Sử Dụng Standard Template Library
Cài Đặt Đồ Thị

Bùi Tiến Lên

Đại học Khoa học Tự nhiên

01/06/2016

Thư viện template chuẩn

Tác giả đầu tiên của template chuẩn (**Standard Template Library - STL**) là Alexander Stepanov. Mục đích của ông là xây dựng thư viện thể hiện tư tưởng lập trình tổng quát (**generic programming**). Thư viện mẫu chuẩn C++ gồm 2 phần:

- Thư viện **string**
- Thư viện template chuẩn

Các khái niệm trong STL được phát triển độc lập với C++

- Lúc ban đầu, STL không phải là một thư viện C++, mà nó được chuyển đổi thành thư viện C++
- Nhiều ý tưởng dẫn đến sự phát triển của STL đã được cài đặt vào trong Java, C#

Lập trình tổng quát

Là phương pháp lập trình đi từ *cụ thể* sang *trừu tượng*. Mục đích của lập trình tổng quát là

- Tăng sự chính xác thông qua những đặc tả
- Tăng khả năng sử dụng lại
- Tăng khả năng thực thi

Ví dụ lập trình tổng quát

Thay vì viết hàm *Max* cho từng kiểu dữ liệu

```
1 double Max(double a, double b)
2 {
3     if(a < b) return b;
4     else return a;
5 }
```

Chúng ta có thể viết thành một hàm *Max* tổng quát như sau

```
1 template <class T> T Max(T a, T b)
2 {
3     if(a < b) return b;
4     else return a;
5 }
```

Thư viện STL

- STL được thiết kế đẹp, hiệu quả và không có kế thừa hay phương thức ảo
- Trước STL các **thuật toán** (**algorithm**) được xây dựng và gắn vào các **bộ chứa** (**container**)
- STL tách rời thuật toán và bộ chứa và chúng tương tác với nhau qua **bộ duyệt** (**iterator**). Việc này giúp cho dễ dàng hiệu chỉnh, bổ sung các thuật toán. Mỗi bộ chứa sẽ có một bộ duyệt riêng của nó

Các thư viện mẫu STL

Thư viện mẫu chuẩn bao gồm 32 thư viện

<algorithm>
<bitset>
<complex>
<deque>
<exception>
<fstream>
<functional>
<iomanip>
<ios>
<iosfwd>
<iostream>
<istream>
<iterator>
<limits>
<list>
<locale>
<map>
<memory>
<new>
<numeric>
<ostream>
<queue>
<set>
<sstream>
<stack>
<stdexcept>
<streambuf>
<string>
<typeinfo>
<utility>
<valarray>
<vector>

Bộ chứa & Bộ duyệt

Mọi bộ chứa trong STL đều có thể tuần tự hóa (**sequence**) nghĩa là có thể chỉ ra phần tử đầu, phần tử cuối và phần tử kế tiếp thông qua vị trí của chúng hay bộ duyệt
Mỗi bộ chứa có ít nhất 2 phương thức

- Phương thức *begin*() chỉ ra vị trí phần tử đầu tiên của bộ chứa
- Phương thức *end*() chỉ ra vị trí đứng sau phần tử cuối cùng của bộ chứa

Bộ chứa & Bộ duyệt (cont.)

Ví dụ tạo ra một lớp chứa *a* kiểu vector gồm 5 phần tử

```
1    vector<int> a;
2    a.push_back(3);
3    a.push_back(4);
4    a.push_back(1);
5    a.push_back(5);
6    a.push_back(7);
```

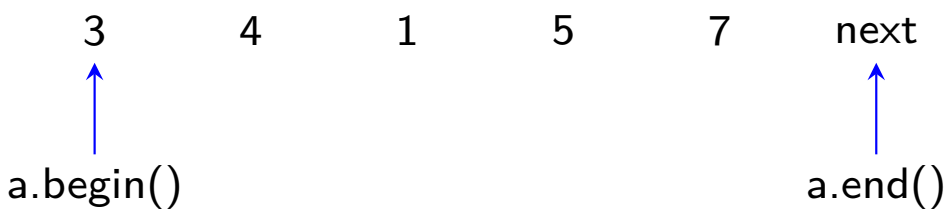

Figure 1: a.begin() sẽ trỏ đến phần tử 3, a.end() trỏ đến phần tử "next"

Bộ chứa & Bộ duyệt (cont.)

