CMPSC 121: Project 7

Jung-woo Sohn (jwsohn@ist.psu.edu)

November 9, 2017

1 Overview

1.1 Function design

In C/C++, efficient design of functions is important when you build up a large-size application. (Please be sure to go over the textbook chapter on pros/cons of function design in procedural programming.)

A function takes (i) input parameters and (ii) returns a value. Your code in the function body processes the input parameters, executes a desired task, and returns the return value.

For example, consider a `sum()` function that takes an integer value for *input parameter*, calculate the sum from one to the input parameter, and finally returns the sum as the *return value*. One example of `sum()` function design can be:

Listing 1: Design of `sum()` function

```
int sum(int toNum)              // function header
{
    int i = 0, sum = 0;

    for (i = 1; i <= toNum; ++i)
    {
        sum = sum + i;
    }

    return sum;
}
```

The following example can be a more effecient function design since it reduces down cpu computation time for going over the loop `toNum` times. Please notice that the function header remains the same; there is no need of any code modification from the code lines calling `sum()` function.

Listing 2: Design of `sum()` function

```
int sum(int toNum)
{
    return toNum * (toNum + 1) / 2;
}
```

In short, factors to consider for function design are:

- Consider a task you want to implement as a function.
  - The task must be simple. (Ideally, the length of function design should not be more than one page when printed out.)

  - Consider a meaningful, representative name for the function related to the task it performs.
  - Adding multi-line comments on how to use the function is always a good idea since one advantage of using function is increased reusability of your code.

- Consider the input parameters needed and their types. Also consider the final return value and its type.
  - **Variable scope** Exact understanding of the variable scope inside a function is very important. Learn how to declare local variables with the scope effective inside the function.
  - **Variable scope inside `*.cpp` source code files** Similarly, variables declared inside a `*.cpp` source code file has its scope limited to the `*.cpp` file. In this Project, note the scope of variables such as `cities[]`.
  - Note that the only variable that will be *surviving* after the function execution is the return value. All the local variables and input parameters are autotmatically deleted from the memory when the execution of the function is over.

1.2 Function prototypes in `*.h` header files

Functions are like various parts in your garage toolbox. However, when you have a large number of functions in your source code, searching for a function you want to use can be a time-consuming task.

Function prototype (actually a rewrite of the function header) is a way to organize "parts" in C/C++. Instead of listing everything including function header and details in the body, you can have a list of function headers as a *summary* of all the functions. Function prototype is another terminology of referring to this listing of function headers.

One practice in C/C++ programming is that function prototypes are often placed (i) in the beginning of the `*.cpp` source code file, or (ii) in a separate header file with `*.h` file extension.

- As a practice of case (i), we will put the prototypes of functions (implemented in `navigate.cpp` file) at the beginning of `welcome.cpp` file.
- For (ii), you will be given a list of function prototypes from `fheaders.h` file. Given this prototypes, you are supposed to write the implementation details of the functions in `functions.cpp` file.

  When you use a header file to put the function prototypes, make it sure to "include" the header file inside your `*.cpp` file. This is done by putting `#include "header.h"` directive.[1] By using `#include` directive, you are creating an effect of "copy-and-paste" the header file into the `*.cpp` source code.

1.3 Use of functions: calling

Once the function implementations are ready, you can use the functions as you use collected parts for your garage work. We use the term "calling" when you use a function in your source code.

In this Project, you are supposed to call functions you wrote whenever needed. The following Listings of 3 and 4 is an example of calling the `sum()` function.

Listing 3: Calling `sum()` function from `main()`

```
    #include <iostream>
    using namespace std;
```

[1]In other words, whether you use header files or not for your function prototypes, the prototype and function body has to be in the same scope — so that the compiler can recognize them together when compiling.

```
int sum(int toNum)
{
    return toNum * (toNum + 1) / 2;
}

int main()
{
    /* Calling example 1 */
    cout << "Sum from 1 to 100" << sum(100) << endl;

    /* Calling example 2 */
    cout << "Sum from 1 to 1234" << sum(1234) << endl;

    /* Calling example 3 */
    cout << "Sum from 98 to 3432" << ( sum(3432) - sum(98) ) << endl;

    return 0;
}
```

Listing 4: Prototyping of `sum()` function

```
#include <iostream>
using namespace std;

int sum(int toNum);     // NOTE: do not miss the semicolon
                        // int sum(int); is also fine

int main()
{
    /* Calling example 1 */
    cout << "Sum from 1 to 100" << sum(100) << endl;

    /* Calling example 2 */
    int s = sum(1234);
    cout << "Sum from 1 to 1234" << s << endl;

    /* Calling example 3 */
    cout << "Sum from 98 to 3432" << ( sum(3432) - sum(98) ) << endl;

    return 0;
}

int sum(int toNum)
{
    return toNum * (toNum + 1) / 2;
}
```

2 Instructions

2.1 Implementation of various functions in `functions.cpp`

- Bring the source code files from the previous Project 6 into a new programming project.
- Copy and paste the following code listing into a `fheaders.h` file. This header files shows the specifications of the functions to be implemented in `functions.cpp` file.

