Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ

УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра комплексной информационной безопасности

электронно-вычислительных систем (КИБЭВС)

СТЕК, ОЧЕРЕДЬ, ДЕК

Отчет по лабораторной работе 2

по дисциплине "Методы программирования"

Выполнила

Студент гр. 735

\_\_\_\_\_\_\_\_С.О.Сарыглар

20.04.2017

Проверил

Доцент кафедры КИБЭВС

канд. техн. наук

\_\_\_\_А.С. Романов

20.04.2017

Томск 2017

1 Введение

Цель работы :

Научиться работать со структурами данных – стек ,очередь, дек, реализованными с помощью массивов и связных списков.

Задания:

Язык программирования – любой.

Работа состоит из общей части и индивидуального задания:

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. Очередь

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

В шаблоне классов реализованы операции с очередью:

-инициализация очереди (конструктор);

- разрушение очереди с освобождением занятой динамической памяти (деструктор);

- занесение элемента в очередь;

- извлечение элемента из очереди;

- печать состояния очереди;

* 1. 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.

В шаблонном классе реализованы следующие операции со стеком:

-инициализация стека (конструктор);

- разрушение стека с освобождением занятой динамической памяти (деструктор);

- занесение элемента в стек;

- извлечение элемента из стека;

- печать состояния стека.

* 1. Дек (универсальная двухсторонняя очередь)

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

В шаблоне классов реализованы операции с очередью:

-инициализация очереди (конструктор);

- разрушение очереди с освобождением занятой динамической памяти (деструктор);

- занесение элемента с левого конца;

- занесение элемента с правого конца;

- извлечение элемента с левого конца;

- извлечение элемента с правого конца;

- печать состояния очереди с использованием указателя на левый конец очереди;

- печать состояния очереди с использованием указателя на правый конец очереди.

1. Индивидуальное задание

2.1 Используя разработанные классы решить задачу, соответствующую варианту.

Вариант 23

Последовательность символов, ограниченную точкой, занести в два стека, содержащих гласные и согласные буквы. Вывести текст и элементы из обоих стеков.

2.2 Залить все программы в новый проект на свой аккаунт github.

2.3 Написать отчет и защитить у преподавателя.

2 Ход работы

2.1 Общие сведения

**Стек.**

Структура данных типа стек применяется очень часто, например, при распознавании синтаксиса в компиляторах, для передачи параметров функциям, при выполнении вызова функции и возвращения из нее.

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) .

**Для работы со стеком необходим дескриптор содержащий:**

– имя стека;

– адрес верхней границы стека;

– указатель на свободный;

– адрес нижней границы;

– описание элемента (длина записи).

Когда стек пуст, указатель указывает на нижнюю границу стека.

**Основные операции над стеком:**

– создать стек;

– включить запись в стек;

– исключить запись из стека;

– очистить стек;

– определить объем стека (количество записей).

**1. Создание стека.** При создании последовательного представления необходимо знать предельный размер стека. Под предполагаемый предельный размер резервируется блок памяти. Предельный размер рассчитывается исходя из знания о длине записи *R* и максимально возможного количества записей в стеке *n* .

**2. Включение записи в стек.** При включении записи в стек необходимо проверить: есть ли место для записи (указатель стека указывает на верхнюю границу стека ). Если нет, то выдать сообщение «переполнение стека», если место есть, то указатель стека переместить вверх на одну запись и записать данные.

**3. Исключить запись из стека.** Проверить: есть ли в стеке информация (указатель стека равен нижней границе?). Если нет, то выйти. Можно выдать сообщение: «Стек пуст». Если есть, то считать данные и переместить указатель на одну запись вниз.

**4. Очистить стек.** В указатель стека помещается значение нижней границы стека.

**5. Проверка объема стека** сводится к определению количества записей в стеке.

Стековые структуры широко применяются в трансляторах, при реализации всевозможных подпрограмм и многоуровневых прерываний, а также при решении задач, алгоритмы которых лучше всего описываются рекурсивными методами.

