Contents

1 Assignment 1: Integers **1**
1.1 Assignment Goals 1
1.2 Due Date 1
1.3 Submission Link 1

2 Overview **2**
2.1 Python integers 2
2.2 C++ integers 2

3 Part A: Integer Limit Calculations **2**

4 Part B: Demonstration of Python integers **3**

5 Part C: Demonstration of C++ integers. **3**
5.1 Measuring execution time. 4

1 Assignment 1: Integers

1.1 Assignment Goals

The first assignment goals are to ensure

- you have the development environments available to you
- you have signed up for the assignment submission system
- you can write simple C++ and Python programs
- you understand binary, bytes, and basic principles of integer mathematics.

1.2 Due Date

This assignment is due 2016-09-12 at midnight

1.3 Submission Link

You can submit here: week 1 submit link

2 Overview

2.1 Python integers

In Python, there is one base type for integers: `int`. It can represent any positive or negative integer no matter how large.

Here is an example:

```
>>>bignum=2**(2**30)-1
>>>print(bignum.bit_length()/8/1e6)
134.217728
```

The number `bignum` requires 134 MB (megabytes) of storage.

Do not try to print `bignum`, it will take too long. However, we can use `bin(bignum)` to take a look at its representation:

```
>>>bin(bignum)[:10]
'0b11111111'
```

2.2 C++ integers

In C++, there are many flavors of integer. The basic one is `int`, but there are also `long` and `short` integers, and signed and unsigned integers.

Read the following two documents:

- Fundamental types (at cppreference) and
- Variables and types (at cplusplus)

3 Part A: Integer Limit Calculations

Using python, calculate and print a table for the capability of integers using 1, 2, 4, and 8 bytes of storage

Use the following format string:

```
Table = "{:<6} {:<22} {:<22} {:<22}"
```

to print both the header and the data. This string can be used to create a format: the braces `{}` indicate fields, `<` means left justify, and the number indicates the width of the field to use for this data.

The command to print the header is:

```
print(Table.format('Bytes','Largest Unsigned Int','Minimum Signed Int','Maximum Signed Int')
```

The first two lines of the table should be

Bytes	Largest Unsigned Int	Minimum Signed Int	Maximum Signed Int
1	255	-128	127
2	65535	-32768	32767

The filename of the program submitted must be `w1a_limits.py`

4 Part B: Demonstration of Python integers

Write a python program that does the following:

- reads two integers X and Y using `input()`
- calculates $Z = X!$ - $Y!$

Your program should print out Z, the number of decimal digits of Z, and the number of bytes that are required to store this number

You may use `math.factorial()` to check your answer but you must calculate the factorial yourself.

Here is an example output when X=11 and Y=7

```
39911760
8
4
```

This means that $Z = 11!$-$7! = 39911760$, which clearly has 8 decimal digits, and it requires 4 bytes of storage.

Your program must be submitted as `w1b_factdiff.py`

5 Part C: Demonstration of C++ integers.

Consider the following code segment:

```
short unsigned int m=1;

while (m>0)
    m++;
```

Although *logically*, this is an infinite loop, in practice what happens is that eventually `m` will be represented in memory (binary) as all ones, i.e. it will be `0b111...111`. When it is incremented again, the result is `0b1000...000` but the `1` does not fit into the storage allocated for `m` and so `m` will become zero. This is called "wrap around".

Write a C++ program that will:

- measure how long an `short unsigned int` takes to "wrap around" from a starting value of 1
- measure how long an `unsigned int` takes to "wrap around" from a starting value of 1
- estimates how long a `long unsigned int` takes to "wrap around" from a starting value of 1

Your program should print out the following:

```
short unsigned int time (microseconds): 1290.3
unsigned int time (seconds): 2.3
long unsigned int time (years): 1.201
```

except the numbers should be calculated by your code. The measured times will vary: this is natural and ok. Do not do any averaging.

This program will take approximately 2.3 seconds to run (the time for `int` to wraparound), since the time for `short unsigned int` to wrap around is very small and the time for `long unsigned int` is only an estimate.

Note that the units must match the ones shown: microseconds, seconds, and years.

The filename of the program submitted must be `w1c_timing.cpp`

5.1 Measuring execution time.

The following C++ program is provided as a example of a simple method for measuring how long a code segment executes. In the example, we count up to one billion. Note that the code must be compiled using the C++14 standard.

You may re-use this code for your assignment.

```
// Timing Code
//
// This is a simple example of using the clock() function of <ctime>
// to measure how long a code block took to run.
//

#include <iostream>
#include <ctime>

using namespace std;

int main()
{

    clock_t start_clock,end_clock;
```

```
    start_clock = clock();  // Timing starts here

    int i = 0;

    while ( i < 1'000'000'000 )
    {
         i++;
    }

    end_clock = clock();    // Timing stops here


    double seconds = (double)(end_clock-start_clock) / CLOCKS_PER_SEC;


    cout << "counting to one billion took " << seconds << " seconds" << endl;

}
```

The code is also available here: timed_example.cpp