Brandon Swanson

August 20h 2014

CS 165

Week 9 Assignment Report

Understanding

Subclassing, polymorphism, and templates, quite a way to approach the end of the term. I have been searching for ways to use these techniques in c++ since the beginning of the term, because I have used them in Java and once they become a familiar design paradigm they can be hard to live without. A better understanding of pointers earlier on certainly would have made working with polymorphism easier.

Early on in course I had written a few simple algorithm functions with different versions for each parameter type. And just as the lecture describes, wondered if there was a better way. I had created them by copying the same function and only changing the types in the function signature. When I heard about templates I was sure this must be what I was looking for. Learning about templates and typename/typedef certainly helped me better understand many of the STL container and library functions. Before learning this syntax I knew that containers could hold many different types but had no idea of the real magic behind it or the reason for the special <> syntax when declaring them.

Exercise Components (design && testing && reflection discussed in each)

EMPLOYEE

I do not have a great deal to say about this particular exercise, I think the learning objective was to practice subclassing and specialization. The requested subclass merely added a few extra fields to the class and then needed to modify the print check function and provide a separate printing function. I

decided to model the output of my print function on that of the print check, creating borders with underscores and other characters as well as labeling it as “Employee ID” . This helped in testing because it made the output easy to find and check.

RPG

I posted on Piazza a question about overriding vs re-defining. I was more familiar with environments where all functions are virtual and wondered about the possible pitfalls of re-defining such as member functions that call other member functions, in a polymorphism context this would mean that members of the parent function could be called, when a child class actually had re-definitions.

This RPG assignment is a good example of a behavior that would not be possible with overridden classes. In my class structure each of the 4 classes calculates its damage using the methods of its parents (balrog call demon call creature).

I came up with a pattern I had not yet tried up and till this point but that I think will continue to be useful. I wanted (for the theoretical rpg game) to be able to only return an int value from the get damage function without any extra output, but also wanted a way to log the results of the special damage modifiers.

To this end I added a stringstream member variable to the creature class that could be extracted at any time. This allowed me not only to create a well formated output of damage calculations but also an easy way to trace the method calls to ensure they were using the appropriate chain of child-parent-ancestor.

After testing using this string stream output I still wanted a way to check that the calculations were accurate, as it was necessary to have a lot of outputs the verify the 10% and 5% chances for demons and elves. For that purpose I created an output that showed their average damage after 1000 calls to the method.

I also chose to make the getSpecies functions for the class creature and Demon pure virtual

string getSpecies()=0;

making these two classes abstract types in this implementation. Creature sounds like from its name an abstract class, It could be possible to have a demon type with specialized types for balrog and cyberdemon. The person I paired up with allowed for instances of the creature type. When reading the book this week they presented the employee class as a template for subclassing, but then in the PrintCheck() function for employee it has error output like “should not be called contact your programmer” I was yelling at the pages of my book, they later had a note that this should have been a pure virtual function. It seems like good form that if it is an incomplete type and not all information is available for some of the methods then it should be an abstract type and therefore not possible to have an un-implemented or un-specialized instance of it.

Finally I wonder how to achieve the same behavior that these classes have, the way they use their parent and ancestor methods for getDamage(), in a way that works with polymorphism. In the testing of these classes I had to write separate functions because the methods had to be called on the separate subclasses Elf, Human etc. If I had a container of Creature types holding various humans, elves, balrogs, etc. all held as pointers to creatures, then calling getDamage would only execute the creature method and would not calculate demon damage or elf magic. This would be no good if I was actually designing an RPG I would want to be able to calculate the attacks of a party of any number of any combination of creatures.

One possible way to achieve this kind of behavior would be to have nested functions defined by the parent abstract class. GetDamage() could be an implemented method that after calculating generic creature damage calls the pure virtual function defined in the creature class passing in base damage, and all descendent's could follow the same pattern. This is not an easy to follow call graph though.

TEMPLATE INSERTION SORT

For this I chose to write two separate methods, one that sorts an entire array and another that inserts an element into an array of unknown sized but with the first n elements already sorted. This

way the two functions can be used together on an unsorted array or to fill an array sorting while filling. After comparing with Lisa I think that these functions are destined to be hard to read. It is an algorithm that is not very complicated but if you jump in to coding before writing out the manipulations on paper it can be fraught with errors.

