Lab 2: An Introduction to Condtional Execution in C

CSE/IT 113

"When you come to a fork in the road, take it."

— Attributed to Yogi Berra

"Each decision we make, each action we take, is born out of an intention."

— Sharon Salzberg

"How do we convince people that in programming simplicity and clarity–in short: what mathematicians call "elegance"–are not a dispensable luxury, but a crucial matter that decides between success and failure?"

— Edsger Dijkstra

---

1 Introduction

In Lab 1, you created a program which read a variety of input from the user. The numbers were used to calculate the results of various geometric operations and the results were displayed to the screen (stdin). Unlike a real calculator, the user could not choose which calculation to perform. In this lab, you will learn how to have the user choose the desired calculation.

Lab 2 introduces the `if` statement and the `switch` statement. These statements are called conditional execution because they allow us to choose one action from one or more possible actions.

Lab 2 also introduces the concept of reading character input from the keyboard and performing comparisons on character values.

2 Requirements

In order to receive full credit for this lab, you must **write a simple geometry calculator program**. The program will ask the user what calculation to perform, ask for input, and then carry out the desired calculation. If there is an error in user input at any step along the way, the program will print a message and exit the program.

Your program ***must*** be able to execute all the geometry calculations from Lab 1.

You ***must*** use

- a nested `switch` statement to determine which calculation to perform. You will first ask what general type of calculation (Circle, Regular Polygon, Rectangle, or Triangle) the user wants. And based on that input, determine which operation to perform, get user input to perform the calculation, and carry out the calculation.
- error check user input at every step of the way. You should accept both upper and lower case values for menu options, and since this is geometry the values the user entered should be greater or equal to zero. If an error occurs, print a message to the screen and exit the program. Depending on the error, you will either handle it with a **`switch`** or a **`if`** statement. See the sample code (below) for an example.
- Name your source code for the geometry calculator `lab2.c`

2.1 Sample output

The output of your program will look as follows, where the red text indicates input from the user and is not output from your program. The blue text is information that will be individual to your calculator or output that is calculated based on the values input by the user.

Important: The output must be calculated by your program. That is the point of this program. Do *not* print the exact output as given here as an example of the program in operation. Your program should perform the user-specified arithmetic operation based on the numerical values and operator read in from the keyboard.

```
Welcome to Wile E. Coyote's Geometry Calculator!

Guaranteed to Pythagorize the Roadrunner in his tracks!

Please select a geometry calculation:
```

```
C. Circles

P. Regular Polygons

R. Rectangles

T. Right Triangles

Please enter your choice (C, P, R, T): C

A. Area of a circle

C. Circumference of a circle

Please enter your choice (A, C): A

Enter the radius of the circle:  2

The area of a circle with radius 2.0 is 12.566371
```

or if the user made an error in the value of the radius entered:

```
Welcome to Wile E. Coyote's Geometry Calculator!

Guaranteed to Pythagorize the Roadrunner in his tracks!

Please select a geometry calculation:

C. Circles

P. Regular Polygons

R. Rectangles

T. Right Triangles

Please enter your choice (C, P, R, T): C

A. Area of a circle

C. Circumference of a circle

Please enter your choice (A, C): A

Enter the radius of the circle:  -2

Error:  radius has to be greater or equal to zero
```

```
Goodbye.
```

2.2 Characters and Comparisons

Up to this point you have dealt with two data types: `int` for integers and `double` for floating-point numbers. This lab introduces a third type `char`, which holds a single character (one byte) such as `A, B, C, a, b, c` etc. To assign a character to a variable we use *single* quotes. For example,

```
1 char c = 'a';
2 printf("%c\n", c); /* what does this print -- try it*/
```

Note that we are using *single* quotes. The following is wrong

```
1 char c = "a"; /* this is not correct */
```

Double quotes are used to indicate strings (you will learn what strings are exactly later in the course). For right now, recognize that 'a' is not the same as "a" and single quotes are used for character data types.

Also make sure you understand the difference between numbers and their character representation. For example, `'5'` is not the same as `5`. `'5'` is a character type and has an ASCII value of `0x35 or in decimal 53`; `5` is an integer type and represents the number `5`. Remember like UNIX, character types are case-sensitive: 'a' is not the same as 'A'.

To compare character types you use the equality operator (==)

```
1
2  char c = 'd';
3  char d = 'd';
4
5  if (c == d) {
6          printf("c is d\n");
7  }
8  else {
9          printf("c is not d\n");
10 }
```

What does the above code fragment print? Try it.

