Chapter 8

Strings and Vectors
Overview

8.1 An Array Type for Strings
8.2 The Standard string Class
8.3 Vectors
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[11];
  ```
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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>i</td>
<td>M</td>
<td>o</td>
<td>m</td>
<td>!</td>
<td>$\backslash 0$</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
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'
Initializing a C-string

- To initialize a C-string during declaration:
  ```
  char my_message[20] = "Hi there.";
  ```
  The null character '\0' is added for you

- Another alternative:
  ```
  char short_string[ ] = "abc";
  ```
  but not this:
  ```
  char short_string[ ] = {'a', 'b', 'c'};
  ```
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:
    ```c
    int index = 0;
    while (our_string[index] != '\0')
    {
        our_string[index] = 'X';
        index++;
    }
    ```
  - This code depends on finding the null character!
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:

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

Example: 

```c
#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'
== 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."
```
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
More C-string Functions

- The cstring library includes other functions
  - `strlen` returns the number of characters in a string
    ```c
    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:
      ```c
      char string_var[20] = "The rain";
      strcat(string_var, "in Spain");
      ```

      Now `string_var` contains "The rain in 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:
  ```c
  char string_var[20] = "The rain";
  strncat(string_var, "in Spain", 11);
  ```

`Display 8.1 (1)`

`Display 8.1 (2)`
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;
    ```
The extraction operator `>>` can fill a C-string
- Whitespace ends reading of data
- Example:  
  ```cpp
  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! 
  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
  ```cpp
  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
    ```cpp
    in_stream >> c_string;
    in_stream.getline(c_string, 80);
    ```
  - Replace `cout` with the name of an output-file stream
    ```cpp
    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 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-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("#123") returns 0 because # is not a digit
C-string to long

- Larger integers can be converted using the predefined function `atol`
  - `atol` returns a value of type `long`
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
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 8.2...
    - 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 8.2 (1)
Display 8.2 (2)
Confirming Input

- Function get_int, from Display 8.3...
  - 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
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 \texttt{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;
    ```
The Standard string Class

![Diagram of string class with properties: p, 5, length, 10, capacity, and a name containing the string "Karen".]
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
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.

Display 8.4
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

  ```cpp
  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`
- The use of `getline`, and `cin.get` for string input are demonstrated in
  - Display 8.5 (1)
  - Display 8.5 (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";
    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:

```cpp
string s1, s2;
ggetline(cin, s1) >> s2;
```

`cin >> s2;` returns
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);`
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
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
    - last_name[i] provides access to a single character as in an array
    - Index values are not checked for validity!

Display 8.6
The string class member function length returns the number of characters in the string object:

Example:

```java
int n = string_var.length();
```
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 8.7.
string class to numbers

C++11 has new functions to convert a string class object to a number

```cpp
int i;
double d;
string s;
i = stoi("35"); // Converts the string "35" to an integer 35
d = stod("2.5"); // Converts the string "2.5" to the double 2.5
```

C++11 has new functions to convert a string class object to a number

```cpp
string s = to_string(1.2*2); // "2.4" stored in s
```
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
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
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.
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.
Palindromic Testing: The Program

- The entire palindrome testing program is found in

  Display 8.8 (1)
  Display 8.8 (2)
  Display 8.8 (3)
  Display 8.8 (4)
Palindrome Testing Program (part 1 of 4)

//Test for palindrome property.
#include <iostream>
#include <string>
#include <ctype>
using namespace std;

void swap(char& v1, char& v2);
//Interchanges the values of v1 and v2.

string reverse(const string& s);
//Returns a copy of s but with characters in reverse order.

string remove_punct(const string& s, const string& punct);
//Returns a copy of s with any occurrences of characters
//in the string punct removed.

string make_lower(const string& s);
//Returns a copy of s that has all uppercase
//characters changed to lowercase, other characters unchanged.

bool is_pal(const string& s);
//Returns true if s is a palindrome, false otherwise.

