C++ Setup (1)

Installing compiler and configuring with VS Code

For compiling C++ programs to machine code (executables), you need g++ compilers. Generally with Mac and Linux, g++ comes preinstalled. Check using the command

```
g++ --version
```

Copy

For installing on Windows, you need to install mingw and then add it to PATH. Tutorial: https://code.visualstudio.com/docs/cpp/config-mingw#_prerequisites

You can also setup a VS Code extension for features like Code Completion, Running on VS Code by clicking button etc.

https://code.visualstudio.com/docs/languages/cpp

https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner

How it all works?

1. You write your code in a file with extension `.cpp` . For example: `main.cpp`
2. The you convert your CPP file to a machine executable using the compiler: `g++ main.cpp` .
3. This creates an executable file which depends on the OS. For example, in Windows it creates `a.exe` whereas in UNIX systems (Linux/Mac) it creates an `a.out` .
4. You can execute this file to run the program and get output. `./a.out`
5. If you want to change the name of the executable, you can pass the `-o` argument: `g++ main.cpp -o main.out`

Let's try it out

1. Create a new file and open it in VSCode, or some text editor.
2. Paste the following code

```
#include<iostream>

using namespace std;

int main() {
   cout<<"Hello World!"<<endl;
   return 0;
}
```

3. Save the file with the name `main.cpp` .
4. Now compile the file and create an executable using the following command: `g++ main.cpp -o main.out` (Replace it with `g++ main.cpp -o main.exe` in case of Windows machine)
5. You can see a new file `main.out` (or `main.exe`) got created in the same directory (or current working directory)
6. Let's run the program now. Run `./main.out` if you are in Unix systems (Mac / Linux) or `.\main.exe` if you are in Windows machine. You should see "Hello World!" printed on your terminal.

💡 In most cases you would not need a local setup, there are many online IDEs out there and most platforms also gives you with an IDE. However, it is always good to learn to set it up locally and how to compile and run a C++ code. In some platforms (like Codeforces) you don't get an IDE and also when you attend onsite programming contests, it might be faster for you to write things locally than on an online IDE.

Understanding our basic C++ program (1)

```
#include<iostream>

using namespace stdl

int main() {
   cout<<"Hello World"<<endl;
   return 0;
}
```

Copy

Let's understand it part by part

1. `#include<iostream>` : This is how we import libraries in C++. Consider it same as `import axios from "axios"` . The general structure is `#include<library_name>` . `iostream` is a library that imports the I/O objects.
2. `using namespace std;` : In C++, a namespace is a collection of related names or identifiers (functions, class, variables) which helps to separate these identifiers from similar identifiers in other namespaces or the global namespace. Almost everytime, we will be using the `std` namespace only, so don't worry about this a lot.
3. `int main()` : `main` function is the entry point for a file / program. `int` is the return type of the main method. Generally we will return 0 to indicate successful completion. We can also return error codes to indicate what kind of error occurred. However, in our case we will only return 0.
4. `cout` : Used for printing the output to the terminal (or some file). Similar to `console.log()` in Javascript
5. `endl` : Prints a new line after the output is printed. In C++, we need to manually specify newlines using `endl` or `\n` . For example:

```
cout<<"Hello ";
cout<<"World";

Output: Hello World
```

Copy

1. `return 0` : It returns 0 from the main function.

What libraries to import?

An obvious question comes to our mind, what are the different libraries we need to import? We can import different libraries separately for getting different functionalities and data structures.

For example, `iostream` for I/O operations, `stack` for Stacks, `vector` for vectors (dynamic arrays).

However, don't worry! You don't need to import all of them. There is a special library called `bits/stdc++.h` that includes all these common libraries. You will in 99% of the cases (if not 100), not need to import anything else. Import it using:

`#include<bits/stdc++.h>`

💡 bits/stdc++ library is distributed with GCC. So incase, you see error like bits/stdc++.h is not found, install GCC (in case of Mac you can use `brew install gcc`)

I/O operations (1)

Output

For output, we can generally use `cout` statement.

```
cout<<"Hello World!";
```

Copy

To print new lines, we can use `endl` or `\n` .

So let's see what the difference is.

`'\n'` : It inserts a new line to the output stream.

`endl` : It inserts a new line and flushes the output stream.

So eventually

`cout<<endl;`

is equivalent to using

`cout<<'\n'<<flush;`

But which one to use? `\n` is much faster than `endl` . So use `\n` every time if possible (unless you need to flush the output). However, when you are printing a single element or some fixed set of elements (like 100), you can use `endl` as well.

