Spring 2011

OOP Final Exam

25 (sub)problems in total, 4% for each subproblem

1 For each part below, the starred line contains an error. Figure it out and ***explain***.

a)
```
queue<int> q;
for (int i=1;i<=9;i++) q.push(i);
cout << accumulate(q.begin(),q.end(),0);     //*
```

b)
```
ostream& operator<<(ostream& os,const deque<int>& d)
{
    const deque<int>::iterator it;           //*
    for (it=d.begin();it!=d.end();++it) os << *it;
    return os;
}
```

c)
```
vector<int> v;
for (int i=1;i<=9;i++) v.push_front(i);      //*
```

d)
```
auto_ptr<int> p(new int(7)),q(p);
cout << *p;                                  //*
```

2 For each declaration below, determine if it is (1) a copy ctor, (2) a ctor but not a copy ctor, or (3) not a ctor at all. (4%)

```
① string::string(string);     ③ string::string(string,int=1);
② string::string(string&);    ④ string::string(string&,int=1);
```

3 Consider

```
class string {
public:
    string(const char* ="");
    string(string&);                     // 1 ⎫
    string(const string&);               // 2 ⎬ Only one exists.
    explicit string(const string&);      // 3 ⎭
};
```

Two of the three copy ctors make the following code illegal. Figure them out and ***explain***.

```
string s="snoopy";
```

4 The definition of the following ctor is erroneous. Figure the error out and correct it.

```
template<typename T>
vector<T>::vector(size_type n,const T& val)
:   _size(n),_capacity(n),_data((T*)operator new[](n*sizeof(T)))
{
    for (int i=0;i<n;i++) _data[i]=val;
}
```

5 Consider the class **list** of integer singly linked lists given in the lecture and recall that the class **list::iterator** supports forward iterators.

a) Define **operator->**

```
int* list::iterator::operator->() const;
```

in terms of **operator***.

b) Define the postfix **operator++**

```
const list::iterator list::iterator::operator++(int);
```

in terms of the prefix **operator++**.

c) Why we insist that the postfix **operator++**, as shown in b), should return by const value, but **list::begin()**, as shown below, may return by value?

```
list::iterator list::begin();
```

6 a) What is wrong with the following defintion of the generic function **accumulate**? How to correct it?

```
template<class InputIterator,class T>
T accumulate(InputIterator first,InputIterator last,T init)
{
    T r=init;
    for (InputIterator it=first;it!=last;it=it+1) r=r+*it;
    return r;
}
```

b) Given a STL list **list<int> a;**

Which way of computing the sum of list elements runs faster and why?

1) **accumulate(a.begin(),a.end(),0)**

2) **accumulate(a.rbegin(),a.rend(),0)**

7 a) Define the following function, as given in the lecture,

```
string operator+(const string&,const string&);
```

to concatenate two **string** objects and return the resulting **string** object as function value.

b) Recall that, in STL, the operator function of part a) is overloaded with

```
string operator+(const char*,const string&);
string operator+(const string&,const char*);
```

What is wrong with the call

```
operator+("snoopy","pluto")
```

How to make it work?

8 Consider the following template class

```
template<class T1,class T2>
struct pair {
   T1 first; T2 second;
   pair() : first(),second() {}
   // other members omitted
};
```

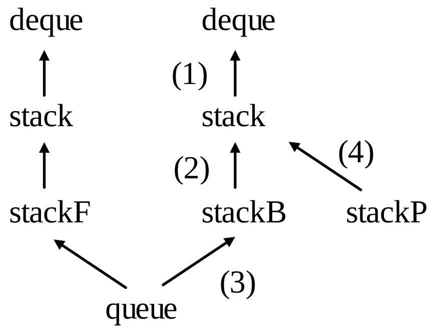
Define a necessary ctor outside the class body to enable the following code:

```
pair<int,unsigned> a;
pair<unsigned,int> b(a);
```

9 Fill in the following blanks.

Inheritance	Virtual or not?	What is (are) inherited?
private / protected	NA	(1)
public	non-virtual	(2)
	virtual	(3)
	pure virtual w/o implementation	(4)

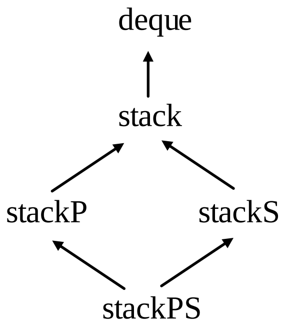
10 Consider the class hierarchy discussed in the lecture



a) What kind of inheritance is used in (1)? in (2)? in (3)? in (4)?

b) In order for **queue** objects to manipulate **stackF** and **stackB** subobjects, but prevent outsiders to manipulate **stackF** and **stackB** objects, how should the classes **stackF** and **stackB** be designed? (DON'T write any code, just explain.)

11 Consider the class hierarchy discussed in the lecture



a) Write down the implicitly generated ctor for class **stackPS**.

b) Show the *four* upcasts that occur during the construction of a **stackPS** object:

```
stackPS s;
```

12 Consider the **vector** class of HW#6 and the creation of a vector object

```
vector<vector<int> > v(2,vector<int>(3,5));      //*
```

a) Draw a picture showing the internal structure of the vector **v**.

b) Suppose that we do not define our own copy ctor and rely on the implicitly generated copy ctor. Explain why the vector **v** cannot be created.

c) Fill in the following blanks to print out the vector **v**

```
for (vector<vector<int> >::iterator rit=v.begin();rit!=v.end();++rit)
    for (vector<int>::iterator cit= (1) ;cit!= (2) ;++cit)
        cout << *cit << ' ';
```

13 Given

```
int a[9]={3,2,4,5,2,2,6,7,8};
list<int> b(a,a+9);
```

What is the difference between **b.remove(2);b.push_back(9);**

and **remove(b.begin(),b.end(),2);b.push_back(9);**

14 Consider the following class

```
class X {
public:
    X() : x(new int) {}                  // single object
    X(int n) : x(new int[n]) {}          // array object
private:
    int* x;
};
```

How would you define the dtor to deallocate the storage obtained by both ctors?

(Hint: Introduce a new private data member to distinguish the ctor called.)