

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Отчет по выполнению практического задания № 1.6

# Тема:

Двунаправленные динамические списки

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Тринеев П. С.

Фамилия И.О.

Группа: ИКБО-32-22

Номер группы

Москва – 2023

**Цель:** получение знаний и практических навыков управления двунаправленным списком в программах на языке С++.

1. Постановка задачи – условие задания, требования в соответствии с вариантом

**1 Задание.** Разработать многомодульную программу, которая демонстрирует выполнение всех операций, определенных вариантом, над линейным двунаправленным динамическим списком.

Требования к разработке.

1. Разработать структуру узла списка, структура информационной части узла определена вариантом. Для определения структуры узла списка, используйте тип struct или class. Сохраните определение структуры узла и прототипы функций в заголовочном файле.

2. Разработайте функции для выполнения операции над линейным двунаправленным динамическим списком:

• создание списка;

• вставку узла;

• удаление узла;

• вывод списка в двух направлениях (слева направо и справа налево);

• поиск узла с заданным значением (операция должна возвращать указатель на узел с заданным значением).

3. Дополнительные операции над списком, указанные вариантом, оформите в виде функций и включите в отдельный файл с расширением cpp. Подключите к этому файлу заголовочный файл с определением структуры узла.

4. Разработайте программу, управляемую текстовым меню, и включите в меню демонстрацию выполнения всех операций задания и варианта.

5. Проведите тестирование операций.

• Оцените сложность алгоритма первой дополнительной операции.

6. Оформите отчет по разработке программы в соответствии с требованиями задания по однонаправленному списку.

**Задание 2:**

Вывести зачетную книжку студентов, в выводе использовать номер зачетный книжки (буквенно-цифровой), номер группы, оценка. После удалить все записи, где ученикам проставлен неуд. (2) и вывести изменный список.

1. **Список** — это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза.

**Двунаправленный (двусвязный) список** – это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы. При этом два соседних элемента должны содержать взаимные ссылки друг на друга.

Основными операциями над списками являются:

* переход к очередному элементу списка;
* добавление в список нового элемента;
* поиск заданного элемента;
* удаление элемента из списка.

Выполнение этих операций основывается на использовании и изменении указателей.

В отличие от очередей и стеков, элемент в список может быть до­бавлен в любое место.

2.1 Структура узда – имеет указатель на следующий и предыдущий элемент массива, а также тип int хранящийся информации.