int main()
{
    string str;
    cout << "Enter a candidate for palindrome test\n"
    << "followed by pressing Return.\n"
    << "\n";
    getline(cin, str);

    if (is_pal(str))
        cout << "\"" << str << "\" is a palindrome.\n";
    else
        cout << "\"" << str << "\" is not a palindrome.\n";
    cout << endl;
    return 0;
}
Palindrome Testing Program (part 2 of 4)

```c
void swap(char& v1, char& v2)
{
    char temp = v1;
    v1 = v2;
    v2 = temp;
}

string reverse(const string& s)
{
    int start = 0;
    int end = s.length();
    string temp(s);

    while (start < end)
    {
        end--;
        swap(temp[start], temp[end]);
        start++;
    }

    return temp;
}

//Uses <cctype> and <string>
string make_lower(const string& s)
{
    string temp(s);
    for (int i = 0; i < s.length(); i++)
        temp[i] = tolower(s[i]);

    return temp;
}
```
Palindrome Testing Program (part 3 of 4)

```c++
string remove_punct(const string& s, const string& punct) {
    string no_punct; // initialized to empty string
    int s_length = s.length();
    int punct_length = punct.length();

    for (int i = 0; i < s_length; i++)
    {
        string a_char = s.substr(i,1); // A one-character string
        int location = punct.find(a_char, 0);
        // Find location of successive characters
        // of src in punct.

        if (location < 0 || location >= punct_length)
            no_punct = no_punct + a_char; // a_char not in punct, so keep it
    }

    return no_punct;
}

// uses functions make_lower, remove_punct.
bool is_pal(const string& s) {
    string punct("\",\.;?!"\ "); // includes a blank
    string str(s);
    str = make_lower(str);
    string lower_str = remove_punct(str, punct);

    return (lower_str == reverse(lower_str));
}
```
Palindrome Testing Program (part 4 of 4)

Sample Dialogues

Enter a candidate for palindrome test followed by pressing Return.
Madam, I'm Adam.
"Madam, I'm Adam." is a palindrome.

Enter a candidate for palindrome test followed by pressing Return.
Radar
"Radar" is a palindrome.

Enter a candidate for palindrome test followed by pressing Return.
Am I a palindrome?
"Am I a palindrome?" is not a palindrome.
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 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;
    ```
Accessing vector Elements

- Vectors elements are indexed starting with 0
  - [ ]'s are used to read or change the value of an item:
    
    ```cpp
    v[i] = 42;
    ```
    
    ```cpp
    cout << v[i];
    ```
  - [ ]'s cannot be used to initialize a vector element
Vector in memory

std::vector<
class instance

... internal service object
another internal object
ptr to the first element
ptr to the past-the-end element
...

Memory Address Space

data array in memory

1st item  2nd item  3rd item  ...  nth item  past-the-end item

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:
  
  ```cpp
  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:

```cpp
for (unsigned int i = 0; i < sample.size(); i++)
    cout << sample[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
    
    ```cpp
    #include <vector>
    ```
  
  - Vector names are placed in the standard namespace so the usual using directive is needed:
    
    ```cpp
    using namespace std;
    ```

Display 8.9
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 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?
Chapter 8 -- End
### Some Predefined C-String Functions in `<cstring>` *(part 1 of 2)*