Listing 5: `fheaders.h` file

```
/* collection of function header prototypes. By including this fheader.h. the
 * code in the cpp file will have access to all the functions specified here.
 * The details of the functions are in functions.cpp file.
 */

using namespace std;

string getCityName(int);
int nextRandomCityIndex(int currentLocation);

int getCurrentLocation();
void setCurrentLocation(int);

int getCriminalLocation();
void setCriminalLocation(int);
void setNextCriminalLocationRandom();

int getArrestCounter();
void decrementArrestCounter();
void resetArrestCounter(int);

int getHours();
void setHours(int);
```

- For the detailed implementations of functions, start with the following template for `functions.cpp` file. Copy and paste the Listing 6 into `functions.cpp` file and start making modifications as needed. Things to note are:
  - Use of vars.
  - How getters and setters are used to access the vars from outside
  - Random number generation

Listing 6: `functions.cpp` file template

```
#include <iostream>
#include <string>
#include <cstdlib>          // for rand()
#include <ctime>            // for srand() and seed number generation for rand()

using namespace std;

const int NUM_CITIES = 4;
string cities[] = {"New York", "London", "Cairo", "Rio de Janeriro"};
```

```
int hours = 0;      // Remaining hours. This is for upcoming projects.

int currentLocation = 0;
int criminalLocation = 0;

int arrestCounter = 3;          // Criminal three turns away


string getCityName(int i)
{
    return cities[i];
}

int getCurrentLocation()
{
    return currentLocation;
}

void setCurrentLocation(int l)
{
    currentLocation = l;
}


/* implement getCriminalLocation() function here */



/* implement setCriminalLocation() function here */



/* No need to modify nextRandomCityIndex(). Use the function as shown. */

int nextRandomCityIndex(int currentLoc)
{
    int nextIndex = 0;

    do
    {
        srand(time(NULL));
        nextIndex = rand() % NUM_CITIES;    // 0, 1, ..., NUM_CITIES - 1
    } while (nextIndex == currentLoc);      // if duplicate, recalculate

    return nextIndex;
}

/* put a proper function header for setNextCriminalLocationRandom() here */
{
    int curLoc = 0, nextLoc = 0;

    curLoc = getCriminalLocation();
    nextLoc = nextRandomCityIndex(curLoc);

    /* How do you update the criminalLocation? Put a proper statement of
       calling setCriminalLocation() here. (HINT: how can you store the value in
       the local variable of nextLoc into criminalLocation? Think in terms of
       input parameters/return value) */
}


/* implement getArrestCounter() function here */



void decrementArrestCounter()
{
    --arrestCounter;
}

void resetArrestCounter(int r)
{
    arrestCounter = r;
}

/* implement getHours() function here */



/* implement setHours() function here */
```

2.2 Use of designed functions

2.2.1 Overview

With all the functions implemented, a number of code lines in our Carmen Sandiego application can be simplified. Before making modifications, let us see what kind of technique we will use for this simplification.

From the Listing 6, note that a number of variables we decleared either in `main()` or `welcome()` (such as `cities[]` and `currentLocation`) are now placed in `functions.cpp`. Then, a problem related to variable scope can be arising: *How can you access all the variables located in `functions.cpp` from `welcome.cpp`?* Obviously, the scope for those variables are limited to `functions.cpp`.

In this project (and particularly for the advanced topic of Object Oriented Programming (OOP)), functions provide the methods to access those variables. (especially functions with `get...()` or `set...()` namings)

But why do we want to go through all of this chore — bury important variables deep down somewhere, design functions as a method to access them, and learn how to properly use those functions to access the variables? This related to the topic of *encapsulation* and abstraction in Object Oriented Programming.

An easy analogy for encapsulation can be like this. Consider a device — like a toaster or a DVD player. Note that the internals of a DVD player is separated from outside with a casing. In fact, access to the electric parts inside are strongly discouraged.

Under this situation, how does a user operate this DVD player? While you cannot access the inside, you can still use buttons on the case, such as play, stop, record, and pause buttons.