2.2 Рисунки выполнения операции на двунаправленном списке

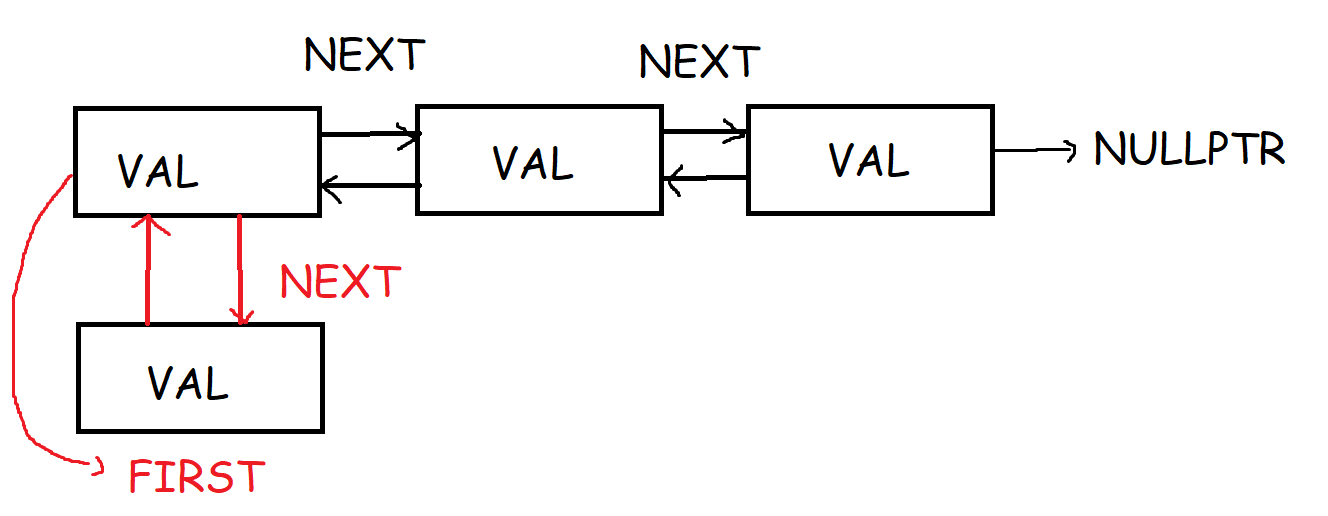


Рис. 1 схема операции вставки узла перед первым узлом

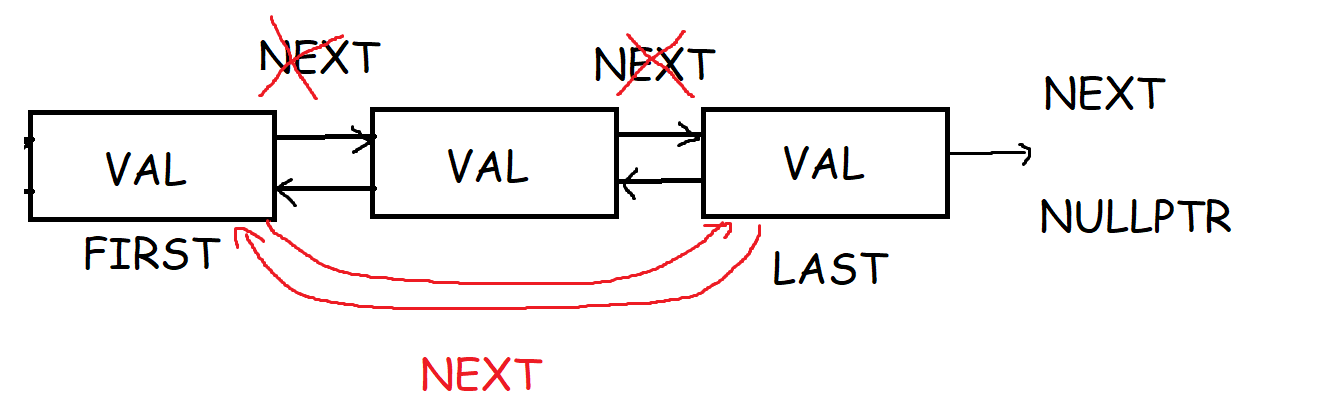


Рис. 2 схема операции удаления узла по ключу (общий случай)

