CSE 332 Studio Session I on Design Patterns in C++

*Design patterns* provide a vocabulary for identifying and describing both common design problems and known reusable approaches to solving them. In this studio you will be introduced to applying two relevant design patterns (Iterator and Factory Method) to several basic programming problems. Each design pattern has a name, a description of the problem, a solution description, and contextual factors (such as the pattern’s intent) to consider when selecting or applying the pattern. More details about design patterns are at [http://en.wikipedia.org/wiki/Design_Patterns_(book)](http://en.wikipedia.org/wiki/Design_Patterns_(book)) (descriptions of the patterns are linked at the bottom of the page) and [http://en.wikipedia.org/wiki/Design_pattern_(computer_science)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)).

In this studio you will again work in small groups, as before students who are more familiar with the material are encouraged to help those for whom it is less familiar, and asking questions during the studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished please send your answers to the required exercises, and to any of the enrichment exercises you completed, in an e-mail to the cse332@seas.wustl.edu course e-mail account, with the subject line “Design Patterns Studio I”. The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones. Please make sure as you work through these exercises that each team member has a chance to participate actively – e.g., take turns coding, looking up details, debugging, etc., as you work.

**PART I: REQUIRED EXERCISES**

1. Find your team members in the studio area, sit down at/around and log in to one of the Windows machines, open up Visual Studio and create a new Visual C++ Win32 Console Application project. Change the signature of the main function in the source file that Visual C++ generated to match the one that was specified for the previous studios (using `argc` and `argv` as the names of the arguments passed to the main function). Write down the names of the team members who are present (please catch up anyone arriving late on the work, and add their name) as the answer to this exercise.

2. *(Iterator pattern)*. In your main function, declare a vector of strings, and push back all of the strings in the `argv` array into that vector. As the answer to this exercise, describe how your solution applies the Iterator design pattern (specifically, what are the expressions used to (1) start and the beginning, (2) move to the next element in the array, (3) access the current element of the array, and (4) decide when all of the elements in the array have been seen?).

3. Write a for loop that uses the vector’s `[ ]` operator to print out all of strings in the vector on a single line, with spaces in between them. Build your solution, fix any errors or warnings, and run your program to make sure the right information is being printed out when you run the program with different command line arguments (your program should print out all of them including the program name). As the answer to this exercise, show one of the command lines you gave and the output the program produced.

4. Change the type of container used in the program to be a list instead of a vector. As the answer to this exercise, please show the error message that was produced, and say what important difference between the STL vector and list containers is causing this problem.
5. **(Factory Method pattern).** Change the for loop from the previous exercise so that instead of using an integer variable to represent the positions in the container, it instead uses an iterator of the appropriate type for that container (**list<string>**::iterator). **Hints:** The list’s **begin()** method returns an iterator positioned at the first element in the container, and its **end()** method returns an iterator positioned just after the last element in the container; iterators over a list can be compared using operators == and !==; the container element corresponding to an iterator’s position can be obtained using the * (dereference) operator. Build and run your program, and make sure that it produces the same output as in exercise 3. As the answer to this exercise, describe how your solution applies the Iterator design pattern in this case (specifically, what are the expressions used to (1) start at the beginning, (2) move to the next element in the container, (3) access the current element of the container, and (4) decide when all of the elements in the container have been seen?).

6. Modify your solution for the previous exercise so that it again uses a vector instead of a list. Build and run your program and confirm that it produces the same output as in the previous exercise. As the answer to this exercise, describe how (a generic programming variation of) the Factory Method design pattern is used in this solution to make essentially the same code work generically for both vectors and lists.

PART II: ENRICHMENT EXERCISES (Optional)

7. Instead of using an integer to access the elements in **argv**, instead declare a pointer to pointer to char (**char ****) variable and use it to access each of the positions in the **argv** array and push them back into the container. **Hints:** the array name itself (**argv**) can be used as the starting position for the pointer; **argv** plus the number of elements in **argv** can be used as the position just past the last element; dereferencing a pointer to pointer to char gives you a pointer to char, which is what you want to push back into the container. As the answer to this exercise, describe how your solution applies the Iterator design pattern in this case (specifically, what are the expressions used to (1) start at the beginning, (2) move to the next element in the array, (3) access the current element of the array, and (4) decide when all of the elements in the array have been seen?).

8. Modify the for loop that prints out the contents of the container so that it does so in reverse order (from last to first). **Hints:** Both vectors and lists support a reverse iterator type (for example, for a list it could be (**list<string>**::reverse_iterator) that moves in the opposite direction when it’s incremented. The last position in a container that supports reverse iterators is given by its **rbegin()** method and the position just before the first element is given by its **rend()** method. Build and run your program, and confirm that the strings are printed out in the reverse of the order they were given to the program. As the answer to this exercise, please show a command line and your program’s output for it.