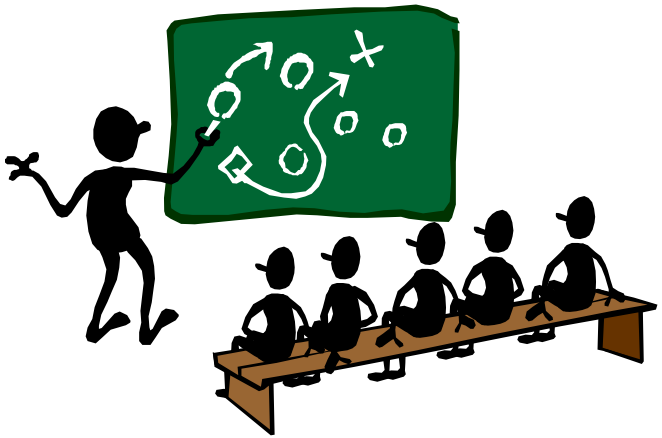
C++ Programming Language
Chapter 9 Strings



Juinn-Dar Huang
Associate Professor
jdhuang@mail.nctu.edu.tw

April 2011

Learning Objectives

- C-style strings (C-strings)
  - zero-terminated arrays of characters
  - functions in <cstring> and <iostream>
- Command line arguments
- Character manipulation tools
  - character I/O (from istream and ostream)
    - functions: get, put, putback, peek, and ignore
  - character manipulating functions in <cctype>
- Standard class string
  - superior to old C-strings in many perspectives
  - lots of string processing functions available

Introduction

Two string types:

- Legacy C-string
  - represented as an array of characters
  - end of string marked with a null character, '\0'
  - old string style inherited from C
- String class
  - newly provided in C++
  - part of the C++ standard library
  - uses templates (Chapter 16, 19)

C-Style Strings

- Array of characters
  - one character per indexed variable
  - one extra character is ALWAYS at the end: '\0'
    - called null character
    - end marker of string
- C-string example

  string literal "Hello" is a C-string

  char str[20] = "Hello world!"; // C-string

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Variables

- Array of characters:

  char s[**10**];

  declare a c-string variable to hold **up to 9** characters

  plus **1** null character

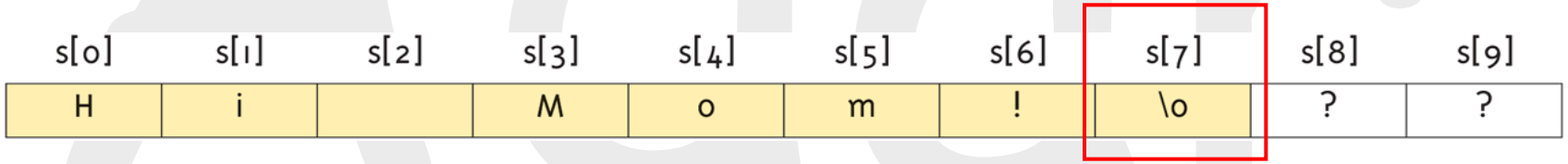
- Typically, partially-filled arrays
  - in practice, declare big enough to hold the longest possible string
- Only difference from an ordinary array of characters
  - **MUST** end with a null character!

C-String Storage

- A C-style string:

  char s[10];

  – if s contains a string “Hi Mom!”, it is stored as:



  – ‘\0’ is **implicitly** appended at the end

  – the value of ‘\0’ is actually 0

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Initialization

- C-string initialization

  char msg[20] = “Hi there.”; // needn’t fill up the entire array
  char msg[10] = “Hello world!” // error! string literal is too long

- Array size can be omitted

  char shortString[] = **“abc”**; // initialized by a **string literal**
  - automatically makes size one more than length of **string literal**
  - in this case, equivalent to shortString[**4**]

  - NOT same as: char shortString[] = {‘a’, ‘b’, ‘c’};
  - in this case, equivalent to shortString[**3**]

C-String Indices

- A C-string is still an array
- Can read and modify indexed variables

```
char ourString[5] = “Hi”;
 // ourString[0] is ‘H’
 // ourString[1] is ‘i’
 // ourString[2] is ‘\0’
 // ourString[3] is ‘\0’
 // ourString[4] is ‘\0’
 ourString[0] = ‘L’;  // ok!
 char ch = ourString[1];  // ok!
```

End of C-String

- Can manipulate indexed variables

  char happyString[7] = “DoBeDo”;
  happyString[6] = ‘Z’;

  – here, ‘\0’ is overwritten by ‘Z’
- Many C-string manipulation functions don’t stop traversing a given string until ‘\0’ is reached
  – e.g., strlen(); (calculate the length of a given C-string)
- Hence, if ‘\0’ is accidentally overwritten
  ➔ unpredictable results for those functions
  ➔ usually disasters!