💡 Note: By default, all streams (input and output) are tied together. Tied streams ensure that one stream is flushed automatically before each I/O operation on the other stream. To untie, we can use
`cout.tie(NULL);`

Input

For reading input, we generally use `cin` statement.

```
int x;
cin>>x;

// We can even read multiple space or newline separated inputs like this
int x, y;
cin>>x>>y;
```

Copy

Fast IO

Generally, as mentioned above, the input and output streams are tied, which causes flushing before every IO operation. This can make program execution slower in case of large input size. So we can untie the streams using the statements `cin.tie(NULL)` and `cout.tie(NULL)` . Just add this at beginning of your program.

`std::ios_base::sync_with_stdio`

Sets whether the standard C++ streams are synchronized to the standard C streams after each input/output operation. In practice, this means that the synchronized C++ streams are unbuffered, and each I/O operation on a C++ stream is immediately applied to the corresponding C stream's buffer. This makes it possible to freely mix C++ and C I/O.

In addition, synchronized C++ streams are guaranteed to be thread-safe (individual characters output from multiple threads may interleave, but no data races occur).

If the synchronization is turned off, the C++ standard streams are allowed to buffer their I/O independently, which may be considerably faster in some cases.

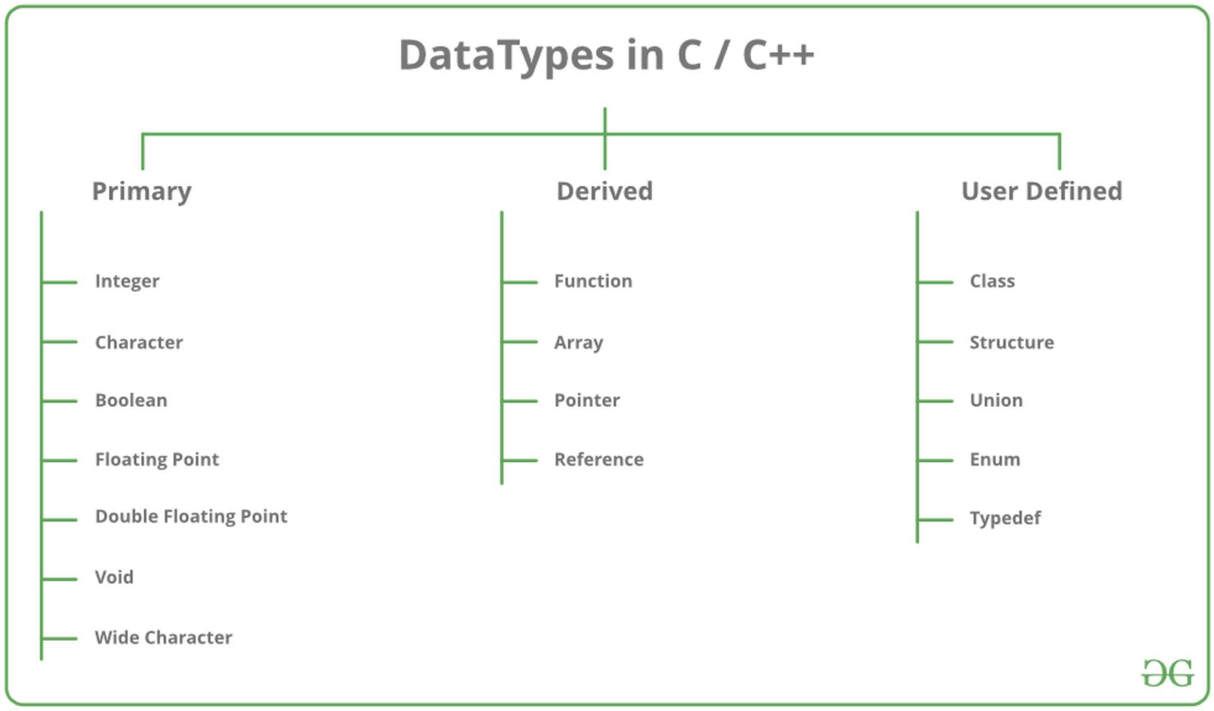
So overall, add these 2 things before the start of any program. Example:

```
#include<bits/stdc++.h>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);

  // Your program here
}
```

Data types in C++ (1)

C++ is a strictly typed language. So you need to specify the data type of all variables before assigning them values or using them. Generally, data types can be categorised into 3 types:

1. Primary / Built-in data types
2. Derived data types
3. User defined data type



Sizes of Primary data types

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Conditional statements in C++ (1)

C++ like Javascript has the following conditional statements:

1. if else
2. switch
3. conditional operator

If Else statements

```
if (condition_1) {
  // Executed if condition_1 is true
} else if (condition_2) {
  // Executed if condition_1 is false and condition_2 is true
} else {
  // Executed if both condition_1 and condition_2 are false
}
```
Copy

We can even have nested if else statements or just single if statement

```
// Only if statement
if (condition) {}
```
Copy

```
// Nested if else
if (condition_outer) {
   if (condition_inner) {}
   else {}
} else {}
```

Switch statements

```
switch(variable) {
   case 1:
       // When variable is 1
       break;
     case 2:
         // When variable is 2
         break;
     default:
         // When none of the case statements is satisfied
}
```

Copy

💡 The case value must be of type integer or character

💡 Like Javascript, default statement is optional. Also break keywords are optional.

