

## OOP Final

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

1 Fill in the following blanks.

a) // implement a queue as a sequential array

```
class queue {
public:
    queue(int n) : _____ {}
    void push(int n) { _____ }
    void pop() { _front=(_front+1)%(_capacity+1); }
    int& front(){ return _q[_front]; }
    // other members omitted
private:
    int _front, _back, _capacity;
    int* _q;
};
```

b) // print out a list of int lists

```
for (list<list<int> >::iterator it1=a.begin(); it1!=a.end(); ++it1) {
    for (_____; _____; ++it2)
        cout << *it2;
    cout << endl;
}
```

2 Each part contains one or two errors. Figure out the errors.

a) `template<typename T>`

```
void p(T&) { T::value_type *p; }
```

b) `class string {`

```
public:
```

```
    static void dstring(const char* s)
    { strncpy(_dstring, s, _dsize-1); }
```

```
private:
```

```
    static const int _dsize=16;
    static char _dstring[_dsize]="";
};
```

c) `void r(auto_ptr<int> q) {}`

```
int main() { auto_ptr<int> p(new int(7)); r(p); cout << *p; }
```

- d) `class string {`  
`public:`  
`string(const char* s="")`  
`: _size(strlen(s)), _capacity(_size),`  
`_data(strcpy(new char[_capacity+1],s)) {}`  
`private:`  
`size_t _capacity,_size; char* _data;`  
`};`
- e) `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; }`
- f) `template<class InputIterator,class T>`  
`T accumulate(InputIterator first,InputIterator last,T init)`  
`{`  
`T r=init;`  
`for (InputIterator it=first;it<last;it+=1) r+=*it;`  
`return r;`  
`}`

3 Answer the following questions briefly

- a) Given a class **X** with a ctor **X::X(int)**  
Under what conditions will the following two declarations  
**X a(7); X a=7;**  
have the same effect, regardless of optimization?
- b) Let **X** be a class. How would you prevent an **X** object from being passed by value?

4 Show the output of the following code

```
class str {
public:
    str(const char* s="") : s(s) { cout << s << " constructed" << endl; }
    str(const str& rhs) : s(rhs.s) { cout << s << " copy-constructed" << endl; }
    ~str() { cout << s << " destructed"<< endl; }
private:
    string s;
};

int main() { vector<str> v(2,"Snoopy"); }
```

- 5 Suppose that a stack is implemented as a linked list

```
class stack {
public:
    stack() : _top(NULL) {}
    ~stack() { while (!empty()) pop(); }
    typedef size_t size_type;
    size_type size() const {
        return _top==NULL? 0: 1+stack(_top->succ).size();
    }
    // other members omitted
private:
    stack(node* t) : _top(t) {}
    struct node { node(int,node*); int datum; node* succ; }; // *
    node* _top;
};
```

- Suggest a way to make the code work, if the underlined keyword **struct** in the starred line is replaced by the keyword **class**.
- Explain why the member function **size()** doesn't work.
- Suggest a way to make it work.

- 6 Consider the following definition of the **pair** class

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

- What is the difference, if any, if the starred line is written as **pair() {}**
- Define a member for to the **pair** class to enable the following declarations  

```
pair<int,int> x;
pair<long,long> z(x);
```

- 7 Consider the copy assignment operator of the **string** class

```
string string::operator=(const string& rhs)
{
    if (this!=&rhs) {    // *
        delete [] _data;
        _size=rhs._size; _capacity=rhs._capacity;
        _data=strcpy(new char[_capacity+1],rhs._data);
    }
    return *this;
}
```

- As written, the semantics of the assignment of the **string** type is inconsistent with the assignment of any built-in type. What is the inconsistency?
- Modify the code in the starred line so that both **a=a** and **a=b** do nothing to string **a**, where the two strings **a** and **b** are defined by **string a("snoopy"), b(a);**

- 8 Consider the **list** class of HW#7 and the following code

```
int x[3]={1,2,3};
list<list<int> > a;
a.push_back(list<int>(x,x+3));
a.push_back(list<int>(x+1,x+3));
```

- Draw a picture showing the internal structure of the list **a** of int lists.
- Let

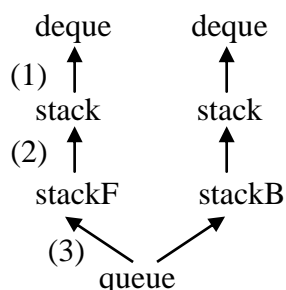
$a = ((x_1, x_2, \dots), (y_1, y_2, \dots), \dots)$

be a list of int lists. Write code to insert the integer 7 between  $y_1$  and  $y_2$

i.e. after the insertion, the list becomes

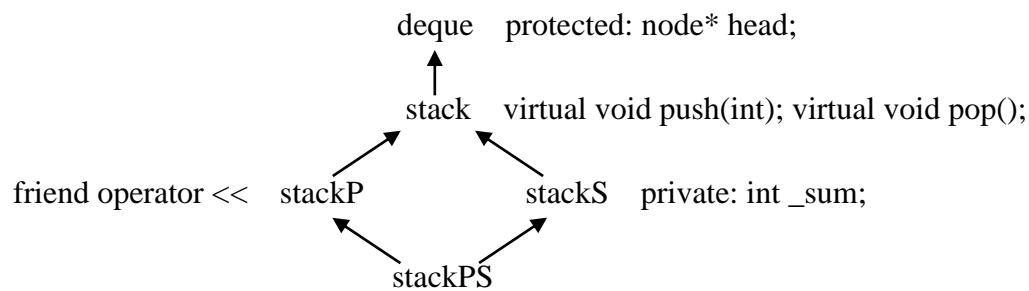
$a = ((x_1, x_2, \dots), (y_1, 7, y_2, \dots), \dots)$

- 9 Consider the class lattice discussed in class



- What kind of inheritance is used for (1)? for (2)? for (3)?
- Define the class **stackF**.

10 Consider the class lattice discussed in class



- Write down the implicitly generated copy ctor for the class **stackPS**.
- Show the implicit casts that occur during the execution of the following code.

```

stackP& s=*new stackPS;
s.push(2);
s.stackP::push(2);

```

11 Consider the following implementation of a stack by a static array

```

class stack {
public:
    stack() : _top(stk) {}
    void push(int n) { *_top=n; ++_top; }
    // other members omitted
private:
    int *_top,stk[80];
};

```

Do we need to define copy ctor for this class? If so, do so. Otherwise, write down the implicitly generated copy ctor.