



# Chapter 8

## Strings and Vectors

Prof. Chien-Nan (Jimmy) Liu  
Dept. of Electrical Engineering  
National Chiao-Tung Univ.

Tel: (03)5712121 ext:31211  
E-mail: jimmyliu@nctu.edu.tw  
<http://mseda.ee.nctu.edu.tw/jimmyliu>



Chien-Nan Liu, NCTUEE

## Overview

### 8.1 An Array Type for Strings

### 8.2 The Standard string Class

### 8.3 Vectors



Chien-Nan Liu, NCTUEE

# An Array Type for Strings (C Style)

- In original C language, strings of characters are stored as **arrays of characters**
  - C-strings use the null character '**\0**' to end a string
    - The Null character is a single character (ASCII code = 0)
  - To declare a C-string variable, use the syntax:  
`char Array_name[ Maximum_cString_Size + 1];`
    - + 1 reserves the additional character needed by '\0'
  - Ex: `char s[11];`
    - Maximum 10 characters in this string



Chien-Nan Liu, NCTUEE

8-3

## 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:

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	?	?

- What is the actual length of this string? 7, 8, or 10??



Chien-Nan Liu, NCTUEE

8-4

# Initializing a C-string

- To initialize a C-string during declaration:

```
char myMessage[20] = "Hi there.;"
```

- The null character '\0' is added for you

- Another alternative:

```
char shortString[ ] = "abc";
```

- Count the required size of the given string (including the null character) and fill in the empty bracket

- But not this:

```
char shortString[ ] = {'a', 'b', 'c'};
```

- If you specify each character by yourself, the **null character** at the end is not automatically added



Chien-Nan Liu, NCTUEE

8-5

# Don't Change '\0'

- Do not replace the null character when manipulating indexed variables in a C-string

- If the null character is lost, the array cannot act like a C-string

- Example:

```
int index = 0;
while (ourString[index] != '\0')
{
    ourString[index] = 'X';
    index++;
}
```

- If the null character is missing, this code never stops!!

- To prevent the case of missing '\0' character, add a size limiter in the loop condition

- Example: while (ourString[index] != '\0' && index < SIZE)



Chien-Nan Liu, NCTUEE

8-6

# Assignment With C-strings

- This statement is **illegal** for C-string:  
`aString = "Hello";`
  - This is an assignment statement, not an initialization
  - Assigning an entire array to another is not allowed in C
- 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 aString[ 11];
strcpy (aString, "Hello");
```

Places "Hello" followed by the null character in aString



Chien-Nan Liu, NCTUEE

8-7

# A Problem With strcpy

- `strcpy` can create problems if not used carefully
  - The declared length of the given array is not checked
  - It is possible for `strcpy` to write characters beyond the declared size of the array
- There is a safer version of strcpy named `strncpy`
  - `strncpy` uses a third argument representing the **maximum number** of characters to copy
  - Example:

```
char anotherString[10];
strncpy(anotherString,
        aStringVariable, 9);
```

This code copies up to 9 characters into anotherString, leaving one space for '\0'



Chien-Nan Liu, NCTUEE

8-8

# == Alternative for C-strings

- The == operator does not work for C-strings, either
  - The function *strcmp* is used to compare C-string variables
  - Example:

```
#include <cstring>
...
if (strcmp(cString1, cString2))
    cout << "Strings are not the same.";
else
    cout << "String are the same.";
```
- strcmp* compares the ASCII codes of each character
  - If the first character is the same, compare the next until different character appears
  - If the two C-strings are the same, strcmp returns 0 (**FALSE?**)
  - Return a negative (positive) value if the numeric code in the first parameter is less (more) → interpreted as **TRUE**



Chien-Nan Liu, NCTUEE

8-9

## More C-string Functions (1/3)

- The <cstring> library includes other functions
  - strlen* returns the number of characters in a string

```
int x = strlen( aString);
```
  - 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 stringVar[20] = "The rain";
strcat(stringVar, "in Spain");
```

→ stringVar will contain "The rainin Spain"
  - strncat* is a safer version of strcat with size limiter
    - Ex: *strncat(stringVar, "in Spain", 11);*



Chien-Nan Liu, NCTUEE

8-10

# More C-string Functions (2/3)

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

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



Chien-Nan Liu, NCTUEE

8-11

# More C-string Functions (3/3)

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

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



Chien-Nan Liu, NCTUEE

8-12

# C-strings as Arguments and Parameters

- C-string variables are arrays
  - While passing C-string arguments to a function, they are used just like arrays
  - Watch for the null character to detect the end of string
    - No size argument is needed in typical cases
- 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
  - Example:

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



Chien-Nan Liu, NCTUEE

8-13

# Reading an Entire Line

- Using the extraction operator >>, it stops reading data at whitespace
  - Example:

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

```
Enter input:
Do be do to you
DobeEnd of Output
```

The string you entered  
Only output two words
- Member function *getline* can read an entire line, including spaces
  - Two input arguments: a C-string variable and an integer that specifies the max number of element allowed to fill (including the null character at the end)



Chien-Nan Liu, NCTUEE

8-14

# 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: The string you entered  
Do be do to you!   
Do be do to you!End of Output



## getline Syntax

- Syntax for using getline is  
`cin.getline(String_Var, Max_Characters + 1);`
    - Max\_Characters + 1 reserves one element for the null character
  - getline stops reading when the specified number of characters have been placed in the C-string
    - Plus one character reserved for the null character
    - getline stops even if the endl has not been reached
  - C-string input and output work the same way with file streams (replace cin/cout with file stream)
    - Example:  
`inStream.getline(cString, 80);  
out stream << cString;`



# C-String to Numbers

- "1234" is a string of characters
- 1234 is a number
- In some cases, 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
- While reading an integer as characters
  - Remove unwanted characters first, then use the 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



Chien-Nan Liu, NCTUEE

8-17

# Functions to Convert C-string

- Larger integers can be converted using the function *atol*
  - *atol* returns a value of type long
- 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 due to the \$
- Those conversion functions (*atoi*, *atoll*, *atof*) can be found in the library *<cstdlib>*
  - To use those functions, use directive *#include <cstdlib>*



Chien-Nan Liu, NCTUEE

8-18

# Code: Numeric Input

- We now know how to convert C-strings to numbers
- How do we deal with the input?
  - 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 integer



Chien-Nan Liu, NCTUEE

```
//Uses iostream, cstdlib, and ctype:  
void readAndClean(int& n)  
{  
    using namespace std;  
    const int ARRAY_SIZE = 6;  
    char digitString[ARRAY_SIZE];  
  
    char next;  
    cin.get(next);  
    int index = 0;  
    while (next != '\n')  
    {  
        if ( (isdigit(next)) && (index < ARRAY_SIZE - 1) )  
        {  
            digitString[index] = next;  
            index++;  
        }  
        cin.get(next);  
    }  
    digitString[index] = '\0';  
    // add null character at the end  
  
    n = atoi(digitString);  
}
```

8-19

# Code: Confirming Input

- In function *getInt*:
  - Uses *readAndClean* to read the user's input
  - Allows the user to reenter the input until the user is satisfied with the number
- Function *newLine* reads all the characters remaining in the input line
- Discard what follows the first character of the user's response
  - The same as in Ch.6



Chien-Nan Liu, NCTUEE

```
//Uses iostream and readAndClean:  
void getInt(int& inputNumber)  
{  
    using namespace std;  
    char ans;  
    do  
    {  
        cout << "Enter input number: ";  
        readAndClean(inputNumber);  
        cout << "You entered "  
            << inputNumber  
            << " Is that correct? (yes/no): ";  
        cin >> ans;  
        newLine( );  
    } while ((ans != 'y') && (ans != 'Y'));
```

8-20

# Test the Robust Input Function

```
//DISPLAY 8.3 Robust Input Function
#include <iostream>
#include <cstdlib>
#include <cctype>

void readAndClean(int& n);
void newLine( );
void getInt(int& inputNumber);
```

```
int main( )
{
    using namespace std;
    int inputNumber;
    getInt(inputNumber);
    cout << "Final value read in = "
        << inputNumber << endl;
    return 0;
}
```

## Sample Dialogue

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



Chien-Nan Liu, NCTUEE

8-21

# Overview

## 8.1 An Array Type for Strings

## 8.2 *The Standard string Class*

## 8.3 Vectors



Chien-Nan Liu, NCTUEE

8-22

# The Standard string Class

- The **string class** allows the programmer to treat strings as a basic data type
  - No need to know the implementation as with C-strings
- The string class is defined in the **<string>** library and the names are in the **std** namespace
- By default, the string is initialized to an empty string
- Another string constructor takes a C-string argument
  - Example:

```
string phrase;           // empty string  
string noun("ants");   // a string version of "ants"
```



