CSCI 123 Introduction to Programming Concepts in C++

Brad Rippe

String and Vectors

Overview

8.1

An Array Type for Strings

An Array Type for Strings

- C-strings can be used to represent strings of characters
  - C-strings are stored as arrays of characters
  - C-strings use the null character '\0' to end a string
    - The Null character is a single character
  - To declare a C-string variable, declare an array of characters:

```
char s[9];
```

C-string Details

- Declaring a C-string as char s[10] creates space for only nine characters
  - The null character terminator requires one space
- A C-string variable does not need a size variable
  - The null character immediately follows the last character of the string
- Example:

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	?	?

C-string Declaration

- To declare a C-string variable, use the syntax:

```
type arrayName[size+1];
```

  – + 1 reserves the additional character needed by '\0'

Initializing a C-string

- To initialize a C-string during declaration:

```
char message[20] = "Hi there.";
```

  - The null character '\0' is added for you

- Another alternative:

```
char fileName[] = “input.txt";
```

  but **not** this:

```
char fileName[] = {‘i', ‘n', ‘p', ‘u', ‘t',
                   ‘.', ‘t', ‘x', ‘t'};
```

C-string error

- This attempt to initialize a C-string does not cause the \0 to be inserted in the array

```
char fileName[] = {‘i', ‘n', ‘p', ‘u', ‘t',
                       ‘.', ‘t', ‘x', ‘t'};
```

Don't Change '\0'

- Do not to replace the null character when manipulating indexed variables in a C-string
  - If the null character is lost, the array cannot act like a C-string
    - Example:

```
int size = 0;
while(myString[size] != '\0')  {
    myString[size] = 'X';
    size++;
}
```

  - **This code depends on finding the null character!**

Safer Processing of C-strings

- The loop on the previous slide depended on finding the '\0' character
  - It would be wiser to use this version in case the '\0' character had been removed

```
int index = 0;
while(myString[index] != '\0’ &&  index < size) {
  myString[index] = 'X';
  index++;
}
```

Assignment With C-strings

- This statement is illegal:

```
myString = "Hello";
```

  - This is an assignment statement, not an initialization
  - The assignment operator does not work with C-strings

Assignment of C-strings

- A common method to assign a value to a C-string variable is to use strcpy, defined in the cstring library
  - Example:

```
#include <cstring>
   …
char myString[11];
strcpy(myString, "Hello");
```

Places "Hello" followed by the null character in myString

A Problem With strcpy

- *strcpy* can create problems if not used carefully
  - *strcpy* does not check the declared length of the first argument
  - It is possible for *strcpy* to write characters beyond the declared size of the array

Problem copying cstrings

```
char cString1[] = "Please email my message to the
   class!";
char cString2[] = "";
strcpy(cString2, cString1);
cout << "cString1 = " << cString1 << endl;
cout << "cString2 = " << cString2 << endl;
```

What happens here?

cString2 only gets allocated enough memory for an empty string

cString1 is given the memory after cString2

when cString1 is copied to cString2, it overwrites part of

cString1 this is bad

A Solution for strcpy

- Many versions of C++ have a safer version of *strcpy* named *strncpy*
  - *strncpy* uses a third argument representing the maximum number of characters to copy
  - Example:

```
char myString[10];
strncpy(myString, myStr, 9);
```

This code copies up to 9 characters into myString, leaving one space for '\0‘

If third argument is less than or equal to the length of source, a null character is not appended automatically to the copied string. If count is greater than the length of source, the destination string is padded with null characters up to length count. strncpy does not check for sufficient space in destination; it is therefore a potential cause of buffer overruns. Keep in mind that count limits the number of characters copied; it is not a limit on the size of destination

Problem copying cstrings

```
char cString1[] = "Please email my message to the
  class!";
char cString2[] = "";
strncpy(cString2, cString1, cString1Size);
cout << "cString1 = " << cString1 << endl;
cout << "cString2 = " << cString2 << endl;
```

What happens here now?

