|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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. Б9121-09.03.04прогин | | | |
|  |  |  | | Д.Ю. Литовченко | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2022 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

**Двухсвязный не упорядоченный список, с добавлением в конец, могут быть повторяющиеся элементы, удаление перед каждым вхождением заданного, операция пересечения двух списков.**

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

struct list\* **add\_node**(struct list\* list, int value)

Добавляет в конец списка ***list*** узел со значением ***value***, возвращает исходный список ***list*** с добавленным элементом.

Входные данные: список, значение. Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление элемента в пустой список | lst {}, 2 | lst {2}(при выводе списка на экран выводится элемент со значением 2) |
| Добавление элемента в заполненный список | lst {4 2}, 5 | lst {4 2 5} (при выводе списка на экран выводятся элементы 4 2 5) |

struct list\* **del**(struct list\* list, int value)

Удаляет из списка ***list*** узел перед значением ***value***, возвращает исходный список ***list*** с удалённым элементом.

Входные данные: список, значение. Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление элемента в пустом списке | lst {}, 2 | lst {}(при выводе списка на экран ничего не выводится) |
| Удаление первого элемента | lst {5 4 3}, 4 | lst {4 3} (при выводе списка на экран выводятся элементы 4 3) |
| Удаление не существующего элемента из середины списка | lst {5 4 3}, 2 | lst {5 4 3} (выводит исходный список без изменения) |
| Попытка удаление элемента, стоящего перед первым | lst {5 4 3}, 5 | lst {5 4 3} (выводит исходный список) |
| Удаление пред последнего элемента | lst {5 4 3}, 3 | lst {5 3} (выводит список без элемента перед заданным) |

struct list\* **Clear\_List**(list\* lst)

Полная очистка списка ***lst***.

Входные данные: список. Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Очистка пустого списка | lst {} | lst {} (ничего не изменится) |
| Очистка заполненного списка | lst {6 2 3} | lst {} (все элементы очищаются) |

void **Listprint\_Head**(list\* lst)

Печать списка ***lst*** с начала.

Входные данные: список. Выходные данные: печать списка.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Печать пустого списка | lst {} | {} (ничего не напечатается) |
| Печать заполненного списка | Lst {6 2 3} | {6 2 3} (печать элементов списка) |

void **Listprint\_Tail**(list\* lst)

Печать списка ***lst*** с конца.

Входные данные: список. Выходные данные: печать списка.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Печать пустого списка | lst {} | {} (ничего не напечатается) |
| Печать заполненного списка | Lst {6 2 3} | {3 2 6} (печать элементов списка) |

bool search(list\* lst, int a)

Поиск элемента со значением ***a*** в списке ***lst***. Если элемент найден, то возвращает **True**, иначе – **False**.

Входные данные: список, значение. Выходные данные: True/False.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Поиск элемента в пустом списке | lst {}, 2 | 0 (False) |
| Поиск существующего элемента в заполненном списке | lst {7 3 5}, 5 | 1 (True) |
| Поиск не существующего элемента в заполненном списке | Lst {7 3 5}, 2 | 0 (False) |

struct list\* Intersection(list\* st1, list\* nd2, list\* finall)

Нахождение элементов с одинаковым значением в списках ***st1*** и ***nd2*** и их добавление в список ***final.***

Входные данные: список, список, список. Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Пересечение двух пустых списков | st1 {}, nd2 {}, finall {} | finall {} |
| Пересечение двух списков с общими элементами | st1 {7 4 2}, nd2 {4 1 7}, finall {} | finall {4 7} |
| Пересечение двух списков без общих элементов | st1 {9 4 6}, nd2 {1 2 3}, finall {} | finall {} |

**3 Текст программы**

#include <corecrt\_malloc.h>

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node

{

int field;

struct node\* next;

struct node\* prev;

};

struct list

{

struct node\* head;

struct node\* tail;

};

struct list\* add\_node(struct list\* list, int value)

{

struct node\* tmp = (struct node\*) malloc(sizeof(struct node));

tmp->field = value;

tmp->next = nullptr;

node\* buffer = list->tail;

if (list->head == nullptr)

{

list->head = tmp;

list->tail = tmp;

}

else

{

buffer->next = tmp;

tmp->prev = buffer;

list->tail = tmp;

}

return(list);

}

struct list\* del(struct list\* list, int value)

{

auto c = list->head;

if (c != nullptr)

{

while ((c->next != nullptr) and (c->field != value))

c = c->next;

if (c->field == value)

{

if (c->field == list->head->next->field)

{

list->head = c;

list->head->prev = nullptr;

free(c->prev);

}

else if (c->prev)

{

c = c->prev;

if (c->prev != nullptr)

c->prev->next = c->next;

if (c->next != nullptr)

c->next->prev = c->prev;

free(c);

}

}

}

return(list);

}

struct list\* Clear\_List(list\* lst)

{

if (lst->head != nullptr)

while (lst->head != nullptr)

{

auto tmp = lst->head;

lst->head = lst->head->next;

free(tmp);

}

return(lst);

}

void Listprint\_Head(list\* lst)

{

for (struct node\* current = lst->head; current; current = current->next)

{

printf("%d ", current->field);

}

cout << endl;

}

void Listprint\_Tail(list\* lst)

{

for (struct node\* current = lst->tail; current; current = current->prev)

{

printf("%d ", current->field);

}

cout << endl;

}

bool search(list\* lst, int a)

{

if (lst->head != nullptr)

{

auto current = lst->head;

while (current->next != nullptr && current->field != a)

{

current = current->next;

}

if (current->field == a)

return(true);

}

else

return false;

}

struct list\* Intersection(list\* st1, list\* nd2, list\* finall)

{

list\* st, \* nd;

st = st1;

nd = nd2;

for (struct node\* current1 = st1->head; current1; current1 = current1->next)

{

for (struct node\* current2 = nd2->head; current2; current2 = current2->next)

{

if (current1->field == current2->field)

{

add\_node(finall, current2->field);

}

}

}

return(finall);

}

int main()

{

struct list lst = { NULL, NULL };

struct list lst2 = { NULL, NULL };

struct list lst3 = { NULL, NULL };

cout << search(&lst, 1) << endl;

for (int i = 1; i < 7; i++)

{

add\_node(&lst, i);

}

for (int i = 4; i < 8; i++)

{

add\_node(&lst2, i);

}

cout << search(&lst, 1) << endl;

add\_node(&lst2, 1);

Listprint\_Head(&lst);

Listprint\_Head(&lst2);

Intersection(&lst, &lst2, &lst3);

Listprint\_Head(&lst3);

Listprint\_Head(&lst);

lst.head->prev = nullptr;

del(&lst, 1);

Listprint\_Head(&lst);

Clear\_List(&lst);

Listprint\_Head(&lst);

}