

# CS1020E: DATA STRUCTURES AND ALGORITHMS I

## Lab 1 Ex 1 – Input Output (Week 3, starting 22 August 2016)

This lab allows you to get some practice in handling the reading of input and printing of output. Please complete this, and move on to attempt Lab 1 Ex 2, **BEFORE** coming for the tutorial+lab session.

### ***Input and Output Streams***

After coding a program, you should allow the program to be fed some input data so that it can be tested. This input data can either be read from one or more *files*, or from the **standard input** (known as `stdin` in unix). The standard input is a stream that allows your program to read data, usually from the keyboard. In C++, the standard input stream is represented by the `istream` typed variable `cin`.

Similarly, your output can either be written to a file, or inserted into the **standard output** (`stdout` in unix). The characters inserted into the standard output appear on your screen when you run the program `a.out`. In C++, the standard output stream is represented by the `ostream` typed variable `cout`.

### ***Our Test Environment***

For CS1020E labs and PE, **DO NOT read from, or write to, files**. **Read from `stdin` and write to `stdout`**. This is because **CodeCrunch will do the job of redirecting** data from test files to `stdin`, and redirecting `stdout` to an output file. It will then compare your program's output with the test data `output`.

**Test your programs locally** BEFORE submitting it to CodeCrunch. Say you have these files:

- `input_output.cpp` source file (your program's code)
- `io1.in, io2.in, io3.in, ...` test input data files
- `io1.out, io2.out, io3.out, ...` expected test output data files

What steps should you take?

1. Compile the program  
`g++ -std=c++11 input_output.cpp`
2. Run the program to check if it works, and if the output format is correct  
`a.out`
3. Run the program against test data redirected from `stdin`  
Examine if some other test cases result in runtime errors  
`a.out < io1.in`
4. Run the program against test data, redirecting `stdout` to a file  
`a.out < io1.in > io1.actual`
5. Compare your program's actual output with the given expected output file  
`diff io1.out io1.actual`
6. Repeat steps 4&5 with other test cases

## Reading / Writing to Input / Output Streams

In lectures, you have learnt the extraction and insertion operators, for reading and writing respectively:

- >> Extraction from istream  
`cin >> intA >> lngB >> dblC;` from <iostream>  
or  
`scanf("%d %ld %lf", &intA, &lngB, &dblC);` C-styled io from <stdio>
- << Insertion to ostream  
`cout << intA << lngB << dblC;` from <iostream>  
or  
`printf("%d %ld %lf", intA, lngB, dblC);` C-styled io from <stdio>

## Running Time

Some problems have **hidden test cases** in CodeCrunch to **test the efficiency** of your algorithm. After a few seconds, CodeCrunch will kill the execution of your program as the time limit has been exceeded (**TLE**).

When reading or writing a large number of lines, running time may be affected by your handling of I/O.

Solutions:

- Use `scanf()` and `printf()` as much as possible  
or
- Completely avoid `scanf()` and `printf()`, using **stream extraction and insertion exclusively**  
Before any input/output, add these two lines of code<sup>1</sup> to your program  
`ios_base::sync_with_stdio(false);`  
`cin.tie(NULL);`  
Use “\n” newline character instead of the `endl` manipulator  
`cout << "Newline after this \n";`

If you use the second method, the output inserted into `cout` may not appear immediately, but only at the end of your program (or when you insert `endl` to `cout`).

## Whitespaces

Since we are comparing the output files **character by character**, the data must **match exactly**. Take care NOT to append additional spaces to the end of any line.

In unix, a line ends with the newline character “\n”. You should therefore ensure that there is a “\n” character at the **end of the last line**.

---

<sup>1</sup> Taken from <http://codeforces.com/blog/entry/5217>

## Input Format

For this module, a program's test input will generally fall under 3 categories:

1. Read N items, given an integer N
2. Read till the item is X, given X (X is known as a sentinel value)
3. Read till end of file (EOF)

After this lab, make sure you are able to handle these 3 types of input format proficiently.

A typical problem looks like the grey box below:

### Problem



40%

Bugs bunny plans to run along a straight path. At  $1 \leq N \leq 500,000$  points along the path, there are beautiful landmarks to either side of the path. **At each** landmark  $x$ , help bugs bunny to calculate the distance  $d_x$  he would have travelled from the start point till that landmark.

You can be assured that all input is valid, each landmark is further away from the start point than the previous, and that any  $d_x$  fits within a 32-bit signed integer.

For practice, the *first line* contains one of {1, 2, 3}, denoting the type of input format given above.

### Format 1 – Read N Items

The *second line* contains  $N$ , the number of points along the path

The *third line* contains  $N$  integers

On that line, the first integer is the distance between the start point and the first landmark

The second integer is the distance between the first and second landmark

...

The  $N^{\text{th}}$  integer contains the distance between the  $(N-1)^{\text{th}}$  and  $N^{\text{th}}$  landmark.

#### Sample Input

1

4

10 10 2 10

#### Sample Output

10 20 22 32

### **Format 2 – Read till -1**

The *second line* contains a number of integers, the last being -1.

On that line, the first integer is the distance between the start point and the first landmark

The second integer is the distance between the first and second landmark

...

The  $N^{\text{th}}$  integer contains the distance between the  $(N-1)^{\text{th}}$  and  $N^{\text{th}}$  landmark.

#### **Sample Input**

```
2
```

```
10 10 2 10 -1
```

#### **Sample Output**

```
10 20 22 32
```

Don't forget that the  $(N+1)^{\text{th}}$  integer is -1, the sentinel, and it is NOT a landmark

### **Format 3 – Read till EOF**

The *second line* contains the **N** integers.

On that line, the first integer is the distance between the start point and the first landmark

The second integer is the distance between the first and second landmark

...

The  $N^{\text{th}}$  integer contains the distance between the  $(N-1)^{\text{th}}$  and  $N^{\text{th}}$  landmark.

#### **Sample Input**

```
3
```

```
10 10 2 10
```

#### **Sample Output**

```
10 20 22 32
```

When typing input into the keyboard, press **Ctrl+D** to manually add the **EOF character**.

When checking for EOF, `cin >> someVariable` will evaluate to false on EOF.

### **Submission**

Your source file should be named `input_output.cpp`

### **Finally**

You may use the skeleton file to help you, but you may delete any part that you find not useful.

Read all instructions carefully! Some problems may have certain restrictions.

- End of Lab 1 Ex 1 -