Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

«СТЕК, ДЕк, очередь»

Отчет по практиктической работе №2

по дисциплине «Методы программирования»

Выполнил:

Студент гр. 745

\_\_\_\_\_\_\_\_Загреба М.А.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2017 г.

Проверил:

Доцент кафедры БИС

\_\_\_\_\_\_\_\_\_Романов А.С.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2017 г.

Томск 2017

1 Введение

Цель работы: научиться работать со структурами данных – стек, очередь, дек, реализованными с помощью массивов и связных списков.

Задание

1. Общее задание.

Создать три класса, соответственно для: стека, очереди, дека.

Желательно при этом, чтобы структуры данных были ДИНАМИЧЕСКИМИ, то есть ограниченны только объемом свободной оперативной памяти. Для этого используйте указатели и динамически распределяемую память.

1.1 Очередь

Разработайте программу, реализующую одностороннюю очередь с использованием шаблонного класса. Программа содержит определение шаблонного класса для работы с односторонней очередью.

1.2 Стек

Разработайте программу, реализующую динамический стек неограниченного размера с использованием шаблонного класса. Программа содержит определение структуры для элемента стека и определение шаблонного класса для работы с динамическим стеком неограниченного размера.

1.3 Дек (универсальная двухсторонняя очередь)

Разработайте программу, реализующую дек с использованием шаблонного класса. Программа содержит определение шаблонного класса для работы с универсальной очередью.

1. Индивидуальное задание

Дан файл из вещественных чисел. Используя очередь, за один просмотр файла напечатать сначала все числа, меньшие a, затем все числа из интервала [a,b], и, наконец, все остальные числа, сохраняя исходный порядок в каждой группе.

2 Ход работы

2.1 Стек

Стек (stack) это линейная структура данных переменного типа. Позволяет включать и исключать элементы только c одного конца, называемого вершина (top) стека. В этом случае первым будет выбран элемент, который был записан в стек последним. Информация в этой структуре данных обрабатывается по принципу: «последним пришел, первым ушел». Как говорят, поддерживает дисциплину LIFO (Last In, First Out). Структура данных стек является структурой с ограниченным доступом, так как доступ разрешается только к элементу, находящемуся в вершине стека.

Стек через динамический массив

#include <iostream>

#include <conio.h>

#include <fstream>

#include "stack1.h"

using namespace std;

using Program::stack;

int main(int argc, char\*\* argv)

{

setlocale(LC\_ALL, "Russian");

stack mystack(5);

mystack.push(10);

mystack.push(30);

mystack.push(50);

cout << mystack.pop() << endl;

cout << mystack.pop() << endl;

cout << mystack.pop() << endl;

system("pause");

return 0;

}

Заголовочный файл

#pragma once

#include <iostream>

namespace Program {

class stack

{

private:

int\* arr;

int max;

int index;

public:

stack(int number)

{

max = 1000;

index = 0;

if (number <= max)

{

arr = new int[number];

max = number;

}

else std::cout << "Переполнение стека";

}

~stack()

{

delete[] arr;

};

bool push(int x)

{

if (index == max)

return false;

else

{

arr[index] = x;

index++;

return true;

}

}

int pop()

{

if (index < 0)

return 0;

else

{

return arr[--index];

}

}

bool empty()

{

if (index < 0)

return true;

else return false;

}

bool full()

{

if (index == max)

return true;

else return false;

}

};

}

Стек через список

#include <iostream>

#include <conio.h>

#include "stack2.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

stack<int>stack;

stack.push(50);

stack.push(30);

stack.push(10);

stack.print();

stack.pop();

stack.print();

stack.pop();

stack.print();

stack.pop();

stack.print();

system("pause");

return 0;

}

Заголовочный файл

#pragma once

#include <iostream>

#include <string>

using namespace std;

template<class T>

class stack {

private:

struct Index // структура для указание элементов (список)

{

Index() :element(0), next(0) {}

Index\* next;

T element;

};

Index\* top;

