Assignment 06: Functional Decomposition and Using Reference Parameters

COSC 1437: Programming Fundamentals II

Objectives

- Practice building more complex solutions using functional decomposition, where we build small functions and reuse them in other functions.
- Practice creating value returning and void functions.
- Learn to use value parameters, to pass in references to values in memory that when changed will be seen by the caller.
- Continue practicing repetition and selection control statements, and using arithmetic expressions.

Description

In this assignment you will continue to work on writing your own functions. As with the previous assignment, you will have to determine the correct function signature (input parameters, return value and function name), and add this information as a function prototype into the assignment header file, as well as add in an implementation into the `.cpp` implementation file.

For this work we will be emphasizing solving larger problems by using functional decomposition. Functional decomposition is a general approach to solving a large problem by breaking it up into smaller (hopefully easier to solve) sub-problems. If the sub-problems are still too complex, you can continue by breaking them down into sub-sub-problems, etc.

In addition, another major goal of this assignment is to understand how reference parameters work and how they might be useful in some situations in writing code. The default behavior in C/C++ (and many languages) is that when you define a parameter to be passed into a function you are writing, a copy of the value will be made and passed into the function. This is known as pass by value.

But sometimes we want to pass in a reference to our actual variable or instance, so that the function can directly manipulate the data we are working on, not a copy of it. Sometimes we might want to do this for performance reasons. For example we might have an object that is very big, so making a copy of it just so a function can work on the copy might take too much time. But other times, as in this assignment, we want the function we call to do some work or changes on the variable / object we are using directly.

Overview and Setup

For all assignments, it will be assumed that you have a working VSCode IDE with remote Dev Containers extension installed, and that you have accepted and cloned the assignment to a Dev Container in your VSCode IDE. We will start each assignment with a description of the general setup of the assignment, and an overview of the assignment tasks.

For this assignment you have been given the following files (among some others):

File Name	Description
`src/assg06-tests.cpp`	Unit tests for the tasks that you need to successfully pass
`src/assg06-library.cpp`	File that contains the code implementations, all your work will be done here

File Name	Description
`include/assg06-library.hpp`	Header include file of function prototypes

As usual, before starting on the assignment tasks proper, you should make sure you have completed the following setup steps.

1. Copy the assignment repository on GitHub by accepting the assignment using the provided assignment invitation link for ‘Assignment 03’ for our current class semester and section.
2. Clone the repository using the SSH url to your local class DevBox development environment. Make sure that you open the cloned folder and restart inside of the correct Dev Container.
3. Confirm that the project builds and runs, though no tests may be defined or run initially for some assignments. If the project does not build on the first checkout, please inform the instructor.
4. You should create the issue for Task 1 and/or for all tasks for the assignment now before beginning the first task. On your GitHub account, go to issues, and create it/them from the issue templates for the assignment.

Assignment Tasks

Lab Task 1 / In-Lab Work : Calculate nth Factorial

For the first task, we want to ultimately solve a more complex problem to count the number of combinations that occur from taking k items from a set on n items. But to solve this problem, we need a way to be able to calculate factorials of numbers. A factorial in mathematics is written and defined as:

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

Or in other words, the factorial of 5 is the product of 5 times 4 times 3 times 2 times 1 which gives a resulting product of 120. The general expression to calculate the n^{th} factorial is given as:

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \cdots \times n - 1 \times n$$

A general note, the 0^{th} factorial 0! is defined to be 1. and $1! = 1$ as well. Your implementation should return correct results if asked to compute the 0^{th} or 1^{st} factorials.

Write a function named `factorial()`. It will take an integer `n` as its input parameter, that specifies that the n^{th} factorial is to be calculated. This function returns an integer results. You will need to define the function prototype and put into the `assg06-library.hpp` header file, and then put the implementation into the `assg06-library.cpp` file. As usual it is suggested you create a stub function that returns 0 so you can enable the `task1` tests and check that your code is compiling and running tests, before you begin implementing the function.

To complete the work for this task, perform the following steps:

1. You will need a for or while loop and a local variable to keep track of the running product. This will be similar to code you have done previously (for example to calculate a running sum).
2. Make sure that you handle the 0^{th} and 1^{st} factorials cases correctly, these are tested in this tasks tests.
3. The computed factorial product of the first n integers should be returned as the result from your function.

Also you may notice we will stop being quite so specific on the steps you need to perform. You create the loop and perform the running product you need here in several different ways. A for loop might be a good idea. But make sure that the base cases of the 0th and 1 factorial work correctly.