# Assignment of Strings

- string variables can be assigned with **= operator**
  - Example:            **string s1, s2, s3;**  
                       ...  
                       **s3 = s2;**
- Quoted strings are type cast to type string
  - Example:            **string s1 = "Hello Mom!";**
- string variables can be concatenated with **+ operator**
  - Example:            **string s1, s2, s3;**  
                       ...  
                       **s3 = s1 + s2;**
  - If s3 is not large enough to contain s1 + s2, more space is allocated automatically



# I/O With Class string

- The insertion operator << can also be 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;`
  - >> stops on encountering whitespace



Chien-Nan Liu, NCTUEE

8-25

# Program Using the Class string

//DISPLAY 8.4: Demonstrates the standard class string

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

int main( )
{
    string phrase;           // Initialized to an empty string
    string adjective("fried"), noun("ants"); // Two ways of initializing
    string wish = "Bon appetit!";             // a string variable

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

    return 0;
}
```

## Sample Dialogue

I love fried ants!  
Bon appetite!



Chien-Nan Liu, NCTUEE

8-26

# getline and Type string

- A *getline* function exists to read entire lines into a string variable
  - It is NOT a member of the istream class
  - Syntax for using this *getline* is different than that used with *cin.getline(...)*
- Syntax: *getline(Istream\_Object, String\_Object);*

- Example:

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

- Output:

```
Enter some input:
Do be do to you!
Do be do to you!END OF OUTPUT
```



Chien-Nan Liu, NCTUEE

8-27

## Demonstrate getline and cin

//DISPLAY 8.5

```
#include <iostream>
#include <string>

int main( )
{
    using namespace std;

    string firstName, lastName, recordName;
    string motto = "Your records are our records.";

    cout << "Enter your first and last name:\n";
    cin >> firstName >> lastName;
    newLine( );
}
```

```
recordName = lastName + ", " + firstName;
cout << "Your name in our records is: ";
cout << recordName << 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;
```

### 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.
```



Chien-Nan Liu, NCTUEE

8-28

# 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
  - Ex: this code stops reading when a '**?**' is read

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



Chien-Nan Liu, NCTUEE

8-29

# 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:

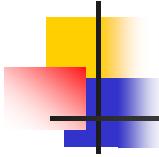
```
string s1, s2;
getline(cin, s1) >> s2;
    |
    v
    returns
    cin
    |
    cin >> s2;
```

- Formal declaration of getline:
  - **istream& getline(istream& ins, string& strVar);**



Chien-Nan Liu, NCTUEE

8-30



# Problem of Mixing cin and getline

- Using `cin >> n`, it stops reading when whitespace or '\n' is found
  - Leaves '\n' and whitespace in the input stream
- In the following example, the `getline` function after `cin` will read an empty line
  - The left '\n' immediately ends getline's reading
  - Example:

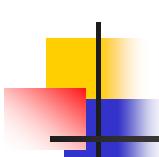
```
int n;
string line;
cin >> n;
getline(cin, line);
```

- How to discard the extra symbol in last reading?



Chien-Nan Liu, NCTUEE

8-31



# ignore Member Function

- `ignore` is a member function of the istream
- `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
  - The read characters are **missed** after this operation
  - Example:

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



Chien-Nan Liu, NCTUEE

8-32

# 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]* accesses a single character as in an array
    - Index values are not checked for validity!
- 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';
```



Chien-Nan Liu, NCTUEE

8-33

## Example: Behave like an Array

```
// DISPLAY 8.6: demonstrates using a string
// object as if it were an array.
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string firstName, lastName;

    cout << "Enter your first and last name:\n";
    cin >> firstName >> lastName;
```

### 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
```

```
cout << "Your last name is spelled:\n";
int i;
for (i = 0; i < lastName.length( ); i++)
{
    cout << lastName[i] << " ";
    lastName[i] = '-';
}
cout << endl;

for (i = 0; i < lastName.length( ); i++)
    cout << lastName[i] << " ";
//Places a "-" under each letter.
cout << endl;

cout << "Good day " << firstName << endl;
return 0;
```



Chien-Nan Liu, NCTUEE

8-34

# string Member Functions

- The member function **length** obtains the number of characters in each string object
  - Not include the NULL character
- The member function **empty** returns true if the string is empty; otherwise, it returns false
- The member function **substr** returns a portion of a string as a string object
  - Two arguments: the first argument is the **starting position** of the sub-string; the second argument is the **number of characters** captured into the sub-string
    - Ex: `substr(5,9)` → starting from position 5, take 9 characters
  - One argument: obtains a substring starting from the specified index



