Sorting, Searching, and Merging

Goal: Practice sorting, searching, and merging in an HMI-related context.

Scenario:

A new feature requires merging two lists of controls into a single sorted list for efficient access.

Steps:

Sorting:

Sort the controls by their ID using std::sort.

Use std::stable\_sort to maintain relative order for controls with equal IDs.

Binary Search:

Search for a control by ID using std::lower\_bound and std::upper\_bound.

Merging:

Merge two sorted lists of controls using std::merge.

Use std::inplace\_merge to combine controls from two different segments in the same list.

Set Operations:

Use std::set\_union and std::set\_intersection to identify common and unique controls.

Program:

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <iterator>

#include <set>

using namespace std;

struct Control

{

string id;

string type;

bool operator<(const Control& other) const

{

return id < other.id;

}

};

int main()

{

vector<Control> controls1 = { {"C5", "Slider"}, {"C2", "Button"}, {"C3", "Slider"}, {"C4", "Button"} };

vector<Control> controls2 = { {"C1", "Button"}, {"C3", "Slider"}, {"C6", "Button"}, {"C7", "Slider"} };

sort(controls1.begin(), controls1.end());

sort(controls2.begin(), controls2.end());

cout << "Sorted controls1: ";

for (const auto& control : controls1)

{

cout << control.id << " ";

}

cout << "Sorted controls2: ";

for (const auto& control : controls2)

{

cout << control.id << " ";

}

stable\_sort(controls1.begin(), controls1.end());

cout << "\nAfter stable sort (controls1): ";

for (const auto& control : controls1)

{

cout << control.id << " ";

}

auto it = lower\_bound(controls1.begin(), controls1.end(), Control{"C3", ""});

if (it != controls1.end())

{

cout << "\nFound control with ID C3 at position: " << distance(controls1.begin(), it);

}

else

{

cout << "\nControl with ID C3 not found!";

}

it = upper\_bound(controls1.begin(), controls1.end(), Control{"C3", ""});

if (it != controls1.end())

{

cout << "\nUpper bound for ID C3 is control with ID: " << it->id;

}

else

{

cout << "\nNo control greater than C3 found!";

}

vector<Control> mergedControls(controls1.size() + controls2.size());

merge(controls1.begin(), controls1.end(), controls2.begin(), controls2.end(), mergedControls.begin());

cout << "\nMerged controls: ";

for (const auto& control : mergedControls)

{

cout << control.id << " ";

}

controls1.insert(controls1.end(), controls2.begin(), controls2.end());

inplace\_merge(controls1.begin(), controls1.begin() + controls1.size() - controls2.size(), controls1.end());

cout << "\nIn-place merged controls1: ";

for (const auto& control : controls1)

{

cout << control.id << " ";

}

set<Control> set1(controls1.begin(), controls1.end());

set<Control> set2(controls2.begin(), controls2.end());

vector<Control> unionControls;

set\_union(set1.begin(), set1.end(), set2.begin(), set2.end(), back\_inserter(unionControls));

cout << "\nUnion of controls (unique controls from both lists): ";

for (const auto& control : unionControls)

{

cout << control.id << " ";

}

vector<Control> intersectionControls;

set\_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), back\_inserter(intersectionControls));

cout << "\nIntersection of controls (common controls): ";

for (const auto& control : intersectionControls)

{

cout << control.id << " ";

}

return 0;

}

Output:

Sorted controls1: C2 C3 C4 C5 Sorted controls2: C1 C3 C6 C7

After stable sort (controls1): C2 C3 C4 C5

Found control with ID C3 at position: 1

Upper bound for ID C3 is control with ID: C4

Merged controls: C1 C2 C3 C3 C4 C5 C6 C7

In-place merged controls1: C1 C2 C3 C3 C4 C5 C6 C7

Union of controls (unique controls from both lists): C1 C2 C3 C4 C5 C6 C7

Intersection of controls (common controls): C1 C3 C6 C7