|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Кафедра прикладной математики, механики, управления и программного обеспечения** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе № 1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9121-09.03.04прогин | | | |
|  |  |  | | Р.А. Башинский | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2022 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Односвязанный упорядоченный кольцевой список без повторов.

Пакет процедур\функций:

1) Инициализация

2) Очистка памяти ( удаление списка)

3) Добавление элемента

4) Поиск заданного элемента

6) Печать списка

7) Удаление заданного элемента (по значению).

8) Объединение 2х списков.

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

void Clear\_List() (Метод класса DynList)

Очищает односвязанный список с головой **head** типа **Node**, поочередно удаляя элементы типа **Node**, присваивая голове значение **Null**.

Входные данные: Значение, объект класса

Выходные данные: Объект класса

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Очистка пустого списка | (Null) | (Null) |
| Очистка непустого списка | (2, 3, 4) | Пустой список (Null) |

void Delete\_Node(int data) (Метод класса DynList)

Удаляет элемент Node из списка с головой **head** типа **Node** со значением переменной **data** типа **integer**.

Входные данные: Значение, объект класса

Выходные данные: Объект класса

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление элемента в пустом списке | (2), (Null) | (Null) |
| Удаление сущ. элемента из начала непустого списка | (2), (2, 3, 4) | (3, 4) |
| Удаление сущ. элемента из середины непустого списка | (3), (2, 3, 4) | (2, 4) |
| Удаление сущ. элемента из конца непустого списка | (4), (2, 3, 4) | (2, 3) |
| Удаление несущ. элемента из непустого списка | (0), (2, 3, 4) | (2, 3, 4) |

bool Find\_Node(int data) (Метод класса DynList)

Находит элемент **Node** в списке с головой **head** типа **Node** со значением переменной **data** типа **integer**. Возвращает **true**, если элемент найден, и **false**, если элемент не найден или список пуст.

Входные данные: Значение, объект класса

Выходные данные: true/false

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Поиск элемента в пустом списке | (2), (Null) | false |
| Поиск сущ. элемента в непустом списке | (3), (2, 3, 4) | true |
| Поиск несущ. элемента в непустом списке | (0), (2, 3, 4) | false |

void Add\_Node(int data) (Метод класса DynList)

Добавляет элемент **Node** в список с головой **head** типа **Node** со значением переменной **data** типа **integer**. Элемент вставляется (упорядочивается) по возрастанию значения элемента. Элементы с повторяющимся значением в список не добавляются.

Входные данные: Значение, объект класса

Выходные данные: объект класса

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление элемента в пустой список | (2), (Null) | (2) |
| Добавление элемента в начало непустого списка | (2), (3, 4) | (2, 3, 4) |
| Добавление элемента в середину непустого списка | (4), (3, 5) | (3, 4, 5) |
| Добавление элемента в конец непустого списка | (5), (3, 4) | (3, 4, 5) |

void Print\_List() (Метод класса DynList)

Печатает список с головой head типа Node в консоль, указывая текущий адрес элемента, его значение и ссылку на слудующий элемент

Входные данные: Объект класса

Выходные данные: Объект класса

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Вывод пустого списка | (Null) | Сообщение, что список пуст |
| Вывод непустого списка | (2, 3, 4) | Вывод значений (2, 3, 4), адресов и указателей элементов |

**3 Текст программы**

#include <iostream>

using namespace std;

class DynList final

{

struct Node

{

int data;

Node\* next;

};

private:

Node\* head;

public:

DynList()

{

head = NULL;

};

void Clear\_List()

{

if (head != NULL)

{

Node\* temp;

while (head->next != head)

{

temp = head->next->next;

delete head->next;

head->next = temp;

}

delete head;

head = NULL;

}

};

void Delete\_Node(int data)

{

Node\* current = head;

if (current != NULL)

{

while (current->next->data != data && current->next != head)

{

current = current->next;

}

if (current->next->data == data)

{

Node\* temp = current->next->next;

if (current->next == head)

{

head = current->next->next;

}

delete current->next;

current->next = temp;

}

}

};

bool Find\_Node(int data)

{

Node\* current = head;

bool flag = false;

if (current == NULL)

{

return false;

}

else

{

if (current->data == data)

{

return true;

}

else

{

while (current->next != head && current->next->data != data)

{

current = current->next;

}

if (current->next->data == data)

{

return true;

}

else

{

return false;

}

}

}

};

void Add\_Node(int data)

{

Node\* current = head;

bool flag = false;

if (current == NULL)

{

Node\* cn = new Node;

cn->data = data;

cn->next = NULL;

head = cn;

head->next = head;

}

else

{

if (data < current->data && data != current->data)

{

while (current->next != head)

{

current = current->next;

}

Node\* cn = new Node;

cn->data = data;

cn->next = head;

current->next = cn;

head = cn;

}

else if (data > current->data)

{

while (current->next != head && current->next->data < data)

{

current = current->next;

}

if (current->next->data != data)

{

Node\* cn = new Node;

cn->data = data;

cn->next = current->next;

current->next = cn;

}

}

}

};

void Print\_List()

{

cout << "------------------------------" << endl;

Node\* current = head;

if (current == NULL)

{

cout << "DynList is empty." << endl;

}

else

{

do

{

cout << "adress: " << current << " ";

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

cout << "next: " << current->next << endl;

current = current->next;

}

while (current != head);

}

cout << "------------------------------" << endl;

};

void Merge(DynList List1, DynList List2)

{

Node\* current1 = List1.head;

Node\* current2 = List2.head;

if (current1 != NULL)

{

do

{

Add\_Node(current1->data);

current1 = current1->next;

} while (current1 != List1.head);

}

if (current2 != NULL)

{

do

{

Add\_Node(current2->data);

current2 = current2->next;

} while (current2 != List2.head);

}

};

};

int main()

{

DynList List\_One;

List\_One.Add\_Node(32);

List\_One.Add\_Node(2);

List\_One.Add\_Node(2);

List\_One.Add\_Node(2);

List\_One.Add\_Node(7);

List\_One.Add\_Node(10);

List\_One.Add\_Node(240);

List\_One.Add\_Node(1);

List\_One.Add\_Node(66);

List\_One.Print\_List();

cout << "------------------------------" << endl;

cout << "Find node 22: " << List\_One.Find\_Node(22) << endl;

cout << "Find node 7: " << List\_One.Find\_Node(7) << endl;

cout << "------------------------------" << endl;

List\_One.Delete\_Node(1);

List\_One.Print\_List();

//List\_One.Clear\_List();

List\_One.Print\_List();

DynList List\_Two;

List\_Two.Add\_Node(4);

List\_Two.Add\_Node(5);

List\_Two.Add\_Node(61);

List\_Two.Add\_Node(544);

List\_Two.Print\_List();

DynList Merged;

Merged.Merge(List\_One, List\_Two);

Merged.Print\_List();

return 0;

};