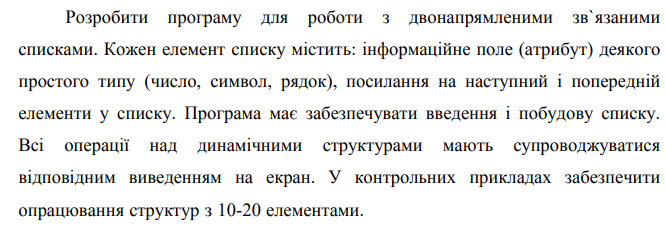
**Лабораторна робота №1**

Робота з динамічними структурами двонапрямленими зв`язаними списками

**Мета роботи**: набуття практичних навичок опрацювання таких динамічних структур, як двонапрямлені зв`язані списки.

*Завдання*





**Хід роботи**

1. **Текст програми**

***main.cpp:***

#include <iostream>

#include <string>

#include "List.hpp"

void writeList(dg::List &list);

void printList(dg::List &list);

void insert(dg::List& list);

template <typename T>

T prompt(const char label[]);

int main() {

dg::List list;

writeList(list);

printList(list);

insert(list);

printList(list);

system("pause");

return 0;

}

void insert(dg::List& list) {

std::cout << "Enter items to insert: " << std::endl;

auto next = list.front();

while (next) {

auto current = next;

next = next->next;

list.insert(current, prompt<int>("> "));

}

}

void writeList(dg::List &list) {

int size = prompt<unsigned int>("Enter size of the List: ");

std::cout << "Enter items:" << std::endl;

for (int i = 0; i < size; i++) {

list.pushBack(prompt<int>("> "));

}

}

void printList(dg::List &list) {

std::cout << "List: ";

auto next = list.front();

while (next) {

std::cout << next->value << " ";

next = next->next;

}

std::cout << std::endl;

}

template <typename T>

T prompt(const char label[]) {

std::cout << label;

while (true) {

T val;

std::cin >> val;

if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Wrong. Try again: ";

}

else {

std::cin.ignore(32767, '\n');

return val;

}

}

}

***List.hpp***

#pragma once

namespace dg {

class List {

public:

List();

struct Node {

int value = 0;

Node\* next = nullptr;

Node\* prev = nullptr;

};

Node\* front();

Node\* back();

void pushFront(int value);

void pushBack(int value);

void popFront();

void popBack();

Node\* getByIndex(int index);

// Вставляет элемент в список после node

void insert(Node\* node, int value);

// Вставляет элемент в список после переданного индекса

void insert(int index, int value);

void removeItem(Node\* node);

void removeItem(int index);

int size();

bool isEmpty();

void clear();

~List();

private:

Node\* m\_front;

Node\* m\_back;

int m\_size = 0;

};

}; // namespace dg

***List.cpp***

#include "List.hpp"

namespace dg {

List::List() {

}

List::Node\* List::front() {

return m\_front;

}

List::Node\* List::back() {

return m\_back;

}

void List::pushFront(int value) {

auto newNode = new Node();

newNode->value = value;

newNode->next = m\_front;

if (m\_front) {

m\_front->prev = newNode;

}

else {

m\_back = newNode;

}

m\_front = newNode;

m\_size++;

}

void List::pushBack(int value) {

auto newNode = new Node();

newNode->value = value;

newNode->prev = m\_back;

if (m\_back) {

m\_back->next = newNode;

}

else {

m\_front = newNode;

}

m\_back = newNode;

m\_size++;

}

void List::popFront() {

if (!m\_front) return;

auto deleted = m\_front;

m\_front = m\_front->next;

delete deleted;

if (m\_front) {

m\_front->prev = nullptr;

}

else {

m\_back = nullptr;

}

m\_size--;

}

void List::popBack() {

if (!m\_back) return;

auto deleted = m\_back;

m\_back = m\_back->prev;

delete deleted;

if (m\_back) {

m\_back->next = nullptr;

}

else {

m\_front = nullptr;

}

m\_size--;

}

List::Node\* List::getByIndex(int index) {

if (index < 0 || index >= m\_size) return nullptr;

Node \* next = m\_front;

for (int i = 1; i <= index; i++) {

next = next->next;

}

return next;

}

void List::insert(Node\* node, int value) {

if (!node) return;

if (node == m\_back) {

pushBack(value);

return;

}

auto newNode = new Node();

newNode->value = value;

newNode->next = node->next;

newNode->prev = node;

if (node->next) node->next->prev = newNode;

node->next = newNode;

m\_size++;

}

void List::insert(int index, int value) {

insert(getByIndex(index), value);

}

void List::removeItem(Node \* node) {

if (!node) return;

if (node == m\_front) {

popFront();

return;

}

if (node == m\_back) {

popBack();

return;

}

if (node->prev) node->prev->next = node->next;

if (node->next) node->next->prev = node->prev;

delete node;

m\_size--;

}

void List::removeItem(int index) {

removeItem(getByIndex(index));

}

int List::size() {

return m\_size;

}

bool List::isEmpty() {

return m\_size == 0;

}

void List::clear() {

Node\* prev = m\_back;

while (prev) {

auto deleted = prev;

prev = prev->prev;

delete deleted;

}

m\_front = nullptr;

m\_back = nullptr;

}

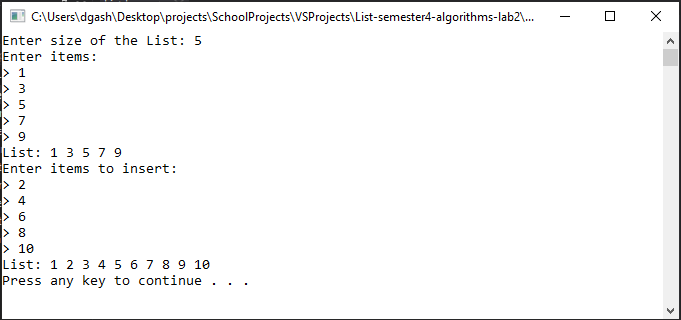
List::~List() {

clear();

}

}; // namespace dg

1. **Результат виконання програми**



**Висновок:** на цій лабораторній роботі я набув практичні навички опрацювання таких динамічних структур, як двонапрямлені зв`язані списки.