Chapter 16: Khuôn mẫu và thư viện ngoại lệ chuẩn (STL)

Topics

1. Hàm khuôn mẫu
2. Lớp khuôn mẫu và sự kế thừa
3. Giới thiệu về thư viện khuôn mẫu chuẩn (Standard Template Library – STL)

16.1 Hàm khuôn mẫu

- Hàm khuôn mẫu: Cách để tạo ra cùng một lúc nhiều định nghĩa cho các hàm chỉ khác nhau mỗi kiểu dữ liệu xử lý – một hàm khuôn mẫu hơi giống như một hàm tổng quát.

- Nếu làm được, viết hàm khuôn mẫu sẽ tốt hơn viết hàm nạp chồng vì chỉ cần viết mã một lần.

Ví dụ

Hai hàm khác nhau duy nhất ở kiểu dữ liệu

```
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;
 }
```

Khuôn mẫu **swap**

Có thể biểu diễn logic của cả hai hàm bằng một hàm khuôn mẫu tổng quát như sau:

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

Sử dụng hàm khuôn mẫu

- Khi một hàm khuôn mẫu được gọi, trình biên dịch đầu tiên tạo ra một định nghĩa ứng với một kiểu dữ liệu cụ thể, kiểu này được suy đoán từ đối số trong lời gọi hàm:

```
int i = 1, j = 2;
swap(i,j);
swap<int>(i,j);
```

- Đoạn mã trên giúp trình biên dịch tạo ra một hiện thể của khuôn mẫu – là một hàm cụ thể, có kiểu `int` được thế vào chỗ của `T`

Lưu ý về hàm khuôn mẫu

- Một hàm khuôn mẫu chỉ là một khuôn mẫu, chưa phải một hàm. Vì thế không thực có mã nào được tạo ra cho đến khi hàm được gọi.
- Nên nó không cần cấp phát bộ nhớ
- Khi truyền một đối tượng lớp nào đó vào một khuôn mẫu, đảm bảo rằng mọi toán tử xuất hiện trong khuôn mẫu đều đã được định nghĩa hay nạp chồng trong lớp đó.

Quy trình định nghĩa hàm khuôn mẫu

- Template thường phù hợp khi có nhiều hàm làm những việc y hệt nhau chỉ là trên các kiểu dữ liệu khác nhau
- Đầu tiên viết hàm trên kiểu cơ bản (vd int), sau đó chuyển thành template:
  - Thêm tiền tố khuôn mẫu (template prefix)
  - Chuyển tên kiểu cụ thể trong hàm thành một tham số kiểu với tên nào đó đã khai trong template (*vd*, T)

16.3 Lớp khuôn mẫu

- Có thể tạo ra các khuôn mẫu (template) cho lớp. Các lớp như vậy làm thành các kiểu dữ liệu trừu tượng.
- Khác hàm khuôn mẫu, lớp khuôn mẫu sẽ được hiện thể hoá bằng nêu trực tiếp tên của kiểu (**`int`**, **`float`**, **`string`**, etc.) vào thời điểm định nghĩa đối tượng

Lớp khuôn mẫu

Xem xét các lớp sau

1. Lớp dùng để kết hợp các số bằng cách cộng chúng:

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

2. Lớp dùng để kết hợp các xâu bằng cách cộng chúng

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

Ví dụ lớp khuôn mẫu

Có thể viết một lớp khuôn mẫu thể hiện được logic của cả 2 lớp trên: cần có template prefix (tiền tố template) để khai báo tham số kiểu:

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

Sử dụng lớp khuôn mẫu

Để tạo một đối tượng của lớp định nghĩa từ template, cần nêu rõ đối số kiểu của template

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

Prints `8.0` and `Hi Ho`

16.4 Lớp khuôn mẫu và kế thừa

- Các lớp khuôn mẫu cũng có thể kế thừa nhau như lớp bình thường
- Có thể dẫn xuất
  - Từ lớp khuôn mẫu ra lớp thường: hãy hiện thể hoá lớp cơ sở (vốn đang là lớp khuôn mẫu) rồi kế thừa từ hiện thể này
  - lớp khuôn mẫu kế thừa ra lớp khuôn mẫu
  - Các cách kết hợp 2 lựa chọn trên

16.5 Giới thiệu về Thư viện khuôn mẫu chuẩn

- Standard Template Library (STL): một thư viện chứa các khuôn mẫu cho các cấu trúc dữ liệu và giải thuật phổ dụng
- Dùng STL giúp xây dựng chương trình nhanh hơn, khả chuyển hơn

