CMPS 12A
Introduction to Programming
Lab Assignment 7

The purpose of this assignment is to introduce the C programming language, including standard input-output functions. Begin by reading the first six chapters "C Programming Notes" by Steve Summit which can be found at

http://www.eskimo.com/~scs/cclass/notes/top.html

Six chapters may sound like a lot, but much of it can be skimmed, since it is largely a review of topics already covered. You will find that C is very similar to Java when it comes to things like arithmetic expressions, control structures, and function calls. One area where the two languages differ significantly is in the way they do input and output. The best place to start is the HelloWorld program:

```
/*  HelloWorld.c   */
#include<stdio.h>

int main(){
   printf("Hello, World!\n");
   return 0;
}
```

Copy this program to a file called `HelloWorld.c`, then compile it by doing

```
% gcc -o HelloWorld HelloWorld.c
```

The Unix command `gcc` invokes the *Gnu C Compiler*, which is an open source C compiler freely available for most platforms. The `-o` option to `gcc` tells the compiler what to call the executable object file, in this case `HelloWorld`. If you leave out the option `-o HelloWorld`, the executable will be called `a.out`. The last item on the line is the source code file name, i.e. `HelloWorld.c`. List the contents of your current directory and you should see a new executable file called `HelloWorld`. Run it by typing the name of the file at the command line as follows:

```
% HelloWorld
Hello, World!
%
```

Recall that to run a Java program we needed to type: `java program_name` (unless we've placed the Java program in an executable jar file, as in a previous lab.) As we see, to run a C program, one types the name of the executable file. Why this difference? Java programs are executed by a "simulation" of a computer called the *Java Virtual Machine* (JVM). In a Unix environment, the JVM is invoked by the command `java`. Thus `java program_name` runs the JVM with one command line argument, i.e. the name of the `.class` file containing function `main()` for the program. This additional layer of software abstraction is not present for C programs since there is no simulation - you run the executable binary file directly on the computer hardware.

The first thing to notice in the source file `HelloWorld.c` is that there is no `class` keyword. Java is known as an Object Oriented Programming (OOP) language, meaning that data structures and the procedures that operate on them are grouped together into one language construct, called the *class*.

Common behavior amongst classes is specified explicitly through the mechanism of inheritance. The C language however does not directly support OOP, although OOP can be implemented with some effort. C is known as a procedural programming language, which means that data structures and functions are separate language constructs. There are no classes and no inheritance. Most control structures such as loops (while, do-while, for), branching (if, if-else, switch), and function calls are virtually identical in the two languages.

Comments in C are specified by bracketing them between the symbols `/*` and `*/`, and may span several lines. For instance

```
/* comment */
```

or

```
/*
   comment
   comment
*/
```

or

```
/*
*  comment
*  comment
*  comment
*/
```

are all acceptable styles. Note that the single line `// comment` familiar from Java, is not recognized by all C compilers, although the `gcc` compiler does accept this form of comment by default.

Any line beginning with # is a *preprocessor directive*. These directives, which are literal text substitutions, are executed during the first phase of compilation. Commands that start with `#include` are roughly equivalent to `import` statements in Java. The line `#include<stdio.h>` inserts the library header file `stdio.h` into the program. The commands in `stdio.h` specify functions for performing standard input-output operations.

A C program must contain exactly one function called `main()`, which is the entry point for program execution. The general skeleton for `main()` is

```
int main(){
   /* variable declarations */
   /* executable statements */
   return 0;
}
```

Observe that function `main()` has return type `int`. A return value of 0 indicates to the caller (namely the operating system) that execution was nominal and without errors. Typically main will call other functions, which may in turn call other functions. Function definitions in C look very much like they do in Java.

```
DataType FunctionName(DataType Variable1, DataType Variable2, ...){
   /* local variable declarations */
   /* executable statements */
   /* return statement (provided return type is not void) */
}
```

One important difference however is that local variable declarations must appear at the beginning of the function body, and may not be interspersed with executable statements, as they can in Java.

