ФГБОУ ВО “Чувашский государственный университет им.

И.Н. Ульянова”

Кафедра вычислительной техники

Лабораторная работа №1

Стеки. Очереди.

Вариант 29

Выполнил:

Студент гр. ИВТ-41-22

Ярдыков Эдуард Евгеньевич

Проверил:

кандидат технических наук

Павлов Леонид Александрович

Чебоксары 2023

**Цель работы** – ознакомление со способами реализации стеков и очередей и выполняемых над ними операций; получение практических навыков программирования задач с использованием стеков и очередей.

**Подготовка к работе**

1. **Реализовать стек и очередь с использованием последовательного (массив) и связанного (динамические списковые структуры) распределений элементов. Для каждого способа представления написать подпрограммы, реализующие операции над стеками и очередями:**

* **Процедура PUSH(*S*,*x*), которая в стек *S* помещает элемент *x*;**

Массив

t := t+1

**if** t > m

**then** {переполнение}

**else** S(t) := x

Односвязный список

new(l)

l.info := x

l.next := t

t := l

* **Функция POP(S), которая удаляет элемент из вершины стека S, т.е. операция x:=POP(S) удаляет элемент из вершины стека S и присваивает его значение переменной x (не определена для пустого стека);**

Массив

**if** t = 0

**then** {Стек пуст}

**else** {x = S(t)

t = t-1}

Односвязный список

**if** t = Ʌ

**then**{стек пуст}

**else**{x := t.info

l := t

t := t.next

dispose(l)}

* **Функция EMPTY(*S*), которая возвращает значение TRUE или FALSE в зависимости от того, является ли стек *S* пустым или нет;**

Массив

**if** t=0

**then** {True}

**else** {False}

Односвязный список

**if** t=Ʌ

**then**{True}

**else**{False}

* **Функция STACKTOP(*S*), которая определяет значение элемента в вершине стека *S* без его удаления (не определена для пустого стека);**

Массив

**if** t=0

**then** {Стек пуст}

**else** {x = S(t)}

Односвязный список

**if** t = Ʌ

**then**{стек пуст}

**else**{x := t.info}

* **Процедура INSERT(*Q*,*x*), которая помещает элемент *x* в конец очереди** *Q*;

Массив

r := (r+1)**mod** m

**if** r = f

**then** {переполнение}

**else** Q(r):= x

Односвязный список

new(r.next)

r := r.next

r.info := x

r.next := Ʌ

* **Функция REMOVE(Q), которая удаляет элемент из начала очереди Q, т. е. операция x:=REMOVE(Q) удаляет элемент из начала очереди Q и присваивает его значение переменной x (не определена для пустой очереди);**

Массив

**if** r = f

**then**{очередь пуста}

**else** {f := (f+1)**mod m**

x := Q(f)

Односвязный список

**if** f=r

**then**// очередь пуста

**else** x := f.next.info

l := f

f := f.next

dispose(l)

* **Функция EMPTY(Q), которая возвращает значение TRUE или FALSE в зависимости от того, является ли очередь Q пустой или нет.**

Массив

**if** r=f

**then** {True}

**else** {False}

Односвязный список

**if** f=r

**then** {True}

**else** {False}

Примечание:

t-вершина стека;

m-размер массива , в котором хранится стек или очередь;

r-указатель на конец очереди;

f-указатель на символ перед начальным символом очереди;

1. Разработать метод поддержания в одном линейном массиве двух стеков, при котором ни один из стеков не переполняется до тех пор, пока весь массив не будет заполнен. При этом стек никогда не перемещается внутри массива на другие позиции. Написать подпрограммы, реализующие операции PUSH1, PUSH2, POP1 и POP2, манипулирующие обоими стеками, учитывая, что стеки растут навстречу друг другу.

