Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №8**

**По дисциплине «Основы алгоритмизации и программирования»**

**Тема: «Динамические структуры данных»**

Выполнил: Шиковец Е.А.

Группа 21-ИТ-1

Проверила: Пантелейко А.Ф

Преподаватель-стажер кафедры ТП

Полоцк 2022

**Цель работы:** изучить методы организации списочных структур в динамической памяти. Реализовать алгоритмы помещения и изъятия элементов из стека, дека или очереди.

**Вариант № 4**

Создайте указанную динамическую структуру данных (ДСД) на основе списка для хранения данных, указанных в варианте задания. Напишите следующие функции:

1. Функции для создания новой ДСД и ее удаления.

2. Функции добавления элементов (все, что необходимы для полноценной работы).

3. Функции удаления элементов (все, что необходимы для полноценной работы).

4. Функция печати всех элементов ДСД.

5. Функция очистки ДСД (то есть удаления всех вложенных в нее элементов).

#include <iostream>

#include <stdlib.h>

#include <cstdlib>

using namespace std;

class Dequeue

{ //дек Dequeue

int data, count\_;

Dequeue \*start, \*end, \*next, \*prev;

public:

Dequeue() : count\_(0), start(NULL), end(NULL){}; //Инициализация с помощью конструктора по умолчанию

void push\_front(double); // добавление в начало

void push\_back(double); // добавление в конец

void size(); // размер дека

void pop\_front(); // удаление первого

void pop\_back(); // удаление последнего

void front(); // печать первого

void back(); // печать последнего

~Dequeue();

};

void Dequeue::push\_front(double x)

{

Dequeue \*temp = new Dequeue;

temp->data = x;

temp->next = NULL;

count\_++;

if (!start)

{

temp->prev = NULL;

start = temp;

end = start;

}

else

{

temp->next = start; //Указываем, что следующим элементом списка относительно добавленного, будет первый элемент существующего списка

start->prev = temp; //Предыдущий за последним существующим это непосредственно сейчас добавляемый элемент списка

start = temp; //После того как указали что есть настоящий и что следующий, объявляем, что первый существующий это только что добавленный элемент

}

cout << x << " was pushed at front\n";

}

void Dequeue::push\_back(double x)

{

Dequeue \*temp = new Dequeue;

temp->data = x;

temp->next = NULL;

count\_++;

if (!start)

{

temp->prev = NULL;

start = temp;

end = start;

}

else

{

temp->prev = end; //Указываем, что предыдущим элементом списка относительно добавленного, будет последний элемент существующего списка

end->next = temp; //Следующий за последним существующим это непосредственно сейчас добавляемый элемент списка

end = temp; //После того как указали что есть настоящий и что предыдущий, объявляем, что последний существующий это только что добавленный элемент

}

cout << x << " was pushed at back\n";

}

void Dequeue::size()

{

cout << "The size of dequeue is " << count\_ << endl;

}

void Dequeue::pop\_front()

{

if (!count\_)

{

cout << "Deck is empty\n";

return;

}

if (start->next)

{ //Если удаляем первый, но есть и другие, то

Dequeue \*temp = start;

cout << temp->data << " was deleted\n";

start = start->next;

start->prev = NULL;

delete temp;

count\_--;

return;

}

else if (start == end)

{ //Если удаляем первый, но в деке только 1 элемент

cout << start->data << " was deleted\n";

start->next = NULL;

start = NULL;

delete start; //Удаляем указатель на начало

count\_ = 0;

return;

}

}

void Dequeue::pop\_back()

{

if (!count\_)

{

cout << "Deck is empty\n";

return;

}

if (start != end)

{

Dequeue \*temp = end;

cout << temp->data << " was deleted\n";

end = end->prev; //Отодвигаем хвост назад

end->next = NULL;

delete temp; //Очищаем память от бывшего хвоста

count\_--;

return;

}

else

{

cout << start->data << " was deleted\n";

start->next = NULL;

start = NULL;

delete start; //Удаляем указатель на начало

count\_ = 0;

return;

}

}

void Dequeue::front()

{

if (count\_)

cout << "Front is " << start->data << endl;

else

cout << "Deck is empty\n";

}

void Dequeue::back()

{

if (count\_)

cout << "Back is " << end->data << endl;

else

cout << "Deck is empty\n";

}

Dequeue::~Dequeue()

{

while (start)

{

end = start->next;

delete start;

start = end;

}

}

int main()

{

system("Color 09");

system("CLS");

Dequeue D;

int n = -1;

double value;

while (n != 0)

{

cout << "Menu\n";

cout << "1. Push front\n";

cout << "2. Push back\n";

cout << "3. Size\n";

cout << "4. Pop front\n";

cout << "5. Pop back\n";

cout << "6. Print front\n";

cout << "7. Print back\n";

cout << "0. Exit\n";

cin >> n;

switch (n)

{

case 1: // push front

system("CLS");

cout << "Value: ";

cin >> value;

system("CLS");

D.push\_front(value);

break;

case 2: // push back

system("CLS");

cout << "Value: ";

cin >> value;

system("CLS");

D.push\_back(value);

break;

case 3: // size

system("CLS");

D.size();

break;

case 4: // pop front

system("CLS");

D.pop\_front();

break;

case 5: // pop back

system("CLS");

D.pop\_back();

break;

case 6: // print front

system("CLS");

D.front();

break;

case 7: // print back

system("CLS");

D.back();

break;

case 0:

system("CLS");

system("Color 07");

D.~Dequeue();

return 0;

break;

default:

system("CLS");

cout << "Check your input\n";

break;

}

}

return 0;

}

**Вывод:**

Изучены принципы работы с ДСД и функции для работы с ней. Разработана программа с использованием динамических структур.