6.13.2D Vector In C++ With User Defined Size

Students:
Section 6.14 is a part of 2 assignments: CSC108 CH06.10-6.27 C6B
Please browse to this assignment through BlackboardLearn so zyBooks knows where to send your activity. Learn more
Includes: CA
Due: 05/06/2025, 11:59 PM EDT

6.14 Char arrays / C strings

A programmer can use an array to store a sequence of characters, known as a ***string***. Char arrays were the only kinds of strings in C++'s predecessor language C, and thus are sometimes called **C strings** to distinguish them from C++'s string type. An example is `char movieTitle[20] = "Star Wars";`. Because a string can be shorter than the character array, a string in a char array must end with a special character known as a ***null character***, written as '\0'. Given a string literal like "Star Wars", the compiler automatically appends a null character.

PARTICIPATION ACTIVITY 6.14.1: A char array declaration and initialization with null-terminated string.

Captions

Feedback?

A char array of size 20 can store strings of lengths 0 to 19. The longest string is 19, not 20, since the null character must be stored.

If a char array is initialized when declared, then the char array's size may be omitted, as in `char userText[] = "Hello";`. The compiler determines the size from the string literal, in this case 6 = 1 (for the null character), or 7.

An array of characters ending with a null character is known as a ***null-terminated string***.

Output streams automatically handles null-terminated strings, printing each character until reaching the null character that ends the string.

Figure 6.14.1: Printing stops when reaching the null character at each string's end.

```
#include <iostream>
using namespace std;

int main() {
   char cityName[20] = "Forest Lake";  // Compiler appends null char

   // In each cout, printing stops when reaching null char
   cout << "City:" << endl;          // Compiler appends null char to "City:"
   cout << cityName << endl;

   return 0;
}
```

```
City:
Forest Lake
```

Feedback?

PARTICIPATION ACTIVITY 6.14.2: Char array strings.

Indicate whether the array declaration and initialization are appropriate.

1) `char firstName[10] = "Amoxy";`
   - True
   - False

2) `char lastName[10] = "Michelson";`
   - True
   - False

3) `char favoriteMuseum[10] = "Smithsonian";`
   - True
   - False

4) Printing catBreed will print 19 characters.
   `char catBreed[20] = "Persian";`
   - True
   - False

Feedback?

After a string is declared, a programmer may not later assign the string as in `movieTitle = "Indiana Jones";`. That statement tries to assign a value to the char array variable rather, than copying each character from the string on the right into the array on the left. Functions exist to copy strings, such as strcpy(), discussed elsewhere.

A programmer can traverse a string using a loop that stops when reaching the null character.

A common error is to look for the string's array size rather than stopping at the null character. Such looping visits unused array elements beyond the null character. An even worse common error is to loop beyond the last valid element, which visits memory locations that are not part of the array. These errors are illustrated below. Notice the strange characters that are output as the contents of other memory locations are printed out, the program may also crash.

Figure 6.14.2: Traversing a C string.

```
#include <iostream>
using namespace std;

int main() {
   char userStr[20] = "12345678901234567890";  // Input string
   int i;

   // Prompt user for string input
   cout << "Enter string (<20 chars): ";
   cin >> userStr;

   // Print string
   cout << endl << userStr << endl;

   // Look for '0'
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (userStr[i] == '0') {
         cout << "Found '0'." << endl;
      }
   }
   cout << endl;

   // The following is an ERROR.
   // May print chars it shouldn't.
   // Accesses beyond valid index char.
   cout << "**" << // Print opening "
   for (i = 0; i < 20; ++i) {
      cout << userStr[i];
   }
   cout << "**" << endl;  // Print closing "

   // The following is an even WORSE ERROR.
   // Accesses beyond valid index range.
   // Program may crash.
   cout << "**" << // Print opening "
   for (i = 0; i < 30; ++i) {
      cout << userStr[i];
   }
   cout << "**" << endl;  // Print closing "

   return 0;
}
```

```
Enter string (<20 chars): test@gmail.com

test@gmail.com
Found '0'.

**test@gmail.com0T0P**
**test@gmail.com0T0P0P!**
```

Feedback?

The above output is machine and compiler dependent. Also, some values aren't printable so don't appear in the output.

PARTICIPATION ACTIVITY 6.14.3: C string errors.