The function `printf()` , which belongs to the library `stdio.h`, prints formatted text to *stdout*. As you know, stdout is a data stream that, by default, is connected to the terminal window. The first argument to `printf()` is known as a *format string* and consists of two types of characters. The first type are simply characters to be printed. The second type, known as *format specifiers*, define the way the remaining arguments will be printed. A format specifier consists of a percent sign followed by a format code. There must be exactly the same number of format specifiers as there are remaining arguments, and the format specifiers are matched with the remaining arguments in order. For example

```
printf("There are %d days in %s\n", 30, "April");
```

prints the text

```
There are 30 days in April
```

Observe that the newline character "`\n`" is explicitly placed in the output stream. Some common format specifiers are:

```
%c    character
%d    signed decimal integer
%f    floating point number (type double or float)
%s    string of characters
%e    floating point number in scientific notation
%%    prints a percent sign
```

See a good reference on C, such as the URL mentioned above, for other format specifiers. Note that Java contains a method `System.out.printf()` that behaves (almost) exactly like C's `printf()`. The following example prints values of some local variables, along with the usual hello message.

```
/*  HelloWorld2.c */
#include<stdio.h>

int main(){
   int a = 6;
   double x = 2.3;
   char ch = '$';
   char hello_string[] = "Hello, World!";

   printf("a = %d, x = %f, ch = %c\n", a, x, ch);
   printf("%s%c", hello_string, '\n');

   return 0;
}
```

Notice that a string in C is just a char array, and is initialized in much the same way as a String in Java. Also notice that when declaring an array in C, the brackets go after the array name, not after the data type. The output of the above program is:

```
a = 6, x = 2.300000, ch = $
Hello, World!
```

The library `stdio.h` also includes a function called `scanf()`, a general purpose input function that reads the from stream *stdin* normally associated with the keyboard, and stores the information in the variables in its argument list. It can read all the built-in data types and automatically convert them to the proper internal format. The first argument to `scanf()` is a format string consisting of three types of characters: format specifiers, whitespace characters, and non-whitespace characters. The format specifiers consist of a percent sign followed by a format code, and tell `scanf()` what type of data is to be read. For example `%s` reads a string while `%d` reads an integer. Some common format codes are:

```
%c    char
%d    int
%f    float
%lf   double (that's the letter 'l' between % and f, not the number 1)
%s    string (i.e. char array)
```

See a good C reference for other `scanf()` codes, which are similar but not identical to `printf()`'s format codes. Note the URL mentioned at the beginning of this document does not cover `scanf()`. A short listing of both `printf()`'s and `scanf()`'s format codes can be found at

http://nick-lab.gs.washington.edu/cworkshop/formaters.html

The format string is read left to right and the characters are matched, in order, with the remaining arguments to `scanf()`. A whitespace character (i.e. space, newline, or tab) in the format string causes `scanf()` to skip over one or more whitespace characters in the input stream. In other words, one whitespace character in the format string will cause `scanf()` to read, but not to store, any number (including zero) of whitespace characters up to the first non-whitespace character. A non-whitespace character in the format string causes `scanf()` to read and discard a single matching character in the input stream. For example, the control string `" %d, %s"` causes `scanf()` to first skip any number of leading whitespace characters, then read one integer, then read and discard one comma, then skip any number of whitespace characters, then read one string. If the specified character is not found, `scanf()` will terminate. All the variables receiving values through `scanf()` must be passed by reference, i.e. the address of each variable receiving a value must be placed in the argument list. In C, the "*address-of*" operator `&` gives the address of a variable. Thus if `x` is a variable, then the expression `&x` evaluates to the address in memory referred to by the variable `x`. The following program asks the user for three integers, then echoes them back to the screen.

```
/* BasicIO.c */
#include<stdio.h>
#include<stdlib.h>

int main(void){
   int x, y, z;

   printf("Enter three integers separated by");
   printf(" commas, then press return: ");
   scanf(" %d, %d, %d", &x, &y, &z);
   printf("The integers entered were %d, %d, %d\n", x, y, z);
   return 0;
}
```

