Setting up the C Compiler

Edit the file .bashrc in your home directory (not your cs246 directory) and insert the following lines:

```
export CFLAGS=-g
export LDLIBS=-lm
```

This turns on debugging support for the compiler and automatically make it automagically link with the math library (for functions like sqrt, exp, and trig functions). Then copy the .bashrc file to .profile.

When you start a terminal window, bash executes your .bashrc file. When you ssh in, you are running a "login" shell and bash instead executes your .profile file.

Test things by starting a new terminal window and running the commands

```
printenv CFLAGS
```

and

```
printenv LDLIBS
```

You should see the values `-g` and `-lm`.

Hello, world

Copy hello.c from /home/mathcs/courses/cs246 to your cs246 directory. Use cat to display the file. Edit it and add the usual comments at the top.

To compile the program, use

```
make hello
```

This will produce an executable file called hello. Use `ls -l` to ensure you have the hello program and that it's executable. Then run it with `./hello`.

Turn the program in with turnin using the assignment name chello (not hello).

Tux

The linux mascot is a penguin named tux. Copy the file tux.aa from /home/mathcs/courses/cs246 to your cs246 directory.

Write a program called tux.c that prints tux. It should have a single printf statement.

First make a copy of tux.aa and call it tux.c. The edit tux.c to add necessary things at the top (comments, includes, declaration of main and turn the artwork into a printf.

In C, two consecutive string literals (like `"foo" "bar''`) separated only by whitespace are concatenated by the compiler to produce a single string. If you have multiple lines to print out that are just string literals, you can do it with one printf like this:

```
printf("Line 1\n"
       "Line 2\n"
       "Line 2\n");
```

This is much easier to read than using several printfs and more efficient as well.

The picture of tux contains some special characters (quotes and backslashes). You will need to escape these with backslashes.

You can test your program by compiling it (with make) and running it. To make sure the output is correct, redirect the output to another file (for example, tux.new). Then compare the output to tux.aa with diff.

```
diff tux.aa tux.new
```

If your output is correct, diff will print nothing. Otherwise it will print a summary of differences between the two files.

A Hollow Square

Write a program called hsquare that prints a hollow square as shown in the sample runs below.

You must have the following ingredients.

1. A function declared as follows:

   ```
   void printchars(char c, int n) {
     ...
   }
   ```

   The printchars function will print n copies of the character c, all on the same line, and nothing else. In particular, it will not print a newline at the end. You can print a character with printf using a `%c` format or with the putchar function.

   The return type void means that the function does not return a value, as in Java.

   ```
   printf("%c", 'x');
   putchar('x');
   ```

   In C, character literals are surrounded by single quotes and strings (even of length 1) are surrounded by double quotes. So `'c'` is a char and `"c"` is a string. The two are not interchangeable.

2. The program will take two command line arguments. The first is the size of the square. The second is the character to print.

   The program has to check for the presence of 2 arguments. If they are not present, it will print a usage message and exit with status 1.

   ```
   if (argc != 3) {
     fprintf(stderr, "usage: hsquare n c\n");
     exit(1);
   }
   ```

3. After you check the number of arguments, you will need to convert the first argument to an int and the second to a char.

   ```
   int n = atoi(argv[1]);
   char c = argv[2][0];
   ```

4. There are 3 cases to consider, n = 0, n = 1, and n > 1. Use a switch statement to handle the cases.

```
switch (n) {
  case 0:
    // Code for n = 0
    break;
  case 1:
    // Code for n = 1
    break;
  default:
    // Code for n > 1
    break;
}
```

Here are some sample runs.

```
> ./hsquare 0 x
> ./hsquare 1 x
x
> ./hsquare 2 x
xx
xx
> ./hsquare 3 %
%%%
% %
%%%
calvin 16:21:26 > ./hsquare 7 \(
(((((((
(     (
(     (
(     (
(     (
(     (
(((((((
```

Important: The printchars function is only responsible for printing a sequence of characters - not for printing the entire square. The main program will use printchars to print sequences of the characters for the outline of the square and also for printing the spaces in the middle.

Important: You must test for the correct number of command lines arguments before you try to use them. Otherwise your program will crash. In any program that uses command line arguments, you need to check for their presence for making any use of them. Using them before checking is like walking through a door and then checking to see if it's open. Disaster follows.

Hollow Diamond

Modify the hsquare program to print a hollow diamond. Call it hdiamond.c. The command line arguments are the same as for hsquare except that the size of the diamond is 2n + 1 where n is the first argument.

Sample runs:

```
> ./hdiamond 0 +
+
> ./hdiamond 1 p
 p
p p
 p
> ./hdiamond 2 B
  B
 B B
B   B
 B B
  B
> ./hdiamond 3 @
   @
  @ @
 @   @
@     @
 @   @
  @ @
   @
> ./hdiamond 4 @
    @
   @ @
  @   @
 @     @
@       @
 @     @
  @   @
   @ @
    @
```