Exercise 3.1

```
// sum_50_100_using.cpp

#include <iostream>

using std::cout;
using std::endl;

int main()
{
    unsigned sum = 0;
    for (unsigned i = 50; i <= 100; ++i)
        sum += i;
    cout << sum << endl;
    return 0;
}
```

Exercise 3.2

```
// read_line.cpp

#include <iostream>
#include <string>

int main()
{
    std::string line;

    while (std::getline(std::cin, line))
        std::cout << line << std::endl;

    return 0;
}
```

```
// read_word.cpp

#include <iostream>
#include <string>

int main()
{
    std::string line;

    while (std::cin >> line)
        std::cout << line << std::endl;
```

```cpp
    return 0;
}
```

Exercise 3.3

With the `string` input operator, both leading spaces and trailing whitespaces are discarded but no whitespace is discard with `getline`.

Exercise 3.4

```cpp
// string_comparison.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s1, s2;

    if (std::cin >> s1 >> s2) {
        if (s1 == s2) {
            std::cout << "Both strings are equal." << std::endl;
        } else {
            if (s1 > s2)
                std::cout << "First string is larger." << std::endl;
            else
                std::cout << "Second string is larger." << std::endl;
        }
    }

    return 0;
}
```

```cpp
// string_comparison_bis.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s1, s2;

    if (std::cin >> s1 >> s2) {
        if (s1.size() == s2.size()) {
            std::cout << "Both strings have the same size." << std::endl;
        } else {
            if (s1.size() > s2.size())
                std::cout << "First string is longer." << std::endl;
            else
                std::cout << "Second string is longer." << std::endl;
        }
    }

    return 0;
}
```

Exercise 3.5

```
// concatenate.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s, tmp;

    while (std::cin >> tmp)
        s += tmp;

    std::cout << s << std::endl;

    return 0;
}
```

```
// concatenate.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s, tmp;

    if (std::cin >> tmp) {
        s += tmp;
        while (std::getline(std::cin, tmp))
            s += " " + tmp;
    }
    std::cout << s << std::endl;

    return 0;
}
```

Exercise 3.6

```
// to_X.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s("Foo bar");

    for (auto &c : s)
        c = 'X';

    std::cout << s << std::endl;

    return 0;
}
```

Exercise 3.7

I think it would not mutate the `string` as we would modify only a copy of each character.

Here is the previous program modified. With some compiler options we get a warning.

```
// do_nothing.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s("Foo bar");

    for (auto c : s)
        c = 'X';
    std::cout << s << std::endl;

    return 0;
}
```

The output is the `string` unmodified as expected.

Exercise 3.8

```
// to_X_while.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s("Foo bar");

    decltype(s.size()) i = 0;
    while (i < s.size()) {
s[i] = 'X';
++i;
    }

    std::cout << s << std::endl;

    return 0;
}
```

```
// to_X_traditional_for.cpp

#include <iostream>
#include <string>

int main()
{
    std::string s("Foo bar");

    for (decltype(s.size()) i = 0; i < s.size(); ++i)
        s[i] = 'X';

    std::cout << s << std::endl;

    return 0;
}
```

I prefer the range `for` approach as it is less error prone (no need to take care of bounds), it's faster to write and easier to read.

Exercise 3.9

This program is not valid because `s` is initialized to the empty `string` so we can't use the subscript operator (it's undefined behavior).

Exercise 3.10

```
// remove_punctuation.cpp

#include <iostream>
#include <string>
#include <cctype>

int main()
{
    std::string input, output;

    if (getline(std::cin, input)) {
        for (auto c : input) {
            if (!std::ispunct(c))
                output += c;
        }
    }

    std::cout << output << std::endl;

    return 0;
}
```

Exercise 3.11

Yes this range `for` is legal. The type of `c` is `const int&`, we can't test this by trying to assign to `c` and see the compiler error.

Exercise 3.12

(a) Legal, `ivec` is initialized to an empty `vector` of `vector<int>`.

(b) Illegal, `svec` hold strings not vectors of `int`.

