**CS 221 Data Structures**

**Assignment 2**

Due Tuesday December 3rd by 10 am

Total points 90

Course weighting 15%

File name: Assignment2\_2014\_P1\_P2\_P3.doc

Last revised: Wednesday, 20 November 2024 at 11:11 A11/P11

This assignment requires you to solve three programming problems, and to implement your solution in C++. You will be assessed by your final delivery. This is an individual assignment. No collaboration is permitted.

Problems:

1. Modify the Graph class available in folder N:\CLASS\GanchevG\CS221\ Class15\_Graphs\Programs\GraphTraversals\ in the following ways:
2. Modify function dftAtVertex()to perform a complete traversal of a graph starting at the node specified as an actual argument. Perform part (a) before the other parts of Problem 1 and submit it in a separate folder, because the other parts require modifications of readGraph()and the other traversal functions.

[8 points]

1. Modify function readGraph() to read from a file a modified representation of the adjacency lists that after each adjacent vertex number contain the cost of visiting this vertex from the vertex for which this is the adjacency list. The cost will be a number of type double. For example, am input file may contain:

5

0 2 1.5 4 2.5 -999

1 3 0.5 -999

2 3 0.9 4 0.8 -999

3 -999

4 2 0.8 3 2.2 4 0.2 -999

In this example, the vertices adjacent from vertex 0 are 2 and 4, the cost of visiting vertex 2 from vertex 0 is 1.5, the cost of visiting vertex 4 from vertex 0 is 2.5, and no other vertices can be visited from vertex 0.

[8 points]

**For parts c) and d) consider only connected graphs.**

1. Override all traversal functions, in addition to printing the visited vertices, to return the total cost of a traversal.

[8 points]

1. Modify all traversal functions to select from the available non-visited adjacent vertices to visit first the vertex with the smallest cost of the edge to it.

[8 points]

1. Modify class SharedString from folder N:\CLASS\GanchevG\CS221\Class19\_MemoryManagement\ProgramsCh15\SharedStringRenamed\ in the following two ways:
   1. Add a private member function compare(SharedString) that returns -1 or 0 or 1 depending on whether the string is lexicographically less than, equal to, or greater than the argument. Use this function to overload the operators <, ==, and >.

[8 points]

* 1. Implement operator[] in class SharedString. Write this operator so that changes to a character value alter only the target string and not any shared strings. For example:

SharedString a = "Jim";

SharedString b = a;

b[0] = 'f'; // a should still have value Jim

But if the current character is the same, do not make any change.

Provide a test program to test your class.

[12 points]

1. Write a class template for a map implemented by a hash table with a fixed type for the keys – std::string and a template type parameter for the values. Do NOT make the type of the keys a type parameter. Use open hashing (chaining) with an array or vector of STL linked lists (this is a fixed requirement. Do NOT make your own list).

a) Provide the necessary constructors/destructor and following public member functions. Please do not change their names:

* + Function size that returns the number of key-value pairs in the map
  + Function count that returns the number of elements with a specific key
  + Function erase with an argument key k that removes a key-value pair identified with the key k. If k does not match the key of any element in the map, the function should throw an out\_of\_range exception.  
    Example: erase("Harry");
  + Function key\_compare without arguments that returns a “key comparison” object - a function object comparing two keys and returning true if the first argument (a key) is less than the second argument (also a key) in some order chosen by you (for example lexicographical, or by the length of the key, or another order chosen by you) and false otherwise. In a comment specify what is being compared.
  + The default constructor should create an empty map.
  1. Overload the [] operator to return the value associated with the key provided as argument. If the key does not exist, the operator should throw an out\_of\_range exception   
     Examples:

testMap["Harry"] = 2;   
will set the value corresponding to "Harry" to 2.

cout << testMap["Harry"];   
will print 2, which is the value corresponding to the key "Harry" in the map.

If the key "Harry" does not exist, the operator should throw an out\_of\_range exception.

* 1. Provide a function set with arguments string (a key) and the type of the value, that will enter the key-value pair specified by its arguments into the map.   
     Example:   
     testMap.set("Harry", 2)  
     will enter the pair ("Harry",2) into the map. If the key "Harry" already exists, its corresponding value will be set to 2.

d) Overload the operator == .

e) Provide a test program to thoroughly test your template with at least two different types of values.

**Hint:** First implement a class for values of some fixed data type, for example integer or string, and then convert it to a class template. It makes sense to submit both the non-template and the template versions.

**Hint:** For storing key-value pairs you can use our class template Pair which we used in class exercises, or the STL class template std::pair (documented at <https://en.cppreference.com/w/cpp/utility/pair>).

[38 points]

**Development Requirements**

1. **Constraints**. Coding must use C++ and generate a DOS executable. Program should compile and run without errors on the computers in Higgins labs.
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.
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\Assignment2\ in a subdirectory YourName\Code. Please name the folder with your name starting with your last name. You should name the source files containing your main programs problem1.cpp, problem2.cpp etc.   
NOTE: Please do not submit whole Visual Studio projects or solutions!!

You should also submit in the folder X:\Dropoff\CS\ganchevg\CS221\Assignment2\ in a subdirectory YourName\Documentation:

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

* A class diagram or a structure chart showing the usage dependencies between the modules of your program (class diagram if modules are classes, structure chart if modules are functions)
* Optionally, structure diagrams or pseudocode showing the design of the algorithms

1. A test report showing:
2. Check-points, with a clearly indicated result (Y or N)
3. Test data and results in a table with three columns:
   * test input (printed)
   * expected results (printed)
   * the actual results of your testing
4. A brief analysis of any known errors, which the program still produces or a statement that there are no known errors
5. 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 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 or out of order documentation items.

## A feature that is either not shown on the test report 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 report 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 function/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. There is no use of goto statements. There is no use of automatic typing. Please do NOT include <bits/stdc++.h>.

Program Organization

A program is organized of modules of three types: class definitions placed in .h files, class implementations placed in .cpp files and applications (including a main function) placed in .cpp files. If a function uses another function or a class, use forward declarations (function prototypes), 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, forward class declarations, global declarations of constants, main program, function definitions.

Test Reports

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, indicating the check point actual results (by circling or checking the or highlighting either Y or N) and indicating whether the expected output was given for the test data. This converts your test plan to a test report.

At the end of the test report please include a brief analysis of any known errors, which the program still produces or a statement that there are no known errors.

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:

1. 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.
2. Naming  
   Variables, constants and functions are given clear and useful names. Single-letter identifiers are not acceptable.
3. Function/Method Documentation  
   Each function is documented with a brief (one-sentence) explanation of its purpose, and the purpose of its parameters and return value.
4. 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 several blank lines or a row of asterisks, etc. Comments are required for global declarations, at the start of every function and also where any special language feature or technique is used. Comments are *not* required on 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 2 Grading Sheet** Name

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

*(check the boxes below)*

* Problem 1
* Problem 2
* Problem 3
* Problem 4

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

3. **Documentation submitted**

*(check boxes of items attached)*

* Class Diagrams/Structure charts
* Pseudocode/structure diagrams
* Test plans
* Source code
* Printed reports

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 \_\_\_\_