University of Eswatini

Department of Computer Science

Final Main Examination: December 2018

Title of paper : Computer Programming II

Course Number : CSC213/CS244

Time Allowed : Three (3) hours

This paper may not be opened until permission has been granted by the invigilator

INSTRUCTIONS

1. Answer all questions in section A.
2. Answer only one (1) question in section B.
3. This exam consists of 11 printed pages including the cover page.
4. The Exam user_id, password, tree, context and server name will be provided by the chief invigilator.
5. Read the complete question paper carefully before starting to work on the problem.
6. Write pseudo codes (hand-written) in the provided answer folder.
7. Submit written answer folder and zipped project folder
8. Use the last 10 minutes to check your submissions
9. The names of all your files(project, source file and output files) should have following format

   S------(Project Name)

   S------.cpp (source file)

   S------.TXT (data files)

   The dashes in file names are the six digits of your UNESWA student identity number.

SPECIAL REQUIREMENTS:

1. For each student, a standalone PC with working Visual Studio 2010 C++ compiler.
2. Students **should not** have access to the internet.

ANSWER FORMAT

1. Where required, write (in your answer folder) a detailed pseudo-code.
2. Compile and test your code. Make sure you submit code with no syntax errors. Where necessary comment statements that have syntax errors.
3. Provide sufficient comment in your source code.
4. Output from your program must be properly formatted.

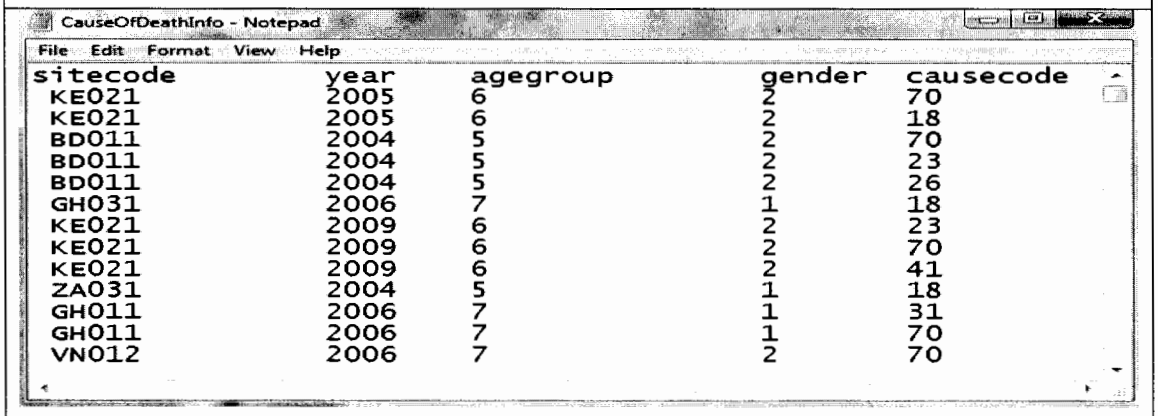
DATA

1. The required data text files, and ANNEX_A source files, are stored in the folder EXAM2018_CSC213_DATA_ANNEX and will be provided by the chief invigilator.
2. Except where instructed, the data files and the ANNEX source files should not be modified. However, where necessary content can be used in your program.

PROBLEM:

The task is to design a program which can be used to extract and analyse information about the causes of death from three separate files (**CauseOfDeathInfo.txt, CauseData.txt** and **SiteData.txt**). The data is based on verbal autopsy (VA) interviews contributed by fourteen different Health Demographic Surveillance System (HDSS) sites in sub-Saharan Africa and eight sites in Asia. Each HDSS site is committed to long-term longitudinal surveillance of circumscribed populations, typically each covering around 50,000 to 100,000 people. Households are registered and visited regularly with a rate varying from once to several times per year. The given files contain vital events which were registered at each of such visits, and any deaths recorded are followed up with verbal autopsy interviews which can be used to inform probably cause of death. The program must read, combine and extract required information from the three files. The figure that follows shows sample content of the files.

