Object Oriented Software Development

Strings and Vectors

**Synopsis**

This laboratory session is intended to enable you to:

* Learn to declare, initialize, and use strings
* Learn about C-string values and variables
* Learn about the standard string class
* Learn about vectors

**Introduction to Strings C-string Functions**

In the previous lab, you learned to create arrays of different sizes to handle some numerical calculations.  In this lab, you will use arrays of characters which will be referred to as **strings**.  We can create a string in two different ways, 1) using the C-string style, and 2) using the string class.

To create an array using a C-string, we first need to declare an array of characters to store the characters that make the string (word).  You have to keep in mind that when you use the C-string style to create a string, you are required to have an extra space at the end for character '\0' which is known as the **null character**.  The null character, '\0', marks the end of the string.  In general, a string in which the null character marks its end is referred to as a **C-string**.  The following array is an example of a C-string, which allows us to store 7 or fewer characters:

char mystring[8];

Note this array can hold 8 characters, but the last one is kept for '\0'.  If we initialize this array with "Hello":

char mystring[8] = "Hello";

Or by using:

char mystring[] = "Hello";

Then we will have:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Imaginary**  **Memory**  **Address** | 1023 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 |
| **Array Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Indexed**  ***mystring* Variable** | mystring  [0] | mystring  [1] | mystring  [2] | mystring  [3] | mystring  [4] | mystring  [5] | mystring  [6] | mystring  [7] |
| **Array**  **Content** | H | e | l | l | o | \0 | ? | ? |

Note that we haven't used the last two elements and we do not know the content of those locations are.

Important Note:

Please remember that the above two statements used to initialize the array are equivalent, but the following IS NOT equivalent to these two:

char mystring[] = {'H', 'e', 'l', 'l', 'o'};

The above two initializations put a '\0', after the 'o', but this one doesn't put the '\0' at any place in the array.

**Comparing two strings**

When we dealt with numbers, it was easy to compare two numbers to see whether they are the same using ==.  However, when we deal with C-strings, we cannot use this method.  For example, if we want to check whether mystring is the same as yourstring, we cannot use the following:

char mystring[] = "Hello";   
char yourstring[] = "hello";

if(mystring == yourstring)      WRONG

In order to do this, we will use a function strcmp*.*

if( strcmp(mystring, yourstring) == 0)

The *strcmp* function will return 0 if the strings are the same.  The *strcmp* function uses the ordering relationship known as **lexicographic order** to determine whether these two are the same.

Important Note:

In order to use the *strcmp* function, you need to use the *#include <cstring>* in your programs.  The following table shows a list of some of the functions in that header file.

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Description** | **Cautions** |
| *strcpy*  **use:** *strcpy(Target\_String, Source\_String)* | Copies the C-string variable *Source\_String* into the C-string variable *Target\_String* | Does not check to see if the *Target\_String* is large enough to hold the *Source\_String* |
| *strncpy*  **use:** *strncpy(Target\_String, Source\_String, Limit)* | The same as *strcpy* except that at most *Limit* characters are copied. | Not implemented in all versions of C++ |

|  |  |  |
| --- | --- | --- |
| *strcat*  **use:** *strcat(Target\_String, Source\_String)* | Concatenates (put together) the C-string variable *Source\_String* onto the end of the C-string variable *Target\_String* | Does not check to see if the *Target\_String* is large enough to hold both strings together |
| *strncat*  **use:** *strncat(Target\_String, Source\_String, Limit)* | The same as *strcat* except that at most *Limit* characters are appended | Not implemented in all versions of C++ |
| *strlen*  *int length;*  **use:**  *length* **=** *strlen(Source\_String)* | Returns an integer equal to the length of the *Source\_String*, excluding the '\0' character. |  |
| *strcmp*  **use:**  *strcmp(String\_1, String\_2)* | * Returns 0 if *String\_1* and *String\_2* are the same. * Returns a value < 0 if *String\_1* is less than *String\_2* * Returns a value > 0 if *String\_1* is greater than *String\_2*   Thus, it returns a non-zero value when the two C-strings are not the same. | When the two C-strings are the same it returns 0, which converts to false.  That may cause confusion as you may expect to get 1 when the two C-strings are the same. |
| *strncmp*  **use:**  *strncmp(String\_1, String\_2, Limit)* | The same as *strcmp* except that at most *Limit* characters are compared | Not implemented in all versions of C++ |

