1. (16%) Programming Related Questions

(1) (2%) True or false? A class implementing an abstract data type can have public data members.

True.

(2) (2%) How to let a class provide the access of some of its members to all of its subclasses, but not to other classes?

use protected.

(3) (2%) What is the name of the **Linux** program that you can call to check for memory leak? You only need to write the name of the program. You are not required to tell how it is called.

valgrind

(4) (2%) Consider the following function `foo`:

```
int foo(int a, int b, int c = 3, int d = 1)
{
    return a + b + c + d;
}
```

If `foo` is called as

```
int y = foo(3, -2, 4);
```

a b c

3 - 2 + 4 + 1 = 6

then what is the value of y?

6.

(5) (3%) Given a class `Matrix` which represents a matrix of real values, how do you declare an overloaded addition operator that takes two matrices and returns their sum, as a member function of the class? Assume that the declaration is put inside the class definition.

Matrix& operator=(const Matrix& m);

(6) (2%) Suppose that `Vec` is a templated container over type `T`. However, it stores object by pointers-to-`T`, instead of the actual instance `T`. For the following code, write which of the rules of Existence, Ownership, and Conservation it violates, or "none" if it does not violate any of them.

```
void foo(Vec<int> &l) {
    int x = 3;
    l.insert(&x);
    l.print(); // print list l
}
```

Ownership Rule

(7) (3%) Given the following C++ statements, what is the final value of the variable `x`, `y`, and `z`?

```
int x = 3, y = 1, z = 4;
(z = y = x) *= 4;
```

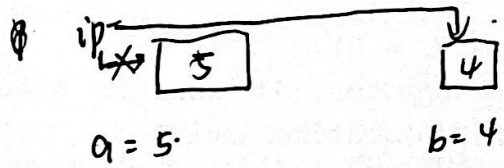
x=12, y=12, z=12

2. (15%) Correctness

For each of the following codes, there is **at most one** major problem. If there is one, write down the problem. You don't need to tell us how to fix it. If there is none, write "None".

(1) (3%)

```
void test()
{
    int *ip = new int(5);
    int a = *ip;
    ip = new int(4);
    int b = *ip;
    b = a*b;
    delete ip;
}
```

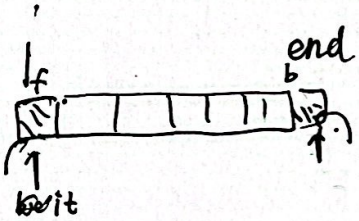


the former memory which contains 5 is not deleted.

(2) (3%)

```
#include <iostream>
#include <list>
using namespace std;

void foo() {
    list<int> lst;
    lst.push_back(5);
    lst.push_front(4);
    list<int>::iterator it;
    for(it = lst.begin(); it < lst.end(); ++it)
        cout << *it << endl;
}
```



None.

(3) (3%)

```
class foo {
   int a;
public:
   foo(int A = 0);
   // Constructor. It assigns A to data member a.
   // Implementation omitted.
   void setVal(int A);
   // MODIFIES: this
   // EFFECTS: sets data member a to A. Implementation omitted.
};

class bar: public foo {
public:
   void play_song();
   // EFFECTS: plays a song. Implementation omitted.
};

void test() {
   foo x;
   bar *pb = &x;
   pb->setVal(1);
}
```

~~the actual type of pb is a pointer to derive class b~~

pb 指 cannot use the setVal() function in foo class

(4) (3%)

```
#include <iostream>
using namespace std;
class myList
{
   // OVERVIEW: a mutable list of ints
public:
   virtual void insert(int v) = 0;
   // MODIFIES: this
   // EFFECTS: add v to the end of the list
   virtual bool query(int v) = 0;
   // EFFECTS: return true if v is in the list, false otherwise
   virtual int  size() = 0;
   // EFFECTS: return the number of elements in the list
};
```

```
void foo()
{
    myList l;
    cout << l.size() << endl;
}
```

the object l can't directly use a pure virtual function.

