++ program must contain a function named main. It serves as the entry point for the program. The computer will start running the code from the beginning of the main function.

While doing competitive programming many people tend to define int as long long in the preprocessor statement which may lead to problem because then the main function also will have return type long long that's why some people use signed main instead of int main. This is because main must always return an int. The main function can return any value also (not necessarily 0), returning a value is just a signal (status code to be more formal) to the OS that the program ran successfully, but to ensure maximum portability of your program either return 0 or EXIT_SUCCESS in the main function.

```
#include <iostream>
#define int long long
using namespace std;

signed main(){
    //Code here
    return 0;
}
```

Day 2

Headers in C++

Headers are nothing but pre-written libraries in C++, which houses useful functions which one can reuse easily without worrying about the efficiency and the code to that function (since they are written by some of the best programmers out there). These files are included by writing `#include` which actually signals the compiler to preprocess these files before compiling the actual file. Due to this, all statements starting with a `#` are known as preprocessor directives
One can learn more about header files from:

1. Video
2. Article 1
3. Article 2

List of all C++ headers: Article

Compilers and their importance

It is due to a compiler that our C++ code gets converted into object files.
You can learn about compilers and how they work from:

1. Video
2. Article

A list of some commonly used compilers: Article (PS: Most Linux/Windows users use GCC Compiler while Mac users are advised to use clang as xcode is compatible with it. Mac users should use gcc knowing that data allocation type of error can occur anytime during their work.)

Linker

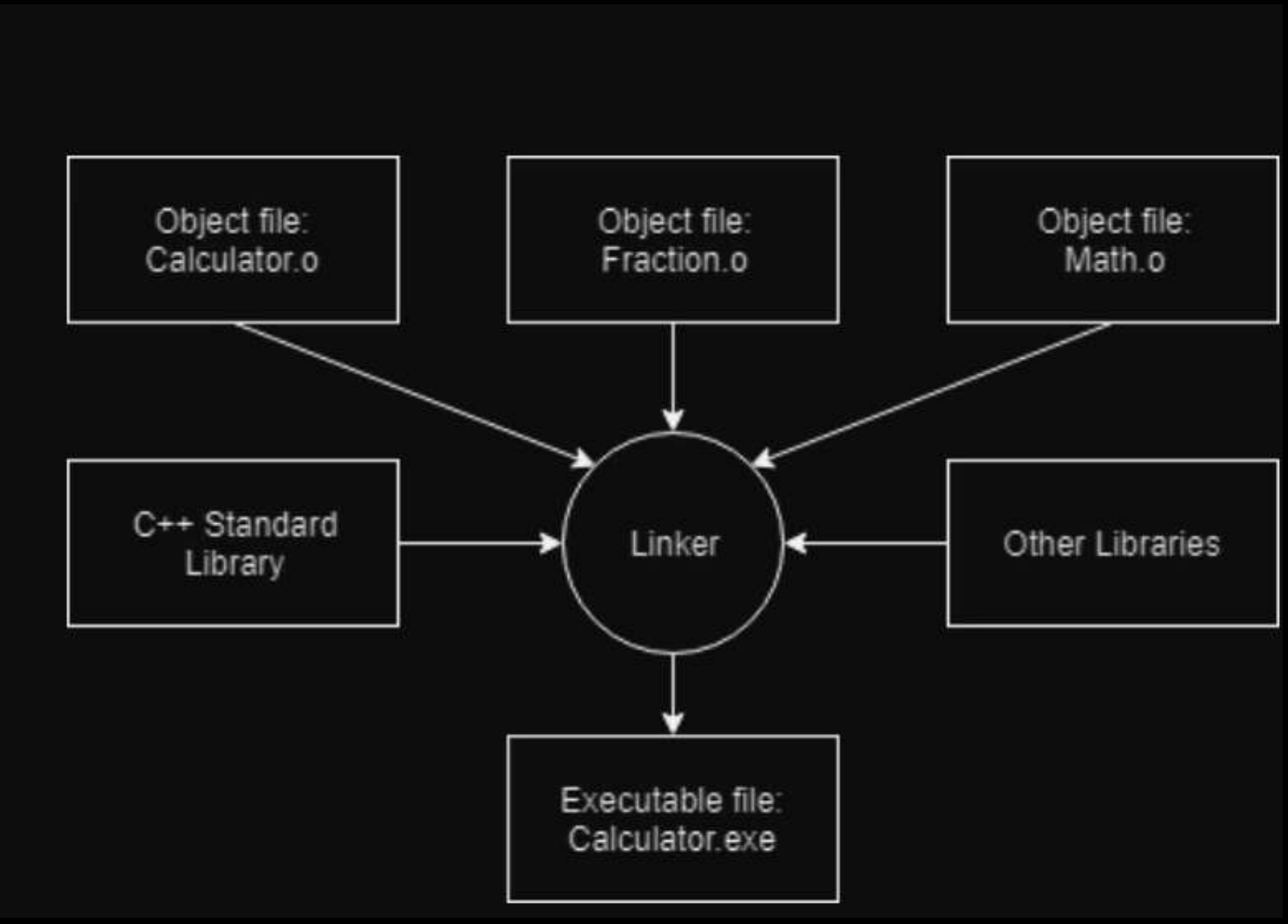
After the compiler creates one or more object files, then another program called the linker kicks in. The linker has two jobs to perform:

1. To take all the object files made by the compiler and the library files and make a single executable program from that.

2 . To make sure all cross-file dependencies are resolved properly. To understand what are cross-file dependencies, you can consider this example, if you define something in one .cpp file, and then use it in another .cpp file, the linker connects the two together. If the linker is unable to connect a reference to something with its definition, you'll get a linker error, and the linking process will abort.

You can learn more about linkers and how they work from:

1. Video
2. Article 1
3. Article 2
4. Discussion



Pairs and Tuples in C++

Pair is used to bind two different values in one object(which may not have same data type). Learn more about pairs here whereas tuples is used to bind three different values together. Learn more about it here

In order to initialise a pair we can use make_pair() function whereas to initialise a tuple we can use make_tuple function. Learn more about this functions here and here

The values of a pair can be accessed in the following way ->
pair.first and pair.second
But we cannot access the values of a tuple using tuple.first, tuple.second etc. We need to use the get() function. Learn more about it here

Tie function is also used to make a tuple object, but it is more generally used to unpack the elements of a tuple. Learn about it here and here

In C++17 and later, the language has added several new features that make working with pairs and tuples even more convenient. Some of these features include:

1. Structured binding: C++17 introduced a new syntax for unpacking the elements of a tuple or pair into separate variables. For example, instead of using 'std::get<0>()' to access the first element of a tuple, you can use the following syntax:

```
auto [x, y, z] = my_tuple;
auto [a, b] = my_pair;
```

2. Template parameter pack: C++17 introduced a new syntax for handling variable-length lists of template parameters. This can be used to create a tuple with a variable number of elements. For example:

```
template <typename... Types>
auto maketuple(Types&&... args){
    return std::tuple<Types...>(std::forward<Types>(args)...);
}
int main(){
  auto myTuple=maketuple(1,2,3,4,5);
}
```

Day 3

New Stuff added to C++

C++ is mostly a superset of C but with some added features like:

Namespace feature was added in C++

OOPS

The main idea behind developing C++ was to make it object oriented. The focus is on objects and manipulations around these objects. This makes it much easier to manipulate code, unlike procedural or structured programming which requires a series of computational steps to be carried out. That is why C++ finds usage in many things like:

(1) Game Development
(2) Database Management Software
(3) Stimulation and modelling system
and many more which you can find here.

More content on OOPS in C++ is in week 2

Exception / Error Handling

Another advantage of C++ over C is that it supports Exception Handling. Exception Handling allows us to tackle code which might get error on use, it allows us to catch the error and give the user appropriate response (or execute different code) if error occurs. If you are familiar with any other programming language that supports Exception Handling, you might heard of the keywords try, catch and throw. C++ also uses try, catch and throw keywords for Exception Handling.
Those of you who aren't familiar

=> try encloses the block of code that we want to ideally run (might contain error)
=> catch encloses the block of code executed if an error occurs
=> throw keyword allows the programmer to manually throw error in the try code if a specific scenario or error occurs.

More content on Exception Handling is in week 3.

Templating

One of the most useful feature of C++, which can help your code to be short and concise. If you want to make same functions, but with different data types, then a normal approach to this could be to simply make a function for each data type by copy pasting the original function and making some tweaks. However, this process could be tedious because there may be many functions and many data types as-well, error-prone and simply inefficient as you are repeating yourself. A smart approach would be to use templating, which allows us to define functions that are consistent with every data-type (or most of the data-type) and we can mention the data-type whenever using the function. You can read more about Templating here or watch a video here.