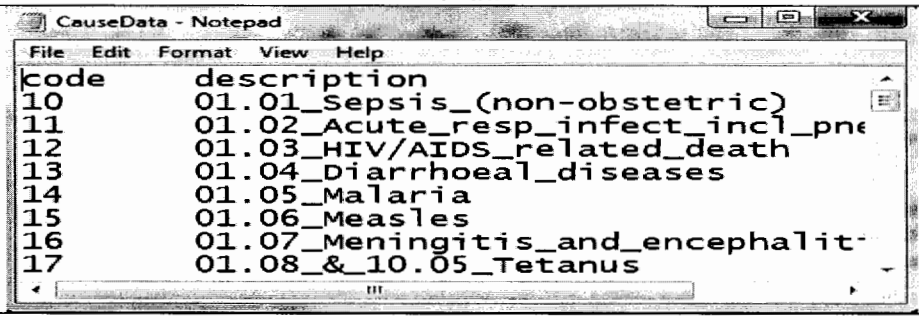
CauseOfDeathInfo.txt



sitecode	year	agegroup	gender	causecode
KE021	2005	6	2	70
KE021	2005	6	2	18
BD011	2004	5	2	70
BD011	2004	5	2	23
BD011	2004	5	2	26
GH031	2006	7	1	18
KE021	2009	6	2	23
KE021	2009	6	2	70
KE021	2009	6	2	41
ZA031	2004	5	1	18
GH011	2006	7	1	31
GH011	2006	7	1	70
VN012	2006	7	2	70

Notes: This file contains 176834 records and may take a longer time to process. For testing purposes, a smaller sample of the file called **train_cod_data.txt** (with only 1000 records) is provided in the data folder.

CauseData.txt



SiteData.txt

SiteData - Notepad

File Edit Format View Help

code	country	name
BD011	Bangladesh	ICDDR-B_:_Matlab
BD012	Bangladesh	ICDDR-B_:_Bandarban
BD013	Bangladesh	ICDDR-B_:_Chakaria
BD014	Bangladesh	ICDDR-B_:_AMK
BF031	Burkina_Faso	Nouna
BF041	Burkina_Faso	Ouagadougou
CI011	Ivory__Cost	Taabo
ET031	Ethiopia	Awlaelo
GH011	Ghana	Navrongo
GH031	Ghana	Dodowa
GM011	The_Gambia	Farafenni
ID011	Indonesia	Purworejo
IN011	India	Ballabgarh
IN021	India	Vadu
KE011	Kenya	Kilifi
KE021	Kenya	Kisumu
KE031	Kenya	Nairobi
MW011	Malawi	Karonga
SN011	Senegal	Bandafassi
VN012	Vietnam	Filabavi
ZA011	South_Africa	Agincourt
ZA031	South_Africa	Africa_Centre

For each of the given fields in the cause of death text file (sitecode, year, agegroup, gender and causecode), the program must generate and output a token frequency table and a summary of statistics similar to figure shown below. The example given below is for the **year** field values;

C:\Users\user\Desktop\CSC213_STUFF\CSC213_EXAM_DEC2018\Dec2018_SampleSol_v4\Debug\D...

```
==========================================
CAUSE OF DEATH ANALYSIS BY : YEAR OF DEATH
==========================================
Value      #Cases          Percent
1992       292             0.2  %
1993       527             0.3  %
1994       491             0.3  %
1995       581             0.3  %
1996       543             0.3  %
1997       495             0.3  %
1998       1708            1.0  %
1999       1774            1.0  %
2000       4148            2.3  %
2001       4567            2.6  %
2002       5056            2.9  %
2003       13303           7.5  %
2004       16681           9.4  %
2005       17418           9.8  %
2006       18134           10.3 %
2007       17232           9.7  %
2008       19923           11.3 %
2009       20434           11.6 %
2010       19375           11.0 %
2011       11660           6.6  %
2012       2492            1.4  %

SUMMARY STATISTICS
------------------
UNIQUE TOKEN COUNT    =  21
MEAN                  =  8420.67
STANDARD DEVIATION    =  8057.73
MINIMUM VALUE         =  292
MAXIMUM VALUE         =  20434
```

SECTION A

(Compulsory – Answer all questions)

QUESTION 1 – 45 marks

Based on the definition of **TokenCount** record/structure provided in Annex A, and assuming all token frequency tables are stored as a list of token count records, [for instance **std::list<TokenCount> TokenFrequencyTable**] , write suitable code to perform the following tasks which will lead to a possible solution to the given problem.

