CS-212: LAB 1 - Refresher Lab

Learning Objective:

- Preparing your development area
- Compiling a program with gcc
- Use printf and scanf for input and output
- Use if-statements control what code gets executed
- Write functions with pointers and value parameters
- Capture your output in a text file

This lab has two parts:

- Part-A: You will copy a simple program into a code file (.c file).
- Part-B: You re-organize part A so that the main function is nothing but function calls.
- There is an extra credit worth up to 10%

Preparing your development area:

- If you are not connected to the university network, please follow the instructions in the following link to connect from off-campus: https://www.binghamton.edu/its/about/organization/operations-and-infrastructure/networking/off-campus-connecting.html

- It is possible for Binghamton University students to use a facility called a "Secure Socket Shell" or SSH for short in order to log on to a Binghamton University Linux server machine using a terminal window on your laptop. For this, you will need software to enable opening an SSH window from your laptop:

    - **Windows (the easy option**): Open the command prompt by clicking the Start button and type CMD. At the prompt, type one of the following:
      ssh <userid>@bingweb.binghamton.edu
      ssh <userid>@harveyv.binghamton.edu
      You may be prompted a yes/no question, if so, type yes
      Type your password. Nothing will appear on the screen as you type the password. This is normal.

    - **Windows (MobaXterm option):** If you are on a Windows machine, a free product called MobaXterm is available for you to use. Other SSH client software is available, but MobaXterm is easy, and everything you need is available in the free edition. Download and install the free edition from

https://mobaxterm.mobatek.net/. Once you have installed MobaXterm, start it, and create a new session by clicking on the "Session" icon. Then, starting an "SSH" session, specifying a Binghamton University Server (bingweb.binghamton.edu) as the remote host and your PODS userid as the username. Leave the "Port" at the default of 22. None of the advanced settings need to be modified.

- **Mac OS and Linux:** If you are on a Mac OS or Linux machine, a ssh client is already installed on your computer. Open a terminal and type the below command to access the Linux Server (where <userid> is your PODS userid). The first time you do this, ssh will ask if you accept the host's key fingerprint. I say "yes" at this point. After the first time, this question will not appear:
  ssh <userid>@bingweb.binghamton.edu

- Your account on the Linux server uses a U-Drive that is assigned to you by the University. I find it very useful to mount your network U-Drive on your laptop. This way, you can edit files locally, using your local editor, and compile/run on the Linux server using the SSH window. Please follow the instructions in the following link to mount your network U-Drive: https://www.binghamton.edu/its/helpdesk/u-drive.html

- Feel free to use the editor of your choice. I prefer to use Notepad++ 😊

- You may find value in downloading Microsoft Visual Studio 2022 Community. Although the final version of your code must run on gcc, I often find VS 2022 useful for debugging.

Lab 1 Setup:

1. Open a new terminal window to SSH into your account on the [bingweb.binghamton.com] server. When logged in, make sure you are in your home directory by typing the following command if you are not already there. The "~" is just a shortcut symbol that means home directory:
   cd ~

2. Create a directory named "CS212" by typing:
   mkdir CS212

3. Change your current directory to the "CS212" directory by typing:
   cd CS212

4. Confirm that you are in the "CS212" directory by typing:
   pwd

5. Repeat steps 2 through 4 to create a directory for Lab1.

6. Create a new c file for you to write your lab.
   > <Last Name><First Initial>_Lab1.c

   Example: I would create FoosJ_Lab1.c

   Note: We will not use a header file for this lab. We will start using header files in lab 2.

PART A:

Open your code file (Lab1.c) and type the program below:

1. Copy in the IPO from IPO for CS-212.txt.
2. Fill out the first block with your name, lab # and data (You can just use the current date)
3. You can fill out the other blocks after you're done.
4. Copy the code below into your .c file.

NOTE: All your code must be documented, properly spaced, and properly indented

Write your include statements and macros/constants:

```c
// Include for printf and scanf
#include <stdio.h>

// Macros
#define SCHOOL "Binghamton University"
#define NAME "Put your name here"
#define LAB "Lab 1"

// Constant
const double PI = 3.14;
```

Write the beginning of your main and declare your variables:

```c
int main(void)
{
    int selection;
    int valueA;
    int valueB;
    int valueC;

    double area;
```

Write the code to print your header to the screen:

```
// Move the cursor down one line
printf("\n");

// Print header to the screen
printf("************************************************************\n");
printf("%s\n", SCHOOL);
printf("%s\n", NAME);
printf("%s\n", LAB);
printf("************************************************************\n");

// Move the cursor down one line
printf("\n");
```

Note: You do not need the exact same number of *

Write the code to display a menu and prompt the user for a selection:

```
// Move the cursor down one line
printf("\n");

// Print menu to the screen
printf("Select a shape to calculate for area calculation:\n");
printf("1. Parallelogram\n");
printf("2. Trapezium\n");
printf("3. Ellipse\n");

// Prompt user for an answer and store the answer in a variable
printf("Enter selection: ");
scanf("%d", &selection);

// Move the cursor down one line
printf("\n");
```

