Министерство образования РФ

Пермский национальный исследовательский политехнический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 11

ПО ИНФОРМАТИКЕ ЗА I СЕМЕСТР

Вариант 5

|  |
| --- |
| Выполнил студент:  Отинов Иван Юрьевич  Группа РИС-20-1бз  Шифр 20-ЭТФ-635  Кафедра ИТАС:  Полякова Ольга Андреевна |

ПЕРМЬ 2020

# Лабораторная работа №11 "Информационные динамические структуры"

**Вариант №5**

**Цель:**

Знакомство с динамическими информационными структурами на примере одно- и двунаправленных списков.

**Задача:**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Для каждого вариант разработать следующие функции:

1. Создание списка.
2. Добавление элемента в список (в соответствии со своим вариантом).
3. Удаление элемента из списка (в соответствии со своим вариантом).
4. Печать списка.
5. Запись списка в файл.
6. Уничтожение списка.
7. Восстановление списка из файла.

**Вариант задания:**

Записи в линейном списке содержат ключевое поле типа int. Сформировать однонаправленный список. Удалить из него К элементов, начиная с заданного номера, добавить К элементов, начиная с заданного номера;

**Текст программы**

#include <iostream>

using namespace std;

// КЛАСС ОДНОНАПРАВЛЕННЫЙ СПИСОК

class UnidirectionalList {

private:

// класс узла списка

class NODE {

public:

NODE\* pNext;// указатель на следующий элемент очереди

int number;// значение элемента очереди

// конструктор узлового класса с присвоением параметров

NODE(int n, NODE\* pNext = nullptr)

{

this->number = n;

this->pNext = pNext;

}

};

// указатель на голову списка

NODE\* head;

int size; // колво элементов

public:

// конструктор по умолчанию

UnidirectionalList() { this->head = nullptr; this->size = 0; }

// деструктор класса

~UnidirectionalList() { this->clear(); }

// очищение всего списка

void clear() { while (this->size) this->pop\_front(); }

// удалиние первого элемента

void pop\_front()

{

NODE\* tmp = this->head;

this->head = this->head->pNext;

delete tmp;

this->size--;

}

// добавление элемента в конец списка

void push\_back(int number)

{

if (this->head == nullptr) this->head = new NODE(number);

else {

NODE\* current = this->head;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new NODE(number);

}

this->size++;

}

// удаление элемента по индексу

bool removeAt(int inx)

{

if (inx > this->size) return false;

if (inx == 0) this->pop\_front();

else {

NODE\* current = this->head;

for (int i = 0; i < inx - 1; i++) {

current = current->pNext;

}

NODE\* toDelete = current->pNext;

current->pNext = toDelete->pNext;

delete toDelete;

this->size--;

}

return true;

}

// получения размера списка

int getSize() { return this->size; }

// вывод информации по заданному индексу

void displayAt(const int inx)

{

int counter = 0;

NODE\* current = this->head;

while (current != nullptr)

{

if (counter == inx) {

cout << current->number << " ";

break;

}

current = current->pNext;

counter++;

}

}

// вывод всей очереди

void displayAll()

{

if (this->size == 0) { cout << "Список пуст...\n"; return; }

cout << "\nСписок элементов очереди:\n";

for (int i = 0; i < this->size; i++) this->displayAt(i);

cout << "\n";

}

// поиск информации по заданному индексу

NODE\* getElementByIndex(int index)

{

NODE\* curr = this->head;

int counter = 0;

// поиска необходимого индекса узла с головы

while (curr != nullptr)

{// пока список не пройдем весь, сравниваем номер текущего узла с индексом

//возврат найденного узла

if (counter == index) return curr;

// переход к новому узлу

curr = curr->pNext;

counter++;

}

}

// вставка элемента в список после элемента с индексом index

bool insert(int value, int index)

{

NODE\* curr = this->head;

int counter = 0;

// поиска необходимого индекса узла с головы

while (curr != nullptr)

{// пока список не пройдем весь, сравниваем номер текущего узла с индексом

// вставка значения после элемента с индексом index

if (counter + 1 == index) {

NODE\* obj = new NODE(value);

obj->pNext = curr->pNext;

curr->pNext = obj;

this->size++;

// данные успешно добавлены

return true;

}

// переход к новому узлу

curr = curr->pNext;

counter++;

}// while

// не удалось добавить данные

return false;

}

// создание списка (его наполнение с клавиатуры)

void createList()

{

int nInputVal;

cout << "[ЗАПОЛНЕНИЕ СПИСКА]\n" <<

"Вводите значения последовательно, нажимая Enter.\n" <<

"Чтобы прервать добавление элементов, введите ноль.\n";

this->clear();

while (true) {

cout << "Ввод числа: ";

cin >> nInputVal;

if (nInputVal == 0) break;

this->push\_back(nInputVal);

}

}

// добавить К элементов, начиная с заданного номера

void addFromPos(int K, int pos)

{

for (int i = pos; i < pos + K; i++)

insert(i + 1, i);

}

// удалить из списка, x2 элементов, начиная с заданного номера x1

bool removeRange(int x1, int x2)

{

//проверка на корректный интервал

if (x1 < 0 || x1 + x2 >= this->getSize()) return false;

// удаление элементов

int temp = 0;

for (int i = x1; i < x1 + x2; i++) {

this->removeAt(i + temp);

temp--;

}

return true;

}

// запись списка в файл

bool saveToFile(const char fileName[50])

{

if (this->size == 0) return false;

FILE\* file;

// открытие файла для записи

fopen\_s(&file, fileName, "w");

// запись данных в файл

int value = 0;

for (int i = 0; i < this->size; i++) {

// запись в файл

value = getElementByIndex(i)->number;

fprintf(file, "%d ", value);

}

// закрытие потока файла

fclose(file);

return true;

}

// чтение файла в список

bool readFile(const char fileName[50])

{

FILE\* file;

// открытие файла в режиме чтения

fopen\_s(&file, fileName, "r");

if (file == NULL) return false;

//чтение строк файла

int value = 0;

while (!feof(file)){

fscanf\_s(file, "%d ", &value);

// добавление элемента в список

push\_back(value);

}

fclose(file);

return true;

}

};

int main()

{

setlocale(LC\_ALL, "RU");

// создание объекта класса однонаправленного списка

UnidirectionalList list;

list.createList();

list.displayAll();

// Удалить из списка, К элементов, начиная с заданного номера

int countToRemove = 0;

int pos = 0;

cout << "Удалить из исписка элемены в количестве K = ";

cin >> countToRemove;

cout << "Номер, с которого нужно начать удаление N = ";

cin >> pos;

bool bSuccess = list.removeRange(pos, countToRemove);

if (bSuccess) {

cout << "Список после удаления..\n";

list.displayAll();

}

// добавить К элементов, начиная с заданного номера

pos = 0;

int countAdd = 0;

cout << "Добавить элементов в количестве K = ";

cin >> countAdd;

cout << "Номер, с которого нужно начать добавление N = ";

cin >> pos;

list.addFromPos(countAdd, pos);

cout << "Список после добавления..";

list.displayAll();

// сохранение списка в файл

bSuccess = list.saveToFile("output.txt");

if (bSuccess) {

cout << "Список сохранен в файл..\n";

}

cout << "Список уничтожен..\n";

list.clear();

list.displayAll();

// запись данных из файла в список

cout << "Чтение из файла..\n";

bSuccess = list.readFile("output.txt");

if (bSuccess) {

list.displayAll();

}else cout << "Не удалсь считать файл \n";

cout << endl;

system("pause");

}

**Результаты работы программы:**