(a) Write a function to calculate the probability of a token. The function only takes 2 integer numbers as arguments, the frequency/count of a specific token and total count of all tokens, and returns the ratio of these two numbers expressed as a percentage. **[3 marks]**

(b) Write a function that takes a token frequency table (list of token count records) as an argument and computes the sum of all the token counts/frequencies. **[5 marks]**

(c) Using the function obtained in (b), write a function that takes a token frequency table (list of token count records) and computes the mean/average of all the token counts or frequencies. The mean is simple the sum of all token counts divided by the number of unique tokens. **[5 marks]**

(d) Write a function that takes a token frequency table (list of token count records) as an argument, and returns the standard deviation of the token counts or frequencies. Note that given a list of **n** values, say ($x_1, x_2, \ldots, x_n$), with the mean/average of the values denoted as $\bar{x}$, the standard deviation, denoted s, can be calculated using the following formula: $s = \sqrt{\frac{\Sigma_{i=0}^{n} (x_i - \bar{x})^2}{n-1}}$. **[12 marks]**

(e) Write a function that takes a token frequency table (list of token count records) as an argument, and returns the minimum recorded count/frequency. **[5 marks]**

(f) Write a function that takes a token frequency table (list of token count records) and returns the maximum recorded count/frequency. **[5 marks]**

(g) Using the functions defined in previous tasks, complete the missing code in the function **showTokenFrequencySummaryStats** provided in ANNEX A. **[5 marks]**

(h) Write a function called **evaluateTokenProbability** that iterates through all token records in a token frequency table and calculates the token probability. This function uses the probability function defined in task 1 above. Here is the suggested prototype for this function.

void evaluateTokenProbability (std::list<TokenCount>& TokenFrequencyTable)

[5 marks]

(i) Test your functions from previous tasks using a randomly generated frequency table as shown in the main function example below. The getSampleTokenFrequencyTable function generates a random frequency table and is provided in ANNEX A folder.

```
int _tmain(int argc, _TCHAR* argv[])
{
        // Testing functions using a randomly generated frequency table
        std::list<TokenCount> FT = getSampleTokenFrequencyTable ();
        displayTokenFrequencyTable(FT, std::cout);
        std::cout << std::endl;
        evaluateTokenProbability(FT);
        displayTokenFrequencyTable(FT, std::cout);
        showTokenFrequencySummaryStats(FT, std::cout);
        system("pause");
}
```

QUESTION 2 – 25 marks

(a) Whereas the testing code in Question 1 uses a randomly generated token frequency table, we instead want to extract this information from the cause of death text file. Therefore, write a function called **extractTokensFromCODFile** that takes two arguments (a cause of death text file and a field selector integer value) and extracts all tokens of the selected field from the given cause of death text file to a token frequency table. For instance, the **CauseOfDeathInfo**.txt contains 5 fields namely: sitecode, year, agegroup, gender and causecode which will be referencedas fields 1, 2, 3, 4 and 5 respectively. When the field selector = 1, the function extracts only the **sitecode** labels to a token frequency table. Similarly, when the field selector = 2, the function extracts only the **year** labels to a token frequency table. The same applies to all the other fields. **All fields are to be treated as strings (labels) not numbers**. The function must return a list of token counts. Write proper pseudocode for this function. Here is a recommended prototype for this function:

std::list<TokenCount> **extractTokensFromCODFile** (char* codFilename, int fieldSelector)

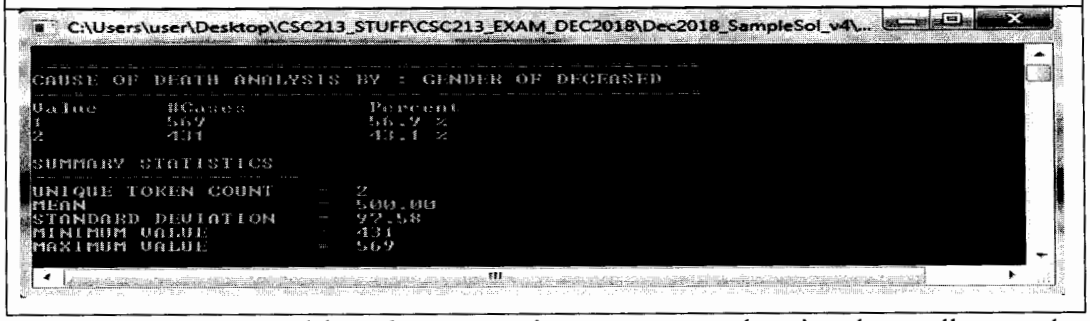
Whereas the function reads all the field values from an input line, it only extracts to the frequency table the appropriate field value as per the field selector. For each new token, a new token count record, with frequency = 1 and probability=0, is created and inserted into the frequency table. For an existing token, the frequency count is incremented by 1.

