# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра САПР**

# ОТЧЕТ

**по лабораторной работе №1**

# по дисциплине «Алгоритмы и структуры данных»

**вариант 10**

Студент гр. 2301: Мещеряков М.А.

Преподаватель: Пестерев Д.О.

Санкт-Петербург

2023

# 1. Постановка задачи

Реализовать объект в виде односвязного списка с набором методов/функций:

1. Добавление в конец списка.
2. Добавление в начало списка.
3. Удаление последнего элемента.
4. Удаление первого элемента.
5. Добавление элемента по индексу.
6. Получение элемента по индексу.
7. Удаление элемента по индексу.
8. Получение размера списка.
9. Удаление всех элементов списка.
10. Замена элемента по индексу на передаваемый элемент.

11.Проверка на пустоту списка.

12.Меняет порядок элементов в списке на обратный. 13.Вставка другого списка в список, начиная с индекса. 14.Вставка другого списка в конец.

1. Вставка другого списка в начало.
2. Проверка на содержание другого списка в списке. 17.Поиск первого вхождения другого списка в список. 18.Поиск последнего вхождения другого списка в список. 19.Обмен двух элементов списка по индексам.

# Описание алгоритма

Программа «Реализация односвязного списка» написана на языке программирования C++ при помощи компилятора Visual Studio 2022.

Программа реализует односвязный список.

Каждая функция/метод списка в программе реализована с помощью собственной функции, в некоторые из них передаются определенные параметры

Список реализуется с помощью класса Node и класса SingleLinkedList. Класc Node состоит из значения и указателя на следующий элемент. Класс SingleLinkedList имеет два указателя (head, tail). Head - указывает на первый элемент списка, а tail - на последний элемент списка.

# 2. Описание функций/методов:

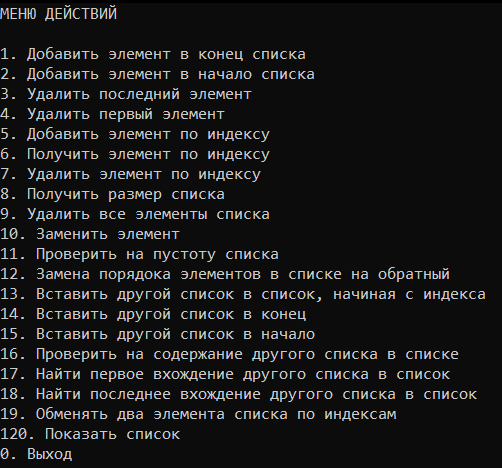
|  |  |  |
| --- | --- | --- |
| Идентификатор | Описание | Вр-нная сложность  алгоритма |
| void ClearList() | Очистка списка | О(n) |
| bool EmptyCheck() | Проверка на пустоту списка | О(1) |
| void AddBack(int data) | Добавление элемента в конец списка.  Передаваемые параметры: значение элемента | О(1) |
| void DeleteBack() | Удаление последнего элемента списка | О(1) |
| void DeleteFirst() | Удаление первого элемента списка. | О(1) |
| void Insert(int index, int data) | Добавление элемента по индексу.  Алгоритм: Вначале получаем адрес элемента с индексом с помощью метода GetByIndex (). Если такого объекта нет, то метод insert() завершает свою работу. Иначе через указатель right сохраняем адрес следующего после left объекта, создаем новый объект Node с данными data. Указатель next объекта left приравниваем node, а у node next приравниваем right. В итоге настраиваем все необходимые связи односвязного списка. В конце делаем проверку, если right равен NULL, то вставляется последний элемент (в конец списка), значит, нужно переместить указатель tail в конец, т.е. присвоить ему значение node. | О(n) |
| Node\* GetByIndex(int index) | Нахождение элемента по индексу.  Алгоритм: Если значение index меньше нуля, то возвращается NULL, т.к. такого индекса не существует. Далее, создаем временный указатель node, который изначально равен указателю head. Также создается счетчик n для индексов. Запускается цикл, в котором проверяем, существует ли объект node и если да, то дополнительно проверяем, чтобы n было не равно index и указатель next текущего объекта был не равен NULL. То есть, если будет найден элемент с индексом index, то цикл завершится. Также цикл завершится если будет достигнут конец списка. В конце возвращаем либо адрес найденного элемента, если n равно k, либо значение NULL, если объект с указанным индексом отсутствует в списке. | О(n) |