Header File and Library

- Declaring C-strings
  - require no C++ library
  - no need to include any header files
  - built into standard C++ (just like int, double, …)

- Extra manipulation functions
  - need to include the header file <cstring>
  - typically included when using C-strings

C-String Assignment (1/2)

C-strings are not like other built-in types

- Assignment

```
int val = 10;  // ok, val is initialized with 10
val = 20;      // ok, val is assigned with 20
char str[10] = “Hello”;    // str is initialized with a string literal
char str2[10] = “world!”
str = “world!”;            // error!! str is a constant pointer
str = str2;                // error!! str is a constant pointer
int ia[10], ib[10];
ia = 100;                  // error!! ia is a constant pointer
ia = ib;                   // error!! ia is a constant pointer
```

- The reason is ➔ the type of str:
  - array of characters, or
  - a **constant pointer** pointing to character
  - that is, str cannot be on the left side of assignment operator

“借殼上市”的下場

P31

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Assignment (2/2)

Q: How to do C-string assignment?

A: Use a library function: char* strcpy(char* dest, **const** char* src);

- a built-in library function declared in <cstring>
- string copy from src to dest char-by-char until '\0' is reached
- **NO** checks for string size! programmer's responsibility!

- Example

```
char str1[10] = "Hello";
char str2[20] = "world!";
strcpy (str1, "C++"); // ok now!
strcpy(str1, "Adar's handsome");      // no compilation error,
                                      // but big runtime problem!
strcpy(str1, str2);     // ok!
```

C-String Comparison (1/2)

C-strings are not like other built-in types

- Comparison

  char str1[10] = “Hello”; char str2[10] = “Hello”;

  bool eq1 = (str1 == str2); // eq1 ← false!!!

  int a1[2] = {1, 2}; int a2[2] = {1, 2};

  bool eq2 = (a1 == a2); // eq2 ← false

- The reason is → the type of str:
  - array of characters, or
  - a **constant pointer** pointing to character
  - that is, the value of str represents a specific address

“借殼上市”的下場

P34

C-String Comparison (2/2)

Q: How to do C-string comparison?

A: Use a library function: int strcmp(const char* str1, const char* src);

- a built-in library function declared in <cstring>
- compare str1 against str2 using **lexicographic order**
- return value:
  - negative ➔ str1 < str2
  - 0 ➔ str1 == str2 (str1 and str2 are identical)
  - positive ➔ str1 > str2

- Example

```
char str1[10] = “Hello”;
char str2[20] = “world!”
if ( strcmp(str1, str2) == 0 )
    cout << “Same!\n”  << endl;
```

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Function: strlen()

- Check <cstring> for all available C-string related functions in C++ standard library

- Get string length ➔ size_t strlen(const char*);

  char myString[10] = “dobedo”;
  cout << strlen(myString);

  – return number of characters in a string
    - not including the last null character ‘\0’
  – result here: 6 (not 10)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Function: strcat()

- Concatenate two strings
  ➔ char* strcat(char* dest, const char* src);
  - append src to dest
  - **NO** checks for string size! programmer's responsibility!

```
char str1[30] = "Hello ";
char str2[20] = "wonderful world!\n";
strcat(str1, str2);
cout << str1;          // "Hello wonderful world!"
strcat(str1, str2);    // a big runtime problem here
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Arguments and Parameters

- Recall: C-string is actually an array of characters
- So C-string parameter is array parameter
  - call-by-pointer-value
  - use **const** to protect C-string arguments whenever possible

```
char* strcpy(char* dest, const char* src);

int strcmp(const char* str1, const char* str2);

size_t strlen(const char* s);

char* strcat(char* dest, const char* src);
```

C-String Output with Operator <<

- Output C-strings with operator <<

