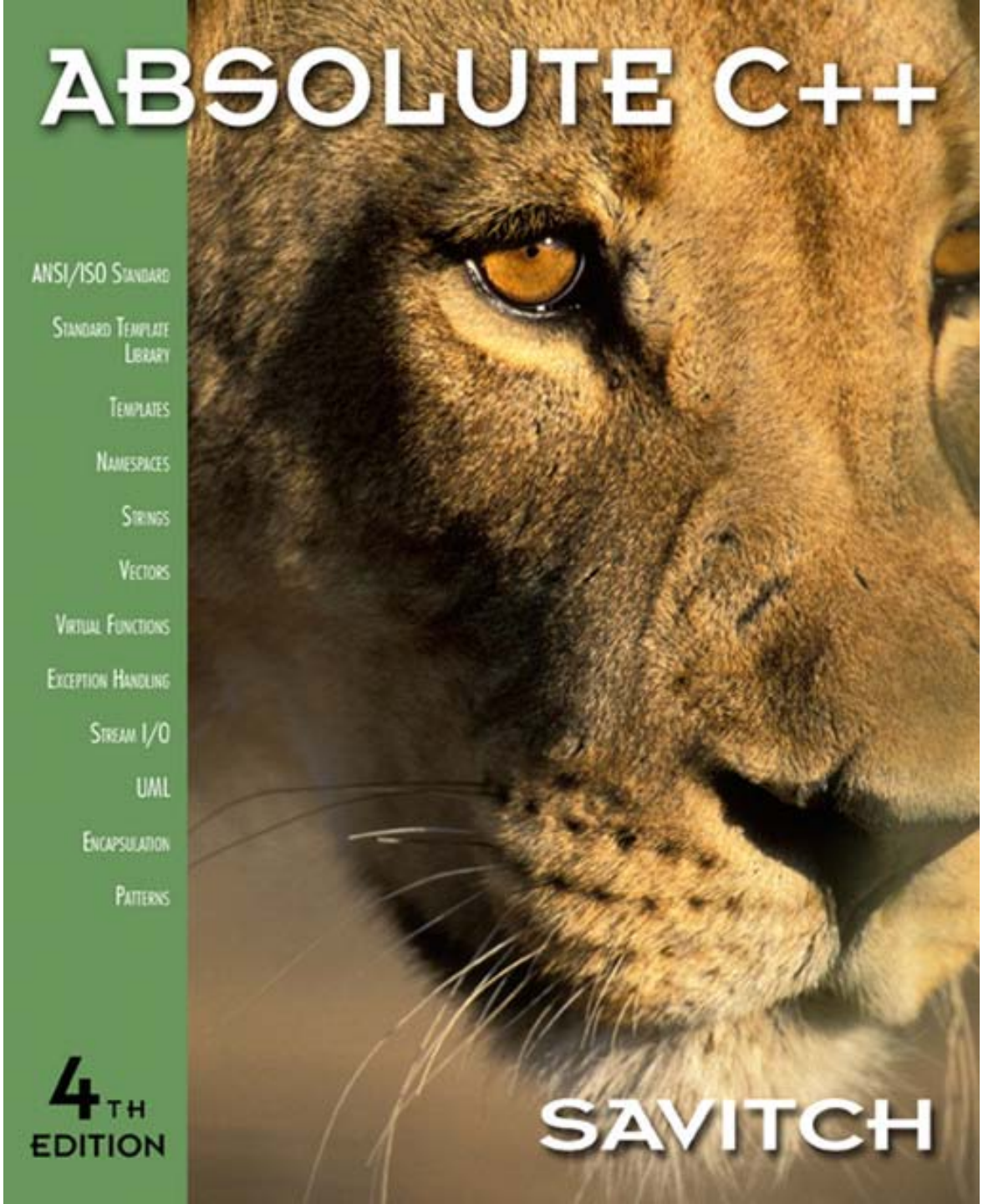
Bài 9: Xâu

Giảng viên: Hoàng Thị Điệp
Khoa Công nghệ Thông tin – ĐH Công Nghệ



Chapter 9

Strings

PEARSON Addison Wesley

Mục tiêu bài học

- Xâu kí tự kiểu mảng
  - Kiểu xâu kí tự của C (xâu C)
- Các công cụ xử lý kí tự
  - Đọc/ghi kí tự
  - Hàm thành viên get, put
  - putback, peek, ignore
- Lớp chuẩn string
  - Xử lý xâu

