An Introduction to Competitive Programming and C++

CS Team Captains

September 17th, 2020

1 Introduction

Welcome to NCSSM's CS Team, one of the best clubs on campus! Our main focus as a club is on programming competitions. We'll meet weekly throughout the year and learn about algorithms and data structures together. We hope to prep everyone for the United States of America Computing Olympiad (USACO), but also participate in a number of other competitions throughout the year. Our sponsor is the wonderful Dr. John Morrison who teaches a multitude of programming classes here at NCSSM. Your captains for the year will be:

- Siddhant Dubey
- Sreyas Adiraju
- Jonathan Xu
- Evan Zhang

2 Schedule

We meet every week on Thursday from 8-9 PM. Every meeting will consist of a lecture delivered by a few of the captains on Computer Science topics. Once in a while, we'll also have in-school contests for members of the club. If any changes occur in the schedule, you'll hear about it in the Facebook group.

If you ever miss a meeting, you can find the materials for it on our website: `https://ncssmcsclub.github.io/`.

3 Contests

3.1 USACO

USACO, the flagship programming contest of the United States of America, is an online contest that occurs once a month starting in December and ending with the US Open in March. The contest lasts for around four days each

month starting on a Friday and ending on a Monday evening. There are four divisions: Bronze, Silver, Gold, and Platinum that get progressively harder in difficulty. When you create your account, you start off in the Bronze division and can promote to higher divisions by meeting the cutoff score for each contest. The best programmers of the season are selected for the USA's IOI (International Olympiad of Informatics) training camp. Our material is geared towards preparing for the USACO so you should definitely try out at least one contest.

3.2 Other Contests

- **Virginia Tech's High School Programming Contest**: This is an online contest typically held in December. There are 13 difficult problems that you will try to solve in 5 hours. There's no limit on the number of teams that can participate from a school, so we don't hold tryouts. The max size for a team is 3 people. It is unknown if this contest will happen this year.
- **UVA's HSPC**: This is an in-person contest at the University of Virginia that occurs in the spring. We can take two teams of four and will hold tryouts sometime in the second semester.
- **Various Online Judges**: These are websites like Codeforces and AtCoder. They hold contests weekly and have an ELO system to rank you in comparison to other competitive programmers. They're pretty fun to participate in and don't last that long, so participate when you can, they're good practice.

4 What is Competitive Programming?

Competitive programming is an art form. It's creative problem-solving at its finest, a combination of hard analytical thinking and creativity. Competitive programmers use their knowledge of algorithms and data structures and logical reasoning skills to solve challenging algorithmic problems in a limited time frame.

The preferred language of Competitive Programmers is C++ with a large minority choosing Java. This is due to their relative run-time efficiency compared to other languages like Python. Of course, you should use whatever you feel comfortable with, but these lectures will be done in C++. The lecture notes will also contain Java code.

5 Introduction to C++

C++ is a low-level general-purpose programming language that is extremely quick thanks to its low-level nature. It is faster than other general-purpose languages like Python and Java by a decent margin. C++ is very similar to

Java in syntax and both languages are **statically typed** and compiled. C++ is also **Object Oriented** like Java is, meaning you can write classes, structs, and other cool things.

6 Format of a C++ Program

C++ functions center around the main() method that is run at the start of a program. The program can have various helper methods and global variables (you'll end up making a lot of arrays this way). Using global variables lets your code run quicker and it is good practice. In general, global variables are the easiest to work with and pass through different functions, instead of passing a local variable in the main function.

6.1 Your first C++ Program!

```
#include<iostream>
//This is a preprocessor command and it is executed before the program is compiled.

using namespace std;
//This line allows you to use functions from the STL without typing out std::

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

Let's walk through what this code is doing. The **include** statement in line 1 is a **preprocessor** command and is executed before the program is compiled. **Include, using, namespace, int,** and **return** are all **keywords** that are reserved by C++. The real meat of our code is occuring within the main() method. The cout function is responsible for outputting things on to the terminal in C++ and is used like so: cout << output. You can chain multiple 'put' operators together to combine outputs. In this case we combine "Hello, world!" with the keyword endl, which as you might have guessed acts as an endline character. This means whatever comes after the endl will be put on a new line and not the same line as previous outputs. The final line in the main() method is the return statement. All methods need return statements but you'll be ok if you don't add one to the main() method. However, it is good practice to include a return statement even when you don't need one.

7 Basic Input/Output in C++

C++ actually doesn't have a built in input/output system, but you can use the iostream header to make up for it. Contest problems will require you to read input and write output in one of two ways:

- From a file using either the **fin/fout** routines in the iostream header or the **fprint/fscanf** routines in cstdio. USACO falls under this category.
- From **stdin/stdout**, using either the **cin/cout** routines in the iostream header or the **printf/scanf** routines in cstdio. This type of problem is typically used by Codeforces and other online judges.

For simplicity, we'll be using the iostream functions. To use them, make sure you do include¡iostream¿ at the top of your C++ file. Here's an example block of code where we're reading in some input and writing our answer to a file for a USACO problem.