|  |  |  |
| --- | --- | --- |
| *getline*  **use:**  *cin.getline(String\_variable, Max\_characters+1)*  *or*  *ifstream  s;  // an input stream*  *s.getline(String\_variable, Max\_characters+1)* | Reads a line of input up to *Max\_characters* length and places the C-string of characters on that line into a C-string variable *String\_variable* | If the line is longer than the maximum length defined, it only copies the first Max\_characters-1 characters on that line |

**Questions**

1. Locate c8e1-1 and type the program into an editor. Save this as c8e1.cpp then compile and run.

**C-String Input and Output**

Note the way we initialized the strings from the keyboard:

cin >> s1 >> s2;

and we display their values using:

cout << s1 << " and " << s2 << " are NOT the same \n";

When we did the reading, a blank space between the two strings that we type will tell the compiler which word goes to which string. Thus, if we would enter: "Black board", s1 would take "Black" and s2 would take "board".  The same thing would happen if we would have entered these two as:

Black

board

**C-String-to-Number Functions**

The functions **atoi**, **atol**, and **atof** can be used to convert a C-string of digits to the corresponding numeric value.  The first two functions convert the C-string to integers of type *int* or *long,* respectively.  The last one will convert the C-string to *double*.

Example:

int x = atoi("675");

This sets the value of x to 675.

In order to use these functions, you need to have #include <cstdlib>in your program.

**Questions**

1. Create a new program and call this c8e2.cpp. Copy the code from c8e1.cpp into c8e2.cpp and modify the c8e2.cpp program to ask users for two strings then:
   1. Append the second string to the end of the first string and display the resulting string.
   2. Display a message on whether or not the word is a Palindrome. Such strings are the same when you write them backward. For example: race + car = racecar … this word written backwards is also racecar, thus, it is a palindrome

**The Standard String Class**

In the previous activity, you learned to create C-strings.  The C-strings were simply arrays of characters terminated with the null character, '\0'.  To manipulate these strings, you used some functions.  In almost all cases, you as the programmer need to somehow keep track of number of elements that you have stored in a C-string.

The standard *string class* is defined in the library *<string>* and the definitions are placed in the *std* namespace.  The class string allows you to treat string values and string expressions very much like values of a simple type.  For example, when s1, s2, and s3 are objects of type string, to concatenate string s2 at the end of string s1 and to store the resulting string into s3, we will use:

s3 = s1 + s2;

Also, to initialized a string s4, we no longer need *strcpy* function and we can use:

s4 = "Hello World";

The string class has a constructor that initializes the string to empty and one that initializes the string to a desired string.  Following are the examples for these two:

string empty\_string;   
string something("hello");

The first one uses a constructor to create the empty\_string as an empty string and the second one initializes the string to "hello".  The following two lines are equivalent:

string something("hello");   
string something = "hello";

**Questions**

1. Locate c8e3-1 and type the program into an editor. Save this as c8e3.cpp then compile and run. This is an example in which a string class is used.

As you may have noticed, in this program "+" is overloaded in the string class such that it now does the concatenation of strings.

The standard string class has several functions that make things very easy.

**Questions**

1. Locate c8e4-1 and type the program into an editor. Save this as c8e4.cpp then compile and run. This program illustrates a few examples in which functions make things very easy.

