Ngôn ngữ lập trình

Bài 7:
Khuôn mẫu (Template) và
Thư viện chuẩn (STL)

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Bộ Môn Công Nghệ Phần Mềm – Khoa CNTT
Trường Đại Học Thủy Lợi

Nội dung

1. Nhắc lại về vector
2. C-string và lớp String
3. Khuôn mẫu hàm
4. Khuôn mẫu lớp

Bài giảng có sử dụng hình vẽ trong cuốn sách "*Absolute C++*. W. Savitch, Addison Wesley, 2002"

1. Nhắc lại về vector

MỘT KHUÔN MẪU LỚP (CLASS TEMPLATE)

Cơ bản về vector

- Dùng để lưu trữ tập dữ liệu CÙNG KIỂU, giống mảng,
- Nhưng vector có thể phình to hoặc thu nhỏ kích thước trong lúc chạy chương trình (không giống như mảng có kích thước cố định)
- Thư viện: #include <vector>
- Ví dụ khai báo
  - *vector<int> vIA;* // Khai báo một vector chứa dữ liệu kiểu int
  - *vector<int> vIB (10);* // Khai báo một vector có kích thước ban đầu là 10, chứa dữ liệu kiểu int
  - *vector<int> vIC (10, 2);* // Khai báo một vector có kích thước ban đầu là 10, chứa dữ liệu kiểu int và dữ liệu được khởi tạo giá trị 2

Một số hàm thành viên của vector

Phương thức	Mục đích
v.assign(n,e)	Gán tập giá trị mới cho vector, thay thế nội dung hiện tại của nó đồng thời thay đổi kích thước
v[i] hoặc v.at[i]	Tham chiếu đến phần tử thứ i của vector
v.clear()	Làm rỗng vector
v.pop_back()	Xóa phần tử cuối cùng của vector
v.push_back(e)	Thêm phần tử e vào cuối của vector
v.resize(new_size)	Thay đổi kích thước của vector
…	

Danh sách đầy đủ có thểm xem tại đây

Sử dụng iterator

- Trong lập trình hướng đối tượng (OOP), một iterator là một đối tượng cho phép lập trình viên duyệt qua (traverse) các phần tử trong một container, như danh sách (list), mảng, vector …

```
vector<int>::iterator it = v.begin();
while(it !=  v.end())
{
   cout << *it << "   ";
   it++;

}
```

2. C-string và lớp string

Mục tiêu

- C-Strings: một kiểu mảng cho chuỗi ký tự
- Các công cụ thao tác ký tự (char)
  - Character I/O, cin
  - Hàm thành viên: *get, put*
  - Một số hàm khác: *pushback, peek, ignore* …
- Lớp String chuẩn
  - Xử lý chuỗi ký tự với lớp String

Hai cách biểu diễn chuỗi (string)

- C-strings
  - Một mảng với các phần tử có kiểu cơ sở *char*
  - Chuỗi được kết thúc với kí tự null,"\0"
  - Là phương thức cũ được kế thừa từ C
- Lớp String
  - Sử dụng khuôn mẫu (template)

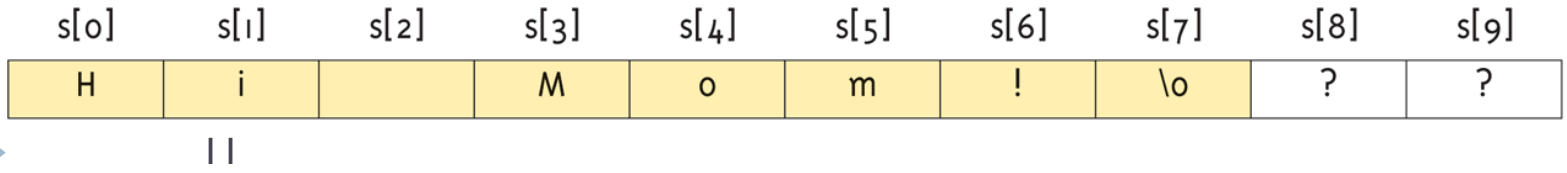
C-strings

- Một mảng các phần tử với kiểu cơ sở *char*
  - Mỗi phần tử của mảng là một ký tự
  - Ký tự mở rộng "\0"
    - Được gọi là ký tự rỗng (null character)
    - Là dấu hiệu kết thúc một chuỗi ký tự
- Chúng ta đã sử dụng C-strings!
  - Ví dụ: literal "Hello" được lưu trữ như một c-string