Giới thiệu

- Hai kiểu xâu kí tự:

1. Xâu C
   - Mảng với kiểu cơ sở là char
   - Đánh dấu kết thúc xâu bằng null, '\0'
   - Kĩ thuật "cũ" thừa kế từ C
2. Lớp string
   - Sử dụng khuôn mẫu

Xâu C

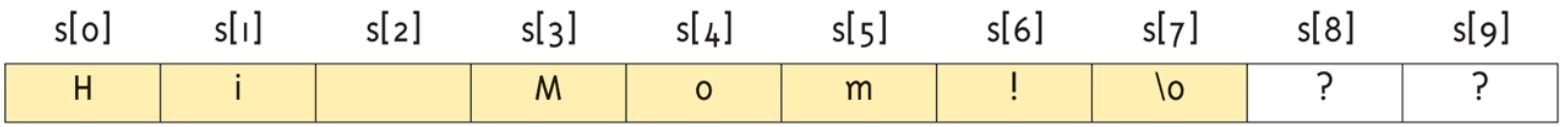
- Mảng với kiểu cơ sở là char
  - Mỗi biến đánh chỉ số là 1 kí tự
  - Thêm 1 kí tự: ‘\0’
    - Gọi là “kí tự null”
    - Đánh dấu kết thúc xâu
- Trong các ví dụ trước ta đã sử dụng xâu C
  - Hằng giá trị "Hello" được lưu dạng xâu C

Biến kiểu xâu C

- Mảng kí tự:
  char s[10];
  - Khai báo 1 biến kiểu xâu C có thể chứa tối đa 9 kí tự
  - + 1 kí tự null
- Thường là mảng chưa đầy
  - Khai báo nó đủ lớn để lưu xâu kích thước lớn nhất cần tới
  - Đánh dấu kết thúc bằng null
- Khác biệt duy nhất với mảng chuẩn:
  - Phải chứa kí tự null

Việc lưu trữ xâu C

- Một mảng chuẩn:
  char s[10];
  - Nếu s chứa xâu kí tự "Hi Mom!", nó được lưu như sau:



Khởi tạo xâu C

- Có thể khởi tạo xâu C:
  char myMessage[20] = "Hi there.";
  - Không cần điền đầy toàn bộ mảng
  - Bước khởi tạo đặt '\0' ở cuối
- Có thể bỏ qua kích thước mảng:
  char shortString[] = "abc";
  - Tự động đặt kích thước bằng chiều dài xâu trong ngoặc kép cộng 1
  - KHÔNG giống:
    char shortString[] = {'a', 'b', 'c'};

Chỉ số trong xâu C

- Một xâu C là một mảng
- Có thể truy cập tới các biến đánh chỉ số của xâu C.
  char ourString[5] = "Hi";
  - ourString[0] là ‘H’
  - ourString[1] là ‘i’
  - ourString[2] là ‘\0’
  - ourString[3] là không xác định
  - ourString[4] là không xác định

Thao tác dựa trên chỉ số của xâu C

- Có thể thao tác trên các biến đánh chỉ số
  char happyString[7] = "DoBeDo";
  happyString[6] = 'Z';
  - Hãy cẩn thận!
  - Ở đây '\0' (null) bị ghi đè bằng 'Z'!
- Nếu null bị ghi đè, xâu C không còn hoạt động như một xâu nữa!
  - Không lường được kết quả!