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Cautions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(Target_String_Var, Src_String)</code></td>
<td>Copies the C-string value <code>Src_String</code> into the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to make sure <code>Target_String_Var</code> is large enough to hold the value <code>Src_String</code>.</td>
</tr>
<tr>
<td><code>strncpy(Target_String_Var, Src_String, Limit)</code></td>
<td>The same as the two-argument <code>strcpy</code> except that at most <code>Limit</code> characters are copied.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcpy</code>. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td><code>strcat(Target_String_Var, Src_String)</code></td>
<td>Concatenates the C-string value <code>Src_String</code> onto the end of the C string in the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to see that <code>Target_String_Var</code> is large enough to hold the result of the concatenation.</td>
</tr>
</tbody>
</table>
### Some Predefined C-String Functions in `<cstring>` (part 2 of 2)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strncat()</code></td>
<td>The same as the two-argument <code>strcat</code> except that at most <code>Limit</code> characters are appended.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcat</code>. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td><code>strlen()</code></td>
<td>Returns an integer equal to the length of <code>Src_String</code>. (The null character, '\0', is not counted in the length.)</td>
<td></td>
</tr>
<tr>
<td><code>strcmp()</code></td>
<td>Returns 0 if <code>String_1</code> and <code>String_2</code> are the same. Returns a value &lt; 0 if <code>String_1</code> is less than <code>String_2</code>. Returns a value &gt; 0 if <code>String_1</code> is greater than <code>String_2</code> (that is, returns a nonzero value if <code>String_1</code> and <code>String_2</code> are different). The order is lexicographic.</td>
<td>If <code>String_1</code> equals <code>String_2</code>, this function returns 0, which converts to <code>false</code>. Note that this is the reverse of what you might expect it to return when the strings are equal.</td>
</tr>
<tr>
<td><code>strncpy()</code></td>
<td>The same as the two-argument <code>strcat</code> except that at most <code>Limit</code> characters are compared.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcmp</code>. Not implemented in all versions of C++.</td>
</tr>
</tbody>
</table>
C Strings to Integers (part 1 of 2)

// Demonstrates the function read_and_clean.
#include <iostream>
#include <cstdlib>
#include <ctype>

void read_and_clean(int& n);
// Reads a line of input. Discards all symbols except the digits. Converts
// the C string to an integer and sets n equal to the value of this integer.

void new_line();
// Discards all the input remaining on the current input line.
// Also discards the '\n' at the end of the line.

int main()
{
    using namespace std;
    int n;
    char ans;
    do
    {
        cout << "Enter an integer and press Return: ";
        read_and_clean(n);
        cout << "That string converts to the integer " << n << endl;
        cout << "Again? (yes/no): ";
        cin >> ans;
        new_line();
    } while ( (ans != 'n') && (ans != 'N') );
    return 0;
}
C-Strings to Integers (part 2 of 2)
//Uses iostream, stdlib, and ctype:
void read_and_clean(int& n)
{
    using namespace std;
    const int ARRAY_SIZE = 6;
    char digit_string[ARRAY_SIZE];
    char next;
    cin.get(next);
    int index = 0;
    while (next != '\n')
    {
        if (isdigit(next)) && (index < ARRAY_SIZE - 1)
        {
            digit_string[index] = next;
            index++;
        }
        cin.get(next);
    }
    digit_string[index] = '\0';
    n = atoi(digit_string);
}
//Uses iostream:
void new_line()
{
    using namespace std;
    //The rest of the definition of new_line is given in Display 5.7.

Sample Dialogue

Enter an integer and press Return: $ 100
That string converts to the integer 100
Again? (yes/no): yes
Enter an integer and press Return: 100
That string converts to the integer 100
Again? (yes/no): yes
Enter an integer and press Return: 99%
That string converts to the integer 99
Again? (yes/no): yes
Enter an integer and press Return: 23% *12
That string converts to the integer 23512
Again? (yes/no): no
DISPLAY 8.3 Robust Input Function (part 1 of 3)

1   //Demonstration program for improved version of get_int.
2   #include <iostream>
3   #include <cstdlib>
4   #include <cctype>
5   void read_and_clean(int& n);
6   //Reads a line of input. Discards all symbols except the digits. Converts
7   //the C string to an integer and sets n equal to the value of this integer.