Biến c-string

- Khai báo: *char s[10]*
  - Khai báo một biến c-string để lưu trữ 9 ký tự
  - Và kí tự thứ 10 là ký tự *null* ("\0")
- Chỉ có một điểm khác với mảng chuẩn:
  - C-strings phải chứa ký tự *null* !
- Khởi tạo một c-string: *char s[10] = "Hi Mom!"*
  - Không cần thiết phải điền đầy đủ (kích thước) mảng
  - Đặt ký tự "\0" ở cuối
- Có thể bỏ qua kích thước mảng:

  char shortString[] = "abc";



Thao tác với c-string qua chỉ số

- Một c-string LÀ một mảng => có thể truy cập thành viên thông qua chỉ số (index)
- Ví dụ: *char ourString[5] = "Hi";*
  - ourString[0] là "H“
  - ourString[1] là "i“
  - ourString[2] là "\0“
  - ourString[3] là không xác định (unknown)
  - ourString[4] là không xác định (unknown)
- Chú ý: nếu thực hiện phép gán *ourString[2] = “a”;*
  - Ghi đè ký tự “\0” (null) bởi ký tự “a”
- Nếu ký tự *null* bị ghi đè, c-string không còn hoạt động như c-string nữa! => kết quả không dự đoán được

Toán tử = và == với c-strings

- C-strings không giống những biến khác
  - Không thể sử dụng phép gán hoặc so sánh
  - Chỉ có thể sử dụng toán tử "=" lúc khởi tạo một c-string!

```
char aString[10];
aString = "Hello";      // KHÔNG HỢP LỆ
```

- Phải sử dụng hàm thư viện cho phép gán: *strcpy(aString, "Hello");*
  - Một hàm được xây dựng sẵn trong **<cstring>**
  - Đặt giá trị của aString bằng với "Hello"
  - KHÔNG kiểm tra kích thước!

So sánh c-strings

- Không thể sử dụng toán tử "==" để so sánh c-strings

```
char aString[10] = "Hello";
char anotherString[10] = "Goodbye";
aString == anotherString;                    // KHÔNG hợp lệ
```

- Phải sử dụng thư viện hàm:

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

Danh sách hàm thao tác chuỗi trong <cstring> (1/2)

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)

Danh sách hàm thao tác chuỗi 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++.

Hàm STRLEN()

- "STRing LENgth" – độ dài của chuỗi
- Trả về số lượng ký tự
  - Không bao gồm ký tự null
- Ví dụ:

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

  - Giá trị trả về: 6

Hàm strcat()

- "STRing ConcATnate"
- Dùng để nối chuỗi

```
char stringVar[20] =  "The rain";
strcat(stringVar," in Spain");
```

  - Kết quả: *stringVar* bây giờ là "The rain in Spain "

Đối số và tham số c-string

- Nhớ lại: c-string là một mảng
- Vì vậy có thể dùng c-string làm tham số mảng
  - c-string được truyền vào hàm có thể bị thay đổi bởi hàm tiếp nhận!
- Giống như mảng, thông thường cũng truyền cả kích thước của c-string vào hàm
  - Hàm cũng "có thể" sử dụng kí tự "\0" để kiểm tra kích thước
  - Do đó tham số kích thước có thể không cần nếu hàm không thay đổi tham số c-string
  - Sử dung "*const*" để bảo vệ những đối số c-string không bị thay đổi