|  |  |  |
| --- | --- | --- |
| void Erase(int index) | Удаление элемента по индексу. Вначале проверяем корректность индекса, если он отрицателен, то метод завершает свою работу. Если index равен нулю, то удаляется первый элемент, а для этого уже есть метод DeleteFirst(). Его мы и вызываем. В противном случае, получаем указатель на index-1 элемент. Затем, указатель node указывает на index-й удаляемый элемент. Если он не существует, то метод erase() завершает свою работу. Иначе формируем еще один указатель right на объект после удаляемого. Далее, указатель next должен ссылаться на right и в конце проверка, если удаляется последний элемент, то tail нужно перевести на предпоследний. В конце освобождаем память из-под удаляемого элемента.  Передаваемые параметры: индекс. | О(n) |
| void AddFirst(int data) | Добавление элемента в начало  Алгоритм: создается новый объект списка. Указатель next этого объекта должен вести на объект head. После этого указатель head перемещаем на начало списка и еще проверяем, если добавляется первый элемент в список (tail равен NULL), то tail должен быть равен node. Передаваемые параметры: значение элемента | О(1) |
| void Replace(int index, int data) | Изменение значения элемента по индексу. Находим элемент по индексу и меняем его значение.  Передаваемые параметры: индекс, передаваемое значение. | О(n) |
| void Reverse() | Порядок элементов списка становится обратным. Алгоритм: добавляем в этот же список элементы в обратном порядке, в прошлом порядке удаляем.  Передаваемые параметры: объект двусвязного списка | О(n) |
| void InsertListByIndex() | Вставка одного списка в другой по индексу. | О(𝑛) |
| void InsertListBeginnig() | Вставка одного списка в другой в начало.  Передаваемые параметры: объект двусвязного списка | О(n) |
| void InsertListBack() | Вставка одного списка в другой в конец. | О(n) |
| void FindLastOccur(SingleLinkedList& main\_list) | поиск последнего вхождения списка в список. Алгоритм: вводим список, проверяемый на вхождение, затем проверяем на вхождение и запоминаем элемент, входящий в оба списка. В конце выводим его  Передаваемые данные: односвязный список | О(m + n\*m) |
| void ShowByIndex(int index) | Вывод элемента.  Алгоритм: получаем узел при помощи GetByIndex() и есл значение не null, то выводим элемент  Передаваемое значение: индекс элемента | O(n) |
| unsigned LengthOfList() | Вывод длины списка. | O(n) |
| void ShowList() | Вывод списка. | O(n) |
| void Swap() | Обмен двух элементов по индексам. | O(n) |
| void FindFirstOccur(SingleLinkedList& main\_list) | поиск первого вхождения списка в список. Алгоритм: вводим список, проверяемый на вхождение, затем проверяем на вхождение и выводим его.  Передаваемые данные: односвязный список | О(m + n\*m) |
| bool СheckListInList(SingleLinkedList& main\_list) | Проверка последнего вхождения списка в список. Алгоритм: вводим список, проверяемый на вхождение, затем проверяем на вхождение.  Передаваемые данные: односвязный список | О(m + n\*m) |

* 1. **Пример работы программы**



*Рисунок 1 Пример ввода элементов*



*Рисунок 2 Меню*

# Листинг

#include<iostream>

using namespace std;

enum CMD {

CMD\_erase,

CMD\_insert,

CMD\_length,

CMD\_empty\_check,

CMD\_delete\_first,

CMD\_add\_first,

CMD\_add\_back,

CMD\_delete\_back,

CMD\_clear,

CMD\_show,

CMD\_show\_list,

CMD\_replace,

CMD\_reverse,

CMD\_insert\_list\_by\_index,

CMD\_insert\_list\_back,

CMD\_insert\_list\_beginnig,

CMD\_check\_list\_in,

CMD\_find\_first\_occur,

CMD\_find\_last\_occur,

CMD\_swap,

CMD\_exit,

CMD\_none

};

class SingleLinkedList //Односвязный список

{

private:

class Node // Узел

{

public:

int data;

