OOP12 MACHINE TEST

General information

1 Time 2011/6/14 6:00~10:00 pm

2 Score 4 problems; 100 points

3 Problems

Problem	Score	File	Compiler used
A	20	A.cpp	VC++ or GNU C++
B	30	B.cpp	VC++ or GNU C++
C	25	C.cpp, deque6.h	GNU C++
D	25	D.cpp, vector7.h	GNU C++

4 General requirements

- Open course web site, lecture notes, homework solutions, and nothing else.
- Use the stipulated algorithm, if any.
- Comments are not required.
- Suffice it to run the sample test. However, you shall present a general solution for each problem – any solution tailored for the sample test data will come to nought.
- Be honorable!
  Any activity unrelated to the test such as browsing the web, chatting on web, playing game, etc., is strictly prohibited.

Problem A

Generic algorithm

Write the following generic algorithm

```
template<class InputIterator,class Predicate>
bool any_of(InputIterator first,InputIterator last,
                                        Predicate pr);
```

that returns false if **[first,last)** is empty or if there is no iterator **it** in the range **[first,last)** such that **pr(*it)** is true, and true otherwise.

For example,

```
int a[9]={1,2,3,4,5,6,7,8,9};
any_of(a,a,[](int x){ return x==7; })     // false ∵ empty
any_of(a,a+9,[](int x){ return x==7; }) // true  ∵ there exists 7
any_of(a,a+9,[](int x){ return x==0; }) // false ∵ there exists no 0
```

Required algorithms

For the purpose of this machine test, you are asked to implement this function in three different ways, depending on the categories of the iteartors.

1 For input and forward iterators
  Scan **[first,last)** from **first** to **last**, using **operator++**.
2 For bidirectional iterators
  Scan **[first,last)** from **last** to **first**, using **operator--**.
3 For random access iterators
  Divide **[first,last)** into two ranges whose distances are equal or differ by one. Recursively solve the two ranges.

Sample test

Suffice it to run the sample test contained in file A.cpp.

Sample run

```
01110
```

Problem B

STL string

Extend the **`String`** class given in the lecture with the member function

```
String& insert(size_type pos,const String& str);
```

This function inserts **`str`** into the string at position **`pos`**, where **`pos`** $\leq$ **`size()`** is assumed to be true.
More formally, let
`*this` $= c_0 c_1 \ldots c_{pos-1} c_{pos} \ldots c_{n-1}$ where $n =$ **`size()`**
`str` $= d_0 d_1 \ldots d_{k-1}$ where $k =$ **`str.size()`**
then the result is
`*this` $= c_0 c_1 \ldots c_{pos-1} d_0 d_1 \ldots d_{k-1} c_{pos} \ldots c_{n-1}$
Note that a string may be inserted into itself, i.e. **`this==&str`**. In this case, the result is
`*this` $= c_0 c_1 \ldots c_{pos-1} c_0 c_1 \ldots c_{pos-1} c_{pos} \ldots c_{n-1} c_{pos} \ldots c_{n-1}$

Required storage allocation strategy

1 **`capacity()`** $\geq$ **`size()+str.size()`**
In this case, no expansion is needed. Thus, the capacity is left unchanged.

2 **`capacity()`** $<$ **`size()+str.size()`**
In this case, expansion is needed.
Set **`capacity()`** $= 2^m - 1$ where m is the smallest integer satisfying $2^m - 1 \geq$ **`size()+str.size()`**
Note: Basically, you shall repeatedly double the space until m is found.

Example

Before: **`capacity()`** $= 15$, **`size()`** $= 11$, **`str.size()`** $= 8$
After: **`capacity()`** $= 31$, **`size()`** $= 19$

Before: **`capacity()`** $= 0$, **`size()`** $= 0$, **`str.size()`** $= 6$
After: **`capacity()`** $= 7$, **`size()`** $= 6$

Hint: Pay attention to **`size()`** $= 0$, **`rhs.size()`** $= 0$, and self-insertion.

Sample test

1 Suffice it to run the sample test contained in file B.cpp.

2 **`insert`** has already been declared within class **`String`** defined in file B.cpp.
You need only define it outside the class body.
Beware: The class is named **`String`**, where **`S`** is a capital.

Sample run

```
Snoopy
6 15
Snoopy
6 7
SnoopyPluto
11 15
SnoopyGarfieldPluto
19 31
GarfieldPluto
13 15
GarfieldPlutoGarfieldPluto
26 31
SnoopyGarfieldSnoopyGarfieldPlutoPluto
38 63
SnoopyGarfieldSnoopyGarfieldPlutoPlutoGarfield
46 63
```

Problem C

STL deque

Extend the **deque** class of HW#6 with the following two members.

1 **deque<T>::deque(initializer_list<T> init);** (10%)

This initializer-list constructor constructs a deque by copying the elements of the initializer list **init** to it.

Example

```
deque<int> d{1,2,3,4,5};
```

constructs a deque with 5 integers 1, 2, 3, 4, and 5 (from **begin()** to **end()**).

2 **template<typename Predicate>**
void deque<T>::remove_if(Predicate pr); (15%)

where **Predicate** denotes a function type of the form **T** → **bool**.