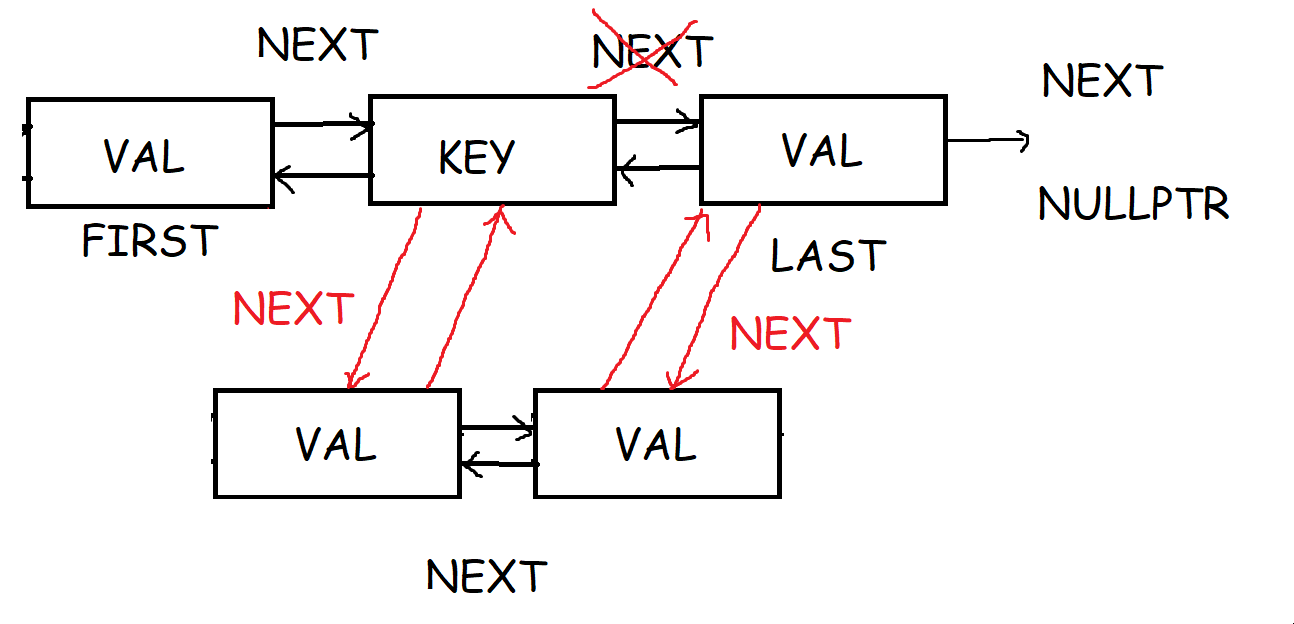


Рис. 3 схема операции вставки в список L за узлом

2.3 Изобразите структуру данных, которая будет использоваться в операциях

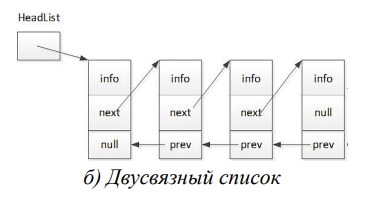


Рис. 4 Структура данных.

* 1. Привести таблицу тестов для тестирования каждой операции

|  |  |  |
| --- | --- | --- |
| Задания | Вводимые значения | Выводимые значения |
| 1 задание | {"1", {1, 5}},{"2", {1, 5}},{"3", {2, 5}},{"1", {1, 2}} | ID: 1. Group ID: 1. Marc: 2  ID: 1. Group ID: 1. Marc: 5  ID: 2. Group ID: 1. Marc: 5  ID: 3. Group ID: 2. Marc: 5 |
| 2 задание | {"1", {1, 5}},{"2", {1, 5}},{"3", {2, 5}},{"1", {1, 2}} | ID: 1. Group ID: 1. Marc: 5  ID: 2. Group ID: 1. Marc: 5  ID: 3. Group ID: 2. Marc: 5 |
| 3 задание | {"1", {1, 5}},{"2", {1, 5}},{"3", {2, 5}},{"1", {1, 2}} | ID: 1. Group ID: 1. Marc: 2 |
| 4 задание | {"1", {1, 5}},{"2", {1, 5}},{"3", {2, 5}},{"1", {1, 2}} | Remove element  ID: 1. Group ID: 1. Marc: 5 ID: 2. Group ID: 1. Marc: 5 |

1. Представить код программы на языке С++.

#include <fstream>

#include <string>

#include <vector>

#include "iostream"

using namespace std;

template<typename T>

struct Node

{

T val;

Node\* next;

Node\* prev;

Node(T \_val) : val(\_val), next(nullptr), prev(nullptr) {};

};

/\* sorts the linked list by changing next pointers (not data) \*/

template<typename T>

void MergeSort(Node<T>\*\* headRef)

{

Node<T>\* head = \*headRef;

Node<T>\* a;

Node<T>\* b;

/\* Base case -- length 0 or 1 \*/

if ((head == NULL) || (head->next == NULL)) {

return;

}

/\* Split head into 'a' and 'b' sublists \*/

FrontBackSplit(head, &a, &b);

/\* Recursively sort the sublists \*/

MergeSort(&a);

MergeSort(&b);

/\* answer = merge the two sorted lists together \*/

\*headRef = SortedMerge(a, b);

}

/\* See https:// www.geeksforgeeks.org/?p=3622 for details of this

function \*/

template<typename T>

Node<T>\* SortedMerge(Node<T>\* a, Node<T>\* b)

{

Node<T>\* result = NULL;

/\* Base cases \*/

if (a == NULL)

return (b);

else if (b == NULL)

return (a);

/\* Pick either a or b, and recur \*/

if (a->val <= b->val) {

result = b;

result->next = SortedMerge(a, b->next);

}

else {

result = a;

result->next = SortedMerge(a->next, b);

}

return (result);

}

/\* UTILITY FUNCTIONS \*/

/\* Split the nodes of the given list into front and back halves,

and return the two lists using the reference parameters.