**Очередь (буфер)**

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).

**Операции со структурой данных** типа кольцевая очередь (кольцевой буфер):

– создать структуру типа буфер;

– очистить буфер (сделать пустым);

– найти первый элемент буфера;

– вставить элемент в конец буфера;

– удалить первый элемент из буфера;

– определить, пуст ли буфер.

Для работы с буфером **создается дескриптор включающий:**

– имя очереди (буфера);

– адрес начала буфера;

– адрес конца буфера;

– указатель для записи Put;

– указатель для считывания Get;

– счетчик записей ( начальное значение которого равно нулю);

– максимальное количество записей в очереди (буфере).

**Алгоритм для записи информации в буфер.** В начале необходимо проверить, полон ли буфер (счетчик записей в этом случае равен максимально возможному числу записей). Если буфер полон, необходимо выдать сообщение «переполнение буфера». Если в буфере есть место для записи, необходимо проверить значение указателя Put. Если он указывает на конец буфера, то ему необходимо присвоить значение адреса начала буфера. Далее, для любой ситуации записать данные в буфер, переместить значение указателя на следующий свободный для записи адрес, увеличить значение счетчика записей на единицу.

**Алгоритм для считывания**. На первом шаге алгоритма проверяется значение счетчика записей. Если он пуст, то в буфере нет ни одной записи. Поэтому необходимо выдать сообщение, что буфер пуст, и выйти из процедуры считывания информации из буфера. Если в буфере есть информации, необходимо проверить на какой адрес указывает указатель Get . Если указывает на конец буфера, то в этом случае указателю Get необходимо присвоить значение начала буфера. Далее, в любом случае считываются данные и переносятся в другой буфер для последующей обработки, значение указателя перемещается на начало следующей записи и счетчик записей уменьшается на единицу.

**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.

**В дескриптор дека входят:**

– имя дека;

– адрес верхней границы;

– адрес нижней границы;

– указатель нижний;

– указатель верхний;

– количество записей;

– общее количество записей.

**Основные операции над деком.**

– создать структуру данных типа дек;

– очистить дек;

– определить, пусть ли дек?

– добавить значение в начало (конец) дека;

– взять элемент из начала (конца) дека;

– найти начало (конец) дека;

– закончить работу.

## **Списки**

**Линейным списком** называется упорядоченная последовательность данных *х*1, *х*2,...,*хn* , *n* ≥ 0. При этом каждый элемент списка содержит одинаковое количество полей. Количество элементов списка называется его **длиной**. Структурные свойства списка ограничиваются линейным расположением элементов

*x*1...*хk*, *xk*+1... *xn* , где 1 < *k* < *k*+1 < *n.*

Поскольку списки представляют собой упорядоченную последовательность, надо определить первый элемент списка, который чаще всего называют головой списка. Затем второй элемент и т.д. списки редко имеют фиксированную длину. Обычно, их длина меняется в ходе выполнения программы.

Упорядоченность элементов списка может задаваться неявно путем их последовательного расположения, как в логической структуре, так и в памяти ЭВМ. Такой список **называется последовательным**.

Второй способ задания упорядоченности – применение указателей, расположенных в элементах и дающих возможность для любого элемента определить его предшественника или последователя. Такие списки называются **связными**. Если каждая запись содержит лишь один указатель, то список **односвязный**, при большем числе указателей список **многосвязный**.

Списки могут быть: линейными однонаправленными (односвязными), линейными двунаправленными, многосвязными, нелинейными многосвязными.

На основе списков можно организовать такие структуры данных как: стек, очередь, дек, дерево, граф.

Если количество записей постоянно, то линейный список сводится к массиву, записи или таблице.

### **Линейный однонаправленный список**

Для динамического представления списков каждый элемент списка должен содержать кроме смысловой информации (поле данных) еще и поле типа указатель. Кроме того, должен быть указатель, который указывает на начало линейного списка.