Same problem…. But if we know the size of cString1 we can limit this issue

Problem copying cstrings

```
char cString1[] = "Please email my message to the
  class!"; // size is 37 +1 for the null
char cString2[38] = "";
strncpy(cString2, cString1, cString1Size);
cout << "cString1 = " << cString1 << endl;
cout << "cString2 = " << cString2 << endl;
// assume cString1Size is 38
```

How do we fix the problem?

Set the size of cString2 to the number of characters in the source string. This will allocate enough memory so that the issue will be avoided.

Depre-what?

warning C4996: 'strncpy' was declared deprecated

- Deprecated functions means that these particular functions are not recommended to be used. There are better functions which should be used instead.

strcpy_s

From Microsoft MSDN:

"For example, the **strcpy** function has no way of telling if the string that it is copying is too big for its destination buffer. However, its secure counterpart, **strcpy_s**, takes the size of the buffer as a parameter, so it can determine if a buffer overrun will occur. If you use **strcpy_s** to copy eleven characters into a ten-character buffer, that is an error on your part; **strcpy_s** cannot correct your mistake, but it can detect your error and inform you by invoking the invalid parameter handler."

You can use strcpy_s to avoid buffer overflow and the other secure versions of the deprecated functions (*in VS 2005*)

http://msdn2.microsoft.com/en-us/library/8ef0s5kh(VS.80).aspx

This is part of the ISO Standard and not the ANSI standard

ISO - International Organization for Standardization

American National Standards Institute - ANSI

Windows Only

strcpy_s

- **Parameters**
  - strDestination Location of destination string buffer
  - sizeInBytes, sizeInWords Size of the destination string buffer.
  - strSource Null-terminated source string buffer.

- Returns
  - Zero if successful;
    an error otherwise.

Windows Only

== Alternative for C-strings

- The = = operator does not work as expected with C-strings
  - The predefined function strcmp is used to compare C-string variables
  - Example:

```
#include <cstring>
  …
if (strcmp(string1, string2))
  cout << "Strings are not the same.";
else
  cout << "String are the same.";
```

strcmp's logic

- strcmp compares the numeric codes of elements in the C-strings a character at a time
  - If the two C-strings are the same, strcmp returns 0
    - 0 is interpreted as false
  - As soon as the characters do not match
    - strcmp returns a negative value if the numeric code in the first parameter is less
    - strcmp returns a positive value if the numeric code in the second parameter is less
    - Non-zero values are interpreted as true

strcmp()

```
char cString1[] = "Please email my message to the class!";
char cString2[50];

// strcmp compares strings lexicographically returns
// < 0 string1 less than string2 ;0 string1 identical to string2 ; > 0 string1 greater than string2
if(cString1 == cString2) {
	cout << "string 1 is equal to string 2\n";
} else {
	cout << "string 1 is not equal to string 2\n";
}

if(strcmp(cString1, cString2) < 0) {
	cout << "string 1 is less than string 2\n";
} else if(strcmp(cString1, cString2) == 0) {
	cout << "string 1 equal to string 2\n";
} else if(strcmp(cString1, cString2) > 0) {
	cout << "string 1 is greater than string 2\n";
}
```

More C-string Functions

- The **cstring** library includes other functions
  - strlen returns the number of characters in a string

```
int x = strlen(myString);
```

  - strcat concatenates two C-strings
    - The second argument is added to the end of the first
    - The result is placed in the first argument
    - Example:

```
char aString[20] = "The rain";
strcat(aString, "in Spain");
```

Now aString contains "The rainin Spain"

The strncat Function

- strncat is a safer version of strcat
  - A third parameter specifies a limit for the number of characters to concatenate
  - Example:

```
char aString[20] = "The rain";
strncat(aString, "in Spain", 11);
```

C-strings as Arguments and Parameters

- C-string variables are arrays
- C-string arguments and parameters are used just like other arrays
  - If a function changes the value of a C-string parameter, it is best to include a parameter for the declared size of the C-string
  - If a function does not change the value of a   C-string parameter, the null character can detect the end of the string and no size argument is needed

