COMP 211: Principles of Imperative Computation, Spring 2014
Programming 1: Basics

Due: Wednesday, 10 Sep, 6pm

1 Mechanics

You should write your code for this assignment in a file named `prog01.c0`. We recommend storing this file in the following location:

- Windows: open a new cygwin shell and type

  ```
  $ mkdir comp211
  ```

  (Recall from lab that you are not supposed to type the `$`; it is an indication that you should do this in the Unix shell instead of in coin.) This will create a directory located at `C:\cygwin\YOURNAME\comp211`, and you can use emacs/sublime to edit files in there.

- Mac: open Terminal and type

  ```
  $ mkdir comp211
  ```

  (Recall from lab that you are not supposed to type the `$`; it is an indication that you should do this in the Unix shell instead of in coin.) This will create a directory located at `/Users/YOURNAME/comp211/`, and you can use emacs/sublime to edit files in there.

Next, you should use sublime or emacs to create and save a file `prog01.c0` in that directory.

Once you have created `prog01.c0`, to load the file in `coin`, use the terminal (Unix shell) to start coin as follows:

```
coin -d -l string prog01.c0
```

This assumes you are in the directory where `prog01.c0` is located. To get there, assuming you put it in the location suggested above, start a new terminal/cygwin shell and then do

```
cd comp211
```

Each time you make changes to `prog01.c0` in your text editor, you need to reload it in coin. You can do this by typing `#reload`.

To hand in this part of the homework, upload a file named `prog01.c0` to your handin directory on WesFiles, which is located at

```
/courses/COMP-211-01-dlicata/handin/YOURNAME/
```

Ask the course staff if you have trouble doing this.

2 Functions

2.1 Example

As an example of what you'll do below, suppose we want to write a function named `add3` that is given three numbers (each of type `int`) and returns the sum of the three numbers (also of type `int`).

This function would start as follows:

```
int add3(int x, int y, int z) {
  ... code goes here ...
}
```

`add3` is the name of the function. `x` and `y` and `z` are variables which stand for the three arguments to the function. The annotations `int x`, `int y`, and `int z` specify that each argument has type `int`. The annotation `int add3` specifies that the result of the function is supposed to be an integer.

Next, we can fill in the body of the function, to say that the value it returns is the sum of the three numbers:

```
int add3(int x, int y, int z) {
   return (x + y + z);
}
```

Task 1 *Copy this function definition into your* `prog01.c0` *file.*

To use a 3-argument function, you write the function name and then write the three things you want to plug in for the arguments, separated by commas. For example, writing `add3(3,4,5)` will add the numbers 3 4 and 5: evaluating this expression is like evaluating the sequence of commands

```
int x = 3;
int y = 4;
int z = 5;
return (x + y + z);
```

and in the memory that is created by the first three lines, `x + y + z` evalutes to `12`.

Task 2 *Load your* `prog01.c0` *file into coin (see the directions under mechanics above) and then in coin run* `add3(3,4,5)`*, and check that it returns 12. I.e. enter (as usual, don't type the* `--->`*; that is telling you to type it into coin as opposed to the Unix shell)*

```
--> add3(3,4,5)
```

and you should see

```
12 (int)
```

For each of the problems below, you should write the code in your `prog01.c0` file and then test it in coin like you just did for this one.

2.2 Problems

Task 3 (2 points) *Write a function named* `polynomial` *that is given an argument x of type* `int` *and returns the value of the polynomial $x^2 + 2x + 1$ (also of type* `int`*).* For example, when you test it, you should see

```
--> polynomial(3);
16 (int)
```

The function defintion should start as follows:

```
int polynomial(int x) {
   ... code goes here ...
}
```

Task 4 (2 points) *Write a function* `between` *that takes three* `int` *arguments named* `lower` *and* `middle` *and* `upper` *and returns a* `bool`*. The return value should be* `true` *when it is the case both that* `middle` *is bigger than or equal to* `lower`*, and that* `middle` *is (strictly) smaller than* `upper`*. That is, when* `middle` *is between* `lower` *(inclusive) and* `upper` *(exclusive). The return value should be* `false` *otherwise.* For example, when you test it, you should see

```
--> between(1,2,3);
true (bool)
--> between(1,1,3);
true (bool)
--> between(1,2,2);
false (bool)
--> between(1,0,2);
false (bool)
--> between(0,3,2);
false (bool)
```

Hint: use `n < m` (or `n <= m`) to check whether `n` is smaller than (or equal to) `m`.

Task 5 (3 points) *Write a function* `is_letter` *that takes a character* `c` *and returns a* `bool`*, which is true when* `c` *is either a lowercase letter a,b,...,z or an uppercase letter A,B,...,Z; and which is false otherwise.* For example,

```
is_letter('a') == true
is_letter(';') == false
```

The function should start as follows:

```
bool is_letter(char c) {
   ...
}
```

Hint: use `c1 <= c2` to test whether the character `c1` is less than or equal to `c2` alphabetically. E.g. `'a' <= 'b'` is true, but `'b' <= 'a'` is false.

Task 6 (3 points) *Write a function* `palindrome` *that is given a* `string` *argument and returns a* `bool`*, which is true when the given string is a palindrome (the same forwards as backwards). You can assume the argument string has exactly 5 letters.* For example, when you test it, you should see

```
--> palindrome("radar");
true (bool)
--> palindrome("radrr");
false (bool)
```

Hint: use the function `string_charat(s,n)` to get the `nth` character of the string `s`, where the first letter is at position `0`—e.g.

```
--> string_charat("radar",0);
'r' (char)
--> string_charat("radar",1);
'a' (char)
```

Bonus task (not graded): make palindrome work for a string with any number of letters. Hint: use the function `int string_length(string s)` to get the number of characters in a string.