Purpose: This program will explore the utilization of the buffered I/O algorithm to efficiently read/write large quantities of data.

Once working, this program will then be used to compare I/O operation times using different size buffers (See Part B).

The program must be written from scratch.

Program Specifications:
Write an assembly program that will read in a file and outputs a file with the hex values of the first file.

The program will accept filenames as input from the command line arguments. Additionally, the program can include an optional "-p" command line argument that indicates the program should print each byte read in.

Example Command Lines:

```
./a.out inputFile.dat outputFile.txt

./a.out inputFile.dat outputFile.txt -p
```

The -p option can only occur in the last command line argument.

The program should verify the command line count (print a usage guide if there is only a single command line argument).

Open the input and output files and verify that they opened successfully. Otherwise, output an appropriate error message and end the program.

The program should then repeatedly call getByte and writeByte until the input file runs out of data. For each byte of data read in, the write function should expand it to five characters: "0xHH", LINEFEED where the two H are replaced by the hex value for that byte. This means the output file will be 5 times larger than the input. The output buffer should be 5 times larger as well. Once the

end of the input file is reached, the program should use the finalizeOutputBuffer function to write any remaining data in the output buffer to the output file.

This program will require the use of two buffers, one for input and one for output. The output buffer should be five times as large as the input.

If the user uses the -p command line option, the program should additionally call printByte using each byte read in. This will severely affect the performance of the program as console I/O operations are slow.

Use buffer sizes of 100,000 (input) and 500,000 (output).

Once completed, use the program to answer part B. Make sure to set the buffer sizes back to 100,000 and 500,000 before turning in your program.

Required Functions

getByte
Argument 1: Address to file descriptor
Argument 2: Address to input buffer
Argument 3: 64 bit integer value for Buffer Size
Argument 4: Address to buffered bytes count
Argument 5: Address to read bytes count
Argument 6: Address to a byte value for tracking endOfFile
Argument 7: Address to string file access error message
Argument 8: Address to store the byte read in from the buffer
Returns -1 if the operation failed, 0 if the file has no more data, or 1 if the operation succeeded

Reads in one byte using the buffered I/O algorithm.

writeByte
Argument 1: Address to file descriptor
Argument 2: Address to output buffer
Argument 3: 64 bit integer value for Buffer Size
Argument 4: Address to buffered bytes count
Argument 5: Byte value to expand and write to file
Argument 6: Address to string file access error message
Returns -1 if the operation failed, or 1 if the operation succeeded

Converts and writes five characters using the buffered I/O algorithm. When the buffer is full, the function should write it to the file and clear the buffer.

Dividing the byte (argument 5) by 16 will give you the two hex values required.

Examples

```
124 / 16 = 7 r 12
  7 / 16 = 0 r 7

0x7C

255 / 16 = 15 r 15

0xFF
```

Each hex value will need to be adjusted to its appropriate ASCII representation. Check to see if the value is less than 10 or greater than 10 to determine which offset value to use.

7	12
7 < 10	12 > 10
7 + ‘0’ = ‘7’	12 + ‘A’ – 10 = ‘C’

Each byte should be converted into 5 characters:
"0x", the two converted hex digits, linefeed

Once the buffer is full, attempt to write it to the given file and reset the buffer (buffered characters = 0).

finalizeOutputBuffer
Argument 1: Address to file descriptor
Argument 2: Address to output buffer
Argument 3: Address to buffered bytes count
Returns -1 if the operation failed, or 1 if the operation succeeded

Writes any remaining characters in the output buffer to the file.

printByte
Argument 1: an unsigned integer byte by value
Prints the byte in hex format to the console. The string must be stored as a local variable to printByte.

Part B:
In a document file (.docx or .odt), experiment with the buffer size by running the program with buffer sizes of 1, 1000, and 100,000. For each buffer size, run the program using the Linux time program on the combined.txt file. It may take several minutes to run.

```
time a.out combined.txt combinedOutput.txt
```

Create a table using the real, user, and system times and calculate the percentage of time each buffer size spends in system time(sys).

```
(sys time)/(real time)*100%
```

Explain why certain buffer sizes finish faster than others in 200 words or less (preferably much less).

Assembly and Linkage
Make sure to use main: instead of _start: as your starting function and use **g++ -g -no-pie assemblyFile.o** to link your program.

Submission
Once you are satisfied with the program, upload the assembly source code (.asm) file to the class website to Assignment #7 Part A and the document file to Assignment #7 Part B.

Example Executions

```
$ ./a.out inputFile.dat outputFile.txt
$
```

```
$ ./a.out test.txt testOutput.txt -p
0x54
0x65
0x73
0x74
0x69
0x6E
0x67
0x2C
0x20
0x74
0x65
0x73
0x74
0x69
0x6E
0x67
0x2E
0x0A
0x48
0x65
0x6C
0x6C
0x6F
0x2C
0x20
0x57
0x6F
0x72
0x6C
0x64
0x21
$
```