Computer Science Club Introduction

Competitive programming and CCC

We will be spending most of our time this year practicing for the Canadian Computing Competition (CCC).

CCC format: you are given 5 problems in roughly increasing order of difficulty and must solve as many of them as possible in 3 hours. Each problem will have a well defined input and output specification. Your program will be run on a series of test cases, each having different input. Your program will be considered correct if it produces the correct output for all test cases. Your program must also run under a specified time limit and will also have a limit on how much memory it can use.

A few differences between competitive programming and other general programming:

- Do not prompt the user for input, your program will be graded by a computer and should follow the output format specified in the problem statement
- While having clean code is nice, it is not the priority; it is more important to get code that is efficient and works. Note that your code still has to be clean enough that you are able to debug it.

You can find out more about the CCC specifically at https://cemc.uwaterloo.ca/contests/computing.html

Proposed agenda for the year

Week 1	Programming fundamentals: variables, if, for, while, operators, functions
Week 2	Big O, basic data structures: array, list, set, map, hash table, linked list
Week 3	Sorting: bubble sort, merge sort, quick sort, Binary search
Week 4	Graph representation, breadth first search, depth first search
Before CCC	Recursion, prefix sum array and difference array, Dijkstra's algorithm, basic dynamic programming (DP), DP with 2D state, bitmask DP, basic game theory, disjoint set union, minimum spanning tree, segment tree, ternary search, square root decomposition, basic modular arithmetic
Likely will do many of these only after CCC	Balanced binary search tree, binary indexed tree, lowest common ancestor, line sweep, sparse table, Euler tour tree, binary search on binary indexed/segment tree, 2D versions of data structures, strongly connected components, randomized algorithms
Likely will only do a select few of these	Convex hull trick (DP), string algorithms (hashing, KMP, suffix array), centroid decomposition, heavy-light decomposition, min-cut max flow, matrix exponentiation, Fast Fourier Transform, link-cut tree, regular expression implementation, functional programming

Programming Languages

There are going to be three main languages we will be using this year. It is enough if you learn at least one of them. Here is a table with some of the advantages and disadvantages of each.

C++	Java	Python
+ Has the best performance + Overall the best language for competitive programming - Supposedly slightly harder to learn than the other two - Does not have a built-in garbage collector for memory management (not important for CP though)	+ Only a bit slower than C++ - Syntax can get somewhat verbose	+ Supposedly easy to learn + Syntax is short - Very slow

DMOJ

https://dmoj.ca/ is a platform where you can practice competitive programming.

It has many problems, including ones from the CCC, that you can submit to. Your submission will be run on the test cases and you will receive the verdict of your submission. This is the same general process as what is used at the actual CCC.

You can view other people's submissions to a problem once you have solved it.

Problems have a point value, a number between 1 and 50, which is supposed to reflect their difficulty. Start by doing easier problems (3-5 pointers). Skip 1 pointers as most of them are troll questions, mostly from April's Fools contests.

DMOJ also hosts contests, which usually run in roughly the same format as the CCC. Note that the difficulty of the problems may not be the same as the CCC. Contests that are rated are guaranteed to be of decently high quality and will use a format that is close to standard. Unrated contests may still be worth doing, although the problems are not guaranteed to be of as high quality or the format may be unusual.

Meeting format

In person meetings will be more focused on you actually coding and asking for help if needed.

Virtual meetings will be more focused on a teacher explaining a topic. Questions are still welcome, but from past experience we have found that students are usually not very active in these meetings.

Week 1: Programming fundamentals

Week 1: Programming fundamentals

This week we will be learning how to do some basic programming in Python. This will include things like variables, if statements, while loops and for loops. I chose to teach Python because as I said, it is supposedly easier for beginners, and because most people who answered the survey preferred Python.

If you already have a decent amount of experience programming, you will be given some problems to work on.

Problems

Easy:
https://dmoj.ca/problem/ccc18j1
https://dmoj.ca/problem/ccc14j1
https://dmoj.ca/problem/ccc12j2

Medium:
https://dmoj.ca/problem/ccc17j3
https://dmoj.ca/problem/ccc11s1
https://dmoj.ca/problem/ccc10j2

Hard:
https://dmoj.ca/problem/ccc14s3
https://dmoj.ca/problem/ccc17s3
https://dmoj.ca/problem/ccc12s5

Variables

Computers store data in variables. In Python,
`a = 5`
creates a variable named `a` and sets its value to 5.

We can print variables:

```
a = 5
print(a)
```

We can also create multiple variables and do math operations with them:

```
a = 3
b = a * 2
c = b + 5 * b
print(a, b, c)
```

Note that = in programming is not the same as = in math. In most programming languages, = is the assignment operator. For example, the following is valid Python:

```
a = 7
a = a + 2
print(a)
```

Operators in Python

`=` Assignment operator. Sets the variable on the left equal to the expression on the right.
`+` Addition operator. Adds together two numbers.
`-` Subtraction operator. Subtracts one number from an other.
`*` Multiplication operator. Multiplies two numbers together.
`/` Real division operator. Divides one number by another. The result has a decimal if needed.
`//` Floored division operator. Divides one number by another. The result is floored (rounded **down**).
`%` Modulo or remainder operator. Returns the remainder you would get when dividing the left number by the right.
`**` Exponentiation operator. Raises one number to the power of another.

Note that Python does the division operators a bit differently than some other languages. In C++ and Java, `/` is truncated division (rounds towards 0) when both inputs are integers and `/` only does real division when at least one of the operands is a floating point type. `//` is not an operator in C++/Java. `%` is also a bit different. Python's `%` always returns positive numbers, while C++/Java's `%` may return a negative number if one of the inputs is negative.

