|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А.В. Поляков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать функции для работы с двусвязным кольцевым списком.

Функции:

1. Добавление: добавление в список происходит в начало, например, в список 1<->5<->8<-> добавляем 2 и получаем 2<->1<->5<->8.
2. Удаление: удаление представляет собой удаление всех вхождений заданного элемента, у которого совпадает data-поле с введённым значением. Например, есть список 1<->2<->2<->5<->8, при применении функции со значением 2 получаем список   
   1<->5<->8.
3. Разность двух списков: разность двух списков заносится в третий. Например, есть два списка: 1<->2<->3 и 2<->5<->6, их разность заносится в третий: 1<->3.
4. Распечатка списка на экран: выводит список на экран.
5. Нахождение заданного элемента в списке: при первом нахождении элемента с заданным data-полем функция выводит 1. При отсутствии совпадений функция выводит 0.
6. Удаление всего списка: удаление всего списка.

**2 Описание типа + спецификация подпрограмм + тесты**

struct Node {

int data;

Node\* prev;

Node\* next;

};

Node\* head = nullptr

Структура узлов списка. Имеет поля: data типа int – хранит целое число в информационном поле, указатель на next типа Node – хранит указатель на следующий узел списка, указатель на prev типа Node – хранит указатель на предыдущий узел списка.

**function** display1 (Node \*head): void;

Выводит список на экран. Проходит по указателям на next-узлы. Если список пуст (head == nullptr), то выводит сообщение «Список пуст.».

Входные данные: указатель на список. Выходные: вывод значений элементов списка на экран/вывод сообщения «Список пуст.».

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Распечатка пустого списка | head (head == nullptr) | Список пуст. |
| Распечатка не пустого списка | head (1 2 3) | 1 2 3 |

**function** display2 (Node \*head): void;

Выводит список на экран. Проходит по указателям на prev-узлы. Если список пуст (head == nullptr), то выводит сообщение «Список пуст.».

Входные данные: указатель на список. Выходные: вывод значений элементов списка на экран/вывод сообщения «Список пуст.».

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Распечатка пустого списка | head (head == nullptr) | Список пуст. |
| Распечатка не пустого списка | head (1 2 3) | 3 2 1 |

**function** createNode(int value): Node\*;

Создает и возвращает новый узел с заданным значением data-поля и указателями prev и next на nullptr

**function** appendToBegining(Node\*& head, int value): void;

Вставляет в список элементы в начало. С помощью функции createNode создает новый узел. Если головы списка нет (не иницилизирована), добавляемый элемент становится головой списка. Если до добавления голова уже была, то она перемещается на значение нового узла.

Входные данные: указатель на указатель головы списка, значение. Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Вставка в пустой список | head(), 10 | 10 |
| Вставка элемента в не пустой список | head (1 2 5 20), 10 | 10 1 2 5 20 |

**function** deleteAllChosenElements(Node\*& head, int value): void;

Удаляет все элементы списка с data-полем равным int value. Если список пустой, то программа выходит из функции и выводит сообщение на экран: «Список пуст. Невозможно удалить элементы.».

Входные данные: ссылка на указатель головы списка, значение. Выходные данные: список с удалёнными элементами.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление из пустого списка | head (), 10 | ()  «Список пуст. Невозможно удалить элементы.» |
| Удаление из списка с одним элементом | head (10), 10 | () |
| Удаление из не пустого списка | head (1 2 3 3 3 5), 3 | 1 2 5 |
| Удаление из списка элемента со значением, которого нет в списке | head (1 2 3 20), 10 | 1 2 3 20 |

**function** containsElement(Node\* head, int value): bool;

Ищет в списке элемент, data-поле которого равно заданному значению int value. Если находит такой, то выводит true, иначе выводит false.

