The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2024

Lab 1: Tools for EECS 281

Instructions:
Work on this document with your group, then enter the answers on the canvas quiz.

Note:
Please **complete the Prelab** (found in the Lab 1 Canvas folder)! Be prepared before you meet with your lab group, and read this document so that you know what you must submit for full credit. You can even start it ahead of time and then ask questions during any lab section for help completing it.

You MUST include the following assignment identifier at the top of every file you submit to the autograder as a comment. This includes all source files, header files, and your Makefile (if there is one). If there is not autograder assignment, you may ignore this.

Project Identifier: CD7E23DEB13C1B8840F765F9016C084FD5D3F130

To find the starter code for this lab, go here.

1 Logistics

Before you login to CAEN, you'll want to make the correct verion of the compiler and libraries your default. To do this run the following command, being careful to replace `you` with your actual uniqname, and typing the rest exactly (DO NOT copy/paste from the PDF, it will not work; check the discussion forums for one that's safe to copy/paste):

```
ssh you@login.engin.umich.edu "echo 'module load gcc/11.3.0' >> ~/.bash_profile"
```

1. What is the due date of project 1?
   A. 9/15/2024
   B. 9/17/2024
   C. 9/19/2024
   D. 10/10/2024
2. What is the date of the Midterm?
   A. 10/10/2024
   B. 10/15/2024
   C. 10/17/2024
   D. 10/22/2024
3. What percentage of your grade is each project worth?
   A. 1
   B. 5
   C. 10
   D. 20

2 Tools

4. What is the correct login string for CAEN?

   NOTE:
   In this class, when we want to indicate that you should use a variable but we are writing text in a document, we will surround the variable with < and >. For example, for option A in the following answers, Dr. Paoletti's personal login string would be paoletti@login.engin.umich.edu, because he would replace <uniqname> with his uniqname (paoletti).

   A. ssh <uniqname>@login.engin.umich.edu
   B. ssh <uniqname>@engin.umich.edu
   C. ssh <uniqname>@caen.umich.edu
   D. ssh <uniqname>@umich.edu

5. Why do we use Makefiles?
   A. They can create the submission files automatically.
   B. They automate the (sometimes long and complex) compilation process for us.
   C. They can run custom testing scripts.
   D. Typing out compilation commands every time is annoying.
   E. All of the above.

6. Will your program compile on the autograder without the project identifier?
   A. Yes.
   B. NO, and I'll potentially waste the submission.

7. What is the Makefile command to create a submission file that includes custom test cases?
   A. `make`
   B. `make all`
   C. `make partialsubmit`
   D. `make fullsubmit`

8. Which of the following is a debugging tool?
   A. `Makefile`
   B. `perf`
   C. `valgrind`
   D. `cin`

9. What does `perf` do?
   A. Detects segmentation faults in a program and displays line numbers that they occurred at.
   B. Profiles program execution time.
   C. Profiles program memory usage.
   D. Automates the testing process.

To complete the following questions, you have to either login to CAEN (should work for everyone), or have `valgrind` working locally. Under Linux or the WSL, try to run `valgrind` and if it's not installed, it should tell you the command to install it. We haven't had any luck getting `valgrind` to run on a Mac; if you have, please stop in and talk to Professor Darden! If you are having trouble logging on to `CAEN`, please contact a staff member or email `eecs281admin@umich.edu`. After successfully connecting to CAEN, follow the instructions below:

- Visit the lab 1 repository on the EECS 281 GitHub: https://github.com/eecs281staff/lab-01-music-sorting.
- Retrieve the contents of the repository. You should be able to use the `git clone` command followed by the above link.
- If you have the files locally and need them on CAEN, you can do another `git clone` there, or run `make sync2caen` with the `Makefile` to copy your files to CAEN.
- There are two starter file folders that are most important for this lab: `music_sorting` (complete and submit to the autograder) and `valgrind`. For the next two questions, look at the `valgrind` folder and find the `lab1_bad.cpp` file. You'll need to refer to the line numbers in it to answer the questions below.
- Type the following commands to build the program, run it with `valgrind`, and display the results:

```
make valgrind
valgrind ./lab1_bad_valgrind |& more
```

- The `more` program paginates the output, stopping after each full page. You can press `<Enter>` to go forward by one line, the `<space bar>` to go forward by one page, or `q` to quit.

Answer the following two questions based on the results of running the program with `valgrind`. You may have to make your terminal window wider to read the `valgrind` messages more easily. When you're done, use the `make clean` command to remove the executable.

10. What is the **FIRST** error message reported by `valgrind`?
    A. `Invalid read of size 4`
    B. `Invalid write of size 4`
    C. `Use of uninitialised value of size 8`
    D. `Conditional jump or move depends on uninitialised value(s)`
    E. (no errors are reported, the program runs fine and outputs ten 1-digit integers)

11. Which line of code (in `lab1_bad.cpp`) does `valgrind` report the first error occurring on?
    A. Line 60
    B. Line 65
    C. Line 73
    D. Line 76
    E. Line 79

3 Debugging

We have found in previous semesters that students entering 281 are not comfortable in their development environments, specifically with using git, gcc/clang, make, valgrind and debuggers. These tools and more are vital for success in projects and labs in 281, but also in future EECS courses you may take, and in industry. This assignment uses small toy programs to help you get more comfortable and hopefully learn a thing or two before jumping into more complex projects. **For each question, assume that the given code is compiled according to the flags given in the Makefile (shown below), under the CAEN Linux environment.**

Assume that all the programs are complete and that no other source files exist.

