**CS 221 Object Oriented Programming and Data Structures**

**Assignment 1**

Due Friday, October 25th by 10 am

Course weighting: 15%

File name: Assignment1\_2015.doc

Last revised: Thursday, 10 October 2024 at 21:10 A10/P10

This assignment requires you to solve three programming problems, and to implement your solution in C++. For each problem provide a main function to test your solution. You will be assessed by your final delivery. This is an individual assignment. No collaboration is permitted. All submitted work should be your own work.

Problems:

1. Implement a struct Coin and a class Purse containing as a data member an STL multiset of Coin instances. A coin has a name ("penny", "nickel", "dime", and "quarter") and a corresponding numeric value.
   1. Implement the following Purse methods:

* add(multiset<coin>)to add the coins from the argument to the purse,
* totalMoney()to return the total value of the coins in the purse
* pay(amount)to remove coins totaling the argument from the purse
* countPennies(), countNickels(), countDimes() and countQuarters()to return the count of the corresponding coin in the purse.
  1. Use the built-in function qsort to sort an array of Purse pointers in two different ways (for example, by the total value of the quarters and by the total number of coins) .
  2. Overload the < operator and use the STL sort function to sort a collection of purses.

[20 points]

1. Change the linked list implementation available in folder N:\CLASS\GanchevG\CS221\Assignment1\List\ in the following way:
   1. In the List class, add a pointer to the last node. For an empty list, this pointer must be NULL.
   2. Modify the constructor, destructor and the existing member functions to properly maintain this pointer and to use it where this would improve program efficiency and/or readability.
   3. Implement a class Iterator suitable for iterating through the nodes of a List object.

* In addition to a pointer to the list, the Iterator class should have two Node pointers, current pointing to the current node in a list, and previous pointing to the previous node in the same list.
* The Iterator class should have member functions:
  + get to return the data field of the current node
  + next that advances the iterator to the next node,
  + previous that moves the iterator to the previous node.
  + equals to compare two iterators
  1. In the List class implement the following additional member functions
* begin to return an iterator with current pointer pointing to the first node in a list, and previous pointer NULL.
* end to return an iterator with current pointer NULL and previous pointer pointing to the last node in the list.
* insert with arguments Iterator and int to insert before the position pointed to by the current pointer of the iterator a node with a data field set to the int argument
* delete with argument Iterator to remove from the list the node pointed to by the current pointer of the iterator. The function delete should return an iterator with current pointer pointing to the node that before the deletion was after the deleted node.

[50 points]

1. Consider the binary search tree implementation available in folder N:\CLASS\GanchevG\CS221\Assignment1\BinSearchTree\
   1. Implement a member function void printLevel(int level)that will print the values of the data members of the nodes which are at the specified level of the tree.
   2. Implement a traversal member function void postorder(Action \* a) for postorder traversal of a binary search tree that carries out an action other than just printing the node data. The action should be supplied as a pointer to an object of a derived class of the class

class Action

{

public:

virtual void act(string str) {}

};

Use the postorder function and a suitable class derived from Action, to find the node with the largest sum of lengths of the strings stored in its children nodes. Print the node data and the sum of the lengths of the strings stored in its children nodes.

[30 points]

**Development Requirements**

1. **Constraints**. Coding must use C++ and generate an executable file. Use streams for input and output.
2. **Dependencies**. You are encouraged to use global constants, but your program must not declare any global variables, whether of primitive data types, user-defined data types, arrays, file streams, or other. No goto statements are allowed.
3. **Standards**. Your programs must meet the programming standards for this course (attached below).

Delivery

All your source code files (.cpp and .h) and any data files (if applicable) must be placed in the directory X:\Dropoff\CS\ganchevg\CS221\Assignment1 in a subdirectory named by your name starting with your last name. Make separate subfolders Problem1, Problem2 and Problem3. You should name the source files containing your main programs Problem1.cpp, Problem2.cpp and Problem3.cpp.   
NOTE: Please do not submit whole Visual Studio projects or solutions!!

Also submit in subfolder Documentation of your submission folder:

1. Grading sheet (supplied at the end of this file) with sections 1, 2 and 3 completed to show what you have done.
2. For each problem:

* A class diagram of the solution
* Optionally, structure diagrams or pseudocode showing the design of the algorithms
* A test plan showing:

1. Check-points, with a clearly indicated result (Y or N)
2. Test data and results in a table with three columns:
   * test input (printed)
   * expected results (printed)
   * the actual results of your testing written in by hand
3. A brief analysis of any known errors, which the program still produces. If there are no known errors, write a statement to this effect.

