**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЁТ**

**по лабораторной работе №1 (вар.10)**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Списки**

Студент гр. 7302 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руссу В.А.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пестерев Д.О.

Санкт-Петербург

2022

# Задача

Реализовать объект в виде двусвязного списка со следующими методами:

1. void AddTail(int data); // добавление в конец списка
2. void AddHead(int data); // добавление в начало списка
3. void DelTail(); // удаление последнего элемента
4. void DelHead(); // удаление первого элемента
5. void AddIndex(int index, int data); // добавление элемента по индексу
6. void GetIndex(int index); // получение элемента по индексу
7. void DelIndex(int index); // удаление элемента по индексу
8. void GetLength(); // получение размера списка
9. void DelAll(); // удаление всех элементов списка
10. void ReplaceData(int index, int data); // замена элемента по индексу на передаваемый элемент
11. void IfEmpty(); // проверка на пустоту списка
12. int Find(List\* sublist); // поиск первого вхождения другого списка в список

# Описание реализуемого класса и методов

Реализуемый класс List имеет public-методы, описанные выше, и поля указатели на голову и хвост списка, а также размер самого списка. Структура Node включает себя указатели на следующий элемент и на предыдущий, а также поле типа int для хранения данных.

void AddTail(int data) добавляет новый элемент в конец списка путем перенаправления указателей элементов. Временная сложность: О(1).

void AddHead(int data) добавляет новый элемент в начало списка путем перенаправления указателей элементов. Временная сложность: О(1).

void DelTail() удаляет последний элемент списка. Берем указатель на последний элемент списка из поля tail, присваиваем полю tail указатель на предыдущий элемент, а последний обнуляем. Временная сложность: О(1).

void DelHead() удаляет первый элемент списка. Берем указатель на первый элемент списка из поля head, присваиваем полю head указатель на следующий элемент, а первый обнуляем. Временная сложность: О(1).

void AddIndex(int index, int data) добавляет элемент по индексу. Принимает два значения индекс и дату, далее идут проверки на то, не является ли индекс отрицательным и не выходил ли он за пределы списка, если индекс больше длины списка на 1, то вызывается метод AddTail, если индекс равен 1, то вызывается метод AddHead, иначе программа пробежит до нужного индекса и вставит новый элемент перед ним. Временная сложность: О(n).

void GetIndex(int index) возвращает значение элемента по индексу. Пробегает до нужного индекса и выводит данные, если индекс больше списка просит ввести корректный индекс. Временная сложность: О(n)

void DelIndex(int index) удаляет элемент списка по индексу. Пробегает до нужного индекса и удаляет его, перед этим передав соответствующие указатели предыдущему и следующему элементу списка. Временная сложность: О(n)

void GetLength() возвращает значение поля length. Временная сложность: О(1).

void DelAll() удаляет все элементы списка. Пока поле length не будет равно 0, вызывается метод DelIndex, который принимает в себя индекс под номером 1. Временная сложность: О(n).

void ReplaceData(int index, int data) заменяет элемент по индексу. Если индекс введен корректно, то в поле data будет записано новое значение. Временная сложность: О(n).

void IfEmpty() проверка на пустоту списка. Если длина списка равна нулю, то список пустю. Временная сложность: О(n).

void Show() выводит все элементы списка. Получает указатель на голову списка и пока следующий элемент не будет равен NULL, присваиваем текущему указателю, указатель на следующий элемент. Временная сложность: О(n).

int Find(List\* sublist) ищет первое вхождение одного списка в другой. Например, если у нас есть список 1 2 3 4 и список 3 4, мы начинаем пробегаться по первому списку и пока значение первого списка не равно первому значению второго, то идем дальше по первому списку, если значения равны (3 = 3), то начинаем проверять следующие элементы двух списков ( 4 = 4) и так как во-втором списке следующее поле, после 4, будет равно NULL, следует, что первый список содержит второй. Далее выводим первое вхождение. Временная сложность: О(n).

# Пример работы

В данном примере показана реализация всех методов (Рисунок 1).

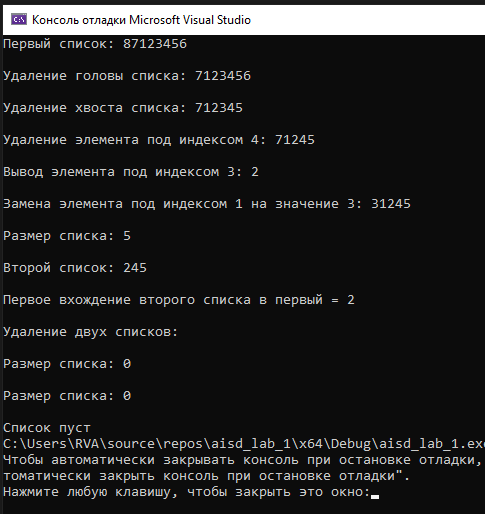


Рисунок 1

# Листинг

#include <iostream>

using namespace std;

struct Node {

Node\* next, \* prev;

int data;

};

class List {

Node\* head, \* tail;

int length = 0;

public:

List() {

head = NULL;

tail = NULL;

}

void AddHead(int data) {

Node\* temp = new Node;

temp->next = temp->prev = NULL;

temp->data = data;

if (head != NULL) {

temp->next = head;

head->prev = temp;

head = temp;

