

University of Eswatini
Department of Computer Science
Final Main Examination: December 2019

Title of paper : Computer Programming II
Course Number : CSC213
Time Allowed : Three (3) hours

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

INSTRUCTIONS

1. Answer Question 1 in section A.
2. Answer only one (1) question in section B.
3. The examination is marked out of 80 marks
4. This exam consists of 15 printed pages including the cover page.
5. The Exam userId, password, tree, context and server name will be provided by the chief invigilator.
6. Read the complete question paper carefully before starting to work on the problem.
7. Write pseudo codes (hand-written) in the provided answer folder.
8. Submit your written answer folder (for pseudocode) and zipped project folder as instructed by the invigilator.
9. Use the last 10 minutes to check your submission. It remains your responsibility to make sure everything is submitted accordingly.
10. 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 EXAM2019_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 STATEMENT

The task is to design a program which can be used to extract and analyse information about the Boston (USA) criminal records. The data is stored in two separate files (**criminal_offense_info.csv** and **bostoncrimeinfo.csv**).

The datasets contain records of crime incident reports, which includes a reduced set of fields focused on capturing the type of incident as well as when and where it occurred. Crime incident reports are provided by Boston Police Department (BPD) to document the initial details surrounding an incident to which BPD officers respond. Records in the new system begin in June of 2015. The dataset has 327, 821 observations (rows) and 17 variables (columns). These variables are:

Variable	Description
INCIDENT_NUMBER	A unique incidence report number
OFFENSE_CODE	Criminal offense code
OFFENSE_CODE_GROUP	Criminal offense group
OFFENSE_DESCRIPTION	Description of criminal incidence
DISTRICT	The district in which incident occurred.
REPORTING_AREA	The district area that reported the incident.
SHOOTING	An indicator(Y or N) of whether gun shots were fired during the incident.
OCCURRED_ON_DATE	Date and time of the incident
YEAR	The year the incident occurred. In this dataset the data was collected between 2015 and 2018.
MONTH	The month in which incident occurred.
DAY_OF_WEEK	The day (Monday to Sunday) of the week in which incident occurred.
HOUR	The hour (0-23)of the day in which incident occurred
UCR_PART	Unique crime reporting (UCR) part or category. Could be Part One, Part Two or Part Three.
STREET,	The street of the crime occurrence
LATITUDE	Latitude
LONGITUDE	Longitude
LOCATION	A pair of latitude and longitude.

The data may be used to answer questions such as: How has crime changed over the years? Is it possible to predict where or when a crime will be committed? Which areas of the city have evolved over this time span? In which area are most crimes are committed? Which day, or time of the day, are crimes most likely to be committed?

For purposes of this examination, a simplified and reduced dataset will be used. The data is stored in two separate files as explained below.

1. **criminal_offense_info.csv** – This file contains 222 criminal offense records using only 3 variables; namely OFFENSE_CODE, OFFENSE_CODE_GROUP and UCR_PART.
2. **bostoncrimeinfo.csv** – contains 327, 821 observations (rows) and 9 variables (columns). The variables are: INCIDENT_NUMBER, OFFENSE_CODE, DISTRICT, REPORTING_AREA, SHOOTING, YEAR, MONTH, DAY_OF_WEEK and HOUR.

To speed up code development and debugging, a smaller version of this file, called **train_bostoncrimeinfo.csv** is provided. It is therefore recommended that you first test your code using the smaller dataset.

