第3章 栈和队列

In [1]:
```
#include <iostream>
using namespace std;
```

3.1 栈

这里使用向量实现栈，并且入栈和出栈是在向量的尾部进行的，如果在首元素进行，
会导致频繁的搬运后面的元素，带来大量的开销。

In [2]:
```
#include "vector.h"
template <class T>
class Stack: public Vector<T> {
public:
    // 声明需要用到的基类中的东西，避免两阶段名字查找带来的编译错误
    using Vector<T>::size;
    using Vector<T>::data;
    using Vector<T>::insert;
    using Vector<T>::remove;

    // 手动调用基类的构造函数,默认初始化一个空栈
    Stack(): Vector<T>(0) {}

    // 声明入栈和出栈函数
    void push(T& e);
    T pop();

    // 返回栈顶元素
    T top(){return data[size - 1];}
};
```

入栈
就是在最后一个元素的后面一个位置插入元素，注意这里
参数是引用，不能直接填1，2，3，需要用变量名代替

In [3]:
```
template <class T>
void Stack<T>::push(T& e){
    insert(size,e);
}
```

In [4]:
```
Stack<int> s;
s.size
```

Out[4]:
```
0
```

In [5]:
```
int a[] = {1, 2, 3};
s.push(a[0]);
s.push(a[1]);
s.push(a[2]);
```

In [6]:
```
s.size
```

Out[6]:
```
3
```

出栈
弹出最后一个元素

In [7]:
```
template <class T>
T Stack<T>::pop(){
    return remove(size - 1);
}
```

In [8]:
```
s.pop()
```

Out[8]:
```
3
```

In [9]:
```
s.pop()
```

Out[9]:
```
2
```

In [10]:
```
s.pop()
```

Out[10]:
```
1
```

In [11]:
```
s.empty()
```

Out[11]:
```
true
```

In [12]:
```
s.size
```

Out[12]:
```
0
```

3.2 队列

由于队列的入队和出队操作涉及到向量或者列表的两端，所以对于向量来说，
必有一端涉及到大量元素搬运，所以这里使用列表实现队列
参考栈的实现方式，队列的实现如下，约定队首为列表的第一个元素

In [13]:
```
#include "List.h"
```

In [14]:
```
template <class T>
class Queue: public List<T>{
    public:
        using List<T>::header;
        using List<T>::tailer;
        using List<T>::insertA;
        using List<T>::remove;

        Queue():List<T>(){}
        void enqueue(T &e);
        T dequeue();
};
```

In [15]:
```
template <class T>
void Queue<T>::enqueue(T& e){
    insertA(header,e);
}
```

In [16]:
```
template <class T>
T Queue<T>::dequeue(){
    return remove(tailer -> pred);
}
```

In [17]:
```
auto q = Queue<int>();
```

In [18]:
```
q.empty()
```

Out[18]:
```
true
```

In [19]:
```
int a[] = {1, 2, 3, 4};
for (int i = 0; i < 4; i++)
    q.enqueue(a[i]);
```

In [20]:
```
q.empty()
```

Out[20]:
```
false
```

In [21]:
```
q.dequeue()
```

Out[21]:
```
1
```

In [22]:
```
q.dequeue()
```

Out[22]:
```
2
```

In [23]:
```
q.dequeue()
```

Out[23]:
```
3
```

In [24]:
```
q.dequeue()
```

Out[24]:
```
4
```

In [25]:
```
q.empty()
```

Out[25]:
```
true
```

In [ ]:
```

```