}

else {

head = temp;

}

length++;

}

void AddTail(int data) {

Node\* temp = new Node;

temp->prev = temp->next = NULL;

temp->data = data;

if (head != NULL) {

temp->prev = tail;

tail->next = temp;

tail = temp;

}

else {

temp->prev = NULL;

head = tail = temp;

}

length++;

}

void DelHead() {

if (head != NULL) {

head = head->next;

head->prev = NULL;

length--;

}

else {

cout << "список пуст";

}

}

void DelTail() {

if (tail != NULL) {

tail = tail->prev;

tail->next = NULL;

length--;

}

else {

cout << "список пуст";

}

}

void AddIndex(int index, int data) {

if (index == 0) {

cout << "input position";

cin >> index;

}

if (index < 1 || index > length + 1){

cout << "Incorrect position !!!\n";

return;

}

if (index == length + 1) {

AddTail(data);

return;

}

if (index == 1) {

AddHead(data);

return;

}

Node\* ins = head;

int i = 1;

while (i < index) {

ins = ins->next;

i++;

}

Node\* previns = ins->prev;

Node\* temp = new Node;

temp->data = data;

if (previns != 0 and length != 1) {

previns->next = temp;

}

temp->next = ins;

temp->prev = previns;

ins->prev = temp;

length++;

}

void GetIndex(int index) {

if (index == 0) {

cout << "input position";

cin >> index;

}

if (index < 1 || index > length + 1) {

cout << "Incorrect position !!!\n";

return;

}

int i = 1;

Node\* getindex = head;

while (i < index) {

getindex = getindex->next;

i++;

}

cout << getindex->data << endl;

}

void DelIndex(int index) {

if (index == 0) {

cout << "input position";

cin >> index;

}

if (index < 1 || index > length) {

cout << "Incorrect position !!!\n";

return;

}

int i = 1;

Node\* delindex = head;

while (i < index) {

delindex = delindex->next;

i++;

}

Node\* prevdelindex = delindex->prev;

Node\* nextdelindex = delindex->next;

if (prevdelindex != NULL and length != 1) {

prevdelindex->next = nextdelindex;

}

if (nextdelindex != NULL and length != 1) {

nextdelindex->prev = prevdelindex;

}

if (index == 1) {

head = nextdelindex;

}

if (index == length) {

tail = prevdelindex;

}

delete delindex;

length--;

}

void GetLength() {

cout << "Размер списка: " << length << endl << endl;

}

int Length() {

return length;

}

void DelAll() {

while (length != 0) {

DelIndex(1);

}

}

void ReplaceData(int index, int data) {

if (index == 0) {

cout << "input position";

cin >> index;

}

if (index < 1 || index > length) {

cout << "Incorrect position !!!\n";

return;

}

int i = 1;

Node\* replacedata = head;

while (i < index) {

replacedata = replacedata->next;

i++;

}

replacedata->data = data;

}

void IfEmpty() {

if (length == 0) {

cout << "Список пуст";

}

}

void Show() {

Node\* temp = head;

while (temp != NULL) {

cout << temp->data;

temp = temp->next;

}

cout << endl;

}

int Find(List\* sublist) {

if (this->Length() == 0 and sublist == NULL) {

cout << "Списки пусты!";

return 0;

}

int score = 0;

int lengthlocal = sublist->Length();

Node\* mainnode = new Node;

Node\* sublistnode = new Node;

Node\* temp = new Node;

mainnode = this->head;

sublistnode = sublist->head;

while (mainnode->next != NULL)

{

temp = mainnode;

while (temp->data == sublistnode->data)

{

temp = temp->next;

sublistnode = sublistnode->next;

if (sublistnode == NULL)

{

cout << "Первое вхождение второго списка в первый = " << score << endl;

return 1;

}

}

sublistnode = sublist->head;

mainnode = mainnode->next;

score++;

}

cout << "В строке нет подстроки";

return 0;

}

};

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

List\* mainlist = new List;

List\* sublist = new List;

cout << "Первый список: ";

mainlist->AddTail(1);

mainlist->AddTail(2);

mainlist->AddTail(3);

mainlist->AddTail(4);

mainlist->AddTail(5);

mainlist->AddTail(6);

mainlist->AddHead(7);

mainlist->AddHead(8);

mainlist->Show();

cout << endl << "Удаление головы списка: ";

mainlist->DelHead();

mainlist->Show();

cout << endl << "Удаление хвоста списка: ";

mainlist->DelTail();

mainlist->Show();

cout << endl << "Удаление элемента под индексом 4: ";

mainlist->DelIndex(4);

mainlist->Show();

cout << endl << "Вывод элемента под индексом 3: ";

mainlist->GetIndex(3);

cout << endl << "Замена элемента под индексом 1 на значение 3: ";

mainlist->ReplaceData(1, 3);

mainlist->Show();

cout << endl;

mainlist->GetLength();

cout << "Второй список: ";

sublist->AddTail(2);

sublist->AddTail(4);

sublist->AddTail(5);

sublist->Show();

cout << endl;

mainlist->Find(sublist);

cout << endl;

cout << "Удаление двух списков: " << endl << endl;

mainlist->DelAll();

mainlist->GetLength();

sublist->DelAll();

sublist->GetLength();

sublist->IfEmpty();

}