CPE 150 Laboratory 8: Functions III

Department of Computer Engineering
Yarmouk University

Summer 2017

1 Objectives

- To understand how to construct programs modularly from pieces called functions.
- To create new functions.
- To understand the mechanisms used to pass information between functions.
- To introduce simulation techinques using random number generation.
- To understand how the visibility of identifiers is limited to specific regions of programs.
- To understand how to write and use functions that call themselves.

2 Lab Note

In this lab, create a header file to include the proper user-defined function prototypes, then create a source file to include only the implementation for these user-defined functions. The main driver for your solution should be in a separate source file that includes the user-defined function using proper directives.

3 Lab Exercise 1 - Circle Calculator

Exercise 1: Write the following program into a source file named `scope.cpp`.

```
#include <iostream>
#include <iomanip>
using namespace std;
// This program will demonstrate the scope rules.
const double PI = 3.14;
const double RATE = 0.25;
void findArea(float, float&);
void findCircumference(float, float&);
int main()
{
    cout << fixed << showpoint << setprecision(2);
    float radius = 12;
    cout <<" Main function outer block" << endl;
    cout <<" LIST THE IDENTIFIERS THAT are active here" << endl << endl;
    {
```

```
        float area;
        cout << "Main function first inner block" << endl;
        cout << "LIST THE IDENTIFIERS THAT are active here" << endl << endl;
        // Fill in the code to call findArea here
        cout << "The radius = " << radius << endl;
        cout << "The area = " << area << endl << endl;
    }
    {
        float radius = 10;
        float circumference;
        cout << "Main function second inner block" << endl;
        cout << "LIST THE IDENTIFIERS THAT are active here" << endl << endl;
        // Fill in the code to call findCircumference here
        cout << "The radius = " << radius << endl;
        cout << "The circumference = " << circumference << endl << endl;
    }
    cout << "Main function after all the calls" << endl;
    cout << "LIST THE IDENTIFIERS THAT are active here" << endl << endl;
    return 0;
}

void findArea(float rad, float& answer)
{
    cout << "AREA FUNCTION" << endl << endl;
    cout << "LIST THE IDENTIFIERS THAT are active here"<< endl << endl;
    // FILL in the code, given that parameter rad contains the radius, that
    // will find the area to be stored in answer
}

void findCircumference(float length, float& distance)
{
    cout << "CIRCUMFERENCE FUNCTION" << endl << endl;
    cout << "LIST THE IDENTIFIERS THAT are active here" << endl << endl;
    // FILL in the code, given that parameter length contains the radius,
    // that will find the circumference to be stored in distance
}
```

Exercise 2: Fill in the following chart by listing the identifiers (function names, variables, constants)

Global	`main`	`main` (Inner 1)	`main` (Inner 2)	`findArea`	`findCircumference`

Exercise 3: For each `cout` instruction that reads:

```
cout << "LIST THE IDENTIFIERS THAT are active here" << endl;
```

Replace the words in all caps by a list of all identifiers active at that location. Change it to have the following form:

```
cout << "area, radius and PI are active here" << endl;
```

Exercise 4: For each comment in bold, place the proper code to do what it says.

Exercise 5: Before compiling and running the program, write out what you expect the output to be. What value for `radius` will be passed by `main` (first inner block) to the `findArea` function? What value for `radius` will be passed by `main` function (second inner block) to the `findCircumference` function?

4 Lab Exercise 2 - Coffee Shop

Jason opened a coffee shop at the beach and sells coffee in three sizes: small (9oz), medium (12oz), and large (15oz). The cost of one small cup is $1.75, one medium cup is $1.90, and one large cup is $2.00. Write a menu-driven program that will make the coffee shop operational. Your program should allow the user to do the following:

1. Buy coffee in any size and in any number of cups.
2. At any time show the total number of cups of each size sold.
3. At any time show the total amount of coffee sold.
4. At any time show the total money made.

Your program should consist of at least the following functions: a function to show the user how to use the program, a function to sell coffee, a function to show the number of cups of each size sold, a function to show the total amount of coffee sold, and a function to show the total money made. All implemented function should **not** return any values so you must use only **call-by-reference** techinques to overcome this restriction. Your program should not use any global variables and special values such as coffee cup sizes and cost of a coffee cup must be declared as named constants.

5 Lab Exercise 3 - Recursive Exponent

Write a program that uses a recursive function `power(base, exponent)` that, when invoked, returns $\text{base}^{\text{exponent}}$. For example, `power(3, 4) = 3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to `1`.
Hint: The recursion step would use the relationship $\text{base}^{\text{exponent}} = \text{base} \times \text{base}^{\text{exponent - 1}}$ and the terminating condition occurs when exponent is equal to 1, because: $\text{base}^1 = \text{base}$

6 Postlab Exercise

e^x can be approximated with a Taylor series: $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{n}{n!}$ where x is any integer and n is a suffieciently large integer, i.e., larger integer means closer approximation of e^x.

Write a program to approximate e^x. The program prompts the user to enter a non-negative integer value for x and the number of terms to be used for expansion n. The program must define the following recursive functions:

- `double factorial(int num)` to recursively calculate the factorial of the passed value.
- `double pow(int x, int pow)` to recusrively calculate x^{pow}.
- `double exp(int x, int n)` to recursively calculate e^x using n terms of the Taylor series.