Samples of the CSV files are shown below:

INCIDENT_NUMBER	OFFENSE_CODE	DISTRICT	REPORTING_AREA	SHOOTING	YEAR	MONTH	DAY_OF_WEEK	HOUR
I152056250	111	A1	112	N	2015	7	Wednesday	12
I162064334	111	A1	NA	N	2016	8	Wednesday	18
I172041830	111	A1	124	Y	2017	5	Saturday	2
I172068628	111	A1	122	Y	2017	8	Sunday	0
I182026852	111	A1	61	N	2018	4	Tuesday	22
I182071662	111	A15	37	N	2018	9	Thursday	3
I152078371	111	A7	24	N	2015	9	Sunday	10
I162002573	111	A7	16	Y	2016	1	Sunday	1
I162022418	111	A7	14	Y	2016	3	Tuesday	22
I162047273	111	A7	23	N	2016	6	Wednesday	8
I162066422	111	A7	21	N	2016	8	Wednesday	6
I162104752	111	A7	NA	N	2016	12	Saturday	22
I172102399	111	A7	35	Y	2017	12	Sunday	16

OFFENSE_CODE	OFFENSE_CODE_GROUP	UCR_PART
111	Homicide	Part One
112	Manslaughter	Other
121	Manslaughter	Other
123	Manslaughter	Other
301	Robbery	Part One
311	Robbery	Part One
315	Robbery	Part One
334	Robbery	Part One
335	Robbery	Part One

For selected variables in the Boston crimes dataset, the program must generate and output a token frequency table and a summary of statistics similar to figure shown below. The variables of

interest are: 1=OFFENSE_CODE; 2 = DISTRICT; 3 = SHOOTING; 4 = YEAR; 5 = MONTH; 6 = DAY_OF_WEEK; 7 = HOUR). For example, the figure below shows the summary frequency table and summary statistics for the **YEAR** variable.

```

=====
BOSTON CRIME ANALYSIS BY : YEAR
=====
Value      #Cases      Percent
2015       53392       16.3 %
2016       99134       30.2 %
2017       100938      30.8 %
2018       74356       22.7 %

SUMMARY STATISTICS
-----
UNIQUE TOKEN COUNT = 4
MINIMUM FREQUENCY RECORD:
    Value: 2015      Frequency: 53392      Prob: 16.3 %
MAXIMUM FREQUENCY RECORD:
    Value: 2017      Frequency: 100938     Prob: 30.8 %

```

SECTION A

(Compulsory – Answer all questions)

QUESTION 1 – 40 marks

Create a new visual studio C++ project called **S<yourid>** and copy the code provide in ANNEX A into the main source file. Based on the definition of **TokenCount** record/structure provided in ANNEX A, and assuming all token frequency tables are declared as a list of token count records, write suitable code to perform the following tasks which will lead to a possible solution to the given problem. Where possible, call other functions already defined or provided in ANNEX A.

- (a) Write a function that takes a token frequency table as an argument (list of token count records) and computes the mean/average of all the token counts/frequencies. **[5 marks]**
- (b) Write a function that takes a token frequency table as an argument, and returns a token count record (a **TokenCount**) which has the minimum recorded count/frequency. **[5 marks]**
- (c) Write a function that takes a token frequency table and returns a token count record which has the maximum recorded count/frequency. **[5 marks]**
- (d) Complete the missing code in the function **showTokenFrequencySummaryStats** provided in ANNEX A. **[5 marks]**
- (e) Run the main statements provided in ANNEX A and verify that the sample frequency table and summary statistics are displayed. The **getSampleTokenFrequencyTable** function generates a random frequency table and is provided in ANNEX A. Generate results for **DAY_OF_WEEK**, take a screen shot and save results in a file called **yourid_Q1_e.png**. Save this file in your test folder. **[5 marks]**

- (f) Modify the `displayTokenFrequencyTable` function such that it displays the total frequencies and probabilities at the end of the frequency table as shown in sample output below. You will note that the probability column may not add up to 100%. How is that possible, and how can it be corrected? [5 marks]

```

C:\Users\HP\Desktop\TEACHING 2020\CSC213_AUGUST_2019\EXAM_2019_2oct\Dec2019SampleS...
BOSTON CRIME ANALYSIS BY : YEAR
=====
BOSTON CRIME ANALYSIS BY : YEAR
=====
Value      #Cases      Percent
2015       42          16.0 %
2016       54          20.6 %
2017       98          37.4 %
2018       68          26.0 %
=====
Total      262         98 %
=====

