Exported for Brian O'Connell on Tue, 27 May 2025 19:34:07 GMT

Programming 10 Pre-Lab Lesson - Files/Arrays

Already Covered

In the past labs, we have already covered the following:

- The parts of a C++ Program
- Data Types
- Input/Output Formatting
- Mathematical Operators
- Loops

Look to the those resources for more information and guidance on these individual concepts.

This Labs Objectives

Today, we will go over the following:

- Making Decisions
  - If statements
  - If – else statements
  - Case statements
    - Break
- File Streams
  - Inputting data from files
  - Outputting data to files
- Arrays

Making Decisions

The following statements should all be familiar due to their prior use. This will just establish the new syntax for C++. Note that it also the same syntax as in Arduino so that will also be familiar.

The **Switch** statement is the only that will be gone over in detail. You have seen it in an Arduino example but it was only for use in the context of that example. This will cover it in greater functional detail.

IFTT

`THESE ARE THE SAME RULES AS FOR WHEN WE USE THESE IN ARDUINO. Including them for reference and review.`

For IF THIS THEN THAT statements, there are a few rules:

- Conditional Expressions need to be in parenthesis () - `(expression)`
    - This includes he entirety of any compound expressions
- Any code intended to be part off the statement must be contained in brackets - `{actions}`
    - The exception is if the action is a single statement. Then if can just follow the expression
    - Multiple action example:

      ```
      if (x==0)
      {
         cout << “x is zero”<<endl;
         z=z+1;
         cout << “z has been incremented”;
      }
      cout << “finished”;
      ```
    - Single action example:

      ```
      if (x==0)
         cout << “x is zero”<<endl;
      cout << “finished”;
      ```

IFTT Syntax

IF Statements

- `if (expression)`

- `{`
- `  action/s;`
- `}`

IF ELSE Statements

- `if (expression)`
- `{`
- `  action/s;`
- `}`
- `else`
- `{`
- `  action/s;`
- `}`

IF ELSE IF Statements

- `if (expression 1)`
- `{`
- `  action/s;`
- `}`
- `else if (expression 2)`
- `{`
- `  action/s;`
- `}`

IF ELSE IF ELSE Statements

- `if (expression 1)`
- `{`
- `  action/s;`
- `}`
- `else if (expression 2)`
- `{`
- `  action/s;`
- `}`

- `else`
- `{`
- `  action/s;`
- `}`

There is no limit to the number of expressions that can be evaluated:

- `if (expression1)`
- `{`
- `  action/s;`
- `}`
- `else if (expression 2)`
- `{`
- `  action/s;`
- `}`
- `else if (expression 3)`
- `{`
- `  action/s;`
- `}`
- `...`
- `else if (expression n)`
- `{`
- `  action/s;`
- `}`
- `else`
- `{`
- `  action/s;`
- `}`

Question 1 Show Correct Answer Show Responses

Is there a functional difference between the following code snippets:
Snippet A:

```
if (expression) {   action/s; } else {   action/s; }
```

Snippet B:

```
if (expression) {   action/s; } else {   action/s; }
```

Snipper C:

```
if (expression) {  action/s; } else {  action/s; }
```

A Yes

B No

C More context required to determine

Operators

C++ uses the same operators to other languages.

Relational Operators

Operator	Description
==	Equal to
!=	Not Equal to
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To

Logical Operators

Logical Operators	Description
!	NOT
&&	Logical AND
||	Logical OR

Question 2

Which of the following statements is TRUE about conditional expressions in C++ IF statements?

?

A	Conditional expressions must be enclosed in parentheses.
B	Conditional expressions must include multiple relational operators.
C	Conditional expressions can modify variables directly.
D	Conditional expressions are only used with else if statements.

HINT

EXPLANATION

Show Responses Show Answer

SWITCH Statements

THIS PART IS NEW. You may have seen this in some Arduino examples but we never formally went over it.

This is a powerful tool in programming that we will discuss the reasons for that in more detail in class. This lesson focuses on it's functional use. There is a short discussion of the underlying efficiency though.

The basic syntax for a **SWITCH** statement is:

```
switch (expression)
{
 case constant1:
    action(s);
    break;
 case constant2:
    action(s);
    break;
```

```
  ...
  default:
     action(s);
}
```

The expression is typically a variable or a simple calculation. The **switch** statement compares the `expression` against each of the `case constants` to determine which case to run. It runs the `action(s)` in that case until it hits `break`. That command, `break`, then ends that statement, jumping to the end.