C-string Output

- C-strings can be output with the insertion operator
  - Example:

```
char msg[] = “What a “;
cout << msg << " Class!\n”;
```

C-string Input

- The extraction operator >> can fill a C-string
  - Whitespace ends reading of data
  - Example:

```
char a[80];
char b[80];
cout << "Enter input: " << endl;
cin >> a  >> b;
cout << a << b << "End of Output";
```

could produce:

```
Enter input:
Do be do to you!
DobeEnd of Output
```

Reading an Entire Line

- Predefined member function *getline* can read an entire line, including spaces
  - *getline* is a member of all input streams
  - *getline* has two arguments
    - The first is a C-string variable to receive input
    - The second is an integer, usually the size of the first argument specifying the maximum number of elements in the first argument *getline* is allowed to fill

Using getline

- The following code is used to read an entire line including spaces into a single C-string variable

```
int lineLength = 0;
int numberOfLines = 0;
char line[LINE_SIZE];
while(true) {
        aInFile.getline(line, LINE_SIZE);  // read the whole line
        if(aInFile.eof()) break;
        lineLength = aInFile.gcount();
        numberOfLines++;
}
cout << "The last line txt " << line << endl;
cout << "The line length " << lineLength << endl;
cout << "The number of lines " << numberOfLines << endl;
```

and could produce:

```
The last line txt Title: Spurs 86, Heat 83
 The line length 24
 The number of lines 432
```

cstring_getline.cpp

getline wrap up

- *getline* stops reading when the number of characters, **less one**, specified in the second argument have been placed in the C-string
  - one character is reserved for the null character
  - *getline* stops even if the end of the line has not been reached

getline and Files

- C-string input and output work the same way with file streams
  - Replace cin with the name of an input-file stream

```
    iStream >> myString;
    iStream.getline(myString, 80);
```

  - Replace cout with the name of an output-file stream

```
    outStream << myString;
```

getline syntax

- Syntax for using *getline* is

```
cin.getline(stringVar, maxChars + 1);
```

  - cin can be replaced by any input stream
  - maxChars + 1 reserves one element for the null character

C-String to Numbers

- “86" is a string of characters
- 86 is a number
- When doing numeric input, it is useful to read input as a string of characters, then convert the string to a number
  - Reading money may involve a dollar sign
  - Reading percentages may involve a percent sign

C-strings to Integers

- To read an integer as characters
  - Read input as characters into a C-string, removing unwanted characters
  - Use the predefined function atoi to convert the C-string to an int value
    - Example: `atoi(“86")` returns the integer 86

      atoi("#33") returns 0 because # is not a digit
  - #include <cstdlib>

C-string to long

- Larger integers can be converted using the predefined function atol
  - atol returns a value of type ***long***
  - ***long int***

C-string to double

- C-strings can be converted to type double using the predefined function *atof*
- *atof* returns a value of type **double**
  - Example:

  atof("9.99") returns 9.99

  atof("$9.99") returns 0.0 because the
  $ is not a digit

Library cstdlib

- The conversion functions

  atoi

  atol

  atof

  are found in the library cstdlib
- To use the functions use the include directive

```
#include <cstdlib>
```

cctype library

Helpful character functions, when examining characters for conversions

isdigit(c)	Returns true if c is a digit
isalpha(c)	Returns true if c is a letter
isalnum(c)	Returns true if c is a letter or a digit
islower(c)	Returns true if c is a lowercase letter
isupper(c)	Returns true if c is a uppercase letter
isspace(c)	Returns true if c is a whitespace character
isprint(c)	Returns true if c is a printable character occupies space on the device
isgraph(c)	Returns true if c is a printable character excluding ‘ ‘
ispunct(c)	Returns true if c is a punctuation character
iscntrl(c)	Returns true if c is a control character ‘\n’, ‘\f’, ‘\v’, ‘\a’, and ‘\b’

