LAB 7

STRINGS

UEE 1303
Department of Electrical and Computer Engineering, Institute of Electronics
National Chiao Tung University

Outline

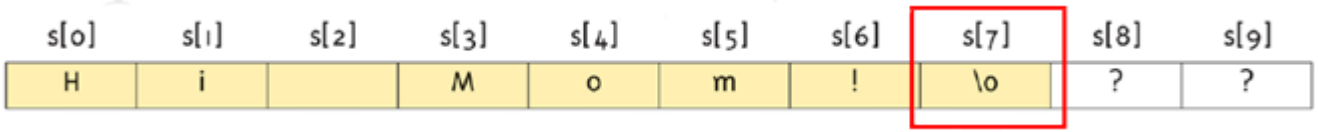
- C-string
- String class

Strings

- 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-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

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 short_String[] = “abc”; // initialized by a string literal

  - automatically makes size one more than length of string literal
  - in this case, equivalent to short_String[**4**]

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

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

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

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;
```

Other Functions

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

    char myString[10] = “dobedo”;

    cout << strlen(myString); //result: 6 (not 10)

- **strcat()**
  - Concatenate two strings char* strcat(char* dest, const char*src);
  - **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

C-String Input with Operator >>

- Input C-strings with operator >>
  - istream& operator>>(istream&, char*);
  - Whitespace is delimiter
  - input breaks at delimiter while using “cin >> …”
  - must be large enough to hold entered string!
  - C++ gives no warnings; programmer’s responsibility!

```
char a[80], b[80];
cout << “Enter input: ”;
cin >> a >> b;
cout << a << b << “End of Output\n”;
```

- Dialogue

  Enter input: Do be do to you!

  DobeEnd of Output

C-String Line Input (1/2)

Q: What if we want to input a string having whitespaces?

A: Use cin.getline()

- getline(char* s, stream_size n)
- 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!

Functions in <cctype> (1/2)

- 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> (2/2)

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)

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 …

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
```

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!
```

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: ”;
cin.clear();
getline(cin, line, ‘?’);
```

  - receives input until ‘?’ is encountered

- getline() returns reference

```
string s1, s2;
getline(cin, s1) >> s2;          // ok to do this
```

Substring and Find Operations (1/3)

- 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 (2/3)

```
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.
```

Substring and Find Operations (1/3)

- Example:

```
string s = "Hello, World!";
size_t pos1 = s.find("World");       // pos1 = 7
size_t pos2 = s.find('o', 5);        // pos2 = 8

string s = "Hello, World!";
size_t pos1 = s.rfind("o");          // pos1 = 8
size_t pos2 = s.rfind('l', 5);       // pos2 = 3 Search for 'l' starting from the 5th position counting backward

string s = "Hello, World!";
size_t pos = s.find_first_of("aeiou");  // pos = 1 (position of the first vowel 'e')

string s = "Hello, World!";
size_t pos = s.find_last_of("aeiou");   // pos = 8 (position of the last vowel 'o')

string s = "Hello, World!";
size_t pos = s.find_first_not_of("Helo,"); // pos = 6 (position of the first character 'W' not in "Helo,")

string s = "Hello, World!";
size_t pos = s.find_last_not_of("dlr!");    // pos = 8 (position of the last character ',' not in "dlr!")
```

Exercise (1/2)

- Input:
  - an integer to choose mode
  - a string to convert

- Output:
  - -1: the program ends
  - 0: replaces substrings in left table with substrings in right table
  - 1: replaces substrings in right table with substrings in left table

A	B
(happy)	^w^
(heart)	<3
(confused)	?_?
(kiss)	-3-
(speechless)	= =

Exercise (2/2)

```
Please enter the mode: 0
Input: Hello (heart), does the C++ program work? (happy)
Output: Hello <3, does the C++ program work? ^w^
-------------------------------------
Please enter the mode: 1
Input: Hmmm, it works...-3-, but I don't know why...?_?.
Output: Hmmm, it works...(kiss), but I don't know why...(confused).
-------------------------------------
Please enter the mode: 0
Input: ...(speechless)
Output: ...= =
-------------------------------------
Please enter the mode: -1
Program ends
```

istream*& ignore(streamsize, int)

- Problem with using cin >> and getline(cin, s) together
  - cin >> will left '\n' in the buffer
  - If then execute getline(), it will receive '\n' immediately
  - use cin.ignore (100, '\n'); to extract characters from the input sequence and discard them until either 100 characters have been extracted or one compares equal to '\n'

```
string s0, s1;
cin >> s0;
cin.ignore(100, '\n');    // If don't add this, s1 will be an empty string
getline(cin, s1);
cout << "s0 = " << s0 << endl;
cout << "s1 = " << s1 << endl;
```

More Function of Class string

- string - C++ Reference - Cplusplus.com