```
char str[20] = “Hello world!”;
cout  << str << endl;
//  using ostream& operator<<(ostream&, const char*);
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Input with Operator >> (1/2)

- Input C-strings with operator >>
  - istream& operator>>(istream&, char*);
  - however, a bit complicated here …

```
char str1[10] = “Hello ”;
char str2[10] = “world!\n”;
char  str3[20], str4[20];
cout << str1 << str2;// ok, output “Hello world!\n”
cin >> str3 >> str4;  // what if “Hello world!” is entered?
```

- Whitespace
  - whitespace is delimiter
  - tab (‘\t’), space (‘ ’), newline (‘\n’), …
  - input breaks at delimiter while using “cin >> …”
  - multiple consecutive tabs/spaces ➔ one delimiter!

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-String Input with Operator >> (2/2)

```
char a[80], b[80];  int c, d;
cin >> c >> d;            // input "  123          45  " is ok
cout << "Enter input: ";
cin >> a >> b;
cout << a << b << "End of Output\n";
```

Note:
Underlined portion typed at keyboard

- Dialogue:

  Enter input: Do be do to you!

  DobeEnd of Output

- Beware of C-string size
  - must be large enough to hold entered string!
  - C++ gives no warnings; programmer's responsibility

```
char str[2];
cin >> str;    // if keying in "Hello", a big runtime problem
```

C-String Line Input (1/2)

- What if we want to input a string having whitespaces?
- Use cin.getline()
  - getline(char* s, streamsize n) is a member function of class istream
  - it can receive an entire input line into C-string

```
char a[80];
cout << “Enter input:” ;
cin.getline(a, 80);
cout << a << “END OF OUTPUT\n”;
```

  - dialogue:

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

C-String Line Input (2/2)

- Can explicitly tell the maximum length to receive:

```
char shortString[5];
cout << “Enter input: ”;
cin.getline(shortString, 5);
cout << shortString << “END OF OUTPUT\n”;
```

  - results:
    Enter input: dobedowap
    dobeEND OF OUTPUT
  - forces FOUR characters only be read
    - the last one for a null character!

Command Line Arguments (1/3)

- Programs invoked from the command line (e.g., a UNIX shell, DOS command prompt) can be sent arguments
  - example: copy c:\foo.txt d:\foo2.txt
  - This runs the program named “copy” and sends in two C-string arguments, “c:\foo.txt” and “d:\foo2.txt”
    - It is up to the COPY program to process the inputs presented to it; i.e. actually copy the files
- Arguments are passed as an array of C-strings to main()

Command Line Arguments (2/3)

- Declaration of main()
  - int main(int argc, char *argv[]);
  - argc specifies how many arguments are supplied
  - name of the program counts ➔ argc will be at least 1
  - argv is an array of C-strings
    - argv[0] holds the name of the program that is invoked
    - argv[1] holds the name of the first parameter
    - argv[2] holds the name of the second parameter
    - and so on

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Command Line Arguments (3/3)

```
// Echo back the input arguments
int main(int argc, char *argv[]) {

    for (int i = 0; i < argc; ++i)
        cout << "Argument "  << i << ": " << argv[i] << endl;

    return 0;
}
```

Sample Execution

```
> Test
Argument 0: Test
```

Invoking Test from command prompt

Sample Execution

```
> Test hello world
Argument 0: Test
Argument 1: hello
Argument 2: world
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Character I/O (1/3)

- Use cin.get() to read one character at a time
  - istream& get(char&);
  - member function of class istream:

```
char nextSymbol;
cin.get(nextSymbol);
```

  - read next character and put into nextSymbol
  - the next character can be ‘ ’ and ‘\t’, … (whitespace)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Character I/O (2/3)

More member functions of class istream

- istream& unget();
  - undo the last get() call
  - cin.unget();
- istream& putback(char);
  - once read, might need to put it back
  - cin.putback(lastChar);
- int peek();
  - return the next character, but leave it there
  - peekChar = cin.peek();
- istream& ignore(streamsize n, int c);
  - skip at most n characters, or a designated delimiter character c is found
  - cin.ignore(1000, '\n');
    - skips at most 1000 characters, or '\n' is found

Check class istream for more details

Character I/O (3/3)

- Use cout.put() to write one character at a time
  - ostream& put(char);
  - member function of class ostream
- Examples:

```
cout.put('a');
char myString[10] = "Hello";
cout.put(myString[1]); // output letter 'e'
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Functions in <cctype> (1/3)

- Upper-lower case conversions
  - int toupper(int);
  - int tolower(int);
- Examples:

```
char ch1, ch2;
ch1 = ‘a’;
ch2 = toupper(ch1); // ch2 = ‘A’
ch2 = toupper(‘B’);   // ch2 = ‘B’
ch2 = toupper(‘5’);   // ch2 = ‘5’
ch2 = tolower(‘A’);   // ch2 = ‘a’
ch2 = tolower(‘b’);   // ch2 = ‘b’
ch2 = tolower(‘5’);   // ch2 = ‘5’
```

Functions in <cctype> (2/3)

int isXXXXX(int c);
return nonzero if true; zero if false

isalnum	Check if character is alphanumeric (function)
isalpha	Check if character is alphabetic (function)
iscntrl	Check if character is a control character (function)
isdigit	Check if character is decimal digit (function)
isgraph	Check if character has graphical representation (function)
islower	Check if character is lowercase letter (function)
isprint	Check if character is printable (function)
ispunct	Check if character is a punctuation character (function)
isspace	Check if character is a white-space (function)
isupper	Check if character is uppercase letter (function)
isxdigit	Check if character is hexadecimal digit (function)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Functions in <cctype> (3/3)

ASCII values	characters	iscntrl	isspace	isupper	islower	isalpha	isdigit	isxdigit	isalnum	ispunct	isgraph	isprint
0x00 .. 0x08	NUL, (other control codes)	x										
0x09 .. 0x0D	(white-space control codes: '\t','\f','\v','\n','\r')	x	x									
0x0E .. 0x1F	(other control codes)	x										
0x20	space (' ')		x									x
0x21 .. 0x2F	!"#$%&'()*+,-./									x	x	x
0x30 .. 0x39	01234567890						x	x	x		x	x
0x3a .. 0x40	:;<=>?@									x	x	x
0x41 .. 0x46	ABCDEF			x		x		x	x		x	x
0x47 .. 0x5A	GHIJKLMNOPQRSTUVWXYZ			x		x			x		x	x
0x5B .. 0x60	[\]^_`									x	x	x
0x61 .. 0x66	abcdef				x	x		x	x		x	x
0x67 .. 0x7A	ghijklmnopqrstuvwxyz				x	x			x		x	x
0x7B .. 0x7E	{|}~									x	x	x
0x7F	(DEL)	x										

For more details, check
http://www.cplusplus.com/reference/clibrary/cctype/

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Class string

- Defined in the standard C++ library
  #include <string>
  using namespace std;
- Can perform assignment, comparison, addition, …

  Example:

```
string s1, s2, s3;
s3 = s1 + s2;          // concatenation
s3 = “Hello Mom!”      // assignment
```

  – note C-string “Hello Mom!” can be assigned to a string using string& operator=(const char*);

Constructors and Assignment

- Ctors
  - string(); // default, an empty string
  - string(const string&); // copy ctor
  - string(const char* s); // a string initialized by s
  - and more …

- Assignment operators (member functions)
  - string& operator=(const string&);
  - string& operator=(const char*);
  - string& operator+=(const string&);
  - string& operator+=(const char*);
  - and more …

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Capacity and Element Access

- Capacity (member functions)
  - size_t size() const; // get string length
  - size_t length() const; // get string length; same as size()
  - bool empty() const; // Is it an empty string?
  - and more …
- Element access (member functions)
  - char& operator[](size_t p);
    // return the reference of pth character in string, no range checking
  - char& at(size_t p);
    // return the reference of pth character in string, with range checking
  - and more …

```
string str("hello");
str[0] = 'H';  // str contains "Hello" now
int i = str.size();  // i = 5
```

Global Functions

- Concatenations
  - string operator+(const string&, const string&);
  - string operator+(const string&, const char*);
  - string operator+(const char*, const string&);
  - and more …
- Comparisons
  - bool operator==(const string&, const string&);
  - bool operator==(const char*, const string&);
  - bool operator==(const string&, const char*);
  - similarly, for !=, >, >=, <, <=
  - using lexicographic order
- Swap
  - void swap(string& lhs, string& rhs); // swap contents of lhs and rhs

P12

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Uses of string

Display 9.4 Program Using the Class string

```
1   //Demonstrates the standard class string.
2   #include <iostream>
3   #include <string>
4   using namespace std;

5   int main( )
6   {
7       string phrase;
8       string adjective("fried"), noun("ants");
9       string wish = "Bon appetite!";

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

13      return 0;
14  }
```

Initialized to the empty string.