NOTE: `break` works the same in any statement. It leaves a statement, ie jumps to the ending `}`, even if the condition for that loop has not been fulfilled and there is more code left in that loop or statement. It is typically used, outside of switch statements, as a means to end infinite loops under specified conditions

Without the break at the end of a case, the switch statement would continue on to perform the following cases. It is a necessary at the end of each case if you wish for that not to occur. A common oversight is forgetting to include a `break` at the end of each `case`.

Question 3 Show Correct Answer Show Responses

Given the following code snippet and x is equal to 2, what would the output be?

Efficiency

A common observations is that **Switch** statements function similarly to an **IF-ELSEIF-ELSE** statement.

SWITCH Version	**IF-ELSEIF-ELSE** Version

`switch (x) {`
`  case 1:`
`    cout << "x is 1";`
`    break;`
`  case 2:`
	`if (x == 1) {`
	`  cout << "x is 1";`
	`}`
	`else if (x == 2) {`
	`  cout << "x is 2";`

`    cout << "x is 2";`	`}`
`    break;`	`else {`
`  default:`	`  cout << "x unknown";`
`    cout << "x unknown";`	`}`
`}`	

This is true when it comes to the coding syntax. Some may even find it easier to use, depending on preference.

A `switch statement` is limited to equivalency comparisons of a limited number of data types. Traditionally this has been limited to integers and characters that only take up a single byte but modern compilers allow for a broader range of available data types to be used. `if-else` statements do allow for more complex and a wider range of expressions and that's part of their appeal as well as a reasoning for their use instead.

The main benefit to using a `switch` is in the processing and running of the executable program that contains it. In an `if-else` chain, all the conditions have to be evaluated meaning that code at the end of the chain is delayed until all previous expressions have been evaluated. At our scale, this is less important but with many many more options, it becomes more important and useful to take advantage of this feature of a `switch` statement. For instance, computer operating systems use a switch statement to handle inputs from a keyboard since there are 100s of possible standardized signals that come from the keyboard and need to be immediately evaluated to determine what character that signal should be evaluated as. A `switch` jumps *immediately* to the intended `case`, having the options already been indexed by the compiler into either a branch table or binary search, which essentially means that the compiler organized those code segments into a more efficiently searched form for the computer. How it did so is more advanced than we'll cover.

File Streams

A **File Stream** functions similarly to the **Standard Input / Output Streams**. It sends and receives information from a file rather than the environment's default source, the console window. In the same way that the input stream and output streams are handled separately with `cin` and `cout`, information input and output along a filestream is also handled separately by establishing a separate input or output object to contain those connections to the file, `ifstream` or `ofstream`.

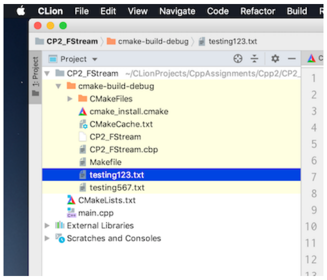
It may be best to start with an example and then describe what's going on in it.

Example Program for File Streams

The following is just an example to see how reading & writing files works

- Download the example file, **testing123.txt**, made available in the Top Hat folder.
- Create a new project in CLION

- Put the downloaded file into the **cmake-build-debug** folder of your c++ project:
    - The file testing567.txt will be created later when you run the example.
- Make sure that the **testing123.txt** file is in the **cmake-build-debug** folder of your c++ project, **NOT** the project folder or the CMakeFiles folder.



Example project folder hierarchy of where your input file should be and the output file will be created.

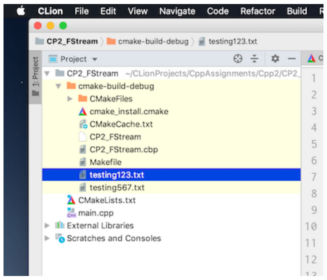
Example code:

- Raw code (Also available below):
    - https://raw.githubusercontent.com/boconn7782/CourseCode/main/CPP/P10_Fstream_Ex
- File for upload:
    - https://raw.githubusercontent.com/boconn7782/CourseCode/main/CPP/testing123.txt
    - right-click and then select save as to download the file
    - Also available in Top hat in the same folder as this lesson.

```
#include <iostream> // Needed for normal cin & cout
#include <fstream> // Needed to read or write files on disk
#include <string> // Needed to use String Data types
using namespace std;
int main() {
  string x; //data type for input, output variable
  ifstream infile; //setup input stream variable
```

```
    ofstream outfile; //setup output stream variable
    infile.open("testing123.txt",ios::in); //open the input file
    outfile.open("testing567.txt",ios::out); //open the output file
    if (!infile) //did input file open OK?
    {
      cout << "Unable to open sample.txt file" << endl;
      cout << "Program to force exit...\n";
      return(1); // Generic exit with error warning.
    }
    while (!infile.eof()) { //This waits for the entire file to be read before moving on.
      infile >> x; // Take in the data from the test file
      cout << x << endl; // Write the value to the console (Note how it comes out)
      outfile << x << endl; // write the value to output file
    }
    outfile.close(); //close output file
    infile.close(); //close input file
    return 0;
}
```

For the following question, you want to take a screenshot that looks similar to this but with your output file given a different, unique name. Simply having it be your name or initials is the safest bet, but it's just to show you've run the example to see how file uploads work.



Example project folder hierarchy of where your input file should be and the output file will be created.

NOTE: you put the input file in the `'cmake-build-debug'` folder NOT the `'CMakeFiles'` folder. That's a common error and one to avoid. This was stated earlier but its that common that I'm repeating it here too.

Question 4

Show Correct Answer

Grade Responses

Students can browse their computer or drag and drop file

Max file size: **50MB**

Input File Stream

Using these methods, you can either open a file to read from it or write to it at any given time. You can not do both. To switch, you would have to close it and then reopen it for the other process.

To start an **Input File Stream** from a file and use it, you must:

- Create an object for that stream
    - `ifstream infile; //Create ifstream object`
- Connect it to a file source
    - `infile.open(“name.filetype”); //Connect object to file`
- Extract content from the file
    - `infile>>variable_to_input;  //Stream from file`
- Then, when finished, close that file stream
    - `infile.close();  //close the input file`

Please note that, in this case, `infile` is just a variable name. It contains the `ifstream` object information. It is just a descriptive variable name and could be named anything else and function the same. We could literally name it `anything`, it just wouldn't be that descriptive a variable name. If you wanted you could name all of your file objects after characters from your favorite obscureTV show but naming a file `RedGreen` would not be too helpful in describing its content or use for others. Descriptive variables and useful comments are still highly recommended.

Once an `ifstream` object is created, it is connected to a file using the command open where the given argument is the name of that file, `“name.filetype”`in the above case. In the provided example program, that was `"testing123.txt"`. The default location for these files is the **cmake-build-debug** folder of your c++ project. There are ways to change this or look in different locations, we will not get into that in this course. The use of `ios::in` and `ios::out` in the example program is only necessary in some operating systems and compilers, but not most. Their use is only necessary if your first attempt at compiling produces an error with the attempt to open the file. It just forces the operation mode to be input or output, but are the default setting in modern IDEs and compilers.

NOTE: Another concern is if you're defining the input argument for `open()` using a string variable, you need to append it with `.c_str()` for `fstream` object commands to be able to interpret the non-native data type. For example, if the filename was stored in the variable FN, then the command would be `infile.open(FN.c_str())`. This also applies for output file stream objects.

Information is inputted using the extraction operator `>>` on the `ifstream` object, the same as with `cin`. This takes in information, storing it in the `variable_to_input` as a buffer to contain the content fr the program's use. It will take in file content until:

- the information in the file runs out,
- the capacity of the `variable_to_input` is achieved,
- or a break character is found in the file stream.

Once that information is read, the program cannot reread it simply. The next time the `>>` is used, it will be extracting from where the last invocation left off. To restart from the beginning of the file, you would have to close the file and reopen it. The command `close()` simply closes the file stream to the assigned file. It can later be reopened or the file assigned can be changed.

Run the example program to see what characters would be considered break characters. Note that since we used a string variable for storing the information, it was not a limiting factor of the file stream's buffer capacity. Open the text files (double click on it) to see how it is currently formatted and how it is outputted to output file.