12. What is wrong with the following program?

```
int add(int a, int b);
int main() {
   int x = 1;
   int y = 5;
   return add(x, y);
} // main()
```

A. `add()` has no definition.

B. `add()` is called with `x` and `y`, but accepts `a` and `b`.

C. The program does not compile because `main()` cannot return a value of 6.

D. A function cannot be called after a `return` statement.

E. Nothing is wrong with this program.

13. What is wrong with the following program?

```
int main() {
   int x = 0;
   return x + y;
} // main()
```

A. `main()` cannot return the result of an arithmetic expression.

B. `main()` cannot return `x + y` because `x + y` is a `double`.

C. `main()` does not contain a declaration for `y`.

D. `main()` cannot return an integer.

E. Nothing is wrong with this program.

14. What is wrong with the following program?

```
#include <vector>
int *get_some_ints() {
   std::vector<int> ints = {1, 2, 3, 4, 5};
   return ints.data();
} // get_some_ints()
int main() {
   int *some_ints = get_some_ints();
   delete[] some_ints;
   return 0;
} // main()
```

   A. The memory pointed to by `some_ints` is freed twice.

   B. `main()` leaks the memory pointed to by `some_ints`.

   C. A function cannot return a pointer.

   D. `some_ints` is a pointer and not an array, so `delete` should be used instead of `delete[]`.

   E. Nothing is wrong with this program.

15. What is wrong with the following program?

```
struct Thing { };
int main() {
   Thing a;
   Thing b;
   bool less = a < b;
}
```

   A. `main()` cannot declare an instance of a `Thing` object.

   B. `main()` must have a return value.

   C. `Thing` cannot have an empty definition, so this code does not compile.

   D. `main()` tries to use the $<$ operator, which is not defined for the `Thing` type.

   E. Nothing is wrong with this program.

16. What is wrong with the following program?

```
int main() {
   int some_ints[5] = { 1, 2, 3, 4, 0 };
   for (int *p = some_ints; p; ++p)
      *p = 0;
   return 0;
} // main()
```

   A. `main()` indexes out of the bounds of `some_ints`.

   B. `main()` leaks the memory pointed to by `some_ints`.

   C. `some_ints` points to uninitialized memory.

   D. The `for` loop is missing a curly brace, so the code will not compile.

   E. Nothing is wrong with this program.

17. What is wrong with the following program?

```
#include <iostream>
int sum_ints() {
   int *some_ints = new int[50];
   for (int i = 0; i < 50; ++i) {
      some_ints[i] = i;
   } // for i
   int sum = 0;
   for (int i = 0; i < 50; ++i) {
      sum += some_ints[i];
   } // for i
   return sum;
} // sum_ints()
int main() {
   std::cout << sum_ints();
} // main()
```

A. `sum_ints()` indexes out of the bounds of `some_ints`.

B. `sum_ints()` leaks the memory pointed to by `some_ints`.

C. `some_ints` points to uninitialized memory.

D. `main()` must have a return value.

E. Nothing is wrong with this program.

18. What is wrong with the following program?

```
int factorial(int n) {
   return n * factorial(n - 1);
} // factorial()
int main() {
   return factorial(3) - 6;
} // main()
```

A. `factorial(3)` returns an uninitialized value.

B. A mathematical operation cannot follow a `return` statement.

C. `factorial(3)` never returns (and may cause a stack overflow).

D. `main()` cannot return the result of a recursive function.

E. Nothing is wrong with this program.

19. What is wrong with the following program?

```
#include <iostream>
int what_is_2x281() {
   int x, y = 281;
   return x += y;
} // what_is_2x281()
int main() {
   std::cout << "What is 2 x 281?\n" << what_is_2x281();
} // main()
```

A. The `+=` operator cannot be used after a `return` statement, since `x` would be returned before being added to `y`.

B. `what_is_2x281()` returns an uninitialized value.

C. The type of `y` is not defined.

D. `main()` must have a return value.

E. Nothing is wrong with this program.

20. What is wrong with the following program?

```
void takes_an_integer(int x) {}
int main() {
   size_t x;
   std::cin >> x;
   takes_an_integer(x);
} // main()
```

A. The code does not compile because `takes_an_integer()` has an empty definition.

B. `main()` must have a return value.

C. `takes_an_integer()` takes an `int` but is called with a `size_t`, which may cause a loss of precision.

D. `size_t` is not a valid type.

E. Nothing is wrong with this program.

4 File Input & Output

21. This code applies to the next two questions.

```
int main(int argc, char []*argv) {
   int i;
   // Read in an integer from a file.
} // main()
```

(a) Which line(s) of code would read an integer from a file using file redirection?