### **Двунаправленный список**

В ряде задач необходимо иметь возможность продвижения по связному списку в обоих направлениях. Для этого в каждый элемент вводится дополнительный указатель, задающий продвижение по списку в обратном направлении

**В дескриптор двунаправленного списка входит:**

– имя списка (идентификатор);

– тип элементов списка;

– указатель начала списка;

– указатель конца списка;

– указатель текущего элемента списка.

При использовании двунаправленных списков ускоряются процессы поиски в информационном массиве, но увеличивается расход памяти под указатели.

2.2 Код программы

**Мain**

#include <windows.h>

#include <stdio.h>

#include <iostream>

#include <string>

#include "List.cpp"

#include "Stack.cpp"

#include "Queue.cpp"

#include "Dec.cpp"

using namespace std;

void queueTest() //проверка очереди

{

Queue<int> q("Очередь1"); //создание экземпляра очереди

q.push(10); //добавление в нее нескольких элементов

q.push(20);

q.push(30);

q.push(40);

q.push(50);

q.print(); //вывод состояния очереди на экран

while (!q.isEmpty()) //опустошение очереди

{

cout << q.pop() << ", ";

}

cout << endl;

}

void stackTest() //проверка стека

{

Stack<int> s("Стек1"); //создание экземпляра стека

s.push(60); //добавление в него нескольких элементов

s.push(70);

s.push(80);

s.push(90);

s.push(100);

s.print(); //вывод состояния стека на экран

while (!s.isEmpty()) //опустошение стека

{

cout << s.pop() << ", ";

}

cout << endl;

}

void decTest() //проверка дека

{

Dec<int> d("Дек1"); //создание экземпляра дека

d.pushLeft(-1); //добавление в него нескольких элементов слева

d.pushLeft(-2);

d.pushLeft(-3);

d.pushLeft(-4);

d.pushLeft(-5);

d.pushRight(1); //добавление в него нескольких элементов справа

d.pushRight(2);

d.pushRight(3);

d.pushRight(4);

d.pushRight(5);

d.printLeft(); //печать состояния очереди с использованием указателя на левый конец очереди

d.printRight(); //печать состояния очереди с использованием указателя на правый конец очереди.

bool isLeft = true;

while (!d.isEmpty()) //опустошение стека

{

if (isLeft)

cout << d.popLeft() << ", "; //поочередно, то слева то справа извлекаем элемент

else

cout << d.popRight() << ", ";

isLeft = !isLeft;

}

cout << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian"); //для коректной работы русской раскладки в консоли

//queueTest();

//stackTest();

decTest();

string input = "init"; //строка в которую будет вводится строка для разделения на гласные и согласные буквы

while (input[input.length() - 1] != '.') //пока не введена строка заканчивающаяся точкой

{

cout << "Введите последовательность латинских букв, ограниченную точкой без пробелов:\n"; //запращиваем ввод

cin >> input; //и читаем строку

}

Stack<char> gl("Гласные"); //стек гластных букв

Stack<char> sogl("Согласные"); //стек согласных букв

string gls = "AaEeIiOoUuYy"; //строка содержащая все гласные буквы

bool isSogl; //флажек того что текущая буква согласная

for (int i = 0; i < input.length() - 1; ++i) //каждую букву (за исключением точки) из входной строки

{

isSogl = true; //первоначально считаем текущую букву согласной

for (int j = 0; j < gls.length(); ++j) //сопостовляем с каждой гласной из строки gls

if (input[i] == gls[j]) //если соответствие найдено

{

gl.push(input[i]); //помещаем текущую букву в стек гласных

isSogl = false; //текущая буква на согласная

break; //прекращаем сопостовление для текущей буквы

}

if (isSogl) //если текущая буква распознана как согласная

sogl.push(input[i]); //помещаем ее в стек согласных

}

gl.print(); //вывод стека гласных на экран

sogl.print(); //вывод стека согласных на экран

system("pause");

return 0;

}

**List**

#ifndef LIST

#define LIST

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

template <class T>

class List //класс реализующий шаблонный односвязный список, на нем основана очередь и стек

