11 – Exercises on Iterators and STL algorithms

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

```cpp
#include <algorithm>
#include <vector>
#include <array>

std::vector<int> v = {40, 10, 30, 20};

// std::sort -- sorts a range in place [begin, end)
std::sort(v.begin(), v.end());               // ascending: 10 20 30 40
std::sort(v.begin(), v.end(), std::greater<int>()); // descending: 40 30 20 10

// std::find -- returns an iterator to the first match, or end() if not found
auto it = std::find(v.begin(), v.end(), 30);
if (it != v.end())
    std::cout << *it << std::endl; // 30

// std::copy -- copies a range into another container
std::vector<int> dst(v.size());
std::copy(v.begin(), v.end(), dst.begin());

// std::vector::insert -- inserts element(s) at a position
v.insert(v.begin() + 1, 99);      // insert 99 at index 1
v.insert(v.end(), other.begin(), other.end()); // append a whole range
```

Key vocabulary:

- All STL algorithms work on **iterator ranges**: a `begin` iterator (pointing to the first element) and an `end` iterator (pointing one past the last element). You can use iterators contained between `begin` and `end`.
- `std::find` returns `end()` when no match is found — always check before dereferencing.
- `std::copy` does **not** require the destination to have enough space already; it will resize as needed.
- `std::vector::insert` with a range iterator pair appends or splices another container's contents.

Part A — Sorting and searching

Problem 1 — Trace the output

What does this program print?

Sort and find on a track list

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

int main() {
    std::vector<int> durations = {340, 185, 252, 95, 210, 163};

    std::sort(durations.begin(), durations.end());

    for (int i = 0; i < durations.size(); i = i + 1)
        std::cout << durations[i] << " ";
    std::cout << std::endl;                      // line A

    auto it = std::find(durations.begin(), durations.end(), 252);
    if (it != durations.end())
        std::cout << "found: " << *it << std::endl; // line B

    auto it2 = std::find(durations.begin(), durations.end(), 999);
    if (it2 == durations.end())
        std::cout << "not found" << std::endl;   // line C

    std::cout << (it - durations.begin()) << std::endl; // line D

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line D computes the distance between `it` and `durations.begin()`. What does this value represent?

Problem 2 — Descending sort and custom comparator

What does this program print?

Descending sort

```cpp
#include <iostream>
#include <algorithm>
#include <array>

bool below_60(const int s) { return s < 60; }
int main() {
    std::array<int, 5> scores = {72, 95, 40, 88, 55};

    std::sort(scores.begin(), scores.end(), std::greater<int>());

    for (int i = 0; i < scores.size(); i = i + 1)
        std::cout << scores[i] << " ";
    std::cout << std::endl;                 // line A

    // find the first score below 60 after sorting
    auto it = std::find_if(scores.begin(), scores.end(),
                           below_60);
    if (it != scores.end())
        std::cout << *it << std::endl;      // line B

    return 0;
}
```

Line A:

Line B:

Note: The function `below_60` is a predicate: a function that returns `bool`. `std::find_if` returns an iterator to the first element for which the predicate is `true`.

Problem 3 — Spot the bug

Each snippet has one bug. Identify and fix it.

Three buggy uses of sort and find 1/2

```cpp
// (1) Should sort only the first three elements of the vector
std::vector<int> v = {50, 10, 40, 30, 20};
std::sort(v.begin(), v.begin() + 4);
```

Three buggy uses of sort and find 2/2

```cpp
// (2) Should print the found value, but may crash
std::vector<int> notes = {60, 62, 64};
auto it = std::find(notes.begin(), notes.end(), 63);
std::cout << *it << std::endl;

// (3) Should sort the whole array in descending order
std::array<int, 4> arr = {3, 1, 4, 2};
std::sort(arr.begin(), arr.end(), std::less<int>());
```

Bug 1:

Fix 1:

Bug 2:

Fix 2:

Bug 3:

Fix 3:

Part B — Copying and inserting

Problem 1 — Trace the output

What does this program print?

Copy and insert

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

void print(const std::vector<int>& v) {
    for (int i = 0; i < v.size(); i = i + 1)
        std::cout << v[i] << " ";
    std::cout << std::endl;
}

int main() {
    std::vector<int> a = {10, 20, 30};
    std::vector<int> b(3);

    std::copy(a.begin(), a.end(), b.begin());
    print(b);                                   // line A

    b[1] = 99;
```

```
    print(a);                                   // line B
    print(b);                                   // line C

    std::vector<int> c = {40, 50};
    a.insert(a.end(), c.begin(), c.end());
    print(a);                                   // line D

    a.insert(a.begin() + 1, 99);
    print(a);                                   // line E

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line E:

After line B, `a[1]` is still `20` even though we set `b[1] = 99`. Why?

Part C — Putting it all together

Problem 1 — Write the code: playlist merger

Write a function:

```
std::vector<int> merge_playlists(std::vector<int> a, std::vector<int> b);
```

that:

1. Appends all elements of `b` to the end of `a` using `insert`.
2. Sorts the resulting vector in ascending order.
3. Removes duplicate values using `std::unique`.
4. Returns the result.

Then write a `main` that merges these two playlists and prints the result:

```
std::vector<int> p1 = {340, 185, 252, 185};
std::vector<int> p2 = {95, 252, 210};
```

Expected output:

```
95 185 210 252 340
```

Problem 2 — Write the code: setlist search

A setlist is stored as a `std::vector<std::string>` of song titles. Write a function:

```
int find_song(const std::vector<std::string>& setlist,
              const std::string& title);
```

that returns the **zero-based position** of `title` in `setlist`, or `-1` if it is not found. Use `std::find` and `std::distance`.

Then write a `main` that declares the following setlist, calls `find_song` for three different titles, and prints the result of each search.

```
std::vector<std::string> setlist = {
    "So What",
    "Freddie Freeloader",
    "Blue in Green",
    "All Blues",
    "Flamenco Sketches"
};
```

Expected output:

```
"Blue in Green" found at position 2
"All Blues" found at position 3
"Giant Steps" not found
```