1. Problem Statement

The problem statement is to write a Monte Carlo simulation in C/C++ of the virtual memory replacement algorithms Least Recently Used (LRU), First In First Out (FIFO), and Clock. 1,000 page number traces (experiments) will be iteratively generated. Each memory trace generated will contain 1,000 page numbers separated into 10 regions where each region has a base page number equal to 10 times its region number; the trace will be adjusted using a normal, random number distribution with a mean of 10 and a standard deviation of 2. Each algorithm will be tested for working set sizes from 4-20.

The average (over the 1,000 experiments) number of page faults for each algorithm for each working set size will be computed. These results will be plotted in a manner similar to Figure 8.16 in the textbook using an external program such as Microsoft Excel or LibreOffice Calc. The chart will be included in the report.

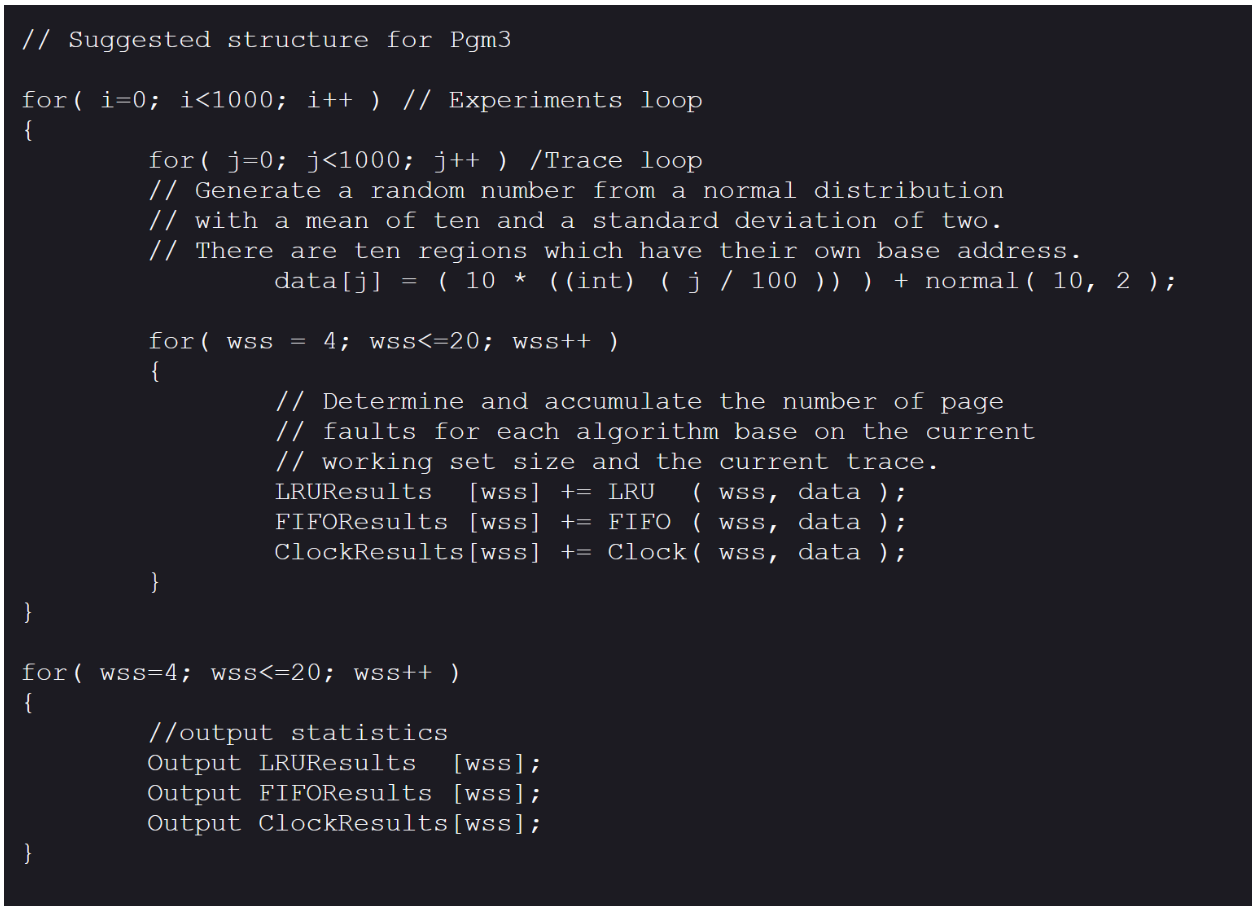
The report will include a conclusion with a recommended algorithm along with the decision making process based on experiences implementing and analyzing this program. The report will also identify any special libraries which must be installed to facilitate the build and execute process. The code and report along with any additional materials required to build and execute the program will be uploaded to the designated eLearning page.