Index\* last;

public:

stack() :last(0), top(0) {}

~stack() {

while (top)

{

last = top->next;

delete top;

top = last;

}

}

void print() {

if (last == NULL)

{

cout << " Пустой стек ";

}

else

{

for (Index\* x = last; x != 0; x = x->next)

{

cout << x->element<< ' ';

}

cout << '\n';

}

}

void push(T val) {

Index\* Temp;

Temp = new Index;

Temp->element = val;

if (last == 0)

{

last = Temp;

}

else

{

Temp->next = last;

last = Temp;

}

}

T pop() {

if (last == 0)

{

cout << " Пустой стек ";

}

else

{

T tmp = last->element;

Index\* delx = last;

last = last->next;

delete delx;

cout << "Удаление вершины стека: \n";

return tmp;

}

}

};

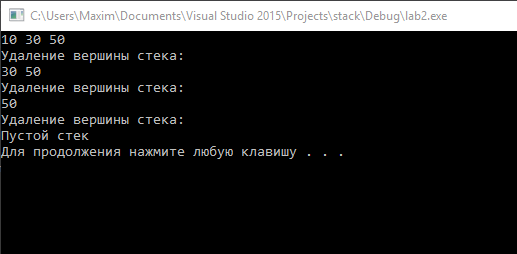


Рисунок 2.1 – Пример работы стека

2.2 Очередь

Очередь – это линейная структура переменного размера. Исключение из очереди допускается с одного конца – с начала очереди. Включение элементов возможно лишь с противоположного конца. Данные в такой структуре обрабатываются в порядке их поступления по принципу: «первым пришел, первым ушел». Очередь - это структура типа FIFO (First In, First Out). Данная структура часто применяется в технике, при считывании информации с измерительных приборов. Запись в такой буфер производится по прерыванию, а считывание – когда есть время для обработки поступившей информации.

#include "queue.h"

#include <fstream>

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

queue<float>que;

que.push(1);

que.push(2);

que.push(3);

que.push(4);

que.print();

que.pop();

que.print();

que.pop();

que.print();

que.pop();

que.print();

que.pop();

que.print();

system("pause");

return 0;

}

Заголовочный файл

#pragma once

#include <iostream>

#include <string>

using namespace std;

template<class T>

class queue

{

private:

struct Index

{

Index() :next(0), element(0) {}

Index\* next;

T element;

};

Index\* top;

Index\* last;

public:

queue() :top(0), last(NULL) {}

~queue()

{

while (top)

{

last = top->next;

delete top;

top = last;

}

}

void push(T val)

{

Index\* Temp = new Index;

Temp->element = val;

if (top == 0)

{

top = Temp;

last = Temp;

return;

}

last->next = Temp;

last = Temp;

}

T& pop()

{

if (top == 0)

{

cout << " Пустая очередь ";

}

else

{

T tmp = top->element;

Index\* delx = top;

top = top->next;

delete delx;

cout << "Очередь состоит из: \n";

return tmp;

}

}

void print()

{

if (top == 0)

{

cout << " Пустая очередь ";

}

else

{

for (Index\* x = top; x != 0; x = x->next)

cout << x->element << ' ';

cout << '\n';

}

}

};

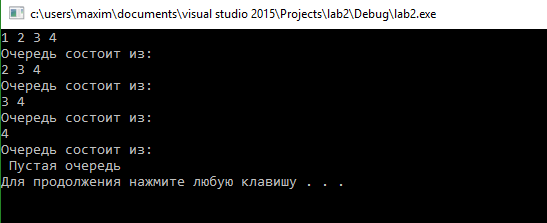


Рисунок 2.2 – Пример работы очереди

2.3 Дек

Дек – линейная динамическая структура данных, представляет собой очередь с двумя концами. Частными случаями является дек с ограниченным входом и дек с ограниченным выходом. Дек можно обрабатывать как очередь или как стек.

