

CSCI-2275 Programming and Data Structures Recitation 2

Instructors: Rhonda Hoenigman

Learning objectives:

- Compile source files from terminals
- Use command line arguments
- Read and write files

1. Compile C++ files from terminal

To run a program on your laptop, you need to compile the source code first, and then execute it. The compiler for C++ code is called g++, and you should have already installed it from your first lab.

Changing directory

To run a C++ file you need to make sure that you are in the correct directory. Directories are relative paths and you will have to navigate through directories and subdirectories till you reach the correct location.

Here are some of the important commands:

1. To see all files and directories in your current location :
 - a. Unix - ls
 - b. Windows - dir
2. To go to a subdirectory - cd _____

Compile source code

Then, once you're in the correct directory, you can run the compiler. This step is to compile your source code to a executable program for the computer. The computer doesn't recognize your code, after all.

To compile the file main.cpp, use command:

```
g++ -std=c++11 -o main main.cpp
```

The flag `-std=c++11` is to indicate that we will use C++ 11 standard to compile the file.

Now to run your command we can just write the name of the file we have after our `-o` tag.

1. Unix - `./main`
2. Windows - `main`

Caution

The name of your program must follow the flag “-o”. A common mistake is to put your source code file name after “-o”. This will overwrite your source code, and it'll end up as an empty file.

2. Use command line arguments

When you run your program it evokes the main function in your source code. Main function can also receive arguments. A prototype of main function looks like this:

```
int main(int argc, char const *argv[])
```

Notice that there are two arguments passed to main function from the terminal. The first one, `argc`, is the total number of arguments you passed to main function when you're running your program on the terminal. The second one, `argv`, is an array storing all the arguments you passed.

We can pass multiple arguments. So we can run something like,

```
./new arg1 arg2 arg3
```

Thus, `argc` is 4, and `argv` is an array of length 4. The first string in the array is the program name `new`, and the rest of them are the strings we typed on the terminal, delimited by spaces or tab.

3. Read/write and parse files

Part of this recitation will focus on teaching fstream and sstream, two of the iostream features in C++. The fstream library contains functionality to read and write to files. We will use the ifstream, which is used to read data from a file. We will also use the stringstream features included in the sstream library, which make it possible to treat a string as a stream of characters that our program can process. The value of stringstream in this recitation is that we can read data from the file into a string, and then use a delimiter to pull out the individual values in the string. Please refer the file given in the folder.

4. Structs and arrays of structs

A struct is a collection of data elements grouped together to create a composite data type. They can be made up of different types, including their structs, which is what makes them useful.

```
//Creating a struct data type: a template to define all instances  
struct CarData {  
    string model;  
    string make;  
    string year;  
};
```

In the above example, CarData is the type of struct we chose. The make, model, and year are members of this struct. Every instance of CarData created will contain all three members.

```
int main() {  
    CarData car; //Create an instance of CarData called car  
}
```

Above is how to create an instance of CarData. This is the same as a normal data type we have seen, such as int, string, or double. After the instance is

created, we can access the members of the instance using the dot notation, such as `car.make` or `car.model` to access the make and model members of `car`.

Structs can also be used as the type of an array, just like any other variable type. In this example, we create an array, called `carArray`, of `CarData` with a fixed size of 10:

```
CarData carArray[10];
for (int i = 0; i < 10; i++) {
    cout << "make:" << carArray[i].make << " model:" << carArray[i].model << " year:"
    << carArray[i].year << endl;
}
```

5. Exercise

For this exercise, you will be writing a program that opens the provided text file of Car information. Starting with the above struct and the appropriate libraries, you should store that file's information in an array of structs. Your program also needs to check if the file opened correctly before reading to the end of the file. Once the data is stored in an array, your program should write out the car information to another file.

The file is available on Canvas.

To start:

1. You will need four arguments from command line. The first, by default, is the name of your executable program name. The second is the filename from which you read in car information. The third is the number of lines in that file. The last is the file to which you write out car information. For example, typing the following command into your terminal

```
./main cars.txt 10 outfile.txt
```

will start the program `main` that reads car information from file `infile.txt` which has 10 lines. Then you will output those car information to file `outfile.txt`.

2. Define an array of structs, and read each line from the input file and store the car information. The struct definition is the same to what we used in section 4.
3. Output car information in the array to your output file. Each line is a car, containing three fields -- make, model, and year. They should be separated by tabs.