Chien-Nan Liu, NCTUEE

8-35

# Demonstrate string Functions

```
#include <iostream>
#include <string>

int main( )
{
    using namespace std;

    string s1("happy");
    string s2(" birthday");
    string s3, s4;

    cout << "Enter a line of text: ";
    getline(cin, s4);
    cout << "s1 length = " << s1.length()
        << "; s2 length = " << s2.length()
        << "; s3 length = " << s3.length()
        << "; s4 length = " << s4.length(); }

    if (s3.empty()) {
        cout << "s3 is empty. Assign s1 to s3.\n";
        s3 = s1;
        cout << "s3 is \" " << s3 << "\" ";
    }

    cout << "After +=, s1 is " ;
    s1 += s2; // s1 = s1 + s2
    cout << s1 << endl;

    cout << "The substring with 11 characters is "
        << endl << s1.substr(0,11) << endl;
    cout << "The substring starting at location 12 is "
        << endl << s1.substr(12) << endl;

    return 0;
}
```

```
Enter a line of text: Using class string
s1 length = 5; s2 length = 9; s3 length = 0; s4 length = 18
s3 is empty. Assign s1 to s3.
s3 is "happy"
After +=, s1 is happy birthday
The substring with 11 characters is happy birth
The substring starting at location 12 is day
```



Chien-Nan Liu, NCTUEE

8-36

# Comparison of strings

- Comparison operators work with string objects
  - Objects are compared using lexicographic order
    - Alphabetical order is determined by the **ASCII code value** for that symbol
  - **==** 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
    - Compare each character one by one until difference is found
    - Larger or smaller is also determined by the ASCII code value



Chien-Nan Liu, NCTUEE

8-37

# More string Member Functions

## Assignment/modifiers

<code>str1 = str2;</code>	Initializes <code>str1</code> to <code>str2</code> 's data,
<code>str1 += str2;</code>	Character data of <code>str2</code> is concatenated to the end of <code>str1</code> .
<code>str.empty()</code>	Returns <code>true</code> if <code>str</code> is an empty string; <code>false</code> otherwise.
<code>str1 + str2</code>	Returns a string that has <code>str2</code> 's data concatenated to the end of <code>str1</code> 's data.
<code>str.insert(pos, str2);</code>	Inserts <code>str2</code> into <code>str</code> beginning at position <code>pos</code> .
<code>str.remove(pos, length);</code>	Removes substring of size <code>length</code> , starting at position <code>pos</code> .

## Comparison

<code>str1 == str2</code> <code>str1 != str2</code>	Compare for equality or inequality; returns a Boolean value.
<code>str1 &lt; str2</code> <code>str1 &gt; str2</code>	Four comparisons. All are lexicographical comparisons.
<code>str1 &lt;= str2</code> <code>str1 &gt;= str2</code>	

## Finds

<code>str.find(str1)</code>	Returns index of the first occurrence of <code>str1</code> in <code>str</code> .
<code>str.find(str1, pos)</code>	Returns index of the first occurrence of string <code>str1</code> in <code>str</code> ; the search starts at position <code>pos</code> .
<code>str.find_first_of(str1, pos)</code>	Returns the index of the first instance in <code>str</code> of any character in <code>str1</code> , starting the search at position <code>pos</code> .
<code>str.find_first_not_of(str1, pos)</code>	Returns the index of the first instance in <code>str</code> of any character not in <code>str1</code> , starting the search at position <code>pos</code> .



Chien-Nan Liu, NCTUEE

8-38

# 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



Chien-Nan Liu, NCTUEE

8-39

## Flow of this Program

```
//uses functions makeLower, removePunct  
bool isPal(const string& s)  
{  
    string punct(",;.:?!\" \"");  
    //includes a blank  
    string str(s);  
    str = makeLower(str);  
    string lower_str = removePunct(str,punct);  
  
    return (lower_str == reverse(lower_str));  
}
```

- 4 key steps:
  - Turn all characters to lower case
  - Remove punctuation
  - Reverse the string
  - Compare two versions

### 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.



Chien-Nan Liu, NCTUEE

8-40

# Code: removePunct

- *removePunct* removes punctuation from a string

- Compare each character to the characters in another string listing the punctuation characters and the space character
- If a match is not found, the character is added to the string noPunct
- After removing the unwanted characters, noPunct is returned

```
string removePunct(const string& s, const string& punct)
{
    string noPunct; //initialized to empty string
    int sLength = s.length( );
    int punctLength = punct.length( );

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

        if (location < 0 || location >= punctLength)
            noPunct = noPunct + aChar;
            //aChar not in punct, so keep it
    }

    return noPunct;
}
```