(c) Legal, `svec` is initialized to a `vector` of ten strings of value `"null"`.

Exercise 3.13

(a) No elements.

(b) Ten elements, each of value 0.

(c) Ten elements, each of value 42.

(d) One element of value 10.

(e) Two elements, the first one of value 10 and the second with the value 42.

(f) Ten elements, each is the empty `string`.

(g) Ten elements, each is the `string` `"hi"`.

Exercise 3.14

```cpp
// store_ints.cpp

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> ivect;
    int n;

    while (std::cin >> n)
        ivect.push_back(n);

    for (auto i : ivect)
        std::cout << i << std::endl;
}
```

Exercise 3.15

```cpp
// store_strings.cpp

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

int main()
{
    std::vector<std::string> svect;
    std::string w;
    while (std::cin >> w)
        svect.push_back(w);

    for (auto s : svect)
        std::cout << s << std::endl;
}
```

Exercise 3.16

```
// check_vectors.cpp

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

int main()
{
    std::vector<int> v1;
    std::vector<int> v2(10);
    std::vector<int> v3(10, 42);
    std::vector<int> v4{10};
    std::vector<int> v5{10, 42};
    std::vector<std::vector<int>> vv1 = {v1, v2, v3, v4, v5};

    for (const auto &v : vv1) {
        for (auto i : v)
            std::cout << i << " ";
        std::cout << std::endl;
    }

    std::vector<std::string> v6{10};
    std::vector<std::string> v7{10, "hi"};
    std::vector<std::vector<std::string>> vv2 = {v6, v7};

    for (const auto &v : vv2) {
        for (auto s : v)
            std::cout << s << " ";
        std::cout << std::endl;
    }

    return 0;
}
```

Exercise 3.17

```
// print_words.cpp

#include <iostream>
#include <string>
#include <cctype>
#include <vector>

int main()
{
    std::cout << "Enter a list of words:" << std::endl;

    std::vector<std::string> svect;
    std::string w;

    while (std::cin >> w)
        svect.push_back(w);

    for (auto &s : svect) {
        for (auto &c : s)
            c = std::toupper(c);
    }

    unsigned i = 1;
    for (auto &s : svect) {
        std::cout << s;
        if (i % 8 == 0)
            std::cout << std::endl;
        else
            std::cout << " ";
        ++i;
    }
    if (i % 8 != 1)
        std::cout << std::endl;

    return 0;
}
```

Exercise 3.18

This program is illegal. We might fix it by using the `push_back` member function:

```
vector<int> ivec;
ivec.push_back(42);
```

Exercise 3.19

```
// three_ways.cpp

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> v1(10, 42);

    std::vector<int> v2 = {42, 42, 42, 42, 42, 42, 42, 42, 42, 42};

    std::vector<int> v3;
    for (size_t i = 0; i < 10; ++i)
        v3.push_back(42);

    if (v1 != v2 || v2 != v3)
        std::cout << "Something wrong!" << std::endl;
    else
        std::cout << "Everything OK!" << std::endl;

    return 0;
}
```

The first way is preferred as it's easier to read and more safe.

Exercise 3.20

```
// adjacent_pair_sum.cpp

#include <iostream>
#include <vector>

int main()
{
    std::cout << "Enter a list of integers:" << std::endl;

    std::vector<int> ivect;
    int i;

    while (std::cin >> i)
        ivect.push_back(i);

    for (decltype(ivect.size()) i = 0; i + 1 < ivect.size(); ++i)
        std::cout << ivect[i] + ivect[i + 1] << std::endl;
    std::cout << std::endl;

    return 0;
}
```

```
// symmetric_sum.cpp

#include <iostream>
#include <vector>

int main()
{
    std::cout << "Enter a list of integers:" << std::endl;

    std::vector<int> ivect;
    int i;

    while (std::cin >> i)
        ivect.push_back(i);

    if (!ivect.empty()) {
        for (decltype(ivect.size()) i = 0; i < ivect.size(); ++i)
            std::cout << ivect[i] + ivect[ivect.size() - 1 - i] << std::endl;
    }

    return 0;
}
```