Write the code to process the parallelogram:

```
// Check to see if user selected Parallelogram
if (selection == 1)
{
    // Prompt user for the length of the base and vertical height
    printf("Enter the base and vertical height separated by a space: ");
    scanf("%d %d", &valueA, &valueB);

    // Calculate the area
    area = valueA * valueB;
}
```

Write the code to process the trapezium:

```
// Check to see if the user selected Trapezium
else if (selection == 2)
{
    // Prompt user of length of the sides
    printf("Enter the length of side A, side B, and the height separated by a spaces: ");
    scanf("%d %d %d", &valueA, &valueB, &valueC);

    // Calculate the area
    area = (double)((valueA + valueB) * valueC) / 2;
}
```

Write the code to process the ellipse:

```
// Assume the user selected ellipse
else
{
    // Prompt user of the minor and major axis
    printf("Enter the length of the minor and major axis separated by a spaces: ");
    scanf("%d %d", &valueA, &valueB);

    // Calculate the area
    area = PI * valueA * valueB;
}

// Move the cursor down one line
printf("\n");
```

Write the code print the name of the shape:

```
// Print the shape name
switch (selection)
{
case 1:
    // Print shape to the screen
    printf("Shape: Parallelogram\n");
    break;

case 2:
    // Print shape to the screen
    printf("Shape: Trapezium\n");
    break;

case 3:
    // Print shape to the screen
    printf("Shape: Ellipse\n");
    break;
}
```

Write the code to print the results for the parallelogram:

```
// Print results
if (selection == 1)
{
    // Print results for parallelogram
    printf("Length of the base: %d:\n", valueA);
    printf("Length of the height: %d:\n", valueB);
    printf("Area: %.3lf\n", area);
}
```

Write the code to print the results for the trapezium:

```
else if (selection == 2)
{
    // Print results for trapezium
    printf("Length of side A: %d:\n", valueA);
    printf("Length of side B: %d:\n", valueB);
    printf("Length of the height: %d:\n", valueC);
    printf("Area: %.3lf\n", area);
}
```

Write the code to print the results for the ellipse:

```
else
{
    // Print results for ellipse
    printf("Length of the minor axis: %d:\n", valueA);
    printf("Length of the major axis: %d:\n", valueB);
    printf("Area: %.3lf\n", area);
}
```

Write the code to finish the main function:

```
    // Print divider
    printf("************************************************************\n");

    // Move the cursor down one line
    printf("\n");

    return 0;
}
```

5. Run your program with the following command:
   gcc Lab1.c -Wall -o Lab1.out

   Note: Your .c file should be named <Last Name><First Initial>_Lab1.c

   The -Wall option will show the warnings, the default will hide some of them.

   If you have errors or warning in your code, fix them before moving on to part B.
6. Execute your code with the following command:
   ./Lab1.out
7. Do a couple of runs and make sure you get correct output before moving on to the next section.

PART B:

Now that you have a working program it's time to break the program into functions:

For part B you will write the following functions:

- PrintHeader
- PrintMenuAndGetSelection
- ProcessSelection
- PrintShape
- PrintResults

Part B requirements:

- Parameters: Determine which parameters you need, which need pointers, and which do not.
- Your main function should have your variable and function calls.
- Your main function should be the first function in your .c file.
- Declare your function prototypes above the main function.
- All function documentation blocks must be present and filled out

Function documentation blocks:

Copy the following documentation block above every function header:

```
//-------------------------------------------------------------------------
// Function Name:
// Description:
//
//
//-------------------------------------------------------------------------
```

You can copy the block in from **Function Documentation Block.txt**

Fill out the documentation blocks with the appropriate information.

HINT: Use the sample programs that are provided on Brightspace as guides to writing functions

EXTRA CREDIT: 10%

If your lab does not run with proper output, no extra credit may be earned.

Update your function PrintMenuAndGetSelection so that it uses a do-while loop to check the input. If an invalid number is entered, an error message must be printed to the screen and the loop must continue. Also, add 0 as an option to quit.

All the code must be documented, properly spaced, and properly indented.

RUNNING YOUR PROGRAM

Your program must run in gcc on the unix operating system with 0 errors and 0 warnings.

To run your program:

```
gcc *.c -Wall
```

This will create an output file named a.out

Valgrind must run with 0 errors and 0 warnings.

To run Valgrind:

```
valgrind --leak-check=full ./a.out
```

Note: These directions are slightly different than above, but they work the same way.

SAVING YOUR OUTPUT:

After you run your program, select all the text from your terminal and press enter. (Pressing the enter key is like selecting copy). Then open a text document (you will have to create this) and select Paste. Don't forget to save your work.

For your output, add the output for three runs, one time for each shape.

PREPARING YOUR LAB TO BE SUBMITTED:

- Before submitting your lab:
- Make sure you filled out the IPO
- Make sure you filled out all your function documentation blocks.
- Check your code for proper spacing, documentation, and indentation
- Double check that you saved your output file

When you're all done, submit the following files to Brightspace:

- <lastName><firstInitial>_Lab1.c
- Lab1_Output.txt

ADDITIONAL PARTIAL CREDIT GRADING RUBRIC:

NOTE: The full rubric is on Brightspace under Misc

None.

Note to Grader: Any partial credit outside of the rubric must be approved by the teacher.