{

protected:

struct Node //структура описывающая узел списка

{

T data; //данные (параметр)

Node \* next = NULL; //указатель на следующий элемент

};

Node \* \_head = NULL; //указатель на первый элемент списка

int \_size = 0; //кол-во элементов

string name; //название списка

public:

List(string name) //конструктор

{

this->name = name;

}

~List() //деструктор

{

clean(); //вызываем метод удаляющий все элементы списка

}

string getName()

{

return name;

}

int size() const

{

return \_size;

}

void clean() //удаление всех элементов списка, сброс состояния эеземпляра в начальное

{

if ((\_head == NULL) || (\_size == 0))

{

return;

}

Node \* current = \_head;

Node \* temp = new Node;

while (current->next != NULL)

{

temp = current;

current = current->next;

delete temp;

temp = NULL;

}

delete current;

\_head = NULL;

\_size = 0;

}

void print() //печать состояния

{

Node \* current = \_head;

while (current != NULL)

{

cout << current->data << ", ";

current = current->next;

}

cout << "\nsize=" << \_size << endl;

}

bool isContains(T data) const //содержит ли список узел со значением data

{

Node \* current = \_head;

while (current != NULL)

{

if (current->data == data)

{

return true;

}

current = current->next;

}

return false;

}

bool isEmpty() //список пуст

{

return \_size == 0;

}

protected:

void add(T data) //добавление элемента в конец списка

{

if ((\_head == NULL) || (\_size == 0))

{

\_head = new Node;

\_head->data = data;

}

else

{

Node \* temp = new Node;

temp->data = data;

temp->next = \_head;

\_head = temp;

}

++\_size;

}

};

#endif

**Queue**

#ifndef QUEUE

#define QUEUE

#include "List.cpp"

using namespace std;

template <class T>

class Queue : public List<T> //класс реализующий шаблонную очередь

{

public:

Queue(string name) : List<T>(name) {}

void push(T data) //занесение элемента

{

if ((List<T>::\_head == NULL) || (List<T>::\_size == 0))

{

List<T>::\_head = new typename List<T>::Node;

List<T>::\_head->data = data;

}

else

{

typename List<T>::Node \* current = List<T>::\_head;

while (current->next != NULL)

{

current = current->next;

}

typename List<T>::Node \* temp = new typename List<T>::Node;

temp->data = data;

current->next = temp;

}

++List<T>::\_size;

}

T& pop() //извлечение элемента

{

if (List<T>::\_size == 0)

throw -1;

T& data = List<T>::\_head->data;

List<T>::\_head = List<T>::\_head->next;

--List<T>::\_size;

return data;

}

T& bottom() //нижний(первый на выход) элемент очереди

{

if (List<T>::\_size == 0)

throw -1;

return List<T>::\_head->data;

}

void print() //печать состояния

{

cout << "Queue name=" << getName() << endl;

List<T>::print(); //печать состояния базового списка (вывод элементов)

cout << "bottom=";

if (List<T>::\_size == 0)

cout << "NULL";

else

cout << List<T>::\_head->data;

cout << "\n\n";

}

};

#endif

**Dec**

#ifndef DEC

#define DEC

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

template <class T>

class Dec //класс реализующий шаблонный дек (двунаправленый список)

{

protected:

struct Node //структура описывающая узел дека

{

T data; //данные (параметр)

Node \* next = NULL; //указатель на следующий элемент

Node \* prev = NULL; //указатель на предыдущий элемент

};

Node \* \_head = NULL; //указатель на первый элемент списка

Node \* \_tail = NULL; //указатель на носледний элемент списка

int \_size = 0; //кол-во элементов

string name; //название списка

public:

Dec(string name) //конструктор

{

this->name = name;

}

~Dec() //деструктор

{

clean(); //вызываем метод удаляющий все элементы списка

}

string getName()

{

return name;

}

int size() const

{

return \_size;