If you successfully complete the function, and save and compile and rerun the tests, you should find that all of the unit tests for Task 1 pass successfully. When you are satisfied your code is correct, create a commit and push your commit to your GitHub classroom repository. It is always a good idea to check the autograder after pushing a commit to ensure it is also compiling and passing the tests for the task you just completed on GitHub.

Task 2: Write Function to Calculate n choose k Combinations

Calculating the number of combinations that can occur if we just k unique items from a set of n is known as the number of combinations n choose k, and is written mathematically as $\binom{n}{k}$. This calculation is useful in many places, for example basic probability. The formula for the number of combinations for n choose k is:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Notice given the set size n and the number of items to be chosen k, that we need to compute 3 different factorials in order to count the total number of combinations. You could work out the formula for this yourself from first principles. For example lets say you have a set of $n = 4$ items, and the items are A, B, C, D. By definition there is only 1 way to choose 0 items from this set of 4 (don't choose any), so $\binom{4}{0} = 1$. Likewise there is only 1 way to choose all 4 items so $\binom{4}{4} = 1$. How many combinations are there for 4 choose 2? The unique combinations when we choose 2 from the set would be $(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)$. Notice that for this measurement, order does not matter so (A, B) and (B, A) are considered the same combination. From enumerating all combinations we see that 6 unique combinations are possible. If we use the formula you should see it agrees

$$\binom{4}{2} = \frac{4!}{2!(4-2)!} = \frac{24}{2 \times 2} = 6$$

So implement a function named `chooseCombinations()` that calculates the number of calculations using the `factorial()` function you implemented in task 1. This function takes two parameters as input, `n` the set size and `k` the number of items being chosen from the set. The function should return the count of the number of combinations using the formula given above.

Do the following steps for this task:

1. It is required that you reuse the `factorial()` function you wrote in Task 1.
   - And in fact you have to reuse this function 3 times here.
2. Perform the calculation to count the number of combinations and return this as the result from this function.
3. Make sure that this function works correctly for $k = 0$ and $k = n$, but if you implemented the `factorial()` function correctly, you shouldn't have to do anything special to get this function to work for those cases.

When finished, save compile and rerun the tests. Once you are satisfied, make and push your commit of Task 2 to your GitHub classroom repository, and check that it compiles and passes tests in the autograder.

Task 3: Function to Swap Values

To practice using reference parameters, you will implement a function that will swap two values in memory for you. In order for this to work as a function, you have to pass in references to the variable locations to the function you write, so that when it swaps the values in the function, it will be swapping the referenced variables, not just copies of the given values.

Swapping two values in C/C++ usually has to be done by declaring a temporary local variable. We won't give the full algorithm/code, but it is easy to find or figure out. You need to copy one of the values to a temporary location. Once you have safely saved one of the values, you can swap the other one into the location that you have saved a copy of. Then you can copy the temporary saved value to the other location to complete the swap.

So for this function create a function named `swap()`. This function will be a `void` function, we don't actually compute and return a results as you have been doing for most of the tasks in the assignments up to this point. Instead you need to pass in two parameters. You will be swapping variables of type integer. So pass in two integer parameters, making sure to pass in the values by reference correctly. Make sure that you declare the prototype for this function in the header file and the implementation in the `.cpp` implementation file before enabling and compiling the `task3` tests. The tests will check that, if your function is given 2 integers, their values have swapped places on return from your swap function.

Perform the following steps task 3:

1. Create a function named `swap()`

2. Make sure the function is a `void` function, all work is returned implicitly by working on the reference parameters given to this function.
3. You must pass in the two integer parameters as reference parameters in order to get this to work in a function.
4. You will need to implement an algorithm to swap the referenced values in your function. The normal way to do this in C/C++ is to declare and use a temporary local variable so you can move the referenced values around successfully.

When finished, save, compile and rerun the tests. Once you are satisfied, make and push your commit of Task 3 to your GitHub classroom repository, and check that it compiles and passes tests in the autograder.

Task 4: A Primitive Sort Using Reference Parameters

For some more practice on both reference parameters and reusing functions, lets create one more function that uses reference parameters. Later we will implement a real sort function in this class. But for this task, we want to write a function that takes in 3 (reference) parameters as input, and then sorts them so that the smallest value ends up in the first parameter, the largest ends up in the third parameter, and the middle value ends up in the middle second parameter.

So for this task, define a function named `threeSort()`. This function is again a `void` function, we will not return an explicit result but instead work is done by acting on its reference parameters. This function should take 3 reference parameters this time, all of type integer.

