**Andrew Tran**

**Lab Experience Eight**

**Objectives:**

1. Using the random number generator for prediction.
2. Utilizing file stream objects.

**Random Number Generation**

To utilize the random number generator in C++ you need to include the following header files:

#include <ctime>

#include <cstdlib>

In addition you will be using the following functions:

srand(time(NULL)) ------ Used to seed the random number generator and only needs to be executed once.

rand() ------ Returns a random number from 0 to RAND\_MAX

Examine your PPT notes on how to generate random numbers in a numeric range.

**File stream objects**

All programs in use today never ask the user for any information, but access the information from an external file. At this time only sequential files will be discussed. A sequential file means to access the data in the order that it has been written to the file. That is, to access record number 6 in the file you have to process records 1 through 5 first.

A sequential file in C++ is a text file created from notepad, another C++ program, or through the full screen editor of the IDE. If you use a word processor to create the file, then you need to save it as a text file. To work with files the following steps are always followed:

1. The file must be opened for either input or output.
   1. If the file is opened for input and does not exist the stream will enter a fail state and all further references to the stream will be ignored. This is a fatal error and the program should be terminated.
   2. If the file is opened for output and does not exist, it will be created. If the file already exists the previous contents will be destroyed permanently.
2. Data is either read or written to the file or both.
3. The file must be closed when the program is finished using the file. This is important when writing to a file since data is written to a memory buffer and once the file is full it is then written to the file. It is possible the buffer is partially full and when the file is closed what remains in the buffer is written to the file.

To use files in a program the fstream header file must be included. fstream means file stream and used to create the programmer-defined file stream objects.

Example:

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

The next step in the process is to declare the file stream objects. This is accomplished by using by using one of the following data types:

|  |  |
| --- | --- |
| File Stream Data Type | Description |
| **ofstream** | Output file stream. Used to declare a programmer-defined identifier that will be associated to an output file. That is, the program will create a file or files. |
| **ifstream** | Input file stream. Used to declare a programmer-defined identifier that will be associated to an input file. That is, the program will use a file already created. |
| **fstream** | File stream. Used to declare a programmer-defined identifier that will be associated to an input or output file. Will not be used at this time but will be discussed later in the course. |

Examples:

// declares an identifier that will be used to write to a file, no file is associated

// with the identifier at this time.

ofstream outfile;

// declares an identifier that will be used to read data from a file, no file is

// associated with the identifier at this time.

ifstream infile;

To associate a file to each of the identifiers on the previous page the open method is used. Example:

infile.open(“funstuff.dat”); // infile is now associated to the file funstuff.dat. So to

// extract data from funstuff.dat the identifier infile

// will be used within the program. Note: funstuff.dat must

// be in the same project folder as your source code.

infile.open(“a:\\funstuff.dat”); // infile is now associated to funstuff.dat located

// on the a drive. Note the two \\. Since the \

// represents an escape sequence two backslashes

// are required to create a single \

outfile.open (“fun.dat”); // outfile is now associated to the file fun.dat. If the

// file does not exist it will be created. If it does

// exist the prior contents are now destroyed permanently.

To close a file the close method is used. Example:

infile.close(); // disassociate the identifier from the file used in the open statement

// thus allowing the identifier to be used again in the same program

// with another file.

infile.clear(); // clears all error bits. I.e. the fail state of the input/output

// stream.

**Reading/Writing Data to a file**

The insertion, extraction, get, getline, ignore all work with file stream objects. Format manipulators can also be used with file stream objects.

Examples:

|  |  |
| --- | --- |
| // This program uses the << operator to write information to a file.  #include <iostream>  #include <iomanip>  #include <fstream>  using namespace std;  int main()  {  ofstream outfile;  outfile.open("demofile.txt");  cout << "Now writing information to the file.\n";    // Write 4 great names to the file  outfile << setw(10) << "Bach" << endl;  outfile << setw(10) << "Beethoven\n";  outfile << setw(10) << "Mozart\n";  outfile << setw(10) << "Schubert\n";    // Close the file  outfile.close();  cout << "Done.\n";  return 0;  } | The contents of the file demofile.txt is shown below.  Bach  Beethoven  Mozart  Schubert |

**Note: To be able to read from a file it is necessary for the programmer to know the format of the data in the file.**

|  |  |
| --- | --- |
| // This program uses the >> operator to read information  // from a file.  #include <iostream>  #include <fstream>  using namespace std;  int main()  {  ifstream infile;  const int SIZE = 81;  char name[SIZE];  infile.open("demofile.txt");  cout << "Reading information \nfrom the file.\n\n";    infile >> name; // Read name 1 from the file  cout << name << endl; // Display name 1  infile.ignore(80,'\n'); // Consume the newline character    infile.getline(name,80); // Read name 2 from the file  cout << name << endl; // Display name 2    infile >> name; // Read name 3 from the file  cout << name << endl; // Display name 3    infile >> name; // Read name 4 from the file  cout << name << endl; // Display name 4  infile.close(); // Close the file  cout << "\nDone.\n";  return 0;  } | Output  Reading information from the file.  Bach  Beethoven  Mozart  Schubert  Done.  Press any key to continue . . . |

When reading data from a file the data is read sequentially with the first data item being placed in the first variable, second data item being placed in the second variable, etc. Recall the extraction operator skips all whitespace and it terminates the read when a whitespace character is encountered.