#include <fstream>

#include <iostream>

#include "dek.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

Dek<char> dek;

dek.push\_left('B');

dek.push\_left('A');

dek.push\_right('C');

dek.push\_right('D');

cout << "Дек с левой стороны \n";

dek.print\_left();

cout << "Дек с правой стороны \n";

dek.print\_right();

dek.pop\_left();

dek.print\_left();

dek.pop\_right();

dek.print\_right();

system("pause");

return 0;

}

Заголовочный файл

#pragma once

#include <iostream>

#include <string>

using namespace std;

template<class T>

class Dek

{

private:

struct Index

{

Index() : element(0), next(0), pred(0)

{

}

Index\* next;

Index\* pred;

T element;

};

Index\* top;

Index\* last;

public:

Dek() : top(0), last(0){}

~Dek()

{

while (top)

{

Index\* tmpIndex;

tmpIndex = top->next;

delete top;

top = tmpIndex;

}

}

void push\_left(T val)

{

Index\* Temp = new Index;

Index\* current = top;

Temp->element = val;

if (top == 0)

{

top = Temp;

top->next = 0;

last = top;

return;

}

Temp->next = top;

top = Temp;

Temp->next->pred = top;

}

void push\_right(T val)

{

Index\* Temp = new Index;

Index\* current = last;

Temp->element = val;

if (top == 0)

{

top = Temp;

top->next = 0;

top->pred = 0;

last = top;

return;

}

last->next = Temp;

last = Temp;

last->pred = current;

}

T pop\_left()

{

if (last == 0)

{

cout << " Пусто \n";

}

else

{

T TempElement = top->element;

top = top->next;

Index\* delx = top->pred;

delete delx;

cout << "Удаление с левой стороны: \n";

top->pred = 0;

return TempElement;

}

}

T pop\_right()

{

if (last == 0)

{

cout << " Пусто \n";

}

else

{

T TempElement = last->element;

last = last->pred;

Index\* delx = last->next;

delete delx;

cout << "Удаление с правой стороны: \n";

last->next = 0;

return TempElement;

}

}

void print\_left()

{

if (top == 0)

{

cout << " Пусто \n";

}

else

{

for (Index\* x = top; x != 0; x = x->next) {

cout << x->element << ' ';

}

cout << '\n';

}

}

void print\_right()

{

if (top == 0)

{

cout << " Пусто \n";

}

else

{

for (Index\* x = last; x != 0; x = x->pred) {

cout << x->element << ' ';

}

cout << '\n';

}

}

};

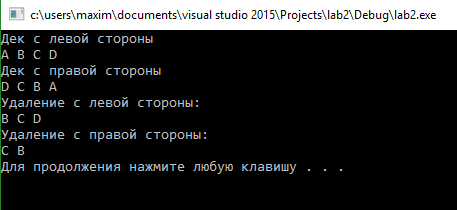


Рисунок 2.3 – Пример работы дека

2.4 Индивидуальное задание

#include <iostream>

#include <fstream>

#include "queue.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

ifstream fin;

queue <double> que;

double num, a, b;

int count = 0;

cout << "Введите границы интервала [a, b] \n";

cin >> a >> b;

fin.open("indiv.txt");

cout << "\nИсходные числа: \n";

while (fin >> num) {

cout << num << ' ';

}

cout << "\n";

fin.close();

fin.open("indiv.txt");

cout << "\nЧисла меньше " << a << ": \n";

while (fin >> num)

{

if (num < a)

{

cout << num << " ";

}

else

{

que.push(num);

++count;

}

}

cout << endl;

cout << "\nСостояние очереди: \n";

que.print();

cout << "\nЧисла из интервала [" << a << ", " << b << "]:\n";

while (count--)

{

double tmp = que.front();

que.pop();

if (tmp > b) {

que.push(tmp);

}

else cout << tmp << " ";

}

cout << endl;

cout << "\nСостояние очереди: \n";

que.print();

cout << "\nЧисла больше " << b << ":\n";

while (!que.empty())

