```cpp
// week11
// algorithm
// page 1

#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric> // for std::iota (to populate the vector)

int main() {
    // Create a vector of integers
    std::vector<int> vec(10);
    std::iota(vec.begin(), vec.end(), 1); // Fill with 1, 2, 3, ..., 10

    std::cout << "Original vector: ";
    for (const auto& val : vec) std::cout << val << " ";
    std::cout << "\n";

    // 1. for_each: Double each element in the vector
    std::for_each(vec.begin(), vec.end(), [](int& n) { n *= 2; });
    std::cout << "After for_each (doubled values): ";
    for (const auto& val : vec) std::cout << val << " "; // 2, 4, 6, ..., 20
    std::cout << "\n";

    // 2. for_each_n: Add 5 to the first 3 elements
    std::for_each_n(vec.begin(), 3, [](int& n) { n += 5; });
    std::cout << "After for_each_n (first 3 elements +5): ";
    for (const auto& val : vec) std::cout << val << " "; // 7, 9, 11, 8, 10, ..., 20
    std::cout << "\n";

    // 3. all_of: Check if all elements are even
    bool allEven = std::all_of(vec.begin(), vec.end(), [](int n) { return n % 2 == 0; });
    std::cout << "Are all elements even? " << (allEven ? "Yes" : "No") << "\n"; // No

    // 4. any_of: Check if any element is greater than 20
    bool hasGreaterThan20 = std::any_of(vec.begin(), vec.end(), [](int n) { return n > 20; });
    std::cout << "Is there any element greater than 20? " << (hasGreaterThan20 ? "Yes" : "No") << "\n"; // Yes

    // 5. find: Find the first occurrence of 14
    auto findIt = std::find(vec.begin(), vec.end(), 14);
    if (findIt != vec.end()) {
        std::cout << "Found element 14 at position: " << std::distance(vec.begin(), findIt) << "\n"; // 3
    } else {
        std::cout << "Element 14 not found\n";
    }

    // 6. find_if: Find the first element greater than 15
    auto findIfIt = std::find_if(vec.begin(), vec.end(), [](int n) { return n > 15; });
    if (findIfIt != vec.end()) {
        std::cout << "First element greater than 15: " << *findIfIt << "\n"; // 16
    } else {
        std::cout << "No element greater than 15 found\n";
    }

    // 7. adjacent_find: Find the first adjacent pair of equal elements
    vec[5] = vec[6]; // Make an adjacent duplicate for demonstration
    auto adjacentIt = std::adjacent_find(vec.begin(), vec.end());
    if (adjacentIt != vec.end()) {
        std::cout << "Found adjacent duplicate: " << *adjacentIt << "\n"; // 12
    } else {
        std::cout << "No adjacent duplicates found\n";
    }

    return 0;
}

// page 2
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric> // For std::iota
#include <iterator> // For std::back_inserter

int main() {
    // Initialize a vector
    std::vector<int> vec(10);
    std::iota(vec.begin(), vec.end(), 1); // Fill with 1, 2, 3, ..., 10

    // 1. count and count_if
    int count3 = std::count(vec.begin(), vec.end(), 3);
    int countEven = std::count_if(vec.begin(), vec.end(), [](int n) { return n % 2 == 0; });
    std::cout << "Count of 3: " << count3 << "\n";   // 1
    std::cout << "Count of even numbers: " << countEven << "\n";
```

```cpp
    // 2. equal
    std::vector<int> vecCopy = vec;
    bool areEqual = std::equal(vec.begin(), vec.end(), vecCopy.begin());
    std::cout << "Are the original vector and the copy equal? " << (areEqual ? "Yes" : "No") << "\n"; // Yes

    // 3. search
    std::vector<int> subVec = {3, 4};
    auto searchIt = std::search(vec.begin(), vec.end(), subVec.begin(), subVec.end());
    if (searchIt != vec.end()) {
        std::cout << "Subsequence found starting at position: " << std::distance(vec.begin(), searchIt) << "\n"; // 2
    } else {
        std::cout << "Subsequence not found\n";
    }

    // 4. copy and copy_if

    // copy
    // std::copy(InputIterator first, InputIterator last, OutputIterator d_first);

    // copy_if
    //std::copy_if(InputIterator first, InputIterator last, OutputIterator d_first, UnaryPredicate pred);

    std::vector<int> copyVec(vec.size());
    std::copy(vec.begin(), vec.end(), copyVec.begin());
    std::cout << "Copy of vector: ";
    for (const auto& val : copyVec) std::cout << val << " "; // 1, 2, 3, ..., 10
