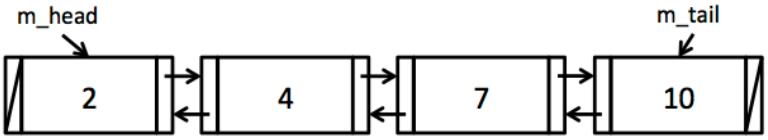
Midterm Practice

TA: Brian Choi (schoi@cs.ucla.edu)
Section Webpage: `http://www.cs.ucla.edu/~schoi/cs32`

*** Make sure you try all exercises by hand! You won't have access to Visual C++ during the exam. ***

1. We will build a sorted doubly linked list without sentinel (dummy) nodes in this exercise. The following shows an example of a list with 4 elements, where the nodes are sorted in the increasing order of their values. The `m_prev` pointer of the head node and `m_next` pointer of the tail node both point to `NULL`. If the list is empty, head and tail pointers both point to `NULL`.



Assume the following declaration of `Node` structure and `SortedLinkedList` class.

```
struct Node
{
    ItemType m_value;
    Node *m_prev;
    Node *m_next;
};
```

```
class SortedLinkedList
{
    public:
        SortedLinkedList();
        bool insert(const ItemType &value);
        Node *search(const ItemType &value) const;
        void remove(Node *node);
        int size() const { return m_size; }
        void printIncreasingOrder() const;
    private:
        Node *m_head;
        Node *m_tail;
        int m_size;
};
```

(a) Implement `SortedLinkedList()`.

```
SortedLinkedList::SortedLinkedList()
{









}
```


(b) Implement `insert()`. If a node with the same value is already in the list, do not insert a new node. Return true if a new node is successfully inserted, and return false otherwise. You may assume that `ItemType` has `<`, `>`, and `==` operators properly implemented.

```
bool SortedLinkedList::insert(const ItemType &value)
{

}
```

(c) Implement `search()`, which returns the pointer to the node with the specified value.

```
Node *SortedLinkedList::search(const ItemType &value) const
{

}
```


(d) Implement `remove()`. Assume `node` is either `NULL` (in which case you would simply return) or a valid pointer to a Node in the list, as found in `search()`.

```
void SortedLinkedList::remove(Node *node)
{









}
```

(e) Implement `printIncreasingOrder()`, which prints the values stored in the list in the increasing order, one value in each line.

```
void SortedLinkedList::printIncreasingOrder() const
{









}
```

(f) The public interface of `SortedLinkedList` has a problem. More precisely, the user of this class can possibly break the integrity of the sorted linked list, only using the public interface of `SortedLinkedList`. Demonstrate this problem with an example. Also, suggest a fix, if you have an idea.

2. The following code prints all elements in the array `a[]` in the order they appear in the array. What is a simple change you can make to the code, such that the elements are printed in the reverse order?

```
1  void printArrayInOrder(const double a[], int n)
2  {
3      if (n == 0)
4          return;
5
6      cout << a[0] << endl;
7      printArrayInOrder(a + 1, n - 1);
8  }
```

3. Given two positive integers `m` and `n` such that `m < n`, the greatest common divisor of `m` and `n` is the same as the greatest common divisor of `m` and `n-m`. Use this fact to write a recursive function `gcd()`. (Suggestion: try a few examples on paper prior to writing code.)

```
int gcd(int m, int n)
{









}
```

4. Write a function `powerOfTwo` that, given a non-negative number `x`, returns 2^x (2^x, or "2 raised to power x") recursively, assuming 2^x is something that can be represented as an integer. Do not use a loop, and do not use the character `'*'` anywhere in your code.

```
int powerOfTwo(int x)
{









}
```


5. Consider the following program.

```
class A
{
   public:
      A() : m_msg("Apple") {}
      A(string msg) : m_msg(msg) {}
      virtual ~A() { message(); }
      void message() const
      {
         cout << m_msg << endl;
      }
   private:
      string m_msg;
};
```

```
class B : public A
{
   public:
      B() : A("Orange") {}
      B(string msg) : A(msg), m_a(msg) {}
      void message() const
      {
          m_a.message();
      }
   private:
      A m_a;
};
```

```
int main()
{
   A *b1 = new B;
   B *b2 = new B;
   A *b3 = new B("Apple");
   b1->message();
   b2->message();
   b3->message();
   delete b1;
   delete b2;
   delete b3;
}
```

How many times will you see the word `Apple` in the output? ____

How about `Orange`? ___

Now make `A`'s `message()` virtual, i.e.,

```
virtual void message() const;
```

How many times will you see the word `Apple` in the output? ___

How about `Orange`? ___

6. Using a stack, write a function that takes in an infix arithmetic expression `exp`, which may involve parentheses (`(`, `)`), curly braces (`{`, `}`), and square brackets (`[`, `]`), and returns true if they are balanced, false otherwise. If the expression does not include any parentheses, curly braces, or square brackets, it should return true.

For example:

`(2 + 4) * 6`	balanced
`[(2 + 4) * {15 - 20}]`	balanced
`({12+30]]`	not balanced
`(({{[[<<<_*_>>>]]}}))`	balanced
`(((( ))))))))))`	not balanced

```
#include <stack>

using namespace std;

bool balanced(const string &exp)
{









}
```

