OOP Midterm

1 Answer the following questions briefly. (16%)

a) Given

```
void p(const void*);
```

and consider the call

```
p(p);
```

Determine the implicit conversion sequence for the argument.

b) Given the following incomplete definition

```
void p() {}
g() { static void (*a[1])()={p}; return a; } // incomplete signature
```

Suppose that what `g` returns is a ***reference*** to array, what is its signature?

c) Consider

1)
```
int* a=static_cast<int*>(operator new(3*sizeof(int)));
for (int i=0;i<3;i++) new (a+i) int(0);
operator delete(a);
```

2)
```
int* a=static_cast<int*>(operator new[](3*sizeof(int)));
for (int i=0;i<3;i++) new (a+i) int(0);
operator delete[](a);
```

As far as the user is concerned, both 1) and 2) have the same effect. But there is a subtle difference in the dynamically allocated storage. What is the difference?

d) Explain why the following function is erroneous.

```
double const& f(int x) { return x; }
```

2 For each problem, give a ***brief explanation*** to your answers. ***No explanation, no credit.*** (20%)

a) Given

```
template<typename T> void p(T) {};     // template 1
template<typename T> void p(T*) {};    // template 2
```

Which template, if any, will be used to instantiate the following call?

```
int a[7];
p(a);
```

b) Given

```
template<typename T> void p(T) {};     // template 1
template<typename T> void p(T&) {};    // template 2
```

Which template, if any, will be used to instantiate the following call?

```
int x;
p(x);
```

2 c) Given

```
template<typename T> struct X { T f(T& x) { return x; } };
```

Which is (are) correct explicit specialization(s) for `T = int`?

1) `template<> struct X<int> { int f(int& x) { return x; } };`
2) `template<> struct X<int> { int& f(int x) { return x; } };`
3) `template<> struct X<int> { int f(int x) { return x; } };`

d) Given

```
template<typename T> T f(T& x) { return x; }
```

Which is (are) correct explicit specialization(s) for `T = int`?

1) `template<> int f<int>(int& x) { return x; }`
2) `template<> int& f<int>(int x) { return x; }`
3) `template<> int f<int>(int x) { return x; }`

e) Given

```
template<typename T> void p(const T&) {}
```

Which is (are) correct explicit specialization(s) for `T = const int*`?

1) `template<> void p(const int*&) {}`
2) `template<> void p(const const int*&) {}`
3) `template<> void p(const int*const&) {}`

3 Consider the following code

```
namespace A {
    float f(float x) { return x+1.0; }        // 1
    double f(double x) { return x+2.0; }      // 2
}
using namespace A;
template<typename T>
T f(T x) { return numeric_limits<T>::is_integer? x: f(x); }  // 3
```

a) Consider the call in function **main**

`f(7.0)`

list all the viable functions and indicate which is the best viable, if any. (4%)

b) Suppose the unqualified call `f(x)` in line 3 is changed to a qualified call `A::f(x).` Explain why this change affects the compilation of the call `f(7)` in function **main**. (4%)

4 a) Fill in the following blanks to complete the code that is meant to binary-search the element **value** in the sequence **[first,last)** ordered by the comparison function **comp** that does < or >. (4%)

Cont'd on the next page

4 a)
```
template<typename T,typename Compare>
bool binary_search(T* first,T* last,const T& value,Compare comp)
{
    if (first==last) return false;
    else {
        int n=last-first;
        T* it=first+n/2;
        if (__(1)__) return true;
        else if (__(2)__)
          return binary_search(first,first+n/2,value,comp);
        else return binary_search(first+n/2+1,last,value,comp);
    }
}
```

b) Given (4%)

```
char* c[6]={"Nationals","Indians","Cubs","Astros","Dodgers","Mets"};
```

Suppose that array **`c`** has been sorted into non-decreasing order by a comparison functor **`less<const char*>()`**, what is wrong with the call

```
binary_search(c,c+6,"Yankees",less<const char*>());
```

Hint: Watch the type of **`"Yankees"`**. Try to convert it.