I/O với C-string

- Xuất dữ liệu với toán tử chèn: <<
  - Do toán tử << đã được nạp chồng cho c-strings!
- Nhập dữ liệu với toán tử: >>
- Chú ý khi nhập dữ liệu: khoảng trắng (whitespace) được dùng để phân cách (delimiter)
  - Tab, space, ngắt dòng (line breaks) bị bỏ qua
  - Dữ liệu đọc vào sẽ dừng ghi bắt gặp delimiter
- Phải ước lượng kích thước c-string đủ lớn để chứa toàn bộ chuỗi, C++ không đưa ra bất kỳ cảnh bảo nào cho các tình huống vượt kích thước!

Ví dụ
nhập dữ liệu cho c-string dùng cin

```
char a[80], b[80];
cout << "Enter input: ";
cin >> a >> b;
cout << a << b << "END OF OUTPUT\n";
```

Nhập vào: Do be do to you!

Kết quả in ra màn hình: DobeEND OF OUTPUT

C-string a nhận giá trị "do"

C-string b nhận giá trị "be"

Nhập dữ liệu cho C-string dùng hàm getline

- Có thể nhận vào cả một dòng cho c-string sử dụng hàm định nghĩa sẵn *getline()*

```
char a[80];
cout << "Enter input: ";
cin.getline(a,80);// chiều dài chuỗi muốn nhập vào là 79?
cout << a << "END OF OUTPUT\n";
```

Nhập vào: Do be do to you!

Kết quả in ra màn hình:

Do be do to you! END OF OUTPUT

Hàm thành viên get()

- Đọc một ký tự một lần
- Là hàm thành viên của đối tượng cin

  char nextSymbol;
  cin.get(nextSymbol);
  - Đọc ký tự tiếp theo và gán cho biến *nextSymbol*
  - Đối số phải là kiểu char, không phải chuỗi !

Hàm thành viên put()

- Hiển thị một ký tự một lần
- Là hàm thành viên của đối tượng cout
- Ví dụ:
  - *cout.put("a"); //* output kí tự "a" ra màn hình
  - *char myString[10] = "Hello";*
    cout.put(myString[1]); // Hiển thị ký tự "e" ra màn hình

Một vài hàm thành viên khác

- putback()
  - Giảm vị trí hiện tại trong stream lùi về một ký tự
  - *cin.putback(lastChar);*
- peek()
  - Trả về ký tự tiếp theo, nhưng không loại bỏ nó khỏi luồng input
  - *peekChar = cin.peek();*
- ignore()
  - Bỏ qua input, cho đến khi gặp ký tự được chỉ định
  - *cin.ignore(1000,"\n");* // bỏ qua nhiều nhất 1000 kí tự cho đến khi gặp "\n"

Danh sách Hàm thao tác ký tự trong thư viện <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.";`

Danh sách Hàm thao tác ký tự trong thư viện <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.

Danh sách Hàm thao tác ký tự trong thư viện <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 string chuẩn

- Được định nghĩa trong thư viện <string>

  #include <string>
  using namespace std;

- Biến string và các biểu thức được xử lý giống như những kiểu đơn giản khác
- Có thể gán, so sánh, cộng

  string s1, s2, s3;
  s3 = s1 + s2; //Concatenation
  s3 = "Hello Mom!" //Assignment

  - Lưu ý: c-string "Hello Mom!" được tự động chuyển thành kiểu string!