Node\* next;

public:

Node(int data)

{

this->data = data;

this->next = nullptr;

}

};

public:

Node\* head, \* tail;

public:

SingleLinkedList()

{

this->head = this->tail = nullptr;

}

~SingleLinkedList()

{

ClearList();

}

void ClearList()

{

while (head != nullptr)

{

DeleteFirst();

}

}

bool EmptyCheck() // Проверка на пустоту списка

{

return (head == nullptr);

}

void DeleteFirst() // Удалние первого

{

if (EmptyCheck()) return;

if (head == tail) // Проверка, что в списке всего один элемент

{

delete tail;

head = tail = nullptr;

return;

}

Node\* node = head;

head = node->next; //перемещаем указатель head на следующий и освобождаем память первого элемента

delete node;

}

void DeleteBack()

{

if (EmptyCheck()) return;

if (head == tail) // Если в списке всего 1 элемент

{

delete tail;

head = tail = nullptr;

return;

}

Node\* node = head;

for (; node->next != tail; node = node->next); // перемещаемся к предпоследнему элементу

node->next = nullptr;

delete tail;

tail = node;

}

void AddBack(int data)

{

Node\* node = new Node(data); // Cоздание элемента

if (head == nullptr) head = node; // Если добавляеся первый элемент, то head ссылается на этот элемент

if (tail != nullptr) tail->next = node; //

tail = node; // перемещение tail в конец списка

}

void AddFirst(int data)

{

Node\* node = new Node(data);

node->next = head;

head = node;

if (tail == nullptr) tail = node; // Если добавляется первый элемент, то выпонляем tail = node

}

Node\* GetByIndex(int index) // функция класса, возвращающая указатель на элемент

{

if (index < 0) return nullptr;

Node\* node = head;

unsigned cntr = 0;

while (node && cntr != index && node->next) // существует ли объект node, неравенство индекса и счетчика и указатель next текущего объекта был не равен nullptr

{

node = node->next;

cntr++;

}

return (cntr == index) ? node : nullptr;

}

void Insert(int index, int data)

{

Node\* left = GetByIndex(index);

if (left == nullptr) return;

Node\* right = left->next;

Node\* node = new Node(data);

left->next = node;

node->next = right;

if (right == nullptr) tail = node;

}

void Erase(int index)

{

if (EmptyCheck()) return;

if (index < 0) return;

if (index == 1)

{

DeleteFirst();

return;

}

Node\* left = GetByIndex(index - 2);

Node\* node = left->next;

if (node == nullptr) return;

Node\* right = node->next;

left->next = right;

if (node == tail) tail = left;

delete node;

}

void ShowByIndex(int index)

{

if (EmptyCheck()) return;

Node\* node = GetByIndex(index-1);

int d = (node != nullptr) ? node->data : 0;

std::cout << d << std::endl;

}

unsigned LengthOfList()

{

unsigned cntr = 0;

for (Node\* node = head; node != nullptr; node = node->next)

{

cntr++;

}

return cntr;

}

void ShowList()

{

if (EmptyCheck()) return;

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

}

void Replace(int index, int data)

{

if (EmptyCheck()) return;

Node \*node = GetByIndex(index-1);

node->data = data;

}

void Reverse()

{

if (EmptyCheck()) return;

int cntr = LengthOfList() - 1;

int temp;

Node\* node = head;

int saved\_cntr = cntr;

for (cntr; cntr >= 0; cntr--)

{

node = GetByIndex(cntr);

temp = (node != nullptr) ? node->data : 0;

AddBack(temp);

}

for (int i = 0; i <= saved\_cntr; i++)

{

DeleteFirst();

}

}

void InsertListBack()

{

int num\_of\_nodes;

int data;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

AddBack(data);