```

- (g) Explain how the `displayTokenFrequencyTable` function could be modified such that it displays the `OFFENSE_CODE_GROUP` and `UCR_PART` in addition to the `OFFENSE_CODE` value, frequency and probability as shown in sample below. Write the pseudocode for the revised function. [10 marks]

```

C:\Users\HP\Desktop\TEACHING 2020\CSC213_AUGUST_2019\EXAM_2019_2oct\Dec2019SampleSol_v5\Debug\Dec2019SampleSo...
BOSTON CRIME ANALYSIS BY : OFFENSE_CODE
=====
Offense_Code  Offense_Group      UCR_Part  #Cases  Percent
1001          Counterfeiting     Part Two   91      0.8 %
1002          Counterfeiting     Part Two   11      0.1 %
1102          Fraud              Part Two   32      0.3 %
1105          Fraud              Part Two   33      0.3 %
1106          Confidence Games   Part Two   59      0.5 %
1107          Fraud              Part Two   77      0.7 %
1108          Fraud              Part Two   24      0.2 %
1109          Fraud              Part Two   46      0.4 %
111           Homicide           Part One   42      0.4 %
112           Manslaughter       Other      54      0.5 %
1201          Embezzlement      Part Two   51      0.4 %
121           Manslaughter       Other      98      0.9 %

```

SECTION B

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

QUESTION 2 – 40 marks

- (a) Whereas the testing code in Question 1 uses a randomly generated token frequency table that is generated using the **getSampleTokenFrequencyTable**, we instead want to extract the token count information from the Boston crime dataset. Therefore,
- (i) write a pseudo code for a function that takes two arguments (Boston crime CSV file and a field selector integer value) and extracts all token count of the selected field from the given Boston crime file to a token frequency table. For instance, the **Bostoncrimeinfo.csv** file contains 9 fields but we want to generate frequency tables for only the following fields of interest: **OFFENSE_CODE**, **DISTRICT**, **SHOOTING**, **YEAR**, **MONTH**, **DAY_OF_WEEK** and **HOUR**. We shall reference these fields as 1, 2, 3, 4, 5, 6 and 7 respectively. For instance, when the field selector = 1, the function extracts only the **OFFENSE_CODE** labels to a token frequency table. The same applies to all the other fields. The function must return a list of token counts. [10 marks]
- (ii) Based on your pseudo code from (i) above write a function called **extractTokensFromBCDFile**. All fields are to be treated as strings (labels) not numbers. [20 marks]

Test your functions using the first version of the **displaySelectiveAnalysis** function provided in ANNEX A.*[remove comment first]*. To save time, consider using the smaller **train_bostoncrimeinfo.csv** dataset. The function can be called in a main function similar to example that follows (if necessary, you can change function names).

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

- (b) Rewrite the **displaySelectiveAnalysis** function such that it takes the name of any file as an additional argument. The code is the same as in the earlier version, except this version can read from the provided filename (i.e. **bcdFilename**) instead of only from **train_bostoncrimeinfo.csv**. Test your revised version of the function as shown below. You should expect to get the same result as in (a). [5 marks]

```
int _tmain(int argc, _TCHAR* argv[])
{
    //Testing

    int fieldSelector = 4; //by YEAR
    char* bcdFilename = "train_bostoncrimeinfo.csv"; //train dataset
    displaySelectiveAnalysis(bcdFilename, std::cout, fieldSelector);
    return 0;
}
```

- (c) Using the revised version of the **displaySelectiveAnalysis** function (from (b) above), generate results for SHOOTING variable from the larger boston crime **bostoncrimeinfo.csv** file. Take a screen shot and save results in a file called **yourid_Q2_e.png**. Save this file in your test folder. [5 marks]

QUESTION 3 – 40 marks

Based on the code obtained in question 1 or 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. You can test your menu system using a token frequency table generate by the **extractTokensFromBCDFile** from question 2 or the **getSampleTokenFrequencyTable** used in question 1.