With the buttons, note that you do not need to learn the internal working of the parts inside when you press the play button. In other words, the learning cost is simplified to “press the play button when you want to play a DVD disc.”

Not only the users, but also the engineers of the DVD plaer have advantage as well. The internal working of the DVD player model can change at any time; new parts might be added. Several chips might have been integrated into one. However, as long as the engineers can make it play a disc — when the play button is pressed, everything will be fine.

These are details of the simplicity and we want to use this Project for practicing the implementation of the simplicity features, using functions and variable scope. In short, the analogy can be:

- **Inside of case (inside of enclosure):** separated file of `functions.cpp` [2]
- **Internal parts:** variables and some functions declared inside `functions.cpp`
- **Buttons such as play, stop, etc.:** functions (headers) declared in `fheaders.h`
- **Internal working of the parts (when a button is pressed):** function bodies implemented in `functions.cpp`
- **Pressing the buttons:** calling functions from `welcome.cpp` or `navigate.cpp` (or outside of `functions.cpp`)

2.2.2 Example output from a complete program

For this Project, we will implement the following features. Please refer to the output listing below for more information.

- Navigation to a next destination: the program will print out an error message if the detective selects the city he/she is on for the next destination.
- Investiation feature: Based on the criminal’s location, print out a hint message for criminal’s location when the detective choose to investigate around.
- Relocation of criminal: assign the criminal for a new location when the detective correctly selects criminal’s location as the next destination. Use random numbers to assign the criminal a new location.

Listing 7: An example output from a complete program

```
$ ./game
***** Welcome to WHERE IN THE WORLD IS CARMEN SANDIEGO *****

Your name, please: Foobar

Welcome, detective Foobar

You have 7 days and 12 hours to arrest a criminal.

You are currently in New York.

```

[2] `fheaders.h` is in the middle, `welcome.cpp` and `navigate.cpp` are outside.

```
Do you want to investigate around? [y] y

She told me that she had always wanted to see Proms festival.

Please select your next destination:

1. New York
2. London
3. Cairo
4. Rio de Janeiro

Please enter your selection: 2

Your input is : 2
Your next destination is : London
You are currently in London.

Do you want to investigate around? [y] y

She wanted to purchase a Portuguese dictionary.

Please select your next destination:

1. New York
2. London
3. Cairo
4. Rio de Janeiro

Please enter your selection: 3

Your input is : 3
Your next destination is : Cairo
You are currently in Cairo.

Do you want to investigate around? [y] y

Nobody. You’d better visit somewhere else.

Please select your next destination:

1. New York
2. London
3. Cairo
4. Rio de Janeiro

Please enter your selection: 4

Your input is : 4
Your next destination is : Rio de Janeriro
You are currently in Rio de Janeriro.

Do you want to investigate around? [y] y
```

```
She told me that she had always wanted to see Proms festival.

Please select your next destination:

1. New York
2. London
3. Cairo
4. Rio de Janeiro

Please enter your selection: 2

Your input is : 2
Your next destination is : London
```

2.2.3 Modification of `navigate.cpp`

For `navigate.cpp`, we want to add the following features. Refer to Listing 8 for the code template.

- Include directive: In order to use functions defined in `functions.cpp`, the source code file has to have the function prototypes defined in the beginning. Note that we put all the prototypes of the functions in `functions.cpp` in `fheaders.cpp` file.

  Then the question becomes how you can put the `fheaders.h` into `navigate.cpp`. For this purpose, you need to use `#include` directive.

  Note that `#include` directive uses `< >` for including system-wide headers and `" "` for user-created custom headers.

- Modification of `navigate()`: In this code template, note that *you do not have* `string cities[]` *variable any more.* They are now relocated to `functions.cpp` file.

  So how can you access `cities[]` while you do not have any access? Use `getCityName()` function properly to gain access.

- A complete `investigate()` function is provided as an example for how to use `get...` or `set...` functions.

