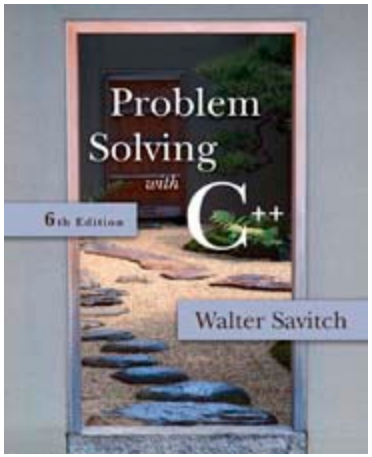
APS105: Lectures 18 &19

Wael Aboelsaadat

wael@cs.toronto.edu

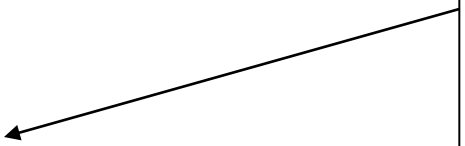
http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



How to get user arguments from command line?

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
        for (int count = 0; count < argc; count ++)
       {
          cout << "commmand line value "
                << count
                 << " : " << argv[count] << endl;
       }
      return 0;
}
```



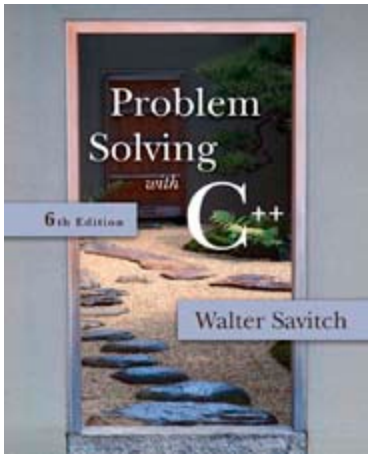
The Operating System passes the number of Arguments in argc while the Arguments are passed in argv

argv is an array, where by each element in that array is itself a char array

Compile this program and try
./a.out 222 111 333
and compare it to
./a.out 222 1 1

Chapter 8

Strings



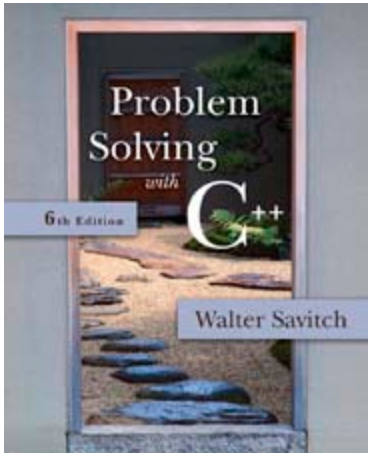
Overview

8.1 An Array Type for Strings

8.2 The Standard `string Class`

8.1

An Array Type for Strings



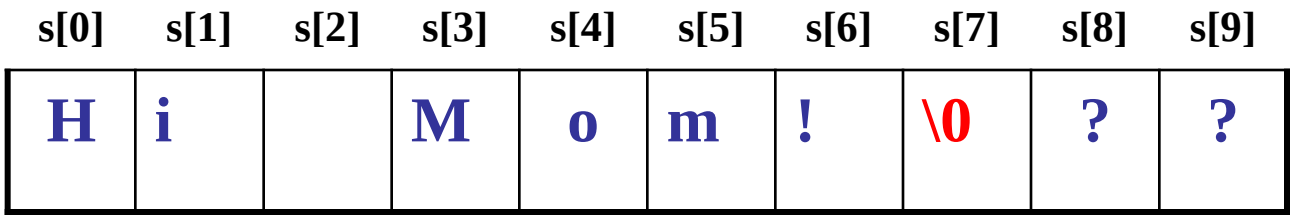
An Array Type for Strings

- 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:



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 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 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'

== Alternative for C-strings

- The = = operator does not work as expected with C-strings
  - The predefined function strcmp is used to compareC-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

```
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:

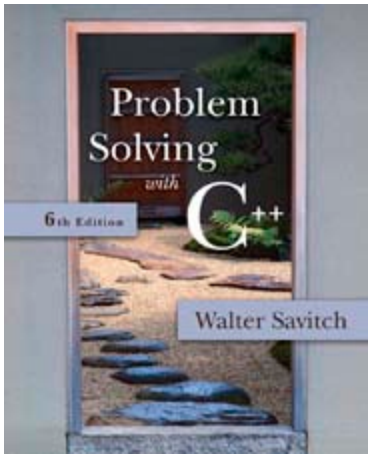
```
char string_var[20] = "The rain";
strcat(string_var, "in Spain");
```

Now string_var contains "The 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 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;
```

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!
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

