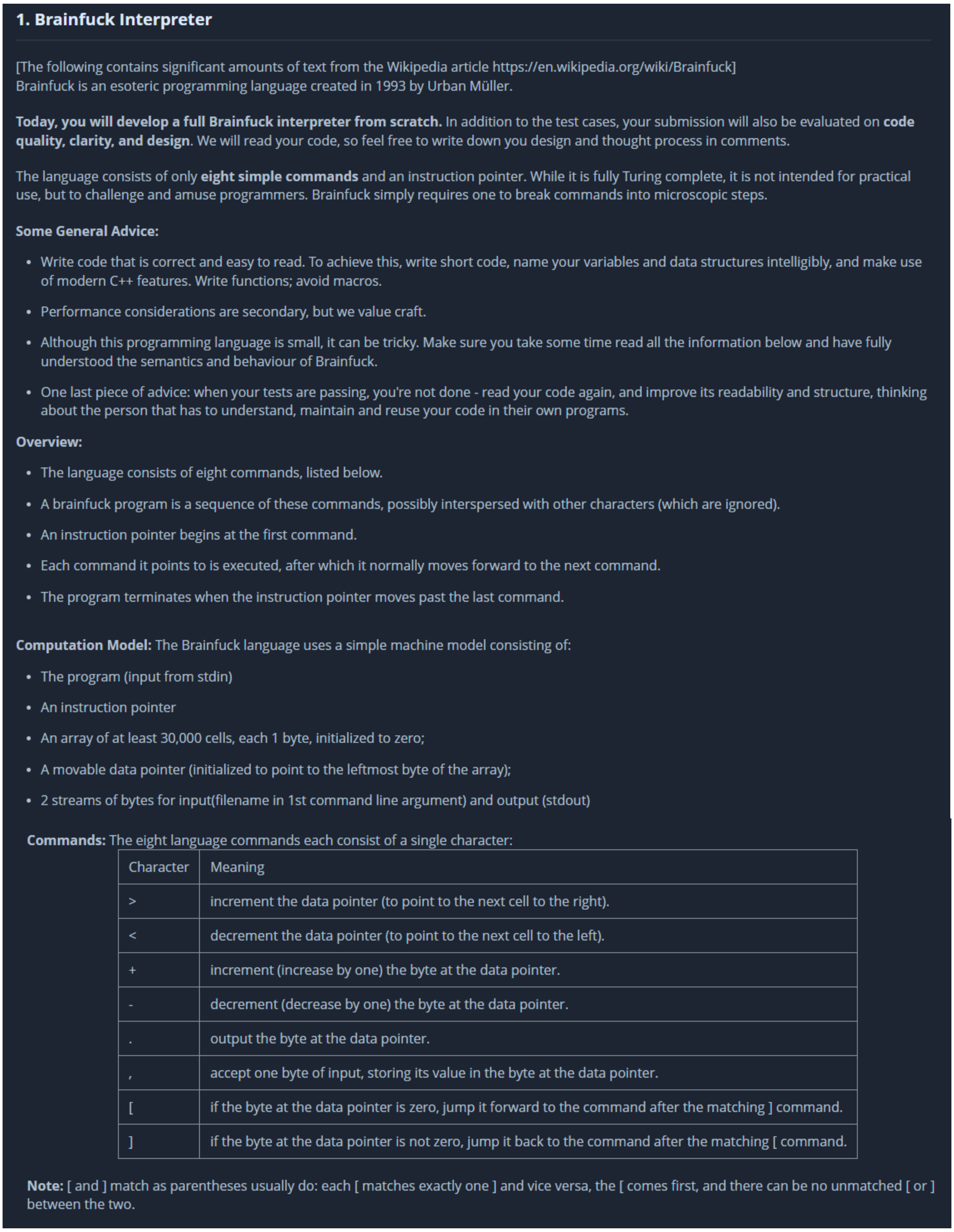
APT Portfolio *2022 Sep24*



1. Brainfuck Interpreter

[The following contains significant amounts of text from the Wikipedia article https://en.wikipedia.org/wiki/Brainfuck]
Brainfuck is an esoteric programming language created in 1993 by Urban Müller.