Chương trình với 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!
```

I/O với lớp string

- Giống như những kiểu khác!
- *string s1, s2;*
  cin >> s1;
  cin >> s2;

Nhập vào:

May the hair on your toes grow long and curly!

s1 nhận giá trị "May"

s2 nhận giá trị "the"

  - Bỏ qua các khoảng trắng (whitespace)

Hàm getline() với lớp string

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

Nhập vào: Do be do to you!

Kết quả in ra màn hình:

Do be do to you! END OF OUTPUT

```
string line;
cout << "Enter input: ";
getline(cin,line,"?");                    // nhập vào các ký tự cho đến khi gặp "?"
```

Câu hỏi

- *int n;*
  string line;
  cin >> n;
  getline(cin,line);
- Nếu nhập vào

  42

  Hello hitchhiker.
- Hai biến n và line có giá trị là gì?
  - Biến n được gán giá trị 42
  - Biến line được một chuỗi rỗng
- Tại sao?
  - cin >> n bỏ qua leading whitespace, để lại ký tự "\n" trên stream cho hàm getline()!

Hàm Xử lý của lớp string

- Có một số hàm giống như c-strings
- Và còn nhiều hơn!
  - Trên 100 hàm thành viên của lớp string chuẩn
- Một vài hàm thành viên
  - .length(): trả về chiều dài của biến string
  - .at(i): trả về tham chiếu tới ký tự ở vị trí i

Danh sách hàm thành viên của lớp string (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)

Danh sách hàm thành viên của lớp string (2/2)

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 c-string và đối tượng của lớp string

- Tự động chuyển kiểu
  - Từ c-string thành đối tượng của lớp string

```
char aCString[] = "My C-string";
string stringVar;
stringVar = aCstring; // Hợp lệ!
```

- Nhưng không thể viết

```
aCString = stringVar; // KHÔNG hợp lệ!
```

  - Không thể tự động chuyển từ đối tượng của lớp string sang c-string
  - Phải sử dụng chuyển tường minh bằng hàm strcpy

```
strcpy(aCString,stringVar.c_str());
```

Tóm tắt c-string và lớp string

- Biến c-string là một mảng các ký tự
  - Cộng thêm ký tự null, "\0"
- C-strings hoạt động giống như mảng
  - Không thể gán, so sánh giống như những biến đơn giản
- Các thư viện <cctype> và <string> chứa nhiều hàm thao tác hữu ích
- cin.get() đọc ký tự đơn tiếp theo
- getline() cho phép đọc toàn dòng
- Đối tượng của lớp string thao tác tốt hơn c-strings

3. Khuôn mẫu

Templates

Mục tiêu

- Khuôn mẫu hàm (Function Templates)
- Khuôn mẫu lớp (Class Templates)
- Khuôn mẫu và Kế thừa
- Thư viện khuôn mẫu chuẩn (STL)

Khuôn mẫu hàm

- Một mô hình (một mẫu) giúp tạo định nghĩa chung cho những hàm CHỈ khác nhau về kiểu dữ liệu mà chúng thao tác.
  - Đây là một hàm chung cho những hàm đó
  - Thích hợp cho những hàm thực thi cùng một tác vụ nhưng với những tham số khác nhau
- Khuôn mẫu hàm tốt hơn so với nạp chồng hàm bởi vì đoạn mã định nghĩa thao tác trong hàm chỉ cần được viết MỘT LẦN

Ví dụ về khuôn mẫu hàm (1/2)

- Giả sử chúng ta có hai hàm sau với mục đích hoán vị giá trị của hai biến
- Hai hàm này chỉ khác nhau về kiểu dữ liệu tham số (kiểu *int* và *char*)

```
void swap(int &x, int &y)
  { int temp = x; x = y;
    y = temp;
  }

void swap(char &x, char &y)
  { char temp = x; x = y;
    y = temp;
  }
```

Ví dụ về khuôn mẫu hàm (2/2)

- Hai hàm này có thể được thay thế bởi MỘT khuôn mẫu hàm sau

