COL 100. Minor 2 Exam - Practice

Name:

Entry Number:

Group:

Notes:

- Total number of questions: 4. Max Marks: 20
- All answers should be written on the question paper itself.
- The last two sheets in the question paper are meant for rough work. If you run out of space, you can answer questions in this rough space. However, please clearly mention in the answer space for the appropriate question that we should look at the rough space for its answer.
- We will collect the question paper (including the last two sheets meant for rough work). We will not be collecting back any other rough sheets.

1. Write a function *reverseVecString*(*Vector*<*string*> & *names*) which inputs a Vector of strings *names* and achieves the following operations:

   (a) It reverses the order of the strings in the Vector. For example, if the input Vector *names* is {"*abc*","*xyz*","*pqr*"} your program should change *nums* so that new contents of the Vector are {"*pqr*","*xyz*","*abc*"}. You should do this operation in place, i.e., all the changes should happen to the original input Vector *nums* and you should not be creating any new Vector to achieve this functionality. [3 marks]

```
void reverseVecString(vector<string>& names){
    int len = names.size();
    for(int i=0; i<len; i++){
        string temp = names[len-1];
        names.pop_back();
        names.insert(names.begin()+i,temp );
    }
}
```

(b) Modify your function in the part above so that the resulting Vector now contains each string in the reversed order, i.e., in this case, if you start with original Vector {"*abc*","*xyz*","*pqr*"}, the resultant Vector should be {"*rqp*,"*zyx*","*cba*"}. As before you should do this by modifying the original Vector. You are free to write auxiliary (helper) functions for this part. [3 marks]

```
void reverseString(string& name){
    int len = name.size();
    for(int i=0; i<len; i++){
        char temp = name[len-1];
        name.erase(name.end() -1);
        name.insert(name.begin()+i,temp );
    }
}

void reverseVecString(vector<string>& names){
    int len = names.size();
    for(int i=0; i<len; i++){
        string temp = names[len-1];
        names.pop_back();
        reverseString(temp);
        names.insert(names.begin()+i,temp );
    }
}
```

2. Write a function $HasIncreasingSubSeq(Vector{<}int{>}\ nums, intk)$ which inputs a Vector of integers *nums* and a number k, and returns true if there is a contiguous subsequence of *nums* which is non-decreasing and whose length is at least k. For example, if the input Vector is $\{12, 4, 5, 7, 10, 2, 13\}$ and $k = 4$, your function should return *true* (due to the presence of contiguous subsequence $\{4, 5, 7, 10\}$. On the other hand, the function should return *false* if the same Vector as earlier is input but $k = 5$. You should appropriatley handle the case when k is greater than the size of the Vector. [4 marks]

```
bool HasIncreasingSubSeq(vector<int> nums, int k) {
    int num1, num2;
    int subseqlen = 1;
    if (k > nums.size()) return false;
    for(int i= 0; i< nums.size()-1; i++){
        num1 = nums[i];
        num2 = nums[i+1];
        if(num1 <= num2)
            subseqlen++;
        else
            subseqlen = 1;
        if(subseqlen == k) return true;
    }
    return false;
}
```

3. Write a function *multiply(Grid<int> A, Grid<int> B)* which inputs two Grids A, B defined over integers and returns a new Grid C which is the matrix multiplication of A and B. For example, if A is of size $m \times n$, B is of size $n \times p$, then the resultant Grid C should of be of size $m \times p$ such that $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$ where C_{ij} denotes the ij^{th} element of matrix C. Similarly, A_{ik} and B_{kj} denote the ik^{th} and kj^{th} elements of A and B, respectively. Your program should return an empty matrix if the dimensions of A and B are such that they can not be multiplied with each other. What is the complexity of your function in terms of m, n, p? You should express your answer using the O notation covered in the class not worry about the constants or lower order terms [5 marks]

```
Grid<int> multiply(Grid<int> A, Grid<int> B){
  int A_row = A.numRows();
  int A_col = A.numCols();
  int B_row = B.numRows();
  int B_col = B.numCols();
  Grid<int> C;
  if(A_col != B_row) return C;
  C.resize(A_row, B_col);
  for(int i=0; i<A_row; i++){
      for(int j=0; j<B_col; j++){
          int temp =0;
          for(int k=0; k<A_col; k++){
              temp += A[i][k] * B[k][j];
          }
          C[i][j] = temp;
      }
  }
  return C;
}
```

Complexity of this function is $O(m * n * p)$, where grid A is of size m x n and grid B is of size n x p.

4. Write a program which inputs two strings from the user: $filename$ and *word*. Your program should then create a input stream over file named $filename$, read the contents of this file, and selectively write those lines which contain one or more occurrences of *word* in them. For example, if the file has the following content:

```
col100 is going well.
some more practice will help.
it is good to know so many students are doing col100.
```

and if the *word* is "col100", then your output file should contain the following lines:

```
col100 is going well.
it is good to know so many students are doing col100.
```

You can write your output to a file named "out.txt". Note that you will need to create a file output stream over "out.txt" to which you should selectively write the contents as described above. Do not forget to close the streams after you are done processing them. [5 marks]

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>

using namespace std;

int main(){
    string word, filename;
    cin >> filename;
    cin >> word;
    ifstream inputfile;
    inputfile.open(filename.c_str(), std::ifstream::in);

    if(!inputfile.is_open())
    {
      cout << "Error: Cannot open " << filename << endl;
      return 0;
    }
    ofstream outputfile;
    outputfile.open("out.txt", std::ifstream::out);
    if(!outputfile.is_open())
    {
      cout << "Error: Cannot open " << "out.txt" << endl;
      return 0;
    }
    string line;
    while (getline(inputfile, line)) {
      istringstream line_ss(line);
      string file_word;
      while(line_ss >> file_word)
      {
        if(file_word.compare(word) == 0)
        {
          outputfile << line << endl;
          break;
        }
      }
    }
    inputfile.close();
    outputfile.close();
}
```

Rough page 1

Rough page 2