Codes

#include <iostream>

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

struct BoundNumbersList {

int bound;

Node\* head;

BoundNumbersList(int value) : bound(value), head(nullptr) {}

};

std::unordered\_map<int, BoundNumbersList\*> lists;

void createList(int bound) {

if (lists.find(bound) == lists.end()) {

BoundNumbersList\* newList = new BoundNumbersList(bound);

lists[bound] = newList;

std::cout << "Bound-numbers list with bound " << bound << " created.\n";

} else {

std::cout << "List with bound " << bound << " already exists.\n";

}

}

void insertNumber(int bound, int value) {

if (lists.find(bound) != lists.end()) {

if (value <= bound) {

Node\* newNode = new Node(value);

newNode->next = lists[bound]->head;

lists[bound]->head = newNode;

std::cout << "Number " << value << " inserted into the list with bound " << bound << ".\n";

} else {

std::cout << "Value exceeds the bound.\n";

}

} else {

std::cout << "List with bound " << bound << " does not exist.\n";

}

}

void deleteLastItem(int bound) {

if (lists.find(bound) != lists.end()) {

if (lists[bound]->head) {

Node\* temp = lists[bound]->head;

if (!temp->next) {

delete temp;

lists[bound]->head = nullptr;

std::cout << "Last item deleted from the list with bound " << bound << ".\n";

return;

}

while (temp->next->next) {

temp = temp->next;

}

delete temp->next;

temp->next = nullptr;

std::cout << "Last item deleted from the list with bound " << bound << ".\n";

} else {

std::cout << "List is empty.\n";

}

} else {

std::cout << "List with bound " << bound << " does not exist.\n";

}

}

void deleteAllExceptItem(int bound, int value) {

if (lists.find(bound) != lists.end()) {

Node\* temp = lists[bound]->head;

Node\* prev = nullptr;

while (temp) {

if (temp->data != value) {

Node\* toDelete = temp;

temp = temp->next;

if (prev) {

prev->next = temp;

} else {

lists[bound]->head = temp;

}

delete toDelete;

} else {

prev = temp;

temp = temp->next;

}

}

std::cout << "Deleted all items except " << value << " from the list with bound " << bound << ".\n";

} else {

std::cout << "List with bound " << bound << " does not exist.\n";

}

}

void reversePrintList(Node\* node) {

if (node) {

reversePrintList(node->next);

std::cout << node->data << " ";

}

}

void displayLists() {

std::cout << "Bound values of stored lists: ";

for (const auto& entry : lists) {

std::cout << entry.first << " ";

}

std::cout << "\n";

}

void displayListItems(int bound) {

if (lists.find(bound) != lists.end()) {

std::cout << "Items in the list with bound " << bound << ": ";

Node\* temp = lists[bound]->head;

while (temp) {

std::cout << temp->data << " ";

temp = temp->next;

}

std::cout << "\n";

} else {

std::cout << "List with bound " << bound << " does not exist.\n";

}

}

void searchForList(int bound) {

if (lists.find(bound) != lists.end()) {

std::cout << "List with bound " << bound << " found.\n";

} else {

std::cout << "List with bound " << bound << " does not exist.\n";

}

}

void mergeLists(int bound1, int bound2) {

if (lists.find(bound1) != lists.end() && lists.find(bound2) != lists.end()) {

int mergedBound = std::max(bound1, bound2);

createList(mergedBound);

Node\* temp1 = lists[bound1]->head;

Node\* temp2 = lists[bound2]->head;

while (temp1) {

insertNumber(mergedBound, temp1->data);

temp1 = temp1->next;

}

while (temp2) {

insertNumber(mergedBound, temp2->data);

temp2 = temp2->next;

}

std::cout << "Lists with bounds " << bound1 << " and " << bound2 << " merged into a new list with bound " << mergedBound << ".\n";

} else {

std::cout << "One or both of the specified lists do not exist.\n";

}

}

int main() {

int choice;

do {

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

std::cout << "1. Create a new bound-numbers list\n";

std::cout << "2. Insert a number in a list\n";

std::cout << "3. Delete the last item in a list\n";

std::cout << "4. Delete all items except a particular item in a list\n";

std::cout << "5. Display numbers in a list in reverse order\n";

std::cout << "6. Display bound values of stored lists\n";

std::cout << "7. Display items in a bound-numbers list\n";

std::cout << "8. Search for a list\n";

std::cout << "9. Merge two lists\n";

std::cout << "10. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1: {

int bound;

std::cout << "Enter the bound value for the new list: ";

std::cin >> bound;

createList(bound);

break;