Output File Stream

To start an **Output File Stream** to a file and use it, you must:

- Create an object for that stream
  - `ofstream outfile; //Create ofstream object`
- Connect it to a file source
  - `outfile.open(“name.filetype”); //Connect object to file`
- Insert content into the file

  -
    ```
    outfile << ”The results are: “ << endl;
    outfile << var1 << var2 << endl;
    //Stream to file (Use the same as cout)
    ```
- Then, when finished, close that file stream
  - `outfile.close();  //close the output file`

Once an `ofstream` object is created, it is connected to a file using the command open where the given argument is the name of that file, `“name.filetype”`in the above case. In the provided example program, that was `"testing567.txt"`. The default location for these files is the **cmake-build-debug** folder of your c++ project. There are ways to change this or look in different locations, we will not get into that in this course. As stated for input file streams, the use of `ios::in` and `ios::out` in the example program is only necessary in some operating systems and compilers, but not necessary for most systems. If the file does not exist, it will create that file. If it does exist, it will overwrite it unless you specify to append to the file. You do this by opening it with the format flag with `ios::app`; ex: `outfile.open(“name.filetype”, ios::app);`.

Information is appended to the file using the insertion operator `<<` on the `ofstream` object, the **exact** same as you would do with `cout`. This also includes any format manipulation made available through using `iomanip` commands. Once information is written to the file, there is no easy way using `fstream` to undo that action. The command `close()` simply closes the file stream to the assigned file. It can later be reopened or the file assigned can be changed.

Question 5   Show Correct Answer   Show Responses

ⓘ **Multiple answers:** Multiple answers are accepted for this question

Based on your observations of the file stream example, what characters will cause a break when reading data in from the `ifstream` object, 'Testing123.txt' in this case? This only applies to when reading the ifstream object using >> and not any of the other methods available from `fstream.h` for those familiar with it.

A	periods - .
B	space
C	integers
D	tabs
E	newlines
F	hashtags - #
G	Parentheses - (or)

Data checking

When dealing with external data, it's useful to check if you either have information to work with or if the information you wanted to output and save actually did.

For checking an input stream, you can simply check if the file exists. An existing object, one successfully created and the file itself is populated with available content, is recognized as `true`. A failed input stream would be one that either could not find the file or connected to a file that has not content. It would be recognized as `false` by the compiler.

- `ifstream infile;`
- `infile.open("testing123.txt",ios::in);`
- **`// What if the file doesn’t exist?!`**
- `// An existing object is recognized as True.`
- `if(infile) {`
- `  // This will run if the file opened successfully`

```
  cout << "Successfully connected to file" << endl;
}
// A failed or empty object is recognize as False
// This uses the not operator, !, and 'NOT False' is equivalent to 'True'
if(!infile) {
  // Specifying NOT a failed or empty object will run trigger this statement.
   // Provide user a warning
   cout << "Error opening the file" << endl;
   return(1); // Typically want to end program to fix issue
}
```

To check if your output was successful, the typical check would be to close the output file stream, establish an input file stream to that same file, and then run the same check above. If it successfully opened and contained content, you can assume you have written information to the intended file.

Arrays

The concept of Arrays and matrices (multi-dimensional arrays) should already be familiar. This section will focus on specifically their use and handling in C++.

Creating arrays

Arrays are created like other variables with defining the data type and specifying the name but with the addition of defining its size as an array in terms of number of elements. This takes the form of:

```
type name [# of elements];
```

Like all variables, you must declare the data type, `type`, and all elements of the array will be of that type. The `name` has the same limitations of other variables. The use of square brackets, `[ ]`, serves as the defining part that makes this variable an array. The `# of elements` specifies the size to be set aside for this array, specifically that # times the allocation size for the datatype. This creation can take 3 forms:

Case 1:

```
double sound[100];
```

Case 1 shows the standard creation by specifying the # of elements. This creates an array of `doubles` named `sound` that contains `100` elements. This sets each element's value to the default value for that data type, which is 0 for the fundamental types.

Case 2:

```
char vowel[] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’} ;
```

Case 2 shows creation by defining the array. This creates an array of `characters` named `vowel`. The # of elements is not given but it's still defined as an array with the use of `[]`. The size is assumed from the number of values provided between `{}`.

Case 3:

```
int velocity[30] = {0, 2, 6};
```

Case 3 is a combination of the first 2 methods. This creates an array of `integers` named `velocity` with the first 3 values defined by the provided array (in `{}`) and the remaining 27 elements, totaling `30` elements, are set to the `default value of 0`. The default value for all data types is their equivalent to 0; 0 in ASCII for characters and strings which is a null character, false for booleans, and 0 for integers and floats.

Question 6

Show Correct Answer

Show Responses

For the following array declarations, how many elements are created and what is the value of the last element?

1 - char greeting [5] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’};

No. of Elements: ______ Value of Last Element ______

2 - int B[ ]={15,3,4,8};

No. of Elements: ______ Value of Last Element ______

3 - int nums[100]={-1,5};

No. of Elements: ____________ Value of Last Element ____________

Calling Array Elements

This is one of the places **where the notation becomes slightly confusing,** so please take note of that.

First of all, the indices of arrays go from `0` to `n-1` for an array of `n` elements. This differs from MATLAB which starts at 1. The other point that becomes problematic is that square brackets are used to define the size but also used to call array elements.

- `char vowel[5] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’} ;`
- `// Defines an array of 5 characters`
- `cout<< vowel[0]; // Prints ‘a’`
- `cout<< vowel[4]; // Prints ‘u’`

So for an array created with `N` elements (`int example[N]`), you would call the first element with `0` (`example[0]`) and the last element with `N-1` (`example[N-1]`).

Outputting Array Information

In MATLAB, you could print an entire array or matrix by calling it's variable. You did not have great format control but it did work. In C++, you again have to output things by the individual elements, and can not natively just output the full array.

For example, given this code:

- `// Defines an array of 6 integers`
- `int velocity[6] = {0, 2, 6};`
- `cout << velocity;`

You end up with an array `velocity` that is equivalent to `{0, 2, 6, 0, 0, 0}`. You might expect `cout << velocity` to output something like `0 2 6 0 0 0` or `{0, 2, 6, 0, 0, 0}`. In actuality it will output something like `0x7ffee59b5a30` which is the location on your computer where the array information is stored. To output the elements of the array, you'd have to cycle through it and print each individually:

- `int velocity[6] = {0, 2, 6};`
- `int n = sizeof(velocity) // Gets size of array`
- `for (int i = 0; i < n; i++) {`
- `  cout << array[i] <<‘, ’;`
- `}`

Editing Arrays

In MATLAB, large subsets of an array could be edited with a single command. That is NOT the case with more traditional programming languages. In fundamental C++, you are limited to editing one element at a time.

Basic example in C++:

- `int n = 20;`
- `int score[n];`
- `for (int i=0; i<n; i++)`
- `{`
- `cout<<“Enter person’s score: ”<<i+1;`
- `cin>>score[i];`
- `}`

To edit multiple elements, your code needs to navigate directly to each element and reset the value. In MATLAB, you can bulk change content but you can't in C++. Look at the comparative example below:

In MATLAB:

- `x = 1:10; % Using the colon operator to create a unit array from 1 to 10`
- `disp(x) % Would output to console as: 1 2 3 4 5 6 7 8 9 10`
