Chapter 11
Strings and Vectors

Overview
• An Array Type for Strings (11.1)
• The Standard string class (11.2)
• Vectors (11.3)

Copyright © 2005 Pearson Addison-Wesley. All rights reserved.

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[11];

Copyright © 2005 Pearson Addison-Wesley. All rights reserved.

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:

Copyright © 2005 Pearson Addison-Wesley. All rights reserved.

C-string Declaration
• To declare a C-string variable, use the syntax:
  char Array_name[ Maximum_C_String_Size + 1];
  • + 1 reserves the additional character needed by ’\0’

Copyright © 2005 Pearson Addison-Wesley. All rights reserved.

Initializing a C-string
• To initialize a C-string during declaration:
  char my_message[20] = "Hi there.\0";
  • The null character ’\0’ is added for you
• Another alternative:
  char short_string[ ] = "abc";
  but not this:
  char short_string[ ] = {’a’, ’b’, ’c’};

Copyright © 2005 Pearson Addison-Wesley. All rights reserved.
C-string error

- This attempt to initialize a C-string does not cause the \0 to be inserted in the array
  - char short_string[ ] = {'a', 'b', 'c'};

Don't Change '\0'

- Do not 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 index = 0;
      while (our_string[index] != '\0')
      {
        our_string[index] = 'X';
        index++;
      }
      ```
      - 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 (our_string[index] != '\0' && index < SIZE)
    {
      our_string[index] = 'X';
      index++;
    }
    ```

Assignment With C-strings

- This statement is illegal:
  ```
  a_string = "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 a_string[ 11];
    strcpy (a_string, "Hello");
    ```
    - Places "Hello" followed by the null character in a_string

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
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 another_string[10];
    strncpy(another_string, a_string_variable, 9);
    This code copies up to 9 characters into another_string, leaving one space for ‘0’

More C-string Functions

- The cstring library includes other functions
  - strlen returns the number of characters in a string
    int x = strlen( a_string);
  - 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 string_var[20] = "The rain";
      strcat(string_var, "in Spain");
      Now string_var contains "The rain in Spain"

== 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(c_string1, c_string2))
      cout << "Strings are not the same.";
    else
      cout << "String are the same.";

The strcat Function

- strcat is a safer version of strcat
  - A third parameter specifies a limit for the number of characters to concatenate
  - Example:
    char string_var[20] = "The rain";
    strcat(string_var, "in Spain", 11);

Display 11.1 (1)
Display 11.1 (2)

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

C-strings as Arguments and Parameters

- C-string variables are arrays
- C-string arguments and parameters are used just like 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 news[ ] = "C-strings";
cout << news << " Wow." << endl;

C-string Input

- The extraction operator `>>` can fill a C-string
  - Whitespace ends reading of data
  - Example: `char a[80], b[80];
cout << "Enter input: " << endl;
cin >> a >> b;
cout << a << b << "End of Output";
  - could produce:
    Enter input: 
    Do be do to you!
    Do be do to you!End 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
  - `char a[80];
cout << "Enter input: n";
cin.getline(a, 80);
cout << a << End Of Output n";
  - and could produce:
    Enter some input:
    Do be do to you!
    Do be do to you!End of Output

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
    - `in_stream >> c_string;
in_stream.getline(c_string, 80);`
  - Replace `cout` with the name of an output-file stream
    - `out_stream << c_string;`
getline syntax

- Syntax for using getline is
  
  `cin.getline(String_Var, Max_Characters + 1);`

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

C-string to long

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

C-String to Numbers

- "1234" is a string of characters
- 1234 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-string to double

- C-strings can be converted to type double using the predefined function `atof`
  
  `atof` returns a value of type double
  
  - `atof("$9.99")` returns 0.0 because the $ is not a digit

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("1234")` returns the integer 1234
  - `atoi("#1234")` returns 0 because # 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>`
Numeric Input

• We now know how to convert C-strings to numbers
• How do we read the input?
  • Function read_and_clean, in Display 11.2…
    • Reads a line of input
    • Discards all characters other than the digits 0 through 9
    • Uses atoi to convert the “cleaned-up” C-string to int

Display 11.2 (1)
Display 11.2 (2)

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;

Display 11.3 (1)
Display 11.3 (2)

Confirming Input

• Function get_int, from Display 11.3…
  • Uses read_and_clean to read the user’s input
  • Allows the user to reenter the input until the user is satisfied with the number computed from the input string

Display 11.3 (1)
Display 11.3 (2)

Section 11.1 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?

Assignment of Strings

• Variables of type string can be assigned with the = operator
  • Example: string s1, s2, 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, s2, s3;
    ... s3 = s1 + s2;
  • If s3 is not large enough to contain s1 + s2, more space is allocated

Display 11.2 (1)
Display 11.2 (2)
**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 Class string**

- 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.

**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.
  - 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);
    ```

**Getline Example**

- This code demonstrates the use of getline with string objects.
  ```
  string line;
  cout "Enter a line of input:
  getline(cin, line);
  cout << line << "END OF OUTPUT"
  ```
- 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
  ```
- The use of getline, and cin.get for string input are demonstrated in:
  ```
  Display 11.5 (1)
  ```
  ```
  Display 11.5 (2)
  ```
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

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

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:  
    ```cpp
    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

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:
  ```cpp
  string s1, s2;
generate(cin, s1) >> s2;
  ```