Today, you will develop a full Brainfuck interpreter from scratch. In addition to the test cases, your submission will also be evaluated on **code quality, clarity, and design**. We will read your code, so feel free to write down you design and thought process in comments.

The language consists of only **eight simple commands** and an instruction pointer. While it is fully Turing complete, it is not intended for practical use, but to challenge and amuse programmers. Brainfuck simply requires one to break commands into microscopic steps.

Some General Advice:

- Write code that is correct and easy to read. To achieve this, write short code, name your variables and data structures intelligibly, and make use of modern C++ features. Write functions; avoid macros.
- Performance considerations are secondary, but we value craft.
- Although this programming language is small, it can be tricky. Make sure you take some time read all the information below and have fully understood the semantics and behaviour of Brainfuck.
- One last piece of advice: when your tests are passing, you're not done - read your code again, and improve its readability and structure, thinking about the person that has to understand, maintain and reuse your code in their own programs.

Overview:

- The language consists of eight commands, listed below.
- A brainfuck program is a sequence of these commands, possibly interspersed with other characters (which are ignored).
- An instruction pointer begins at the first command.
- Each command it points to is executed, after which it normally moves forward to the next command.
- The program terminates when the instruction pointer moves past the last command.

Computation Model: The Brainfuck language uses a simple machine model consisting of:

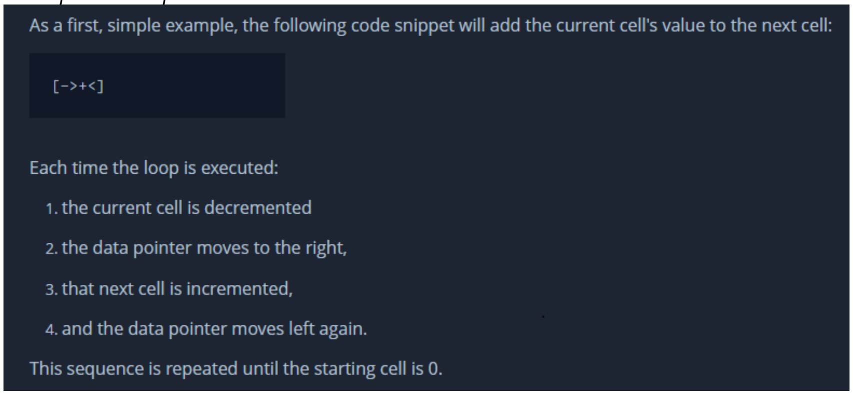
- The program (input from stdin)
- An instruction pointer
- An array of at least 30,000 cells, each 1 byte, initialized to zero;
- A movable data pointer (initialized to point to the leftmost byte of the array);
- 2 streams of bytes for input(filename in 1st command line argument) and output (stdout)

Commands: The eight language commands each consist of a single character:

Character	Meaning
>	increment the data pointer (to point to the next cell to the right).
<	decrement the data pointer (to point to the next cell to the left).
+	increment (increase by one) the byte at the data pointer.
-	decrement (decrease by one) the byte at the data pointer.
.	output the byte at the data pointer.
,	accept one byte of input, storing its value in the byte at the data pointer.
[	if the byte at the data pointer is zero, jump it forward to the command after the matching] command.
]	if the byte at the data pointer is not zero, jump it back to the command after the matching [command.

Note: [and] match as parentheses usually do: each [matches exactly one] and vice versa, the [comes first, and there can be no unmatched [or] between the two.

