Chapter 11

Strings and Vectors

Created by David Mann, North Idaho College
Overview

- An Array Type for Strings (11.1)
- The Standard string class (11.2)
- Vectors (11.3)
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:

```
```

```
    H i   M o m ! \0   ?   ?
```
To declare a C-string variable, use the syntax:

```c
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'};
Assignment With C-strings

- This statement is illegal:

```c
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:
    ```c
    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:
  ```cpp
  #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:
    ```
    char string_var[20] = "The rain";
    strncat(string_var, "in Spain", 11);
    ```
C-strings can be output with the insertion operator

Example:

```c
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:
    ```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
    ```
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
  
  ```c++
  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

\[
\textit{cin}.\text{getline}(\textit{String\_Var}, \textit{Max\_Characters} + 1);
\]

- \textit{cin} can be replaced by any input stream
- \textit{Max\_Characters} + 1 reserves one element for the null character
"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 \texttt{cstdlib}

- To use the functions use the include directive

  \#include <cstdlib>
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:

    ```cpp
    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 11.4**
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);
This code demonstrates the use of getline with string objects.

```cpp
string line;
cout "Enter a line of input:\n";
ggetline(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

```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;
getline(cin, s1) >> s2;
```
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 `iostream` class.
- `ignore` can be used to read and discard all the characters, including the newline character `\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!
The string class member function `length` returns the number of characters in the string object:

Example:

```cpp
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';
    ```

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

```c
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
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);`
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:

```cpp
for (unsigned int i = 0; i < sample.size(); i++)
  cout << sample[i] << endl;
```
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
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;
```
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
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`
    - Example: `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`