Variadic Function Template : This method allows templates to take variable number of arguments which can't be done in the normal method. It uses template parameter pack or ellipsis (`...`) to accept any number of arguments of any type. Learn more about Variadic Function Template here. You can learn more about Template parameter pack here. The following is an implementation of Variadic Function Template.

```
template <typename... Ts>
void print(Ts... args) {
    (std::cout << ... << args) << std::endl;
}
int main(){
  print(1,2,3,4);
}
```

A more extensive Standard Library

- The standard library of C++ incorporates most of the standard C library headers. To know more about usage of C library in C++, read this article.
- The standard library of C++ was developed using the Standard Template Library (STL), and shares a lot of feature, neither being the superset of other. You will learn more about STL in the further sections. STL and C++ standard library are different.
- C++ sort() or IntroSort or Introspective sort is the best sorting algorithm around. It is a hybrid sort (which means it uses multiple sorting algorithms) as it uses Quicksort, Heapsort and Insertionsort. Read more about IntroSort here.
- Bonus : Boost Library

Day 4

Classes in C++

One of the major reason for shifting to C++ from C was to solve the OOPS problem. We will not discuss OOPS here but will show you how classes are implemented in C++

Watch this video of 'The Cherno' to get an intro to classes and how they are implemented

Also take a look at this GFG article which will cover most of the things about classes in C++ here

Classes Vs Structs

Now you might be wondering that all the things that we can do using classes can be done using structs in C++. Note that C++ structs are more powerful than C structs in the sense that you can declare functions inside C++ structs. Moreover you can also manage visibility of functions and variables

Basically, there is little to none difference between them
Watch this video to understand it more clearly

Auto Keyword

The 'auto' keyword was added in C++-11. It was a step towards making C++ a more loosely typed language. Like in C, you have to mention the data-type of every variable that you declare, in C++ you can use the auto keyword which automatically detect the data type of the variable

Note that when you declare a variable you have to initialise it their itself if you are using 'auto' keyword

Learn more about it here

Iterators in C++

Iterators as the name suggests is an object that can iterate over elements in a container and provide access to individual element. Watch this video to learn about iterators in C++

Some functions that are used on iterators are discussed here

Iterator based looping of containers :

You can iterate over the elements of the container using the following syntax

```
int main(){
    vector <int> v = {1,2,3,4,5,6};
    for(int i: v){
        // i is a copy of the element
        cout << i << " ";
    }
}
```

```
int main(){
    vector <int> v = {1,2,3,4,5,6};
    for(const int &i: v){
        // i is the lvalue reference
        cout << i << " ";
    }
}
```

Note that while iterating over the container like this always the copy of the current element is created. To avoid this behavior you can use lvalue reference more on this in later weeks
OPTIONAL:
If you are interested to learn more about iterators you can see how you can write your own iterators

Data Structures Vs Containers

Data structure is a term from computer science, it can be used in a theoretical discussion without reference to any specific language. There are many data structures discussed in books about algorithms, like queues, trees, hash-tables etc. A data structure can be implemented in virtually any language.

Container is a term rather used in the context of a specific language, when there is an available library of already implemented containers that the programmer can use right away. Several distinct libraries can implement the same (abstract) data structure (for instance, a simply linked list), but with distinct containers, i.e. distinct source code and distinct names (one could be called "queue", another one "fifo", and a third one "stack"). You can learn more about containers here

Day 5

The C++ Standard Template Library (STL):

What is STL?
STL stands for Standard Template Library. It is a collection of template classes and template functions that provide a generic way of programming.

STL was written way before C++ and was written by Alexander Stepanov. It is often confused with the C++ Standard Library, which in fact was inpired by the STL.

It has four components:
- Containers
- Algorithms
- Functions
- Iterators

Article on STL.

Containers

Containers or container classes store objects and data in a variety of ways.
They can be further subdivided into 4 categories-

I) Sequence containers - They store data that can be accessed in a sequential manner.
- Vector : dynamic contiguous array.
- List : non contiguous array.
- Deque : Double-ended queue.
- Array : contiguous and non dynamic.
- forward_list (Introduced in C++11) Singly-linked list.