    std::cout << "\n";

    // copy if
    std::vector<int> evenVec(vec.size());
    auto evenEnd = std::copy_if(vec.begin(), vec.end(), evenVec.begin(), [](int n) { return n % 2 == 0; });
    std::cout << "Copy of even numbers: ";
    for (auto it = evenVec.begin(); it != evenEnd; ++it) std::cout << *it << " "; // 2, 4, 6, 8, 10

    // update of back inserter
    std::vector<int> source = {1, 2, 3, 4, 5};
    std::vector<int> destination;

    // Use std::back_inserter to append elements from source to destination

    // use back_inserter you don't have to do: vector<int> destination(source.size());
    std::copy(source.begin(), source.end(), std::back_inserter(destination));

    // Print the destination vector
    for (int num : destination) {
        std::cout << num << " "; // 1, 2, 3, 4, 5
    }

    // 5. transform (unary)
    std::vector<int> squaredVec(vec.size());
    std::transform(vec.begin(), vec.end(), squaredVec.begin(), [](int n) { return n * n; });
    std::cout << "Squared values: ";
    for (const auto& val : squaredVec) std::cout << val << " "; // 1, 4, 9, ..., 100
    std::cout << "\n";

    // 6. transform (binary)
    std::vector<int> vec2(10);
    std::iota(vec2.begin(), vec2.end(), 10); // Fill with 10, 11, 12, ..., 19
    std::vector<int> sumVec(vec.size());
    std::transform(vec.begin(), vec.end(), vec2.begin(), sumVec.begin(), [](int a, int b) { return a + b; });
    std::cout << "Sum of vec and vec2: ";
    for (const auto& val : sumVec) std::cout << val << " "; // 11, 13, 15, ..., 29
    std::cout << "\n";

    return 0;
}


// page 3
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <iterator> // for std::ostream_iterator

int main() {
    // 1. generate
    std::vector<int> vec(10); // 0, 0, 0, ..., 0
    int val = 1;
    std::generate(vec.begin(), vec.end(), [&val]() { return val++; });
    std::cout << "Generated vector: ";
    for (const auto& n : vec) std::cout << n << " "; // 1, 2, 3, ..., 10
```

```
    std::cout << "\n";

    // 2. reverse
    std::reverse(vec.begin(), vec.end());
    std::cout << "Reversed vector: ";
    for (const auto& n : vec) std::cout << n << " "; // 10, 9, 8, ..., 1
    std::cout << "\n";

    // 3. shuffle
    std::random_device rd;
    std::mt19937 g(rd());
    std::shuffle(vec.begin(), vec.end(), g);
    std::cout << "Shuffled vector: ";
    for (const auto& n : vec) std::cout << n << " ";// Random order
    std::cout << "\n";

    // 4. partition
    std::partition(vec.begin(), vec.end(), [](int n) { return n % 2 == 0; });
    std::cout << "Partitioned vector (evens first): ";
    for (const auto& n : vec) std::cout << n << " "; //  how is the result vector be like?  10, 8, 6, 4, 2, 9, 7, 5, 3, 1
    std::cout << "\n";

    // 5. sort
    std::sort(vec.begin(), vec.end());
    std::cout << "Sorted vector: ";
    for (const auto& n : vec) std::cout << n << " ";
    std::cout << "\n";

    // 6. stable_sort
    std::vector<std::pair<int, char>> vecPair = {{2, 'B'}, {3, 'C'}, {2, 'A'}, {1, 'D'}};
    std::stable_sort(vecPair.begin(), vecPair.end(), [](auto& a, auto& b) { return a.first < b.first; });
    std::cout << "Stable sorted pairs: ";
    for (const auto& p : vecPair) std::cout << "(" << p.first << ", " << p.second << ") "; // (1, D), (2, B), (2, A), (3, C)
    std::cout << "\n";

    return 0;
}



// page 4
#include <iostream>
#include <vector>
#include <algorithm> // For max_element, min_element, max, min

int main() {
    // Example vector
    std::vector<int> vec = {10, 3, 5, 7, 2, 8};

    // 1. max_element
    auto maxIt = std::max_element(vec.begin(), vec.end());
    std::cout << "Largest element in the vector: " << *maxIt << "\n"; // 10

    // 2. min_element
    auto minIt = std::min_element(vec.begin(), vec.end());
    std::cout << "Smallest element in the vector: " << *minIt << "\n"; // 2

    // 3. max
    int a = 15, b = 20;
    std::cout << "Greater of " << a << " and " << b << ": " << std::max(a, b) << "\n";  // 20

    // 4. min
    std::cout << "Smaller of " << a << " and " << b << ": " << std::min(a, b) << "\n";  // 15

    return 0;
}


// page 5
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator> // For std::back_inserter