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:  
    ```cpp
    cin.ignore(1000, '\n');
    ```
    reads up to 1000 characters or to "\n"

getline Declarations

- These are the declarations of the versions of getline for string objects we have seen
  - `istream& getline(istream& ins, string& str_var, char delimiter);`
  - `istream& getline(istream& ins, string& str_var);`

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
    - `last_name[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 = string_var.length();
    ```

Program Example: Palindrome Testing

- A palindrome is a string that reads the same from front to back as it does from back to front
  - This program ignores spaces and punctuation
  - Upper and lowercase versions of letters are considered the same letter
  - Examples:
    ```
    Able was I'ere I saw Elba.
    Madam, I'm Adam.
    A man, a plan, a canal, Panama.
    Racecar
    ```

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[6] << endl;
    cout << str.at(6) << endl;
    str[2] = 'X';
    str.at(2) = 'X';
    ```

Other string class functions are found in Display 11.7

Palindrome Testing: remove_punct

- remove_punct removes punctuation from a string
  - remove_punct compares each character in the string to the characters in a string containing all the punctuation characters and the space character.
  - If a match is not found, the character is added to the string no_punct
  - no_punct, the original string less any punctuation or spaces, is returned

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

Palindrome Testing: substr

- The substr member function is used to locate a substring within a string
  - remove_punct uses substr to extract a single character at a time from the source string. The character is stored in a_char.
  - remove_punct then uses function find to see if the character in a_char is in the string of punctuation characters
Palindrome Testing: The Program

- The entire palindrome testing program is found in:
  - Display 11.8 (1)
  - Display 11.8 (2)
  - Display 11.8 (3)
  - Display 11.8 (4)

String Objects to C-Strings

- Recall the automatic conversion from C-string to string:
  - char a_c_string[] = "C-string";
  - string_variable = a_c_string;
- Strings are not converted to C-strings
- Both of these statements are illegal:
  - a_c_string = string_variable;
  - strcpy(a_c_string, string_variable);

Converting strings to C-Strings

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

Section 11.2 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?

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;

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);`

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
  - `[]` is now be used to assign elements 0 through 9
  - `push_back` is used to assign elements greater than 9

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 < sample.size(); i++)
    cout << sample[i] << endl;`

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 < sample.size(); i++)
      cout << sample[i] << endl;

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;
    ```

Display 11.9
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 11.3 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?

Display 11.1

(1/2)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Caveats</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy()</code></td>
<td>Copies the C-string value from <code>Src_Strings</code> into the C-string variable <code>Target_Strings</code>.</td>
<td>Does not check to make sure that <code>Target_Strings</code> is large enough to hold the value of <code>Src_Strings</code>.</td>
</tr>
<tr>
<td><code>strncpy()</code></td>
<td>The same as the two argument <code>strcpy()</code> except that it does not check if the destination string can be copied.</td>
<td>If <code>limit</code> is a non-negative value, this function is faster than the two argument version of <code>strcpy()</code>. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td><code>strcpy()</code></td>
<td>Copies the C-string value from <code>Src_Strings</code> into the C-string variable <code>Target_Strings</code>.</td>
<td>Does not check to see that <code>Target_Strings</code> is large enough to hold the result of the concatenation.</td>
</tr>
</tbody>
</table>
Display 11.5
(1/2)

Program Using the Class string (part 1 of 2):

```java
int main()
{
    string first_name, last_name, record_name;
    int i = 0; // index into string names
    while (!i到达后退) { // until we reach the end of the strings
        cout << "Enter name: ", >> first_name, >> last_name;
        if (last_name[0] = = '"')
            cout << "Enter record name:");
        i = i + 1;
    }
    return 0;
}
```

Sample Dialogue:

Enter your first and last name:
- A. B. Surname
- Your name is on our records for A. B. Surname
- Please enter a better (last) name.
- Our records are not updated.
- Our records do not exist.

Display 11.7

Display 11.8
(1/4)

Program Testing Program (part 1 of 4):

```java
main()
{
    string name = "John Doe";
    string address = "123 Main St."
    string phone = "555-1234"
    cout << "Hello, " << name << "!
    cout << "Your address is: " << address << ".
    cout << "Your phone number is: " << phone << ".
    return 0;
}
```

Sample Dialogue:

- Hello, John Doe!
- Your address is: 123 Main St.
- Your phone number is: 555-1234.

Display 11.8
(2/4)
Display 11.8
(3/4)

Palindrome testing program (part 2 of 4)

```java
public class Palindrome {
    public static boolean isPalindrome(String s) {
        int length = s.length();
        int start = 0;
        int end = length - 1;
        while (start < end) {
            if (s.charAt(start) != s.charAt(end)) {
                return false;
            }
            start++;
            end--;
        }
        return true;
    }
}
```

Display 11.8
(4/4)

Palindrome testing program (part 4 of 4)

Sample Dialogues

Enter a candidate for palindrome test followed by pressing return.
Haddon, I'm Aman...
Haddon, I'm Aman...

Enter a candidate for palindrome test followed by pressing return.
Radar
Radar

Enter a candidate for palindrome test followed by pressing return.
Am I a palindrome?
Am I a palindrome?

Display 11.9

Using Vectors

```java
public class Vector {
    int[] data;
    int count;

    public Vector() {
        data = new int[0];
        count = 0;
    }

    public void add(int value) {
        data = Arrays.copyOf(data, count + 1);
        data[count] = value;
        count++;
    }

    public void remove(int index) {
        System.arraycopy(data, index + 1, data, index, count - index - 1);
        count--;
    }
}
```