THAT’S ALL YOU GOT IN C++

1. *Source code 🡪 compiler 🡪 Assembler 🡪 Object code 🡪 Linker 🡪 Executable file 🡪 Loader*
2. C++ header files

Preprocessor directives - these get processed before the compilation of the source code.

In C++ it’s not necessary that every header file ends with .h extension but it’s a must in case of C.

The preprocessor directive #include directs the compiler to process these files before the execution of the source code.

Standard library header files - #include<stdio.h> or #include “iostream”

User defined header files - #define limit 5

1. Assembly Language: It’s neither in binary form nor high level. It is an intermediate state that is a combination of machine instructions and some other useful data needed for execution.
2. Assembler: For every platform (Hardware + OS) we will have an assembler. They are not universal since for each platform we have one. The output of the assembler is called an object file. Its translates assembly language to machine code.
3. Interpreter: An interpreter converts high-level language into low-level machine language, just like a compiler. But they are different in the way they read the input. The Compiler in one go reads the inputs, does the processing, and executes the source code whereas the interpreter does the same line by line. A compiler scans the entire program and translates it as a whole into machine code whereas an interpreter translates the program one statement at a time. Interpreted programs are usually slower with respect to compiled ones.
4. Relocatable Machine Code: It can be loaded at any point and can be run. The address within the program will be in such a way that it will cooperate with the program movement.
5. Loader/Linker: It converts the relocatable code into absolute code and tries to run the program resulting in a running program or an error message (or sometimes both can happen). Linker loads a variety of object files into a single file to make it executable. Then loader loads it in memory and executes it.
6. Linker error occurs when the compiler finally executes the file by
7. Main is a special function because it being declared to return int if not done so it will automatically assume it to return 0.
8. Long double is most accurate
9. Rounding Off

a. floor(x) – rounds off the value of x to closest integer which is less than the given value. (1.71—1)

b. ceil(x) – rounds off the value of x to the closest integer which is greater than the given value. (1.42 – 2)

c. trunc(x) – removes all the digits after decimal point

d. round(x) – rounds the number to closest integer.

1. Precision – sets the precision correct to decimals,  cout << fixed << setprecision(2)<<x
2. [Functions](functions.cpp)

In case of call by value the value of the original caller may change if changes are made in the function but if we pass the variable by reference there is no change in the original variable even if some operation is done on it in the function.

1. [Return statement vs exit statement](https://www.geeksforgeeks.org/return-statement-vs-exit-in-main/)

In case of exit(0) destructor is not called and the program exits while in case of return(0) the destructor is allowed to work.

1. Floating point arithmetic is so silly –

double a=0.1, b= 0.2 , c=0.3;

if(a+b==c) cout<<"not executed!";

else cout<<"executed!";

1. Sizeof(arr)/sizeof(arr[0]) – is the limit of iterations
2. Static 🡪 If you declare a variable with static keyword then it is initialized to 0. (\*variable next to it is initialized to 1).
3. stoi(str) 🡪 it returns the integer value of a string
4. initializing an array with constant in a range

int arr[16]={[0 ... 15] = 1};

1. When we use inbuild function sort() in C++ STL’s we get ascending order sorting but if we want to sort in descending order we can do🡪Sort(v.begin(),v.end() , greater<int>());
2. 2D arrays are stored in memory either as row major or column major but in java it creates individual arrays separately in linear fashion with not necessarily being in contiguous location.
3. lexicographical order 🡪 lexicographic or lexicographical order is a generalization of the alphabetical order of the dictionaries to sequences of ordered symbols
4. to find the middle of the linked list use two pointer use and iterate the slow by 1 and fast by 2 and while fast is not null till end the slow will point to the middle element for even.
5. there is a memory-efficient version of Doubly Linked List that can be created using only one space for the address field with every node. This memory efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bitwise XOR operation to save space for one address. In the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.

1. cout<<a[0][0]<<" "<<a[1]; // this prints value for the first and address for the second command.
2. The operator \* is used for dereferencing and the operator & is used to get the address. These operators cancel effect of each other when used one after another. We can apply them alternatively any no. of times. For example \*ptr gives us g, &\*ptr gives address of g, \*&\*ptr again g, &\*&\*ptr address of g, and finally \*&\*&\*ptr gives ‘g’
3. New() and delete()
4. Dangling pointer
5. We can make an array using   
   auto \*arr = new int[size];
6. Call by reference is automatically done in functions in case of arrays and objects.
7. To check if a part of string or a character is whitespace, we can use isspace(\*thing\*)
8. size\_t 🡪 The good thing about size\_t is that we can be certain that it is big enough to contain the size of the biggest object our system can handle. For example, it can be a static array of 4 GB. Technically, it can be smaller than, equal to, or larger than an “unsigned int”.
9. we can sort a data by **creating a min heap for the priority queue by the following exact syntax 🡪**

priority\_queue <int, vector<int>, greater<int>> pq;

now push all of the elements and print it by pop method and we get the sorted data.

…………………

1. **memset**(str, 't', **sizeof**(str));
2. Number of Binary Search trees for n nodes is given by = (2n)! / n! \* (n+1)!  
     
   Read more at: <https://edurev.in/question/509896/How-many-distinct-binary-search-trees-can-be-creat>