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;
    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

```
// 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;
}
```

```
// 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';
```

```cpp
    std::cout << s << std::endl;

    return 0;
}
```

The output is the `string` unmodified as expected.

Exercise 3.8

```cpp
// 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;
}
```

```cpp
// 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

```cpp
// 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 << "Everything OK!" << std::endl;
    else
        std::cout << "Something wrong!" << std::endl;

    return 0;
}
```

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

Exercise 3.20

```cpp
// 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;
    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;
}
```

Exercise 3.21

```
// print_size_content.cpp

#include <iostream>
#include <vector>

int main()
{
    const std::vector<int> ivec = {1, 2 , 3};

    std::cout << "The size of the vector is "
              << ivec.end() - ivec.begin()
              << ".\nHere is the content of this vector:"
              << std::endl;

    for (auto it = ivec.cbegin(); it != ivec.cend(); ++it)
        std::cout << *it << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.22

```
// print_text_uppercase.cpp

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

int main()
{
    std::vector<std::string> text = {
        "Hello, World!",
        "Nice to see you!",
        "",
        "Goodbye, World!"
    };

    for (auto it_vect = text.begin();
         it_vect != text.cend() && !it_vect->empty(); ++it_vect) {
        for (auto it_str = it_vect->begin(); it_str != it_vect->cend(); ++it_str)
            *it_str = std::toupper(*it_str);
    }

    for (auto it_vect =  text.cbegin(); it_vect != text.cend(); ++it_vect)
        std::cout << *it_vect << std::endl;

    return 0;
}
```

Exercise 3.23

```
// twice_iterator.cpp
```

```cpp
#include <iostream>
#include <vector>

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

    for (int i = 1; i <= 10; ++ i)
        ivect.push_back(i);

    for (auto it = ivect.begin(); it != ivect.end(); ++it)
        *it *= 2;

    for (auto e : ivect)
        std::cout << e << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.24

```cpp
// adjacent_pair_sum_iterator.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 (auto it = ivect.cbegin(); it + 1 != ivect.cend(); ++it)
        std::cout << *it + *(it + 1) << std::endl;

    return 0;
}
```

```cpp
// symmetric_sum_iterator.cpp
```

```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()) {
        auto it_e = ivect.cend() - 1;
        for (auto it_b = ivect.cbegin(); it_b != ivect.cend(); ++it_b, --it_e)
            std::cout << *it_b + *it_e << std::endl;
    }

    return 0;
}
```

Exercise 3.25

```cpp
// grades_clusters.cpp

#include <iostream>
#include <vector>

int main()
{
    std::vector<unsigned> scores(11, 0);
    unsigned grade;
    while (std::cin >> grade) {
        if (grade <= 100)
            ++*(scores.begin() + grade / 10);
    }

    for (auto it = scores.cbegin(); it != scores.end(); ++it)
        std::cout << *it << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.26

If we wrote `mid = (beg + end) / 2;` the program could never end if the value is not found (`mid` could not change and never be equal to `end`).

Exercise 3.27

(a) Illegal, `buf_size` is not a constant expression.

(b) Legal, `4 * 7 - 14` is a constant expression with value greater than 0.

(c) Illegal, `txt_size()` is not a constant expression.

(d) Illegal, `st` must be able to also hold the null character `'\0'` so must be at least able to store 12 characters.

Exercise 3.28

`sa` is an array of 10 empty strings. `ia` is an array of 10 zero ints. `sa2` is an array of 10 empty strings. `ia2` is an array of 10 ints with undefined values.

Exercise 3.29

Arrays can only store a fix amount of values, the size must be known at compile time and arrays can't be of 0 size.

Exercise 3.30

`ia` is an array of size `10` so the valid subscript values are positive integers strictly inferior to `10`. Here in the last execution of the `for` loop we are using a subscript equal to `10` hence the undefined behavior of this code.

Exercise 3.31

```
// array_of_positions.cpp

#include <iostream>

int main()
{
    int ia[10];
    for (int i = 0; i < 10; ++i)
```

```
        ia[i] = i;

    for (auto i : ia)
        std::cout << i << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.32

```
// copy_array.cpp

#include <iostream>

int main()
{
    int ia1[10], ia2[10];
    for (int i = 0; i < 10; ++i)
        ia1[i] = i;

    for (std::size_t i = 0; i < 10; ++i)
        ia2[i] = ia1[i];

    for (auto i : ia2)
        std::cout << i << " ";
    std::cout << std::endl;

    return 0;
}
```

```
// copy_vector.cpp

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> iv1, iv2;
    for (int i = 0; i < 10; ++i)
        iv1.push_back(i);

    iv2 = iv1;

    for (auto i : iv2)
```

```
        std::cout << i << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.33

The behavior of the program would be undefined.

Exercise 3.34

This code assign `p2` to `p1`, this code is always legal as long as `p1` and `p2` are valid pointers.