- `x(4:7) = 1; % Bulk changing values 4 through 7 to be equal to 1`

- `disp(x) % Would output to console as: 1 2 3 1 1 1 1 8 9 10`

In C++:

- `int x[10] = {1,2,3,4,5,6,7,8,9,10}; // Manually created unit array from 1 to 10`
- `for (int i = 0; i < 10; i++) {`
- `  cout << x[i] <<‘ ’;`
- `} // Would output to console as: 1 2 3 4 5 6 7 8 9 10`
- `for (int j=4; j<=7; j++) {`
- `  x[j]=1;`
- `} // Changes values 4 through 7 to be equal to 1, has to be performed individually though and using a loop to do that.`
- `for (int k = 0; k < 10; k++) {`
- `  cout << x[k] <<‘ ’;`
- `} // Would output to console as: 1 2 3 1 1 1 1 8 9 10`

As for the size of the array, that can not be edited once defined. For that reason, if you do not know the exact size you need, it's typically best practice to create array larger than what you may need.

Multi-dimensional Arrays

Multi-dimensional arrays are pretty straight forward. You simply add another set of square brackets. The most typical you will deal with is 2D arrays, matrices. The general format for these is:

- `General: data_type array_name[# of Rows][# of Columns];`
- `Example: int array2D [5][10];`

This format can be extended for 3+ dimension arrays:

- `int array3D [5][10][25];`
- `int array4D [5][10][25][70];`
- `And so on...`

You will be unlikely to get into arrays of that many dimensions soon. You'll likely only first need to deal with them if you take a course involving machine learning or high level control systems.

Question 7 Show Correct Answer Show Responses

ⓘ **Case sensitive:** The answer for this question is case sensitive

What is the command to create a 2-dimensional array that has 7 columns and 3 rows. It should be able to store numbers that have at least 8 decimal places. The name of the array should be 'Carl'.

Question 8

Which statement best describes the process of editing elements in an array in C++?

?

A You can change multiple elements at once using a single command.

B Each element must be edited individually, often requiring a loop.

C Arrays created in C++ are immutable and cannot be edited.

D Bulk editing is possible by specifying the index range like in MATLAB.

HINT

EXPLANATION

○ Show Responses

Show Answer

P10L1: Character Sorter

There is a walkthrough video at the end to help you through any element you may have struggled with.

For this activity, we will make a text file and fill it with random characters then sort it. This will import data from a file and load that information into an array. We will then manipulate that array using a common sorting algorithm to rearrange them into their ASCII value order.

Activity:

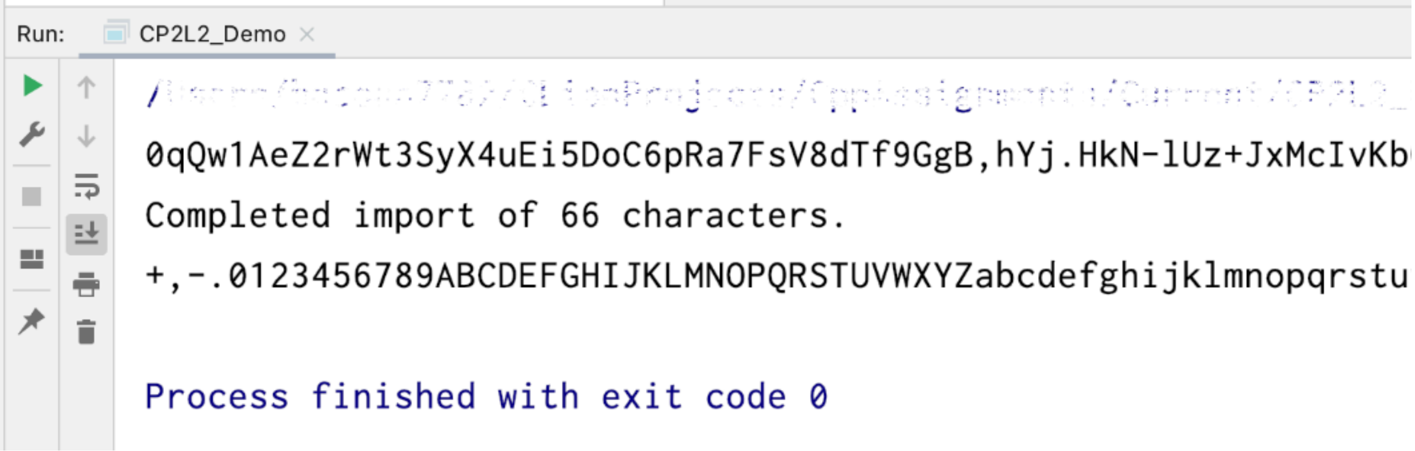
- Create a text file in CLION for testing your code.
  - Populate it with random characters
- Create a C++ program that:
  - Uploads the test file
  - Loads that information into an array
  - Reorders that array based on ASCII value order
  - Prints the original order and new order to the console output

Requirements:

- Use at least 50 characters in your test file
  - At least half are unique, meaning you can have some repeat characters but at least try and keep them mostly unique
- Do not copy the example from below
  - Type out your own message or random characters.

Example Output:

Using a test file made with 1 character from the majority of keys on an U.S. QWERTY keyboard, including capitalization for the letters.



