

# CSCI-1200 Computer Science II — Spring 2006

## Test 1 — Practice Problems

### General Information

- Test 1 will be held **Tuesday, February 7th, 2006 10-11:50am, West Hall Auditorium**. No make-ups will be given except for emergency situations, and even then a written excuse from the Dean of Students office will be required.
- Coverage: Lectures 1-6, Labs 1-3, HW 1-2.
- Closed-book and closed-notes *except for 1 sheet of 8.5x11 inch paper (front & back) that may be hand-written or printed*. Computers, cell-phones, palm pilots, calculators, PDAs, etc. are not permitted and must be turned off.
- All students must bring their Rensselaer photo ID card.
- Below are relevant sample questions from previous tests. The best thing you can do to prepare for this test is practice. Try these problems with pencil & paper first. Then practice programming (with a computer) these exercises and other exercises from lecture, lab, homework and the textbook.

### Practice Problems

1. Write a code segment that copies the contents of a string into a vector of char in reverse order.

**Solution:**

```
// assume the string is s and s has been initialized
vector<char> result;
for ( int i=s.size()-1; i>=0; --i )
    result.push_back( s[i] );
```

2. Write a function that takes a vector of strings as an argument and returns the number of vowels that appear in the string. A vowel is defined as an 'a', 'e', 'i', 'o' or 'u'. For example, if the vector contains the strings:

```
abe
lincoln
went
to
the
white
house
```

Your function should return the value 12. You may assume that all letters are lower case. Here is the function prototype:

```
int count_vowels( const vector<string>& strings )
```

**Solution:**

```
int count_vowels( const vector<string>& strings ) {
    int count = 0;
    for ( unsigned int i=0; i<strings.size(); ++i )
        for ( unsigned int j=0; j<strings[i].size(); ++j )
            if ( strings[i][j] == 'a' || strings[i][j] = 'e' ||
                strings[i][j] == 'i' || strings[i][j] = 'o' ||
```

```

        strings[i][j] == 'u' )
        ++count;
    return count;
}

```

3. Write a function called `less_string` that mimics the effect of the `<` operator on strings. In other words, given strings `a` and `b`, `less_string( a, b)` should return `true` if and only if `a < b`. You may not use `<` on the strings, but you may use `<` on individual characters in the strings. Start by getting the function prototype correct. Here are examples of pairs for which your function should return true:

```

a = "abc", b = "abd"
a = "cab", b = "cabbage"
a = "christine", b = "christopher"

```

### Solution:

```

bool less_string( const string& a, const string& b ) {
    unsigned int min_length = a.size();
    if ( b.size() < min_length ) min_length = b.size();
    while ( i < min_length ) {
        if ( a[i] < b[i] )
            return true;
        else if ( a[i] > b[i] )
            return false;
        ++ i;
    }
    return a.size() < b.size();
}

```

4. What is the output from the following program? We strongly suggest that you draw the contents of the vectors to help you visualize what is happening.

```

void more_confused( vector<int> a, vector<int> & b ) {
    for ( unsigned int i=0; i<2; ++i ) {
        int temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
    cout << "1: ";
    for ( unsigned int i=0; i<a.size(); ++i )
        cout << a[i] << " ";
    cout << endl;
    cout << "2: ";
    for ( unsigned int i=0; i<b.size(); ++i )
        cout << b[i] << " ";
    cout << endl;
}

int main() {
    vector<int> a, b;
    a.push_back(1); a.push_back(3); a.push_back(5);
    b.push_back(2); b.push_back(4);

    more_confused( a, b );
    a[0] = 7; a[1] = 9;
    more_confused( b, a );
}

```

```

    cout << "3: ";
    for ( unsigned int i=0; i<a.size(); ++i )
        cout << a[i] << " ";
    cout << endl;

    cout << "4: ";
    for ( unsigned int i=0; i<b.size(); ++i )
        cout << b[i] << " ";
    cout << endl;

    return 0;
}

```

**Solution:**

```

1: 2 4 5
2: 1 3
1: 7 9
2: 1 3 5
3: 1 3 5
4: 1 3

```

5. Write a function that takes a vector of doubles and copies its values into two vectors of doubles, one containing only the negative numbers from the original vector, the other containing only the positive numbers. Values that are 0 should not be in either vector. For example, if the original vector contains the values:

-1.3, 5.2, 8.7, -4.5, 0.0, 7.8, -9.1, 3.5, 6.6

then the resulting vector of negative numbers should contain:

-1.3, -4.5, -9.1

and the resulting vector of positive numbers should contain:

5.2, 8.7, 7.8, 3.5, 6.6

Start this problem by writing the function prototype as you think it should appear and then write the code.