MAIN MENU

1. Full analysis
2. Selective analysis
3. Exit .

Enter your choice (1-3) :

- **Main Menu 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 seven fields of interest (**OFFENSE_CODE**, **DISTRICT**, **SHOOTING**, **YEAR**, **MONTH**, **DAY_OF_WEEK** and **HOUR**). In short, the **displaySelectiveAnalysis** function is called repeatedly with field selector values from 1 to 7.

SELECTIVE SUB MENU

1. Cases per Offense Code
2. Cases per District
3. Cases by Shots Fired
4. Cases by Year
5. Cases by Month
6. Cases by Day of Week
7. Cases by Hour
8. Return to main menu

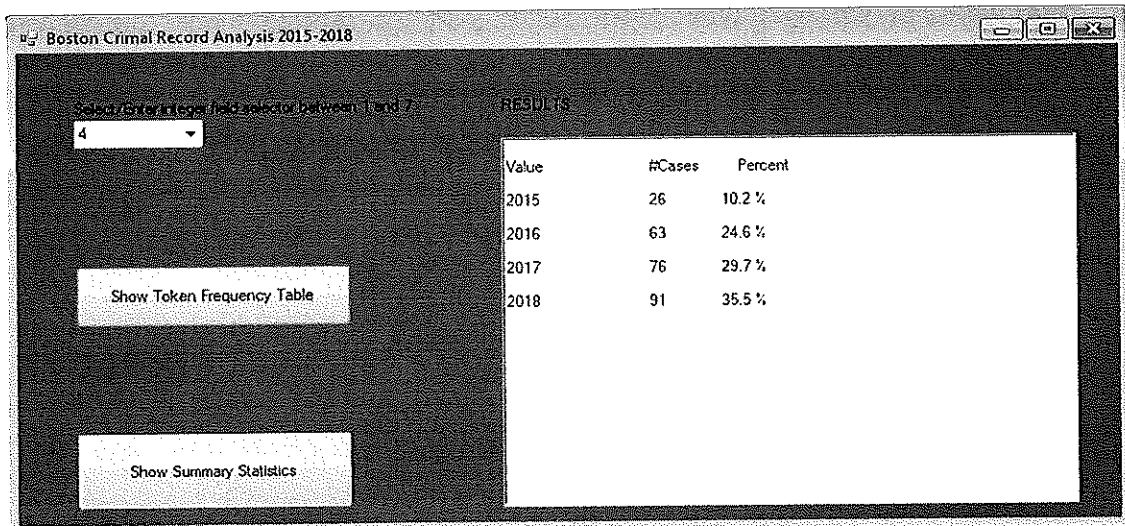
Enter your choice (1-8) :

- **Main Menu Option 2** – The Selective analysis option in-turn presents a selective submenu as shown in the figure. When options 1 to 7 are selected, the **displaySelectiveAnalysis** function is called to display (on the screen/standard output) a corresponding token frequency table and summary statistics. Option 8 returns control to the main menu
- **Option 3** - exits the program


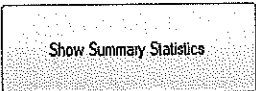
[pseudocode(10) + correct interface(30) = 40 marks]

QUESTION 4 – 40 marks

- (a) In your test folder, create a windows forms application called **YourId_Q4_GUI**.
- (b) Design a graphical user interface (GUI) that allows the user to view frequency tables and summary statistics for the Boston crime dataset. The design should be similar to the figure shown below. The default field selector should be 4 (YEAR) and the results should be similar to the figure shown below. However, values may not be the same if token frequency table is randomly generated. [10 marks]



- (c) Based on the code provided in ANNEX A, and sample token frequency tables generated using the **getSampleTokenFrequencyTable** function, add event handlers for each of the buttons as explained below. You will need to place some of the code provided in ANNEX A into an appropriate header (.h) files in the project. [25 marks]