Thư viện

- Khai báo xâu C
  - Không cần thư viện C++ nào cả
  - Nó có sẵn trong C++
- Các thao tác
  - Cần thư viện <cstring>
  - Thường được khai báo khi dùng xâu C
    - Khi người viết chương trình muốn làm gì đó trên xâu C

= và == trên xâu C

- Biến kiểu xâu C không giống các biến khác
  - Không thể gán hay so sánh trực tiếp:
    char aString[10];
    aString = "Hello"; // KHÔNG HỢP LỆ!
    - Chỉ có thể dùng “=“ khi khai báo kết hợp khởi tạo xâu C!
- Phải dùng hàm thư viện cho phép gán:
  strcpy(aString, "Hello");
  - Hàm có sẵn (trong <cstring>)
  - Đặt giá trị của aString bằng "Hello"
  - Không kiểm tra kích thước!
    - Người viết chương trình phải kiểm soát. Giống như các thao tác khác trên mảng!

So sánh các xâu C

- Không thể dùng toán tử ==

```
char aString[10] = "Hello";
char anotherString[10] = "Goodbye";
```

  - `aString == anotherString;` // Không được phép!

- Phải dùng hàm thư viện:

```
if (strcmp(aString, anotherString))
        cout << "Strings NOT same.";
else
        cout << "Strings are same.";
```

Thư viện <cstring>:
Display 9.1 Một số hàm trên xâu C có sẵn trong <cstring> (1/2)

- Full of string manipulation functions

Display 9.1 Some Predefined C-String Functions in <cstring>

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

(continued)

Thư viện <cstring>:
Display 9.1 Một số hàm trên xâu C có sẵn trong <cstring> (2/2)

Display 9.1 Some Predefined C-String Functions in <cstring>

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

Các hàm trên xâu C: strlen()

- “Chiều dài xâu"
- Khi làm việc với xâu kí tự ta thường cần thông tin chiều dài xâu:

```
char myString[10] = "dobedo";
cout << strlen(myString);
```

  - Trả về số lượng kí tự
    - Không tính null
  - Với ví dụ trên:

6

Các hàm trên xâu C: strcat()

- strcat()
- “Nối xâu":
  ```
  char stringVar[20] = "The rain";
  strcat(stringVar, "in Spain");
  ```
  - Kết quả:
    stringVar chứa "The rainin Spain"
  - Hãy cẩn thận. Bổ sung dấu cách nếu cần

Đối số và tham số kiểu xâu C

- Nhắc lại: xâu C là mảng
- Vì vậy tham số kiểu xâu C là tham số kiểu mảng
  - Xâu C truyền vào hàm có thể bị hàm biến đổi
- Cũng như với mảng, thường thì ta truyền thêm tham số kích thước
  - Hàm cũng có thể dùng '\0' để tìm điểm kết thúc xâu
  - Do đó kích thước không thực sự cần thiết nếu hàm không biến đổi xâu
  - Dùng từ khóa const để bảo vệ đối số kiểu xâu C

Ghi xâu C

- Ta có thể ghi xâu C ra thiết bị xuất (màn hình) dùng toán tử chèn <<
- Thật ra ta đã làm việc này rồi:
  cout << news << " Wow.\n";
  - Trong đó news là một biến kiểu xâu C
- Có thể là do toán tử << đã được nạp chồng cho xâu C!

Đọc vào xâu C

- Có thể đọc dữ liệu từ thiết bị nhập (bàn phím) vào xâu C dùng toán tử trích >>
  - Tuy nhiên, có vấn đề nảy sinh
- Dấu trắng được xem là "kí tự phân cách" (delimiter)
  - Tab, cách, xuống dòng bị bỏ qua
  - Việc đọc dừng khi gặp kí tự phân cách
- Hãy chú ý kích thước của xâu C
    - Phải đủ lớn để chứa xâu nhập vào
    - C++ không cảnh báo về vấn đề này

Ví dụ đọc vào xâu C

- char a[80], b[80];
  cout << "Enter input: ";
  cin >> a >> b;
  cout << a << b << "END OF OUTPUT\n";
- Kết quả thực thi:

  Enter input: Do be do to you!
  DobeEND OF OUTPUT
  - Lưu ý: phần gạch dưới được nhập từ bàn phím
- Xâu C a đọc vào "do"
- Xâu C b đọc vào "be"

Đọc một dòng vào xâu C

- Có thể lấy hết dòng kí tự người dùng gõ từ bàn phím vào xâu C
- Dùng hàm getline() có sẵn:

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

  - Kết quả thực thi:

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

Ví dụ: đối số dòng lệnh

- Chương trình được gọi từ dòng lệnh (ví dụ: UNIX shell, nơi nhắc lệnh DOS) có thể nhận đối số
  - Ví dụ: COPY C:\FOO.TXT D:\FOO2.TXT
    - Lệnh này chạy chương trình có tên là “COPY” và truyền vào 2 tham số kiểu xâu C “C:\FOO.TXT” và “D:\FOO2.TXT”
    - Việc xử lý đầu vào diễn ra bên trong chương trình COPY (với ví dụ này là sao tệp có tên chỉ định)
- Đối số được truyền vào hàm main dưới dạng mảng các xâu C

Ví dụ: đối số dòng lệnh

- Dòng đầu của main
  - int main(int argc, char *argv[])
  - argc xác định số lượng đối số truyền vào. Nó tính cả tên của chương trình. Do đó argc luôn ≥ 1.
  - argv là một mảng các xâu C.
    - argv[0] lưu tên của chương trình được gọi
    - argv[1] lưu tham số thứ nhất
    - argv[2] lưu tham số thứ hai
    - v.v.

Example: Command Line Arguments

```
// Test.cpp
// In ra màn hình các đối số truyền qua dòng lệnh
int main(int argc, char* argv[]){
    for(int i = 0; i < argc; i++){
        cout << "Doi so " << i << ": " << argv[i] << endl;
    }
    return 0;
}
```

Ví dụ gọi chương trình lần 1

```
> Test
Doi so 0: Test
```

Gọi chương trình Test từ nơi nhắc lệnh

Ví dụ gọi chương trình lần 2

```
> Test hello world
Doi so 0: Test
Doi so 1: hello
Doi so 2: world
```

Bàn thêm về getline()

- Có thể chỉ định tường minh kích thước đọc vào:

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

  - Kết quả thực thi:

```
Enter input: dobedowap
dobeEND OF OUTPUT
```

  - Bắt buộc chỉ đọc vào 4 kí tự
    - Kí tự cuối là null được bổ sung tự động

Đọc/ghi kí tự

- Dữ liệu đọc vào và ghi ra
  - Tất cả đều được xử lý dạng kí tự
  - Ví dụ: số 10 ghi ra là ‘1’ và ‘0’
  - Việc chuyển đổi được thực hiện tự động
    - Dùng các tiện ích bậc thấp
- Ta cũng có thể dùng các tiện ích bậc thấp này

Hàm thành viên get()

- Đọc từng kí tự một
- Hàm thành viên của đối tượng cin:
  char nextSymbol;
  cin.get(nextSymbol);
  - Đọc kí tự tiếp theo và đưa vào biến nextSymbol
  - Đối số phải có kiểu char
    - Không thể là xâu

Hàm thành viên put()

- Ghi từng kí tự một ra thiết bị xuất
- Hàm thành viên của đối tượng cout:
  cout.put('a');
  - In chữ cái 'a' ra màn hình

  char myString[10] = "Hello";
  cout.put(myString[1]);
  - In chữ cái 'e' ra màn hình

Các hàm thành viên khác

- putback()
  - Sau khi lấy dữ liệu từ luồng vào biến, có thể ta cần trả nó lại cho luồng
  - cin.putback(lastChar);
- peek()
  - Trả về kí tự tiếp theo trong luồng, nhưng vẫn để nó lại luồng
  - peekChar = cin.peek();
- ignore()
  - Bỏ qua luồng nhập tới khi nào gặp kí tự chỉ định
  - cin.ignore(1000, '\n');
    - Bỏ qua tối đa 1000 kí tự tới khi gặp '\n'

Ví dụ: Chương trình dùng hàm putback

```
// Nguồn: http://cplusplus.com/reference/iostream/istream/putback/
// istream putback
#include <iostream>
using namespace std;
int main(){
    char c;
    int n;
    char str[256];
    cout << "Nhap vao 1 so nguyen duong hoac 1 tu: ";

    cin.get(c);
    if((c >= '0') && (c <= '9')){
        cin.putback(c);
        cin >> n;
        cout << "Ban da nhap vao so " << n << endl;
    }else{
        cin.putback(c);
        cin >> str;
        cout << "Ban da nhap vao tu " << str << endl;
    }

    cin.ignore(80, '\n');
    cin.get();
    return 0;
}
```