5 Consider te followng code

```
int c(int m,int n,vector<vector<int> >& cache)                // 1
{
    if (cache[m-n][n]==0)
        cache[m-n][n]=m==n||n==0? 1: c(m-1,n,cache)+c(m-1,n-1,cache);
    return cache[m-n][n];
}
int c(int m,int n)
{
    vector<vector<int> > cache(m-n+1,vector<int>(n+1,0));    // 2
    return c(m,n,cache);                                      // 3
}
```

a) Given the call **`c(5,2)`**, draw a picture showing the internal data structure bound to the vector object **`cache`** created in line 2. You shall also indicate the initial values contained in the structure. (4%)

b) Suppose the vector object is passed by value, i.e. the type declarator **`&`** in line 1 is removed. Does the resulting code still work? Why or why not? (4%)

Cont'd on the next page

5 c) Suppose that lines 2 and 3 are replaced by

```
return c(m,n,vector<vector<int> >(m-n+1,vector<int>(n+1,0)));
```

What else must be changed, if any, so as to make the code work? (4%)

6 Fill in the blanks to complete the code that is meant to use ***backtracking*** to compute $c(m, n)$ defined by: $c(m, n) = 1$, if $m = n$ or $n = 0$

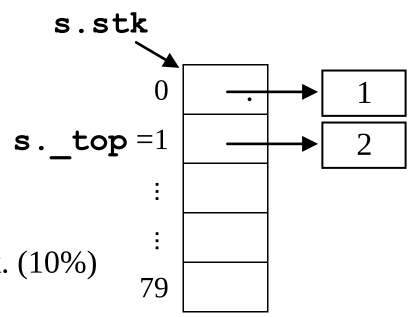
$c(m, n) = c(m–1, n–1) + c(m–1, n)$, otherwise (6%)

```
int c(int m,int n)
{
    stack s;
    int r=0;
    do {
        while (__(1)__) {__(2)__}
        r+=1;
        if (!s.empty()) {__(3)__}
        else return r;
    } while (true);
  }
```

7 The diagram on the r.h.s. depicts the internal strusture of the stack object **s** obtained by executing the following code:

```
stack s; s.push(1); s.push(2)
```

Note that the type of `s.stk` is `int**`, and the pointed-to storage is dynamically alocated.



a) Fill in the blanks to complete the implementation of stack. (10%)

```
class stack {
public:
    stack() : stk(__(1)__),_top(-1) {}
    ~stack() { __(2)__ }
    void push(int n) { __(3)__ }
    void pop() { __(4)__ }
    int& top() { __(5)__ }
    bool empty() const { return _top==-1; }
private:
    int **stk,_top;
};
```

b) Write down a CDT function that is equivalent to the compiled code of the ADT function `bool stack::empty() const { return _top==-1; }` (4%)

8 Consider

```
template<typename T,typename Bfn>
T accumulate(T* first,T* last,T init,Bfn f)
{
    T result=init;
    for (T* it=first;it!=last;++it)
        result=f(result,*it);        // Beware of the order of the arguments
    return result;
}
```

a) Use **accumulate** as well as **numeric_limits** and **max** from the Standard Template Library to define the following ***overloaded*** function template for computing the maximum element of the array **a** of **n** elements of some numeric type **T**.

```
template<typename T>
T max(T* a,int n);
```

Hint: Beware of overloading. Everything in STL locates in the **std** namespace. (4%)

b) Given

```
char* c[6]={"Nationals","Indians","Cubs","Astros","Dodgers","Mets"};
```

Note that there are three strings of even length, namely, `"Cubs"`,`"Astros"`, and `"Mets"`.

Now, let `char r[11]="";`

Define a function

`char* h(char*,char*);` // You are asked to define this ***very short*** function.

so that the call

`accumulate(c,c+6,r,h)`

sets the string **r** to `"3"`, i.e. `atoi(r)` = 3 is the number of even-length strings in array **c** returns **r** as a function value.

Hint: You may use the non-standard function **itoa**.

Example: The call `itoa(5,s,2)` converts 5 to a base-2 integer and stores the result as a string `"101"` in **s**. The string **s** is also returned as a function value.

Hint: Beware of the order of the two arguments in the call `f(result,*it)`. (4%)

9 Given

```
int a[3]={1,2,3};
for (int i=1;i<=3;i++) new (a) int(77);
for (int i=0;i<3;i++) cout << a[i] << " ";
```

a) What is the output of the code? (4%)

b) Suppose the output of the code is `77 77 77`

Define a necessary function to make it work. (4%)

Hint: Define a ***very short*** overloaded `operator new` to manage the storage.