Week 8

Intro to Computer Science - CSI 1430

Hello and welcome to the weekly resources for CSI 1430: Introduction to Computer Science! This week's resource will be covering the course material that will be covered in week 8 of the course.

The semester is halfway through, so the pace for most classes picks up a little. Make sure to try to keep your spirits high, do small things to reward yourself for finishing assignments and such, and just try to give yourself some breaks! **Just as a reminder:** These documents are intended for going over concepts that are discovered in class, as they are a concise and to the point review of the materials. But remember, these documents are not a replacement for lecture and they cannot cover everything, so make sure you get help on any specific concepts you struggle with as well as go over lecture notes, programs, and the textbook.

Reminder: If you have any questions about these study guides, group tutoring sessions, private 30 minutes tutoring appointments, the Baylor Tutoring Youtube channel or any tutoring services we offer, please visit our website https://baylor.edu/tutoring or call our drop in center during open business hours. Monday-Thursday 9am - 8pm on class days (254) 710-4135.

Keywords: *Vectors, String and CString Functions*

TOPIC OF THE WEEK

Vectors

Last week we talked about Arrays within C++. One of the limitations of an Array is it having to have a fixed size. This means that once the array is declared, then there can not be a change to the size of an array. To combat this, we can use what is called a vector. A vector has the ability to change size as we add more elements to it. Vectors are included within the C++ Standard Template Library, the STL, and we can use vectors by using the '#include <vector>' preprocessor directive.

For example, you can declare a vector of integers, named *myVector,* by doing the following declaration:

```
vector<int> myVector();
```

This creates an empty vector of integers. We are able to add numbers to the vector by using the push_back() function! If we wanted to add the numbers 1 & 2 to the *myVector* vector, then we can do the following:

```
myVector.push_back(1);
myVector.push_back(2);
```

Now we have a vector with the value, 1, in position 0...and value, 2, in position 1. Also, although we know that there are two numbers inside of the vector, there are instances, such as looping, that we would like to know the size of the vector internal to the program (rather than just knowledge). For this, we can use a size() function similar to that of strings, in order to find out the current size of a vector.

```
myVector.size();          //Returns a size of 2
```

Lastly, we can also remove values from a vector by using the pop_back() function. Now there is a limitation of the pop_back function...which is that we can only delete the LAST value from a vector (a.k.a from the back of the vector). Therefore, if we use the following, we will only be able to remove the value 2:

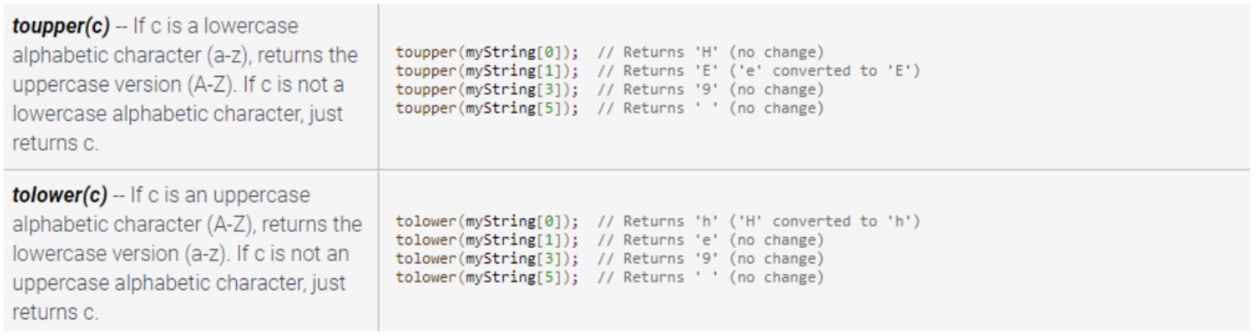
```
myVector.pop_back();            // Removes the value 2 from the vector
```

Besides that, most of the vector functionality is the same as that of an array. We can still access elements that already exist by position. This can be used for access or modification of the position in the vector. For example, if we wanted to change the first position of the current array to the value 100, we can still do ‘myVector[0] = 100;’