II) Adaptive containers - They provide a different interface for sequential containers.
- Queue : a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.
- Priority Queue : a queue in which the top element is either greatest or lowest.
- Stack : a linear data structure that is open at only one end and the operations are performed in Last In First Out (LIFO) order.

III) Associative containers - They implement sorted data structures that can be quickly searched (in O(log n)).
- Set : container used to store unique values arranged in ascending or descending order.
- Multiset : container that can store multiple same values arranged in ascending or descending order.
- Map : container that stores key value pairs, no two maps can have same key.
- Multimap : container that also stores key value pairs, however, two maps can have same key.

IV) Unordered associative containers (Introduced in C++11) - They are similar to associative containers, but as the name suggests, their elements are unordered.
- Unordered Set
- Unordered Multiset
- Unordered Map
- Unordered Multimap

More study material :
Videos-

- Video 1 (see after day 7 preferably)
- Video 2

If you prefer reading,
- Article 1
- Article 2

Day 6

Algorithms

The header algorithm defines a collection of functions specially designed to be used on a range of elements. They act on containers and provide means for various operations for the contents of the containers.

Some of the most used algorithms on vectors (must for one's who want to excel in competitive programming):

- Sort

This function internally uses IntroSort. In more details it is implemented using hybrid of QuickSort, HeapSort and InsertionSort. By default, it uses QuickSort but if QuickSort is doing unfair partitioning and taking more than N*logN time, it switches to HeapSort and when the array size becomes really small, it switches to InsertionSort. More on sort() here.

If you want to explore QuickSort, HeapSort and InsertionSort, then refer:
- Article on QuickSort.
- Article on Heap Sort.
- Article on Insertion Sort.

- Partition algorithm:

  You can read about it here.

- Some useful array algorithms

  You can read about them here.

- C++-98 introduced a special container valarray:

  It holds and provides mathematical operations on arrays efficiently. Read about them here.

For all algorithms in detailed version, refer:

- Article 1
- Article 2

More about the algorithms library in Week 3...

Day 7

Iterators and Functors

Many functions in the C++ standard library operate with iterators, a variable that points to an element in a data structure, it is quite similar in usage to a simple pointer used in C or C++

The two most commonly used iterators are -
1. begin() : points to the first element in the data structure.
2. end() : points to the position just after the last element.

We can define iterators for different data structures as follows :

1. For a vector ‘v’ , v.begin() and v.end()
2. For a set ‘s’, s.begin() and s.end()
   and so on

These iterators are used in conjunction with the different STL functions.
Some very important and commonly used functions are :

1. sort()

This is used to sort a vector or an array.
Vector - sort(v.begin(), v.end());
Array - sort(a, a+n);
In the case of array, we need not use any iterator, and a simple pointer is given as the parameter.

2. reverse()

This is used to reverse the elements of an array, and the implementation is similar to sort.

Declaring the begin() iterator for a vector ‘v’ can be done as follows -
auto it = v.begin();

‘*itr‘ returns the element pointed by the iterator.

To read about Iterators in detail,
This article has implementation of some less commonly used iterators
This part contains some important information about iterators with reference to Competitive Programming.

Several extremely important functions such as find() in vectors and sets and lower_bound(), upper_bound() in sorted vectors and sets return iterators.
Find, Lower_Bound and

Functors

Functors are objects that can be treated as though they are a function or function pointer, used along with STL.

A functor (or function object) is a C++ class that acts like a function. Functors are called using the same old function call syntax. To create a functor, we create a object that overloads the operator(). One is that unlike regular functions, they can contain state, meaning you can apply a function with a condition(which is not hardcoded) , they differ from function pointers in this respect. Functors are heavily used with generic programming. Many STL algorithms are written in a very general way, so that you can plug-in your own function/functor into the algorithm. You can use the following video or follow this discussion to learn more about functors. You can see the standard functors made in cpp here.

Now let's see how a functor can be used by other functions (such as for_each, which takes a range and a function to apply on each element of the given range.)

```cpp
#include <bits/stdc++.h>
class ElementPrinter{
public:
    void operator()(const std:: string& i_element) const {
        std:: cout << i_element << "\n";
    }
};

int main () {
  std:: vector <std:: string> strings {"A", "thing", "of", "Beauty"};
  std:: for_each(strings.begin(), strings.end(), ElementPrinter());
  return 0;
```