If the length is odd, the extra node should go in the front list.

Uses the fast/slow pointer strategy. \*/

template<typename T>

void FrontBackSplit(Node<T>\* source,

Node<T>\*\* frontRef, Node<T>\*\* backRef)

{

Node<T>\* fast;

Node<T>\* slow;

slow = source;

fast = source->next;

/\* Advance 'fast' two nodes, and advance 'slow' one node \*/

while (fast != nullptr) {

fast = fast->next;

if (fast != nullptr) {

slow = slow->next;

fast = fast->next;

}

}

/\* 'slow' is before the midpoint in the list, so split it in two

at that point. \*/

\*frontRef = source;

\*backRef = slow->next;

slow->next = nullptr;

}

template<typename T>

struct NodeList

{

Node<T>\* first;

Node<T>\* last;

NodeList() : first(nullptr), last(nullptr) {}

bool is\_empty()

{

return first == nullptr;

}

void push\_back(T \_val)

{

Node<T>\* p = new Node(\_val);

if (is\_empty())

{

first = p;

last = p;

return;

}

last->next = p;

p->prev = last;

last = p;

}

void push\_back(Node<T>\* new\_current)

{

if (is\_empty())

{

first = new\_current;

last = new\_current;

return;

}

last->next = new\_current;

new\_current->prev = last;

last = new\_current;

}

void print(bool reverse = false)

{

if (is\_empty()) return;

Node<T>\* p;

if (reverse)

p = last;

else

p = first;

while (p)

{

std::cout << p->val << " ";

if (reverse)

{

if (p->prev == p)

break;

p = p->prev;

}

else

{

if (p->next == p)

break;

p = p->next;

}

}

std::cout << "\n";

}

Node<T>\* find(T& \_val)

{

if (first != nullptr && first->val == \_val)

{

return first;

}

if (last != nullptr && last->val == \_val)

{

return last;

}

Node<T>\* p = first;

Node<T>\* p2 = last;

while (true)

{

if (p && p->val != \_val)

{

p = p->next;

}

if (!p)

break;

if (p->val == \_val) break;

if (p2 && p->val != \_val)

{

p2 = p2->prev;

}

if (!p2)

break;

if (p2->val == \_val)

{

p = p2;

break;

}

if (p == p2)

break;

}

return (p && p->val == \_val) ? p : nullptr;

}

void remove\_first(bool delete\_node = true)

{

Node<T>\* old = first;

if (first->next)

{

first->next->prev = nullptr;

first = first->next;

}

else

{

first = nullptr;

}

old->prev = nullptr;

old->next = nullptr;

if (delete\_node)

delete old;

}

void remove\_last(bool delete\_node = true)

{

if (last->prev)

{

Node<T>\* p = last->prev;

p->next = nullptr;

p->prev = nullptr;

if (delete\_node)

delete last;

last = p;

}

else

{

if (delete\_node)

delete last;

last = nullptr;

}

}

void remove\_element(Node<T>\* current, bool delete\_node = true)

{

if (first == current)

{

remove\_first(delete\_node);

return;

}

if (last == current)

{

remove\_last(delete\_node);

return;

}

current->prev->next = current->next;

current->next->prev = current->prev;

current->next = nullptr;

current->prev = nullptr;

if (delete\_node)

delete current;

}

void remove(T \_val)

{

if (is\_empty()) return;

if (first->val == \_val)

{

remove\_first();

}

else if (last->val == \_val)

{

remove\_last();

return;

}

Node<T>\* slow = first;

Node<T>\* fast = first->next;

while (true)

{

if (slow->next && slow->next->val != \_val)

{

slow = slow->next;

}

if (slow->val == \_val)

{

remove\_element(slow);

break;

}

if (!slow->next)

break;

if (fast->prev && fast->prev->val != \_val)

{

fast = fast->prev;

}

if (fast->val == \_val)

{

remove\_element(fast);

break;

}

if (!fast->next)

break;

if (slow == fast)

break; // not found

}

}

void sort\_list()

{

MergeSort<T>(&first);

rebuild\_nodes();

}

void rebuild\_nodes()

{

Node<T>\* elm = first;