I think that arrays (collections) and templates are an obvious partnership, it makes repeating the same process over items of different kinds easier and more functionally decomposed. I first tested the algorithm on just an array of ints, then when adding other types I realized a template function for the output, sort, output sorted array pattern would save me writing a bunch of different for loops.

Code Trade Project

Review of Lisa Percival's Assignment 8

Initial Reactions

The program seems robust, checking inputs correctly and executes the rules of the game of life properly. I like the simplicity of the error checking in the GetInt function, I was unaware before seeing Lisa's code that the failbit was set on cin upon an incorrect type input (ie alpha or floating point). This was a very elegant and robust way to check for invalid input.

When trying to test the randomly generating worlds I started picking large numbers to try to have a world that didn't result in the immediate death of all the cells. In this I quickly discovered that there was no upper bound range checking on the input for the number of living cells, and large numbers caused the program to hang. To test this problem I calculated the maximum number of cells possible (80*22 = 1760) and tested values near this number, 1759 filled the screen with one empty cell, 1760 filled the screen, and 1761 or anything larger caused the program to hang.

Before looking into your code I suspected this was from an algorithm seeking to randomly choose positions that had not already been chosen. This was indeed the case, on line 182 the for loop increment is decreed if the randomly chosen. In this case checking the input for a maximum value would solve this, using a random selection from a list of decreasing sized would also ensure constant time.

Very good use of functional decomposition to make the program flow and the machinations of the int main() process very easy to follow. This is further helped by the call graph diagrams included in

the report. These graphs make it very easy to see the relationships between the pieces of code.

I was a bit perplexed at first that I was asked for a number of cells to place, then given the option of choosing a shape, and then after the shape was placed the remaining available were randomly distributed. This seemed to present the possibility of interference with the stable patters and interfering with testing. I was curious as to why the shapes couldn't be placed by themselves, and if random was selected then inputing the number of lives cells to place?

Your final implementation does a good job of adhering to the initial design. You established your use of 2d char array, and the general flow of choosing a starting position and advancing through generations. The differences between the two weeks were small implementation details (like modulo formula and death/birth checking) that were born out of testing, as is to be expected.

Your testing plan was very thorough, but as you mentioned in the observed behavior, the operations of your code on a single cell in the corner are hard to observe, and the need to separate the calculations of dying cells and birthing ones was found through watching the behavior of a stable pattern. I would recommend practicing using the computer to tell you more than you can spot on your own, like logging the x,y of all its calculated neighbors.

Grading for week 8

a. Code

i. Implementation A clean, effective, well executed

ii. Error handling B good other than large number of cells

iii. Comments and coding style A+

very well explained functions, and good use of comments

Iv. Extra features B good interface for placing shapes at specific locations

b. Report

I. Understanding A good understanding of program specs and challenges

ii. Design A good adaptation after testing. And good adherence to original design

iii. Testing A used original test plan, documented discovered problems,

iv. Reflection A describes the assumptions made in week 7 and revelations made in week 8

Week 9 Exercise Components

ADMINISTRATOR

- Runs as expected, although why no paystub for the admin I entered in

- Input doesn't check for blank “” input, (hitting return key with nothing entered)

RPG

- It looks like we have different interpretations of the Balrogs double attacks and chances for extra demon damage. I gave them two demon attacks. Others on IRC agreed with yours.
- Good use of getters for private fields, why not use protected though, This structure would leave Creature needing to define the specifics of set-strength() for all classes, might be a good way to do it though.

INSERTION SORT

- you only present test cases for pairs that are duplicate numbers, does it work if they aren't
- the < operator for pair is defined as true if both items are less than the other. The < has sort of strict rules for definition because by defining that operator you can use your custom class in containers like set and map, but they compare equality by swapping the parameters passed in and if both configurations return false then the items are equal. So these < need to be defined in a strict way so that if it returns false in one configuration with items that are not duplicate then it must return true in the other configuration.
- Line 132-136 this while loop could be made into a for loop, No semantic difference but I think they are safer to use when you are using an incremented value
    - for(int j=i ; j>0 && ary[j] < ary[j-1] ; j --){
    - you can have any combinations of && || ! in that second position of the for loop

As always your comments are dynamite and the code for these exercises is well organized in appropriate functions.