```
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 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
  - 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>
```

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

Display 8.3 (1)

Display 8.3 (2)

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, 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
  - ```
    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

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

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:

```
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

Member Function length

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

```
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';
```

Equivalent

Equivalent

Other string class functions are found in **Display 8.7**

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

Display 8.1 (1/2)

Some Predefined C-String Functions in `<cstring>` (*part 1 of 2*)

Function	Description	Cautions
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strncpy(`*Target_String_Var*`,` *Src_String*`,` *Limit*`)`	The same as the two-argument `strcpy` except that at most *Limit* characters are copied.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the C-string value *Src_String* onto the end of the C string in the C-string variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.

Display 8.1 (2/2)

Some Predefined C-String Functions in `<cstring>` (*part 2 of 2*)

`strncat(`*Target_String_Var*`,` *Src_String*`,` *Limit*`)`	The same as the two-argument `strcat` except that at most *Limit* characters are appended.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcat`. Not implemented in all versions of C++.
`strlen(`*Src_String*`)`	Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.)	
`strcmp(`*String_1*`,` *String_2*`)`	Returns 0 if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (that is, returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic.	If *String_1* equals *String_2*, this function returns 0, which converts to `false`. Note that this is the reverse of what you might expect it to return when the strings are equal.
`strncmp(`*String_1*`,` *String_2*`,` *Limit*`)`	The same as the two-argument `strcat` except that at most *Limit* characters are compared.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcmp`. Not implemented in all versions of C++.

Display 8.2 (1/2)

C Strings to Integers (*part 1 of 2*)

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

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;
}
```

Display 8.2 (2/2)

Back Next

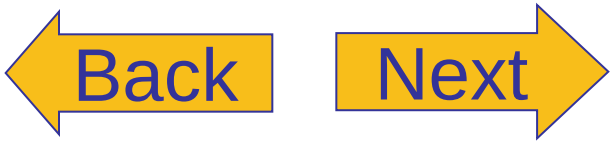
C-Strings to Integers ***(part 2 of 2)***

```
//Uses iostream, cstdlib, and cctype:
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% &&5 *12
That string converts to the integer 23512
Again? (yes/no): no
```

Display 8.3 (1/3)



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)

Display 8.3 (2/3)

Back Next

DISPLAY 8.3 Robust Input Function *(part 2 of 3)*

```
8   void new_line( );
9   //Discards all the input remaining on the current input line.
10  //Also discards the '\n' at the end of the line.
11  void get_int(int& input_number);
12  //Gives input_number a value that the user approves of.
13  int main( )
14  {
15      using namespace std;
16      int input_number;
17      get_int(input_number);
18      cout << "Final value read in = " << input_number << endl;
19      return 0;
20  }
21  //Uses iostream and read_and_clean:
22  void get_int(int& input_number)
23  {
24      using namespace std;
25      char ans;
26      do
27      {
28          cout << "Enter input number: ";
29          read_and_clean(input_number);
30          cout << "You entered " << input_number
31               << " Is that correct? (yes/no): ";
32          cin >> ans;
33          new_line( );
34      } while ((ans != 'y') && (ans != 'Y'));
35  }
36  //Uses iostream, cstdlib, and cctype:
37  void read_and_clean(int& n)
```

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

```
38  //Uses iostream:
39  void new_line( )
```

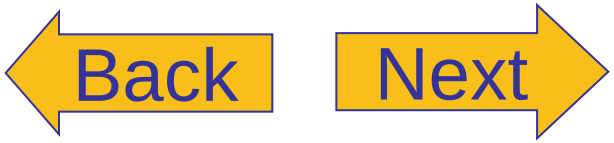
<The rest of the definition of `new_line` is given in Display 8.2.>

Sample Dialogue

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

(continued)

Display 8.3 (3/3)



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

```
Enter input number: $77*5xa
You entered 775 Is that correct? (yes/no): no
Enter input number: 77
You entered 77 Is that correct? (yes/no): no
Enter input number: $75
You entered 75 Is that correct? (yes/no): yes
Final value read in = 75
```

Display 8.4

Program Using the Class string

```
//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;
}
```

Initialized to the empty string

Two ways of initializing a string variable

Sample Dialogue

```
I love fried ants!
Bon appetite!
```

Display 8.5 (1/2)

Program Using the Class `string` (*part 1 of 2*)

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

void new_line( );

int main( )
{
    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;
}
```

Display 8.5 (2/2)

Program Using the Class `string` (*part 2 of 2*)

```
//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

```
//Demonstrates using a string object as if it were an array.
#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
```

Display 8.6

Back Next

Display 8.7

Back Next

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`.