Mỗi bộ duyệt (con trỏ) của một bộ chứa có ít nhất 2 toán tử

- Toán tử $++$ di chuyển sang phần tử kế tiếp của bộ chứa
- Toán tử * lấy giá trị

Bộ chứa & Bộ duyệt (cont.)

```
1    vector<int>::iterator p;
2    p = a.begin();
3    p++;
```

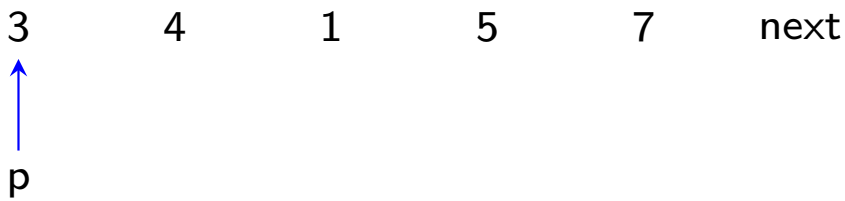

Figure 2: sau khi thực hiện dòng lệnh 2

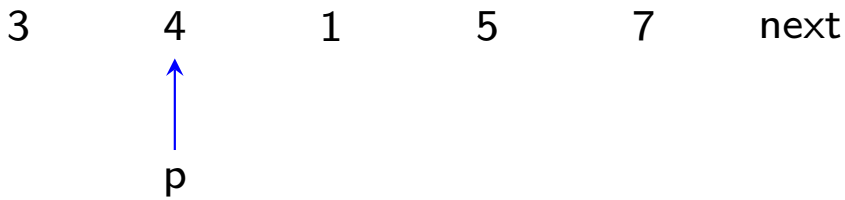

Figure 3: sau khi thực hiện dòng lệnh 3

Bộ chứa & Bộ duyệt (cont.)

Đoạn code duyệt tất cả các phần tử trong lớp chứa và in ra giá trị của từng phần tử

```
1     for(vector<int>::iterator pa=a.begin(); pa!=a.end(); pa++)
2         cout << *pa << endl;
```

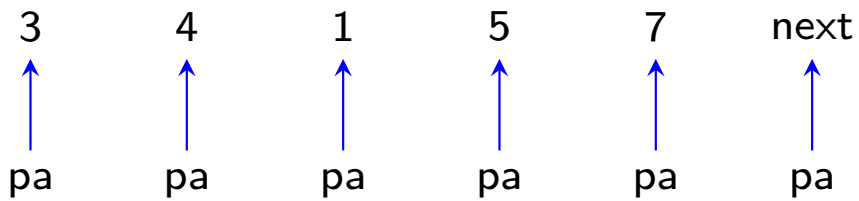

Figure 4: pa lần lượt trỏ đến từng phần tử

Bộ chứa & Bộ duyệt (cont.)

Một số lớp chứa cung cấp toán tử chỉ số để truy cập đến từng phần tử của

- Lớp vector

```
1 |     for(int i=0; i<a.size(); i++)
2 |         cout << a[i] << endl;
```

- Lớp map

```
1 |     map<string, int> b;
2 |     b["lan"] = 1;
3 |     b["dung"] = 4;
4 |     b["khang"] = 2;
5 |     map<int, string> c;
6 |     b[1] = "tin hoc";
7 |     b[3] = "toan hoc";
8 |     b[2] = "ly";
```

Sử dụng thư viện thuật toán của STL

- Sử dụng hàm *find* để tìm kiếm một phần tử trong một lớp chứa

```
1     int x = 3;
2     if(find(a.begin(), a.end(), x) != a.end())
3         cout << "a co chua x";
4     else
5         cout << "a khong chua x";
```

- Sử dụng hàm *sort* để sắp xếp một lớp chứa

```
1     sort(a.begin(), a.end());
```

- Sử dụng hàm *reverse* để đảo ngược thứ tự cho một lớp chứa

```
1     reverse(a.begin(), a.end());
```

Sử dụng STL để cài đặt đồ thị

- Khai báo thư viện STL cần thiết
- Khai báo macro
- Khai báo các cấu trúc dữ liệu
- Cài đặt các hàm liên quan

Khai báo các thư viện STL cần thiết

Các thư viện STL C++ cần thiết để cài đặt đồ thị

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <map>
5  #include <stack>
6  #include <queue>
7  #include <vector>
8  #include <algorithm>
9
10 using namespace std;
```

Khai báo các macro

Khi xử lý đồ thị những thao tác sau đây hay sử dụng

- Kiểm tra một phần tử x có nằm trong tập s hay không?
- Duyệt tuần tự các phần tử của tập s với px là con trỏ tới phần tử hiện hành

```
1 #define In(x, s) (find((s).begin(), (s).end(), (x)) != (s).end())
2 #define Foreach(px, s) for(px = (s).begin(); px != (s).end(); px++)
```

Khai báo các cấu trúc dữ liệu

Các cấu trúc dữ liệu cho đỉnh, cạnh