```
template<class T>
void swap(T &x,T &y)
 {
   T temp = x; x = y;
    y = temp;
 }
```

Sử dụng khuôn mẫu hàm

- Khi gọi một khuôn mẫu hàm với một kiểu dữ liệu, trình biên dịch sẽ tạo một định nghĩa hàm thực sự từ khuôn mẫu này dựa theo kiểu dữ liệu của tham số

  int i = 1, j = 2;

  swap(i,j);

- Đoạn mã trên sẽ khiến trình biên dịch khởi tạo khuôn mẫu hàm với kiểu dữ liệu ***int*** thay thế cho kiểu tham số ***T***

Bài tập cho khuôn mẫu hàm

- Viết một khuôn mẫu hàm tìm kiếm một phần tử trong một mảng và in ra vị trí của phần tử đó trong mảng nếu tìm thấy, ngược lại in ra -1
- *template<class T>*
  int search(const T a[], ~~int numberUsed~~, T target)
  { … }

Một vài lưu ý cho khuôn mẫu hàm

- Khuôn mẫu hàm không sử dụng bộ nhớ
- Mã thực sự chỉ được tạo khi tên khuôn mẫu được gọi
- Khi truyền một đối tượng của lớp cho một khuôn mẫu hàm, phải đảm bảo rằng mọi toán tử được chỉ định trong khuôn mẫu đã được định nghĩa hoặc nạp chồng trong định nghĩa của lớp
- Mọi kiểu dữ liệu chỉ định trong khuôn mẫu hàm phải được dùng bên trong thân của khuôn mẫu hàm
- Lời gọi hàm phải truyền đầy đủ tham số (với kiểu dữ liệu) được chỉ định trong khuôn mẫu hàm
- Khuôn mẫu hàm có thể được nạp chồng – với danh sách tham số khác nhau
- Giống như các hàm thông thường, khuôn mẫu hàm phải được định nghĩa trước khi gọi

Khuôn mẫu lớp

- Có thể định nghĩa khuôn mẫu cho lớp. Những lớp kiểu này định nghĩa những kiểu dữ liệu trừu tượng
- Không giống như khuôn mẫu hàm, một khuôn mẫu lớp được khởi tạo bằng cách cung cấp cụ thể kiểu dữ liệu (ví dụ: *int*, *float*, *string*, …) khi định nghĩa đối tượng

Ví dụ về khuôn mẫu lớp (1/2)

- Xem xét hai lớp sau
- Một lớp để cộng hai số nguyên

```
class Joiner
{
        public:
                int combine(int x, int y)
                        {return x + y;}
};
```

- Một lớp để nối hai chuỗi

```
class Joiner
{
        public:
                string combine(string x, string y)
                        {return x + y;}
};
```

Ví dụ về khuôn mẫu lớp (2/2)

- Hai lớp trên có thể được thay thế bởi CHỈ một khuôn mẫu lớp sau

```
template <class T>
  class Joiner
  {
        public:
              T combine(T x,T y)
                    {return x + y;}
  };
```

Sử dụng khuôn mẫu lớp

```
Joiner<double> jd;
Joiner<string> sd;
cout << jd.combine(3.0, 5.0);
cout << sd.combine("Hi ", "Ho");
```

Kết quả in ra màn hình: *8.0* và *Hi Ho*

Bài tập khuôn mẫu lớp

- Cài đặt giao diện lớp sau

