1. **(30 points)** Write a complete C++ program that reads in a sequence of words from cin and outputs both the first word in alphabetical order and the last word in alphabetical order. The words will be mixtures of capital and small letters. In order to properly alphabetize the words, therefore, you must convert everything to small letters (or to capital letters). There will be no digits and no punctuation in the input, only letters. As an example, given the following input

```
hello good darn right bye Zero
Kansas Toto Dorothy
```

Your program should output:

**First word:** bye  
**Last word:** zero

You may safely assume there is at least one word in the input.

**Solution:** Here is a solution without sorting. It is more efficient than a solution based on sorting, but this is not a major concern here.

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

void convert_to_smalls( string& s )
{
    for( unsigned int i = 0; i<s.size(); ++i )
        s[i] = tolower( s[i] );
}

int main()
{
    bool is_start = true;
    string first, last, s;
    while ( cin >> s )
    {
        convert_to_smalls( s );
        if ( is_start )
            { first = last = s; is_start = false; }
        else if ( s < first )
            first = s;
        else if ( s > last )
            last = s;
    }
}
```
cout << "First word: " << first << "\nLast word: " << last << endl;
return 0;
}
Here is a solution based on sorting:

```cpp
#include <algorithm>
#include <cctype>
#include <vector>
#include <iostream>
#include <string>
using namespace std;

void convert_to_smalls( string& s )
{
    for( unsigned int i = 0; i<s.size(); ++i )
        s[i] = tolower( s[i] );
}

int main()
{
    bool is_start = true;
    string s;
    vector<string> words;
    while ( cin >> s )
    {
        convert_to_smalls( s );
        words.push_back( s );
    }
    sort( words.begin(), words.end() );

    cout << "First word: " << words.front() << "\nLast word: " << words.back() << endl;
    return 0;
}
```
2. **(12 points)** Show the output from the following code segment. Note that the numbers 1:, 2:, etc. in the cout statements do not necessarily reflect the order of output.

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

void foo( int& a, int& b )
{
    a = 15; b = 16;
    cout << "1: " << a << " " << b << endl;
}

void bar( int& a, int b )
{
    int t = a; a = b; b = t;
    cout << "2: " << a << " " << b << endl;
}

int main()
{
    int x = 10, y = 20;
    bar( x, y );
    cout << "3: " << x << " " << y << endl;
    foo( y, x );
    cout << "4: " << x << " " << y << endl;
    return 0;
}

Solution:

2: 20 10
3: 20 20
1: 15 16
4: 16 15
3. **(13 points)** Show the output from the following code segment. Hint: we strongly suggest that you carefully draw the contents of the lists and use this to figure out what happens with the iterators and iterator operations.

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

int
main()
{
    list<int> a, b;
a.push_back( 15 ); a.push_back( 12 ); a.push_back( 13 );
b.push_back( 2 ); b.push_back( 8 ); b.push_back( 4 ); b.push_back( 6 );

    list<int>::iterator p = a.begin();
    list<int>::iterator q = b.begin();
    ++ q;

    int temp = *q;
    *q = *p;
    *p = temp;

    p = q;
p ++;
*p = 20;
q --;
*q = 11;

for ( p = a.begin(); p != a.end(); ++p )
    cout << *p << " ";
cout << endl;

for ( p = b.begin(); p != b.end(); ++p )
    cout << *p << " ";
cout << endl;
}

Solution:

8 12 13
11 15 20 6
4. Here is the Course struct from the posted solution to the Week 4 homework.

```c++
struct Course {
    string course_id;
    list<string> students;
    unsigned int max_students;
};
```

Use this in solving each of the following problems.

(a) (15 points) Write a function that sorts a vector of Course structs by increasing enrollment. In other words the course having the fewest students should be first. If two courses have the same number of students, the course with the smaller maximum number of students allowed should be listed first. You will need to create a separate bool function to compare two Courses.

Solution:

```c++
bool CompareEnroll( const Course& c1, const Course& c2 )
{
    return c1.students.size() < c2.students.size() ||
           ( c1.students.size() == c2.students.size() &&
             c1.max_students < c2.max_students );
}

void OrderByEnrollement( vector<Course>& classes )
{
    sort( classes.begin(), classes.end(), CompareEnroll );
}
```
(b) **(15 points)** Write a function that takes as parameters a const vector of Course structs, two const strings representing two different student ids (you can assume that they are in fact different), and a vector of strings. The function should find the ids of each course in which BOTH students are enrolled, and store these ids in the vector of strings. Here is the function prototype:

```cpp
void both_students( const string& stu1, const string& stu2, const vector<Course>& courses, vector<string>& both )
```

Hint: using the `find` function (twice) will make this much easier.

**Solution:**

```cpp
{ 
 for ( vector<Course>::const_iterator c_itr = courses.begin(); 
      c_itr != courses.end(); ++ c_itr )
 {
   if ( find( c_itr->students.begin(), c_itr->students.end(), stu1 ) !=
       c_itr->students.end() && 
       find( c_itr->students.begin(), c_itr->students.end(), stu2 ) !=
       c_itr->students.end() )
     both.push_back( c_itr -> course_id );
 }
}
```
5. **(15 points)** Write a function that takes a string as its only argument and returns as its result a new string that contains the letters that appear in the string in alphabetic order, with no repetitions. For simplicity, you can assume that all letters in the string are lower case, but you must be able to handle whitespace and punctuation characters. For example, given the string

```plaintext
hello, how are you?
```

the function should return the string

```plaintext
aehlorwuy
```

Here is the function prototype:

```plaintext
string letters( const string& s )
```

**Solution:**

```plaintext
{
    string res;
    for ( unsigned int i=0; i<s.size(); ++i )
        if ( isalpha( s[i] ) && find( res.begin(), res.end(), s[i] ) == res.end() )
            res += s[i];
    sort( res.begin(), res.end() );
    return res;
}
```