}

case 2: {

int bound, value;

std::cout << "Enter the bound value of the list: ";

std::cin >> bound;

std::cout << "Enter the value to insert: ";

std::cin >> value;

insertNumber(bound, value);

break;

}

case 3: {

int bound;

std::cout << "Enter the bound value of the list: ";

std::cin >> bound;

deleteLastItem(bound);

break;

}

case 4: {

int bound, value;

std::cout << "Enter the bound value of the list: ";

std::cin >> bound;

std::cout << "Enter the value to keep: ";

std::cin >> value;

deleteAllExceptItem(bound, value);

break;

}

case 5: {

int bound;

std::cout << "Enter the bound value of the list: ";

std::cin >> bound;

std::cout << "Items in reverse order: ";

reversePrintList(lists[bound]->head);

std::cout << "\n";

break;

}

case 6:

displayLists();

break;

case 7: {

int bound;

std::cout << "Enter the bound value of the list: ";

std::cin >> bound;

displayListItems(bound);

break;

}

case 8: {

int bound;

std::cout << "Enter the bound value to search for: ";

std::cin >> bound;

searchForList(bound);

break;

}

case 9: {

int bound1, bound2;

std::cout << "Enter the bound value of the first list: ";

std::cin >> bound1;

std::cout << "Enter the bound value of the second list: ";

std::cin >> bound2;

mergeLists(bound1, bound2);

break;

}

case 10:

std::cout << "Exiting the program.\n";

break;

default:

std::cout << "Invalid choice. Please try again.\n";

}

} while (choice != 10);

// Clean up memory (delete lists and nodes)

for (auto& entry : lists) {

Node\* current = entry.second->head;

while (current) {

Node\* temp = current;

current = current->next;

delete temp;

}

delete entry.second;

}

return 0;

}

Improved code

#include <iostream>

using namespace std;

struct Node {

int value;

int bound;

Node\* next;

Node(int val, int b) : value(val), bound(b), next(nullptr) {}

};

class BoundNumbersList {

private:

Node\* head;

public:

BoundNumbersList() : head(nullptr) {}

void createList(int b) {

bool found = false;

Node\* curr = head;

while (curr) {

if (curr->bound == b) {

found = true;

break;

}

curr = curr->next;

}

if (!found) {

head = new Node(0, b);

cout << "New list with bound value " << b << " created." << endl;

} else {

cout << "A list with bound value " << b << " already exists." << endl;

}

}

void insertFront(int b, int val) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

Node\* newNode = new Node(val, b);

newNode->next = curr->next;

curr->next = newNode;

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

void insertMiddle(int b, int val, int after) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

Node\* temp = curr->next;

while (temp && temp->value != after)

temp = temp->next;

if (temp) {

Node\* newNode = new Node(val, b);

newNode->next = temp->next;

temp->next = newNode;

} else {

cout << "Value " << after << " not found in the list." << endl;

}

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

void deleteFront(int b) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr && curr->next) {

Node\* temp = curr->next;

curr->next = temp->next;

delete temp;

} else {

cout << "List with bound value " << b << " not found or is empty." << endl;

}

}

void deleteNode(int b, int index) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

Node\* prev = nullptr;

Node\* temp = curr->next;

int i = 0;

while (temp && i < index) {

prev = temp;

temp = temp->next;

i++;

}

if (temp) {

if (prev)

prev->next = temp->next;

else

curr->next = temp->next;

delete temp;

} else {

cout << "Index out of range." << endl;

}

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

BoundNumbersList\* merge(BoundNumbersList& other) {

int newBound = max(head->bound, other.head->bound);

BoundNumbersList\* merged = new BoundNumbersList();

merged->head = new Node(0, newBound);

Node\* curr1 = head->next;

Node\* curr2 = other.head->next;

Node\* curr = merged->head;

while (curr1 && curr2) {

if (curr1->value < curr2->value) {

curr->next = new Node(curr1->value, newBound);

curr = curr->next;

curr1 = curr1->next;

} else if (curr2->value < curr1->value) {

curr->next = new Node(curr2->value, newBound);

curr = curr->next;

curr2 = curr2->next;