[pseudoce(8) + actual code(12) = **20 marks**].

(b) Test your functions using the first version of the **displaySelectiveAnalysis** function provided in ANNEX A. The function can be called in a main function similar to example that follows (if necessary, you can change function names). **[5 marks]**

```
int _tmain(int argc, _TCHAR* argv[])
{   //Testing
    int fieldSelector = 4; //by gender
    displaySelectiveAnalysis(std::cout, fieldSelector);
     return 0;}
```

Expected Results:



For testing purposes and in order to save time, test your code using the smaller sample **train_cod_data.txt** input file. That is use the first version of the **displaySelectiveAnalysis** function. You will eventually need to test your code using the much bigger **CauseOfDeathInfo.txt** input file, and may need the second version of the **displaySelectiveAnalysis** function which allows you to specify any filename. The sample results shown in the figure above are based on the bigger **CauseOfDeathInfo**.txt text file

SECTION B

(Answer only one(1) question from this section)

QUESTION 3 – 30 marks

Based on the code obtained in question 2, change the test code in the main function such that it uses an interactive menu-based user interface. The program must repeatedly display the menu until the exit option is chosen. Write the pseudo code for your main function in the answer folder.

```
MAIN MENU
1. Full analysis
2. Selective analysis
3. Exit .

Enter your choice (1-3) :
```

- **Option 1** – The Full Analysis option writes a report to a text file, say **full_report.txt**, containing the analysis (token frequency tables and summary statistics) for each of the five fields (sitecode, year, agegroup, gender, and causecode). In short, the displaySelectiveAnalysis function is called repeatedly with field selector values from 1 to 5.

```
SELECTIVE SUB MENU
1. Cases per Site
2. Cases per Year
3. Cases by Age Group
4. Cases by Gender
5. Cases by Cause of Death
6. Return to main menu

Enter your choice (1-6) :
```

- **Option 2** – The Selective analysis option in-turn presents a selective submenu as shown in the figure. When options 1,2,3,4 and 5 are selected, the displaySelectiveAnalysis function is called to display (on the screen/standard output) a corresponding token frequency table and summary statistics. Option 6 returns control to the main menu
- **Option 3** - exits the program

[pseudocode(10) + correct interface(30) = **30 marks**]

QUESTION 4 – 30 marks

(a) Define a site record structure, **called Site**, to store site information. That is the structure must have fields for the site code, country and name **[3 marks]**

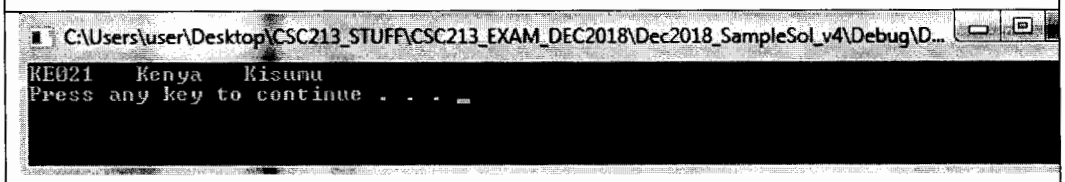
(b) Define a function called **initSiteData** that initializes a site record. The function takes as an argument, a site record (as described above), and values for each fields. The function simple sets the record fields to the given values. Here is a suggested prototype of the function. **[4 marks]**

Here is a sample of how the function could be used and tested in the main function

Sample testing code

```
Site S; //declare site record
initSiteData (S, "KE021",  "Kenya","Kisumu"); // initialize site record
std::cout << S.code << "\t" << S.country << "\t" << S.name << std::endl;
```

Expected result



(c) Write a function, called **getSiteData,** that extracts specific site details from a given site data text file. The function arguments include the site code and the name of the text file containing the site information, say **SiteData.txt**. It returns a site record. In the example below the site code is **KE021**. The function only extracts a single matching record from the given site data text file. In your answer folder write the pseudocode for this function. Here is a suggested prototype for the function: `Site getSiteData (const char* sitecode, char* siteDataFilename)`