**Lab Exercises**

**Exercise 1**

Do problem 18 (Population Bar Chart) on page 297.

Copy and paste your program into your word document. Capture the output window and paste it below your program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Programmer: Andrew Tran

//

// Description: To produce a bar chart to reflect the population of a town at 20 year

// intervals

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

ifstream inputFile;

int population;

int year = 1900;

int stars;

inputFile.open("People.txt");

cout << "PRAIRIEVILLE POPULATION GROWTH" << endl;

cout << "(each \* represents 1,000 people)" << endl << endl;

while (year <= 2000)

{

cout << year << ' ';

inputFile >> population;

for (stars = 1; stars < population; stars += 1000)

{

cout << '\*';

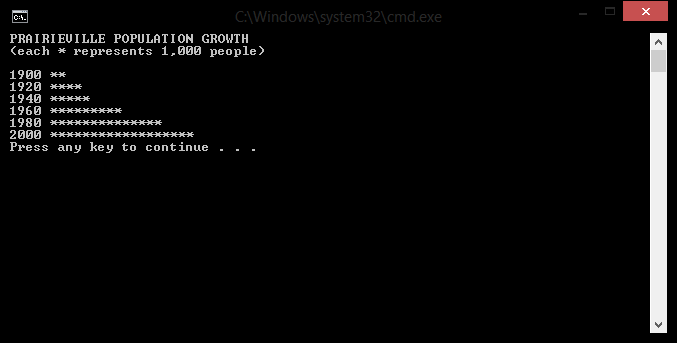
}

cout << endl;

year += 20;

}

}



**Exercise 2**

Mighty Casey plays centerfield for the Toledo Mudhens and has the following lifetime hitting percentages:

|  |  |
| --- | --- |
| Out | 35% |
| Walk | 16 % |
| Single | 20% |
| Double | 15% |
| Triple | 9% |
| Home Run | 5% |

The above is called a probability distribution. A probability distribution is used in predicting the likelihood of an event occurring. See the following explanation of probability distributions using the example of flipping a coin at: <http://stattrek.com/Lesson2/ProbabilityDistribution.aspx>. **Notice the distribution adds up to 100%.**

Write a program to simulate a large number of times at bat (1000 or more) for Mighty Casey counting the number of outs, walks, singles, etcetera to predict Mighty Casey’s batting average for next season and slugging percentage. This means you have to generate a random number (hint: between 1 and 100) and based upon the value (i.e. probability) will determine if Mighty Casey gets a hit or an out. If it is a hit, then your program needs to determine what type of hit was based upon the probabilities given in the table above.

Use the formulas below to calculate the batting average and slugging percentage of Mighty Casey.

batting average = number of hits / (number of times at bat – number of walks)

slugging percentage = (number of singles + number of doubles \* 2 + number of triples \* 3

+ number of homeruns \* 4) /(number of times at bat – number of walks)

Copy and paste your program into your word document. Capture the output window and paste it below your program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Programmer: Andrew Tran

//

// Description: To display various statistics using a simulation to calculate

// Mighty Casey's batting information over a large amount of times

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

int main()

{

int count = 1;

double out = 0, walk = 0, single = 0, two = 0, three = 0, homeRun = 0;

srand(time(NULL));

do {

int number = rand() % 100;

if (number <= 35)

out++;

else if (number <= 51)

walk++;

else if (number <= 71)

single++;

else if (number <= 86)

two++;

else if (number <= 95)

three++;

else if (number <= 100)

homeRun++;

count++;

} while (count <= 1000);

double average = (single + two + three + homeRun) / (count - walk);

double slugging = (single + (two \* 2) + (three \* 3) + (homeRun \* 4)) / (count - walk);

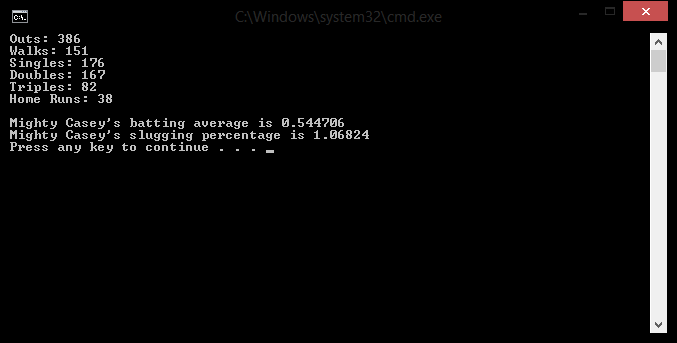
cout << "Outs: "<< out << "\nWalks: " << walk << "\nSingles: " << single << "\nDoubles: " << two << "\nTriples: " << three << "\nHome Runs: " << homeRun << endl;

cout << "\nMighty Casey's batting average is "<< average << endl;

cout << "Mighty Casey's slugging percentage is " << slugging << endl;

return 0;

}



**Due Dates:**

As specified on the D2L Drop box folder for lab 8.

**What to hand in:**

1. Hand in a print out of your word document.
2. Hand in a print out of your program.
3. Compress the .cpp and the word processed document into a single compressed file called **{yourname}Lab8.zip** e.g. timwrennlab8.zip Note:**If your name is not part of the zip filename, I will not open the zipped file.** A popular file compression program is [winzip](http://www.winzip.com/), which you can download to use on a trial basis.
4. Place the compressed file into the lab 8 dropbox folder.