Các hàm thao tác với kí tự:
Display 9.3 Một số hàm trong <cctype> (1/3)

Display 9.3 Some Functions in <cctype>

FUNCTION	DESCRIPTION	EXAMPLE
`toupper(`*Char_Exp*`)`	Returns the uppercase version of *Char_Exp* (as a value of type `int`).	`char c = toupper('a');` `cout << c;` **Outputs:** A
`tolower(`*Char_Exp*`)`	Returns the lowercase version of *Char_Exp* (as a value of type `int`).	`char c = tolower('A');` `cout << c;` **Outputs:** *a*
`isupper(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is an uppercase letter; otherwise, returns `false`.	`if (isupper(c))` `  cout << "Is uppercase.";` `else` `  cout << "Is not uppercase.";`

Các hàm thao tác với kí tự:
Display 9.3 Một số hàm trong <cctype> (2/3)

Display 9.3 Some Functions in <cctype>

FUNCTION	DESCRIPTION	EXAMPLE
`islower(Char_Exp)`	Returns `true` provided *Char_Exp* is a lowercase letter; otherwise, returns `false`.	`char c = 'a';` `if (islower(c))` `  cout << c << " is lowercase.";` **Outputs:** a is lowercase.
`isalpha(Char_Exp)`	Returns `true` provided *Char_Exp* is a letter of the alphabet; otherwise, returns `false`.	`char c = '$';` `if (isalpha(c))` `  cout << "Is a letter.";` `else` `  cout << "Is not a letter.";` **Outputs:** Is not a letter.
`isdigit(Char_Exp)`	Returns `true` provided *Char_Exp* is one of the digits `'0'` through `'9'`; otherwise, returns `false`.	`if (isdigit('3'))` `  cout << "It's a digit.";` `else` `  cout << "It's not a digit.";` **Outputs:** It's a digit.
`isalnum(Char_Exp)`	Returns `true` provided *Char_Exp* is either a letter or a digit; otherwise, returns `false`.	`if (isalnum('3') && isalnum('a'))` `  cout << "Both alphanumeric.";` `else` `  cout << "One or more are not.";` **Outputs:** Both alphanumeric.

Các hàm thao tác với kí tự:
Display 9.3 Một số hàm trong <cctype> (3/3)

`isspace(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is a whitespace character, such as the blank or newline character; otherwise, returns `false`.	`//Skips over one "word" and sets c` `//equal to the first whitespace` `//character after the "word":` `do` `{` `  cin.get(c);` `} while (! isspace(c));`
`ispunct(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is a printing character other than whitespace, a digit, or a letter; otherwise, returns `false`.	`if (ispunct('?'))` `  cout << "Is punctuation.";` `else` `  cout << "Not punctuation.";`
`isprint(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is a printing character; otherwise, returns `false`.	
`isgraph(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is a printing character other than whitespace; otherwise, returns `false`.	
`isctrl(`*Char_Exp*`)`	Returns `true` provided *Char_Exp* is a control character; otherwise, returns `false`.	

Lớp chuẩn string

- Đã định nghĩa trong thư viện:
  ```
  #include <string>
  using namespace std;
  ```
- Biến và biểu thức string
  - Xử lý như với các kiểu đơn
- Có thể gán, so sánh, cộng:
  ```
  string s1, s2, s3;
  s3 = s1 + s2;          // Nối
  s3 = "Hello Mom!"      // Gán
  ```
  - Lưu ý là xâu C "Hello Mom!" được chuyển tự động sang kiểu string!