2. Approach

Due to the increasing complexity of the programs, Microsoft Visual Studio Code is now replaced by Codeblocks IDE as the program's work environment. The Linux Ubuntu environment is the same as it has been since Program 1. This will allow for faster and more convenient compilations for the sake of testing; this will also allow for the implementation of libraries.

This program makes use of the Boost library. The random and normal distribution header files will be used to be able to randomly generate from a normal distribution. Please make sure this library is installed when compiling the file.

We were generously provided a sample code structure for Program 3. A screenshot of this code has been provided in this report. Minor deviations may be made from the provided code where the programmer deems fit; however, generally the code will be followed as closely as possible to honor the requested structure.

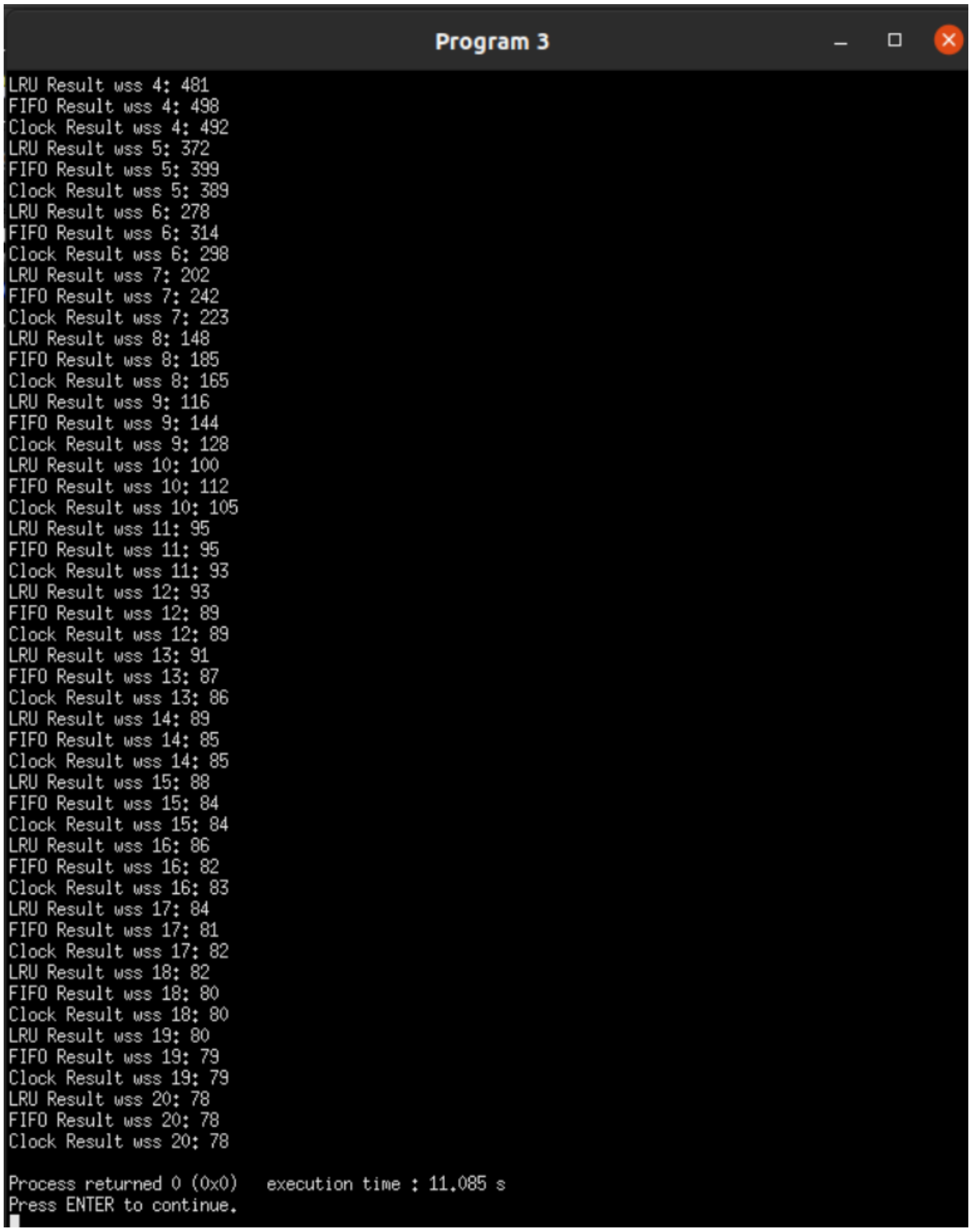


```
// Suggested structure for Pgm3

for( i=0; i<1000; i++ ) // Experiments loop
{
        for( j=0; j<1000; j++ ) /Trace loop
        // Generate a random number from a normal distribution
        // with a mean of ten and a standard deviation of two.
        // There are ten regions which have their own base address.
                data[j] = ( 10 * ((int) ( j / 100 )) ) + normal( 10, 2 );

        for( wss = 4; wss<=20; wss++ )
        {
                // Determine and accumulate the number of page
                // faults for each algorithm base on the current
                // working set size and the current trace.
                LRUResults  [wss] += LRU  ( wss, data );
                FIFOResults [wss] += FIFO ( wss, data );
                ClockResults[wss] += Clock( wss, data );
        }
}

for( wss=4; wss<=20; wss++ )
{
        //output statistics
        Output LRUResults  [wss];
        Output FIFOResults [wss];
        Output ClockResults[wss];
}
```

