2G03 Homework 3, 2020

Due end of day, Tuesday Oct 6th (suggested) or Thursday Oct 8th (hard deadline)

All code must be present on phys-ugrad in your **HW3** directory.

1 Summing numbers

1.1 A naive approach

To help your understanding of loops let's start with a simple example. We're going to think about a program that adds all the numbers from 1 to 10. A naive way of doing this can be seen below, **which you should write out yourself (name it `naivesum.cpp`)**

```
#include <iostream>

using namespace std;

int main()
{
  int total = 0;
  total = total + 1;  // total = 1
  total = total + 2;  // total = 3
  total = total + 3;  // total = 6
  total = total + 4;  // total = 10
  total = total + 5;  // total = 15
  total = total + 6;  // total = 21
  total = total + 7;  // total = 28
  total = total + 8;  // total = 36
  total = total + 9;  // total = 45
  total = total + 10; // total = 55

  cout << total << endl;
  return 0;
}
```

A quick note here: when we write `total = total + 1`, the variable `total` will store whatever the value of `total` was previously plus 1, if this is confusing I've put the value that total is equal to after each line of code has executed as a comment to make it explicit.

Run this program and verify the final answer is 55.

1.2 Using a for-loop

Clearly our repetitive technique is going to exhaust itself quickly. We have no hope of writing a program that adds all the numbers from, say, 1 to 1000, in this fashion, let alone integrating

a function using approximate sums (e.g. Midpoint rule). This is where the loop structure, and in particular the for-loop structure, comes to the rescue. A for-loop (loop in general), tells the computer to repeat a set of instructions over and over until a stopping condition is met. First we'll look at how to rewrite the above code with a for-loop, and then we'll sort out just how the for-loop syntax works. **Copy the code below into a file called `forsum.cpp`. Run this more elegant version and verify the sum is still 55.**

```
#include <iostream>

using namespace std;

int main()
{
  int total = 0;

  for(int i=1; i<=10; i++)
  {
    total = total + i;
  }

  cout << total << endl;
  return 0;
}
```

Sometimes when we're learning to code it's a bit difficult to really see what the for-loop is doing. It's helpful to print out the counter variable to the terminal so we can see the inner workings of the program explicity. **Modify the above code as follows**:

```
#include <iostream>

using namespace std;

int main()
{
  int total = 0;

  for(int i=1; i<=10; i++)
  {
    total = total + i;
    cout << "i = " << i << endl;
    cout << "total = " << total << endl;
  }

  cout << total << endl;
```

```
    return 0;
}
```

Compile and run this version of the program. Record what you see on the terminal. Explain the output.

1.3 User defined sum

You can use cin to allow the use to input the number to count up to. Use **cp** to copy forsum.cpp to a new source file `usersum.cpp`. The required new code is as follows:

1. Add an additional int variable called `n` to store this.
2. Add a line using cout to ask the user what number to sum to
3. Add a line using cin to read the number into n
4. Change the `for` loop to count up to `n` instead of 10.
5. For the final output with cout, indicate the value of n used and that it is the final sum.

You can look at lecture examples like `calc.cpp` or the code in section 2 to see what the added code might look like. Once it is working, you may way to remove the extra output inside the loop.

Compile and run your program and record the final output for n=101.

2 If Statements

Let's get some practice with if statements. **Copy the following code into a file called** `compare.cpp` ,

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;

    cout << "Please enter two integers." << endl;
    cin >> a;
    cin >> b;

    if(a>b) cout << "The first is larger." << endl;
    else    cout << "The first is not larger." << endl;

    return 0;
}
```

Write a program called `even.cpp`. The program should print out text to prompt the user, take an integer using `cin`, and print "True" if the number is even and "False" if the number is odd.

Hint: the modulo operator `%` can be used to calculate the remainder of dividing an integer by another integer.

3 Practice

Write a program to determine whether or not an integer entered by a user is prime.

Your source code file should be isprime.cpp. Your program should print a prompt to the terminal, accepts an integer from the user, and then print out the word "True" if the integer is prime, and "False" if the integer is not.

Hint: You need to use an if statement and a for loop in this program. A prime number is a natural number (integer) greater than 1 that is not a product of two smaller natural numbers. A prime numbers can only be divided exactly by itself and 1 so your experience with even should help. You should think about the definition of a prime number and how you can use a loop to test if this is true. Also consider what the ending condition should be and how an `if` statement can provide an addition exit for a loop.

Compile and run your program and record the result for the numbers from 1 to 10

Hint: If your program is not doing the correct thing, use the printing values with cout strategy to see what is happening similar to 1.2.