Display 9.4
Chương trình dùng lớp 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!
```

Đọc/ghi với lớp string

- Giống như các kiểu khác!
- string s1, s2;
  cin >> s1;
  cin >> s2;
- Kết quả thực thi:
  Người dùng gõ vào:
  May the hair on your toes grow long and curly!
- Toán tử trích bỏ qua dấu trắng:
  s1 đọc vào giá trị "May"
  s2 đọc vào giá trị "the"

getline() với lớp string

- Để đọc hết dòng người dùng gõ vào:

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

- Kết quả thực thi:

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

  - Tương tự như cách dùng getline() cho xâu C

Phiên bản getline() khác

- Có thể chỉ định kí tự phân cách:
  ```
  string line;
  cout << "Enter input: ";
  getline(cin, line, '?');
  ```
  - Nhận dữ liệu tới khi gặp dấu '?'
- getline() trả về tham chiếu
  - ```
    string s1, s2;
    getline(cin, s1) >> s2;
    ```
  - Cho kết quả là: (cin) >> s2;
    ```

Lỗi thường gặp: Kết hợp các phương pháp đọc

- Hãy cẩn thận khi kết hợp cin >> var với getline
  - int n;
    string line;
    cin >> n;
    getline(cin, line);
  - Nếu nhập vào: 42
    Hello hitchhiker.
    - Biến n sẽ được đặt bằng 42
    - line đặt bằng xâu rỗng!
  - cin >> n bỏ qua dấu trắng, để '\n' lại trong luồng cho getline()!

Xử lý xâu với lớp string

- Có các phép toán như cho xâu C
- Và hơn thế nữa!
  - Hơn 100 thành viên của lớp string chuẩn
- Một số hàm thành viên:
  - .length()
    - Trả về độ dài của biến string
  - .at(i)
    - Trả về tham chiếu tới kí tự ở vị trí i

Display 9.7 Hàm thành viên của lớp string chuẩn (1/2)

Display 9.7 Member Functions of the Standard Class `string`

EXAMPLE	REMARKS
Constructors	
`string str;`	Default constructor; creates empty `string` object `str`.
`string str("string");`	Creates a `string` object with data `"string"`.
`string str(aString);`	Creates a `string` object `str` that is a copy of `aString`. `aString` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/Modifiers	
`str1 = str2;`	Allocates space and initializes it to `str2`'s data, releases memory allocated for `str1`, and sets `str1`'s size to that of `str2`.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately.
`str.empty( )`	Returns `true` if `str` is an empty `string`; returns `false` otherwise.

(continued)

Display 9.7 Hàm thành viên của lớp string chuẩn (1/2)

Display 9.7 Member Functions of the Standard Class `string`

EXAMPLE	REMARKS
`str1 + str2`	Returns a `string` that has `str2`'s data concatenated to the end of `str1`'s data. The size is set appropriately.
`str.insert(pos, str2)`	Inserts `str2` into `str` beginning at position `pos`.
`str.remove(pos, length)`	Removes substring of size `length`, starting at position `pos`.
Comparisons	
`str1 == str2  str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2  str1 > str2`	Four comparisons. All are lexicographical comparisons.
`str1 <= str2  str1 >= str2`	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`.
`str.find_first_not_of (str1, pos)`	Returns the index of the first instance in `str` of any character *not* in `str1`, starting search at position `pos`.

Chuyển đổi giữa xâu C và đối tượng string

- Chuyển đổi kiểu tự động
  - Từ xâu C thành đối tượng string:
    char aCString[] = "My C-string";
    string stringVar;
    stringVar = aCstring;
    - Hoàn toàn hợp lệ và đúng!
  - aCString = stringVar;
    - Không hợp lệ!
    - Không thể tự động chuyển thành xâu C
  - Phải dùng lệnh chuyển đổi tường minh:
    strcpy(aCString, stringVar.c_str());

Tóm tắt

- Biến kiểu xâu C là “mảng kí tự”
  - Với kí tự null (‘\0’) bổ sung ở cuối
- Xâu C hoạt động như mảng
  - Không thể gán, so sánh như các biến đơn
- Các thư viện <cctype> & <string> có nhiều hàm xử lý giúp xử lý xâu dễ dàng
- cin.get() đọc kí tự tiếp theo trong luồng
- getline() đọc hết 1 dòng trong luồng
- Thao tác với đối tượng string tiện lợi hơn xâu C

Chuẩn bị bài tới

- Đọc chương 10 giáo trình: Con trỏ và Mảng động