(continued)
void new_line( );
//Discards all the input remaining on the current input line.
//Also discards the '
' at the end of the line.
void get_int(int& input_number);
//Gives input_number a value that the user approves of.
int main( )
{
  using namespace std;
  int input_number;
  get_int(input_number);
  cout << "Final value read in = " << input_number << endl;
  return 0;
}

//Uses istream and read_and_clean:
void get_int(int& input_number)
{
  using namespace std;
  char ans;
  do
  {
    cout << "Enter input number: ";
    read_and_clean(input_number);
    cout << "You entered " << input_number
    << " Is that correct? (yes/no): ";
    cin >> ans;
    new_line();
  } while ((ans != 'y') && (ans != 'Y'));
}

//Uses istream, cstdlib, and cctype:
void read_and_clean(int& n)

<The rest of the definition of read_and_clean is given in Display 8.2.>

//Uses istream:
void new_line( )

<The rest of the definition of new_line is given in Display 8.2.>

Sample Dialogue

Enter input number: $57
You entered 57 Is that correct? (yes/no): no

(continued)
**DISPLAY 8.3  Robust Input Function (part 3 of 3)**

<table>
<thead>
<tr>
<th>Enter input number: $77*5xa</th>
<th>You entered 775 Is that correct? (yes/no): no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter input number: 77</td>
<td>You entered 77 Is that correct? (yes/no): no</td>
</tr>
<tr>
<td>Enter input number: $75</td>
<td>You entered 75 Is that correct? (yes/no): yes</td>
</tr>
<tr>
<td>Final value read in = 75</td>
<td></td>
</tr>
</tbody>
</table>
**Program Using the Class string**

```cpp
// Demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string phrase;
    string adjective("fried"), noun("ants");
    string wish = "Bon appetite!";

    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl
         << wish << endl;

    return 0;
}
```

**Sample Dialogue**

I love fried ants!
Bon appetite!
Program Using the Class string (part 1 of 2)

```cpp
//Demonstrates getline and cin.get.
#include <iostream>
#include <string>

void new_line()
{
    using namespace std;

    string first_name, last_name, record_name;
    string motto = "Your records are our records."
;
    cout << "Enter your first and last name:\n"
; cin >> first_name >> last_name
; new_line()
;
    record_name = last_name + ", " + first_name;
    cout << "Your name in our records is: ";
    cout << record_name << endl;

    cout << "Our motto is\n" << motto << endl;
    cout << "Please suggest a better (one-line) motto:\n";
    getline(cin, motto);
    cout << "Our new motto will be:\n"
; cout << motto << endl;

    return 0;
}
Program Using the Class string (part 2 of 2)

```cpp
//Uses iostream:
void new_line( )
{
    using namespace std;

    char next_char;
    do
    {
        cin.get(next_char);
    } while (next_char != '\n');
}
```

Sample Dialogue

Enter your first and last name:
   B'Elanna Torres
Your name in our records is: Torres, B'Elanna
Our motto is
Your records are our records.
Please suggest a better (one-line) motto:
   Our records go where no records dared to go before.
Our new motto will be:
   Our records go where no records dared to go before.
A string Object Can Behave Like an Array

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

int main()
{
    string first_name, last_name;

    cout << "Enter your first and last name:\n";
    cin >> first_name >> last_name;

    cout << "Your last name is spelled:\n";
    int i;
    for (i = 0; i < last_name.length(); i++)
    {
        cout << last_name[i] << " ";
        last_name[i] = '-';
    }
    cout << endl;
    for (i = 0; i < last_name.length(); i++)
    {
        cout << last_name[i] << " "; //Places a "-" under each letter.
    }
    cout << endl;

    cout << "Good day " << first_name << endl;
    return 0;
}
```

Sample Dialogue

Enter your first and last name: John Crichton
Your last name is spelled: C r i c h t o n
-----------------------
Good day John
## Member Functions of the Standard Class `string`

### 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`. 
Using a Vector

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

int main()
{
    vector<int> v;
    cout << "Enter a list of positive numbers.  \n" << "Place a negative number at the end. \n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. \n";
        cout << "v.size() = " << v.size() << endl;
        cin >> next;
    }

    cout << "You entered: \n";
    for (unsigned int i = 0; i < v.size(); i++)
        cout << v[i] << " \n";
    cout << endl;
    return 0;
}
```

Sample Dialogue

Enter a list of positive numbers. Place a negative number at the end.
2 4 6 8 -1
2 added. v.size() = 1
4 added. v.size() = 2
6 added. v.size() = 3
8 added. v.size() = 4
You entered:
2 4 6 8