Example 1 : Simple



As a first, simple example, the following code snippet will add the current cell's value to the next cell:

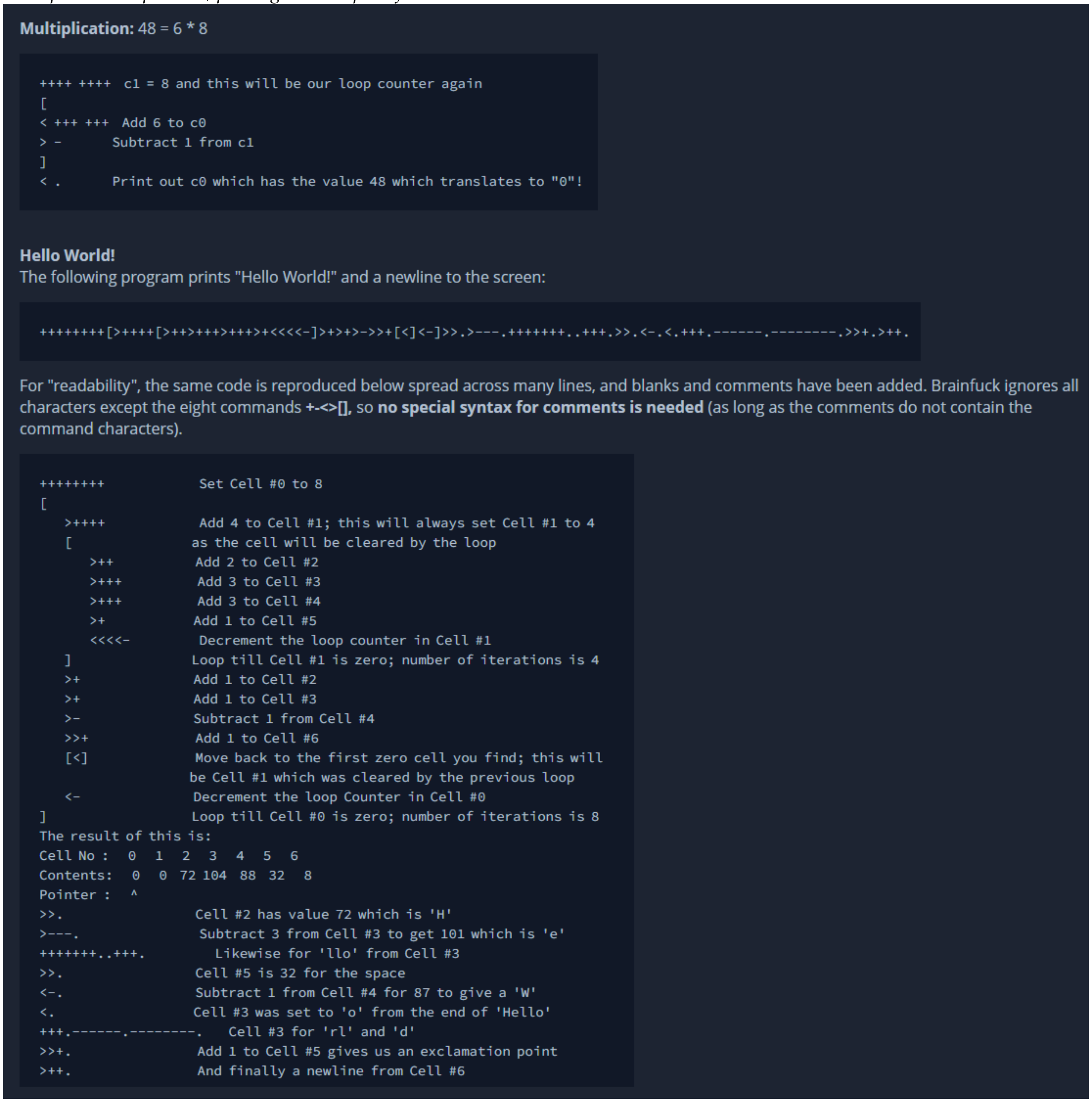
```
[->+<]
```

Each time the loop is executed:

1. the current cell is decremented
2. the data pointer moves to the right,
3. that next cell is incremented,
4. and the data pointer moves left again.

This sequence is repeated until the starting cell is 0.