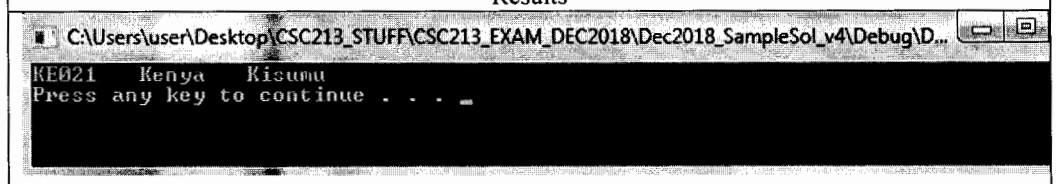
[pseudocode(6) + actual code(14) =**20 marks**]

Here is a sample call to the function and expected results

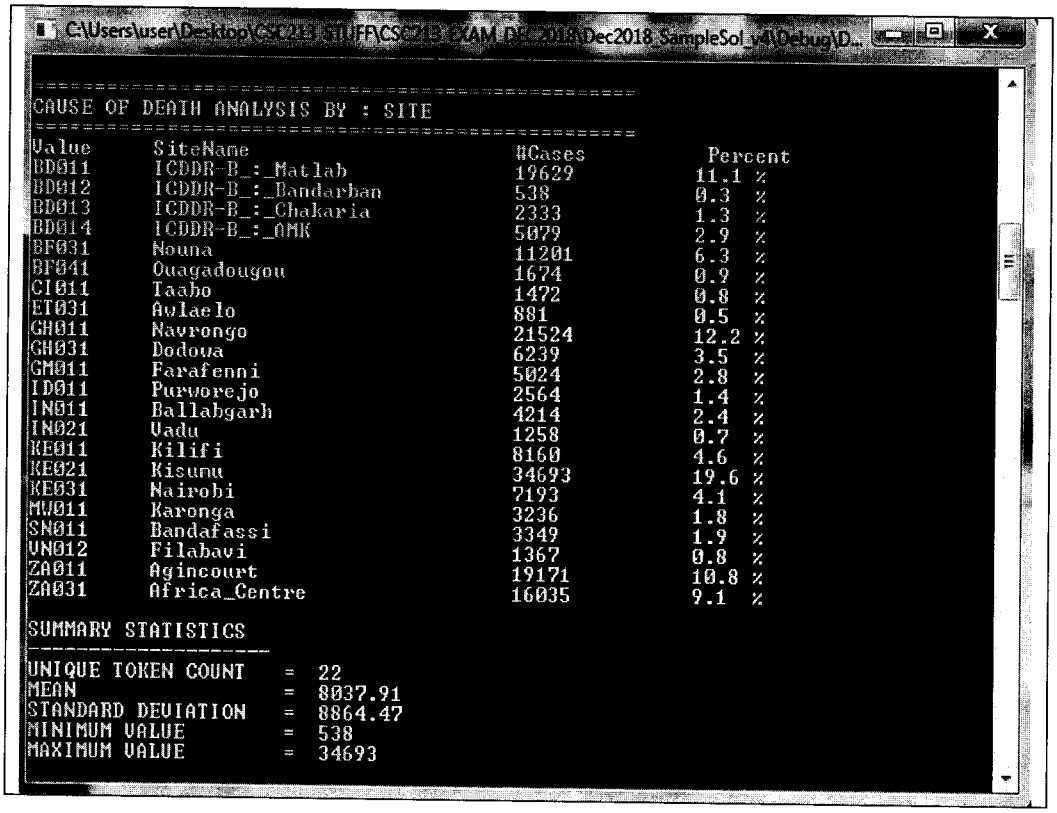
Sample code

```
std::string siteCode = "KE021";
Site S = getSiteData (siteCode.c_str(), "sitedata.txt");
std::cout << S.code << "\t" << S.country << "\t" << S.name << std::endl;
```

Results



(d) Modify the **displayTokenFrequencyTable** function such that when the field selector (a parameter for this function) is 1 (site), the function calls the getSiteData function to display the site name in addition to the site code, frequency and probability. Modify the displaySelectiveAnalysis function such that it calls the revised **displayTokenFrequencyTable** function. The expected results when field selector =1 is as shown below. **[3 marks]**



ANNEX A:

