**CSE-381: Operating Systems**

**Exercise #4**

Max Points: 20

**Note: If you are using your personal machine then prior to commencing work on this exercise, you may need to install XMing, Putty, and WinScp as illustrated in LinuxEnvironment.pdf (and shown in the videos in the Handouts folder).**

|  |
| --- |
| **You should save and rename this document using the naming convention MUid\_ex4.docx (example: raodm\_ex4.docx).**  **Objective**: The objective of this exercise is to:   * Gain some familiarity with debugging C/C++ programs on a Linux machine and review concepts from earlier classes. * Use vectors, iterators, and algorithms available in C++ for problem solving.   **Submission**: Once you have completed this exercise, upload this MS-Word document (duly filled with the necessary information) and the C++ program that you developed with the following naming convention:   * ***MUid*\_ex4.doc**: This document (renamed) and duly filled-in. * ***MUid*\_ex4\_2.cpp**: A C++ program to analyze words in a text document.   Fill in answers to all of the questions. For almost all the questions you can simply copy-paste appropriate text from the shell/PuTTY window into this document. You are expected refer to [LinuxEnvironment.pdf](https://niihka.muohio.edu/access/content/group/b05b7d30-d8c6-4312-9559-4d980565cbaf/Handouts%20_%20Video%20Tutorials/LinuxEvironment.pdf) document available in Handouts folder off Niihka. You may discuss the questions with your instructor (preferably only when all else fails as this is a learn-by-doing style exercise). |

|  |  |
| --- | --- |
| **Name:** |  |

# Preliminaries

1. Log onto the Linux server for this course via the following steps (that were covered in the previous exercises and as illustrated in the [LinuxEnvironment.pdf](https://niihka.muohio.edu/access/content/group/b05b7d30-d8c6-4312-9559-4d980565cbaf/Handouts%20_%20Video%20Tutorials/LinuxEvironment.pdf)):
   1. Run the X-Server Xming.
   2. Use PuTTY to log into the Linux server cse381-f12.csi.muohio.edu.
   3. When you log onto the server, you will be presented with a shell (**$**) prompt. You need to perform various tasks by typing commands at the shell prompt and pressing the enter (↵) key.
   4. Start emacs and ensure you see the graphical screen for emacs.

# Part #1: Debugging C++ programs [5 points]

|  |  |
| --- | --- |
| C:\Users\Victoria\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\MARLET97\MP900314351[1].jpg | Ensure that the C++ source for this exercise is named with the convention *MUid*\_ex4\_1.cpp (example: raodm\_ex4\_1.cpp), where *MUid* is your Miami University unique ID. |

**Background**: Debugging using the GNU debugger (gdb) is an important skill to successfully troubleshoot C++ programs. This exercise is designed to provide a brief introduction to gdb.

**Exercise**: This is a simple exercise that is designed to introduce debugging under Linux using emacs and gdb. Perform the exercise in the following manner:

1. Using emacs, copy-paste the following program into a C++ source file named ***MUid*\_ex4\_1.cpp**.

|  |
| --- |
| #include <iostream>  #include <string>  int main() {  std::string fullName;  std::cout << "Enter full name: ";  getline(std::cin, fullName);  std::string firstName = fullName.substr(0, 3);  std::string lastName = fullName.substr(4);    std::cout << "First name = " << firstName  << ", Last name = " << lastName  << std::endl;  return 0;  } |

1. Ensure that the program compiles and runs to produce output as shown below (user inputs are highlighted in red just in the document for clarity):

|  |
| --- |
| raodm@cse381-f12:~/cse381$ g++ -std=c++0x -g -Wall raodm\_ex4\_1.cpp -o raodm\_ex4\_1  raodm@cse381-f12:~/cse381$ ./raodm\_ex4\_1  Enter full name: **indiana jones**  First name = ind, Last name = ana jones  raodm@cse381-f12:~/cse381$ |

1. Now wait for your instructor to introduce the basics of debugging a program under Linux (as illustrated in the [LinuxEnvironment.pdf](https://niihka.muohio.edu/access/content/group/b05b7d30-d8c6-4312-9559-4d980565cbaf/Handouts%20_%20Video%20Tutorials/LinuxEvironment.pdf)). Your instructor will cover the following:
   1. Running the program under gdb from emacs.
   2. Setting up breakpoints in emacs.
   3. Running the program, stepping through the program, and continuing the program.
   4. Exiting from gdb.
2. Rerun the program under gdb and set a breakpoint in gdb on the line that prints the results (highlighted in the code snippet above). Run the program and provide “hal smith” as the input when prompted for input. Next your program should hit the breakpoint set earlier returning you to (gdb) prompt. At the (gdb) prompt type suitable commands (you may also refer to [LinuxEnvironment.pdf](https://niihka.muohio.edu/access/content/group/b05b7d30-d8c6-4312-9559-4d980565cbaf/Handouts%20_%20Video%20Tutorials/LinuxEvironment.pdf)) for the following questions and copy-paste the value displayed by gdb in to this document.