}

}

void InsertListBeginnig()

{

int num\_of\_nodes;

int data;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

cin >> data;

AddFirst(data);

for (int i = 0; i < num\_of\_nodes - 1; i++)

{

cin >> data;

Insert(i, data);

}

}

void Swap()

{

if (EmptyCheck()) return;

int first;

int second;

int temp\_data;

Node\* first\_node;

Node\* second\_node;

Node\* temp;

cout << "Введите индекс первого элемента \n";

cin >> first;

cout << "Введите индекс второго элемента \n";

cin >> second;

first\_node = GetByIndex(first-1);

second\_node = GetByIndex(second-1);

temp = GetByIndex(first - 1);

temp\_data = temp->data;

Replace(first, second\_node->data);

Replace(second, temp\_data);

}

void InsertListByIndex()

{

int index;

int num\_of\_nodes;

int data;

cout << "Введите индекс \n";

cin >> index;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

for (int i = 0, j = index - 1; i < num\_of\_nodes; i++, j++)

{

cin >> data;

Insert(j, data);

}

}

};

bool СheckListInList(SingleLinkedList& main\_list)

{

if (main\_list.EmptyCheck()) return false;

bool flag = false;

int occurence = 0;

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; i < num\_of\_list\_in\_list; i++)

{

for (in\_index; in\_index < num\_of\_nodes; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data)

{

occurence++;

}

}

if (occurence == num\_of\_nodes)

{

flag = true;

return flag;

}

in\_index += num\_of\_list\_in\_list;

occurence = 0;

}

return flag;

}

void FindFirstOccur(SingleLinkedList& main\_list)

{

bool flag = true;

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; (i < num\_of\_list\_in\_list) && flag; i++)

{

for (in\_index; (in\_index < num\_of\_nodes)&&flag; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data)

{

cout << "Элемент\n";

cout << list.GetByIndex(in\_index)->data;

cout << "\n Индекс\n";

cout << in\_index;

flag = false;

}

}

}

}

void FindLastOccur(SingleLinkedList& main\_list)

{

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data = 0;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; i < num\_of\_list\_in\_list; i++)

{

for (in\_index; in\_index < num\_of\_nodes; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data) {

data = list.GetByIndex(in\_index)->data;

}

}

in\_index += num\_of\_list\_in\_list;

}

cout << "Элемент\n";

cout << data;

cout << "\n Индекс\n";

cout << in\_index;

}

void ShowMenu()

{

cout << "МЕНЮ ДЕЙСТВИЙ\n\n";

cout << "1. Добавить элемент в конец списка\n";

cout << "2. Добавить элемент в начало списка\n";

cout << "3. Удалить последний элемент\n";

cout << "4. Удалить первый элемент\n";

cout << "5. Добавить элемент по индексу\n";

cout << "6. Получить элемент по индексу\n";

cout << "7. Удалить элемент по индексу\n";

cout << "8. Получить размер списка\n";

cout << "9. Удалить все элементы списка\n";

cout << "10. Заменить элемент\n";

cout << "11. Проверить на пустоту списка\n";

cout << "12. Замена порядока элементов в списке на обратный\n";

cout << "13. Вставить другой список в список, начиная с индекса\n";

cout << "14. Вставить другой список в конец\n";

cout << "15. Вставить другой список в начало\n";

cout << "16. Проверить на содержание другого списка в списке\n";

cout << "17. Найти первое вхождение другого списка в список\n";

cout << "18. Найти последнее вхождение другого списка в список\n";

cout << "19. Обменять два элемента списка по индексам\n";

cout << "120. Показать список\n";

cout << "0. Выход\n";

}

CMD user\_cmd\_input()

{

CMD incoming\_cmd = CMD\_none;

unsigned user\_input;

cin >> user\_input;

switch (user\_input)

