### **Task 1:**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include<string>

using namespace std;

struct Control {

int id;

string type;

string state;

bool operator==(const Control& other) const {

return (id == other.id) && (type == other.type) && (state == other.state);

}

};

void printControl(const Control& ctrl) {

cout << "ID: " << ctrl.id << ", Type: " << ctrl.type << ", State: " << ctrl.state <<endl;

}

int main() {

vector<Control> controls = {

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

{2, "slider", "invisible"},

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

{4, "slider", "visible"},

{5, "button", "disabled"},

{6, "slider", "invisible"},

{7, "button", "visible"},

{8, "slider", "visible"},

{9, "button", "disabled"},

{10, "slider", "visible"}

};

cout << "All Controls:" <<endl;

for\_each(controls.begin(), controls.end(), printControl);

int targetId = 5;

auto foundControl = find\_if(controls.begin(), controls.end(),

[targetId](const Control& ctrl) { return ctrl.id == targetId; });

if (foundControl != controls.end()) {

cout << "\nFound control with ID " << targetId << ": ";

printControl(\*foundControl);

} else {

cout << "\nControl with ID " << targetId << " not found!" <<endl;

}

auto invisibleControl =find\_if(controls.begin(), controls.end(),

[](const Control& ctrl) { return ctrl.state == "invisible"; });

if (invisibleControl != controls.end()) {

cout << "\nFirst invisible control: ";

printControl(\*invisibleControl);

} else {

cout << "\nNo invisible controls found!" <<endl;

}

auto adjacentControl =adjacent\_find(controls.begin(), controls.end(),

[](const Control& a, const Control& b) {

return a.state == b.state;

});

if (adjacentControl != controls.end()) {

cout << "\nFound consecutive controls with the same state: ";

printControl(\*adjacentControl);

printControl(\*(adjacentControl + 1));

} else {

cout << "\nNo consecutive controls with the same state found!" <<endl;

}

int visibleCount =count\_if(controls.begin(), controls.end(),

[](const Control& ctrl) { return ctrl.state == "visible"; });

cout << "\nNumber of visible controls: " << visibleCount <<endl;

int disabledSlidersCount =count\_if(controls.begin(), controls.end(),

[](const Control& ctrl) { return ctrl.type == "slider" && ctrl.state == "disabled"; });

cout << "Number of disabled sliders: " << disabledSlidersCount <<endl;

vector<Control> controlsSubset1 = {controls[1], controls[2], controls[3]};

vector<Control> controlsSubset2 = {controls[0], controls[1], controls[2]};

if (equal(controlsSubset1.begin(), controlsSubset1.end(), controlsSubset2.begin())) {

cout << "\nThe two subranges are identical!" <<endl;

} else {

cout << "\nThe two subranges are not identical!" << endl;

}

return 0;

}

All Controls:

ID: 1, Type: button, State: visible

ID: 2, Type: slider, State: invisible

ID: 3, Type: button, State: disabled

ID: 4, Type: slider, State: visible

ID: 5, Type: button, State: disabled

ID: 6, Type: slider, State: invisible

ID: 7, Type: button, State: visible

ID: 8, Type: slider, State: visible

ID: 9, Type: button, State: disabled

ID: 10, Type: slider, State: visible

Found control with ID 5: ID: 5, Type: button, State: disabled

First invisible control: ID: 2, Type: slider, State: invisible

Found consecutive controls with the same state: ID: 7, Type: button, State: visible

ID: 8, Type: slider, State: visible

Number of visible controls: 5

Number of disabled sliders: 0

The two subranges are not identical!

### **Task 2:**

#include <iostream>

#include <vector>

#include <set>

#include <string>

#include <algorithm>

#include <iterator>

using namespace std;

int main() {

vector<string> dynamicWidgets = {"Speedometer", "Tachometer", "FuelGauge", "GPS"};

set<string> staticWidgets = {"Logo", "WarningLights", "TimeDisplay"};

cout << "Dynamic Widgets:\n";

for (auto it = dynamicWidgets.begin(); it != dynamicWidgets.end(); it++) {

cout << \*it << "\n";

}

cout<<endl;

cout<<"enter a specific widget to find in set"<<endl;

string widgetToFind;

cin>>widgetToFind;

auto found = staticWidgets.find(widgetToFind);

if (found != staticWidgets.end()) {

cout << widgetToFind << " is found in static widgets.\n";

} else {

cout << widgetToFind << " is NOT found in static widgets.\n";

}

cout<<endl;

vector<string> combinedWidgets;

copy(dynamicWidgets.begin(), dynamicWidgets.end(), back\_inserter(combinedWidgets));

copy(staticWidgets.begin(), staticWidgets.end(), back\_inserter(combinedWidgets));

cout<<endl;

cout<<"Enter a specific widget to find in combined container"<<endl;

string specific\_widget;

cin>>specific\_widget;

auto combinedFound =find(combinedWidgets.begin(), combinedWidgets.end(), specific\_widget);

if (combinedFound != combinedWidgets.end()) {

cout << "Found 'GPS' in the combined widgets list.\n";

} else {

cout << "'GPS' NOT found in the combined widgets list.\n";

}

cout<<endl;

cout << "\nCombined Widgets List (Dynamically Printed):\n";

for (const auto& widget : combinedWidgets) {

cout << widget << "\n";

}

cout<<endl;

cout<<"Enter widget for searching"<<endl;

string widgetToSearch;

cin>>widgetToSearch;

auto searchResult =find(combinedWidgets.begin(), combinedWidgets.end(), widgetToSearch);

if (searchResult != combinedWidgets.end()) {

cout << widgetToSearch << " is found in the combined widgets list.\n";

} else {

cout << widgetToSearch << " is NOT found in the combined widgets list.\n";

}

return 0;

}

Dynamic Widgets:

Speedometer

Tachometer

FuelGauge

GPS

enter a specific widget to find in set

TimeDisplay

TimeDisplay is found in static widgets.