Reading input

```
n = int(input())
```

Will read a single line from the console, convert the value of that line to an integer, and set the value of variable n to it.

First exercise

Write a program that reads in 3 integers, your marks over 3 tests, (each on a new line) and outputs their average.

First exercise solution

```
a = int(input())
b = int(input())
c = int(input())

print((a + b + c) / 3)
```

If statements

An if statement only executes the code in its body if the condition is true.

```
a = int(input())

if a == 0:
      print(a, "is 0")
if a < 0:
      print(a, "is less than 0")
if a > 0:
      print(a, "is greater than 0")
```

If, elif, else

Here is a program that reads in the current temperature in Celsius, and outputs what state water would be in at that temperature.

```
tempr = int(input())

if tempr >= 100:
      print("It is very hot.")
      print("Water would boil at this temperature.")
elif tempr >= 0:
      print("The temperature is somewhat reasonable.")
      print("Water would turn to liquid.")
elif tempr >= -273:
      print("It is cold.")
      print("Water freezes at this point.")
else:
      print("This is impossible.")
```

Comparison and logical operators

`==` Returns true if the two operands are equal.
`!=` Returns true if the two operands are not equal.
`<` Returns true if the first operand is less than the second.
`>` Returns true if the first operand is greater than the second.
`<=` Returns true if the first operand is less than or equal to the second.
`>=` Returns true if the first operand is greater than or equal to the second.

`and` Returns true if both operands are true.
`or` Returns true if at least one of the operands is true.
`not` Returns true if its single operand is false.

Exercise 2

Write a program that reads in a single integer, and outputs fizz if it is divisible by 5, buzz if it is divisible by 7, fizzbuzz if it is divisible by both 5 and 7, and the number itself otherwise.

Hint: you can use the % operator to check whether one number is divisible by another.

Note: this question is a slightly easier version of the one on the survey. That question required you to do this for every number in between 1 and N.

Exercise 2 solution

```
n = int(input())

if n % 5 == 0 and n % 7 == 0:
     print("fizzbuzz")
elif n % 5 == 0:
     print("fizz")
elif n % 7 == 0:
     print("buzz")
else:
     print(n)
```

Booleans

A boolean variable can be either True or False.

```
a = True
b = False
c = 3 < 5

print(a)
print(b)
print(c)

if a or b:
     print("At least one of a and b are True")
```

While loop

A while loop keeps executing the code in its body while the condition is true.

```
n = int(input())

i = 0
while i < n:
    print(i)
    i = i + 1
```

More assignment operators

`a += b` is the same as `a = a + b`
`a -= b` is the same as `a = a - b`
`a *= b` is the same as `a = a * b`
`a /= b` is the same as `a = a / b`
`a //= b` is the same as `a = a // b`
`a %= b` is the same as `a = a % b`
`a **= b` is the same as `a = a ** b`

For example, `i = i + 1` can be rewritten as `i += 1`

Exercise 3

Write a program that reads in a single integer N, and outputs the first N perfect squares (starting from 1^2 and including N^2).

Exercise 3 solution

```
n = int(input())

i = 1
while i <= n:
     print(i*i)
     i += 1
```

Exercise 4

Write a program that reads in an integer and counts how many decimal digits it has.

Note: do not use strings.

Exercise 4 solution

```
n = int(input())

if n == 0:
    print(1)
else:
    digits = 0
    while n > 0:
        digits += 1
        n //= 10
    print(digits)
```

For loops

```
for i in range(10):
    print(i)
```

This would print all numbers between 0 inclusive and 10 exclusive.

In math and programming (when talking about programming, not in actual code), we write this range as [0, 10). Square brackets mean inclusive while parenthesis mean exclusive.

For loops

```
for i in range(3, 10):
    print(i)
```

Will print all the integers in [3, 10).

```
for i in range(3, 10, 2):
    print(i)
```

Will print integers in [3, 10), but instead of each number being 1 larger than the previous one, each number will be 2 larger than the previous one. For example, the above code will print the numbers 3, 5, 7, 9.

I do not have any fancy math notation for this :(

Exercise 5

Read in a single integer and output all of its factors in increasing order.

Exercise 5 solution

```
n = int(input())

for i in range(1, n+1):
     if n % i == 0:
         print(i)
```

Exercise 6

Write a program that reads in a single integer N, and for each integer `i` in [1, N], outputs `i is prime` if the number is prime and `i is not prime` otherwise.

Exercise 6 solution

```
n = int(input())

for i in range(1, n+1):
    isPrime = True

    for j in range(2, i):
        if i % j == 0:
            isPrime = False

    if i == 1:
        isPrime = False

    if isPrime:
        print(i, is "prime")
    else:
        print(i, "is not prime")
```

```
n = int(input())

for i in range(1, n+1):
    isPrime = i != 1

    j = 2
    while j*j <= i:
        if i % j == 0:
            isPrime = False
            break

    print(i, "is prime" if isPrime else "is not prime")
```

```
#include <bits/stdc++.h>
using namespace std;

int main(){

}
```

`#include <bits/stdc++.h>`
`using namespace std;`
Gives you access to all standard library functions.

C++ will start executing instructions you put in the `main` function.

Printing

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    cout << “Hello, World!” << endl;
}
```

Printing

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    cout << “Hello, World!” << endl;
    cout << “You can print “ << “multiple things ” << “like this.” << endl;
}
```

Note that you must put a semicolon at the end of each line that has a statement.

Variables

Computers store data in variables. In C++, you can declare variables in the following way:

```
int a = 5;
```

This creates a variable named a, which is an integer and has initial value 5.

We can print variables:

```
int a = 5;
cout << a << endl;
```

We can also perform operations on variables:

```
int a = 3;
int b = a * 2;
int c = a + b;
cout << b + 4 << “ “ << c - 5 << endl;
```