Example 3 : Complicated, please goto example 2 first



Multiplication: 48 = 6 * 8

```
++++ ++++  c1 = 8 and this will be our loop counter again
[
< +++ +++  Add 6 to c0
> -      Subtract 1 from c1
]
< .      Print out c0 which has the value 48 which translates to "0"!
```

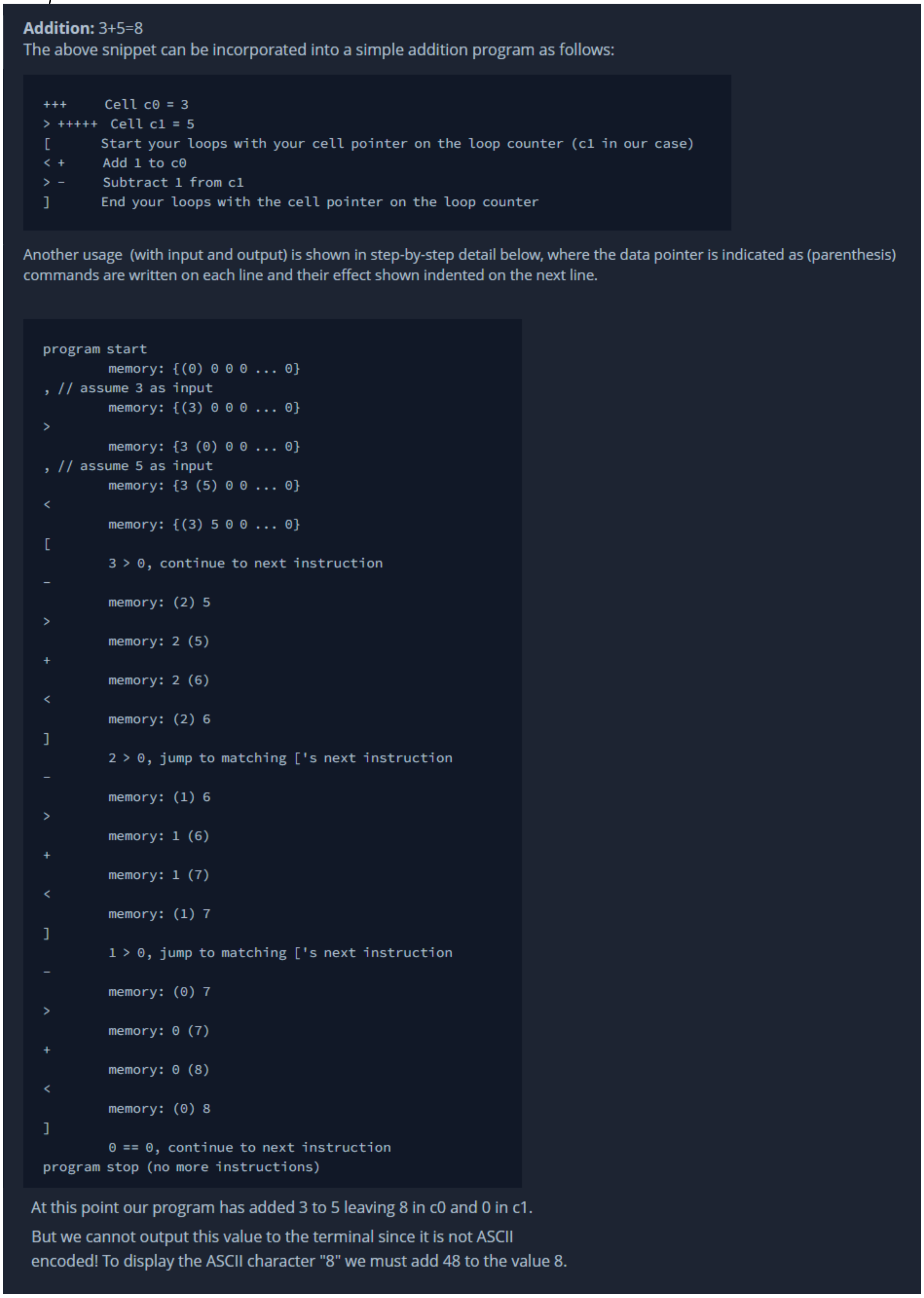
Hello World!
The following program prints "Hello World!" and a newline to the screen:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>++.
```

For "readability", the same code is reproduced below spread across many lines, and blanks and comments have been added. Brainfuck ignores all characters except the eight commands **+-<>[]**, so **no special syntax for comments is needed** (as long as the comments do not contain the command characters).

