Computer Science II
Lecture 2
Introduction and Review
Exercise Solutions

1. From **Section 2.3 Exercise**

   (a) Consider the following code fragment:

   ```
   std::string a, b, c;
   std::cin >> a >> b >> c;
   ```

   and the input

   ```
   all-cows   eat123
         grass.  every good boy
   deserves fudge!
   ```

   What will be the values of strings `a`, `b` and `c` at the end of the code fragment?
   Solution: The effect is as though we assigned

   ```
   a = "all-cows";
   b = "eat123";
   c = "grass.";
   ```

   If you are unsure of why this occurs, review the rules on string input from the lecture notes.

   (b) Write a C++ program that reads in two strings, outputs the shorter string on one line of output, and then outputs the two strings concatenated together with a space between them on the second line of output.
   Solution:

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

   int main()
   {
     string s1, s2;
     cin >> s1 >> s2;

     if ( s1.length() < s2.size() )
       cout << s1 << endl;
     else
       cout << s2 << endl;

     string result = s1 + '  ' + s2;
     cout << result << endl;
     return 0;
   }
   ```

2. **Section 3.10 Exercises**

   (a) After the above code constructing the three vectors, what will be output by the following statement?

   ```
   cout << a.size() << endl
        << b.size() << endl
        << c.size() << endl;
   ```

   Solution:

   ```
   0
   100
   10000
   ```

   (b) Write code to construct a vector containing 100 doubles, each having the value 55.5.

   Solution:

   ```
   vector<double> f(100, 55.5);
   ```

   (c) Write code to construct a vector containing 1000 doubles, containing the values 0, 1, $\sqrt{2}$, $\sqrt{3}$, $\sqrt{4}$, $\sqrt{5}$, etc. Write it two ways, one that uses `push_back` and one that does not use `push_back`.

   Solution:

   ```
   vector<double> g;
   for ( unsigned int i=0; i<1000; ++i ) g.push_back( sqrt(i) );

   vector<double> h;
   for ( unsigned int i=0; i<1000; ++i ) h[i] = sqrt(i);
   ```

3. **4.5 Exercise**: Write an iterative version of `intpow`.

   Solution:

   ```
   int intpow( int n, int p )
   {
     int sol = 1;
     for ( unsigned int i=1; i<=p; ++i )
       sol *= p;
     return sol;
   }
   ```

4. **Exercises from Section 4.7**

   What will `print_vec` print when called in the following code?

   ```
   int main()
   {
      vector<int> a;
      a.push_back( 3 );  a.push_back( 5 );  a.push_back( 11 );
      a.push_back( 17 );
      print_vec( a );
   }
   ```

Solution:

```
0: 3
1: 5
2: 11
3: 17
```

How can you change the second `print_vec` function as little as possible to write a recursive function to print the contents of the vector in reverse order?

Solution: Reverse the two lines inside the `if`:

```
print_vec( v, i+1 );
cout << i << ": " << v[i] << endl;
```

5. **4.9 Exercises**

   (a) Write a non-recursive version of binary search

   Solution:

```
-----------------------------------------------------

bool binsearch( const vector<double>& v, double x )
{
  int low = 0;
  int high = v.size()-1;

  while ( low < high )
    {
      int mid = (low+high)/2;

      if ( x <= v[mid] )
        high = mid;
      else
        low = mid+1;
    }

  return x == v[low];
}

-----------------------------------------------------
```

   (b) If we replaced the if-else structure inside the recursive binsearch function (above) with

```
if ( x < v[mid] )
  return binsearch( v, low, mid-1, x );
else
  return binsearch( v, mid, high, x );
```

would the function still work correctly?

Solution: No! The function will tend to go into an infinite recursion. Consider the example when `v` contains just 3.1 and 4.1, `x = 3.9`. Then the function will continue to call itself over and over with `low = 0` and `high = 1`. This is a subtle issue because it only shows up when the search interval gets down to 2 locations. Getting the details right is hard!