Chapter 8

Strings and Vectors
Objectives

In this chapter, you will learn about:

- An Array Type for Strings
- The Standard `string` Class
- Vectors
An Array Type for Strings
C-String Values and C-String Variables

- The C Language Representation of Strings (C-Strings)

- The C-String Variable:
  - Array of Characters
  - The Null Character '\0' used to Mark End of String in Array
    - The '\0' is a Single Character Stored in One Indexed Array Variable
C-Strings
Declaration

- Lets Declare a C-String for 10 Characters:
  ```c
  char s[11];
  ```
- The 10 Characters are stored in Indexed variables:
  ```c
  s[0], s[1], ..., s[9]
  ```
- The null character ‘\0’ is stored s[10]

Example:

```
K E N T   S T A T E \0
```
Consider C-string variable:

```
char s[11];
```

Then 10 or **less** Characters can be stored in C-String variable s:

Example:

```
[0x0] [0x0] [0x0] [0x0] [0x0] [0x0] [0x0] [0x0] [0x0] [0x0] [0x0]
C S 2 3 0 2 1 \0 ?? ??
```

C-String Variable, s Stores 8 Characters
C-String Variable
Initialization

1. Initialize in Declaration Statement (with Size specified):
   - char your_string[ 19] = “To Be Or Not To Be”;
     
     \[
     \begin{array}{cccccccccccc}
     & & & & & & & & & & & \\
     T & o & B & e & O & r & N & o & t & T & o & B & e & \textbf{\textbackslash o}
     \end{array}
     \] 
   - The null character is automatically added

2. Initialize in Declaration Statement (Omit size)
   - char my_string[ ] = “To Be”; 
     
     \[
     \begin{array}{cccc}
     & & & \\
     T & o & B & e & \textbf{\textbackslash o}
     \end{array}
     \] 
   - Variable has size one character longer than length of string
Consider initialization:

- Char our_string[ ] = {'T', 'o', ' ', 'B', 'e'};

Is our_string a C-string variable?
Consider initialization:

- char our_string[ ] = {'T', 'o', ' ', 'B', 'e'};

Is our_string a C-string variable?

The array our_string does not contain a null character → Not a C-string
C-String Variable
Manipulating Index Variables

Caution

- Do not to Destroy the null character when Manipulating Indexed Variables in a C-string

Example:
```c
char our_string[10] = "To Be";
int index = 0;
while (our_string[index] != '\0')
{
    our_string[index] = 'X';
    index++;
}
```

- This code depends on finding the null character!

The array no longer acts like a C-string when '\0' is replaced
Illegal Assignment statement:
- `our_string = "To Be";`
- The assignment operator `=` does not work with C-String
# C-String Assignment

**Assignment using `strcpy`**

```c
#include <cstring>  // `strcpy` is defined in this library

char my_string[6];
strcpy (my_string, "Hello");  // Places "Hello" followed by the null character in `my_string`
```

- `strcpy` does not check the declared length of the first argument, `my_string`
- `strcpy` may copy characters **beyond** the declared size of the array:
  ```c
  char my_string[6];
  strcpy (my_string, "Hello World");
  ```


**C-String**

**Safer Assignment**

```c
#include <cstring>  // strncpy is defined in this library
...
char another_string[10];
strncpy (another_string, a_string, 9);

// Copies up to 9 characters from a_string to another_string leaving one
// space for ‘\0’
```
The `==` operator does not work with C-strings. Use function `strcmp` (i.e. String Compare) instead.

Example

```c++
#include <cstring>    // strcmp is defined in this library

if (strcmp(c_string1, c_string2))
    cout << "Strings are not the same.";
else
    cout << "String are the same.";
```
C-String

`strcmp`

- `strcmp` compares the numeric codes (e.g., ASCII representation) of elements in the C-strings one character at a time
- `strcmp` returns:
  - **0** if the two C-strings are the same
    - Note 0 is interpreted as `false`
  - **Negative Value** if the numeric code in the first parameter is less than that of the second
  - **Positive Value** if the numeric code in the second parameter is less than that of the first
    - Note Non-zero values (Positive/Negative) are interpreted as `true`
C-String
cstring Library Functions..

**strlen**

`strlen` returns the number (Integer value) of characters in a string.

```c
int x = strlen(a_string);
```

**strcat**

`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] = “Summer”;        
strcat(string_var, “Holidays”);         
Now string_var contains “SummerHolidays”
```

Do you see a problem with `strcat`?
**C-String**

cstring Library Functions..

`strncat` is a safer version of `strcat`

```c
strncat(rithm, " Day in Georgia", 4);
```

Specifies a limit for the number of characters to append

Now, `rithm` contains 11 characters plus the null terminator:
Rainny Day\0

Refer to Pages 456 and 983 for more pre-defined functions
Recall, C-string variable is a char array.

Hence if C-string variable is an array parameter:

- Since functions can change array parameters:
  - Include an additional int parameter for the declared size of the C-string variable.
  - The function will use the declared size to manipulate the C-string.
Use the insertion operator to output a C-string

Example:

```c++
char new_course[ ] = "Requirements Engineering ";
cout << new_course << " Wow." << endl;
```
Use the extraction operator >> to read characters into a C-string:

Note: Whitespaces delineate data items

Example:

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

Output could produce:

Enter input:
Do be do to you!
DobeEnd of Output
Use member function: getline
- To read a line of input (including spaces) and store the characters into a C-string variable
- getline is a member of all input streams

Syntax

```cpp
cin.getline(String_Var, Max_Character +1);
```

- One line of input will be read from Input Stream
- Maximum characters read
- Holds characters read
- Reserved for ‘\0’
The following code is used to read an entire line including spaces into a single C-string variable:

```cpp
char phrase[80];
cout << "Enter input:\n";
cin.getline(a, 80);
cout << a << "! " << End Of Output\n";
```

**Output**

Enter input
To Be Or Not To Be
To Be Or Not To Be! End Of Output
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;
  ```
C-string
Number Conversion

- Recall “1234” is different from 1234
  - “1234" is a sequence of characters (String)
  - 1234 is a number
- When designing numeric input, it is useful to read data as a string of characters, edit the string then convert the string to a number
  - If reading money input may or may not have dollar sign
  - If reading percentages input may or may not have percent sign at the end
To read an integer as characters

- Read input as characters into a C-string, **edit** unwanted characters
- Use the predefined function **atoi** to convert the C-string to an integer value

Example:

```c
#include <cstdlib>

- atoi("1234") returns the integer 1234
- atoi("#123") returns 0 because # is not a digit
```

Use pre-define functions to edit the C-string: **isdigit**()
C-string

atof

- C-strings can be converted to type double using predefined function `atof`
- `atof` returns a value of type double

Example:
- `#include <cstdlib>`
- `atof("$9.99")` returns 0.0 because the `$` is not a digit
The Standard String Class

- Recall C-string:
  - Array of Character terminated by ‘\0’
  - Programmer needs to track low-level storage details for string manipulation

- The string class allows:
  - Programmer to treat strings as another Basic data type
  - No need to worry about low-level details
The Standard string Class

Using string Class

- To use the string class you need to insert the following lines:

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

Why?
Use the operator “=” to assign a value to a string

Example: Assign a string object to another string object

```cpp
string s2, s4, s6;
...
  s2 = s4;
```

Example: Assign a C-string to string object

```cpp
string s2, s4, s6;
...
  s6 = "To Be Or Not To Be";
```

Typed cast to string Object
The Class string
Concatenating string objects

Use the “+” operator to Concatenate string objects

Example:

```cpp
string s2, s4, s6;
...
s2 = s4 + s6;
```

If the combined length of s4 and s6 exceeds the “capacity” of s2, more space will be allocated for s2.
The Class string Constructs

- Default Construct
  - string phrase;

- Using C-string argument:
  - string noun("ants")
  - string noun = “ants”;
The Class string
Concatenating string objects and C-strings

- You can Concatenate string objects and C-strings:

Example

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

How?
- Type conversion of C-string to string object
- Or Using overloading for operator +
The Class string

Output string objects

- Use Insertion Operator “<<” to output string objects

  Example
  ```cpp
  string phrase;
  phrase = "To Be Or Not To Be";
  cout << phrase;
  ```

- Use the Extraction Operator “>>” to input value into string objects:

  Example
  ```cpp
  string s1, s2;
  cin >> s1;
  cin >> s2;
  ```

  User types: **The Kent State University**

  Blank character/Whitespace is a delimiter for cin

  Cannot use **cin** to read a blank character
The Class string
Reading Blank Character/Entire Line

- Use `cin.get` to read a blank character into a char variable not string object
  - `char next_char;`
  - `cin.get( next_char);`

- How do you read an entire line into a string variable?
  - Use the getline non-member function:
  - Syntax:
    ```
    getline(Istream_Object, String_Object);
    ```
The Class string
Reading Entire Line

Example -1 Read entire line into string object

```cpp
string line;
cout "Enter a line of input:\n";
getline(cin, line);
cout << line << "END OF OUTPUT\n";
```

Example -1 Possible Output

Enter some input:
Spring is Here!
Spring is Here!END OF OUTPUT

User types

Reads until ‘\n’ is encountered
The Class string

Reading Entire Line

Example -2 Read up to an end-of line marker into string object

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

Example -1 Possible Output

Enter some input:
Summer? No Spring is Here!
SummerEND OF OUTPUT

The getline function stops reading at a character specified in the 3rd Argument
The Class string

getline

- getline (cin, s1)
  - Reads in a line of text into s1
  - returns a reference to its first argument (cin)

- Example:

  ```
  string s1, s2;
  getline(cin, s1) >> s2;
  ```

  **Results in:**

  ```
  cin >> s2;
  ```
The Class string
getline Declarations

- `istream& getline(istream& ins, string& str_var);`

- `istream& getline(istream& ins, string& str_var, char delimiter);`
The Class string

Mixing cin >> variable; and getline

- Recall cin >> n:
  - Skips leading blank characters
  - Reads non-blank characters
  - Stops reading when a blank character is encountered

- Consider code segment:
  ```cpp
  int n;
  string line;
  cin >> n;
  getline(cin, line);
  ```