Enter a specific widget to find in combined container

Tachometer

Found 'GPS' in the combined widgets list.

Combined Widgets List (Dynamically Printed):

Speedometer

Tachometer

FuelGauge

GPS

Logo

TimeDisplay

WarningLights

Enter widget for searching

Logo

Logo is found in the combined widgets list.

### **Task 3:**

#include<iostream>

#include<vector>

#include<algorithm>

#include<random>

#include<string>

using namespace std;

int main()

{

vector<string> Control\_States={"slider","visible","invisible","disable","slider"};

cout<<"copy to create a backup of the control list."<<endl;

vector<string>backup(Control\_States.size());

copy(Control\_States.begin(),Control\_States.end(),backup.begin());

cout<<"Backup Controls:"<<endl;

for(auto Backup:backup)

{

cout<<Backup<<endl;

}

cout<<endl;

cout<<"fill to set all states to disabled temporarily."<<endl;

fill(Control\_States.begin(),Control\_States.end(),"disabled");

for(auto Control\_State:Control\_States)

{

cout<<Control\_State<<endl;

}

cout<<endl;

cout<<"generate to generate random states (visible, invisible, disabled) for testing."<<endl;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(0, 2);

vector<string>states={"visible","invisible","disable"};

generate(Control\_States.begin(), Control\_States.end(), [&]() {

return states[dis(gen)];

});

for(auto Control\_State:Control\_States)

{

cout<<Control\_State<<endl;

}

cout<<endl;

cout<<"transform to change the state of all sliders to invisible."<<endl;

transform(Control\_States.begin(), Control\_States.end(),Control\_States.begin(), [](string& state)

{

return (state == "slider")?"invisible":state;

});

for(auto Control\_State:Control\_States)

{

cout<<Control\_State<<endl;

}

cout<<endl;

cout<<"replace to replace disabled with enabled for testing."<<endl;

replace(Control\_States.begin(),Control\_States.end(),"disable","enabled");

for(auto Control\_State:Control\_States)

{

cout<<Control\_State<<endl;

}

cout<<endl;

cout<<"reverse to reverse the control order "<<endl;

reverse(Control\_States.begin(),Control\_States.end());

for(auto Control\_State:Control\_States)

{

cout<<Control\_State<<endl;

}

}

copy to create a backup of the control list.

Backup Controls:

slider

visible

invisible

disable

slider

fill to set all states to disabled temporarily.

disabled

disabled

disabled

disabled

disabled

generate to generate random states (visible, invisible, disabled) for testing.

disable

invisible

visible

invisible

invisible

transform to change the state of all sliders to invisible.

disable

invisible

visible

invisible

invisible

replace to replace disabled with enabled for testing.

enabled

invisible

visible

invisible

invisible

reverse to reverse the control order

invisible

invisible

visible

invisible

enabled

### **Task 4:**

#include<iostream>

#include<string>

#include<vector>

#include<algorithm>

using namespace std;

class Control

{

public:

int id;

string name;

Control()

{

}

Control(int id,string name)

{

this->id=id;

this->name=name;

}

bool operator<(const Control& other) const {

return id < other.id;

}

void print()

{

cout<<"ID: "<<id<< " name: "<<name<<endl;

}

};

int main()

{

vector<Control>controls1={ Control(2, "Slider"),

Control(1, "Button"),

Control(2, "Checkbox"),

Control(5, "Label")

};

vector<Control> controls2 = {

Control(4, "Textbox"),

Control(1, "Button"),

Control(6, "RadioButton"),

Control(3, "Checkbox")

};

cout<<"Sort the controls by their ID using sort."<<endl;

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

for(auto control:controls1)

{

control.print();

}

cout<<endl;

cout<<"stable\_sort to maintain relative order for controls with equal IDs."<<endl;

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

for(auto control:controls1)

{

control.print();

}

cout<<endl;

cout<<"Search for a control by ID using lower\_bound"<<endl;

auto it=lower\_bound(controls1.begin(),controls1.end(),Control(3," "));

if(it!=controls1.end())

{

cout<<"found"<<endl;

it->print();

}

else

{

cout<<"not found"<<endl;;

}

cout<<endl;

cout<<"Search for a control by ID using upper\_bound"<<endl;

auto it1=upper\_bound(controls1.begin(),controls1.end(),Control(1," "));

if(it1!=controls1.end())

{

cout<<"found"<<endl;

it1->print();

}

else

{

cout<<"not found"<<endl;;

}

cout<<endl;

cout<<"Merge two sorted lists of controls using merge."<<endl;

vector<Control>ctrl3(controls1.size() + controls2.size());

merge(controls1.begin(),controls1.end(),controls2.begin(),controls2.end(),ctrl3.begin());

for(auto ctrl4:ctrl3)

{

ctrl4.print();

}

}

Sort the controls by their ID using sort.

ID: 1 name: Button

ID: 2 name: Slider

ID: 2 name: Checkbox

ID: 5 name: Label

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

ID: 1 name: Button

ID: 2 name: Slider

ID: 2 name: Checkbox

ID: 5 name: Label

Search for a control by ID using lower\_bound

found

ID: 5 name: Label

Search for a control by ID using upper\_bound

found

ID: 2 name: Slider

Merge two sorted lists of controls using merge.

ID: 1 name: Button

ID: 2 name: Slider

ID: 2 name: Checkbox

ID: 4 name: Textbox

ID: 1 name: Button

ID: 5 name: Label

ID: 6 name: RadioButton

ID: 3 name: Checkbox