Numeric Input

- We now know how to convert C-strings to numbers
- How do we read the input?
  - Function getATeam()

```
string fileData;
char score[20];
// get the first team
aInFile >> fileData;
aTeams[aTeamSize][aTeamIndex] = fileData;
aInFile >> score;
cleanScore(score);
aWinsLosses[aTeamSize][aTeamIndex] = atoi(score);
```

Uses atoi to convert the "cleaned-up" C-string to int
From the data in nbaScores.txt

Title: Sonics 96, Bobcats 89
Description: Ray Allen's spinning three-point play with 1:25 left capped a 34-point game, and the Seattle SuperSonics pulled away late for a 96-89 win over the Charlotte Bobcats on Sunday night.
Link: http://www.nba.com/games/20070304/CHASEA/boxscore.html?rss=true

Section 8.1 Conclusion

- Can you
  - Describe the benefits of reading numeric data as characters before converting the characters to a number?
  - Write code to do input and output with C-strings?
  - Use the atoi, atol, and atof functions?
  - Identify the character that ends a C-string?

8.2

The Standard **`string`** Class

The Standard string Class

- The string class allows the programmer to treat strings as a basic data type
  - No need to deal with the implementation as with C-strings
- The string class is defined in the string library and the names are in the standard namespace
  - To use the string class you need these lines:

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

Assignment of Strings

- Variables of type string can be assigned with the = operator
  - Example:

```
string s1;
string s2;
string s3;
…
s3 = s2;
```

- Quoted strings are type cast to type string
  - Example:

```
string s1 = "Hello Mom!";
```

Using + With strings

- Variables of type string can be concatenated with the + operator
  - Example:

```
string s1;
string s2;
string s3;
 …
s3 = s1 + s2;
```

  - If s3 is not large enough to contain s1 + s2, more space is allocated

Using + With strings

```
char cString1[80] = "I wonder if you're really";
char cString2[] = " going to read this??";

string string1 = "There might be ";
string string2 = "a real chance that you will!";
// the book uses strcat
// Only with VS 2005 - strcat_s() because strcat has been deprecated. strcat_s() is a
    more secure version of strcat(), but you can use strcat too!
cout << "Output the cstrings\n";
cout << strcat_s(cString1, cString2) << endl;

// the + operator doesn't add the characters but
// concatenates the strings together creating one string
cout << "Output the strings\n";
cout << string1 + string2 << endl;
```

Windows Only

string Constructors

- The default string constructor initializes the string to the empty string
- Another string constructor takes a C-string argument
  - Example:
  -

```
string phrase;          // empty string
string noun("ants");    // a string version
                        //  of "ants"
```

Mixing strings and C-strings

- It is natural to work with strings in the following manner

```
string phrase = "I love" + adjective + " "
                          + noun + "!";
```

  - It is not so easy for C++! It must either convert the null-terminated C-strings, such as "I love", to strings, or it must use an overloaded + operator that works with strings and C-strings

I/O With strings

- The insertion operator << is used to output objects of type string
  - Example:
    ```
    string s = "Hello Mom!";
    cout << s;
    ```
- The extraction operator >> can be used to input data for objects of type string
  - Example:
    ```
    string s1;
    cin >> s1;
    ```
    - >> skips whitespace and stops on encountering more whitespace

I/O with string

```
string word;
while(inFile >> word) {
        // statements here
        // work happening here
}
```

getline and Type string

- A *getline* function exists to read entire lines into a string variable
  - This version of *getline* is not a member of the istream class, it is a **non-member** function of the string class
  - Syntax for using this *getline* is different than that used with cin: cin.getline(…)
- Syntax for using *getline* with string objects:
  getline(istream_Object, string_Object);
- From #include <string>

getline Example

- This code demonstrates the use of getline with string objects
  -
```
string line;
cout "Enter a line of input:\n";
getline(cin, line);
cout << line << "END OF OUTPUT\n";
```

Output could be:

```
Enter some input:
Do be do to you!
Do be do to you!END OF OUTPUT
```

Character Input With strings

- The extraction operator cannot be used to read a blank character
- To read one character at a time remember to use cin.get
  - cin.get reads values of type char, not type string