**Solution:**

```

void copy_pos_neg( const vector<double>& original,
                  vector<double>& negatives,    // passing by reference is required
                  vector<double>& positives )   // passing by reference is required
{
    negatives.clear(); positives.clear(); // not necessary
    for ( unsigned int i=0; i<original.size(); ++i ) {
        if ( original[i] < 0 )
            negatives.push_back( original[i] );
        else if ( original[i] > 0 )
            positives.push_back( original[i] );
    }
}

```

6. Give an order notation (“O”) estimate of the worst-case number of operations required by the following sorting function. Briefly justify your answer:

```

void ASort( vector<double>& A ) {
    int n = A.size();
    for( int i=0; i<n-1; ++i ) {
        // Find index of next smallest value
        int small_index = i;
        for ( int j=i+1; j<n; ++j ) {
            if ( A[j] < A[small_index] )
                small_index = j;
        }

        // Swap with A[i]
        double temp = A[i];
        A[i] = A[small_index];
        A[small_index] = temp;
    }
}

```

**Solution:** This is  $O(n^2)$ . Clearly the outer loop requires  $n - 1$  iterations. Each iteration of the outer loop has a constant number of operation before and after the inner loop. The inner loop has  $n - i - 1$  iterations, which is at most  $n - 1$ . Each iteration requires constant time. Thus, each iteration of the outer loop is  $O(n)$ , giving a total of  $O(n^2)$  for all iterations.

7. Write a function that takes an array of floats and copies the entries that have an **even** subscript to a vector of floats, doing so **in reverse order**. For example, given an array containing the 8 values:

5.1, -1.7, -1.4, 0.4, 13.2, 1.5, 11.3, 3.4

the vector should contain the values (in order):

11.3, 13.2, -1.4, 5.1

after the function is completed. You may assume that the vector of floats is initially empty (the `size()` member function returns 0). The return type of the function must be `void`. Start by specifying the function prototype. Think carefully about the parameters needed and their types.

**Solution:**

```

void array_to_vector( float a[], int n, vector<float> & v ) {
    int start;
    if ( n%2 == 0 )
        start = n-2;
    else
        start = n-1;
    for ( int i=start; i>=0; i-=2 )
        v.push_back( a[i] );
}

```

8. Consider the following declaration of a `Point` class, including an associated non-member function:

```

class Point {
public:
    Point();
    Point( double in_x, double in_y, double in_z );
    void get( double & x, double & y, double & z ) const;
    void set( double x, double y, double z );
    bool dominates( const Point & other );
private:
    double px, py, pz;
};

bool dominates_v2( const Point & left, const Point & right );

```

- (a) Provide the implementation of the default constructor — the constructor that takes no arguments. It should assign 0 to each of the member variables.

**Solution:**

```
Point::Point()
{
    xp = yp = zp = 0.0;
}
```

- (b) The member function `dominates` should return `true` whenever each of the point's coordinates is greater than or equal to each of the corresponding coordinates in the other point. For example, given

```
Point p( 1.5, 5.0, -1 );
Point q( 1.4, 5.0, -3 );
Point r( -3, 8.1, -7 );
```

Then

```
p.dominates( q )
```

should return `true` but the function calls

```
p.dominates( r )    r.dominates( q )    q.dominates(r)
```

should each return `false`. Write member function `dominates`.

**Solution:**

```
bool
Point::dominates( const Point& other )
{
    return xp >= other.xp && yp >= other.yp && zp >= other.zp;
}
```

- (c) The function `dominates_v2` should be a non-member function version of `dominates`. Its behavior should be essentially the same as the member function version, so that for the `Point` objects defined in (b):

```
dominates_v2( p, q )
```

should return `true` but the function calls:

```
dominates_v2( p, r )    dominates_v2( r, q )    dominates_v2( q, r)
```

should each return `true`. Write `dominates_v2`.

**Solution:**

```
bool dominates_v2( const Point & left, const Point & right ) {
    double xpl, ypl, zpl;
    left.get( xpl, ypl, zpl );
    double xpr, ypr, zpr;
    right.get( xpr, ypr, zpr );
    return xpl >= xpr && ypl >= ypr && zpl >= zpr;
}
```

9. Write a complete program (you don't need to provide the `#include` statements) that (a) reads in a sequence of strings until a string starting with the letter 'x' is read, and (b) then reads in another sequence of strings until the end of input is reached. For each string in the second sequence, the program should output the string and the number of times it occurs in the first sequence. For example, given the input:

```
hello now good now no hello first last second won now
song xray now silly first
```

The program should output:

```
now 3
silly 0
first 1
```

You should write functions as appropriate to make your program clean and modular.

**Solution:**

```
int
count_in_words( const vector<string> & words,
                const string & one_word )
{
    int count = 0;
    for ( unsigned int i=0; i<words.size(); ++i )
        if ( one_word == words[i] ) count ++;
    return count;
}

int main()
{
    vector<string> words;
    string in_word;
    bool done = false;

    while ( !done && (cin >> in_word) )
    {
        done = in_word[0] == 'x';
        if ( !done ) words.push_back( in_word );
    }

    while ( cin >> in_word )
        cout << in_word << ' ' << count_in_words( words, in_word ) << endl;

    return 0;
}
```