{

cout << que.front() << " ";

que.pop();

}

cout << endl;

cout << "\nСостояние очереди: \n";

que.print();

cout << endl;

system("pause");

return 0;

}

Заголовочный файл

#pragma once

#include <iostream>

#include <string>

using namespace std;

template<class T>

class queue

{

struct Index

{

Index() :next(0), element(0) {}

Index\* next;

T element;

};

private:

Index\* top;

Index\* last;

public:

queue() :top(0), last(0) {}

~queue()

{

while (top)

{

last = top->next;

delete top;

top = last;

}

}

bool empty() const

{

return top == 0;

}

const T& front() const

{

return top->element;

}

void push(T val)

{

Index\* Temp = new Index;

Temp->element = val;

if (top == 0)

{

top = Temp;

last = Temp;

return;

}

last->next = Temp;

last = Temp;

}

T& pop()

{

if (top == 0)

{

cout << " Пустая очередь \n";

}

else

{

T tmp = top->element;

Index\* delx = top;

top = top->next;

delete delx;

return tmp;

}

}

void print()

{

if (top == 0)

{

cout << " Пустая очередь \n";

}

else

{

for (Index\* x = top; x != 0; x = x->next)

cout << x->element << ' ';

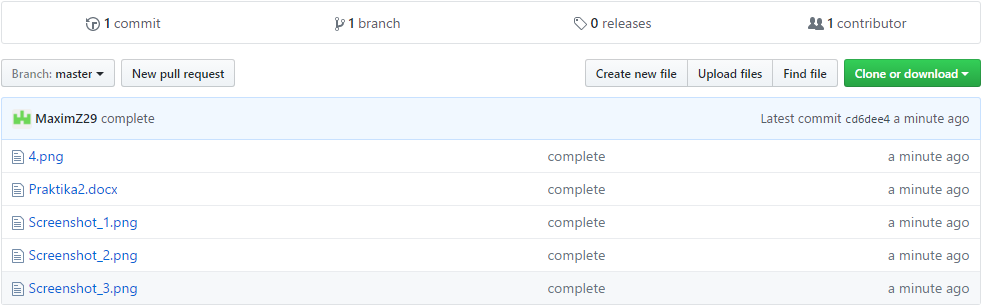
cout << '\n';

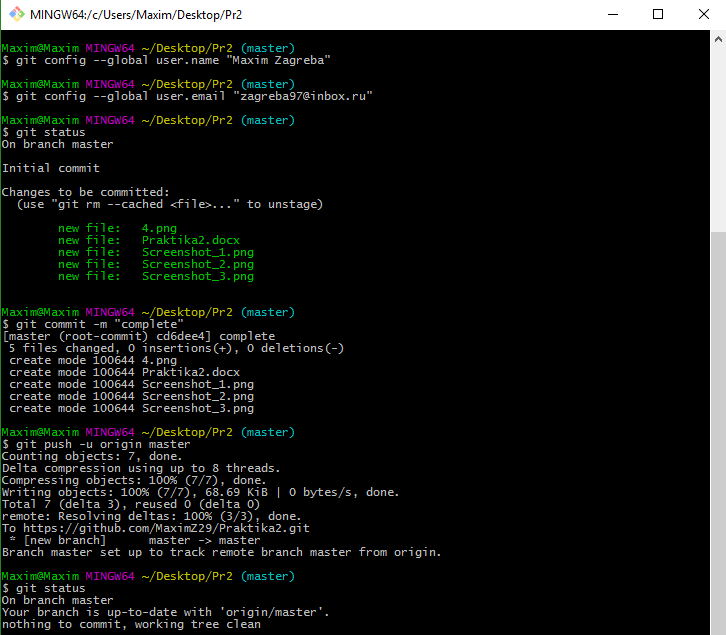
}

}

};

2.5 Работа с GitHub





3 Заключение

В результате практической работы были получены навыки работы со структурами данных – стек, очередь, дек, реализованными с помощью массивов и связных списков.