HIGHLIGHTS

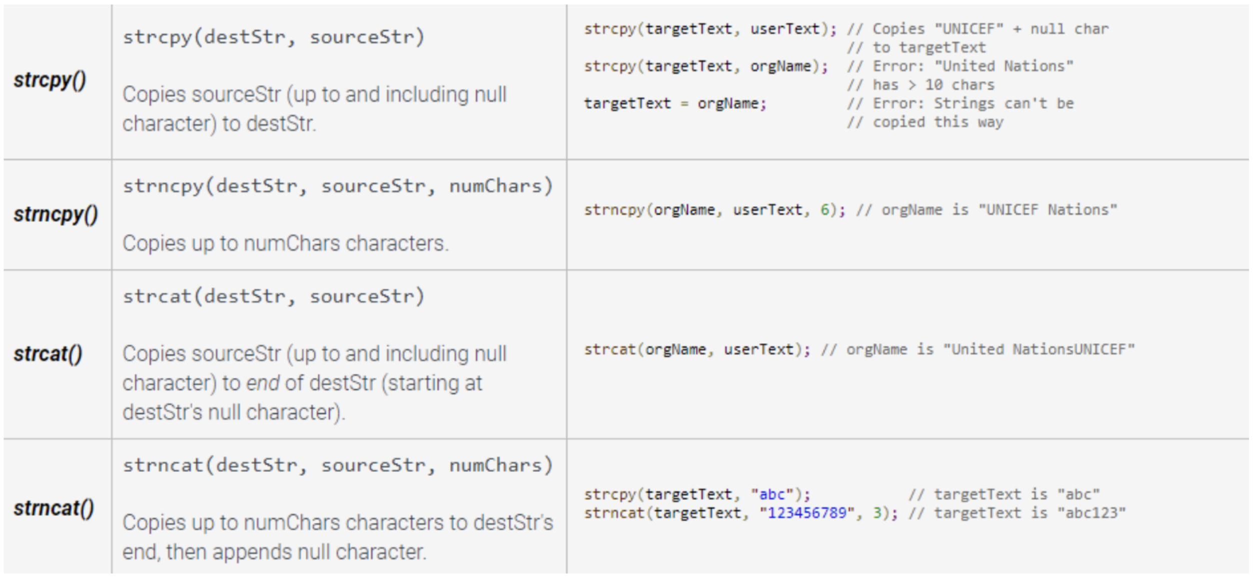
String and CString Functions

One common thing that I have been asked in almost every one of my sessions mentions something about string access or manipulation. Within the string library there are so many different functions that are available to achieve these types of access. All of these functions are located within the standard library for strings. Listed below are the most important functions to know about for this class.



toupper(c) -- If c is a lowercase alphabetic character (a-z), returns the uppercase version (A-Z). If c is not a lowercase alphabetic character, just returns c.	`toupper(myString[0]);  // Returns 'H' (no change)`
	`toupper(myString[1]);  // Returns 'E' ('e' converted to 'E')`
	`toupper(myString[3]);  // Returns '9' (no change)`
	`toupper(myString[5]);  // Returns ' ' (no change)`

tolower(c) -- If c is an uppercase alphabetic character (A-Z), returns the lowercase version (a-z). If c is not an uppercase alphabetic character, just returns c.	`tolower(myString[0]);  // Returns 'h' ('H' converted to 'h')`
	`tolower(myString[1]);  // Returns 'e' (no change)`
	`tolower(myString[3]);  // Returns '9' (no change)`
	`tolower(myString[5]);  // Returns ' ' (no change)`



strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including null character) to destStr.
		`strcpy(targetText, userText); // Copies "UNICEF" + null char`
		`                               // to targetText`
		`strcpy(targetText, orgName);  // Error: "United Nations"`
		`                               // has > 10 chars`
		`targetText = orgName;          // Error: Strings can't be`
		`                               // copied this way`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters.	`strncpy(orgName, userText, 6); // orgName is "UNICEF Nations"`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr (up to and including null character) to *end* of destStr (starting at destStr's null character).	`strcat(orgName, userText); // orgName is "United NationsUNICEF"`

strncat()	`strncat(destStr, sourceStr, numChars)`
	
	Copies up to numChars characters to destStr's end, then appends null character.
		`strcpy(targetText, "abc");            // targetText is "abc"`
		`strncat(targetText, "123456789", 3); // targetText is "abc123"`