```
#include <iostream>

using namespace std;

int main(){
    ifstream fin("prob_name.in")
    ofstream fout("prob_name.out")
    int a, b, c;
    fin >> a >> b >> c;
    cout << "a: " << a << "b: " << b << endl;
    fout << a << b << c << endl;
}
```

As you can see, the program is reading in some variables a, b, and c from a file and then writing those same variables to another file. The cin routine also serves to take in input but does so from the terminal rather than a specific file.

8 Variables, Data Types, and Operators

The data types in C++ are very similar to ones from other programming languages, so if you've coded before this should all seem familiar. If not, no problem, you're about to learn a lot. This section will cover primitive data types like **ints, strings, chars, bools, floats, and doubles**. We'll also be covering the all powerful arrays.

8.1 Numerical Data Types

8.1.1 The Integer

The **int** data type requires 4 bytes of memory space and goes from -2147483648 to 2147483647.

8.1.2 The Float

The **float** data type is used to store single precision decimal values.

8.1.3 The Double

The **double** data type is used to store double precision decimal values.

8.1.4 The Long Long

The **long long** (long long int) data type has a larger capacity than the int data type and can go from -2^{63} to $2^{63}-1$. Use this when you think you're going to be dealing with large numbers that an int can't hold.

8.1.5 The Boolean

The **bool** data type can contain one of two values: true or false. You can also use 1 (true) and 0 (false). These are very important when you're writing programs that require **conditional logic**.

8.1.6 Code example!

C++ is statically typed, which means that you have to declare the type of every variable you create. Here's an example of how you would do that with the numerical data types that you were just exposed to.

```
#include <iostream>

using namespace std;

int main(){
    int x = 10;
    double y = 1.01;
    float z = 1.1;
    long long a = 200000000000;
    bool is_sreyas_cool = true;
    return 0;
}
```

8.2 Arithmetic and Assignment Operators

In addition to all the standard binary operators(+, -, *. /) and the simple assignment operator (=), C++ also supports the modulo operator(%), compound assignment(+=, -=, *=, /=, %=), bitwise operators and unary operators (++, –, !). The modulo operator gives the remainder after division, for example: 4 % 3 = 1. It is also important to note that there are two types of division: integer and float division. Integer division uses two // while float division uses 1 backslash. Integer division is truncating, which means that it doesn't round and just takes the whole number that is present. When you're working with floating points, you should expect precision errors. As for the order in which these operators work, C++ does have operator precedence that closely matches PEMDAS. Just use parentheses and you'll be fine.

8.3 Relational Operators

In addition to arithmetic and assignment operators, C++ also features relational operators used to perform conditional tests. These operatores return booleans and include:

- **Equivalence tests: == and !=**: == is checking for equality while != checks for inequality.
- **Comparisons: <, >, <= and >=**

Additionally, C++ allows for the compounding of conditions with logical operators: **logical and** (&&) and **logical or** (||) . C++ uses short-circuit evaluation for these operators: if the first condition is enough, the second condition will not be evaluated.

8.4 Strings and Characters

Dealing with Strings and Characters is essential for any program, because all programs need some level of human interaction.

- The **char** (character) data type, holds a single character, for example: 'a'.
- The **string** data type is what you would use to hold any sort of textual data that is longer than one character.
- It is important to remember that **chars** are in fact treated as numbers. C++ uses an encoding (ASCII) to map numbers 1-255 to printable characters.

Here's a code example that shows you how to create strings and characters in your own programs.

```
#include <iostream>

using namespace std;

int main(){
    char a = 'a';
    string word = "cool";
    return 0
}
```

You may have noticed that single quotes are used for characters and double quotes are reserved for strings. This may shock the Python users among you, but it is a harsh reality of programming in C++.

8.5 Arrays

Arrays in C++ are collections of items. Whereas Python lists can contain different data types, all the items in a C++ array must be the same data type. These could be any of the primitive data types we've already discussed. There are multiple ways to initialize arrays and they're shown in the code block below. The commonality in between them is that you always need to put the data type the array contains before the name of the array.

```
//Note: This code is within a main() method.
//Array Declaration by specifying a size:
int arr[10];

//You can also initialize an array with a size that is a variable:
int size = 100;
int arr2[size];

//Array declaration by initializing elements:

int arr3[] = {1,2,3,5}

//Array declaration by specifying a size and initializing elements:

int arr4[4] = {1,2,3,100}
```

When you initialize an array without specifying any elements in the array, you make an empty array where every space in the array is empty. Arrays are extremely valuable tools you'll be using a lot in your Competitive Programming journey.

8.5.1 Indexing Arrays

Oftentimes you want to be able to retrieve an element in a particular location in the array. In C++, arrays are 0 indexed, which means the first element in the array is at position 0, the second element is at position 1, and so on.

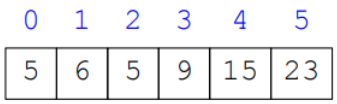

Figure 1: An Array

To output the *nth* element of an array *arr*, you would write the following line of code:

```
cout << arr[n-1];
```

9 Conditional Logic

Conditional statements enable you to create control flow in your programs, and are essential in most algorithimic tasks.

9.1 If/Else Statements

If-else statements are an essential tool for any programmer and they take the following form in C++

```
//Note: This code is within a main() method.
if(/*boolean condition*/){
    //Code to execute if true
}
else{
    //Code to execute if false
}
```

The curly braces that you see in the code are used to define a block of code, establishing a scope for variables defined within it. However, if you only have one line of code to execute in a statement, you can omit the curly braces.

9.2 Else If and Nested If statements

Sometimes you just have a lot of conditions to check, and in that case you can chain them together in an **if ... else if ... else** structure.

```
//Note: This code and all future snippets that look like it are in a main() method.
int x = 3;
if(x < 2){
    x = 2;
}
else if(x == 2){
    x += 2;
}
else{
    cout << "SUCCESS!" << endl;
}
```

The code above checks if some integer x is less than 2, which x is not, then checks to see if it is equal to 2, and if it is neither of those two things the code outputs "success" to the terminal.

Other times, you have different things to check based on the first conditional statement. In that case you would use **nested ifs** which place one condition within another conditional statement. For example:

```
if(x <= 17 && y > 0){
    if(x <= 2 && y > 3)
        cout << "cool" << endl;
}
else{
    cout << "not cool" << endl;
}
```

Huzzah! You now know how to use conditional logic in C++ and you're one step closer to being a Competitive Programmer god.

10 Loops

Loops are an essential part of any Competitive Programmer's toolbox. They allow you to execute similar code an arbitrary number of times and are invaluable in competitive programming. They have an infinitude of uses and it is of the utmost importance that you are comfortable with using loops.

10.1 For Loops

For loops are used to do something **for** a specified number of times. The for loop header consists of three parts:

- An **initialization**, often used to declare and assign an iterator, typically an **int i** or **j**.
- A **condition** used to determine whether the loop runs.
- An update step, typically used to increment the iterator.

The loop body contains the code that you want to run in the for loop. Here's an example of a for loop that is outputting the first n elements in an array.

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

In the for loop above our iterator is an integer i which is initialized with a value of 0. Our condition is that i must be less than n, and our update step adds 1 to i every time the for loop body executes. The body outputs the ith element of the array starting with the 0th element.

10.2 More Complex and Advanced For Loops

The above for loop is the more traditional "C Style" for loop, and all four of the above parts are optional. The following for loops are valid:

```
int i = 0;
for(; arr[i] > 7; i++);

for(;;){
    //infinite loop
}
```

The **enhanced for-loop** allows you to more conveniently iterate through array-like types:

```
int[] arr = {2, 3, 4, 5, 6, 7};

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

cout << endl;
```

This type of for loop is also known as a *for each* loop on account of it going through each element of an array.

10.3 While Loops

While loops execute until the supplied **termination condition** returns **false**.

The **while** loop has a few important uses (in graph algorithms):

```
//Iterating while inserting/removing
//Don't worry about this code block, you'll understand it soon enough.
while(!Q.empty()){
    Thing t = Q.top();
    Q.pop();
}

//Screening faulty input data
while(!valid()){
    //Reprompt user for input
}

//Fake for loop
int i = 0;
while(i < n){
    cout << arr[i] << endl;
    i++;
}
```

11 Functions

A **function** in C++ is a code block that is only run when it is called on. Functions have different data types that they return which include all the primitive data types we have already discussed and others like the **void** type. **Void** simply means there is no value, so it is used for function types that don't return any sort of value.

```
//hello, add(), and main() are all functions.
//Just like variables you have to specify the type of data a function will return.
void hello(){
    cout << "Hello" << endl;
}

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

int main(){
    hello(); //Outputs "Hello"
    cout << add(1, 2);
    return 0;
}
```

You'll often see Competitive Programmers using functions to make their code look neater as well as modularize it so that it becomes easier to test.

12 Conclusion

This should serve as a good quick introduction to C++, but you won't really know C++ until you start writing a lot of code. A good place to get practice when starting out is Hackerrank. For more challenging problems, check out CSES and Codeforces.

For "homework" attempt the first 5 problems on CSES's site: *Weird Algorithm, Missing Number, Repetitions, Increasing Array, and Permutations.* You'll come out feeling more confident in your C++ skill and a warmed-up brain ready to tackle more Competitive Programming problems as they come. However, and I cannot stress this enough, **DO NOT LOOK UP THE ANSWERS!** If you are having difficulty, reach out to one of us captains or another club member, but don't look up a solution unless you've struggled for an hour or two. Contests like the USACO give you 4-5 hours for 3-4 problems and most people send the entire time period getting 1 or 2 problems done. It is natural to struggle when you're starting with something but it is essential that you embrace that struggle rather than run away from it. Happy Coding!