- Suppose input:
  ```
  42
  Computer Science
  ```

- What is the output of code?
- Can you explain why?
The Class string

Member function: 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 argument: the maximum number of characters to discard
    - Second argument: the character that stops reading and discarding
  - Example: `cin.ignore(1000, '\n');`
    - Discards rest of line up to '\n'
    - If '\n' not found, discards the leading 1000 characters
The Class string
Processing string

- Can perform all C-string Operations and more using class strings

- Accessing a Character in string Object:
  ```
  string my_name = "John Doe";
  my_name[4] = '-';
  cout << my_name;
  John-Doe
  ```

Caution: my_name[i] does not validate the index i
The Class string

**string length**

- Use the class member function, `length`, to find the number of characters in a string

- **Length of String:**
  ```cpp
  string my_name = "John Doe";
  int = size;
  size = my_name.length();
  cout << size;
  ```
  `8`
The Class string
Accessing String Character

- Safer to use Member Function `at` to index a string object instead of the `[i]` indexing notation
  - The `at` checks for illegal index values:

```
string str("Mary");
cout << str[6] << endl;
```

Illegal operation – Why?
May not generate any error message

```
string str("Mary");
cout << str.at(6) << endl;
```

Illegal operation – gives error message
Terminates the program execution

```
str[3] = ‘k’;
```

```
str.at(3) = ‘k’;
```
Can compare two string objects using the following operators:

`==`, `<`, `>`, `<=`, `>=`

See Page 479 for other string class functions
The Class string
Converting between string Objects and C-strings

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

```
char c_phrase[ ] = "To Be Or Not To Be";
string s_phrase =   c_phrase;
```

- You cannot convert strings to C-strings
  - char c_phrase [ ] = “To Be Or Not To Be”;
  - string s_phrase = “Spring Time”;
  - c_phrase = s_phrase;
  - strcpy(c_phrase, s_phrase);

Illegal Statements!
The Class string

The C-String Version of a string Object

- To obtain a C-string corresponding to string object:
  - Use the Member function `c_str()`;

```cpp
char c_phrase[5];
string s_phrase = "Kent";
strcpy (c_phrase, s_phrase.c_str());
```
Vectors

Basics

- Vectors are
  - Similar to Arrays
    - Base Type (e.g., int, double, char, float)
    - Store elements of the same type
  - But, unlike Arrays they can Grow and Shrink in Length while the Program Runs
    - Vector size is not fixed

So how do we declare a vector?

- Standard Template Library (STL)
  - Standard Data Structures to organize and manipulate data items
    - Container Classes: Vectors, Stack, Queues, lists
    - Generic Algorithms: Template Functions for Searching, Sorting, ..
Vectors

Declaration

- To Declare an Empty Vector (v) with a Base type `int`:
  - `vector <int> v;`

- To Declare an Empty Vector (s) with a Base type `string`:
  - `vector <string> s;`
Vectors

Accessing

How do you access elements of a vector?
- Use Indexing [], similar to arrays:

Example:
- Assume vector (v) is already initialized
- Then to change the $i^{th}$ element:

```
v[ i ] = 42;
cout << "The result is " << v[ i ];
```

Note, use indexing to:
- Read a vector element that already has a value
- Change a Vector element that already has a value

Cannot use indexing to initialize a vector
Vectors

Initializing

- To add an element to a vector position for the first time
  - Use Member function: `push_back`

- Example:

```cpp
vector<double> sample;
sample.push_back(2.0); // Adds 2.0 in position 0
sample.push_back(4.2); // Adds 4.2 in position 1
sample.push_back(6.4); // Adds 6.4 in position 2
```

- `push_back` adds an element in the next available position
Vectors

Size

- To determine number of elements in a vector (Length of vector)
- Use Member function: `size`

Example Vector Initialization:

```cpp
vector<double> sample;
sample.push_back(2.0); // Adds 2.0 in position 0
sample.push_back(4.2); // Adds 4.2 in position 1
sample.push_back(6.4); // Adds 6.4 in position 2

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

- Member Function `sample.size` returns an unsigned integer
Vectors

Default Construct for Initialization

- Construct to Initialize Vector to 0 values:

- Example:
  - `vector <int> v(10)`  // Initializes v[0], v[1], …, v[9] to zero

After the initialization, now, we can:

- “assign” values to elements v[0], v[1], …, v[9]
  - Use Indexing [

- But use `push_back` to “assign” elements v[10], v[11], …
Vector Class

- To use the vector class
  - Include the following two lines:

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

Pitfalls

- Using Square brackets beyond the Vector Size
  - Consider

\[ v[i] = n; \quad \text{where } i > v\text{.isize() } \]

- This will not generate an error message, but your code will not work properly
A vector's capacity refers to the number of elements allocated in memory for storing the vector. Accessible using the capacity() member function.

A vector’s 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.
Vectors

Controlling Vector Capacity

To Increase the Capacity of a Vector:
- Use member function `reserve()`

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

To shrink a vector:
- Use member function `resize()`
- Example:
  ```
  v.resize(24); // elements beyond 24 are lost
  ```