Agenda / Learning Objectives:

1. Run the following command and extract lab27.tar in your venus account (note the dot):

   cp ~ctse/cs111/lab27.tar . ; tar xvf lab27.tar
2. Go over the tic-tac-toe.cpp at a high level and estimate:
   - How long will it take for us to write something similar to it?
   - How to conquer legacy code – freeCodeCamp.org
3. Go over examples of using **break**, **continue**, **post-increment/decrement** and **pre-increment/decrement**.
4. Identify the key string functions that show up frequently in the exam.
5. Read the key points, tips and pitfalls provided by the textbook (see below) after your final exams.
6. Understand the usual format of the final exam:
   - Appear every semester:
     i. Title Lines
     ii. Tracing
     iii. Short Code Blocks
   - Usually the hardest question: **Recursion**
   - One or two questions: **1D/2D Array**
   - Others:
     i. String and Chars
     ii. File I/O
     iii. Arguments to main()
     iv. Nested Loops/Drawing Patterns

Below are key concepts copied from the teacher's guide for chapter 9 of Absolute C++ textbook:

C-strings are the kind of strings C++ gets from its C language heritage. A C-string is an ordinary array of char with the additional termination requirement. The last character position used in the array must be indicated by **a null character** ('\0') in the next position.

The Standard C++ library provides the class string. This class is defined in the header <string>. This class has many useful member functions and overloaded operators that make string processing easier and more intuitive.

We call the type of string C++ inherits from C, C-strings. We call a C++ Standard string class object simply a "string".

Key Points

Escape Sequences. The text's discussion of cstring variables points out that the null character '\0' is the terminator for cstring variables. The text points out that the library functions that process cstrings (including the iostream functions) use the null character as a sentinel indicating the last character of the cstring. The backslash (\) signals that the next character to be dealt with in a way different than it would be handled without the backslash. The backslash is called the 'escape character' because it escapes the normal meaning of certain characters, allowing them to have special meaning. (And it removes or escapes special meaning of characters that have special meaning, such as the backslash itself. To print a backslash, use two backslashes \\ in the cstring literal. Here the first backslash escapes the special meaning of the second backslash, which is inserted in the output stream.)

If we just put a '0' into some position in a cstring, the encoding for 0, 48 decimal, or 30 hexadecimal, is stored at that character position. (See Appendix 3 for the decimal encoding of the printing ASCII characters.) To cause the null character to be stored, we use the \ before the 0 to escape the usual meaning of 0. This tells the compiler that we want the numeric value of the null character to be embedded in the cstring. (The value of the null character really is zero, not the normal encoding of the character 0.)

C-string values and C-string variables. A c-string is a sequence of characters terminated by a null symbol. It is the default type for literal strings using "".

Character Manipulation with get and put. The get and put functions allow your program to read in one character of input or store one character at a time. It is interesting to note that cin.get() returns an int. The put member takes a char argument which causes the int to be converted to a char. Casts may be required depending on your use.

The Standard Class `string`. The string class overloads many operators: + and += for concatenation, = for assignment, <, <=, >, and >=, for string comparison use lexicographic ordering, and the indexing operation, [i], for access to characters in the string. There is no range checking for indexing on string objects. If range checking is needed, there is a string member function called at() that is range checked. It provides many member functions that provide useful functionality.

I/O with the class string. The insertion, <<,and extraction, >>, operators are overloaded for string objects. By default, extraction to a string object

```
string s;
cin >> s;
```

ignores initial whitespace, just at extraction to a C-string. This operation then extracts the next sequence of nonwhite characters, and stops input at the next white space.
There is a version of getline defined specifically for string objects. The syntax is

```
string s;
getline(cin, s);
```

This version of getline is necessarily a stand-alone function. The committee of authors of the string library had two choices. They could have rewritten the iostream library to add a getline member function, or they could have written a stand-alone getline function. The iostream library was already written and stable. Just as you have to write a stand-alone overloading of the output operator for your classes, these authors chose to write a stand-alone overloaded version of getline for this pair of arguments.

The getline function extracts from the input stream characters up to the delimiter character. The characters from the start to the delimiter are placed in the string variable, and the termination character is discarded. The delimiter character defaults to the newline character, '\n'. By supplying a third argument, the default termination character can be changed. It is a bad practice to mix cin >> x style input and getline.

Converting Between string objects and C-strings. Several functions require C-string values as arguments. The c_str() member can be used to retrieve a null terminated C-string r-value from a string object.

Tips

Lack of null termination in a C-String. The following declarations are not equivalent:

```
char shortStringA[] = {'a', 'b', 'c'};
```

and

```
char shortStringB[] = "abc";
```

The first is not null-terminated, while the second is. A quick way to convince the student that these are not equivalent is to put these declarations in a program and send them to the screen. You should get: "abc" from the first output statement, then "abc" followed by several ugly characters from memory locations following the array, up to the next '\0' (or the next memory location that is protected, resulting in a segmentations violation).

Pitfalls

Dangers in Using functions defined in <cstring>. A danger in using functions from <cstring> lies in using arrays of char without a null terminator as the preconditions prescribe. Each of the functions strcpy, strcat and strcmp depend on the terminating null character for their action. Improper use of these functions can result in crashed programs or even worse, security breaches. In addition, the iostream output function expects there to be a terminating null character. Otherwise, as we note elsewhere in this IRM, the output routine will run on until either we encounter a null character or protected memory. The strncpy, strncat, and strncmp functions are slightly safer as they take a third argument that specifies how many characters to process.

Not leaving enough space for the null terminator. Please be certain the student is aware of the necessity for providing space for the null terminator when using getline as well as in other C++ string uses. A typical call might look like this

```
cin.getline(stringVariable, MAX_CHARACTERS + 1);
```

The getline member function overwrites the first argument (let's call it arg1) with the smaller of strlen(arg1) characters (including the null terminator) and MAX_CHARACTERS. A null character is always appended after the MAX_CHARACTERS number of characters. This caution is expressed in the text's getline sidebar. It bears emphasis.

Using = and == with C-Strings. Since c-strings are pointers to arrays of chars, = simply copies the pointer and == compares pointer addresses. Instead the strcpy or strcmp routines should be used and adequate space must exist in the target prior to use of strcpy.

Unexpected \n in Input. When reading input you must account for every character, including the newline character. To clear the input stream of any leftover newlines, use the ignore function.

Mixing cin >> variable; and getline. When using `cin >> n`, the newline is left in the input stream. This is not a problem if all input is done this way, as the newline will be skipped on the next input since all whitespace is skipped. However, if getline is used in combination with `cin >> n`, then the getline call may read the newline and exit. Use the `cin.ignore(1000,'\n')` function to ignore any remaining newlines.