(5) (3%)

```
class IntSet {
    // OVERVIEW: a mutable set of integers
    int *elts;    // pointer to dynamic array
    int sizeElts; // capacity of the array
    int numElts;  // current occupancy
public:
  IntSet &operator=(const IntSet &is);
    // other unrelated methods omitted.
};

IntSet &IntSet::operator=(const IntSet &is) {
    delete[] elts;
  ① sizeElts = is.sizeElts;
    elts = new int[sizeElts];
  ② for (int i = 0; i < is.sizeElts; i++)
        elts[i] = is.elts[i];
  ③ numElts = is.numElts;
    return *this;
}
```

None

3. (12%) Virtual Function and Inheritance

Given the following class declarations:

```
class Foo {
public:
    void f()
    { cout << "Foo::f" << endl; }
    virtual void g() = 0;
    virtual void h()
    { cout << "Foo::h" << endl; }
};

class Bar: public Foo {
public:
    void g()
    { cout << "Bar::g" << endl; }
    void h()
    { cout << "Bar::h " << endl; }
};

class Baz: public Bar {
public:
    void f()
    { cout << "Baz::f" << endl; }
    void h()
    { Bar::h(); }
};

class Quux: public Baz {
public:
    void h()
    { cout << "Quux::h" << endl; }
};
```

For each of the following commented line of code, write what is printed out at that line.

```
Foo *fp;
Bar bar;
Baz baz;
Quux quux;

bar.f();    // (1)
bar.g();    // (2)

baz.f();    // (3)
baz.h();    // (4)

fp = &bar;
fp->f();    // (5)
fp->g();    // (6)
fp->h();    // (7)

fp = &baz;
fp->f();    // (8)
fp->g();    // (9)
fp->h();    // (10)

Foo &fr = quux;
fr.g();     // (11)
fr.h();     // (12)
```

(1) Foo::f

(2) Bar::g

(3) Baz::f

(4) Bar::h

(5) Foo::f

(6) Bar::g

(7) Bar::h

(8) ~~Foo::f~~ Baz::f

(9) Bar::g

(10) Bar::h

(11) Bar::g

(12) Quux::h.

4. (10%) Binary Tree

Recall from project two the definition of a **binary tree**—a binary tree is well-formed if:

- It is the empty tree, or
- It consists of a node, called the **root** node, plus two children, called the left subtree and the right subtree, each of which is a well-formed binary tree.

A partial definition of the class `BinaryTree` is provided below.

```
class Object
{
    // OVERVIEW: an Object class. Details omitted
    ...
};

struct node
{
    node *left; // Left subtree, NULL if empty
    node *right; // Right subtree, NULL if empty
    Object *val; // A pointer to an Object contained by this node
};

class BinaryTree
{
    // OVERVIEW: contains a binary tree of Objects
private:
    node *root; // The root of the binary tree, NULL if empty

public:
    ...
    BinaryTree(const BinaryTree &tree); // copy constructor
};
```

The class `BinaryTree` is a container. Note that each node of the tree contains only a pointer-to-`Object`, not an instance of `Object`. According to the at-most-once invariant and the existence, ownership, and conservation rules, the `BinaryTree` implementation **owns inserted `Objects`, is responsible for copying them if the tree is copied, and must destroy them if the tree is destroyed.**

Your task is to implement the copy constructor for this ADT. In writing the target function, you may need helper functions. If they are needed, please also write the implementations of these helper functions together with their specification comments. You can assume that the `Object` class has a copy constructor defined. **You can only write your solution on the following lines.**

可以调用 Object(_);

5. (47%) Templated Least Recently Used Set

Many objects exhibit *temporal locality*. For example, you are more likely to access memory region you have recently accessed. In this problem, you will write a *Least Recently Used Set (LruSet)*, which is a templated class that:

- Helps keep track of the least recently accessed element.
- Provides fast removal of the least recently accessed element.

An element is called *the least recently accessed element* if it has not been accessed (queried or inserted) for the longest period among all elements in the set. Similarly, we define *the most recently accessed element* to be the most recently inserted or queried element. For example, if you insert 1, 2, 3 and then query element 2, then the most recently accessed element is 2 and the least recently accessed element is 1. As a result, elements will be removed in the order of 1, 3, and 2.

