# Midterm Practice Solutions

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

1. (a)

```
SortedLinkedList::SortedLinkedList()
{
    m_head = m_tail = NULL;
    m_size = 0;
}
```

(b)

```
bool SortedLinkedList::insert(const ItemType &value)
{
    Node *p = m_head;
    Node *q = NULL;

    while (p != NULL)               // Find the first element with a greater value
    {                               // than the input value.
        if (value == p->m_value)
            return false;

        if (value < p->m_value)
            break;

        q = p;
        p = p->m_next;
    }

    Node *newNode = new Node;       // The new node must sit between q and p.
    newNode->m_value = value;
    newNode->m_next = p;
    newNode->m_prev = q;

    if (p != NULL)                  // Is there a following node?
        p->m_prev = newNode;
    else
        m_tail = newNode;

    if (q != NULL)                  // Is there a preceding node?
        q->m_next = newNode;
    else
        m_head = newNode;

    m_size++;
}
```

Copyright Brian Choi 2011. All Rights Reserved.

(c)

```
Node *SortedLinkedList::search(const ItemType &value) const
{
    for (Node *p = m_head; p != NULL; p = p->m_next)
    {
        if (p->m_value == value)
            return p;
    }
    return NULL;
}
```

(d)

```
void SortedLinkedList::remove(Node *node)
{
    if (node == NULL)
        return;

    if (node != m_head)
        node->m_prev->m_next = node->m_next;
    else
        m_head = m_head->m_next;

    if (node != m_tail)
        node->m_next->m_prev = node->m_prev;
    else
        m_tail = m_tail->m_prev;

    delete node;
    m_size--;
}
```

(e)

```
void SortedLinkedList::printIncreasingOrder() const
{
    for (Node *p = m_head; p != NULL; p = p->m_next)
        cout << p->m_value << endl;
}
```

Copyright Brian Choi 2011. All Rights Reserved.

(f)

See the following example.

```
SortedLinkedList linkedList;
linkedList.insert(20);
linkedList.insert(30);
linkedList.insert(40);

Node *p = linkedlist.search(30);
p->m_value = 100;
```

What will the list look like? What will you see if you call `printIncreasingOrder()`?

You can add a protection by making the returned Node pointer constant.

```
const Node *search(const ItemType &value) const;
```

2.

Swap lines 6 and 7 to get:

```
printArrayInOrder(a + 1, n - 1);
cout << a[0] << endl;
```

This is like saying "print all the elements in the rest of the array, and then print the first one."
Some of you may say you can apply recursion backwards, which is also a valid solution.

3.

```
int gcd(int m, int n)
{
    if (n == m)
        return m;

    n = n - m;

    if (m < n)
        return gcd(m, n);

    return gcd(n, m);
}
```

Can you improve this by exploiting the fact that greatest common divisor of `m` and `n` (where `m < n`) is equal to the greatest common divisor of `(n % m)` and `m`?

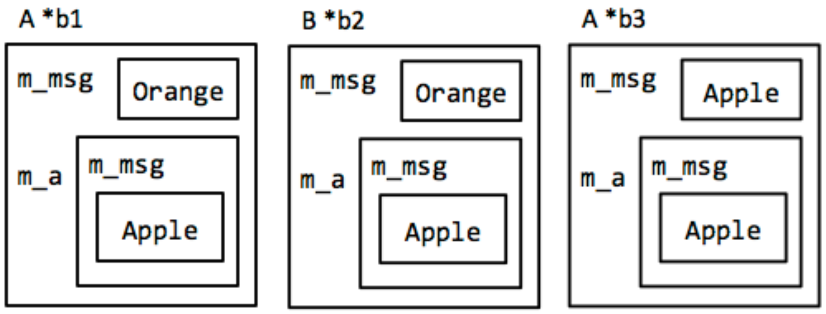
4. Noting the fact that $2^x = 2^{x-1} + 2^{x-1}$:

```
int powerOfTwo(int x)
{
    if (x == 0)
        return 1;

    return powerOfTwo(x - 1) + powerOfTwo(x - 1);
}
```

Of course, this is SLOW! You would never implement this function this way. This is just for practice.

5.



If `A::message()` is not `virtual`, you get:

```
Orange    // b1->message()
Apple     // b2->message()
Apple     // b3->message()
Apple     // destroying m_a of b1
Orange    // destroying b1
Apple     // destroying m_a of b2
Orange    // destroying b2
Apple     // destroying m_a of b3
Apple     // destroying b3
```

6 `Apple`'s and 3 `Orange`'s

If `A::message()` is `virtual`, you get:

```
Apple
Apple
Apple
Apple
Orange
Apple
Orange
Apple
Apple
```

7 `Apple`'s and 2 `Orange`'s

6.

```
bool balanced(const string &exp)
{
    stack<char> parenStack;

    for (int i = 0; i < exp.size(); ++i)
    {
        char ch = exp[i];

        switch(ch)
        {
            case '(':
            case '{':
            case '[':
                parenStack.push(ch);
                break;

            case ')':
            case '}':
            case ']':
                if (parenStack.empty())  // Extra closed paren?
                    return false;
                if (parenStack.top() == '(' && ch == ')' ||
                    parenStack.top() == '{' && ch == '}' ||
                    parenStack.top() == '[' && ch == ']')
                {
                    parenStack.pop();
                }
                else
                {
                    return false;        // Mismatch?
                }
                break;
        }
    }

    if (!parenStack.empty())             // Extra open paren?
        return false;

    return true;
}
```

Copyright Brian Choi 2011. All Rights Reserved.