OOP Midterm

1 Write a piece of code that uses STL **sort** and function object to sort the array
```
int a[15]={4,5,2,5,4,3,5,4,1,3,3,5,5,4,2};
```
into non-decreasing order. ***You shall include the necessary headers.*** (4%)

2 What would be the contents of the following array
```
int h[9]={5,6,2,7,9,8,4,1,3};
```
after being turned into a min-heap by the buildheap function of HW#3? (4%)

3 What is wrong with the following function template? **Hint:** Recall HW#1 (4%)
// This function computes $a \times 2^k$. If a is of floating-point type, use multiplication. Otherwise,
// if a is of integral type, use left shift for efficiency.
```
template<typename T>
T multiply(T a,unsigned k)
{
    if (numeric_limits<T>::is_integer) return a<<k;
    else return a*(1<<k);
}
```

4 Write a piece of code to deallocate the storage allocated by the code below. (4%)
```
int **p=new int*(new (operator new(sizeof(int))) int(7));
```

5 The placement new expression **new (operator new[](7*sizeof(int))) int[7]** calls a built-in **operator new** function to obtain storage.
Write down the definition of the built-in **operator new** function. (4%)

6 Write a piece of code that uses STL vector to create a $2 \times 2 \times 2$ vector **v** that contains eight integers of **int** type, all initialized to 2. ***You shall include the necessary header.*** (4%)

7 (Continuing 6) Draw a diagram showing the data structure bound to the vector **v**. (6%)

8 Given
1) **void p(int);**
2) **void p(unsigned);**
Consider **char c; p(c)**.
For each function, state the required implicit conversion sequence. Which, if any, is the best viable function? (4%)

9 Given

```
int a[2]={1,2}; int& f() { return a[0]; }
```

For each expression below, determine if it is legal in C++. ***Explain***. (4%)

1) `++f()`
2) `reinterpret_cast<char(&)[4]>(a)[0]='c'`

10 Given

```
int (*a[2])[3];
```

For the call

```
p(a);
```

what is the deduced type for **T**, for each function template below? (4%)

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

11 Recall that STL contains the function template

```
template<class T>
const T& max(const T& x,const T& y) { return x<y? y: x; }
```

Given the code

```
int z=new int(2);
const double w=std::max<double>(*z,3.4);
```

Draw a diagram showing the storage bound to **x**, **y**, **z** and **w**. (6%)

12 (Continuing 11)

Consider the following code given in the lecture

```
using namespace std;                    // 1
template<typename T>
T max(const T& x,const T& y) { return std::max(x,y); }
```

Can we replace line 1 by

`using std::max;`? ***Why or why not?*** (4%)

13 (Continuing 11 and 12)

Consider the following explicit specialization

```
template<> void max(const int& x,const int& y) { cout << x+y; }
```

Then,

1) It is a specialization of **`std::max`** mentioned in Problem 11.
2) It is a specialization of **`::max`** mentioned in Problem 12.
3) None of the above

Choose one and ***explain***. (4%)

14 For each type below, indicate whether it is legal or not. (4%)

1) `int&[3]` 2) `int(*[3])()` 3) `int*&` 4) `int (&())[2]`

15 The C++ library has a function called **`set_unexpected`** that takes a pointer to a function of type **`void()`** and returns a pointer to a function of the same type.
Write down the prototype of the function **`set_unexpected`**. (4%)

16 Given

```
template<typename T> void p(T) {};      // template A
template<typename T> void p(T&) {};     // template B
int x=2;
```

For each call below, which template, if any, will be chosen for instantiation? ***Explain.*** (4%)

1) `p(x);`
2) `p((int)x);`

17 Consider the following metaprogram

```
template<typename T,int n>
inline int sum(T (&a)[n])
{
    return a[0]+sum(reinterpret_cast<T(&)[n-1]>(a[1]));
}
template<typename T>
T sum(T (&a)[1]) { return a[0]; }
```

Given

```
int a[3]={1,2,3};
cout << sum(a);   // *
```

What does the compiled code of the starred line look like? (4%)

18 (Continuing 17) (4%)
The metaprogram of problem 17 works only for one-dimensional arrays. The following function templates are meant to sum up all the elements of type **`U`** of a k-dimensional array, for any k. Fill in the blank to make it work.

```
template<typename U,typename T,int n>          // U is the element type
inline U sum(T (&a)[n]) { return ______; }     // Fill in this blank
template<typename U,typename T>
inline U sum(T (&a)[1]) { return sum<U>(a[0]); }
template<typename T>
inline T sum(T& a) { return a; }
```

19 The following program separated in 3 files has ***one*** error. Figure it out and ***explain***. (4%)

File `X.h`

```
struct X {
  X();
  int x;
};
X::X() : x(777) {}
```

File 1.cpp

```
#include "X.h"
X x;
```

File 2.cpp

```
#include "X.h"
#include <iostream>
extern X x;
int main()
{ std::cout << x.x; }
```

20 The following ADT stack has ***two*** serious problems. Figure them out and ***explain***. (4%)

```
class stack {
public:
    stack() : _top(80),stk(new int[80]) {}
    void push(int n) { stk[--_top]=n; }
    void pop() { top++; }
    int& top() { return stk[_top]; }
    const int& top() const { return stk[_top]; }   //*
    bool empty() const { return _top==80; }
    int _top,*stk;
};
```

21 (Continuing 20)

Write down a CDT function that is equivalent to the compiled code of the ADT function in the starred line. (4%)

22 Write the function template

```
template<typename T,typename Ufn,typename Bfn>
T faccumulate(T* first,T* last,T init,Ufn p,Bfn f);
```

that behaves like **`accumulate`**, except that it takes one more boolean-valued unary function **`p`** as a parameter and accumulates only those elements x's in the range **`[first,last)`** for which **`p(`**x**`)`**'s are true. (6%)

23 (Continuing 22)

Given

```
int b[20];  // array elements unspecified
```

show how to use **`faccumulate`** to sum up the ***even*** integers in the array **`b`**. (6%)

Requirement

You ***have to*** define a class template that supports an unary **`operator()`** for testing whether an integer is even and write a piece of code to sum up the ***even*** integers in the array **`b`**.