```
1   //Class for a pair of values of type T:
2   template<class T>
3   class Pair
4   {
5   public:
6       Pair( );
7       Pair(T firstValue, T secondValue);
8       void setFirst(T newValue);
9       void setSecond(T newValue);
10      T getFirst( ) const;
11      T getSecond( ) const;
12  private:
13      T first;
14      T second;
15  };
```

Khuôn mẫu lớp và kế thừa

- Khuôn mẫu có thể được kết hợp với kế thừa
- Chúng ta có thể:
  - Kế thừa một lớp thông thường từ một khuôn mẫu lớp
  - Kế thừa một khuôn mẫu lớp từ một khuôn mẫu lớp khác

Thư viện khuôn mẫu chuẩn

- STL – Standard Template Libray
  - Một thư viện bao gồm những khuôn mẫu được sử dụng thường xuyên cho cấu trúc dữ liệu và thuật toán (algorithms)
- Chương trình có thể được phát triển nhanh hơn nếu chúng ta sử dụng những khuôn mẫu sẵn có này
- Hai kiểu cấu trúc dữ liệu quan trọng trong STL
  - **Bộ chứa (container)**: những lớp lưu trữ dữ liệu và
  - **Bộ lặp (iterator)**: giống con trỏ, cung cấp cơ chế để truy cập các thành viên trong một container

Bộ chứa (Container)

- Có hai kiểu bộ chứa (container) trong STL
  - Bộ chứa tuần tự (sequential containers): tổ chức và truy xuất dữ liệu một cách tuần tự, giống như kiểu mảng. Bao gồm: ***vector***, ***dequeue*** và ***list***
  - Bộ chứa liên kết (associative containers): sử dụng key để cho phép các phần tử có thể được truy cập một cách nhanh chóng. Bao gồm: ***set***, ***multiset***, ***map*** và ***multimap***

Tạo đối tượng container

- Tạo một danh sách (list) của kiểu int

  list<int> mylist;

- Tạo một vector của những đối tượng string:

  vector<string> myvector;

Bộ lặp (Iterator)

- Tổng quát hóa khái niệm con trỏ (pointer), được sử dụng để truy xuất thông tin trong bộ chứa (container)
- Có nhiều loại lặp:
  - Lặp tiến (forward) : sử dụng toán tử ++
  - Lặp hai chiều (bidirectional): sử dụng ++ và –
  - Truy cập ngẫu nhiên (random-access)
  - Input: có thể sử dụng với đối tượng *cin* và *istream*
  - Output: có thể sử dụng với đối tượng *cout* và *ostream*

Container và iterator

- Mỗi lớp container định nghĩa:
  - Một kiểu iterator, sử dụng để truy xuất các thành viên của nó
  - Những hàm trả về iterator
    - begin(): đặt iterator vào phần tử đầu tiên
    - end(): đặt iterator vào phần tử cuối cùng
- Iterator hỗ trợ các thao tác giống con trỏ (**iter*, *iter* ++, *iter* --)
- Kiểu của một iterator được quyết định bởi kiểu của container

```
list<int>::iterator x;
list<string>::iterator y;
```

Duyệt qua một container

- Xét một vector

```
vector<int> v;
 for  (int k=1; k<= 5; k++)
   v.push_back(k*k);
```

- Duyệt qua vector này sử dụng iterator

```
vector<int>::iterator iter = v.begin();
 while (iter != v.end())
   { cout << *iter << " "; iter++}
```

Kết quả in ra màn hình: **1 4 9 16 25**

Giải thuật

- STL bao gồm một số giải thuật được cài đặt như những khuôn mẫu hàm thực thi trên các containers
- Yêu cầu khai báo file tiêu đề "*algorithm*" (*#include <algorithm>*)
- Tập hợp các giải thuật bao gồm
  - **binary_search**
  - **for_each**
  - **max_element, min_element**
  - **random_shuffle**
  - **find**
  - **sort**
  - …

Sử dụng giải thuật trong stl

- **max_element(iter1, iter2)**: tìm phần tử lớn nhất trong một khoảng giới hạn bởi iter1 và iter2 của container
- **min_element(iter1, iter2)**: tương tự với phần tử nhỏ nhất
- **random_shuffle(iter1, iter2**): đảo ngẫu nhiên các giá trị trong khoảng giới hạn bởi iter1 và iter2
- **sort(iter1, iter2**): sắp xếp theo giá trị tăng dần của khoảng giới hạn bởi iter1 và iter2

Giáo trình Tham khảo

- **Giáo trình chính: W. Savitch, *Absolute C++*, Addison Wesley, 2002**
- Tham khảo:
  - A. Ford and T. Teorey, *Practical Debugging in C++*, Prentice Hall, 2002
  - Nguyễn Thanh Thủy, *Kĩ thuật lập trình C++*, NXB Khoa học và Kĩ Thuật, 2006