Button	Event handling
	When the user clicks the Show Token Frequency Table, the frequency table for the selected field must be displayed on the display box on the right. The code is similar to the statement in the displayTokenFrequencyTable function provided in ANNEX A, except that output is written to the display box instead of an output stream.
	When the user clicks the Show Summary Statistics button, the summary statistics for the selected field must be displayed on the display box on the right. The code is similar to the statement in the showTokenFrequencySummaryStats function provided in ANNEX A, except that output is written to the display box instead of an output stream.
<p>Note: The two buttons may not produce same results since the token frequency tables are randomly generated -- Reading the tokens from a file as in question 2 can resolve this problem.</p>	

ANNEX A:

(This code is provided in the data folder: EXAM2019_CSC213_DATA_ANNEX_A)

```
//ANNEX A : CODE –
//NOTE : DOCUMENTATION COMMENTS REMOVED TO REDUCE NUMBER OF PAGES
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>

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

// need to manipulate string using algorithms
#include <string>
#include <algorithm>

/*****
 *          USEFUL STRUCT/FUNCTION DEFINITIONS          *
 *****/

/*****
 * Record/structure definition to store token count information
 *****/
struct TokenCount{
    std::string Value;
    int Frequency;
    double Prob;
};

void initTokenCount (TokenCount& TC, std::string val="", int freq=0, double prob = 0.0)
{
    TC.Value = val;
    TC.Frequency = freq;
    TC.Prob = prob;
}

std::ostream& operator<< (std::ostream& os, TokenCount TC)
{
    os << "Value: " << TC.Value
        << "\t Frequency: " << TC.Frequency
        << "\t Prob: " << TC.Prob << " %"
        << std::endl;
    return os;
}

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

int getTokenFrequencyTableSize(std::list<TokenCount> TokenFrequencyTable)
{
    return TokenFrequencyTable.size();
}

void displayTokenFrequencyTable(std::list<TokenCount> TokenFrequencyTable,
                                std::ostream& os)
{
    //loop through all Token Count records in the Token Frequency Tables
    // and write values to an output stream
    os<<std::left<<std::setw(60)<<"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(60)<< it->Value
    << std::setw(15)<< it->Frequency
    <<      std::fixed << std::setw(5) << std::setprecision(1)
    << it->Prob << "%"<< std::endl;
}

void displayReportHeader (std::ostream& os, int fieldSelector=1)
{
    os << "\n===== " << std::endl;
    os << "BOSTON CRIME ANALYSIS BY : ";

        if (fieldSelector == 1)
            os << "OFFENSE_CODE" << std::endl;
        else if (fieldSelector == 2)
            os << "DISTRICT" << std::endl;
        else if (fieldSelector == 3)
            os << "SHOOTING" << std::endl;
        else if (fieldSelector == 4)
            os << "YEAR" << std::endl;
        else if (fieldSelector == 5)
            os << "MONTH" << std::endl;
        else if (fieldSelector == 6)
            os << "DAY_OF_WEEK" << std::endl;
        else if (fieldSelector == 7)
            os << "HOUR" << std::endl;

        else
            os << "INVALID FIELD SELECTED" << std::endl;

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

int FieldSelectorWidth (int fieldSelector=1)
{
    if (fieldSelector == 1) // OFFENSE_CODE;
        return 15;
    else if (fieldSelector == 2) // DISTRICT;
        return 10;
    else if (fieldSelector == 3) // SHOOTING;
        return 10;
    else if (fieldSelector == 4) // YEAR;
        return 10;
    else if (fieldSelector == 5) // MONTH;
        return 10;
    else if (fieldSelector == 6) // DAY_OF_WEEK;
        return 15;
    else if (fieldSelector == 7) // HOUR;
        return 10;
    else
        return 20;
}

void displayTokenFrequencyTable(std::list<TokenCount> TokenFrequencyTable,
                                std::ostream& os, int fieldSelector=1)
{
    //loop through all Token Count records in the Token Frequency Tables
    // and write values to an output stream
    os<<std::left<<std::setw(FieldSelectorWidth(fieldSelector))<<"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(FieldSelectorWidth(fieldSelector))<< it->Value
        << std::setw(15)<< it->Frequency
        <<      std::fixed << std::setw(5) << std::setprecision(1)
        << it->Prob << "%"<< std::endl;
}

void displayPleaseWaitText (int fieldSelector, char* inputTextFilename="")
{

