IN2029 Programming in C++
Solutions to Exercises 2

1. The additional code (inside the `if` statement) is

```
    // compute the average
    double sum = 0;
    for (vec_size i = 0; i < v.size(); ++i)
        sum += v[i];
    cout << "average = " << sum/n << '\n';
```

   Recall the `vec_size` has been defined as an alias of the unsigned type of the value returned by `v.size()`.

   Could we have used `auto` instead of `vec_size` here? That is,

```
    for (auto i = 0; i < v.size(); ++i)
```

   The type of `i` here will be the type of `0`, that is, `int`. That makes the comparison `i < v.size()` a comparison of an `int` and a value of a signed type, with the potential pitfalls discussed in the lecture. In this case, we know that `i` will never be negative, so it produces the correct result, but some compilers will warn about it.

2. Sorting the vector puts the values we want to ignore on the ends, and then we can just examine the rest.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// scoring by ignoring the highest and lowest and averaging
        the rest
int main() {
    // read numbers from the standard input
    // and store them in a vector
    cout << "Please enter a series of numbers\n";
    double x;
    vector<double> v;
    while (cin >> x)
        v.push_back(x);
```

```
    auto n = v.size();

    // compute the score
    if (n > 2) {
        sort(v.begin(), v.end());

        // average all but the lowest and highest numbers
        double sum = 0;
        using vec_size = vector<double>::size_type;
        for (vec_size i = 1; i < n-1; ++i)
            sum += v[i];
        cout << "score = " << sum/(n-2) << '\n';
        return 0;
    } else {
        cout << "Not enough to score\n";
        return 1;
    }
}
```

3. This is similar to the word-reading program in the slides, except that the body of the loop updates a count, which is printed at the end.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    int count;
    string s;
    count = 0;
    while (cin >> s)
        ++count;
    cout << count << '\n';
    return 0;
}
```

4. We need to retain the longest word we have seen so far, in the variable **`longest`**.

```
#include <iostream>
#include <string>

using namespace std;
```

```
int main() {
    string longest;
    string w;
    while (cin >> w)
        if (w.size() > longest.size())
            longest = w;
    cout << longest << '\n';
    return 0;
}
```

5. There are a couple of ways of doing this. This one pops words from the end of the vector until it is empty:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    // read all the words
    vector<string> v;
    string s;
    while (cin >> s)
        v.push_back(s);

    // print the words in reverse order
    while (v.size() > 0) {
        cout << v.back() << '\n';
        v.pop_back();
    }
    return 0;
}
```

An alternative is to access the elements from position `v.size()-1` down to 0:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
```

```
    // read all the words
    vector<string> v;
    string s;
    while (cin >> s)
        v.push_back(s);

    // print the words in reverse order
    for (vector<string>::size_type i = 1; i <= v.size(); ++
          i)
        cout << v[v.size() - i] << '\n';
    return 0;
}
```

Alternatively, the last loop could been written:

```
    for (vector<string>::size_type i = v.size(); i > 0; --i
          )
        cout << v[i-1] << '\n';
```

or even

```
    for (auto i = v.size(); i > 0; --i)
        cout << v[i-1] << '\n';
```

because here the type of `i` is the type of `v.size()`.

However, it could **not** be written as

```
    // this will probably cause a memory error
    for (vector<string>::size_type i = v.size()-1; i >= 0;
          --i)
        cout << v[i] << '\n';
```

or

```
    for (auto i = v.size()-1; i >= 0; --i)
        cout << v[i] << '\n';
```

This is because the type of `i` is unsigned (whether we declare it or have it inferred by `auto`), which means that the test `i >= 0` always returns true, so after `i` hits 0 it will become a very large unsigned number. See the caution on unsigned types in the lecture for more on this issue.

6. The idea is that if we keep track of the smallest and largest values encountered, we can subtract them from the total at the end.

```
#include <iostream>

using namespace std;

// scoring by ignoring the highest and lowest and averaging
        the rest
int main() {
    cout << "Please enter a series of numbers\n";
    double x;
    if (cin >> x) {
        int count = 1;
        double sum = x;
        double least = x;
        double greatest = x;
        while (cin >> x) {
            ++count;
            sum += x;
            if (x > greatest)
                greatest = x;
            if (x < least)
                least = x;
        }
        if (count > 2) {
            double score = (sum - greatest - least)/(count
                   -2);
            cout << "score = " << score << '\n';
        } else
            cout << "Not enough to score";
    } else
        cout << "Not enough to score";
    return 0;
}
```