Given the following char array declaration, which of the following code snippets are bad? `char userText[10] = "Cat";`

1) `for (i = 0; userText != '\0'; ++i) {`
   `  // Print userText[i]`
   `}`
   - OK
   - Bad

2) `for (i = 0; userText[i] != '\0'; ++i) {`
   `  // Print userText[i]`
   - OK
   - Bad

3) `for (i = 0; i < 10; ++i) {`
   `  // Print userText[i]`
   `}`
   - OK
   - Bad

4) `for (i = 0; i < 4; ++i) {`
   `  // Print userText[i]`
   `}`
   - OK
   - Bad

5) `userText = "Bus";`
   - OK
   - Bad

Feedback?

Yet another common error with C strings is for the program user to enter a string larger than the character array. That may cause the input statement to write to memory locations outside the array's locations, which may corrupt other parts of program or data, and typically causes the program to crash.

zyDE 6.14.1: Reading in a string too large for a C string.

Run the program, which simply reads an input string and prints it one character at a time. Then, lengthen the input string beyond 10 characters, and run again. The program might work, if the extra memory locations being assigned don't matter. Try larger and larger strings, and see if the program fails (be sure to scroll to the bottom of the output to look for erroneous output or an error message).

Load default template...

```
#include <iostream>
using namespace std;

int main() {
   char userStr[10]; // Input string
   int i;

   // Prompt user for string input
   cout << "Enter string (<10 chars): ";
   cin >> userStr;

   // Print 1 char at a time
   cout << endl;
   for (i = 0; userStr[i] != '\0'; ++i) {
      cout << userStr[i] << endl;
   }

   cout << endl;
```

Hello

Run

Feedback?

C string usage is fraught with common errors. C++ introduced its own string type, as in `string myString;` and accessible after `#include <string>`, to reduce those errors and increase programmer convenience. C strings are still used in some legacy code and are thus good to learn. C++ provides common functions for handling C strings, which can be used by including the following: `#include <cstring>`.

The following programs for illustration, showing how a string is made up of individual character elements followed by a null character. Normally a programmer would not create a string that way.

Figure 6.14.3: A C string is an array of characters, ending with the null character.

```
#include <iostream>
using namespace std;

int main() {
   char nameArr[5];

   nameArr[0] = 'A';
   nameArr[1] = 'l';
   nameArr[2] = 'a';
   nameArr[3] = 'n';
   nameArr[4] = '\0';  // Null character

   cout << nameArr << endl;

   nameArr[3] = 'i';   // Oops, overwrote null char
   cout << nameArr << endl;  // "Alpha" still work

   return 0;
}
```

```
Alan
Alai
```

Feedback?

When printing a string stored within a character array, each character within the array will be printed until the null character is reached. If the null character is omitted, the program would print whatever values are found in memory after the array, until a null character happens to be encountered. Omitting the null character is a serious logical error.

It just so happens that the null character '\0' has an ASCII encoding of 0. Many compilers initialize memory to 0s. As such, omitting the '\0' in the above program would not always cause erroneous execution. Like a nail in the road, that bug in your code is just waiting to wreak havoc.

PARTICIPATION ACTIVITY 6.14.4: C string without null character.

Given:

```
char userText[10];
userText[0] = 'A';
userText[1] = 'a';
userText[2] = 'r';
userText[3] = 'l';
...
userText[3] = 'a';
```

1) The first four characters in userText are now: Cars.
   - True
   - False

2) The compiler generates an error, because element 3 is the null character and can't be overwritten.
   - True
   - False

3) Printing userText should work fine because the new string is 4 characters, which is still much less than the array size of 10.
   - True
   - False

Feedback?

CHALLENGE ACTIVITY 6.14.1: Char arrays / C strings.

zyDE 6.14.1: Hello!

Start

Declare and initialize a C string called inputString with the value "1234".

Ex: The output is:

```
The string is: 1234
```

```
#pragma GCC diagnostic push
#pragma GCC diagnostic error "-Wwrite-strings" // Ensures that the intended style of solution is used
#include <iostream>
using namespace std;

int main() {
   char /* Your code goes here */

   cout << "The string is: " << inputString << endl;

   return 0;

#pragma GCC diagnostic pop
```

1 2 3

Check Next level

Feedback?

How was this section? Provide section feedback

Activity summary for assignment: CSC108 CH06.10-6.27 C6B
Due: 05/06/2025, 11:59 PM EDT

0 / 30 points

Please browse to this assignment through BlackboardLearn so zyBooks knows where to send your activity. Learn more

Completion details

6.15 C-String library functions