```

```

std::cout << "extracting ";

        if (fieldSelector == 1)
            std::cout << "OFFENSE_CODE" << std::endl;
        else if (fieldSelector == 2)
            std::cout << "DISTRICT" << std::endl;
        else if (fieldSelector == 3)
            std::cout << "SHOOTING" << std::endl;
        else if (fieldSelector == 4)
            std::cout << "YEAR" << std::endl;
        else if (fieldSelector == 5)
            std::cout << "MONTH" << std::endl;
        else if (fieldSelector == 6)
            std::cout << "DAY_OF_WEEK" << std::endl;
        else if (fieldSelector == 7)
            std::cout << "HOUR" << std::endl;
        else
            std::cout << "UNKNOWN FIELD " << std::endl;

std::cout << "tokens from " << inputTextFilename << ".txt" << " text file " << std::endl;
std::cout << "..... PLEASE WAIT" << std::endl << std::endl;
}

int getTotalTokenCount (std::list<TokenCount> TokenFrequencyTable)
{ int sum =0;
  for(std::list<TokenCount>::iterator it = TokenFrequencyTable.begin(); it != TokenFrequencyTable.end(); it++)
      sum = sum + it->Frequency;
  return sum;
}

double calculatePercentage (int count , int total)
{
    return count*1.0/total*100;
}

void evaluateTokenProbability (std::list<TokenCount>& TokenFrequencyTable)
{
    int totalCount = getTotalTokenCount (TokenFrequencyTable);
    for (std::list<TokenCount>::iterator it = TokenFrequencyTable.begin(); it != TokenFrequencyTable.end(); it++)
        it->Prob = calculatePercentage (it->Frequency , totalCount);
}

void showTokenFrequencySummaryStats(std::list<TokenCount> TokenFrequencyTable, std::ostream& os)
{
    os << "\nSUMMARY STATISTICS " << std::endl;
    os << "-----" << std::endl;
    os << "UNIQUE TOKEN COUNT = " << "....ADD CODE HERE" << std::endl;
    os << "MEAN FREQUENCY = " << "....ADD CODE HERE" << std::endl;
    os << "MINIMUM FREQUENCY RECORD:" << std::endl;
    os << "\t" << "....ADD CODE HERE" << std::endl;

    os << "MAXIMUM FREQUENCY RECORD:" << std::endl;
    os << "\t" << "....ADD CODE HERE" << std::endl;
}

/*
void displaySelectiveAnalysis(std::ostream& os, int fieldSelector=1)
{
    displayReportHeader(os, fieldSelector);
    std::list<TokenCount> FT = extractTokensFromBCDFile ("train_bostoncrimeinfo.csv", fieldSelector);
    evaluateTokenProbability(FT);
    displayTokenFrequencyTable(FT, os, fieldSelector);
    showTokenFrequencySummaryStats(FT, os);
}

std::list<TokenCount> getSampleTokenFrequencyTable (int fieldSelector=2)

