Working with STL Algorithms

Goal: Practice commonly used STL algorithms in the context of managing HMI control states (e.g., button visibility, slider values).

Scenario:

Imagine an HMI dashboard with buttons and sliders. Each control has a unique ID and a state (visible, invisible, disabled).

Steps:

Define the data structure:

Create a struct Control:

cppCopy codestruct Control { int id; // Unique ID std::string type; // "button" or "slider" std::string state; // "visible", "invisible", "disabled"};

Initialize the container:

Create a std::vector<Control> and populate it with sample controls (5 buttons and 5 sliders).

Use the following algorithms:

std::for\_each: Iterate through all controls and print their details.

std::find: Find a control with a specific ID.

std::find\_if: Find the first invisible control.

std::adjacent\_find: Check for consecutive controls with the same state.

std::count: Count the number of visible controls.

std::count\_if: Count sliders that are disabled.

std::equal: Compare two subranges of controls to check if they are identical.

Implementation Example:

cppCopy codeauto invisibleControl = std::find\_if(controls.begin(), controls.end(), [](const Control& ctrl) { return ctrl.state == "invisible"; });

Output Results:

Print the results of each algorithm in a user-friendly format.

Program:

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

using namespace std;

struct Control

{

int id;

string type;

string state;

};

int main()

{

vector<Control> controls = {

{1, "button", "visible"},

{2, "button", "invisible"},

{3, "button", "disabled"},

{4, "button", "visible"},

{5, "button", "invisible"},

{6, "slider", "visible"},

{7, "slider", "invisible"},

{8, "slider", "disabled"},

{9, "slider", "visible"},

{10, "slider", "invisible"}

};

cout << "All controls:" << "\n";

// for\_each

for\_each(controls.begin(), controls.end(), [](const Control& c)

{

cout << "ID: " << c.id << ", Type: " << c.type << ", State: " << c.state << "\n";

});

// find

auto searchID = 6;

auto found = find\_if(controls.begin(), controls.end(), [searchID](const Control& c)

{

return c.id == searchID;

});

cout << "\nSearching element " << searchID << ": ";

if(found != controls.end())

{

cout << found->state;

}

else

{

cout << "Not found";

}

// find\_if

auto first = find\_if(controls.begin(), controls.end(), [](const Control& c)

{

return c.state == "invisible";

});

cout << "\nFirst invisible: ";

if(first != controls.end())

{

cout << first->id;

}

else

{

cout << "Not found";

}

// adjacent\_find

auto consecutive = adjacent\_find(controls.begin(), controls.end(), [](const Control& a, const Control& b)

{

return a.state == b.state;

});

if(consecutive != controls.end())

{

cout << "\nSame state: " << consecutive->id << " and " << (consecutive + 1)->id;

}

else

{

cout << "\nNo adjacent controls of same state found.";

}

// count

vector<string> states;

for(auto& c: controls)

{

states.push\_back(c.state);

}

cout<<"\nCount the number of visible controls using count: ";

int visibleCount=count(states.begin(),states.end(),"visible");

cout<<visibleCount;

// count\_if

auto disabled = count\_if(controls.begin(), controls.end(), [](const Control& c)

{

return c.type == "slider" && c.state == "disabled";

});

cout << "\nNumber of disabled in slider: " << disabled;

// equal

bool Identical = equal(controls.begin(), controls.begin() + 2, controls.end() -2, [](const Control& a, const Control& b)

{

return a.type == b.type && a.state == b.state;

});

cout << "\nEqual: " << (Identical ? "Yes" : "No");

return 0;

}

Output:

All controls:

ID: 1, Type: button, State: visible

ID: 2, Type: button, State: invisible

ID: 3, Type: button, State: disabled

ID: 4, Type: button, State: visible

ID: 5, Type: button, State: invisible

ID: 6, Type: slider, State: visible

ID: 7, Type: slider, State: invisible

ID: 8, Type: slider, State: disabled

ID: 9, Type: slider, State: visible

ID: 10, Type: slider, State: invisible

Searching element 6: visible

First invisible: 2

No adjacent controls of same state found.

Count the number of visible controls using count: 4

Number of disabled in slider: 1

Equal: No