Node<T>\* lst\_elm = nullptr;

elm->prev = nullptr;

while (elm != nullptr)

{

if (elm->next)

{

elm->next->prev = elm;

lst\_elm = elm->next;

}

elm = elm->next;

}

last = lst\_elm;

while (lst\_elm != nullptr)

{

if (lst\_elm->prev)

lst\_elm->prev->next = lst\_elm;

lst\_elm = lst\_elm->prev;

}

}

};

struct my\_struct

{

std::string id;

int group, mark;

bool cmp;

my\_struct(std::string \_id, int \_group, int \_mark, bool \_cmp) : id(\_id), group(\_group),

mark(\_mark), cmp(\_cmp) {};

bool operator==(const my\_struct& a)

{

if (cmp && a.cmp)

return mark == a.mark;

return group == a.group;

}

bool operator!=(const my\_struct& a)

{

if (cmp && a.cmp)

return mark != a.mark;

return group != a.group;

}

bool operator<=(const my\_struct& a)

{

return id >= a.id;

}

};

std::ostream& operator<<(std::ostream& os, const my\_struct& num)

{

os << "ID: " << num.id << ". Group ID: " << num.group << ". Mark: " << num.mark << "\

n";

return os;

}

NodeList<my\_struct>\* create\_my\_struct(std::vector<std::pair<std::string, std::pair<int, int>>>&

data)

{

using namespace std;

int i = 0;

NodeList<my\_struct>\* lst = new NodeList<my\_struct>;

for (auto c : data)

{

lst->push\_back(my\_struct(c.first, c.second.first, c.second.second, true));

}

return lst;

}

/\*

\*Removes from one list and place into another list element with specific mark.

\*Input: NodeList<my\_struct>& MyStruct list form which we remove;

\*NodeList<my\_struct>& NewMyStruct list for which we remove;

\*const int& \_val mark what we search.

\*Output: nothing

\*/

void remove\_and\_copy(NodeList<my\_struct>& MyStruct, NodeList<my\_struct>& NewMyStruct,

const int& \_val)

{

my\_struct struct\_to\_search = my\_struct("dont care", 0, \_val, true);

Node<my\_struct>\* find = MyStruct.find(struct\_to\_search);

while (find)

{

if (find)

{

MyStruct.remove\_element(find, false);

NewMyStruct.push\_back(find);

}

find = MyStruct.find(struct\_to\_search);

}

}

/\*

\*Removes from one list and place into another list element with specific mark.

\*Input: NodeList<my\_struct>& MyStruct list form which we remove;

\*const ing& \_val group what we search.

\*Output: nothing

\*/

void remove\_els(NodeList<my\_struct>& MyStruct, const int& \_val)

{

my\_struct struct\_to\_search = my\_struct("dont care", \_val, 0, false);

Node<my\_struct>\* find = MyStruct.find(struct\_to\_search);

while (find)

{

if (find)

{

MyStruct.remove\_element(find);

}

find = MyStruct.find(struct\_to\_search);

}

}

void main()

{

using namespace std;

vector<std::pair<std::string, std::pair<int, int>>> data = { {"1", {1, 5}},{"2", {1, 5}},{"3",

{2, 5}},{"1", {1, 2}}, };

NodeList<my\_struct>\* lst = create\_my\_struct(data);

NodeList<my\_struct>\* new\_lst = new NodeList<my\_struct>;

if (lst->is\_empty())

{

cout << "my\_struct is emppty\n";

return;

}

lst->sort\_list();

lst->print();

remove\_and\_copy(\*lst, \*new\_lst, 2);

lst->print();

new\_lst->print();

cout << "remove element\n";

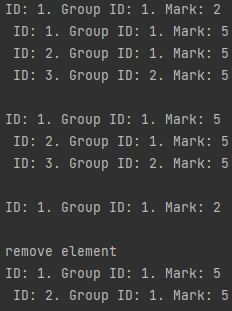
remove\_els(\*lst, 2);

lst->sort\_list();

lst->print();

}

1. Представить результат тестирования программы: скриншоты выполнения каждой операции



1. Привести выводы по полученным знания и умениям.

Вывод: мы получили знания и практические навыки управления

двунаправленным списком в программах на языке С++.