Listing 8: Template for `navigate.cpp`

```
/* include proper headers and fheaders.h here */

/* Question: What is the role of including fheaders.h here? What happends if
   it is missing? */



/* NOTE: no need for further modification of investigate(). However, please
   make it sure to go over what message is returned when the user picked up a
   wrong location for the criminal */
string investigate(int location)
{
    string investigation[] = {
            "She exchanged her money into dollars.",
            "She told me that she had always wanted to see Proms festival.",
            "She checked out books related to pyramids and ancient mysteries.",
            "She wanted to purchase a Portuguese dictionary." };

    string notSeen[] = {
            "Sorry but I have not seen anybody close to your description.",
            "I might have seen the person but it was a long time ago. I do not remember anything.",
            "Nobody. You'd better visit somewhere else.",
            "Well, I can contact you if I happen to see any person like him... or her." };

    if (location == getCriminalLocation())
    {
        setNextCriminalLocationRandom();
            // move the criminal to a different city using a random number index

        return investigation[getCriminalLocation()];
    }
    else
    {
        return notSeen[location];
    }
}

int navigate()
{
    /* declare variables */
    int menu = 0;
    string destination = "";

    /* print destination lists and get user input */

    cout << "Please select your next destination:" << endl;
    cout << endl;

    cout << "1. New York" << endl;
    cout << "2. London" << endl;
    cout << "3. Cairo" << endl;
    cout << "4. Rio de Janeiro" << endl;
    cout << endl;

    cout << "Please enter your selection: ";
    cin >> menu;

    /* Repeat loop when the detective selected the same place for destination */
    while (menu < 1 || menu > 4 || /* FILL IN THE BLANK HERE with a condition */ )
    {
        cout << "You entered a wrong number." << endl;
        if (getCurrentLocation() == menu - 1)   // NOTE how to use getCurrentLocation() here
        {
            cout << "Your destination has to be different from the current location." << endl;
        }

        cout << "Please enter your selection: ";
        cin >> menu;
    }
    /* print out the menu and destination */

    cout << endl;
    cout << "Your input is : " << menu << endl;
    cout << "Your next destination is : " << /* FILL IN THE BLANK */ << endl;
            // menu - 1 because of array indexing starting from zero.
            // Use getCityName() properly instead of cities[menu - 1] here.

    return menu - 1;
}
```

2.2.4 Modification of `welcome.cpp`

Refer to Listing 9 for the code template.

- Similarly for `navigate.cpp`, properly include system-wide header files and/or `fheaders.h` file using `#include` directive.
- To access the functions defined in `navigate.cpp` (such as `navigate()` and `investigate()`), proper function prototypes are needed for `welcome.cpp`. Place the function prototypes at the beginning.
- **Implementation of investigation feature**: in case the user wants to investigate around, call `investigate()` function and display the information about criminal's next destination. If not, skip the investigation.
- Repeat the loop for 4 times for the testing purpose.

Listing 9: Template for `welcome.cpp`

```
/*  Project 7: Description goes here
    Programmer: Your name (abc123@psu.edu)
    Date: 01/01/2015
    (Optional) Assistance with cout statements provided by Rob the tutor
*/

/* Put proper #include directives here. Do you need to include fheaders.h or
   not? */


/* Put proper function prototypes to access the functions in navigate.cpp file */




/* put a proper function header for welcome() here */
{
    int totalHours = 0;
    int cityIndex = 0;
    string name = "";


    /* print welcome screen. Get user name input from keyboard */
    cout << "***** Welcome to WHERE IN THE WORLD IS CARMEN SANDIEGO *****" << endl;
    cout << endl;

    cout << "Your name, please: ";
    cin >> name;


    /* print out the user name */

    cout << endl;
    cout << "Welcome, detective " << name << endl;
    cout << endl;


    /* print out time information */

    totalHours = 7 * 24 + 12;            // 7 and 1/2 days remaining
    /* Store totalHours into hours variable in functions.cpp. Use setHours() * function */


    /* Use getHours() properly for the blanks below */
    cout << "You have " << /* FILL IN THE BLANK */ / 24 << " days";
    cout << " and " << /* FILL IN THE BLANK */ % 24 << " hours to arrest a criminal." << endl;
    cout << endl;


    /* put a proper return statement for detective name here */
}

int main()
{
    string name = welcome();

    int i = 0;

    /* Loop statement for repeating the block below 4 times? */
    {
        int cityIndex;
        /* Store current location number of the detective to cityIndex. Use
           getCurrentLocation() funciton */

        char input;

        cout << "You are currently in " << /* FILL IN THE BLANK */ << "." << endl;
        cout << endl;

        /* investigate on Carmen's destination */
        cout << "Do you want to investigate around? [y] ";
        cin >> input;

        if (input == 'y' || input == 'Y')
        {
            cout << endl;
            cout << investigate(cityIndex) << endl;
            cout << endl;
        }
        else
        {
            cout << endl;
            cout << "Okay, skipping investigations." << endl;
            cout << endl;
        }

        /* select a destination and then update the location */
        cityIndex = /* FILL IN THE BLANK */;

        /* Update the new location stored in cityIndex using setCurrentLocation() here */

    }   // end of loop

    return 0;
}
```