The `LruSet` ADT will be based on a **double-ended singly-linked list. The basic idea is to put the least recently accessed element in one end and the most recently accessed element in the other end of the linked list.** A partial class definition is as follows:


```
template <class T>
class LruSet {
    // OVERVIEW: a templated least recently used set

    struct node_t {
        node_t* next;
        T value;
    };
    node_t *first;
    node_t *last;

public:
    bool isEmpty() const;
    // EFFECTS: returns true if the set is empty
    //          and false otherwise.

    void insert(const T &v);
    // MODIFIES: this
    // EFFECTS : Queries v first by calling the query function.
    //           Inserts v into the set if and only if v is not
    //           in the set.

    bool query(const T &v);
    // MODIFIES: this
    // EFFECTS : Returns true if v is in the set and false otherwise.
    //           If query succeeds, also make v the most recently
    //           accessed element.

    T remove();
```

```
    // REQUIRES: set is not empty.
    // MODIFIES: this
    // EFFECTS:  Removes and returns the least recently accessed
    //           element from the set.

    LruSet();
    // Default constructor that constructs an empty set.

    LruSet(const LruSet &other);
    // Copy constructor.

    ...
};
```

In this problem, you will complete templated `LruSet` ADT. **For this problem, illegible solutions WILL BE discarded upon grading.** It is your job to think first before writing down anything.

6

(1) (6%) The ADT misses two important methods. Write their **declarations.** Suppose that the declarations are put within the class definition. Briefly describe why they are needed.
Example: if you believe that the class needs an `add` method taking an integer argument and returning an integer, simply write:

```
int add(int arg);
```

① LruSet& operator = (const LruSet &);

② ~LruSet ( );

According to the Big three rule, once one of copy constructor, assignment operator, destructor exists, other two should exist too.

3

(2) (4%) If we remove the `query` method from the ADT and change the `insert` method so that it does not need to do query first and it always inserts the given argument (only for this subproblem), then what data structure you learned in the course does the `LruSet` reduce to? Please briefly justify your answer.

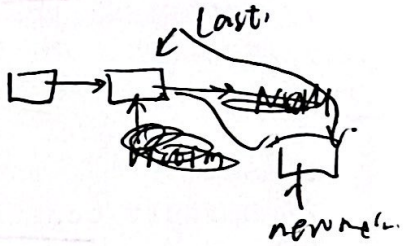
Queue. For example, insert 1, 2, 3. The removal order is 1, 2, 3, too.

3.5

(3) (5%) The `insert` and `remove` functions can be **implemented by simply always inserting elements at one end and removing at the other**. Complete the following `remove` method.

```
...emove() {
node_t* victim = ~~new node_t~~ (3-a) ;
T value = victim->value;
(3-b) = victim->next;
if ( (3-c) ) (3-d) ;
(3-e) ;
return value;
}
```

取出并返回



(3-a) ~~new node_t *first~~ *first 0.5

(3-b) first |

(3-c) ~~victim->next == NULL~~ isEmpty() |

(3-d) first = NULL.

(3-e) delete victim; |

往尾加

(4) (8%) Implement the `insert` method. You may call other class methods if necessary. You may not define any new helper functions. **You can only write your solution on the following lines.**

```
1| void insert (const T& v){
2|   if (!query(v)){
3|     node_t* newnode = new node_t;
4|     ~~last = newnode;~~ last->next = newnode;
5|     last = newnode;
6|     newnode->next = NULL.
7|   } else {
8|     node_t* find.
9|     find->prev = find->next // remove and add to the
10|
11|
12|
13|   }
14|
15|
16|
17|
18|
```

(5) (4%) For the implementation of `LruSet`, why would we prefer a singly-linked list over a doubly-linked list? Why would we prefer a double-ended list over a single-ended list?

① because in query, we can just search for the value in one direction, it'll save some unnecessary pointer node-t* prev

② because we will ~~do some~~ insert at the last position and remove at the first position. It'll be more convenient

(6) (5%) A client uses our implementation as follows:

```
LruSet<int> set;
set.insert(1); set.insert(2); set.insert(3); set.insert(4);
set.query(5); set.query(1); set.remove(); set.query(4);
set.insert(2); set.query(3);
```

After the execution of the above code, based on the provided information, please label each node with the integer it contains in the following figure. **Circle the node pointed by** `last`. Note that there might be more blanks than you actually need.

`first` -> 1 -> 4 -> 2 -> 3 -> last.

(7) (15%) Write the implementation for the method `query`. As copying unknown type `T` might bear a huge performance cost, your implementation should avoid copying instances of `T` as much as possible. You may call other class methods if necessary. You may not define any new helper functions. **You can only write your solution on the following lines.**