```
++++++++              Set Cell #0 to 8
[
    >++++             Add 4 to Cell #1; this will always set Cell #1 to 4
    [                 as the cell will be cleared by the loop
        >++           Add 2 to Cell #2
        >+++          Add 3 to Cell #3
        >+++          Add 3 to Cell #4
        >+            Add 1 to Cell #5
        <<<<-         Decrement the loop counter in Cell #1
    ]                 Loop till Cell #1 is zero; number of iterations is 4
    >+                Add 1 to Cell #2
    >+                Add 1 to Cell #3
    >-                Subtract 1 from Cell #4
    >>+               Add 1 to Cell #6
    [<]               Move back to the first zero cell you find; this will
                      be Cell #1 which was cleared by the previous loop
    <-                Decrement the loop Counter in Cell #0
]                     Loop till Cell #0 is zero; number of iterations is 8
The result of this is:
Cell No :   0   1   2   3   4   5   6
Contents:   0   0  72 104  88  32   8
Pointer :   ^
>>.                   Cell #2 has value 72 which is 'H'
>---.                 Subtract 3 from Cell #3 to get 101 which is 'e'
+++++++..+++.           Likewise for 'llo' from Cell #3
>>.                   Cell #5 is 32 for the space
<-.                   Subtract 1 from Cell #4 for 87 to give a 'W'
<.                    Cell #3 was set to 'o' from the end of 'Hello'
+++.------.--------.    Cell #3 for 'rl' and 'd'
>>+.                  Add 1 to Cell #5 gives us an exclamation point
>++.                  And finally a newline from Cell #6
```

Example 2 : Medium



Addition: 3+5=8
The above snippet can be incorporated into a simple addition program as follows:

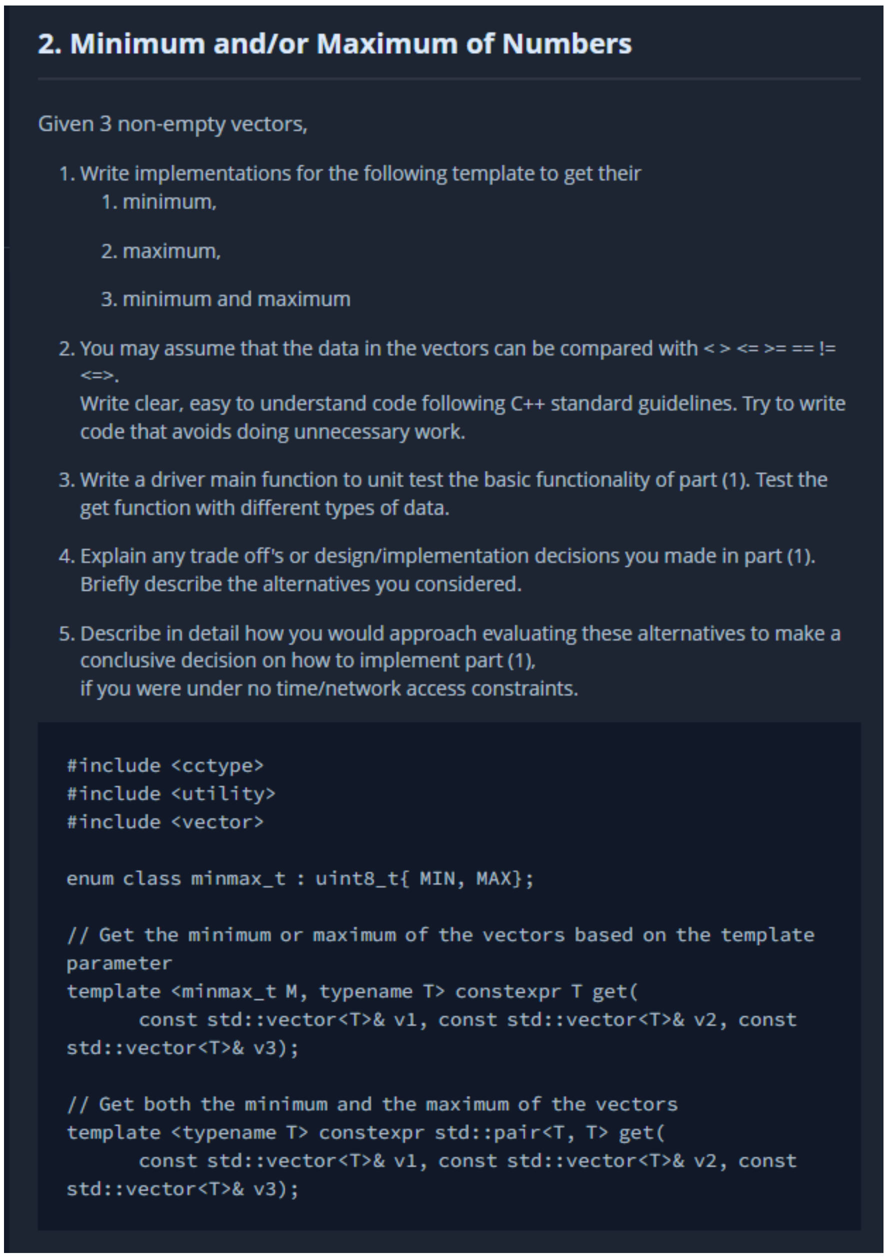
```
+++     Cell c0 = 3
> +++++  Cell c1 = 5
[       Start your loops with your cell pointer on the loop counter (c1 in our case)
< +     Add 1 to c0
> -     Subtract 1 from c1
]       End your loops with the cell pointer on the loop counter
```

Another usage (with input and output) is shown in step-by-step detail below, where the data pointer is indicated as (parenthesis) commands are written on each line and their effect shown indented on the next line.