Another Version of getline

- The versions of *getline* we have seen, stop reading at the end of line marker '\n'
- *getline* can stop reading at a character specified in the argument list
  - This code stops reading when a '?' is read

```
string line;
cout <<"Enter some input: \n";
getline(cin, line, '?');
```

getline Returns a Reference

- getline returns a reference to its first argument

- This code will read in a line of text into s1 and a string of non-whitespace characters into s2:

```
string s1;
string s2;
```

$$\underbrace{\texttt{getline(cin, s1)}}_{\text{returns}\ \downarrow}\ \texttt{>> s2;}$$

```
cin >> s2;
```

-

getline Declarations

- These are the declarations of the versions of getline for string objects we have seen
  - Syntax:

```
#include <string>
istream& getline( istream& is, string& s, char delimiter = '\n' );
```

Mixing cin >> and getline

- Recall cin >> n skips whitespace to find what it is to read then stops reading when whitespace is found
- cin >> leaves the '\n' character in the input stream
  - Example:

```
int n;
string line;
cin >> n;
getline(cin, line);
```

leaves the '\n' which immediately ends getline's reading…line is set equal to the empty string

ignore

- ignore is a member of the istream class
- ignore can be used to read and discard all the characters, including '\n' that remain in a line
  - Ignore takes two arguments
    - First, the maximum number of characters to discard
    - Second, the character that stops reading and discarding
  - Example:

```
cin.ignore(1000, '\n');
```

reads up to 1000 characters or to '\n'

String Processing

- The string class allows the same operations we used with C-strings…and more
  - Characters in a string object can be accessed as if they are in an array
    - fileName[i] provides access to a single character as in an array
    - Index values are not checked for validity!

Member Function length

- The string class member function length returns the number of characters in the string object:
  - Example:

```
int n = fileName.length( );
```

- This provides much more power than cstrings in that we don’t have to keep size, string object do that for us

Member Function at

- at is an alternative to using [ ]'s to access characters in a string.
  - at checks for valid index values
  - Example:

```
string str("Mary");
cout << str[1] << endl;
cout << str.at(1) << endl;
str[2] = 'X';
str.at(2) = 'X';
```

Equivalent $\begin{array}{l}\nearrow\\\searrow\end{array}$

Equivalent $\begin{array}{l}\nearrow\\\searrow\end{array}$

Member Functions of the Standard Class string

Example	**Remarks**
Constructors	
`string str;`	Default constructor creates empty `string` object `str`.
`string str("sample");`	Creates a `string` object with data `"sample"`.
`string str(a_string);`	Creates a `string` object `str` that is a copy of `a_string`; `a_string` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`. Does not check for illegal index.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`. Same as `str[i]`, but this version checks for illegal index.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/modifiers	
`str1 = str2;`	Initializes `str1` to `str2`'s data,
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`.
`str.empty( )`	Returns *`true`* if str is an empty string; *`false`* otherwise.
`str1 + str2`	Returns a string that has `str2`'s data concatenated to the end of `str1`'s data.
`str.insert(pos, str2);`	Inserts `str2` into `str` beginning at position pos.
`str.remove(pos, length);`	Removes substring of size `length`, starting at position pos.
Comparison	
`str1 == str2   str1 != str2`	Compare for equality or inequality; returns a Boolean value.

`str1 < str2    str1 > str2`
`str1 <= str2   str1 >= str2`
	Four comparisons. All are lexicographical comparisons.

Finds	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position pos.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position pos.
`str.find_first_not_of (str1, pos)`	Returns the index of the first instance in `str` of any character not in `str1`, starting the search at position pos.