You are required to reuse your previous `swap()` function to implement the `threeSort()` here. The general approach to do the sort is as follows. Lets call the three parameters `a, b, c` and we consider `a` as the first parameter that should end up with the smallest integer value of the 3 passed in, and `c` is the third that should get the largest value. So to sort these three values:

1. Compare `a` and `b` and if `a` is larger swap it with `b` (using your `swap()` function).
   - As a result of this comparison and possible first swap, you are guaranteed that the larger of the two values originally in `a` and `b` is now in `b`.
2. Then compare `b` and `c` and swap them if `b` is the larger.
   - So from these first two comparisons and potential swaps, you should be able to convince yourself that the largest value has to now be in `c` (though the smallest might not yet be in `a`).
3. Now that the largest is in `c` we can ensure everything is sorted by comparing and swapping `a` and `b` again if needed. So repeat step 1 to compare and swap if `a` is bigger than `b`.

The result of these 3 comparisons and conditional swaps will be that the smallest of the 3 original values is guaranteed to be in `a` and the largest is guaranteed to end up in `c`.

To complete this function do the following steps:

1. Declare the `threeSort()` function with three reference parameters. As should now be usual, create the prototype and put in the header file, and have the implementation in the `.cpp` file.
2. Sort the 3 values by performing the described 3 comparisons and conditional swaps.
   - You are required to reuse your `swap()` function here whenever you need to swap some values.

When finished, save compile and rerun the tests. Once you are satisfied, make and push your commit of Task 4 to your GitHub classroom repository, and check that it compiles and passes tests in the autograder.

Assignment Submission

For this class, the submission process is to correctly create pull request(s) with changes committed and pushed to your copied repository for grading and evaluation. For the assignments, you may not be able to complete all tasks and have all of the tests successfully finishing. This is OK. However, you should endeavor to have as many of the tasks completed before the deadline for the assignment as possible. Also, try and make sure that you only push commits that are building and able to run the tests. You may loose points for pushing a broken build, especially if the last build you submit is not properly compiling and running the tests.

In this assignment, up to 50 points will be given for having completed at least the initial Task 1 / In Lab task. Thereafter 25 additional points are given by the autograder for completing tasks 2 and 3 successfully.

Program Style

At some point you will be required to follow class style and formatting guidelines. The VSCode environment has been set up to try and format your code for some of these guidelines automatically to conform to class style requirements. But not all style issues can be enforced by the IDE/Editor. The instructor may give you feedback in your pull request comments and/or create issues for you for the assignment that you need to address and fix. You should address those if asked, and push a new commit that fixes the issue (or ask for clarification if you don't understand the request). In general the following style/formatting issues will be required for programs for this class:

1. All programs must be properly indented. All indentation must be consistent and lined up correctly. Class style requires 2 spaces with no embedded tabs for all code indentation levels. The editor style checker should properly indent your code when you save it, but if not you may need to check or correct this if code is misaligned or not properly indented.
2. Variable and function names must use `camelCaseNameingNotation`. All variable and function names must begin with a lowercase letter. Do not use underscores between words in the variable or function name. Often function names will be given to you, but you will need to create variables, and maybe some functions, that conform to the naming conventions.
   - Global constants should be used instead of magic numbers. Global constants are identified using `ALL_CAPS_UNDERLINE_NAMING`.
   - User defined types, such as classes, structures and enumerated types should use camel case notation, but should begin with an initial upper case letter, thus `MyUserDefinedClass`.
3. You are required to use meaningful variable and function names. Choosing good names for code items is an important skill. The code examples and starting code tries to give examples of good and meaningful names. In general, do not use abbreviations. Single variable names should be avoided, except maybe for generic loop index variables `i`, `j`, etc. Make your code readable, think of it as writing a document to communicate with other developers (and with your instructor who will be evaluating your code).
4. There are certain white space requirements. In general there should usually never be more than 1 blank line in a row in your code. Likewise there should usually not be more than 1 blank space on a line. There should be 1 blank space before and after all binary operators like `+`, `*`, `=`, `or`.
5. Function documentation is required for all regular functions and all class member functions. You need to follow the correctly formatted Doxygen function documentation format. We will use function documentation generation, and you should be sure your documentation can be built without emitting warnings or errors. Likewise all files should have a file header documentation at the top. You should edit the file header of files where you add in new code (not simply uncommenting existing code). Make sure the information has your correct name, dates, and other information.
6. Practice using proper Git commit messages. You should refer to issues and tasks correctly in commit messages.

Additional Information

The following are suggested online materials you may use to help you understand the tools and topics we have introduced in this assignment.

- Git Tutorials
- Git User Manual
- Git Commit Messages Guidelines
- Test-driven Development and Unit Testing Concepts
- Catch2 Unit Test Tutorial
- Getting Started with Visual Studio Code
- Visual Studio Code Documentation and User Guide
- Make Build System Tutorial
- Markdown Basic Syntax