```cpp
// week10

// iterator
int main(){

    // pointer and array
    int const size = 4;
    char array[size] = {'a', 'b', 'c', 'd'};
    char * beginning  = array;
    char * one_past_end = array + size;
    for(char const * current_element = beginning; current_element != one_past_end; ++ current_element){
        cout << *current_element;
    }
    cout << endl;

    // .begin() and .end()
    std::vector<char> vec = {'a', 'b', 'c', 'd'};
    for (std::vector<char>::iterator iter{vec.begin()};
        iter != vec.end(); ++iter) {
        std::cout << *iter;
    }
    std::cout << std::endl;

    // iterator type
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    std::vector<int>::iterator it;

    for (it = numbers.begin(); it != numbers.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    // other type of iterator

    // Reverse iteration
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    for (auto it = numbers.rbegin(); it != numbers.rend(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;  // Output: 5 4 3 2 1

    // Constant iteration
    const std::vector<int> numbers = {1, 2, 3, 4, 5};
    for (auto it = numbers.cbegin(); it != numbers.cend(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;  // Output: 1 2 3 4 5

    // Constant reverse iteration
    const std::vector<int> numbers = {1, 2, 3, 4, 5};
    for (auto it = numbers.crbegin(); it != numbers.crend(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;  // Output: 5 4 3 2 1

    //capacity
    //Returns the total capacity of the vector, which is the maximum number of elements it can hold before a new memory allocation is needed.
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    std::cout << "Capacity: " << numbers.capacity() << std::endl;

    return 0;
}



// reserve and capacity about the vector

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

    std::cout << "Initial capacity: " << numbers.capacity() << std::endl;

    // Reserve capacity for 100 elements
    numbers.reserve(100);

    std::cout << "Capacity after reserve(100): " << numbers.capacity() << std::endl;

    // Adding elements up to the reserved capacity
    for (int i = 0; i < 50; ++i) {
        numbers.push_back(i);
    }

    std::cout << "Size after adding 50 elements: " << numbers.size() << std::endl;
    std::cout << "Capacity after adding 50 elements: " << numbers.capacity() << std::endl;

    // Adding more elements beyond the reserved capacity
    for (int i = 50; i < 150; ++i) {
        numbers.push_back(i);
    }

    std::cout << "Size after adding 150 elements: " << numbers.size() << std::endl;
    std::cout << "Capacity after adding 150 elements: " << numbers.capacity() << std::endl;

    return 0;
```

```
}

/*
Initial capacity: 0
Initial capacity: 0
Capacity after reserve(100): 100
Size after adding 50 elements: 50
Capacity after adding 50 elements: 100
Size after adding 150 elements: 150
Capacity after adding 150 elements: 200
*/

// stream iterator
int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5};

    // Create an output stream iterator
    std::ostream_iterator<int> output_iterator(std::cout, " "); // Output: 1 2 3 4 5

    // Copy elements from the vector to the output stream
    std::copy(numbers.begin(), numbers.end(), output_iterator); // Output: 1 2 3 4 5

    return 0;
}




// map

// std::pair
#include <utility> // for std::pair

int main() {
    // Creating a pair of a string and an integer
    std::pair<std::string, int> word_count{"Mal", 100};

    // Accessing elements of the pair
    std::cout << "First element: " << word_count.first << std::endl;    // Output: Mal
    std::cout << "Second element: " << word_count.second << std::endl;  // Output: 100


    // Creating a pair using std::make_pair
    auto word_count = std::make_pair("Hello", 42);
    std::cout << "First: " << word_count.first << ", Second: " << word_count.second << std::endl;

    // pair in vector
    std::vector<std::pair<std::string, int>> word_counts = {
        {"Hello", 1},
        {"World", 2},
        {"C++", 3}
    };

    for (const auto& wc : word_counts) {
        std::cout << wc.first << ": " << wc.second << std::endl;
    }


    // pair in map
    // Initialize a map with names as keys and phone numbers as values
    std::map<std::string, int> phone_book{
        {"David Hume", 123456},
        {"Karl Popper", 234567},
        {"BAWR", 345678}
    };

    // Method 1: Using pair to access elements
    for (const std::pair<std::string, int>& entry : phone_book) {
        std::cout << entry.second << " -> " << entry.first << std::endl;
    }

    std::cout << "--------------------" << std::endl;

    // Method 2: Using structured binding (C++17)
    for (const auto& [name, number] : phone_book) {
        std::cout << number << " -> " << name << std::endl;
    }

    // no push_back for map, but insert
    map<string, int> m;
    string word = "hello";

    m.insert(pair<string, int>{word, 1});
    // or
    m.insert({word, 1});
    // the usage of auto
    //without auto
    std::pair<std::map<std::string, int>::iterator, bool> result = m.insert({"Josh", 50});
    //with auto
    auto result = m.insert({"Josh", 50});
    // iterator usage here
    std::map<std::string, int> phone_book{
        {"David Hume", 123456},
        {"Karl Popper", 234567},
```

```
        {"BAWR", 345678}
    };
    std::map<std::string, int>::iterator iter = phone_book.find("David Hume");
    if (iter != phone_book.end()) {
        cout << (*iter).second;
    }
    return 0;
}
```