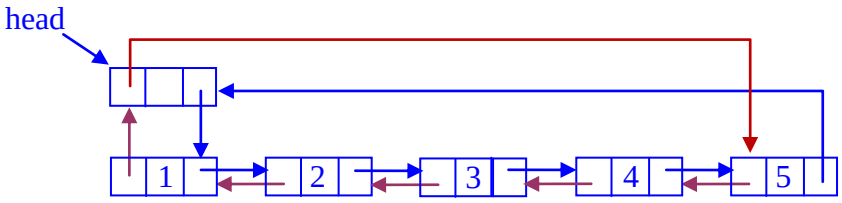
This member function template erases all elements in the deque that satisfy the predicate **pr**.

Example

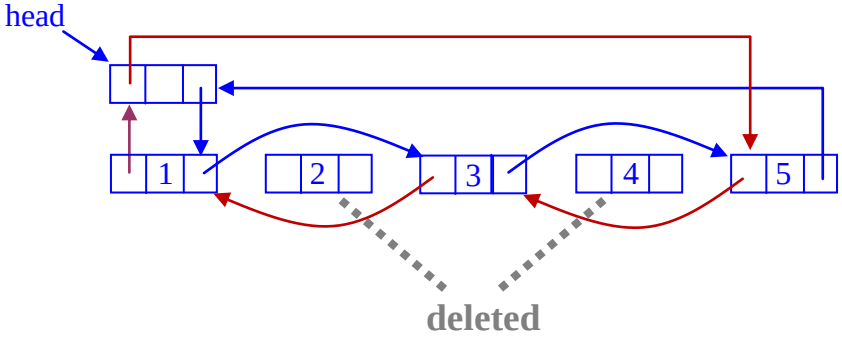
```
deque<int> d{1,2,3,4,5};
d.remove_if([](int x) { return x%2==0; });
```

This call removes the even integers 2 and 4 from the deque **d**.

Before:



After:



Sample test

1 Suffice it to run the sample test contained in file C.cpp.

2 Both members have already been declared within class **deque** defined in file deque6.h. You need only define them outside the class body.

3 Compile your program under GNU C++, say

```
bsd> g++47 -std=c++11 C.cpp
```

Sample run

```
1 3 5 7 9
Pluto Garfield Doraamon Pluto
Garfield Doraamon
```

Problem D

STL vector

Extend the **vector** class of HW#7 with the following two members.

1 **template<typename InputIterator>** (10%)
vector<T>::vector(InputIterator first,InputIterator last);

This templated ctor constructs a vector equal to the range **[first,last)**.
If iterators **first** and **last** are move iterators, it makes n calls to the move ctor of **T**; otherwise, it makes n calls to the copy constructor of **T**, where n is the distance between **first** and **last**.
The size and capacity of the vector are both n.

Example

```
vector<string> u{"Snoopy","Pluto","Garfield"};
vector<string> v(u.begin(),u.end());
```

At this point, vectors **u** and **v** have the same contents. The size and capacity of vector **v** are both 3. Vector **u** remains unchanged.

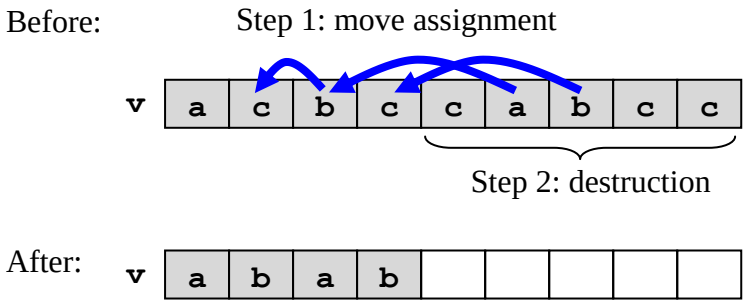
```
vector<string> w(make_move_iterator(u.begin()),
                 make_move_iterator(u.end()));
```

At this point, vector w contains the 3 strings originally owned by vector **u**. On the other hand, the size of vector **u** remains 3, but it contains no strings, as they have already been moved to vector **w**.

2 **template<typename Predicate>**
void vector<T>::remove_if(Predicate pr); (15%)
where **Predicate** denotes a function type of the form **T** → **bool**.
This member function template erases all elements in the vector that satisfy the predicate **pr**.

Example

```
string a("Snoopy"),b("Pluto"),c("Garfield");
vector<string> v{a,c,b,c,c,a,b,c,c};
v.remove_if([&](const string& s) { return s==c; });
```



Sample test

1 Suffice it to run the sample test contained in file D.cpp.

2 Both members have already been declared within class **`vector`** defined in file vector7.h. You need only define them outside the class body.

3 Compile your program under GNU C++, say

```
bsd> g++47 -std=c++11 -rpath=/usr/local/lib/gcc47 D.cpp
```

Sample run

```
Test 1 ...
3 Snoopy Pluto Garfield
3 Snoopy Pluto Garfield
3

Test 2 ...
Pluto    move-assigned to Garfield
Snoopy   move-assigned to Garfield
Pluto    move-assigned to Garfield
Garfield destructed
Garfield destructed
Garfield destructed
Garfield destructed
Garfield destructed
Snoopy  Pluto   Snoopy  Pluto

Test 3 ...
345
56
7
89
```