Exercise 3.35

```
// set_to_zero.cpp

#include <iostream>

int main()
{
    int arr[10];
    for (auto p = std::begin(arr); p != std::end(arr); ++p)
        *p = 0;

    for (auto i : arr)
        std::cout << i << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.36

```
// array_equality.cpp

#include <iostream>

int main()
{
    int v1[3], v2[3];
    std::cout << "Enter the three elements of the first array:" << std::endl;
    for (auto p = std::begin(v1); p != std::end(v1); ++p) {
        if (!(std::cin >> *p)) {
            std::cerr << "Error: bad input" << std::endl;
            return -1;
        }
    }

    std::cout << "Enter the three elements of the second array:" << std::endl;
    for (auto p = std::begin(v2); p != std::end(v2); ++p) {
        if (!(std::cin >> *p)) {
            std::cerr << "Error: bad input" << std::endl;
            return -1;
        }
    }

    if (std::cend(v1) - std::cbegin(v1) != std::cend(v2) - std::cbegin(v2)) {
        std::cout << "Arrays are different" << std::endl;
        return 0;
    }

    for (auto p1 = std::cbegin(v1), p2 = std::cbegin(v2);
         p1 != std::cend(v1); ++p1, ++p2) {
        if (*p1 != *p2) {
            std::cout << "Arrays are different" << std::endl;
            return 0;
        }
    }

    std::cout << "Arrays are equal" << std::endl;

    return 0;
}
```

```
// vector_equality.cpp

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> v1, v2 = {1, 2, 3};
    int t;

    std::cout << "Enter the elements of the vector:" << std::endl;
    while (std::cin >> t)
        v1.push_back(t);

    if (v1 == v2)
        std::cout << "Arrays are equal" << std::endl;
    else
        std::cout << "Arrays are different" << std::endl;

    return 0;
}
```

Exercise 3.37

This program has undefined behavior because it will try to read past the end of the array so anything can happen.

Exercise 3.38

It's meaningless because if we think of pointers as addresses of array elements, subtracting two addresses would give the offset and could be useful information but adding two addresses would make no sense as it would generally depend on the address of the first element of the array.

Exercise 3.39

```
// compare_strings.cpp

#include <iostream>
#include <string>

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

    std::cout << "Enter two strings, one per line:" << std::endl;

    if (std::getline(std::cin, s1) && std::getline(std::cin, s2)) {
        if (s1 < s2)
            std::cout << "The first string is less than the second string.";
        else if (s1 == s2)
            std::cout << "Both strings are equal.";
        else std::cout << "The first string is greater than the second one.";
        std::cout << std::endl;
    }
    return 0;
}
```

```
// compare_cstrings.cpp

#include <iostream>
#include <string>
#include <cstring>

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

    std::cout << "Enter two strings, one per line:" << std::endl;

    if (std::getline(std::cin, s1) && std::getline(std::cin, s2)) {
        const char *p1 = s1.c_str(), *p2 = s2.c_str();
        int cmp = std::strcmp(p1, p2);
        if (cmp < 0)
            std::cout << "The first string is less than the second string.";
        else if (cmp == 0)
            std::cout << "Both strings are equal.";
        else std::cout << "The first string is greater than the second one.";
        std::cout << std::endl;
    }

    return 0;
}
```

Exercise 3.40

```
// cstrings_cat.cpp

#include <iostream>
#include <cstring>

int main()
{
    constexpr const char s1[] = "Hello, ";
    constexpr const char s2[] = "World!";
    char s3[14];

    std::strcpy(s3, s1);
    std::strcat(s3, s2);
    std::cout << s3 << std::endl;

    return 0;
}
```

Exercise 3.41

```
// vector_from_array.cpp

#include <iostream>
#include <vector>

int main()
{
    const int arr[3] = {1, 2, 3};
    const std::vector<int> vec(std::begin(arr), std::end(arr));

    for (auto i : vec)
        std::cout << i << " ";
    std::cout << std::endl;

    return 0;
}
```

Exercise 3.42

```
// vector_from_array.cpp

#include <iostream>
#include <vector>

int main()
{
    const std::vector<int> vec{1, 2, 3};
    int arr[3];

    for (std::size_t i = 0; i < 3; ++i)
        arr[i] = vec[i];

    for (auto i : arr)
        std::cout << i << " ";
    std::cout << std::endl;
    return 0;
}
```

Exercise 3.43

```
// for_loop.cpp

#include <iostream>

int main()
{
    int ia[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {1, 1, 1, 1}};

    for (const int (&col)[4] : ia) {
        for (int i : col)
            std::cout << i << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    for (std::size_t row = 0; row < 3; ++row) {
        for (std::size_t col = 0; col < 4; ++col)
            std::cout << ia[row][col] << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    for (const int (*p)[4] = std::cbegin(ia); p != std::cend(ia); ++p) {
        for (const int *q = std::cbegin(*p); q != std::cend(*p); ++q)
            std::cout << *q << " ";
        std::cout << std::endl;
    }

    return 0;
}
```

Exercise 3.44

```
// for_loop_type_alias.cpp

#include <iostream>
```

```cpp
int main()
{
    int ia[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {1, 1, 1, 1}};

    using ref_array_const_int = const int (&)[4];
    for (ref_array_const_int col : ia) {
        for (int i : col)
            std::cout << i << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    for (std::size_t row = 0; row < 3; ++row) {
        for (std::size_t col = 0; col < 4; ++col)
            std::cout << ia[row][col] << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    using pointer_array_const_int = const int (*)[4];
    using pointer_const_int = const int *;
    for (pointer_array_const_int p = std::cbegin(ia); p != std::cend(ia); ++p) {
        for (pointer_const_int q = std::cbegin(*p); q != std::cend(*p); ++q)
            std::cout << *q << " ";
        std::cout << std::endl;
    }

    return 0;
}
```

Exercise 3.45

```cpp
// for_loop.cpp

#include <iostream>

int main()
{
    int ia[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {1, 1, 1, 1}};

    for (auto &col : ia) {
        for (auto i : col)
            std::cout << i << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    for (auto row = 0; row < 3; ++row) {
        for (auto col = 0; col < 4; ++col)
            std::cout << ia[row][col] << " ";
        std::cout << std::endl;
    }

    std::cout << std::endl;

    for (auto p = std::cbegin(ia); p != std::cend(ia); ++p) {
        for (auto q = std::cbegin(*p); q != std::cend(*p); ++q)
            std::cout << *q << " ";
        std::cout << std::endl;
    }

    return 0;
}
```