{

case 0:

incoming\_cmd = CMD\_exit;

break;

case 1:

incoming\_cmd = CMD\_add\_back;

break;

case 2:

incoming\_cmd = CMD\_add\_first;

break;

case 3:

incoming\_cmd = CMD\_delete\_back;

break;

case 4:

incoming\_cmd = CMD\_delete\_first;

break;

case 5:

incoming\_cmd = CMD\_insert;

break;

case 6:

incoming\_cmd = CMD\_show;

break;

case 7:

incoming\_cmd = CMD\_erase;

break;

case 8:

incoming\_cmd = CMD\_length;

break;

case 9:

incoming\_cmd = CMD\_clear;

break;

case 10:

incoming\_cmd = CMD\_replace;

break;

case 11:

incoming\_cmd = CMD\_empty\_check;

break;

case 12:

incoming\_cmd = CMD\_reverse;

break;

case 13:

incoming\_cmd = CMD\_insert\_list\_by\_index;

break;

case 14:

incoming\_cmd = CMD\_insert\_list\_back;

break;

case 15:

incoming\_cmd = CMD\_insert\_list\_beginnig;

break;

case 16:

incoming\_cmd = CMD\_check\_list\_in;

break;

case 17:

incoming\_cmd = CMD\_find\_first\_occur;

break;

case 18:

incoming\_cmd = CMD\_find\_last\_occur;

break;

case 19:

incoming\_cmd = CMD\_swap;

break;

case 20:

incoming\_cmd = CMD\_show\_list;

break;

default:

cout << "Invalid input!\n";

incoming\_cmd = CMD\_none;

}

return incoming\_cmd;

}

int main()

{

setlocale(LC\_ALL, "Russian");

CMD user\_CMD = CMD\_none;

SingleLinkedList list;

int index;

int data;

bool flag = true;

while (flag)

{

system("cls");

ShowMenu();

user\_CMD = user\_cmd\_input();

switch (user\_CMD)

{

case CMD\_erase:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

list.Erase(index);

break;

case CMD\_insert:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

cout << "Введите значение\n";

cin >> data;

list.Insert(index-1, data);

break;

case CMD\_length:

system("cls");

cout << list.LengthOfList() << "\n";

cout << "\n\n";

system("pause");

break;

case CMD\_empty\_check:

system("cls");

(list.EmptyCheck()) ? cout << " Список пустой\n" : cout << "Список не пустой\n";

cout << "\n\n";

system("pause");

break;

case CMD\_delete\_first:

list.DeleteFirst();

break;

case CMD\_add\_first:

system("cls");

cout << "Введите значение\n";

cin >> data;

list.AddFirst(data);

break;

case CMD\_add\_back:

system("cls");

cout << "Введите значение\n";

cin >> data;

list.AddBack(data);

break;

case CMD\_delete\_back:

list.DeleteBack();

break;

case CMD\_clear:

list.ClearList();

break;

case CMD\_show:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

list.ShowByIndex(index);

cout << "\n\n";

system("pause");

break;

case CMD\_show\_list:

system("cls");

list.ShowList();

cout << "\n\n";

system("pause");

break;

case CMD\_replace:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

cout << "Введите значение\n";

cin >> data;

list.Replace(index, data);

break;

case CMD\_reverse:

list.Reverse();

break;

case CMD\_insert\_list\_by\_index:

system("cls");

list.InsertListByIndex();

break;

case CMD\_insert\_list\_back:

system("cls");

list.InsertListBack();

break;

case CMD\_insert\_list\_beginnig:

system("cls");

list.InsertListBeginnig();

break;

case CMD\_check\_list\_in:

system("cls");

if (СheckListInList(list))

{

cout << "Список есть в списке";

}

else

{

cout << "Список не входит в список";

}

cout << "\n\n";

system("pause");

break;

case CMD\_find\_first\_occur:

system("cls");

FindFirstOccur(list);

cout << "\n\n";

system("pause");

break;

case CMD\_find\_last\_occur:

system("cls");

FindLastOccur(list);

cout << "\n\n";

system("pause");

break;

case CMD\_swap:

system("cls");

list.Swap();

break;

case CMD\_exit:

flag = false;

break;

case CMD\_none:

break;

}

}

}

Ссылка на репозиторий: https://github.com/kombaen/MeshcheryakovAlLabs