```
/[redacted]
0qQw1AeZ2rWt3SyX4uEi5DoC6pRa7FsV8dTf9GgB,hYj.HkN-lUz+JxMcIvKb
Completed import of 66 characters.
+,-.0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu

Process finished with exit code 0
```

Example output when input file is just random characters. You can also use phrases, names, and/or random

Pseudocode:

This could be achieved by a variety of approaches, but there are some very specific things I want you to practice so implement the following pseudocode. :

```
PROGRAM CharacterSorter:
Introduce Program to user
Open test file as an input stream
Check the file stream
Stream file content into an array
Print original array data
Sort array data using a common sorting method
Print sorted array
ENDPROGRAM
```

Like any pseudocode, we can update much of this to add clarity and direction for the programmer. This now incorporates specific content from the lesson (That you are required to use and not program around because you've thought of another way). The original is included in plain text with the parts expanded upon bolded.

```
PROGRAM CharacterSorter:
Introduce Program to user
Open test file as an input stream
Check the file stream
IF file stream not valid
  Provide user warning and end program with error
ENDIF
Stream file content into an array
FOR each element in array
  load character from file stream into array element
ENDFOR
Print original array data
FOR each element in array
  Print element to console output
ENDFOR
Sort array data using a common sorting method –
   This will be expanded later
Print sorted array
FOR each element in sorted array
  Print element to console output
ENDFOR
ENDPROGRAM
```

We can also further simplify things when we notice trends in our pseudocode. Look at lines 9 and 13, they are the same. This suggests we can combine those for loops to get:

```
1.  PROGRAM CharacterSorter:
2.  Introduce Program to user
3.  Open test file as an input stream
4.  Check the file stream
5.  IF file stream not valid
6.    Provide user warning and end program with error
7.  ENDIF
8.  Stream file content into an array
9.  FOR each element in array
10.   load character from file stream into array element
11.   Print original array data
12.   Print element to console output
13. ENDFOR
14. Sort array data using a common sorting method – This will be expanded later
15. Print sorted array
16. FOR each element in sorted array
17.   Print element to console output
18. ENDFOR
19. ENDPROGRAM
```

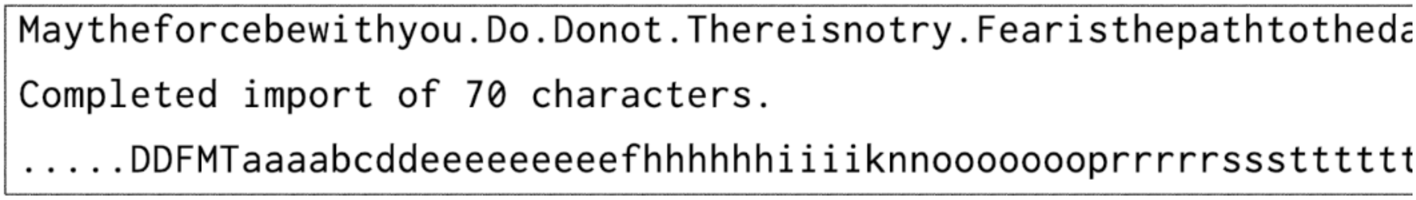
Create Test File

We're going to create a test file for our use before we start. It's not uncommon to create dummy test files that include an idealized version of the intended data, usually generated or selected so it contains no unexpected curveballs or odd edge cases and is much shorter than the future data set. In this case, we're just making a text file and randomly typing or selecting some text.

1. Start a new project using CLION for this lesson
2. Right click in the `cmake-build-debug` folder
3. Select **New > File**
4. Enter a file name for your text name.
    1. `P10L1_*****.txt` replacing `*****` with your first initial and partial last name.
5. Double click n the new file to open it

6. Add content to the file. The content doesn't matter, just try and get at least 50 characters.
    1. Type randomly
    2. Let your cat walk across the keyboard
    3. Copy some movie quotes into the file.

       For Example here's an run of the intended code where the input file contains the GREATEST movie quotes:



```
Maytheforcebewithyou.Do.Donot.Thereisnotry.Fearisthepathtotheda
Completed import of 70 characters.
.....DDFMTaaaabcddeeeeeeeeefhhhhhiiiiknnooooooprrrrrsssttttt
```

Build your program

Use your Pseudocode as your first comments to help scaffold your program. I've provided some starter code that includes some more scaffolding to remind you of the other elements you'll need as well. In this walkthrough, I'll talk through as if expanding just this initial psuedocode:

https://raw.githubusercontent.com/boconn7782/CourseCode/main/CPP/P10L1_starter