(This code is provided in the data folder: EXAM2018_CSC213_DATA_ANNEX_A)

```
#include "StdAfx.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>

//need to include list STL
#include <list>
#include <iterator>


//struct for storing token count
struct TokenCount{
        std::string Value;
        int Frequency;
        double Prob;
};

// overload less than operator
bool operator<(const TokenCount& lhs , const TokenCount& rhs) {
   return (lhs.Value < rhs.Value) ; // compares token records by Value field

}

//get token frequency table size = number of unique tokens
int getTokenFrequencyTableSize(std::list<TokenCount>  TokenFrequencyTable)
{
        return TokenFrequencyTable.size();
}


//display token frequency table
void displayTokenFrequencyTable(std::list<TokenCount> TokenFrequencyTable,
                                                std::ostream& os)
{
        //loop through all Token Count records in the Token Frequency Table
        // and write values to an given output stream
        os<<std::left<<std::setw(10)<<"Value"<<std::setw(15)
                                        <<"#Cases" <<std::setw(5)
                                            <<"Percent" << std::endl;

        for(std::list<TokenCount>::iterator it = TokenFrequencyTable.begin();
                                        it != TokenFrequencyTable.end();  it++)
                os<<std::setw(10)<< it->Value
                  << std::setw(15)<< it->Frequency
                  <<        std::fixed << std::setw(5) << std::setprecision(1)
                  <<    it->Prob << "%"<< std::endl;
}


//display summary statistics
void showTokenFrequencySummaryStats(std::list<TokenCount>  TokenFrequencyTable,
                                                                std::ostream& os)
{
os << "\nSUMMARY STATISTICS " << std::endl;
os << "---------------------------------------------- " << std::endl;
os << "UNIQUE TOKEN COUNT   =  " << "ADD APPROPRIATE CODE HERE " << std::endl;
os << "MEAN                 =  " << "ADD APPROPRIATE CODE HERE " << std::endl;
os << "STANDARD DEVIATION   =  " << "ADD APPROPRIATE CODE HERE " << std::endl;
os << "MINIMUM VALUE        =  " << "ADD APPROPRIATE CODE HERE " << std::endl;
os << "MAXIMUM VALUE        =  " <<  "ADD APPROPRIATE CODE HERE " << std::endl;
}
```

```
//display report header
void displayReportHeader (std::ostream& os, int fieldSelector=1)
{
 os << "\n================================================" << std::endl;
 os << "CAUSE OF DEATH ANALYSIS BY : ";
 if (fieldSelector ==1)
            os << "SITE " << std::endl;
 else if (fieldSelector ==2)
            os << "YEAR OF DEATH" << std::endl;
  else if (fieldSelector ==3)
            os << "AGE GROUP AT DEATH" << std::endl;
  else if (fieldSelector ==4)
            os << "GENDER OF DECEASED" << std::endl;
   else if (fieldSelector ==5)
            os << "CAUSE OF DEATH" << std::endl;

 os << "================================================" << std::endl;
}

//perform selective analysis from train_cod_data.txt –FIRST VERSION
void displaySelectiveAnalysis(std::ostream& os, int fieldSelector=1)
{
        displayReportHeader(os, fieldSelector);
        std::list<TokenCount> FT = extractTokensFromCODFile ("train_cod_data.txt",
                                                                fieldSelector);
        evaluateTokenProbability(FT);
        displayTokenFrequencyTable(FT, os);
        showTokenFrequencySummaryStats(FT, os);
}

//perform selective analysis from any cause of death file –SECOND VERSION
void displaySelectiveAnalysis(char* codFilename, std::ostream& os, int fieldSelector=1)
{
        displayReportHeader(os, fieldSelector);
        std::list<TokenCount> FT = extractTokensFromCODFile (codFilename, fieldSelector);
        evaluateTokenProbability(FT);
        displayTokenFrequencyTable(FT, os);
        showTokenFrequencySummaryStats(FT, os);
}

//get token frequency Table size
int getTokenFrequencyTableSize(std::list<TokenCount>  TokenFrequencyTable)
{
        return TokenFrequencyTable.size();
}


//function to generate sample frequency table for testing
std::list<TokenCount> getSampleTokenFrequencyTable ()
{
        //declare list of token count
        std::list<TokenCount> TokenFrequencyTable;

        //declare token count record
        TokenCount TC;
        for (int year = 1992; year < 2012; year++)
        {

                //*****************************
                //convert year int to string
                std::ostringstream stream;
                stream << year;
                std::string year_str = stream.str();
                //*****************************
                TC.Value = year_str;
                TC.Frequency =  (rand() % 99) + 1;//randomly generated values
                TC.Prob =0;
                TokenFrequencyTable.push_back(TC);
        }
        TokenFrequencyTable.sort();
        return TokenFrequencyTable;
}
/********************************** END ANNEX_A ************************************************/
```