Lab 5

Introduction to C
CSCI 112, Spring 2015

Objectives

- Learn about function prototypes
- Use pointers to pass by reference
- Translate mathematical equations to C expressions
- More practice with a `switch` statement
- More practice with a user menu
- More practice with the math library

Overview

In this lab, you will write a program that computes the following physics motion equations:

$$v_f = v_0 + a \cdot t \tag{1}$$

$$x_f = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \tag{2}$$

$$v_f = \sqrt{v_0^2 + 2 \cdot a(x_f - x_0)} \tag{3}$$

$$x_f = x_0 + \frac{1}{2} \cdot (v_f + v_0) \cdot t \tag{4}$$

where x_0 is the initial position, x_f is the final position, v_0 is the initial velocity, v_f is the final velocity, a is acceleration, and t is time.

See the `Example Execution` section below for detailed examples.

Function Prototypes

In C, all identifiers need to be declared before they are first used. We have seen this for variables, but it is also true for functions. In standard C, all functions must be declared before the first call of the function. A function declaration, also known as a prototype, contains the return type, the function name, and a list of function parameter types.

In past labs, we have avoided this issue by placing the `main` function as the last function in the program. In this lab, you will place your `main` function as the first function in the program. Therefore, you will need to include function declarations for all your functions in the program. Notice that the parameter variable names are absent in the declaration. You will need to give these variable names in your implementation of the function, but you omit them in the function prototype.

These are the function declarations you will need in this lab. Place these lines above your `main` function and below your `#include` statements. You will write the function definition (i.e., implementation) for each of these functions.

```c
// List of all function declarations (prototypes)

// Displays user menu, reads input, and validates the input
int user_menu();

// Equation functions are pass by reference
void equation1(float *);                  // Calculate motion equation 1
void equation2(float *);                  // Calculate motion equation 2
void equation3(float *);                  // Calculate motion equation 3
void equation4(float *);                  // Calculate motion equation 4

// User input functions return a value from user
float get_position_initial(void);         // Prompts user for x0
float get_position_final(void);           // Prompts user for xf
float get_velocity_initial(void);         // Prompts user for v0
float get_velocity_final(void);           // Prompts user for vf
float get_acceleration(void);             // Prompts user for a
float get_time(void);                     // Prompts user for t
```

User Input Functions

The four equations require the use many of the same variables. Instead of duplicating user input requests, you will instead create a function for each of six user value requests. Each of these short functions will consist of one `printf` statement, one `scanf` functions, and a return line. For these functions, you do not need to validate the input the user enters. You may assume the user will enter a float value.

For example, the function `get_position_final()` will prompt the user for a position (x_f), read in a `float` from the user, and return that value.

Equation Functions

You will create four functions, one for each of the motion equations above, named `equation1(·)` through `equation4(·)`. Each of these functions will prompt the user for the terms needed in the calculation, using the functions you completed in the `User Input Functions` section above. This means these four functions will not contain any calls to `printf` or `scanf`, but rather use calls to your user input functions such as `float x0 = get_position_initial();`

The order in which you request the user variables is important. Ask for the variables in the order they first appear in the equation:

Equation 1:

1. initial velocity
2. acceleration
3. time

Equation 2:

1. initial position
2. initial velocity
3. time
4. acceleration

Equation 3:

1. initial velocity
2. acceleration
3. final position
4. initial position

Equation 4:

1. initial position
2. final velocity
3. initial velocity
4. time

The graders will enter in data in the order listed above. Do note change the order of these prompts or your function will calculate an incorrect value.

After requesting all necessary data from the user, your function will perform the equation calculation.

PROTIP: Review the example on Slide 18 of slide deck 09-Pointers.

Previously, you have used `return` statement to return a value from a function. For these four functions, you will instead pass-by-reference. This means that you will pass your function the address of the result variable. Notice that the function declaration given to you requires a pointer variable (e.g., `float * velocity_final`).

After performing your calculation, you will change the value of the result variable using the pointer given to you and the dereferencing operator. These functions will not have a return statement and the return type of the function is `void`.

PROTIP: Ampersand & is address operator and asterisk * is dereferencing operator.

Failure to use pass by reference as described above will result in a significant loss of points.

PROTIP: You will need to include the math library to use the `sqrt` and the `powf` functions. These allow you to take the square root or raise a value to a power, respectively.

User Menu Function

Write a function `user_menu(·)` that prints out the menu,

```
Choose a motion equation 1-4 or choose 5 to QUIT > 0
```

reads in the integer choice from the user, validates the choice is between 1 and 5, and returns the valid menu option the user has chosen. If the user selects an incorrect choice, your code will notify the user with an error message: `Invalid option.  Please try again.` You will then repeat the menu, repeat in the user input, and validate again.