```
1 typedef string                        TVertex;
2 typedef pair<TVertex,TVertex>         TEdge;
3 typedef vector<TVertex>::iterator     iTVertex;
4 typedef vector<TEdge>::iterator       iTEdge;
```

Cấu trúc dữ liệu cho đồ thị

```
1 struct Graph {
2     string            m_kind;
3     vector<TVertex>   m_vertexes;
4     vector<TEdge>     m_edges;
5 };
```

Cài đặt các hàm liên quan

Các hàm cài đặt bao gồm

- Xác định kiểu đồ thị {direct, undirect} (*)

```
1 | void Set_Kind(Graph &G, string kind)
2 | {
3 |     G.m_kind = kind;
4 | }
```

- Thêm đỉnh vào đồ thị

```
1 | void Add_Vertex(Graph &G, TVertex vertex)
2 | {
3 |     if(!In(vertex, G.m_vertexes)) G.m_vertexes.push_back(vertex)
4 | }
```

Cài đặt các hàm liên quan (cont.)

- Thêm cạnh vào đồ thị

```
1 | void Add_Edge(Graph &G, TEdge edge)
2 | {
3 |     Add_vertex(G, edge.first);
4 |     Add_vertex(G, edge.second);
5 |     if(!In(edge, G.m_edges)) G.m_edges.push_back(edge);
6 | }
```

Cài đặt các hàm liên quan (cont.)

- Lấy danh sách đỉnh kề

```
1  vector<TVertex> Get_adjVertexes(Graph &G, TVertex v)
2  {
3      vector<TVertex> vertexes;
4
5      for(int i=0; i<G.m_edges.size(); i++)
6      {
7          if(G.m_edges[i].first == v)
8              if(!In(G.m_edges[i].second, vertexes))
9                  vertexes.push_back(G.m_edges[i].second);
10         if(G.m_edges[i].second == v)
11             if(!In(G.m_edges[i].first, vertexes))
12                 vertexes.push_back(G.m_edges[i].first);
13     }
14     return vertexes;
15 }
```

Cài đặt các hàm liên quan (cont.)

- Hàm xuất thông tin của đỉnh (*)

```
1 ostream& operator<<(ostream& os, const TVertex& v)
2 {
3     os << v;
4     return os;
5 }
```

- Hàm nhập thông tin của đỉnh (*)

```
1 istream& operator>>(istream& is, TVertex& v)
2 {
3     is >> v;
4     return is;
5 }
```

Cài đặt các hàm liên quan (cont.)

- In thông tin của đồ thị

```
1  void Print_Graph(Graph &G)
2  {
3      cout << "Kieu do thi = " << G.m_kind << endl;
4
5      cout << "Danh sach cac dinh" << endl;
6      for(int i=0; i<G.m_vertexes.size(), i++)
7          cout << G.m_vertexes[i] << endl;
8
9      cout << "Danh sach cac canh" << endl;
10     for(int i=0; i<G.m_edges.size(), i++)
11         cout << G.m_edges[i].first << " -- "
12         << G.m_edges[i].second << endl;
13 }
```

Cài đặt các hàm liên quan (cont.)

- Hàm đọc tập tin đồ thị

```
1  void Read_Graph(Graph &G, string const &filename)
2  {
3      ifstream f(filename.c_str());
4      TVertex first, second;
5
6      f >> G.m_kind;
7      while ( f.good() )
8      {
9          f >> first >> second;
10         if ( f.good() || f.eof() )
11             Add_Edge(G, TEdge(first, second));
12     }
13 }
```

Cài đặt các hàm liên quan (cont.)

- Hàm lưu tập tin đồ thị

```
1  void Write_Graph(Graph &G, string const &filename)
2  {
3      ofstream f(filename.c_str());
4
5      f << G.m_kind << endl;
6
7      for(int i=0; i<G.m_edges.size(); i++)
8          f << G.m_edges[i].first << " "
9          << G.m_edges[i].second << endl;
10 }
```

Tiếp tục

Cấu trúc tập tin lưu trữ cho một đồ thị

```
direct
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 4
4 1
```

Chương trình

Đoạn chương trình sau sẽ sử dụng các cấu trúc dữ liệu cũng như hàm cài đặt để tạo ra một đồ thị và in ra thông tin đồ thị

```
1  void main()
2  {
3      Graph G;
4
5      G.Set_Kind("direct");
6
7      G.Add_Vertex("A");
8      G.Add_Vertex("B");
9      G.Add_Vertex("C");
10
11     G.Add_Edge(TEdge("A", "B"));
12     G.Add_Edge(TEdge("B", "C"));
13
14     G.Print_Graph();
15 }
```

Tài liệu tham khảo