Figure 1: Screenshot of programming code structure has been generously provided by the professor. This program will use this pseudocode as a base in order to develop source code. However, small modifications may be made where the programmer sees fit.

3. Solution

In order to provide a visual indication that the program functions, the program will output the average page faults for each working set size. Details on build and execution will be provided in the later appendix. The program will also take a few seconds before it starts running; the rather excessive output is meant to confirm that the program is indeed performing correctly.



```
LRU Result wss 4: 481
FIFO Result wss 4: 498
Clock Result wss 4: 492
LRU Result wss 5: 372
FIFO Result wss 5: 399
Clock Result wss 5: 389
LRU Result wss 6: 278
FIFO Result wss 6: 314
Clock Result wss 6: 298
LRU Result wss 7: 202
FIFO Result wss 7: 242
Clock Result wss 7: 223
LRU Result wss 8: 148
FIFO Result wss 8: 185
Clock Result wss 8: 165
LRU Result wss 9: 116
FIFO Result wss 9: 144
Clock Result wss 9: 128
LRU Result wss 10: 100
FIFO Result wss 10: 112
Clock Result wss 10: 105
LRU Result wss 11: 95
FIFO Result wss 11: 95
Clock Result wss 11: 93
LRU Result wss 12: 93
FIFO Result wss 12: 89
Clock Result wss 12: 89
LRU Result wss 13: 91
FIFO Result wss 13: 87
Clock Result wss 13: 86
LRU Result wss 14: 89
FIFO Result wss 14: 85
Clock Result wss 14: 85
LRU Result wss 15: 88
FIFO Result wss 15: 84
Clock Result wss 15: 84
LRU Result wss 16: 86
FIFO Result wss 16: 82
Clock Result wss 16: 83
LRU Result wss 17: 84
FIFO Result wss 17: 81
Clock Result wss 17: 82
LRU Result wss 18: 82
FIFO Result wss 18: 80
Clock Result wss 18: 80
LRU Result wss 19: 80
FIFO Result wss 19: 79
Clock Result wss 19: 79
LRU Result wss 20: 78
FIFO Result wss 20: 78
Clock Result wss 20: 78

Process returned 0 (0x0)   execution time : 11.085 s
Press ENTER to continue.
```

Figure 2: Codeblocks terminal screenshot of the program running. The outputted results here will be analyzed shortly. Though excessive, the output is meant to compensate for the rather long execution time (here seen as 11 seconds).

