Fall 2022

CPSC2620 Assignment 2

Due: October 11 (Tuesday), 2022 11:55pm

(30 marks)

Please note that due to the potential COVID situation, the requirements of this assignment might be changed and also the compilation and submission procedure might be changed. You will be notified should a situation arise.

Notes:
(*) Please note that no collaboration is allowed for this assignment. Should any plagiarism be identified, all the involved students will get zero for the assignment and will be reported to the Chair of the department for further action. The marker has been asked to check your submissions carefully.
(*) Your submission will be marked on the correctness and readability (including comments) of your program.
(*) Please follow the guidelines to name your files (see below).
(*) Late submission policy: 1 day late: 5% off, 2 days late: 10% off, more than 2 days late: you will get 0 for the assignment.

(*) Write a single Makefile that is used to compile your programs in Questions (1) and (2).

Question (1) (15 marks)

Note: Files to be created for Question (1) are: *dictionary2.cc*, *dictionary2.h* and *testDictionary.cc* (which is used to test the functions in dictionary2.cc). The executable for this question is called *testDictionary*. Please see below for more details.

In this question, you will re-implement the question in Assignment 1 by using vectors instead dynamic arrays.All the functionalities are the same. Of course, you need to make appropriate changes to the function declarations by using vectors (instead of pointers to strings or string arrays).

Also you need to separate the function declarations from the function implementations. (You are starting the journey to OOP.) Function declaration is put into *dictionary2.h* while function implementation is put into *dictionary2.cc*.

Your main function is in a file called *testDictionary.cc*, to test whether your implementation is correct.

For your reference, the functions you need to implement have the following specifications.

// Add aword into the dictionary.
// If aword already exists in the dictionary, display "aword is already added."
// Otherwise, add it to the dictionary and display "aword is added."
void myAdd(vector<string>& dictionary, string aWord);

// Replace oldWord with newWord, if oldWord exists in the dictionary.
// If the oldword does not exist in the dictionary, display "oldword is
// not found.". Otherwise, do the replacement and display 'oldword is
// replaced by newword.'.
void myUpdate(vector<string>& dictionary, string oldWord, string newWord);

// Delete aword from the dictionary.
// If aword does not exist in the dictionary, display "aword is not found."
// Otherwise, delete it from the dictionary and display "aword is deleted."
void myDelete(vector<string>&dictionary,stringaWord);

// Search aword in the dictionary.
// If aword does not exist in the dictionary, display "aword is not found."
// Otherwise, display "aword is found." and display the number of comparisons.
void mySearch(const vector<string> dictionary, string aWord);

// List all the words in the current dictionary, one word one line.
void myList(const vector<string> dictionary);

// List all the words, one on a line, in the dictionary, in the reverse order.
void myRList(const vector<string> dictionary);

// Display "bye bye" and exit to the terminal's prompt.
void myExit();

// uses linear search to look for aword in a list. Returns true
// if found and false if not. Also returns in the "count" parameter the
// number of comparisons needed. pos tells which vector element is
// equal to aword
bool linearSearch(const vector<string> dictionary, int& pos, string aWord, int &count);

Question (2) (15 marks)

Files to be created for Question (2) are: *Matrix.cc*, *Matrix.h*, and *testMatrix.cc* (which is used to test the class Matrix you create). The executable for this question is called *testMatrix*. Please see below for more details.

Write a *CMatrix* class, which represents an *n* x*n*square matrix of integers. Note: A user is going to use your matrix. So to them, the index of row is from 1 to *n* while the index of column is from 1 to *n* as well. But internally, we know that the array index should be from 0 to n – 1. For instance, if the user wants to get the element at (5, 4), this element

actually corresponds to (4, 3) in your 2-dimensional dynamic array in C++. Please pay attention to this.

Your class should include the following member functions. The member variable is called Matrix, which is a 2-dimensional dynamic array, and the member variable Dimension represents the dimension of the matrix.

Constructor: accepts *n* as a parameter and constructs a matrix of the requested size. Initialize all entries to 0.
int getDimension(): returns the dimension of the matrix.
intgetElementAt(int i, int j): returns the element at (*i*, *j*). Use assert to ensure that the indices are within the valid range.
int replaceElementAt(int i, int j, int newint): replaces the element at (*i*, *j*) and return the old element at (*i, j*). Use assert to ensure that the indices are within the valid range.
int setElementAt(int i, int j): set the element at (*i, j*) to be 1 and return the old element at (*i, j*). Use assert to ensure that the indices are within the valid range.
int clearElementAt(int i, int j): set the element at (*i, j*) to be 0 and return the old element at (*i, j*). Use assert to ensure that the indices are within the valid range.
void swapElementsAt(int i1, int j1, int i2, int j2): swaps between the two elements at (*i1*, *j1*) and at (*i2*, *j2*). Use assert to ensure that all the indices are within the valid range.
void resizeMatrix(unsigned int newsize): resizes the matrix from its original size to the *newsize*. We can enlarge the original matrix: we copy the original matrix to the new one and set 0 to all the additional elements in the new one. We can also shrink the original matrix: we copy the elements of the original matrix within the *newsize* to the new one and ignore those beyond the *newsize*.
void addConstant(int constant): adds *constant* to each element in the array.
void readMatrix(): reads n x n elements from the input using *cin*. For example for a 3x3 matrix, the format should be:

Input matrix element at (1, 1): 11
Input matrix element at (1, 2): 2
Input matrix element at (1, 3): 5
Input matrix element at (2, 1): 5
…
Input matrix element at (3, 3): 45
Done reading

void printMatrix(): prints out the elements in the matrix using *cout*. For example, for the above 3 x 3 matrix, the printout would be formatted to:

1	2	5
5	…	…
…	…	45

You are required to use a dynamic 2-dimensional array to implement the matrix. The declaration of the class is put into *matrix.h* while the implementation of the class is put

into *matrix.cc*. Create a test program called *testMatrix.cc*. Your test should show that the member functions of the matrix class work as expected. One good way to do this: create a 4 x 4 array, read elements in it, print it out, change some elements, swap some elements, resize it to 6 x 6, shrink it to 4 x 4, shrink it to 3 x 3, etc. and etc. Each time when you change something in the matrix, just print it out to verify (show) that your program works.

Create two helper functions called *allocateMatrixMemory* and *deallocateMatrixMemory* to dynamically manage the dynamic 2-dimensional array matrixes, as follows. We have discussed the details of those operations in class.

intallocateMatrixMemory(unsigned int n);**
void deallocateMatrixMemory(int M, unsigned int n);**