Homework #6

Demo date: 5/10

Deque, stack, and set partition

Part A – Deque implementation

```
template<typename T>
class deque {
public:
// types
   typedef T value_type;
   typedef size_t size_type;
   typedef T& reference;
   typedef const T& const_reference;
// ctor/copy ctor/dtor
   deque();
   deque(const deque&);
   deque(deque&&);
   ~deque();
// capacity
   size_type size() const;
   bool empty() const;
// modifiers
   void push_front(const T&);
   void push_front(T&&);
   void push_back(const T&);
   void push_back(T&&);
   void pop_front();
   void pop_back();
// element access
   reference front();
   const_reference front() const;
   reference back();
   const_reference back() const;
private:
   struct node;
   node *head;
   size_type _size;
};
```

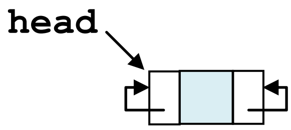
```
template<typename T>
struct deque<T>::node {
   node(const T&,node*,node*);
   node(T&&,node*,node*);
   T datum;
   node *pred,*succ;
};
```

A deque is implemented as a doubly linked list with a header node. Each node in the list contains two pointers **pred** and **succ** that point to its predecessor and successor nodes, respectively.

For example, the declaration

```
   deque<int> d;
```

creates an empty queue whose internal structure is



This node is called a header node. Notice that the **datum** field of the header node contains no datum and shan't be initialized. That is to say, the header node shan't be created by a call to **node**'s ctor:

```
template<typename T>
deque<T>::deque()
:  head(new node(?,nullptr,nullptr))
{
   head->pred=head->succ=head;
}
```

What is the initial value?

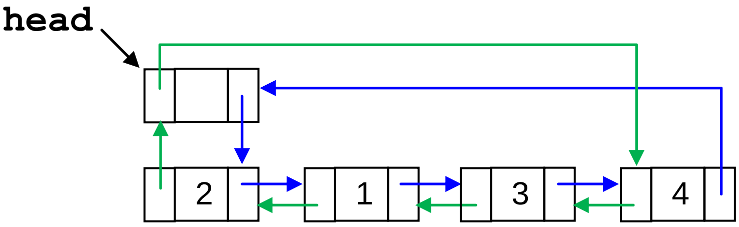
Instead, it shall be allocated by **`operator new`**.

```
template<typename T>
deque<T>::deque()
:   head((node*)operator new(sizeof(node)))
{
    head->pred=head->succ=head;
}
```

After executing the following code,

```
    d.push_front(1);
    d.push_front(2);
    d.push_back(3);
    d.push_back(4);
```

the underlying doubly linked list representing the deque **`d`** becomes



where the blue-colored arrows are the successor pointers, and the green-colored arrows are the predecessor pointers.

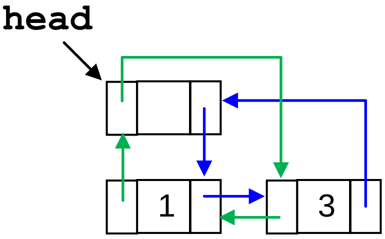
With this snapshot, the operations

`d.front()` shall return 2;
`d.back()` shall return 4;
`d.size()` shall return 4; and
`d.empty()` shall return false.

Subsequently, the operations

```
    d.pop_front();
    d.pop_back();
```

turn the underlying doubly linked list into



Your job for this part is to define all the member functions of the class template **deque** outside the class body.

Part B – Stack implementation

The STL **stack** is a container adaptor. Any sequence container that supports **push_back()**, **pop_back()**, and **back()** can be used to instantiate **stack**. In particular, **deque**, **vector**, or **list** (to be introduced later) can be used.