A. `ifstream readfile; readfile >> i;`

B. `cout << i;`

C. `cin >> i;`

D. `ofstream writefile; writefile << i;`

(b) Which of the following command line commands would run the above program (in `main.cpp`, already compiled to the executable `main`) with input file redirection?

A. `make all`

B. `./main input_file.txt`

C. `./main > input_file.txt`

D. `./main < input_file.txt`

22. A text file contains a list of words that are all on separate lines, with only a trailing newline after every word. One program, program `T1`, reads the file with `cin >>`, reading in to a `string`, `S1`. Another program, program `T2`, reads the file with `getline`, also reading into a `string`, `S2`.

(a) What will be the difference, if any, between the two resulting `string` variables?

A. `S2` will contain a trailing space, while `S1` will not.

B. Both strings will be exactly the same.

C. `S1` will have a newline character at the end, while `S2` will not.

D. `T1` will not read the file.

(b) What if there is a space at the end of each word in the file?

A. `S2` will contain a trailing space, while `S1` will not.

B. Both strings will be exactly the same.

C. `S1` will have a newline character at the end, while `S2` will not.

D. `T1` will not read the file.

5 Getopt

23. You're writing a simple text-based game on the command line. When the user runs your executable (`./281quest`) the game starts at the first level. If the user wants to skip to a certain level, they can instead run the program with the command `./281quest --level 5` (where `--level` is followed by the level they want to skip to). Which of the following correctly specifies this flag for `getopt`?

    A. {`"level", no_argument, nullptr, 'l'` }

    B. {`"level", optional_argument, nullptr, 'l'` }

    C. {`"level", required_argument, nullptr, 'l'` }

24. Short Options You're writing a program that will take five command line options: -p –paoletti, -d –darden, -a –angstadt, -m –markov, and -g –garcia. Options d, m, and g will have required arguments. The rest take no arguments. Which of the following strings is a correct short options string?

    A. `pd:am:g`

    B. `p:d:a:m:g`

    C. `p:dam:g:`

    D. `pd:am:g:`

    E. `p:da:mg`

6 Handwritten Problem

This problem is to be submitted independently. We recommend trying it on your own, checking your answer with your group and discussing solutions, and then submitting it to Gradescope. These will be graded on completion, not by correctness. However, we want to see that you were thinking about the problem.

Writing your solution by hand is good practice for the exams and therefore strongly recommended. You may submit your solution to Gradescope in the form of a PDF, image, or similar. You may also submit your solution to Gradescope in the form of a .cpp file. For this lab, if you implement your solution in `palindrome.cpp` and submit this file to Gradescope, the Gradescope autograder will provide feedback.

Your grade does not depend on the format of your submission or the feedback from the autograder. Your solution is graded solely on completion and not by correctness.

The starter files can be found on Canvas.

25. Linked-List Palindrome Given the `start` and `end` nodes of a doubly linked list, determine if the values create a palindrome. A palindrome is a sequence that is the same backwards and forwards. See the definition of the `Node` struct below, or in `palindrome.h`. Return `true` if the list is a palindrome.

```
struct Node {
   Node *prev;
   Node *next;
   char value;
};

bool isPalindrome(Node *start, Node *end);
```


7 Coding Assignment

You can work on these problems by yourself or with your group, but a solution must be submitted to the autograder for each individual.

For this lab, you will be familiarizing yourself with the `get_opt` function, as well as our Makefile and the autograder. To accomplish this task, go to the `music_sorting` directory from the repository you retrieved from Canvas or GitHub (https://github.com/eecs281staff/lab-01-music-sorting). To clone the directory on to your local machine, go to your terminal and type the command `git clone https://github.com/eecs281staff/lab-01-music-sorting.git`.

There should be two files present, `sorting.h` and `lab1.cpp`. There is nothing for you to do in `sorting.h`, but you need to have it in the same directory as lab1.cpp for the code to compile. Make sure to step through `lab1.cpp` to find and complete all the TODO statements. There is one test file for you to test your code on in the folder.

`lab1.cpp` contains a definition for a `MusicLibrary` object. This object will read a CSV file, put the entries into a vector, and then sort the vector and print the top `n` songs based on several command line options that will be processed with `get_opt`. Carefully read the comments and observe what is happening in the code; there are a lot of concepts contained in the functionality that will help you later in the course.

`sorting.h` contains the `Song` definition, as well as an overloaded `operator<<` for printing the songs. Please look around the file, but there's nothing that you have to do in the code.

Once you have completed the coding portion, take a look at the Makefile. There are a few TODO statements in the Makefile as well. If the Makefile is confusing, don't worry; we will go over it in lab. Once you have a grasp of how to use it, compile your code and use the command `make fullsubmit`.

Finally, submit your code to the autograder. When you run the `make fullsubmit` command in the `Makefile`, there will be a tarball in the directory that contains all of the files. Go to the autograder, click 'choose file' under Lab 1, and find the tarball in your file system. Then upload your submission and watch as the test cases run!

Make sure that all code files submitted to the autograder include the assignment identifier listed on the first page of this assignment, or below.

Project Identifier: CD7E23DEB13C1B8840F765F9016C084FD5D3F130