```
program start
         memory: {(0) 0 0 0 ... 0}
, // assume 3 as input
         memory: {(3) 0 0 0 ... 0}
>

         memory: {3 (0) 0 0 ... 0}
, // assume 5 as input
         memory: {3 (5) 0 0 ... 0}
<

         memory: {(3) 5 0 0 ... 0}
[

         3 > 0, continue to next instruction
-

         memory: (2) 5
>

         memory: 2 (5)
+

         memory: 2 (6)
<

         memory: (2) 6
]

         2 > 0, jump to matching ['s next instruction
-

         memory: (1) 6
>

         memory: 1 (6)
+

         memory: 1 (7)
<

         memory: (1) 7
]

         1 > 0, jump to matching ['s next instruction
-

         memory: (0) 7
>

         memory: 0 (7)
+

         memory: 0 (8)
<

         memory: (0) 8
]

         0 == 0, continue to next instruction
program stop (no more instructions)
```

At this point our program has added 3 to 5 leaving 8 in c0 and 0 in c1.

But we cannot output this value to the terminal since it is not ASCII encoded! To display the ASCII character "8" we must add 48 to the value 8.

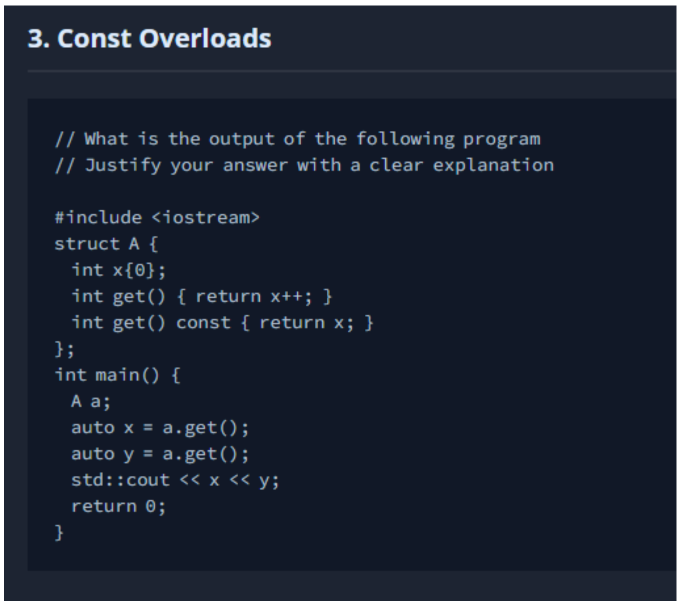


2. Minimum and/or Maximum of Numbers

Given 3 non-empty vectors,

1. Write implementations for the following template to get their
    1. minimum,
    2. maximum,
    3. minimum and maximum
2. You may assume that the data in the vectors can be compared with < > <= >= == != <=>.
   Write clear, easy to understand code following C++ standard guidelines. Try to write code that avoids doing unnecessary work.
3. Write a driver main function to unit test the basic functionality of part (1). Test the get function with different types of data.
4. Explain any trade off's or design/implementation decisions you made in part (1). Briefly describe the alternatives you considered.
5. Describe in detail how you would approach evaluating these alternatives to make a conclusive decision on how to implement part (1),
   if you were under no time/network access constraints.

```
#include <cctype>
#include <utility>
#include <vector>

enum class minmax_t : uint8_t{ MIN, MAX};

// Get the minimum or maximum of the vectors based on the template
parameter
template <minmax_t M, typename T> constexpr T get(
        const std::vector<T>& v1, const std::vector<T>& v2, const
std::vector<T>& v3);