2.3 getchar() and fgets()

Rather than using `scanf()` for user input, you are going to use `getchar()`, which accepts character input.

As you saw in prelab, for character input `getchar()` is preferred over `scanf()`. For example:

```
1 int c;
2 int tmp;
3 printf("Please enter a character: ");
4
5 while ((tmp = getchar()) != '\n')
6     c = tmp;
7
8 printf("c = %c\n", c); /* %c token is for characters */
```

This will print the last character entered before the Enter key is pressed. Try it.

Since some of your input will be floating point or integers, you need a means to grab the input and convert it to a `double` or an `int`. Unfortunately, `scanf()` doesn't work well when you mix character and numeric input. A better way to get user input is to use `fgets()`, which takes in a string – a sequence of characters that is NULL terminated. Once you have a string, it is easy to convert to a `double` using `atof()` or an integer using `atoi()`.

Strings are stored contiguously in memory, one character after another. For example, in memory the string `hello, world\n`, begins with the character h, followed by e, etc. and ends with a NULL character (not shown) following the newline character. So in memory the string looks like this: `hello, world\n\0`. The NULL character or `\0` indicates the end of the string.

In the sample code below, the line `char s[1024]` creates space in memory to hold a string that is at most 1023 characters long. That is not a typo – its one less than 1024 since strings append a NULL character to the end of the string to indicate the end of the string in memory. In the sample code, the function call `fgets(s, 1024, stdin)` reads at most 1023 characters from stdin (terminal) and stores the characters in the variable s. `fgets()` stops reading when you enter a newline (Enter key). Once you have the string, you need to convert it to the appropriate type. For integers, use `atoi()` and for `doubles` use `atof()`. Both of these functions take a string as a parameter and convert it to the appropriate numeric type.

Sample Code

The following program gives an example of nested switch statements, asking for user input, using `getchar` and `fgets`, converting strings to an integer or a `double`, and checking for errors.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int in;
        int tmp;
        int num;
        char s[1024]; /* this is a character array */
        double d;

        printf("Please enter a character between a - c: ");
        while ((tmp = getchar()) != '\n')
                in = tmp;

        switch (in) {
        case 'A':
        case 'a':
                printf("enter an integer between 1 - 5: ");

                /* grab input from stdin and store in s */
                fgets(s, 1024, stdin);

                /* convert input to an int */
                num = atoi(s);

                /* error check num */
                if (num < 1 || num > 5) {
                        printf("error in input\n");
                        printf("%d is not between 1 and 5\n", num);
                        printf("I have to exit\n");
                        printf("Goodbye\n");
                        return 1; /* exits the program */
                }
                else {
                        switch (num) {
                        case 1:
                        case 2:
                        case 3:
                        case 4:
                        case 5:
```

```
42                                  printf("You entered %d\n", num);
43                                  break;
44                          }
45                  }
46                  break;
47
48          case 'B':
49          case 'b':
50                  printf("Enter a real number: ");
51
52                  /* grab input from stdin and store in s */
53                  fgets(s, 1024, stdin);
54
55                  /* convert input to a double */
56                  d = atof(s);
57
58                  printf("You entered %lf\n", d);
59                  break;
60
61          case 'C':
62          case 'c':
63                  printf("%c entered - todo\n", in);
64                  break;
65
66          default:
67                  printf("You entered %c, which is not an a, b, or ↙
                        ↳ c\n", in);
68                  printf("Goodbye\n");
69                  return 2; /* exit the program -- return error ↙
                        ↳ code */
70          }
71
72          return 0;
73
74  }
```

3 Additional Problems

In addition to the the above requirements, turn in the following problems from Chapter 5 of *C Programming: A Modern Approach*: Exercise #11, Programming Projects #1, #7, #8, #11.

If the problem is an exercise, name the source code for these problems `ch5_exN.c`, where N is replaced the exercise number.

If the problem is a programming project, name the source code for these problems `ch5_ppN.c`, where N is replaced the programming project number.

You will submit these problems as part of the lab tarball.

Make sure your code for the additional problems follows the coding style of the class and is commented.

4 Submission guidelines

Make sure you comment your code. Follow the comment and Coding Style from Lab 1.

For submission, you will upload a tar archive containing the following items:

- `lab2.c` file
- a script, named `lab2.script` of you running lab2 program (include all cases of operation)
- and a `README` file as described in the sample `README.pdf` on Canvas.
- `ch5_exN.c` and `ch5_ppN.c` files.

The name of your tar archive should be

`cse113_firstname_lastname_lab2.tar.gz`

Upload the tarball to Canvas before the due date.