This part asks you to implement the STL **stack** by yourself. Define all the member functions outside the class body.

```
#include "deque6.h"   // include our own deque

template<typename T,typname Container=deque<T> >
class stack {
public:                          // default container is deque
// types
   typedef typename Container::value_type value_type;
   typedef typename Container::size_type size_type;
   typedef typename Container::reference reference;
   typedef typename Container::const_reference const_reference;
// ctor                          // container's default ctor
   stack(const Container&);
   stack(Container&& =Container());
   stack(stack&&);               // omit this in GNU C++
// modifiers
   void push(const value_type&);
   void push(value_type&&);
   void pop();
// element access
   reference top();
   const_reference top() const;
// capacity
   bool empty() const;
   size_type size()  const;
private:
   Container c;
};
```

Observe that
1 the default container is a deque, and
2 a stack may be default-initialized by the container's default ctor.

Example 1

```
stack<int> s;
stack<int,deque<int> > t;
```

Both create a stack that is implemented by a deque and initialized by the deque's default constructor.

Example 2

```
vctor<int> v(5,7);
stack<int,vector<int> > s;
stack<int,vector<int> > t(v);
```

Both create a stack that is implemented by a vector. Stack *s* is initialized by the vector's default constructor, but stack *t* is initialized by a vector containing 5 7's

Part C – Stack application

In this part, you are asked to rewrite the set partition problem of HW#5, but this time uses recursion plus a stack to record the subsets generated.

How do recursion and subset enumeration work?

First of all, recall the recurrence

$$\begin{aligned} t[1,j] &= 1, \text{if } j = 0 && (\text{i.e. the subset is } \emptyset) \\ &\quad\ \text{or } j = a_1 && (\text{i.e. the subset is } \{a_1\}) \\ &= 0, \text{otherwise} && (\text{i.e. } j < 0 \text{ or } j > 0 \text{ but } j \neq a_1) \end{aligned}$$

$$t[i,j] = t[i-1, j-a_i] + t[i-1, j], \qquad \text{if } i > 1$$

Let's assume that stack *s* is used to record the subsets generated.

1 On solving $t[i, j],\ i > 1$

a) Before solving $t[i-1, j-a_i]$ recursively, push the element a_i onto stack *s* and pop it off after the recursion returns.

b) Do nothing to stack *s* before and after solving $t[i-1, j]$

2 On solving $t[1, j]$
If $j = 0$, output the contents of stack *s*.
else if $j = a_1$, output a_1 and the contents of stack *s*
Note: Use the slow method mentioned in HW#5 to output *s*.

Requirement

Pass the stack by value using copy and move ctors, as mentioned in C++11 supplementary.

File organization requirement

Your program shall be organized in three files.

1 Deque implementation file (say, deque6.h)
This file contains the implementation of template class **`deque`**.

2 Stack implementation file (say, stack6.h)
This file includes deque6.h and contains the implementation of template class **`stack`**.
Note: If your own deque implementation fails, you may include STL deque.

3 Application file (say, hw6.cpp)
This file includes stack6.h and contains the stack application for the set partition problem.
Note: If your own stack implementation fails, you may include STL stack.

Important notice

You may feel free to change any STL-unrelated name mentioned in these files. Put another way, except for STL classes and public members, you may change the names of

1 private members, e.g. **`node`**, **`head`**, **`_size`**, **`pred`**, **`succ`**, etc.

2 user-defined variables and functions, e.g. **`subset`**, **`o_O`**, etc.

Sample test

Suffice it to run the sample test given in file hw6.cpp, together with the implementation files deque6.h and stack6.h.

Sample output

```
Test 1...
1 6 7
2 5 7
3 4 7
1 2 4 7
3 5 6
1 2 5 6
1 3 4 6
2 3 4 5
8 subset(s) in total

Test 2...
3 7 8
1 2 7 8
4 6 8
1 3 6 8
1 4 5 8
2 3 5 8
1 2 3 4 8
5 6 7
1 4 6 7
2 3 6 7
2 4 5 7
1 2 3 5 7
3 4 5 6
1 2 4 5 6
14 subset(s) in total

Test 3...
0 subset(s) in total
```

```
Test 4...
o_O copied
o_O moved
```

o_O