Chien-Nan Liu, NCTUEE

8-41

# Code: makeLower & Reverse

```
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;
}
```

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

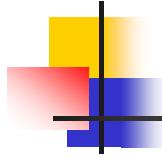
//Uses <cctype> and <string>
string makeLower(const string& s)
{
    string temp(s);
    for (int i = 0; i < s.length( ); i++)
        temp[i] = tolower(s[i]);

    return temp;
}
```



Chien-Nan Liu, NCTUEE

8-42



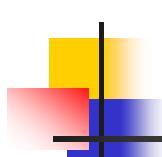
# string Objects to C-strings

- C-string can be converted to string variable automatically
- string variable cannot be converted to C-string
- For example: `char aCString[] = "C-string";`  
                  `stringVar = aCString;`
  - Both of these statements are **illegal**:
    - `aCString = stringVar;`
    - `strcpy(aCString, stringVar);`
- The string class member function `c_str` returns the C-string version of a string object
  - Ex: `strcpy(aCString, stringVar.c_str( ) );`



Chien-Nan Liu, NCTUEE

8-43



## Overview

### 8.1 An Array Type for Strings

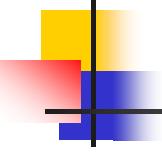
### 8.2 The Standard string Class

### 8.3 *Vectors*



Chien-Nan Liu, NCTUEE

8-44



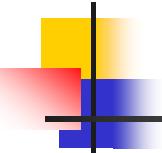
# Vector as a “Better” Array

- C-style pointer-based arrays have great potential for errors and are not flexible
  - A program can easily “walk off” either end of an array, because C++ does not check the subscripts range
  - Two arrays cannot be meaningfully compared with equality operators or relational operators
  - The size of the array must be passed as an additional argument when an array is passed to another function
  - One array cannot be assigned to another with the assignment operator(s)
- C++ Standard class template **vector** represents a more robust type of array with additional capabilities



Chien-Nan Liu, NCTUEE

8-45



# Vectors

- Vectors look like arrays, but they can change size as your program runs
- 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:  
Ex: `vector<string> v;`
- vector class is included in the **<vector>** library
  - Vector names are placed in the standard namespace using namespace std;



Chien-Nan Liu, NCTUEE

8-46

# 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
- Elements are added to a vector using the member function ***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);`
- When a vector runs out of space, its capacity is automatically increased



Chien-Nan Liu, NCTUEE

8-47

# Constructor and Size Functions

- To initialize a specified number of elements in a vector, use its constructor function
  - Example: `vector<int> v(10);`  
→ initializes **10 elements** and set them to 0
    - [ ]'s can now be used to assign elements 0 through 9
    - ***Push\_back*** will assign elements to the location after 9
- The member function ***size*** returns the number of elements in a vector
  - Ex: print each element of a vector after initialization:

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



Chien-Nan Liu, NCTUEE

8-48

# Example: Demonstrate vector Usage

```
//DISPLAY 8.9 Using a Vector
#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. ";
        cout << "v.size( ) = " << v.size() << endl;
        cin >> next;
    }
}
```



Chien-Nan Liu, NCTUEE

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

8-49

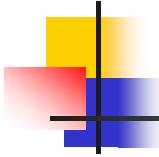
# Operations with vector Objects

- *vector* objects can be compared with another one using the **equality (==)** operators
- You can use the **assignment (=)** operator on vectors
  - Element by element copy of the right hand vector
- You can create a new vector object that is initialized with a copy of an existing vector
- As with C-style arrays, C++ does not perform **bound checking** while using [ ] to access vector elements
- Similar to string, vector provides **member function at** for bound checking. Ex: **v1[2] → v1.at(2)**
  - Throws an exception for invalid subscript



Chien-Nan Liu, NCTUEE

8-50



# Controlling vector Capacity

- 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
  - The actual used memory locations
- When a vector runs out of space, the capacity is automatically increased
  - Member function *reserve()* increases the capacity manually
  - Example: `v.reserve(32); // at least 32 elements`  
`v.reserve(v.size() + 10); // at least 10 more`
- Function *resize* can be used to shrink a vector
  - Example: `v.resize(24); //elements beyond 24 are lost`