Main Function

Since you gave function declarations, your `main` function will appear as the first function in your program. Your `main` function will be responsible for a sentinel loop which will call your function `user_menu()` and process the user choice. If the user chooses 1-4, you will call the appropriate equation function. If the user choose 5, your program will exit.

I have completed much of the `main` function for you, given in Figure 1, to ensure you are passing-by-reference. You need to complete the missing cases 1-4 which will contain function calls to `equation1` through `equation4`, respectively.

Compile & Test

Compile your program using this `gcc` command.

```
$ c99 lab5.c -o program5 -lm
```

The additional `"-lm"` flag is required because to use the math library. `c99` is a shortcut for running `gcc -std=c99`, which uses the C99 standard instead of the default C89 standard. Test your program! Make sure your test each of the equations, your `QUIT` condition, and any input integers that are out of bounds of the menu options.

Protip: Make sure you output what is expected, nothing more and nothing less. Do not change the order or the number associated with each of the menu options, otherwise the grading script will choose incorrectly.

Try to recreate the output of the `Example Execution` by making the same menu choices given in the example. Does your output look the same?

```c
int main (void)
{
    // Print welcome message
    printf("Welcome to the MOTION EQUATION CALCULATOR\n\n");

    // Variable for the user choice from menu
    int user_choice;

    do{
        user_choice = user_menu();      // Print menu, validate choice is between 1 and 5

        // You will pass the address of this variable to your equation functions
        float result;                   // Variable to hold calculated result

        // Handle menu choice selected by user
        switch (user_choice){

            /*
             * TO DO
             * You will complete cases 1 - 4
             */

            case 5 :
                // Exit program if user selects QUIT
                printf("Thank you for using the MOTION EQUATION CALCULATOR. Goodbye.\n");
                return 0;
                break;
        }
        // Print out the calculated result with four digits after the decimal point
        printf("Your result is %.4f.\n\n", result);

        } while(user_choice !=5);       // Repeat until user chooses QUIT

        return 0;                       // Return without error
}
```

Figure 1: `main` Function for Lab5

Example Execution

```
$ ./program5

Welcome to the MOTION EQUATION CALCULATOR

Choose a motion equation 1-4 or choose 5 to QUIT > 0
Invalid option.  Please try again.

Choose a motion equation 1-4 or choose 5 to QUIT > 1
        Enter initial velocity > 1.1
        Enter acceleration > 2.2
        Enter time > 3
Your result is 7.7000.

Choose a motion equation 1-4 or choose 5 to QUIT > 2
        Enter initial position > 6.4
        Enter initial velocity > 3.2
        Enter time > 5
        Enter acceleration > 2.8
Your result is 57.4000.

Choose a motion equation 1-4 or choose 5 to QUIT > 3
        Enter initial velocity > 3.7
        Enter acceleration > 1.9
        Enter final position > 2.1
        Enter initial position > 1.0
Your result is 4.2273.

Choose a motion equation 1-4 or choose 5 to QUIT > 4
        Enter initial position > 3.2
        Enter final velocity > 1.0
        Enter initial velocity > 4.3
        Enter time > 7.1
Your result is 22.0150.

Choose a motion equation 1-4 or choose 5 to QUIT > 6
Invalid option.  Please try again.

Choose a motion equation 1-4 or choose 5 to QUIT > 5
Thank you for using the MOTION EQUATION CALCULATOR.  Goodbye.
```

Resources

- You should attend the Friday Lab session to seek assistance from the TAs and CAs.
- For general questions, check the D2L `Lab 5 Help Forum` to see if another student has already asked your question. Otherwise, post your question on the D2L forum. This forum is intended for general questions that will benefit all students. Do not paste your code nor give away any specific answers to the lab on this forum.
- For specific questions, attend the weekly lab session or attend the TA's office hours.
- You are encouraged to use resources or tutorials on the internet to learn unix or C. Check the class resource list on D2L for some links to useful resources.

Submission

- ASSIGNED: March 3rd
- LAB DAY: March 6th
- DUE DATE: March 20th, 8:00am (** after Spring Break **)

1. Make sure you included ample and informative comments – it is 20% of your grade!
2. Rename your C file to `last_first_lab5.c` and substitute your last and first name.

   PROTIP: If you fail to follow the above file naming conventions, your program may not be graded automatically and you will lose points.

3. Submit your `.c` file to the D2L dropbox Lab5.

   PROTIP: Submit only your C file. Do not submit your object file or your executable program. Do not archive (e.g., `zip`) your file.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work will receive a 50% penalty. Labs submitted more than 24 hours late will not be accepted. The late deadline is Saturday, March 21th, 8:00am.