C++ Lecture 10

* C-strings, Strings, and Vectors
* CIS 251 • Shelby-Hoover Campus

char Arrays as C-strings

* Early versions of C++ did not include the data type string; instead, they relied on the use of char arrays to hold strings of text, a technique inherited from C
* A char array can hold a **C-string** with a length up to one character less than the size of the array
  + What separates a C-string array from an array of individual characters is the use of the **null character**, '\0'
  + The null character indicates that none of the remaining elements in the array are part of the C-string

Declaring, Using a C-string

* Remember that a char array’s size needs to be one element greater than the expected size of the   
  C-string:  
  char myCString[16]; // can store 15 characters plus '\0'
* The assignment operator can only be used to initialize a C-string when it is declared
  + char courseName[10] = "CIS 251"; /\* the elements at subscripts 0 through 6 hold the C-string, the element at subscript 7 holds the null character, and the elements at subscripts 8 and 9 are left empty \*/
  + char courseName[] = "CIS 251"; /\* since the size is omitted, the array is sized to hold the 7 characters of the C-string plus the null character, a total of 8 characters \*/
  + Don’t use the initialization list form of assignment
* The individual elements of a C-string can be accessed using normal array subscript notation, but be careful not to destroy the null character

C-string Length

* Most of the operations performed on a C-string require functions from the library cstring
* The number of characters in a char array being used for the C-string is not the same as the size
* The function strlen(myCString) returns the number of characters in the char array being used for the C-string (not including the null character)
* Example: converting the characters in a C-string to all uppercase:  
    
  char courseName[20] = "C++ Programming";  
  int i;  
  for (i = 0; i < strlen(courseName); i++)  
   courseName[i] = toupper(courseName[i]);

C-string Assignment

* To assign a new value to a C-string, use the function strcpy or strncpy
  + strcpy(courseName, "CIS 285");   
    /\* replaces the contents of courseName with "CIS 285" \*/
  + strcpy runs the risk of overflowing the array bounds or not leaving enough room for the null character
  + strncpy(courseName, "C++ Programming", 7);   
    /\* replaces the contents of courseName with no more than 7 characters from "C++ Programming" to prevent overflowing the usable elements of the array \*/
  + When using strncpy, the third argument should be no greater than one less than the size of the array (leave one array element for the null character)

C-string Comparison

* Remember: an array variable stores the address of the first element
* Using the normal comparison operators between arrays compares their addresses, not their contents
* To compare the contents of two C-string arrays, use the function strcmp or strncmp
  + The arguments to strcmp are the two C-string arrays to be compared; strncmp has a third argument limiting the number of characters to be compared
  + The functions return 0 if the two strings contain the same characters, a negative value if the first argument is “less than” the second, and a positive value if the first argument is “greater than” the second
  + The comparison uses the ASCII values of the characters (**lexicographic order**) rather than alphabetical order
  + A call to strcmp is safe even if the number of characters in each C-string is different; if the characters at the beginning of the longer string are identical to the shorter string, the longer string is “greater than” the shorter string

Comparison Example

* Comparing a first name and last name (note that first would have five elements and last would have four elements, with each array adding one element for the null character):  
    
  char first[] = "John", last[] = "Doe";  
    
  if (strcmp(first, last) < 0)  
   cout << "Your first name is less than your last name." << endl;  
  else if (strcmp(first, last) > 0)  
   cout << "Your first name is greater than your last name." << endl;  
  else  
   cout << "Your first name and last name are the same." << endl;

C-string Concatenation

* Concatenation is the process of adding the contents of one string to the existing contents of another string
* strcat(destination\_str, source\_str);  
  /\* adds the contents of source\_str to the end of the existing contents of destination\_str \*/
* strncat(dest\_str, src\_str, max\_char);  
  /\* limits the concatenation to max\_char characters from src\_str to the end of dest\_str to prevent array bounds problems \*/

C-string Input and Output

* A C-string array variable can be used in an output statement (console or file) just as any other variable is; the characters in the C-string array (up to, but not including, the null character) are displayed or written to the file
* A C-string array variable may also be used in a normal input statement (console or file), but the result may not be desirable
  + The standard input stream insertion operator, >>, stops at whitespace, so only the first “word” of the input will be included
  + If the first “word” has more characters in it than can be stored in the array with the null character, the program will use memory beyond the array to store the extra characters, leading to a runtime error

C-string Line Input

* The input streams have a member function, getline, that can be used to capture an entire line of text (up to a specific number of characters or the newline character) into a char array for a C-string
  + The format of this call is streamName.getline(arrayName, limit + 1);
  + The program reads all of the characters to the end of the line
  + If there are more than limit characters on the line, only the first limit characters will be stored in the array
  + The second argument needs to be 1 more than the number of characters to keep (include one element for the null character)
* Examples
  + cin.getline(myCharArray, 81);
  + myInputFileStream.getline(myCharArray, 21);