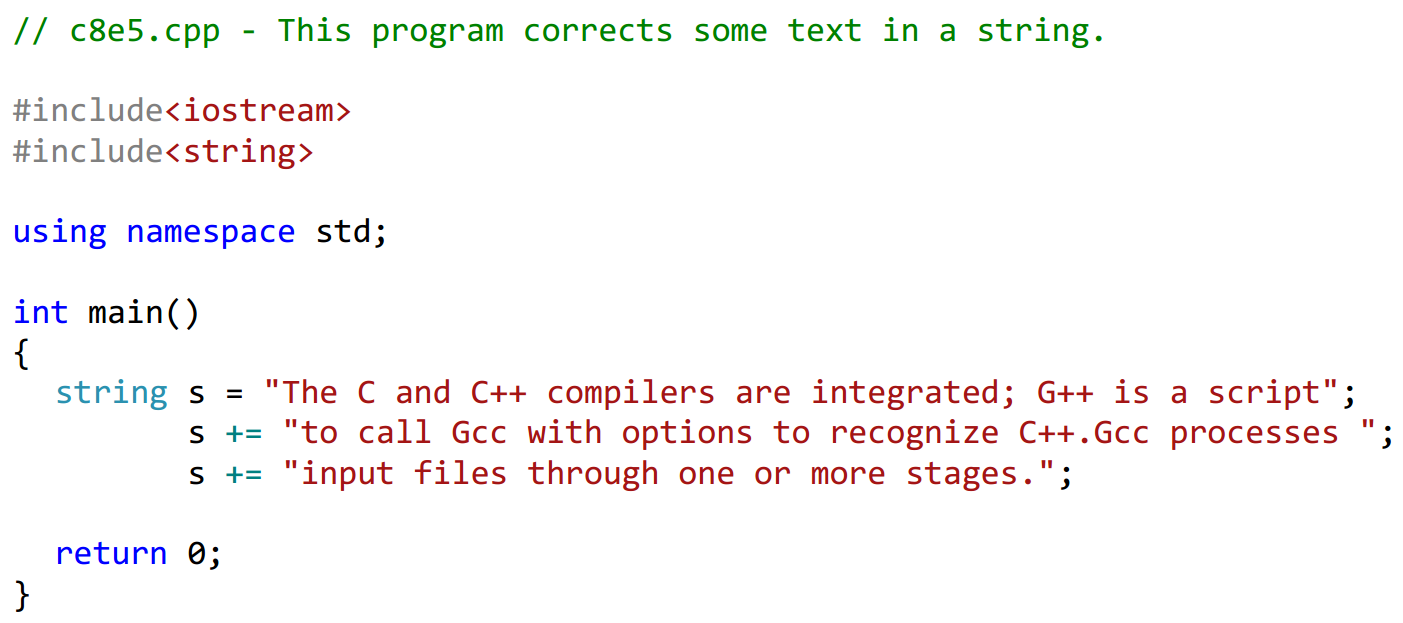
In the above program the statement:

temp1 = newphrase.substr(0, 18); and   
     temp2 = newphrase.substr(18, 10);

have something new in them.  The substr(position, length)function is a function that returns the substring of the calling object starting at *position* and having *length* characters.

**Questions**

1. Create a new program and name this c8e5.cpp. Type the following code into the program.



1. As you will immediately notice, I have written G++ for g++ and Gcc for gcc.  I want you to help me fix these errors by writing a C++ program to replace the incorrect statements with the correct ones.  For now, assume that these lines were read from the keyboard and placed in to the variable s.  You are not allowed to change the variable s, you must create a new variable and make use of the functions to replace the incorrect text from s.

**Converting Between String Objects and C-Strings**

In C++, you can automatically convert and store a C-string in a variable of type string.  Here is an example:

char a\_c\_string[] = "This is my C-string.";   
string string\_variable;   
string\_variable = a\_c\_string;

cout << string\_variable << endl;

This should display:  *This is my C-string*.

This will work fine but you can't convert a string variable to a C-string by going backward, thus,

a\_c\_string = string\_variable;       is ILLEGAL

Similarly,

strcpy(a\_c\_string, string\_variable);  is also ILLEGAL

However, the following:

strcpy(a\_c\_string, string\_variable.c\_str( ) );   is LEGAL

But, then again, the following:

a\_c\_string = string\_variable.c\_str( );   is also ILLEGAL

You may remember in a previous lab, we used an array of characters for the input and output file names.  We can rewrite that lab using the string class.

*"Remember you cannot use a stream of type input to write to a file and vice versa"*

**Questions**

1. Locate c8e6-1 and type the program into an editor. Save this as c8e6.cpp then compile and run. This program reads a file containing some integer values (we do not know how many values).  It displays the number, the number to the power of two, and the sum up to that point. At last it displays the average of these numbers. The program uses a string rather than a character array for the input of the filename.