Two equivalent ways of initializing a string variable

SAMPLE DIALOGUE

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


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

string I/O with >> and <<

- Operators >> and << are overloaded for string type
  - istream& operator>>(istream&, string&);
  - ostream& operator<<(ostream&, const string&);

```
string s1, s2, s3("Hello world!");
cin >> s1 >> s2;
cout << s3;
```

- Results
  User types in: Long live the king!
- Extraction still ignores whitespaces
  s1 receives value "Long"
  s2 receives value "live"

string I/O with getline() (1/2)

- To get a complete input line
  - global function: istream& getline(istream&, string&);

```
string line;
cout << “Enter a line of input: ”;
getline(cin, line);
cout << line << “END OF OUTPUT”;
```

- Dialogue produced

Enter a line of input: Do be do to you!
Do be do to you!END OF INPUT

  - Similar to C-string’s usage of getline()

string I/O with getline() (2/2)

- You can specify your own delimiter character
  - istream& getline(istream&, string&, char delim);

```
string line;
cout << “Enter input: ”;
getline(cin, line, ‘?’);
```

  - receives input until ‘?’ is encountered
- getline() returns reference
  - string s1, s2;
    getline(cin, s1) >> s2; // ok to do this

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Insertion and Deletion Operations

- Append (member functions)
  - string& append(const string&);
  - string& append(const char*);
  - and more …
- Insert (member functions)
  - string& insert(size_t pos, const string& str);
  - string& insert(size_t pos, const chat* s);
  - and more …
- Erase (member functions)
  - string& erase(size_t pos = 0, size_t n = npos);
  - void clear(); // reset to an empty string
- Replace (member functions)
  - string& replace(size_t pos, size_t n, const string& str);
  - string& replace(size_t pos, size_t n, const char* s);
  - and more …

Substring and Find Operations (1/2)

- Substring (member functions)
  - string substr(size_t pos = 0, size_t n = npos) const;
- Find (member functions)
  - size_t find(const string& str, size_t pos = 0) const; // first one
  - size_t find(const char* s, size_t pos = 0) const;
  - size_t rfind(const string& str, size_t pos = npos) const; // last one
  - size_t rfind(const char* s, size_t pos = npos) const;
  - size_t find_first_of(const string& str, size_t pos = 0) const;
  - size_t find_last_of(const string& str, size_t pos = npos) const;
  - size_t find_first_not_of(const string& str, size_t pos = 0) const;
  - size_t find_last_not_of(const string& str, size_t pos = npos) const;
  - and more …

Substring and Find Operations (2/2)

```
string str ("Replace the vowels in this sentence by asterisks.");
size_t found;

found=str.find_first_of("aeiou");
while (found!=string::npos)
{
  str[found]='*';
  found=str.find_first_of("aeiou",found+1);
}

cout << str << endl;
```

```
R*pl*c* th* v*w*ls *n th*s s*nt*nc* by *st*r*sks.
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

C-string and string Conversions

- Conversions
  - from C-string to string

    ```
    char cstr[] = “My C-string”;
    string str;
    str = cstr;  // ok!

    cstr = str;   // compilation error!, cannot auto-convert to C-string
    ```

  - must use explicit conversion from string to C-string
    `const char* c_str() const;  // member function of class string`
  - c_str() returns an array of null-terminated character sequence with the same content as the string object

    ```
    strcpy( cstr, str.c_str() ); // ok now!
    ```

For more details of class string, check http://www.cplusplus.com/reference/string/

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Summary

- C-string is an array of characters
  - always ends with a null character, '\0'
- C-strings act like arrays, not a simple type
  - cannot be directly assigned and compared like simple variables
- <cstring> and <cctype> provide many useful C-string manipulation functions
- Command line arguments
- New class string in C++
  - very powerful; many manipulation functions available
  - prefer C++ string to C-string !
- Operators >> & << are overloaded for C-string & string I/O