C-string Conversion

* The library cstdlib contains three functions that return a numeric equivalent of the characters in a C-string array
  + atoi(myCharArray) returns the int equivalent of myCharArray
  + atol(myCharArray) returns the long equivalent of myCharArray
  + atof(myCharArray) returns the double equivalent of myCharArray
* The argument to each of these functions can be a C-string array or a string literal ("123.45")
* If the argument contains any characters that cannot be treated as part of the numeric conversion type, the function will return 0

The C++ Class string

* The latest ANSI/ISO standard for C++ requires the presence of a string class type that isn’t as rigid as C-strings
  + The initial size of a string variable is not required
  + A string may grow or shrink in size at any time
  + Manipulation of a string object is much easier
* Older compilers may not support string
* The string class requires the library string and the standard namespace

Declaring a string

* Declarations for a string variable follow the same form as for primitive data types:  
    
  string myString1, myString2;
* A string variable may be initialized in one of two ways:  
    
  string firstName("John");  
  string lastName = "Smith";

string Input and Output

* A string variable may be displayed in any normal output statement (console or file)
* A string variable may also be used to capture input using the input stream insertion operator
  + This still only captures the characters up to whitespace (from the console or file)
  + No danger of having too many characters to fit
* Because the string library was written after iostream, the statement to capture a line of text into a string variable is different:  
    
  getline(inputStreamName, stringVarName);

string Line Input Details

* Arguments to the string version of getline:
  + The first argument is the name of the stream
  + The second argument is the name of the string variable to store the line
  + An optional third argument of type char can specify a different character to indicate the “end of the line”
* This version of getline returns the input stream so that it can be used with the input stream insertion operator:  
   getline(cin, str1) >> otherVariable;
* Remember: the insertion operator leaves the trailing whitespace in the stream, so you may need to clear the rest of a line containing input captured using the insertion operator so it won’t be captured in any call to getline:  
   cin.ignore(1000, '\n');

string Processing

* Assignment to a string variable uses the standard assignment operator (=), not a function
* Concatenation can be accomplished using + or +=:  
    
  string dept = "CIS", num = "251", courseName;  
  courseName = dept + " " + num;
* There are two ways to access an individual character in a string variable:
  + Array subscript notation (e.g., myString[0]) allows a program to use or change a character, but it does not check to see if the subscript is valid
  + The at method (e.g., myString.at(0)) allows a program to use or change a character, and it displays an error message if the subscript is invalid
  + Remember: the first character is at position 0, not 1

More string Processing

* The six relational comparison operators can be used to compare the contents of two string variables
  + If the string contents are of the same case, the comparison is strictly alphabetical
  + If of different cases, the comparison is lexicographic (based on the ASCII values of each character)
* To obtain the number of characters in a string variable, use the method .length()
* Display 8.7 lists other methods used for obtaining a substring, determining if a string is empty, inserting one string’s contents in the middle of another string, erasing a set of characters, and finding the position of one string in another string (if the latter contains the former)

string, C-string Conversion

* A C-string variable may be assigned directly to a string variable with no issues
* A program must use the strcpy function to assign a string variable to a C-string variable, but even this cannot be done without additional help
  + The string method .c\_str() returns the C-string equivalent of a string variable
  + This can be invoked on a string variable, and the result can be used as the second argument to strcpy
  + Example:   
      
    strcpy(myCharArray, myString.c\_str());
* Useful for functions that require a C-string (open)

Vectors

* Chapter 7 describes using an array to store a set of values without having to declare an individual variable for each value
* With an array, the number of elements must be predetermined and cannot change during execution
* A vector is similar to an array
  + Stores a set of values of the same type in a group with a single name and declaration
  + Each element is accessed via a subscript
* Unlike an array, a vector can grow and shrink at any time
* A program involving vectors requires the library vector and the standard namespace

Declaring a Vector

* The type and name of a vector must be included at the time of declaration:  
    
  vector<type> vectorName;  
    
  vector<int> grades;
* An optional initial size can be specified in parentheses after the name:  
    
  vector<int> grades(15);

Adding, Processing Elements

* If a value needs to be stored in an element to be added to the end of the vector (increasing the number of elements by one), use the method push\_back(value):  
    
   grades.push\_back(92);
* Each element can be processed using its subscript in square brackets, and a loop can be used to process all of the elements
* The number of elements currently in a vector can be accessed using the method size():  
    
  for (int i = 0; i < grades.size(); i++)  
   cout << grades[i] << endl;

Size and Capacity

* The method size() actually returns a value of type unsigned int, so you may need to use a variable of that type for your loop control variable (compiler-dependent)
* The method resize(newSize) sets the size of the vector to the new size, adding new elements or deleting elements that exceed the new size
* The amount of memory set aside for a vector is actually greater than its size to allow for growth
  + The method capacity() returns the number of elements for which the vector currently has space reserved (including the elements currently in use)
  + The method reserve(newCapacity) allows a programmer to modify the capacity (upward, but possibly not downward)