A more readable version of the output has been provided as a table. This table will also be provided in the Excel document uploaded to eLearning. This is the raw numerical data used to produce the following chart.

wss	**LRU**	**FIFO**	**Clock**
4	481	498	492
5	372	399	389
6	278	314	298
7	202	242	223
8	148	185	165
9	116	144	128
10	100	112	105
11	95	95	93
12	93	89	89
13	91	87	86
14	89	85	85
15	88	84	84
16	86	82	83
17	84	81	82
18	82	80	80
19	80	79	79
20	78	78	78

Figure 3: Table of the output results converted from the terminal output. This is the raw numerical output data used to produce the chart. It is a bit difficult to determine the characteristic data of these policies at a glance; the conclusion will be elaborated upon after the chart.

As requested, the data has been converted into a chart with Microsoft Excel stylized after Figure 8.16 in the textbook. A copy of the Excel document used to produce the chart has been provided in the eLearning submission. The results depict that the performances of the three policies follow that of the textbook rather closely; this will be grounds for a proper analysis under the assumption that the algorithms were properly implemented.

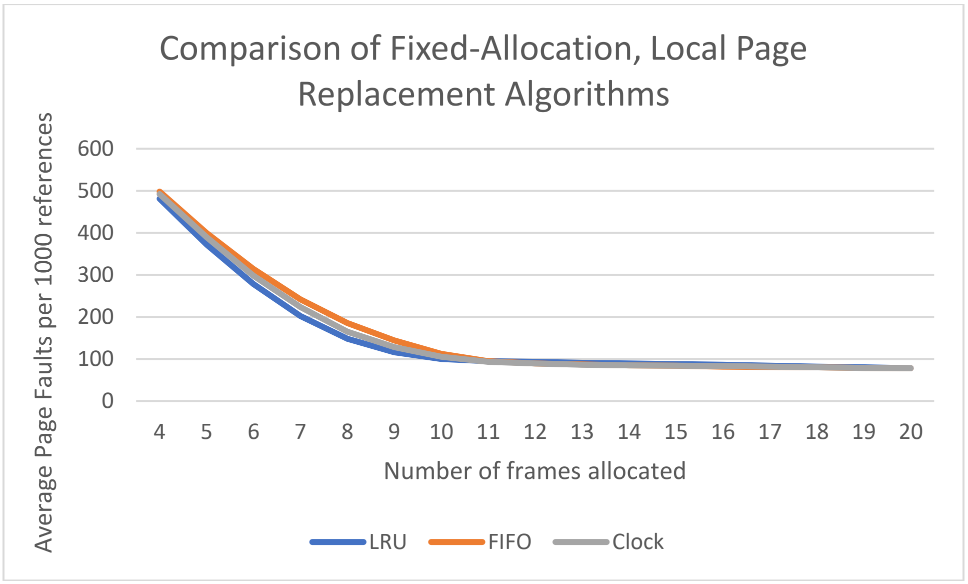


Figure 4: Chart of the output results stylized after Figure 8.16 in the textbook. Each policy is color-coded accordingly for better visibility. Although the 3 policies are at their most different between frame allocation sizes 4-9, at around frame 10 they start to homogenize and eventually have similar performances.

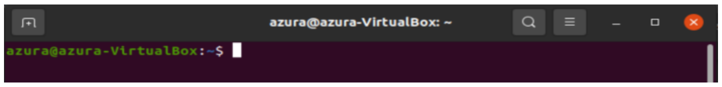
Due to its performance (in terms of relative lack of average page faults), the LRU policy is recommended. Speaking from experiences in implementation and analysis, although I had anticipated the LRU policy to be the most difficult to implement, I found it the most straightforward of the 3. Conversely, the Clock policy I found the most difficult and complicated to implement because I spent a lot of time overthinking the implementation; it was more straightforward than I initially thought it would be. The FIFO policy overall has the simplest implementation method (despite my personal struggles overthinking the program) albeit at the cost of being the worst performing of the three. However, ultimately the page faults become less and less common as the number of allocated frames increases; indeed, all 3 policies tied in output with a working set size of 20.