Standard Template Library

Hai kiểu cấu trúc dữ liệu quan trọng trong STL:

- Containers: các lớp chứa dữ liệu đã được tổ chức theo một cách nào đó
- Iterators (con trỏ chạy): một dạng dữ liệu kiểu con trỏ, dùng để duyệt từng phần tử trong một lớp container

Containers

Có hai loại lớp container trong STL:

- Containers kiểu tuần tự: tổ chức dữ liệu và truy cập các phần tử kiểu tuần tự, ví dụ như trong một mảng.Vd: các kiểu **`vector`**, **`dequeue`**, và **`list`**.
- Containers kiểu liên kết: dùng khoá để truy cập nhanh các phần tử, vd các kiểu **`set`**, **`multiset`**, **`map`**, và **`multimap`**.

Tạo đối tượng Container

- Để tạo một danh sách số **`int`**:

```
list<int> mylist;
```

- Để tạo một vector các **`string`**:

```
vector<string> myvector;
```

- Yêu cầu cần file header **`<vector>`**

Con trỏ chạy (Iterators)

- Là sự tổng quát hoá của con trỏ, dùng để truy cập thông tin trong containers
- Rất nhiều kiểu:
  - tiến (uses **++**)
  - Hai chiều (uses **++** and **--**)
  - Truy cập ngẫu nhiên
  - nhập (có thể dùng với **cin** và đối tượng **istream**)
  - xuất (có thể dùng với **cout** và đối tượng **ostream**)

Containers và con chạy

- Mỗi lớp container sẽ định nghĩa một kiểu con trỏ chạy để truy cập nội dung container đó
- Kiểu của con trỏ chạy được xác định từ kiểu container:

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

`x` là con chạy cho container có kiểu `list<int>`

Containers và con chạy

Mỗi lớp container cũng định nghĩa các hàm trả về các con trỏ chạy:

`begin()`: trả về con trỏ của phần tử đầu

`end()`: trả về con trỏ ứng với cuối container (không phải là trỏ về phần tử cuối)

Duyệt một Container

Cho một vector:

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

Duyệt vector này dùng iterators:

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

In ra **1 4 9 16 25**

Một vài hàm thành viên của lớp **`vector`**

front(), back()	Trả về tham chiếu tới phần tử đầu, cuối của vector ()
size()	Trả về số phần tử trong vector
capacity()	Trả về dung lượng phần tử mà vector có thể chứa được
clear()	Xoá mọi phần tử khỏi vector
push_back(value)	Chèn phần tử value vào cuối vector
pop_back()	Loại bỏ phần tử cuối ra khỏi vector
insert(iter, value)	Chèn phần tử value và ngay trước phần tử trỏ bởi iter

Thuật toán

- STL chứa các thuật toán được cài đặt như các hàm khuôn mẫu, dùng để thực hiện nhiều thao tác khác nhau trên các containers.
- Cần có file **<algorithm>**
- Tập hợp các thuật toán bao gồm:

```
binary_search   count
for_each        find
max_element     min_element
random_shuffle  sort
```

and others

Sử dụng thuật toán trong STL algorithms

- Rất nhiều thuật toán STL xử lý các container trong STL có liên quan đến xác định một con trỏ đầu và cuối
- **`max_element(iter1, iter2)`** tìm phần tử lớn nhất trong container trong khoảng từ phần tử (trỏ bởi) **`iter1`** đến phần tử (trỏ bởi) **`iter2`**
- **`min_element(iter1, iter2)`** tương tự trên, nhưng là phần tử nhỏ nhất

Thuật toán STL

- **`random_shuffle(iter1, iter2)`** xáo trộn ngẫu nhiên các phần tử trong container thuộc khoảng trên
- **`sort(iter1, iter2)`** sắp xếp các phần tử container nằm trong khoảng này

VD random-shuffle

Các số chính phương 1, 4, 9, 16, 25 được lưu trong một vector, hãy xáo trộn vector đó, rồi in ra.

Ví dụ **random_shuffle**

```
int main()
{
    vector<int> vec;
    for (int k = 1; k <= 5; k++)
      vec.push_back(k*k);
    random_shuffle(vec.begin(),vec.end());
    vector<int>::iterator p = vec.begin();
    while (p != vec.end())
    { cout << *p << "  "; p++;
    }
    return 0;
}
```