* Optionally, sample copies of any printed reports produced by your program (if applicable).

Grading Schedule

The assignment will be graded on an A to F scale of grades.

Work which barely meets the minimum requirements and either has problems with usability or readability or does not meet the programming standards will be graded D- to C. Work which shows a useable solution with all the minimum requirements met and meets the programming standards will be graded C+ to B+. Work which in addition demonstrates initiative in design and implementation as evidenced by superior user interaction, additional functionality, robustness and reliability will be graded A- to A. Please fill in the Extensions section of the grading sheet to claim credit for this additional work.

Grading Notes:

1. To get credit for a program feature, it must be coded, tested and documented correctly according to the given standards and be working in all respects.   
   Penalty: points will be deducted for missing documentation items.

## A feature that is either not shown on the test plan as tested or does not work correctly will be given no credit.

## Penalty: points will be deducted for a faulty feature that is shown on the test plan as working.

1. The grading criteria include functionality, non-functional requirements, documentation, and development requirements as indicated on the grading sheet. Pay special attention to method design and internal documentation.

Programming Standards

In the commercial environment, programmers work to the standards of the organization that employs them. Standards usually cover proposals, contracts, analysis and design documents, testing, program documentation, user's guides and project management. In this course we require standards in these specific areas: function/method interface design, program organization, test plans, style of user interaction and internal program documentation.

Function/method Interface Design

Each function/method performs a single task and has a clearly defined interface to users or callers. There is no use of global variables.

Program Organization

A program is organized of modules of three types:

* class definitions placed in header files with file name extension .h (one file per class),
* class implementations placed in .cpp files (one file per class) and
* applications (including a main function) placed in .cpp files.

If a function uses another function or a class, use forward declarations, so that the program can be read in a top-down manner. An application source file may contain the following in this order: function prototypes, global declarations of constants, main program, function definitions.

Test Plans

A test plan is prepared at the design stage to ensure that a program meets every requirement of its specification. It identifies every requirement in the specification and specifies the necessary test or tests to ensure the program meets each requirement. In this course, your test plans should include a list of *check-points* which describe the expected program behavior that does not depend on input data*,* and a table of *test data* with expected results.

At the testing stage, perform the tests and record the actual results on the test plan by circling or checking the check-point result, either Y or N, and indicating whether the expected output was given for the test data.

User Interaction

Prompts and messages should be simple but informative. A user should be provided with a way out of any data entry sequence that is long or difficult to complete. There should be no bad surprises, and definitely no inescapable loops or program crashes!

Internal Documentation

Good internal documentation serves the author and anyone who needs to assess, maintain or work with a program. This is what is expected of your programs:

* Program Heading  
  At the start of each source file is a program heading that identifies the source file's name, the project (assignment), author and the date written. There is a brief description of the file’s purpose in clear and simple English. There is a list of the major items that appear in the source file.
* Naming  
  Variables, constants and functions are given clear and descriptive names. Single-letter identifiers are not acceptable.
* Function/method Documentation  
  Each function/method is documented immediately above its declaration with a brief explanation of its purpose, and the purpose of its parameters and return value.
* Indentation, Layout and Comments  
  Named constants are used in preference to unexplained number or string literals. Indentation is consistent and aids understanding. Each section of the program (function or group of related functions) is separated visually by blank lines or a row of asterisks, etc. Comments are required for global declarations, before every function’s declaration and also where any special language feature or technique is used. Please *do not* comment every line!

Backup and Version Control

You are responsible for disaster recovery of your programs. Keep numbered versions of your source code files, and take back-up copies every day! Lost programs are not an excuse for late delivery.

**Assignment 1 Grading Sheet** Name

1. **Minimum Requirements** *(type your full name)*

*(check the boxes below)*

* Problem 1
* Problem 2
* Problem 3

2. **Extensions***(list extensions completed)*

3. **Documentation submitted**

*(check boxes of items attached)*

* Class Diagrams
* Pseudocode/structure diagrams (optional)
* Test plans
* Source code
* Printed reports (optional)

4. **Non-functional requirements**

- screen layout

- user interaction

- data validation

5. Development Requirements  
- program organization

- dependencies

- data structures

- function/method design

- program layout

- internal documentation

Grade \_\_\_\_