МИНОБРНАУКИ РОССИИ

Федеральное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

(ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова»)

Факультет: «Программное обеспечение»

Кафедра: «Программное обеспечение»

Лабораторная работа №2

Выполнил:

Студент группы Б19-191-1з

Бирюков Филипп Денисович

Проверил:

Старший преподаватель

Еланцев М.О.

Ижевск, 2021

# Оглавление

[Оглавление 2](#_Toc63281208)

[1. ЗАДАНИЕ 3](#_Toc63281209)

[Реализовать структуру данных двунаправленный список: 3](#_Toc63281210)

[Реализовать структуру данных очередь на базе массива: 3](#_Toc63281211)

[2. КОД ПРОГРАММЫ 4](#_Toc63281212)

[LinkedList.h: 4](#_Toc63281213)

[Queue.h: 7](#_Toc63281214)

[Main.cpp: 9](#_Toc63281215)

[3. ПРИМЕР РАБОТЫ 11](#_Toc63281216)

# ЗАДАНИЕ

## Реализовать структуру данных двунаправленный список:

Список должен содержать следующие операции:

* Вставка элемента в конец (целочисленное значение)  
  Вставка должна корректно обрабатывать случай, когда список пуст (указатель на первый узел пуст).
* Вставка элемента перед заданным индексом
* Получение значения элемента по индексу
* Удаление элемента
* Печать всех элементов списка
* Освобождение памяти от структуры данных

## Реализовать структуру данных очередь на базе массива:

Структура данных должна содержать следующие операции:

* Инициализация (с указанием размер буфера под стек или очередь)
* Вставка элемента (целочисленное значение, вставка в конец очереди)
* Извлечение элемента

Операция должна возвращать код ошибки, в случае если в структуре данных нет элементов (например, функция возвращает false в случае, если нет элементов, и true - если элемент был извлечен, сам элемент в таком случае возвращается через параметр по ссылке)

* Клонирование (создание копии структуры данных со всеми заполненным на текущий момент элементами)
* Освобождение памяти от структуры данных

# КОД ПРОГРАММЫ

## LinkedList.h:

#pragma *once*

#ifndef null

#define null nullptr

#endif

#include <iostream>

using *namespace* std;

*struct* Node {

    Node\* prev = nullptr;

    Node\* next = nullptr;

*int* value;

    Node(*int* *value*) {

*this*->value = *value*;

    }

    Node(*int* *value*, Node*\** *next*, Node*\** *prev*) {

*this*->value = *value*;

*this*->next = *next*;

*this*->prev = *prev*;

    }

};

*struct* LinkedList {

    Node\* head = nullptr;

    Node\* tail = nullptr;

*void* add(*int* *value*) {

        if (!head) {

            head = new Node(*value*);

            tail = head;

            return;

        } else {

            Node\* oldTail = tail;

            tail = new Node(*value*, nullptr, oldTail);

            oldTail->next = tail;

            return;

        }

    }

*bool* add(*int* *value*, *int* *index*) {

        if (*index* < 0) return false;

        Node\* curr = head;

        if (curr == null && *index*!=0) return false;

        if (*index* == 0) {

            if (!head) {

                add(*value*);

                return true;

            }

            head = new Node(*value*);

            head->next = curr;

            curr->prev = head;

            return true;

        } else {

            for (*int* i = 1; i<*index*; i++) {

                curr = curr->next;

                if (curr == null) {

                    return false;

                }

            }

            if (curr->next == null) {

                tail = new Node(*value*, null, curr);

                curr->next = tail;

                return true;

            } else {

                curr = curr->next;

                Node\* next = new Node(curr->value, curr->next, curr);

                curr->value = *value*;

                curr->next = next;

                if (tail->next != null) {

                    tail = tail->next;

                }

                return true;

            }

        }

    }

    Node*\** get(*int* *index*) {

        if (head == null) return null;

        Node\* curr = head;

        for (*int* i = 1; i<=*index*; i++) {

            if (curr == null) return null;

            curr = curr->next;

        }

        return curr;