// Get both the minimum and the maximum of the vectors
template <typename T> constexpr std::pair<T, T> get(
        const std::vector<T>& v1, const std::vector<T>& v2, const
std::vector<T>& v3);
```



3. Const Overloads

```
// What is the output of the following program
// Justify your answer with a clear explanation

#include <iostream>
struct A {
  int x{0};
  int get() { return x++; }
  int get() const { return x; }
};
int main() {
  A a;
  auto x = a.get();
  auto y = a.get();
  std::cout << x << y;
  return 0;
}
```

4. Real Number Comparison

```
// What is the output of the following program
// Justify your answer with a clear explanation
// Will the output change if x and y are declared as constexpr

#include <iostream>

void F1() {
     double x = 1.2 / 8.0;
     double y = (32 / 256.0) * 1.2;
     if (x == y) {
             std::cout << "F1: Yes they are equal"
     } else {
             std::cout << "F1: No they are not"
     }
}

void F2() {
     double x = 1.2 / 7;
     double y = (21.0 / 147) * 1.2;
     if (x == y) {
             std::cout << "F2: Yes they are equal"
     } else {
             std::cout << "F2: No they are not"
     }
}

void F3() {
     double x = 1.2 / 7;
     double y = (32.0 / 224) * 1.2;
     if (x == y) {
             std::cout << "F3: Yes they are equal"
     } else {
             std::cout << "F3: No they are not"
     }
}

int main() {
  F1();
  F2();
  F3();
  return 0;
}
```

5. Threads and Exceptions

```
// What are the possible outputs of the following program
// Be specific about the exact wording of the output
// Justify your answer with a clear explanation

#include <iostream>
#include <thread>
#include <mutex>
#include <unistd.h>
#include <exception>

std::mutex mutex1, mutex2;

void ThreadA()
{
    mutex2.lock();
    throw std::runtime_error("Exception in ThreadA");
    mutex1.lock();
    mutex2.unlock();
    mutex1.unlock();
}
void ThreadB()
{
    mutex1.lock();
    throw std::runtime_error("Exception in ThreadB");
    mutex2.lock();
    mutex1.unlock();
    mutex2.unlock();
}
int main()
{
  try {
    std::thread t1( ThreadA );
    std::thread t2( ThreadB );
    t1.join();
    t2.join();
  } catch (const std::exception& ex) {
    std::cout << "Caught in main:" << ex.what() << "\n";
  }
  return 0;
}
```