int main() {
    std::vector<int> set1 = {1, 2, 3, 4, 5};
    std::vector<int> set2 = {3, 4, 6};

    // 1. includes
    bool isIncluded = std::includes(set1.begin(), set1.end(), set2.begin(), set2.end());
    std::cout << "Set2 is included in Set1? " << (isIncluded ? "Yes" : "No") << "\n"; // No

    // 2. set_union
```

```cpp
    std::vector<int> unionResult;
    std::set_union(set1.begin(), set1.end(), set2.begin(), set2.end(), std::back_inserter(unionResult));
    std::cout << "Union: ";
    for (const auto& val : unionResult) std::cout << val << " "; // 1, 2, 3, 4, 5, 6
    std::cout << "\n";

    // 3. set_intersection
    std::vector<int> intersectionResult;
    std::set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), std::back_inserter(intersectionResult));
    std::cout << "Intersection: ";
    for (const auto& val : intersectionResult) std::cout << val << " "; // 3, 4
    std::cout << "\n";

    // 4. set_difference
    std::vector<int> differenceResult;
    std::set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), std::back_inserter(differenceResult));
    std::cout << "Difference (Set1 - Set2): ";
    for (const auto& val : differenceResult) std::cout << val << " "; // 1, 2, 5
    std::cout << "\n";

    // 5. merge
    std::vector<int> mergeResult;
    std::merge(set1.begin(), set1.end(), set2.begin(), set2.end(), std::back_inserter(mergeResult));
    std::cout << "Merged: ";
    for (const auto& val : mergeResult) std::cout << val << " "; // 1, 2, 3, 4, 4, 5, 6
    std::cout << "\n";

    return 0;
}


// page 6
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec = {1, 2, 2, 3, 4, 5, 5, 5};

    // erase
    vec.erase(vec.begin() + 2); // 1, 2, 3, 4, 5, 5, 5

    // Remove all elements equal to 2
    vec.erase(std::remove(vec.begin(), vec.end(), 2), vec.end()); // 1, 3, 4, 5, 5, 5

    // Remove all odd elements
    vec.erase(std::remove_if(vec.begin(), vec.end(), [](int n) { return n % 2 != 0; }), vec.end()); // 4

    // Remove consecutive duplicates
    vec.erase(std::unique(vec.begin(), vec.end()), vec.end()); // 4

    // Output the final vector
    std::cout << "Final vector: ";
    for (int val : vec) std::cout << val << " "; // 4
    std::cout << "\n";

    return 0;
}

// more iterators
#include <bits/stdc++.h>

using namespace std;

int main(){
    // iterator types

    // container iterators
    vector<int> v = {1, 2, 3, 4, 5};
    vector<int>::iterator it = v.begin();
    cout << *it << endl;

    // container const iterators
    vector<int>::const_iterator cit = v.cbegin();
    cout << *cit << endl;

    // Forward Iterator

    // deference and equalities    *it, it == it2, it != it2
    // forward iterator            ++it, --it

    // Bidirectional Iterator      ++it, --it
```

```
    // Random Access Iterator      it += 5, it -= 5, it + 5, it - 5, it[5]

    // pointers are iterators
    int array[3] = {2, 3, 2};
    int *ar_begin = array;
    int *ar_end = array + 3;
    ar_begin++;
    *ar_begin = 4;
    bool b = ar_begin != ar_end;


    // iterator invalidation
    string const original =
        "My dog is named Mal.";
    string copy{original};
    sort(copy.begin(), copy.end());
    cout << copy << endl;

    // prints: "    .MMaadddegilmnosy"


    // iterator invalidation
    copy = original;
    std::string::iterator start =
        copy.begin() + 5;
    sort(start, copy.end());
    cout << copy << endl;

    // prints: "My do    .Maadegilmns"

    // function used in sort
    bool CaseInsensitiveLess(char left, char right) {
        return tolower(left) < tolower(right);
    }

    // ...
    copy = original;
    sort(copy.begin(), copy.end(),CaseInsensitiveLess);
    cout << copy << endl;
    // prints: ".aaddegilMmNnosy"


    // lambda function used in sort
    copy = original;
    sort(copy.begin(), copy.end(),
        [](char left, char right){
            return tolower(left) < tolower(right);
        });
    cout << copy << endl;
    // prints: ".aaddegilMmNnosy"

    // std::ranges::sort
    // sort is a function that sorts a range of elements


    // Important ALgorithms


    return 0;
}
```