} else {

curr->next = new Node(curr1->value, newBound);

curr = curr->next;

curr1 = curr1->next;

curr2 = curr2->next;

}

}

while (curr1) {

curr->next = new Node(curr1->value, newBound);

curr = curr->next;

curr1 = curr1->next;

}

while (curr2) {

curr->next = new Node(curr2->value, newBound);

curr = curr->next;

curr2 = curr2->next;

}

return merged;

}

void displayBounds() {

cout << "Bound values of stored lists:" << endl;

Node\* curr = head;

while (curr) {

cout << curr->bound << " ";

curr = curr->next;

}

cout << endl;

}

void displayList(int b) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

cout << "List with bound value " << b << ": ";

curr = curr->next;

while (curr) {

cout << curr->value << " ";

curr = curr->next;

}

cout << endl;

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

void insertEnd(int b, int val) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

Node\* temp = curr->next;

while (temp && temp->value <= curr->bound)

temp = temp->next;

if (temp || val <= curr->bound) {

Node\* newNode = new Node(val, b);

newNode->next = temp;

if (!temp) {

while (curr->next)

curr = curr->next;

curr->next = newNode;

} else {

Node\* prev = curr->next;

while (prev->next != temp)

prev = prev->next;

prev->next = newNode;

}

} else {

cout << "Value " << val << " exceeds the bound value " << curr->bound << "." << endl;

}

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

void deleteLast(int b) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr && curr->next) {

Node\* temp = curr->next;

if (!temp->next) {

curr->next = nullptr;

delete temp;

} else {

while (temp->next->next)

temp = temp->next;

Node\* last = temp->next;

temp->next = nullptr;

delete last;

}

} else {

cout << "List with bound value " << b << " not found or is empty." << endl;

}

}

void deleteAllExcept(int b, int val) {

Node\* curr = head;

while (curr && curr->bound != b)

curr = curr->next;

if (curr) {

Node\* temp = curr->next;

Node\* prev = curr;

while (temp) {

if (temp->value == val) {

prev = temp;

temp = temp->next;

} else {

prev->next = temp->next;

Node\* toDelete = temp;

temp = temp->next;

delete toDelete;

}

}

} else {

cout << "List with bound value " << b << " not found." << endl;

}

}

};

int main() {

BoundNumbersList list;

int choice, bound, val, after, index;

do {

cout << "1. Create a new list" << endl;

cout << "2. Insert a node in front" << endl;

cout << "3. Insert a node in the middle" << endl;

cout << "4. Delete the front node" << endl;

cout << "5. Delete a node" << endl;

cout << "6. Merge two lists" << endl;

cout << "7. Display bound values" << endl;

cout << "8. Display a list" << endl;

cout << "9. Insert at the end" << endl;

cout << "10. Delete the last node" << endl;

cout << "11. Delete all nodes except one" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the bound value: ";

cin >> bound;

list.createList(bound);

break;

case 2:

cout << "Enter the bound value: ";

cin >> bound;

cout << "Enter the value to insert: ";

cin >> val;

list.insertFront(bound, val);

break;

case 3:

cout << "Enter the bound value: ";

cin >> bound;

cout << "Enter the value to insert: ";

cin >> val;

cout << "Enter the value to insert after: ";

cin >> after;

list.insertMiddle(bound, val, after);

break;

case 4:

cout << "Enter the bound value: ";

cin >> bound;

list.deleteFront(bound);

break;

case 5:

cout << "Enter the bound value: ";

cin >> bound;

cout << "Enter the index of the value to delete (0-based): ";

cin >> index;

list.deleteNode(bound, index);

break;

case 6:

cout << "Merge operation not implemented in this version." << endl;

break;

case 7:

cout << "Display bound values not implemented in this version." << endl;

break;

case 8:

cout << "Enter the bound value: ";

cin >> bound;

list.displayList(bound);

break;

case 9:

cout << "Enter the bound value: ";

cin >> bound;

cout << "Enter the value to insert at the end: ";

cin >> val;

list.insertEnd(bound, val);

break;

case 10:

cout << "Enter the bound value: ";

cin >> bound;

list.deleteLast(bound);

break;

case 11:

cout << "Enter the bound value: ";

cin >> bound;

cout << "Enter the value to keep: ";

cin >> val;

list.deleteAllExcept(bound, val);

break;

case 0:

cout << "Exiting the program." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 0);

return 0;

}