```
/* Title Block */
main() {
// Introduce Program to user
// Open test file as an input stream
// IF file stream not valid
//   Provide user warning and end program with error
// ENDIF
// FOR each element in array
//   load character from file stream into array element
//   Print element to console output
// ENDFOR
// Sort array data using a common sorting method
// FOR each element in sorted array
//   Print element to console output
// ENDFOR
}
```

Set up the parts of your program

1. Libraries & Directives

We have requirements that will need certain libraries:

- Communicating with the console window
- Creating an input file stream and communicating with it

2. Declare Variables

We already know of some information we will need to store and use throughout:

- An input stream object will be needed set to the file we create
- We'll be loading that ddata into an array
  - We don't know what size our file will be so make sure it's oversized

3. Main Program

- We know what we want to do in our main program based on the pseudocode

Some Housekeeping

- `/* Title Block */`
- `...`
- `int main () {`
- `// Introduce Program to user`

Title Block and the introduction & instructions can typically wait till the end, when cleaning up for submission **AS LONG AS YOU DON'T FORGET IT**. Doesn't hurt to tackle it now but you can do that on your own.

Initial libraries

- `// Libraries & Directives`
- `#include... // Load the necessary libraries`
- `using namespace std; // To simplify code`

Based on what has been established as required, load the necessary libraries. A good guess would be at least the ones introduced in this lesson and the last one. You can always go back and load more as you determine they're needed or delete ones you don't end up using.

Open an input file stream

- `// Open test file as an input stream`

Here you'll need to create the `ifstream` object and then `open` it with an input file stream to your test file. The file name can be hard coded into your program. Look at the examples and integrate that into your starter code. There are comments to help guide you.

- `// IF file stream not valid`
- `//   Provide user warning and end program with error`
- `// ENDIF`

Look to the example from earlier in the lesson for this too. This is a good point to test your code to make sure it works by using an incorrect file name. Don't forget to change the file name back to the correct value before you move on.

Import file data to an array

- `// FOR each element in array`
- `//   load character from file stream into array element`
- `//   Print element to console output`
- `// ENDFOR`

For this we will have to create an array, then a for loop that cycle through it. Use another variable to control the array size so you can use that same variable to define the maximum value for your loop:

```
int x = 100;
char A[x];
```

Set up your for loop to iterate through the array elements and assign them a value from the input file stream. Use the examples above to do so.

We use an oversized value of `x` because we don't know how many characters are in the file. That means we want to include a break when we find the end of the file (see the `.eof` used in the earlier example to find the end of the file stream object), so before you add to you're counter for how many characters you've inputted. So your pseudocode becomes:

- `// FOR each element in array – hint: Use x in your for loop expression 2`
- `//   load character from file stream into array element`
- `//   Print element to console output`
- `//   IF at end of the infile`
- `//      Break the for loop`
- `//   ENDIF`
- `//   Counter++`
- `// ENDFOR`
- `cout << endl << "Completed import of " << n << " characters." << endl;`

So if the text file is less than the size of the array, you don't start loading junk characters into the array. Try commenting out the if statement and see what values get printed. If you're text file is bigger, the outer for loop will simply end at `x` characters.

Sort the Array

Here's the hard or easy part

- `// Sort array data using a common sorting method`

I'm not going to have you completely figure this out on your own. Sorting algorithms typically require nested loops and array manipulatino but can be conceptually difficult.

If you feel like challenging yourself, feel free to use a different sorting method. Make sure you're clear which one in your comments. Here's a good resource for finding examples - https://www.geeksforgeeks.org/sorting-algorithms/. There's also a great animation here, https://www.toptal.com/developers/sorting-algorithms, that visually represents all sorting algorithms.

For the easy route **`(Which is a totally fine path to take and the one the vast majority of you will go with)`**, we're going to implement a bubble sort that I'm providing a detailed psuedocode for below. This is a simple sort that passes through an array of size n approximately $\frac{n(n-1)}{2}$ times comparing adjacent elements. If you watch the toptal video, you'll see it's not the most efficient (or the worst) but it it one of the easier ones to conceptually understand.

A Bubblesort will compare adjacent elements, swap them if they're out of order, then compare the next set. It repeats this for the whole data set, starting on the first element and working through the set then starting on the second element and working through the set, then starting at the third, then the fourth, and so on. It's a brute forth approach of just doing the comparisons enough times to be confident you've rearranged things even though in most cases you end up not making a swap.

Here's the pseudocode for a Bubblesort, using some of the variable names I've already referenced above, `n` for the number of characters imported and `A` for the oversized array created. We only need those elements with file data in them though and not all `x` elements, so we use the counter value `n`:

Before going through this, it is worth viewing the Toptal video to understand how a bubblesort works

```
SET n = Length of Array A
FOR j = 0 to n-1
// We leave out the last one because we compare an element vs the element+1. Last element has no +1.
  FOR k = 0 to (n-j-1)
    // Now we're running through all the remaining values after our current value as part of this for loop
    IF element is greater than the next
      // Switch their positions
      temp = element;
      element = next element;
      next element = temp;
    ENDIF
  ENDFOR
ENDFOR
```

In this, too call the next element, you can just use `A[k+1]`. So your comparison can simply be `A[K] > A[K+1]`. I'm leaving the detailed pseudocode but not the C++ syntax though for you to practice translating pseudocode and for loops. This should be a 1 to 1 translation so if you're struggling with this, don't hesitate to ask.

Print the sorted array

1. `// FOR each element in sorted array`