问题求解与实践
——标准模板库(STL)简介

主讲教师： 陈雨亭、沈艳艳

标准模板库(STL)

- **Standard Template Library**
- 容器（containers）
  - **list**（双向链表)、**vector**（类似于大小可动态增加的顺序表）、**queue**（队列）、**stack**（栈）、**string**
- 算法（algorithms）
- 迭代器（iterators）

STL的特点

- STL是以**容器**和**迭代器**为基础的一种**泛型算法**（Generic Algorithms）库
- 所谓**泛型**（Genericity）是指能够在多种数据类型上进行操作
  - **算法**是泛型的，不与任何特定的数据结构或对象类型系在一起
  - **容器**是泛化的，它可以是数组、向量、链表、集合、映射、队列、栈、字符串等等，容器中包含的元素对象可以是任意数据类型，容器提供迭代器用来定址其所包含的元素
  - **迭代器**是泛型的指针，是一种指向其他对象的对象，迭代器能够遍历由对象所形成的序列区间（Range）。迭代器将容器与作用其上的算法分离，大多数的算法自身并不直接操作于容器上，而是操作于迭代器所形成的区间中

在vector中插入删除元素

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
        vector<
        v1.pusl
        v1.pusl
        v1.pusl
        v1.pop
        v1.eras
        cout<<
        return 0;
}
```

```
1
--------------------------------
Process exited after 0.6174 seconds with return value 0
请按任意键继续. . .
```

访问vector中的元素

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
        vector<int>
        v1.push_ba
        v1.push_ba
        v1.push_ba
        v1[3]=10;
        for(int i=0;
            cout<<v

        v1[2]=10;
        cout<<endl
        vector<int>
        for(it=v1.begin(); it<v1.end(); it++)
        if(*it%2==0)  cout<<*it<<"  ";  //用迭代器访问，仅输出偶数
        return 0;
}
```

```
0  1  2
0  10
--------------------------------
Process exited after 0.4984 seconds with return value 0
请按任意键继续. . .
```

stack的简单使用

```
#include<iostream>
#include<stack>
using namespace s
int main()
{
        stack<int>
        int arry[4]=
        for(int i=0;i
                s.pus
        //输出栈中元
        cout<<"栈长
        while(!s.em
        {
                cout
                s.po
        }
        return 0;
}
```

```
栈长度为：4
4 3 2 1
--------------------------------

Process exited after 0.5133 seconds with return value 0
请按任意键继续. . .
```

queue的简单使用

```
#include<iostream>
#include<queue>
using namespace
int main()
{
        queue<int
        int arry[4]=
        for(int i=0;
                q.pu
        //输出队列中
        cout<<"队
        while(!q.em
        {
                cou
                q.po
        }
        return 0;
}
```

```
队列长度为：4
1 2 3 4
--------------------------------

Process exited after 0.4882 seconds with return value 0
请按任意键继续. . .
```

sort算法

- **算法：sort()**
  - 形式1：
    - sort(first, second)
    - 对容器中[first, second)之间的元素排序
    - 要求元素有比较大小的默认方法，比如int、double类型就可比较大小
  - 形式2：
    - sort(first, second, fun)
    - 对容器中[first, second)之间的元素排序，元素之间用fun函数比较大小
    - fun是自定义函数，以两个元素（与容器元素同类型）为参数

The C++ Standard Template Library (STL)

- Review: Polymorphic Containers
  - Suppose we want to model a graphical `Scene` that has an ordered list of `Figures` (i.e., `Rectangles`, `Circles`, and maybe other concrete classes we haven't implemented yet).
    - `Figure` is an abstract base class (ABC)
    - `Rectangle`, `Circle`, etc. are derived classes
  - What should the list look like?

    1. `vector<Figure>`
    2. `vector<Figure&>`
    3. `vector<Figure*>`

The C++ Standard Template Library (STL)

```
Circle c (“red”);
vector<Figure> figList;
figList.emplace_back(c);
```

Objects:

- Copy operations could be expensive
- Two red circles
- Changes to one do not affect the other
- When `figList` dies, its copy of red circle is destroyed
- Risk of static slicing

```
Circle c (“red”);
vector<Figure*> figList;
figList.emplace_back(c);
```

Pointers:

- Allows for polymorphic containers
- When `figList` dies, only pointers are destroyed
- Client code must clean up referents of pointer elements

The C++ Standard Template Library (STL)