}

void clean() //удаление всех элементов списка, сброс состояния эеземпляра в начальное

{

if ((\_head == NULL) || (\_size == 0))

{

return;

}

Node \* current = \_head;

Node \* temp = new Node;

while (current->next != NULL)

{

temp = current;

current = current->next;

delete temp;

temp = NULL;

}

delete current;

\_head = NULL;

\_size = 0;

}

void printLeft() //печать состояния очереди с использованием указателя на левый конец очереди

{

cout << "Stack name=" << getName() << "\nLeft->Right\n";

Node \* current = \_head;

while (current != NULL)

{

cout << current->data << ", ";

current = current->next;

}

cout << "\nsize=" << \_size << endl;

if (\_size == 0)

cout << "bottom=NULL\ntop=NULL";

else

cout << "bottom=" << \_head->data << "\ntop=" << \_tail->data;

cout << "\n\n";

}

void printRight() //печать состояния очереди с использованием указателя на правый конец очереди

{

cout << "Stack name=" << getName() << "\nRight->Left\n";

Node \* current = \_tail;

while (current != NULL)

{

cout << current->data << ", ";

current = current->prev;

}

cout << "\nsize=" << \_size << endl;

if (\_size == 0)

cout << "bottom=NULL\ntop=NULL";

else

cout << "bottom=" << \_head->data << "\ntop=" << \_tail->data;

cout << "\n\n";

}

bool isContains(T data) const //содержит ли список узел со значением data

{

Node \* current = \_head;

while (current != NULL)

{

if (current->data == data)

{

return true;

}

current = current->next;

}

return false;

}

bool isEmpty() //список пуст

{

return \_size == 0;

}

void pushLeft(T data) //занесение элемента с левого конца

{

if ((\_head == NULL) || (\_size == 0))

{

\_head = \_tail = new Node;

\_head->data = data;

}

else

{

Node \* temp = new Node;

temp->data = data;

temp->next = \_head;

\_head->prev = temp;

\_head = temp;

}

++\_size;

}

void pushRight(T data) //занесение элемента с правого конца

{

if ((\_head == NULL) || (\_size == 0))

{

\_head = \_tail = new Node;

\_head->data = data;

}

else

{

Node \* temp = new Node;

temp->data = data;

temp->prev = \_tail;

\_tail->next = temp;

\_tail = temp;

}

++\_size;

}

T& popLeft() //извлечение элемента с левого конца

{

if (\_size == 0)

throw -1;

T& data = \_head->data;

\_head =\_head->next;

if (\_head != NULL)

\_head->prev = NULL;

--\_size;

return data;

}

T& popRight() //извлечение элемента с правого конца

{

if (\_size == 0)

throw -1;

T& data = \_tail->data;

\_tail = \_tail->prev;

if (\_tail != NULL)

\_tail->next = NULL;

--\_size;

return data;

}

};

#endif

**Stack**

#ifndef STACK

#define STACK

#include "List.cpp"

using namespace std;

template <class T>

class Stack : public List<T> //класс реализующий шаблонный стек

{

public:

Stack(string name) : List<T>(name) {}

void push(T data) //занесение элемента

{

List<T>::add(data);

}

T& pop() //извлечение элемента

{

if (List<T>::\_size == 0)

throw -1;

T& data = List<T>::\_head->data;

List<T>::\_head = List<T>::\_head->next;

--List<T>::\_size;

return data;

}

T& top() //элемент на вершине стека

{

if (List<T>::\_size == 0)

throw -1;

return List<T>::\_head->data;

}

void print() //печать состояния

{

cout << "Stack name=" << getName() << endl;

List<T>::print(); //печать состояния базового списка (вывод элементов)

cout << "top=";

if (List<T>::\_size == 0)

cout << "NULL";

else

cout << List<T>::\_head->data;

cout << "\n\n";

}

};

#endif

3 Вывод

Научились работать со структурами данных – стек ,очередь, дек, реализованными с помощью массивов и связных списков.

Выполнили задания, написали отчет.