C++ Basic Input/Output

In case you want to read basic **input/output** in **Java** and **Python**.

When you embark on your C++ programming journey, it's perfectly okay not to dive too deep into the intricacies of the language right from the start. In fact, it's advisable to initially focus on grasping the big picture and building a strong foundation. In this guide, we'll walk you through the basic skeleton of a C++ program and the essential components you need to know to get started.

Including Libraries

C++ is a versatile language, and it relies on libraries to access various functionalities. To perform tasks like input and output, we include specific libraries at the beginning of our code. For instance**,** **#include<iostream>** is used for input and output operations, while **#include<math.h>** allows us to use mathematical functions. Simply put, libraries provide pre-built functions and tools for us to use in our code.

The Generic Skeleton

The generic skeleton of a C++ program consists of two main components: the **library inclusion** and the **main function**. After including the necessary libraries, you declare the main function using int main() { /* Your code here */ return 0; }. This serves as the entry point for your program.

```
#include<iostream>

int main() {
    // Your code here
    return 0;
}
```

Output with cout

To display output in C++, you'll commonly use the cout function from the iostream library. However, you need to specify that it belongs to the std (standard) namespace. For instance, **std::cout << "Hey, Striver!";** will print "Hey, Striver!" to the console. You enclose the text you want to display within double quotation marks.

Code:

If we want to print **Hey, Striver! twice in 2 lines** and we write **std::cout << "Hey, Striver!";** again and again then it will print it consecutively on the same line.

Code:

You can use the **newline character \n** to insert a **line break** within a single std::cout statement. Here's the code and its corresponding terminal output:

Code:

[tabby title="C++ Code]

```
#include<iostream>

int main() {
    std::cout << "Hey,
Striver!" << "\n";
```

```
    std::cout << "Hey,
Striver!";
    return 0;
}
```

Output:

```
Hey, Striver!

Hey, Striver!
```

As you can see, the newline character \n inserts a line break, but the second "Hey, Striver!" is still on the same line as the first one.
You can also use **std::endl** to insert a **newline character** and **flush the output buffer**. Here's the code and its corresponding terminal output:

Code:

Using \n for line breaks in C++ is a common and efficient way to achieve the desired output. It's a simple and straightforward approach, and it's **typically faster** than other methods for adding line breaks, such as using std::endl.

The reason for this is that \n is a simple escape sequence that inserts a newline character, which is a **low-level operation** that directly m**oves the cursor to the beginning of the next line in the output**. On the other hand, std::endl not only adds a newline character but also **flushes the output buffer**. Flushing the buffer can be a more costly operation in terms of performance, especially when you're printing a large amount of text.

using namespace std

By adding using **namespace std**; at the beginning of your program, you're telling the compiler that you want to use all the **names from the std namespace** without explicitly specifying std:: each time. This can make your code cleaner and more concise.

Code:

"using namespace std;" is a useful shortcut for simplifying your C++ code, especially when you're learning the language and writing smaller programs. It helps **reduce clutter** and makes your **code more readable**. However, as your programming projects grow in complexity, consider using it sparingly to avoid potential naming conflicts. It's a tool that can make your C++ journey smoother, so use it wisely.

Taking User Input using cin

One of the fundamental aspects of programming is taking input from the user. In C++, this is achieved with the help of the **cin stream**, which allows you to **receive input** from the user via the terminal or console.

Code:

When you run this program and enter a value (e.g., 10) in the terminal, cin captures that value, stores it in the variable x, and then displays it using cout. Here's how it works:

1. The program waits for user input.

2. You enter a value (e.g., 10) and press Enter.
3. cin reads the entered value and stores it in the variable x.
4. cout then displays the value of x.

To accept multiple inputs, we can simply use the >> **operator** with cin for each variable we want to receive input for. Let's demonstrate this by taking two variables, x, and y, as input and displaying their values:

Code:

When you run this program, it waits for two separate inputs from the user, which should be entered one after the other, separated by spaces or Enter key presses. Here's how it works:

1. The program waits for the first input (for x).
2. You enter a value (e.g., 10) and press Enter.
3. cin reads and stores the value in x.
4. The program then waits for the second input (for y).
5. You enter another value (e.g., 20) and press Enter.
6. cin reads and stores this value in y.

You might be wondering about the meaning of "int x" and "int y." Let's now dive into the topic of data types in C++.

Note:

To make the process more convenient, there's a shortcut that allows you to include almost all standard libraries at once using **#include<bits/stdc++.h>**.

The bits/stdc++.h header is a shortcut that includes a **vast number of standard C++ libraries**, making it easier to access a wide range of functions and classes without specifying each library individually. It's a **time-saving approach** for programmers, especially when you need several standard libraries in your code.

However, it's important to be aware of potential compatibility issues and consider the impact on compile time, especially in large projects. When used judiciously, it can be a valuable asset in streamlining your C++ development process.