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Balloon {
   public:
      Balloon (string colour);
      Balloon (const Balloon& b); // copy ctor
      virtual ~Balloon();
      virtual void speak() const;
   private:
      string colour;
}

Balloon::Balloon(string colour) :
colour(colour) {
   cout << colour << " balloon is born" <<
endl;
}
Balloon::Balloon(const Balloon& b) :
colour(b.colour) {
   cout << colour << " copy balloon is born" <<
endl;
}
```

```
void Balloon:speak() const {
   cout << "I am a " << colour << "balloon" <<
endl;
}
Balloon::~Balloon() {
   cout << colour << " balloon dies" << endl;
}
```

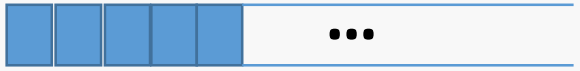
```
// How many Balloons are created?
int main(int argc, char* argv[]) {
   vector<Balloon> v;
   Balloon rb ("red");
   v.push_back(rb);
   Balloon gb ("green");
   v.push_back(gb);
   Balloon bb ("blue");
   v.push_back(bb);
}
```

STL Containers

- C++ 98/03 defines three main data container categories
  - Sequence Containers: `vector, deque, list`
  - Container Adapters: `stack, queue, priority_queue`
  - Ordered Associative `Containers: [multi]set, [multi]map`
- C++11 adds:
  - Addition of `emplace{_front, _back}`
  - Sequence Containers: `array, forward_list`
  - Unordered Associative Containers: `unordered_[multi]set, unordered_[multi]map`
- C++14 adds:
  - Non-member `cbegin, cend, rbegin, rend, crbegin, crend.`

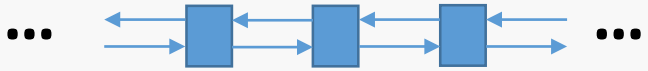
STL Containers – Conceptual View

- Sequence Containers
  - Vector

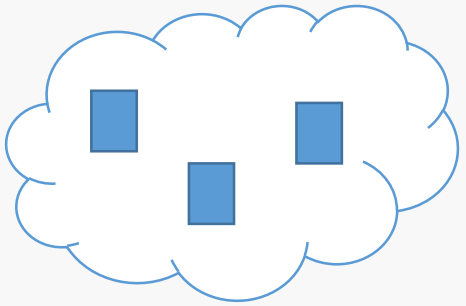


  - Deque

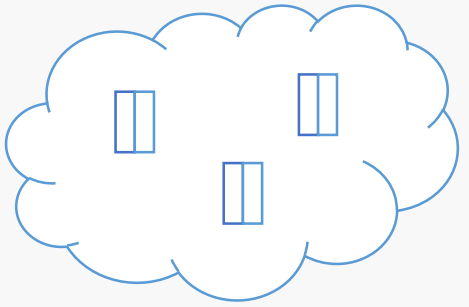
  - List



➢ Sequence Containers
  - [multi]set



  - [multi]map



STL Containers

	STL containers	*Some* useful operations
	all containers	size, empty, emplace, erase
Sequence	vector<T>	[], at, clear, insert, back, {emplace,push,pop}_back
Sequence	deque<T>	[], at, emplace{,_front,_back}, insert, {,push_,pop_}back, {,push_,pop_}front
Sequence	list<T>	insert, emplace, merge, reverse,splice, {,emplace_,push_,pop_}{back,front}, sort
Sequence	array<T>	[], at, front, back, max_size
Sequence	forward_list<T>	assign, front, max_size, resize, clear, {insert,erase,emplace}_after, {push,pop,emplace}_front
Associative	set<T>, multiset<T>	find, count, insert, clear, emplace, erase, {lower,upper}_bound
Associative	map<T1,T2>, multimap<T1,T2>	[]*, at*, find, count, clear, insert, emplace, erase, {lower,upper}_bound
Unordered Associative	unordered_set<T>, unordered_multiset<T>	find, count, insert, clear, emplace, erase, {lower,upper}_bound
Unordered Associative	unordered_map<T1,T2>, unordered_multimap<T1,T2>	[]*, at*, find, count, clear, insert, emplace, erase, hash_function
Container Adaptors	stack	top, push, pop, swap
Container Adaptors	queue	front, back, push, pop
Container Adaptors	priority_queue	top, push, pop, swap
Other	bitset (N bits)	[], count, any, all, none, set, reset, flip

Red means "there's also a stand-alone algorithm of this name"

Can't iterate over stack, queue, priority_queue.

* Not on multimap