Входные данные: указатель на голову списка, значение. Выходные данные: true/false.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Поиск в пустом списке | head (), 10 | false |
| Поиск в не пустом списке (элемент есть в списке) | head (1 2 3 10 20), 10 | true |
| Поиск в не пустом списке (элемента нет в списке) | head (1 2 3 20), 10 | false |

**function** clearNode(Node\*& head): void;

Удаляет все элементы списка таким образом: сначала все элементы, которые идут после головы, а затем голову (head == nullptr). Если список пуст, то программа выводит сообщение «Список пуст. Чистить нечего.» и выходит из функции.

Входные данные: ссылка на указатель головы списка. Выходные данные: пустой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление пустого списка | head () | () |
| Удаление не пустого списка | head (1 2 3 4) | () |

**function** differenceList(Node\* head1, Node\* head2): Node\*;

Находит разность элементов первого и второго списка и возвращает значение в новый третий список. Если оба списка (1-й и 2-й) пусты, то в 3-й список ничего не добавляется. Если второй список пуст, то в 3-й добавляются элементы первого. Если первый список пуст, тогда третий тоже будет пуст.

Входные данные: указатель на голову списка1 и списка2. Выходные данные: третий список с результатом разности.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Оба списка пусты | head1(), head2() | head3() |
| Второй список пуст | head1(1 2 3), head2() | head3(1 2 3) |
| Оба списка не пусты | head1(1 2 3), head2(2 5 6) | head3 (1 3) |
| Первый список пуст | head1(), head2(1 2 3) | head3() |

**3 Текст программы**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

};

Node\* createNode(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->prev = nullptr;

newNode->next = nullptr;

return newNode;

}

void appendToBegining(Node\*& head, int value) {

Node\* newNode = createNode(value);

if (!head) {

head = newNode;

head->next = head;

head->prev = head;

} else {

Node\* tail = head->prev;

newNode->next = head;

newNode->prev = tail;

head->prev = newNode;

tail->next = newNode;

head = newNode;

}

}

void deleteAllChosenElements(Node\*& head, int value) { // сделать для одного элемента

bool isHeadMoved = false;

Node\* current = head;

if (!head) {

cout << "Список пуст.\nНевозможно удалить элементы." << endl;

return;

}

do {

if (current->data == value) {

isHeadMoved = false;

Node\* temp = current;

current = current->next;

current->prev = temp->prev;

current->prev->next = current;

delete temp;

if (head == temp) {

head = current;

isHeadMoved = true;

}

} else {

current = current->next;

}

} while ((current != head) or (isHeadMoved));

}

void display1(Node\* head) {

if (!head) {

cout << "Список пуст." << endl;

return;

}

Node\* current = head;

do {

cout << current->data << " ";

current = current->next;

} while (current != head);

cout << endl;

}

void display2(Node\* head) {

if (!head) {

cout << "Список пуст." << endl;

return;

}

Node\* current = head->prev;

do {

cout << current->data << " ";

current = current->prev;

} while (current != head->prev);

cout << endl;

}

void clearNode(Node\*& head) {

Node\* current = head;

if (!head) {

cout << "Список пуст. Чистить нечего." << endl;

return;

}

do {

Node\* temp = current;

current = current->next;

delete temp;

} while (current != head);

}

bool containsElement(Node\* head, int value) {

if (!head) {

return false;

}

Node\* current = head;

do {

if (current->data == value) {

return true;

}

current = current->next;

} while (current != head);

return false;

}

Node\* differenceList(Node\* head1, Node\* head2) {

Node\* head = nullptr;

if (!head1) {

return head;

}

Node\* current1 = head1;

do {

if (!containsElement(head2, current1->data)) {

appendToBegining(head, current1->data);

}

current1 = current1->next;

} while (current1 != head1);

return head;

}

int main() {

Node\* head = nullptr;

Node\* head2 = nullptr;

Node\* head3 = differenceList(head, head2);

clearNode(head);

clearNode(head2);

clearNode(head3);

return 0;

}