    }

*bool* remove(*int* *index*) {

        Node\* node = get(*index*);

        return removePtr(node);

    }

*bool* removePtr(Node*\** *node*) {

        if (*node* == null) return false;

        if (head == tail) {

            head = null;

            tail = null;

            delete *node*;

            return true;

        }

        if (*node* == head) {

            head = head->next;

            head->prev = null;

*node*->next = null;

*node*->prev = null;

            delete *node*;

            return true;

        }

        if (*node* == tail) {

            tail = tail->prev;

            tail->next = null;

*node*->next = null;

*node*->prev = null;

            delete *node*;

            return true;

        }

        Node\* prev = *node*->prev;

        Node\* next = *node*->next;

        prev->next = next;

        next->prev = prev;

*node*->next = null;

*node*->prev = null;

        delete *node*;

        return true;

    }

*bool* removeByValue(*int* *value*) {

*int* i = 0;

        Node\* node = head;

        while (node != null) {

            if (node->value == *value*) {

                cout << "Node:" << node << " Value:" << node->value << " index:" << i << endl;

                return removePtr(node);

            }

            node = node->next;

            i++;

        }

        return false;

    }

*void* print() {

        cout << "Tail: " << tail << endl;

        cout << "Head: " << head << endl;

        Node\* curr = head;

        while (curr != null) {

            cout << curr->prev << "<<[" << curr->value << "]>>" << curr->next <<endl;

            curr = curr->next;

        }

        cout << "end" << endl;

    }

*void* freeMem() {

        Node\* curr = head;

        while (curr) {

            Node\* toRemove = curr;

            curr = curr->next;

            delete toRemove;

        }

    }

};

## Queue.h:

#pragma *once*

#ifndef null

#define null nullptr

#endif

#include <iostream>

using *namespace* std;

*struct* Data {

*int* value;

*int* resultCode = 200;

    Data(*int* *value*, *int* *code*) {

*this*->value = *value*;

*this*->resultCode = resultCode;

    }

};

*struct* Queue {

*int* capacity = 0;

*bool* isInitialized = false;

*int*\* array = null;

*int* first = 0;

*int* last = 0;

*int* size = 0;

*void* init(*int* *capacity*) {

        if (isInitialized) return;

*this*->capacity = *capacity*;

*this*->isInitialized = true;

*this*->array = new *int*[*capacity*];

    }

*void* enqueue(*int* *value*) {

        if (size == capacity) {

*int*\* oldArray = array;

*int* oldCapacity = capacity;

            capacity = capacity \* 2;

            array = new *int*[capacity];

*int* tempSize = size;

*int* tempFirst = first;

            first = 0;

            last = 0;

            size = 0;

            while (tempSize > 0) {

                enqueue(oldArray[tempFirst++]);

                tempFirst = tempFirst % oldCapacity;

                tempSize--;

            }

        }

        array[last++] = *value*;

        last = last % capacity;

        size++;

    }

    Data*\** dequeue() {

        if (size == 0) {

            return new Data(-1, 404);

        }

        Data\* data = new Data(array[first++], 200);

        first = first%capacity;

        size--;

        return data;

    }

    Queue*\** clone() {

        Queue\* queue = new Queue();

        queue->capacity = *this*->capacity;

        queue->first = *this*->first;

        queue->size = *this*->size;

        queue->last = *this*->last;

        queue->isInitialized = true;

        queue->array = new *int*[capacity];

        for (*int* i = 0; i<capacity; i++) {

            queue->array[i] = *this*->array[i];

        }

        return queue;

    }

*void* freeMem() {

        delete[] *this*->array;

        isInitialized = false;

        capacity = 0;

        size = 0;

        first = 0;

        last = 0;

        array = null;

    }

*void* print() {

*int* temp = size;

*int* tempFirst = first;

        cout << "Head-> ";

        while (temp > 0) {

            cout << array[tempFirst++] << "[" << tempFirst << "]" << " ";

            tempFirst = tempFirst%capacity;

            temp--;

        }

        cout << "<-Tail" << endl;

    }

};

## Main.cpp:

#include <iostream>

#include <string>

#include "LinkedList.h"