Comparison of strings

- Comparison operators work with string objects
  - Objects are compared using lexicographic order (Alphabetical ordering using the order of symbols in the ASCII character set.)
  - = = returns true if two string objects contain the same characters in the same order
    - Remember strcmp for C-strings?
  - <, >, <=, >= can be used to compare string objects

stringFun.cpp

string Objects to C-strings

- Recall the automatic conversion from C-string to string:

```
char cString[] = "C-string";
stringVar = cString;
```

- strings are not converted to C-strings
- Both of these statements are **illegal**:
  - `cString = stringVar;`
  - `strcpy(cString, stringVar);`

Converting strings to C-strings

- The string class member function c_str returns the C-string version of a string object
  - Example:
    ```
    strcpy(cString, stringVar.c_str( ) );
    ```
- This line is still **illegal**
  ```
  cString = stringVar.c_str( ) ;
  ```
  - Recall that operator = does not work with C-strings

Section 8.2 Conclusion

- Can you
  - Show how a string object can be used like a C-string?
  - Write code to read an entire line into a string object?
  - Use the string function at to access individual characters in a string object?
  - Write code to convert a string to a C-string?

8.3

Vectors

Vectors

- Vectors are like arrays that can change size as your program runs
- Vectors, like arrays, have a base type
- To declare an empty vector with base type int:

```
vector<int> v;
```

  - <int> identifies vector as a template class
  - You can use any base type in a template class:

```
vector<string> v;
```

Vector from STL

- Standard Template Library - STL
  - Provides containers, iterators and algorithms that you don’t have to program.
  - They have been given to you for FREE! ☺
  - Java has what’s known as the collections classes which contains similar containers, iterators and algorithms for Java

Accessing vector Elements

- Vectors elements are indexed starting with 0
  - [ ]'s are used to read or change the value of an item:

```
          v[i] = 42;

         cout << v[i];
```

  - [ ]'s cannot be used to initialize a vector element

Initializing vector Elements

- Elements are added to a vector using the member function ***push_back***
  - ***push_back*** adds an element in the next available position
  - Example:

```
vector<double> sample;
sample.push_back(0.0);
sample.push_back(1.1);
sample.push_back(2.2);
```

The size Of A vector

- The member function size returns the number of elements in a vector
  - Example: To print each element of a vector given the previous vector initialization:

```
for (int i= 0; i < teams.size(); i++)
          cout << teams[i] << endl;
```

vectors.cpp

The Type unsigned int

- The vector class member function size returns an unsigned int
  - Unsigned int's are nonnegative integers
  - Some compilers will give a warning if the previous for-loop is not changed to:

```
for (unsigned int i= 0; i < teams.size( ); i++)
    cout << teams[i] << endl;
```

Alternate vector Initialization

- A vector constructor exists that takes an integer argument and initializes that number of elements
  - Example: vector<int> v(10);

    initializes the first 10 elements to 0
    v.size( ) would return 10
    - [ ]'s can now be used to assign elements 0 through 9
    - push_back is used to assign elements greater than 9

Vector Initialization With Classes

- The vector constructor with an integer argument
  - Initializes elements of number types to zero
  - Initializes elements of class types using the default constructor for the class

The vector Library

- To use the vector class
  - Include the vector library

```
#include <vector>
```

  - Vector names are placed in the standard namespace so the usual using directive is needed:

```
using namespace std;
```

vectors.cpp

vector Issues

- Attempting to use [ ] to set a value beyond the size of a vector may not generate an error
  - The program will probably misbehave
- The assignment operator with vectors does an element by element copy of the right hand vector
  - For class types, the assignment operator must make independent copies

vector Efficiency

- A vector's capacity is the number of elements allocated in memory
  - Accessible using the capacity( ) member function
- Size is the number of elements initialized
- When a vector runs out of space, the capacity is automatically increased
  - A common scheme is to double the size of a vector
    - More efficient than allocating smaller chunks of memory

Controlling vector Capacity

- When efficiency is an issue
  - Member function reserve can increase the capacity of a vector
    - Example:

```
v.reserve(32); // at least 32 elements
v.reserve(v.size( ) + 10);  // at least 10 more
```

  - resize can be used to shrink a vector
    - Example:

```
v.resize(24);  //elements beyond 24  are lost
```

Section 8.3 Conclusion

- Can you
  - Declare and initialize a vector of 10 doubles?
  - Write code to increase the size of a vector in at least two different ways?
  - Describe the difference between a vector's size and its capacity?