ECE 209
Fall 2010

Program 1: `tri`

See handout for time

Due Friday, Sept 17, 2010 @ 11:45pm

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

This program will print a triangle of asterisks (*) on the screen. The user will input the size (number of rows in the triangle) and the desired orientation (pointing up, down, left, or right).

The learning objectives of the program are:

- Write a complete C program.
- Use loops and conditional statements.
- Use printf and scanf to perform I/O.
- Use functions to modularize the program.

Program Specification

The entire program must be written in a single source code file, named **tri.c**.

The program will ask the user to input the size of the triangle. You can assume that the user will type a decimal integer, followed by "Enter" (linefeed). The size must be an odd integer between 1 and 25, inclusive. If the user enters an illegal integer, the program prints an error message (see below) and asks again.

The prompt message and the error message must look exactly like the example below. (Characters typed by the user highlighted in gray.)

```
Enter the triangle size: 4
Illegal input -- must be an odd integer between 1 and 25.

Enter the triangle size: 11
```

Once an acceptable size has been entered, the program will ask the user for the orientation of the triangle. If the user enters 1, the point of the triangle will point up. If the user enters 2, the point of the triangle will point down. If the user enters 3, the point of the triangle will point to the right. If the user

enters 4, the point of the triangle will point to the right. If the user enters any other integer, the program asks again. (You may assume that the user will type an integer, followed by linefeed.)

Example:

```
Enter the triangle size: 5

Enter the orientation (1=up, 2=down, 3=right, 4=left): 5
Enter the orientation (1=up, 2=down, 3=right, 4=left): -3
Enter the orientation (1=up, 2=down, 3=right, 4=left): 1
```

The program then skips a line and prints the triangle, according to the desired size and orientation. Consider an upward-facing triangle with size = 5:

```
    *
   * *
  * * *
 * * * *
* * * * *
```

There is one asterisk in the first row, two in the second, etc. There is a single space between each asterisk in a given row. The bottom row starts at the left edge of the output console. Asterisks on adjacent rows are offset by a single space.

A downward-facing triangle with size = 5:

```
* * * * *
 * * * *
  * * *
   * *
    *
```

A right-facing triangle with size = 5:

```
*
 *
* *
 * *
* * *
 * *
* *
 *
*
```

A left-facing triangle with size = 5:

```
    *
   *
  * *
 * *
* * *
 * *
  * *
   *
    *
```

Note that the left/right triangles look squashed horizontally (or stretched vertically), because the space between lines is different than the space between characters on the same line. But the spacing is the same as in the up/down case.

Once the triangle is printed, the program prints a single blank line and then exits (by returning the value EXIT_SUCCESS).

You must write at least one function in addition to main in your program. We strongly suggest that you write a separate function for each orientation. For example, the **upTriangle** function takes one argument (the size) and prints an upward-facing triangle of the given size.

Compiling

On an EOS Linux system (e.g., remote-linux.eos.ncsu.edu), use gcc to compile and link the program as follows:

```
gcc –o tri -g -Wall -pedantic tri.c
```

This compiles your C course code file (**tri.c**) and creates an executable file (**tri**).

Then you run the program like so (as shown above):

```
./tri
```

I'm willing to relax these rules a little. See handout.

For NetBeans users, to set the proper compiler flags (-g -Wall -pedantic):

1. Right-click on the project name (in the Projects display on the left) and select Properties. Under Build, select C Compiler.
2. Under Basic Options, on the line that says Warning Levels, select More Warnings. On the line that says Development Mode, select Debug. (Debug should be the default setting.)
3. Under Command Line, click on the box next to Additional Options. In the text box, type "-pedantic" (without the quotes).
4. Click OK, and then click OK again.

Testing the Program

Your program must run for triangles of all sizes between 1 and 25, and for all four orientations. You don't necessarily need to test all 100 combinations of size and orientation. But it's a good idea to test the biggest, smallest, and a few sizes in between. For a given size, you should definitely test all four orientations.

Example Run

Here is complete example of how the program looks when you run it on an EOS Linux system. The "eos%" string is the prompt printed by the operating system:

```
eos%./tri
Enter the triangle size: 6
Illegal input -- must be an odd integer between 1 and 25.

Enter the triangle size: 7

Enter the orientation (1=up, 2=down, 3=right, 4=left): 99
Enter the orientation (1=up, 2=down, 3=right, 4=left): 1

      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *

eos%
```

Hints and Suggestions

- Don't overcomplicate the program. Do not use anything that we have not covered in class. (For example, don't use an array.)
- You must include **stdio.h** for **printf/scanf** and **stdlib.h** for EXIT_SUCCESS.
- Write your functions in a general way. In other words, don't write code for all of the specific sizes (1, 3, 5, 7, 9, ..., 25). Write a single routine that will work for any size. The main things to figure out are: (1) how many rows to print, (2) how many asterisks on each row, and (3) how many spaces to print before you start printing asterisks.
- However, a good way to think about the problem is to consider the code for a few specific cases, and then generalize that code to handle all cases.
- For compiler errors, look at the source code statement mentioned in the error. Try to figure out what the error is telling you. Try to fix the first error first, and then recompile. Sometimes, fixing the first error will make all the other errors go away. (Because the compiler got confused after the first error.)
- Use a source-level debugger, like *kdbg*, to step through the program if the program behavior is not correct. If you are using NetBeans on your own computer, the debugger is integrated with the editor and compiler, so there's no excuse for not using it.
- For general questions or clarifications, use the Message Board, so that other students can see your question and the answer. For code-specific questions, email your code (as an attachment) to the support list: ece209-001-sup@wolfware.ncsu.edu.

Don't do this!
Email your instructor!

Administrative Info

Updates or clarifications on the Message Board:

Any corrections or clarifications to this program spec will be posted on the Message Board. It is important that you read these postings, so that your program will match the updated specification.

What to turn in:

- Source file – it must be named **tri.c**. Submit via Wolfware to the Program 1 assignment in your problem session section (201, 202, ..., 206). Your grade will be posted on the 209 gradebook, and you will be able to get detailed grading information using the "Retrieve Assignment" option on Wolfware.

Assignment will be submitted via moodle and grades will be available there.

Grading criteria:

15 points: File submitted with the proper name.

15 points: Compiles with no warnings and no errors (using –Wall and –pedantic). (If the program is not complete, then only partial credit is available here. In other words, you won't get 15 points for compiling a few trivial lines of code.)

10 points: Proper coding style, comments, and headers. No unnecessary global variables. No goto. (See the Programs web page for more information.)

10 points: At least one additional function, other than **main**, is used in the program. (We suggest at least four, one for each orientation, but we won't insist on it.)

10 points: The program correctly gets the triangle size from the user, dealing with illegal inputs as specified above.

10 points: The program correctly gets the orientation from the user, dealing with illegal inputs as specified above.

10 points: The program correctly prints an upward-facing triangle of any legal size.

10 points: The program correctly prints a downward-facing triangle of any legal size.

5 points: The program correctly prints a right-facing triangle of any legal size.

5 points: The program correctly prints a left-facing triangle of any legal size.