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the cstring library.
		`if (strchr(orgName, 'U') != NULL) { // 'U' exists in orgName?`
		`   ...  // 'U' exists in "United Nations", branch taken`
		`}`
		`if (strchr(orgName, 'u') != NULL) { // 'u' exists in orgName?`
		`   ...  // 'u' doesn't exist (case matters), branch not taken`
		`}`

strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);    // Assigns 14 to x`
		`x = strlen(userText);   // Assigns 6 to x`
		`x = strlen(targetText); // Error: targetText may lack null char`

strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName, "United Nations") == 0) {`
		`   ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) == 0) {`
		`   ... // Not equal, branch not taken`
		`}`

isalpha(c) -- Returns true if c is alphabetic: a-z or A-Z.	`isalpha('A');          // Returns true`
	`isalpha(myString[0]);  // Returns true because 'H' is alphabetic`
	`isalpha(myString[3]);  // Returns false because '9' is not alphabetic`

isdigit(c) -- Returns true if c is a numeric digit: 0-9.	`isdigit(myString[3]);  // Returns true because '9' is numeric`
	`isdigit(myString[4]);  // Returns false because ! is not numeric`

isalnum(c) -- Returns true if c is alphabetic or a numeric digit. Thus, returns true if either isalpha or isdigit would return true.	`isalnum('A');          // Returns true`
	`isalnum(myString[3]);  // Returns true because '9' is numeric`

isspace(c) -- Returns true if character c is a whitespace.	`isspace(myString[5]);  // Returns true because that character is a space ' '.`
	`isspace(myString[0]);  // Returns false because 'H' is not whitespace.`

islower(c) -- Returns true if character c is a lowercase letter a-z.	`islower(myString[0]);  // Returns false because 'H' is not lowercase.`
	`islower(myString[1]);  // Returns true because 'e' is lowercase.`
	`islower(myString[3]);  // Returns false because '9' is not a lowercase letter.`

isupper(c) -- Returns true if character c is an uppercase letter A-Z.	`isupper(myString[0]);  // Returns true because 'H' is uppercase.`
	`isupper(myString[1]);  // Returns false because 'e' is not uppercase.`
	`isupper(myString[3]);  // Returns false because '9' is not an uppercase letter.`

isblank(c) -- Returns true if character c is a blank character. Blank characters include spaces and tabs.	`isblank(myString[5]);  // Returns true because that character is a space ' '.`
	`isblank(myString[0]);  // Returns false because 'H' is not blank.`

isxdigit(c) -- Returns true if c is a hexadecimal digit: 0-9, a-f, A-F.	`isxdigit(myString[3]); // Returns true because '9' is a hexadecimal digit.`
	`isxdigit(myString[1]); // Returns true because 'e' is a hexadecimal digit.`
	`isxdigit(myString[6]); // Returns false because 'G' is not a hexadecimal digit.`

ispunct(c) -- Returns true if c is a punctuation character. Punctuation characters include: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~	`ispunct(myString[4]); // Returns true because '!' is a punctuation character.`
	`ispunct(myString[6]); // Returns false because 'G' is not a punctuation character.`

isprint(c) -- Returns true if c is a printable character. Printable characters include alphanumeric, punctuation, and space characters.	`isprint(myString[0]);  // Returns true because 'H' is a alphabetic.`
	`isprint(myString[4]);  // Returns true because '!' is punctuation.`
	`isprint(myString[5]);  // Returns true because that character is a space ' '.`
	`isprint('\0');         // Returns false because the null character is not printable`

CHECK YOUR LEARNING

1. **If we wanted to know if the 3rd character within the string, *myString*, is an uppercase character, how would we check that?**
2. **What is a limitation of the pop_back() function?**
3. **True or False: The positions of a vector can be accessed the exact same as those of arrays.**

THINGS STUDENTS MAY STRUGGLE WITH

1. Note that vectors start at an index of 0, just like arrays. This is super important to remember due to making sure you dont cause issues within your program. For most looping and such, the size() function will work perfectly to make sure we are looping over the correct number of elements, since it changes with how many elements are in the vector automatically. However, understanding this is important for accessing the elements of the vectors correctly.
2. Vectors are able to support all the same data types that arrays can. This includes, ints, doubles, strings, chars, etc…
3. Take time to get used to the different *string* manipulation and mutation functions… Sometimes they won't behave like you would like them to and need a little bit of playing around with to fully understand. Understanding string access and manipulation is a common coding area in computer science and software engineering interviews, so there is some merit to diving into understanding their behavior.

Thanks for checking out these weekly resources! Don't forget to check out our website for group tutoring times, video tutorials and lots of other course resources: https://baylor.edu/tutoring. The answers to the 'Check Your Learning' questions are below.

Answers:

1. isUpper(myString[2]);   //Remember that the 3rd element, is the 2nd position
2. pop_back() can only delete the last element of a vector
3. True

Note: All tables were taken from the Baylor CS 1430 zyBook