```

```

{
//declare list of token count
std::list<TokenCount> TokenFrequencyTable;

switch (fieldSelector)
{
    case 1://OFFENSE_CODE
    { //declare token count record
        TokenCount TC;

        //store site values in array
        std::string Offense[] = {
            "111","112","121","123","301","311","315","334","335","338",
            "339","349","351","361","371","381","402","403","404","413",
            "423","432","511","520","521","522","527","530","540","541",
            "542","547","560","561","562","611","612","613","614","615",
            "616","617","618","619","623","624","627","629","633","634",
            "3803","3805","3807","3810","3811","3820","3821","3830","3831"
        };

        for (int index = 0; index < 222; index++)
        {
            TC.Value = Offense[index];
            TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

            TC.Prob =0;
            TokenFrequencyTable.push_back(TC);
        } //end for loop
        break;
    } //end case 1
    case 2://DISTRICT
    { //declare token count record
        TokenCount TC;

        //store site values in array
        std::string District[] = {"E18","D14","B2","A1","A7","C11","D4","E13","B3","C6","A15","E5"};
        for (int index = 0; index < 12; index++)
        {
            TC.Value = District[index];
            TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

            TC.Prob =0;
            TokenFrequencyTable.push_back(TC);
        } //end for loop
        break;
    } //end case 2
    case 3://SHOOTING
    { //declare token count record
        TokenCount TC;

        //store shooting flags values in array
        std::string SHOOTIN_FLAG[] = {"N","Y"};
        for (int index = 0; index < 2; index++)
        {
            TC.Value = SHOOTIN_FLAG[index];
            TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

            TC.Prob =0;
            TokenFrequencyTable.push_back(TC);
        } //end for loop
        break;
    } //end case 3
    case 4://YEAR
    { //declare token count record
        TokenCount TC;
        for (int year = 2015; year <= 2018; 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 generate frequency/count

        TC.Prob =0;
        TokenFrequencyTable.push_back(TC);
    }
    //end case 4

case 5: //MONTH
{ //declare token count record
    TokenCount TC;
    for (int monthNumber = 1; monthNumber <= 12; monthNumber++)
    {
        //*****
        //convert monthNumber int to string
        std::ostringstream stream;
        stream << monthNumber;
        std::string monthNumber_str = stream.str();
        //*****
        TC.Value = monthNumber_str;
        TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

        TC.Prob =0;
        TokenFrequencyTable.push_back(TC);
    }
    //end case 5

case 6: //DAY_OF_WEEK
{ //declare token count record
    TokenCount TC;

    //store day of week values in array
    std::string WeekDay[] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
                            "Friday", "Saturday"};

    for (int index = 0; index < 7; index++)
    {
        TC.Value = WeekDay[index];
        TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

        TC.Prob =0;
        TokenFrequencyTable.push_back(TC);
    }
    //end case 6

case 7: //HOUR
{ //declare token count record
    TokenCount TC;
    for (int hour = 0; hour <= 23; hour++)
    {
        //*****
        //convert hour int to string
        std::ostringstream stream;
        stream << hour;
        std::string hour_str = stream.str();
        //*****
        TC.Value = hour_str;
        TC.Frequency = (rand() % 99) + 1;//randomly generate frequency/count

        TC.Prob =0;
        TokenFrequencyTable.push_back(TC);
    }
    //end case 7

} //end switch
//sort and return list of token records
TokenFrequencyTable.sort();
return TokenFrequencyTable;
}

```

```
/****** END STRUCT/ FUNCTION DEFINITIONS *****/
```

```
// SAMPLE MAIN
int _tmain(int argc, _TCHAR* argv[])
{
    //field selector
    /*****
    if 1 = OFFENSE_CODE; 2 = DISTRICT; 3 = SHOOTING; 4 = YEAR;
        5 = MONTH; 6 = DAY_OF_WEEK; 7 = HOUR;
    *****/
    int fieldSelector = 4;

    //generate sample frequency table
    std::list<TokenCount> FT = getSampleTokenFrequencyTable (fieldSelector);

    //evaluate probabilities and display again
    evaluateTokenProbability(FT);
    displayReportHeader(std::cout, fieldSelector);
    displayTokenFrequencyTable(FT, std::cout, fieldSelector);

    //show token frequency stats
    showTokenFrequencySummaryStats(FT, std::cout);

    system ("pause");
    return 0;
}
```