|  |  |
| --- | --- |
| **Task** | **Data displayed by gdb** |
| Print value of variable fullName [gdb-command: print fullName] |  |
| Print the address of variable fullName [gdb-command: print &fullName] |  |
| Paste the address of variable fullName as reported by two of your neighbors and indicate their unique IDs. |  |
| Are their addresses for fullName reported by your neighbors different than yours [A yes/no answer is sufficient]? |  |
| Print the output of calling the size() method on fullName [gdb-command: print fullName.size()] |  |
| Use gdb to print value of the mathematical expression 0x16 + 0123 + 5 [gdb-command: print 0x16 + 0123 + 5] |  |
| Print the current stack trace [gdb-command: where] |  |

# Part #2: Develop a word analysis program [15 points]

|  |  |
| --- | --- |
| C:\Users\Victoria\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\MARLET97\MP900314351[1].jpg | Ensure that the C++ source for this exercise is named with the convention *MUid*\_ex4\_2.cpp (example: raodm\_ex4\_2.cpp), where *MUid* is your Miami University unique ID. |

**Background**: Unlike other programming languages C++ provides the following set of language constructs (that fall under the umbrella of generic programming) that streamline certain routine operations: vectors, iterators, generic algorithms, and lambda closures. Using these constructs requires a different thought paradigm. This exercise is designed to motivate the use of the algorithm package included in C++.

|  |  |
| --- | --- |
|  | This exercise must be completed using just suitable methods in the algorithms library and very short lambda methods as needed. **You loose points for each looping construct used in your program (as none are really needed)**. All the methods you need for this exercise are available in the Lecture slides off Niihka. |

**Exercise**: This exercise expects you to develop a C++ program that analyzes a text file and prints a histogram (bin size of 3) of the length of unique words in the file as shown in the sample output below (user input is highlighted in red for clarity):

|  |
| --- |
| Enter path to file: few\_words.txt  STATISTICS FOR FILE : few\_words.txt  Number of words : 9  Number of unique words: 6  Histogram for Word Lengths (bin size is 3):  2: ########################################  2: ########################################  2: ########################################  0:  0: |

Here are some directions and tips for completing the exercise:

1. Prompt (see sample output below) and obtain path to a text file from the user. Open the file for reading using an ifstream. If the stream is invalid then print the message “Error open input file.” and return from main() with exit code 1.
2. Load all the words from the file stream into a vector.
3. Sort the vector and extract all the unique words into another vector for use in steps below.
4. Count the number of unique words whose word-length falls into the following five bins: 1-3, 4-6, 7-9, 10-12, 13-. Note that you don’t need to create the bins but just determine the count of number of words in each category and store it as an array or vector so as to print the histogram in the next step. For example, given the unique words: “one”, “two”, “three”, “seven”, “eighteen”, and “nineteen” the values in the histogram entries would be: {2, 2, 2, 0, 0} (interpreted as: 2 unique words of length 1-3, 2 unique words of length 4-6, 2 unique words of length 7-9, zero words of length 10-12, and zero words longer than 12 characters).

For this part of the exercise in the sample solution, I used a std::for\_each loop to iterate over the unique words and passed-in an histogram (an array of 5 integers initialized to zeros) to a lambda (signature: [&histogram](const std::string& word){ }) that appropriately incremented the entry in the array based on the length of the string passed to the lambda. The lambda uses straightforward math to increment the appropriate entry in the histogram.

1. Use the std::max\_element() method to determine the largest bin in the histogram for use in the next step.
2. Having computed the histogram, display the following information about the text file analyzed as shown in the sample outputs.
   1. The path entered by the user
   2. The total number of words in the file.
   3. The number of unique words in the file.
   4. The histogram indicating the number of entries in a bin and a proportional number of ‘#’ characters depending on number of words in each bin. The histogram is displayed such that there are 40 ‘#’ characters displayed for the largest bin and other bins are displayed with a proportionally scaled number of ‘#’ characters. For example assume the histogram has entries {12, 6, 3, 0, 0}, then the first bin is displayed with 40 ‘#’s, the second bin with 20 ‘#’s, the third bin with 10 ‘#’s, and the other bins are displayed with 0 “#”s as shown in the sample output. This can be easily accomplished using the formula: *pound\_count = bin\_size \* 40 / largest\_bin\_size*.

**Sample Output**:

Sample outputs expected from your program after processing input text files few\_words.txt and sherlock.txt (downloadable off Niihka) are shown below. User inputs in the two different runs of the program are shown in red color in the sample outputs below:

|  |
| --- |
| Enter path to file: few\_words.txt  STATISTICS FOR FILE : few\_words.txt  Number of words : 9  Number of unique words: 6  Histogram for Word Lengths (bin size is 3):  2: ########################################  2: ########################################  2: ########################################  0:  0: |
|  |
| Enter path to file: sherlock.txt  STATISTICS FOR FILE : sherlock.txt  Number of words : 1863  Number of unique words: 892  Histogram for Word Lengths (bin size is 3):  95: ##########  379: ########################################  297: ###############################  102: ##########  19: ## |

**Actual Output:** Once you have successfully developed and tested your program, copy-paste the output from your program into the space provided below for the input file gpl.txt (download from Niihka).

|  |
| --- |
| Output from your program from processing gpl.txt goes here. |

# Part 4: Submit files to Niihka

Upload the following two files to Nihhka:

1. Upload this MS-Word document (duly filled with the necessary information) using the naming convention MUid\_ex4.docx.

* ***MUid*\_ex4\_2.cpp**: The C++ program you developed for word analysis.