#include "Queue.h"

*int* main() {

    LinkedList\* list = new LinkedList();

    for (*int* i = 0; i<5; i++) {

        list->add(i);

    }

    list->print();

    list->add(-1,0);

    list->print();

    list->remove(0);

    list->print();

    list->remove(4);

    list->print();

    for (*int* i = 0; i<5; i++) {

        list->add(i+5);

    }

    list->remove(1);

    list->remove(7);

    list->print();

    list->add(7, 6);

    list->print();

    list->removeByValue(116);

    list->removeByValue(7);

    list->print();

    list->removeByValue(8);

    list->print();

    Queue\* que = new Queue();

    que->init(3);

    que->enqueue(1);

    que->dequeue();

    que->enqueue(1);

    que->dequeue();

    que->enqueue(1);

    que->enqueue(2);

    que->enqueue(1);

    for (*int* i = 0; i < 5; i++) {

        que->enqueue(i+5);

    }

    que->print();

    return 0;

}

# ПРИМЕР РАБОТЫ

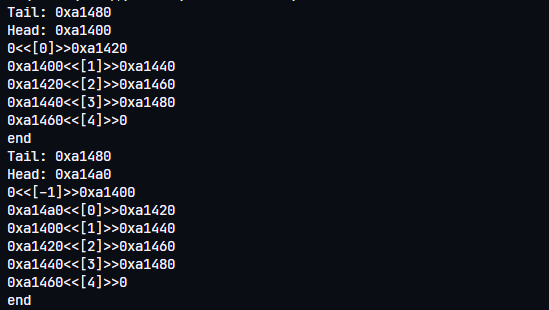
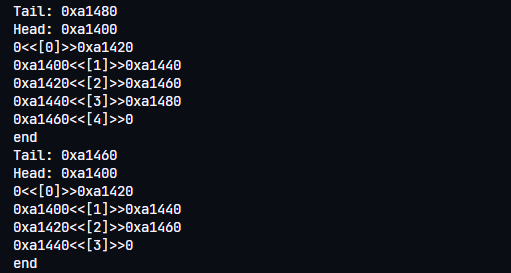
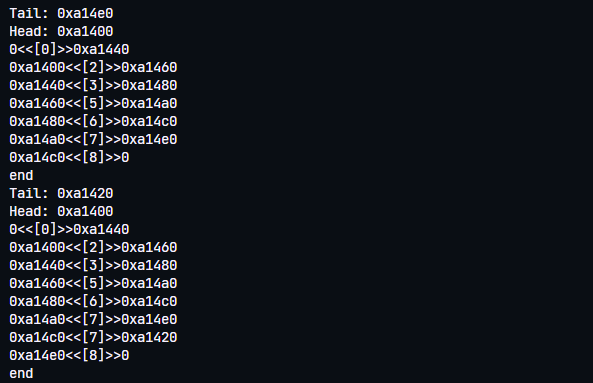


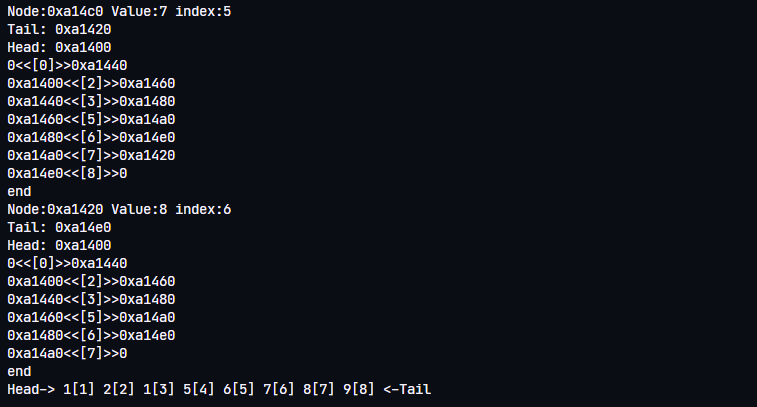
Рисунок 1



Рисунок



Рисунок



Рисунок