A sample run of this program looks like

```
Enter three integers separated by commas, then press return: 12, -7, 13
The integers entered were 12, -7, 13
```

Running again, but this time leaving out the separating commas in the input line gives

```
Enter three integers separated by commas, then press return: 12 -7 13
The integers entered were 12, 4, -4261060
```

Since the comma separating the first and second integers was left out `scanf()` read the first integer, then expected to read and discard a comma but failed to do so, and then just returned without reading anything else. The values printed for `y` and `z` are the random data stored in uninitialized variables, and will no doubt be different when you run the program on the same input. Thus we see that `scanf()` is intended for reading *formatted* input. (This is what the "`f`" it its name stands for). The `scanf()` function returns a number equal to the number of variables that were successfully assigned values. That number can be tested and used within the program, as the following example illustrates.

```
/* BasicIO2.c */
#include<stdio.h>
#include<stdlib.h>

int main(void){
   int n, i;
   double x[3];

   printf("Enter three doubles separated by ");
   printf("spaces, then press return: ");
   n = scanf(" %lf %lf %lf", &x[0], &x[1], &x[2]);
   printf("%d numbers were successfully read: ", n);
   for(i=0; i<n; i++){
      printf("%f ", x[i]);
   }
   printf("\n");

   return 0;
}
```

Notice how the `double` array `x[]` is declared in this program, and how this differs from the way it's done in Java. Also notice that `scanf()`'s format code for a double is `%lf`, while `printf()`'s code for double is `%f`. Some sample runs of this program follow:

```
% BasicIO2
Enter three doubles separated by spaces, then press return: 1.2 3.4 G
2 numbers were successfully read: 1.200000 3.400000
% BasicIO2
Enter three doubles separated by spaces, then press return: monkey at the keyboard!
0 numbers were successfully read:
```

If an error occurs before the first variable is assigned, then `scanf()` returns the pre-defined constant `EOF`. Reading the end-of-file character is considered to be such an error. You can place an end-of-file character into the input stream by typing `control-D`. The value of `EOF` is always a negative integer and is defined in the header file `stdlib.h`. Evidently the value of `EOF` on my system is `-1`, as the following test run illustrates.

```
% BasicIO2
Enter three doubles separated by spaces, then press return: ^D
-1 numbers were successfully read:
```

What to turn in

Write a C program called `Sphere.c` that reads in one double value, then prints out the volume and surface area of the sphere whose radius is the value entered. Define a constant value for π in your program by including the declaration and assignment statement

```
const double pi = 3.141592654;
```

at the beginning of function `main()`. The formulas for volume and surface area of a sphere are:

$$\text{Volume} = \frac{4}{3}\pi\, r^3$$

$$\text{Surface Area} = 4\pi\, r^2$$

Implement these formulas in C just as you would in Java, keeping in mind that an expression like `4.0/3.0` indicates double division, while `4/3` indicates integer division. The standard C library `math.h` contains a function called `pow()` that returns its first argument raised to the power of its second argument, i.e. `pow(a, b)` returns a^b. Put the line `#include<math.h>` at the beginning of your program if you want to use this function. Important note: see `Examples/lab7/Circle.c` on the class webpage for instructions on how to compile such a program.

Your program should behave as follows on sample input:

```
% Sphere
Enter the radius of the sphere: 9
The volume is 3053.628060 and the surface area is 1017.876020.
% Sphere
Enter the radius of the sphere: 115
The volume is 6370626.303536 and the surface area is 166190.251397.
```

Submit your program to the assignment name lab7, and perform the usual checks to make sure that it was received.