Conditional operators

```
condition ? true_statement : false_statement
```

Copy

Examples:

```
bool is_even = (n % 2 == 0) ? true : false;

int absolute_diff = (a > b) ? a - b : b - a;
```

Copy

Looping statements (1)

Similar to Javascript, C++ also supports `for` and `while` loops. There is another looping statement called `do while` but you will rarely encounter this, so we will be skipping it.

for loop

Copy

```
for (int i=0; i<n; i++) {
  // do something
}

// You don't always need to iterate on integer values only. Example:
for(char i='a'; i<='z'; i++) {
  // do something
}

// We also have for-each loop in C++, similar to javascript
for(int x: arr) {
  // You get each array values one by one in x.
}

// We can also have nested loops
for(int i=0; i<n; i++) {
   for(int j=0; j<n; j++) {
      // Nested loop example
   }
}
```

The middle statement is a `conditional statement` and the last statement is `stepping statement`. The conditional statement can be anything, that breaks at some point. The stepping statement, should change the iteration variable in any way. We can also initialise and use multiple iteration variables.

```
// Example of different way of writing iteration variable    Copy
for(int i=0, j=n-1; i<j; i++, j--) {
  // Step 1: i = 0, j = n-1
  // Step 2: i = 1, j = n-2
  // Step 3: i = 2, j = n-3
}
```

Early exits

Sometimes, we might want to break the loop before it runs to completion. Or we might want to skip to the next iteration in some cases. We can do these using `break` and `continue` statements respectively.

```
for(int i=0; i<n; i++) {                                              Co
   if (i == 5) break; // We break when i reaches 5. So it does not continue with rem
   if (i % 2 == 0) continue; // We don't execute the loop for even values of i, just
   sum += i;
}
```

while loop

```
while (i < 10) {                        Copy
  // do something
  i++;
}
```

Functions (1)

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Functions take in some arguments and returns (may or may not) some value. The return type of a function determines what type of data is returned. If return type is `void` it does not return any value.

Examples:

Copy

```
int sum(int x, int y) {
   return x + y;
}


void print_data(int x) {
        cout<<"Number is: " << x <<endl;
}

int find_max(int a, int b) {
   if (a > b) return a;
   return b;
}

string concatenate(string a, string b) {
  string res = "";
  for(char x: a) res.push_back(x);
    for(char x: b) res.push_back(x);
  return res;
}
```

Function in Overloading

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different **types of arguments** or a different **number of arguments**.

```
int add(int a, int b) {
  return a + b;
}

int add(int a, int b, int c) {
   return a + b + c;
}

add(2, 3); // Works
add(2, 3, 4); // Works as well
```

```
int add(int a, int b) {
   return a + b;
}

float add(float a, float b) {
   return a + b;
}

string add(string a, string b) {
   return a + b;
}

add(2, 3); // Works
add(2.2, 3.5); // Works as well
add("ab", "cd"); // Works as well
```

Arrays (1)

Collection of elements of similar type is called an array. Similar to javascript. However, as C++ is strictly typed, we can only store elements of a single type, unlike Javascript.

Each array element has some index. It starts from 0. We will learn more about the actual meaning of indices, once we go through pointers.

Creating Arrays

Approach - 1

```
int arr[] = {1, 2, 3, 4};

// You either need to specify the size or initial elements.
int arr[10];
```

To create dynamic sized arrays, we can do the following:

```
int n;
cin>>n;
int arr[n];

// Now take input
for(int i=0; i<n; i++) {
   cin>>arr[i];
}
```

Approach - 2 (Using vector)

The above arrays created have fixed size (i.e. once created, the size is not changed). If you want to dynamically change array size, you can use pointers to create a dynamic array and change its size using `realloc` . However, there is a C++ collection called `vector` that allows us to create dynamic arrays.

Looping on arrays

Similar to javascript we can run a for loop

```
int arr[n];

for(int i=0; i<n; i++) {
   cin>>arr[i];
}
```

We can also run for-each loops as well, these assigns the array values to the variable directly instead of needing to access by index.

```
for(int x: arr) {
   cout<<x<<" ";
}
```

Accessing elements

We can access elements of array using their index. For example, `arr[3]` gives the element at index 3, which is the 4th element of the array (Remember, array indices are 0 based)

Updating elements

We can assign values to array elements using `=` operator. Example:

```
arr[0] = 10;

arr[i] = i + 10;
```