**Vectors**

Vectors can be thought of as arrays that can grow (and shrink) in length as needed while your program is running.  So far, in C++ programs that you have written, the arrays that were created had a fixed length and the program could not change the length of the array once it started running.  Vectors serve the same purpose as arrays, but their length can be changed when the program is running.  In order to use vectors in your C++ programs, you need to include:

#include <vector>

Similar to arrays, vectors have a base type and will store a collection of values of its base type.  We declare a variable, v, for a vector with base type int as:

vector<int> v;

The notation vector<Base\_Type>is a **template class**, which means you can substitute any type for the Base\_Type*.* Vector elements are indexed starting with 0.  The array square brackets, [ and ], can be used to read or change these elements.

Example:

v[i] = 42;   
cout << "The ith element is " << v[i];

Restrictions: You can use the [ and ] to change the value of the ith element, but you cannot use this notation to initialize that element.  Thus, you can only change an element that has already been given a value.

To add an element to a vector, you will use the push\_backfunction.  You add elements to a vector in order of positions, first at position 0, then position 1, then 2, and so forth.  The function push\_back will add an element in the next available position.  Here is an example:

vector<double> sample;   
sample.push\_back(0.0);   
sample.push\_back(1.1);   
sample.push\_back(2.2);

These lines initialize the elements 0, 1, and 2 of the vector sample to 0.0, 1.1, and 2.2.

The number of elements in a vector is called its **size**. There is a member function, size( ), that can be used to determine the size of a vector. To display the number of elements in vector sample (size of the vector) we will use:

cout << sample.size( );

The size function returns an unsigned int*,* because the returned value should always be positive.

There is a vector constructor which takes one integer and will initialize the number of positions given as the argument.  For example,

vector<int> v(10);

initializes the first 10 elements of the vector v to 0.  Thus, if we put the

cout << sample.size( );

after vector<int> v(10); we will get a size of 10.  Once this is done, then you can use the [ and ] to set other values for those elements.

for(int i = 0; i < v.size( ); i++)   
    v[i] = 2\*i;

Important note: It is worth noting that the number of vector elements for which memory is allocated is called the **capacity** of that vector. The capacity is not the same thing as the **size,** which is number of elements with values.  To determine the capacity of a vector, we use the capacity( ) function.

cout << v.capacity( );

will display the capacity of vector v.

Whenever a vector runs out of capacity and needs room for an additional member, the capacity is automatically increased. The increase is usually done by doubling the existing capacity, but this may not be efficient.  There are two other functions that can be used to change the capacity of a vector.  The first function is called reserve(Limit)*.*  This function will explicitly increase the capacity by Limit.  For example:

v.reserve(32);

will set the capacity to at least 32 elements.  The following call:

v.reserve(v.size( ) + 10);

sets the capacity to at least 10 more than the number of elements currently in the vector.  Note that the reserve( )function can increase the capacity but can not be able to decrease the capacity.  You can reduce the capacity using the resize function:

v.resize(24);

This resizes the vector to 24 elements.  In the case that there are more than 24 elements in the vector, the extra elements will be distorted.

**Questions**

1. Locate c8e7-1 and type the program into an editor. Save this as c8e7.cpp then compile and run. This program will use a vector to read some values and to compute their average.

Run the program for the following cases:

1) 4 numbers: 2 2 2 2 the average in this case is 2

2) 6 numbers: 2 2 2 2 2 2 the average is 2 again

3) 8 numbers: 1 2 3 4 5 6 7 8 the average in this case is 4.5

Did you get the correct answers?

1. Replace the i in the statement sum/count with one of the v.size( ) or v.capacity( ) that you think should produce the correct results. Compile and run the program for the above three cases.  Did you get the correct results in all three cases?  Can you explain the reason for not getting the correct answer?

**Submission Instructions**

1. Create a zipped folder with the following files:
   * c8e1.cpp
   * c8e2.cpp
   * c8e3.cpp
   * c8e4.cpp
   * c8e5.cpp
   * c8e6.cpp
   * c8e7.cpp
2. Upload to the Moodle Link: Chapter 8 Lab Exercises