4. Appendix: Build and Execution

The program makes use of the Boost library. This can be downloaded at https://www.boost.org/. Specifically, this program makes use of the files normal.hpp and random.hpp. A copy of Boost should also be provided on the eLearning upload.

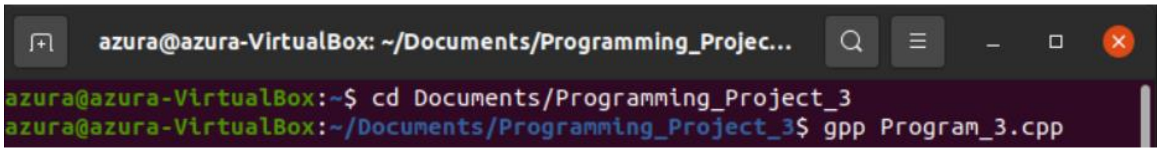
Unlike Program 2, Program 3 requires no special flags in order to compile.

First, open the Terminal.



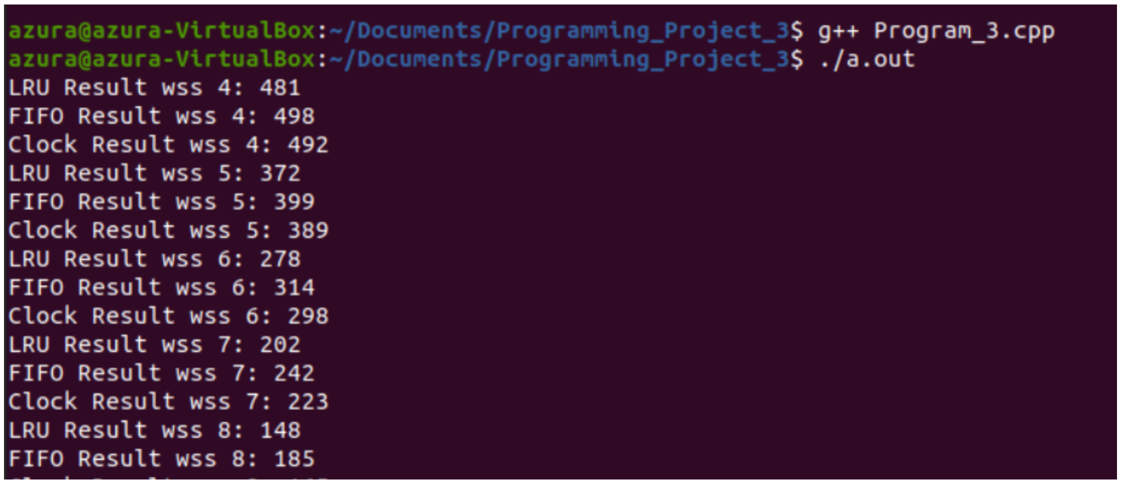
```
azura@azura-VirtualBox:~$ 
```

Change directories into where the file is stored.



```
azura@azura-VirtualBox:~$ cd Documents/Programming_Project_3
azura@azura-VirtualBox:~/Documents/Programming_Project_3$ gpp Program_3.cpp
```

Compile the program with "g++ Programming_3.cpp" and run the output file.



```
azura@azura-VirtualBox:~/Documents/Programming_Project_3$ g++ Program_3.cpp
azura@azura-VirtualBox:~/Documents/Programming_Project_3$ ./a.out
LRU Result wss 4: 481
FIFO Result wss 4: 498
Clock Result wss 4: 492
LRU Result wss 5: 372
FIFO Result wss 5: 399
Clock Result wss 5: 389
LRU Result wss 6: 278
FIFO Result wss 6: 314
Clock Result wss 6: 298
LRU Result wss 7: 202
FIFO Result wss 7: 242
Clock Result wss 7: 223
LRU Result wss 8: 148
FIFO Result wss 8: 185
```

5. References and Resources

- Operating Systems Internals and Design Principals
- Pgm3Notes.txt
- Lecture and Consultation with Instructor