Push1

t1 := t1 + 1

**if** t1 > m1

**then**{Переполнение стека 1}

**else** S(t1) := x

Pop1

**if** t1 = 0

**then** {Стек 1 пуст}

**else** {x = S(t1)

t1 = t1-1}

Push2

t2 := m-1

m2 := m-m1

**if** t2 > m **or** t2 = m1

**then**{Переполнение стека 2}

**else** S(t2) := x

Pop2

**if** t2 = m

**then** {Стек 2 пуст}

**else** {x := S(t2)

t2 := t2 + 1}

Примечание:

t1- Вершина первого стека

t2- Вершина второго стека

m1-число элементов в первом стеке

m2- число элементов во втором стеке

m-число элементов в массиве

1. Реализовать очередь на базе двух стеков. Определить время работы операций с очередью.

* **Процедура INSERT;**

**if** t2 > m2

**then**{очередь

переполнена}

**else Push(S2,x)**

Время ti

t1

t2

t3

Число повторений

1

1

1

Bременная сложность алгоритма есть O(1).

* **Функция REMOVE;**

**if** t2=0

**then** {Список пуст}

**else**{ **while** t2 > 0{

a := **Pop(t2)**

**Push(S1,a)**}

x := **Pop(t1)**

**while t1>0**{

a := **Pop(t1)**

**Push(S2,a)**} }

Время ti

t1

t2

t3

t4

t5

t6

t7

t8

t9

Число повторений

1

1

n+1

n

n

n

n

n-1

n-1

Примечание:

Размерности стеков S1 и S2 равны

T(n)= t1+t2+(n+1)t3+nt4+nt5+nt6+nt7(n-1)t8+(n-1)t9

T(n)=1+1+n+1+n+n+n+n+n-1=1+7n

Bременная сложность алгоритма есть O(n).

1. Реализовать стек на базе двух очередей. Определить время работы стековых операций.

* **Процедура PUSH**

**INSERT(*Q2*,*x*),**

Время выполнения

t1

Число повторений

1

Bременная сложность алгоритма есть O(1).

* **Функция POP**

**if** r2=f2

**then**{Стек пуст}  
**else**{ **while** r2≠f2{

a:=**Remove(Q2)**

**Insert(Q1,a)**}

x:=**Remove(Q1)**

**while** r1≠f2 {

a := **Remove(Q1)**

**Insert(Q2,a)**} }

Время ti

t1

t2

t3

t4

t5

t6

t7

t8

t9

Число повторений

1

1

n+1

n

n

n

n

n-1

T(n)= t1+t2+(n+1)t3+nt4+nt5+nt6+nt7(n-1)t8+(n-1)t9

T(n)=1+1+n+1+n+n+n+n+n-1=1+7n

Bременная сложность алгоритма есть O(n).

1. Реализовать дек (от англ. deque – double ended queue) – структуру данных, представляющую из себя список элементов, в который добавление новых элементов и удаление существующих производится с обоих концов. Эта структура поддерживает как FIFO, так и LIFO, поэтому на ней можно реализовать как стек, так и очередь. В первом случае нужно использовать только методы головы или хвоста, во втором – методы push и pop двух разных концов. Дек можно воспринимать как двустороннюю очередь. Он имеет следующие операции:

* **empty – проверка на наличие элементов,**

**if** h=0 and t=m

**then**{True}

**else**{False}

* **pushBack (запись в конец) – операция вставки нового элемента в конец,**

**if** h=t

**then**{Дек переполнен}

**else**{t := t-1

d(t) := x}

* **popBack (снятие с конца) – операция удаления конечного элемента,**

**if** h = 0

**then** {Дек пуст}

**else** {x := d(t)

t := t-1}

* **pushFront (запись в начало) – операция вставки нового элемента в начало,**

**if** h = t

**then**{Дек переполнен}

**else**{

h := h-1

d(h) := x }

* **popFront (снятие с начала) – операция удаления начального элемента.**

**if** (empty())

**then** {Дек пуст}

**else** {x := d[h]

h := h + 1}

6. Разработать алгоритмы и программы решения задач в соответствии с заданными вариантами (использовать одно из представлений стеков и очередей).

**Блок 1.**

Преобразование префиксной формы записи выражения в постфиксную.

while (!a.empty()) {

string a\_ = a.pop();

if (a\_ >= "a" && a\_ <= "z") {

is\_word = true;

}

else {

temp += b.pop();

temp += b.pop();

temp += a\_;

cout << temp << endl;

is\_word = false;

b.push(temp);

temp.clear();

}

if (is\_word) {

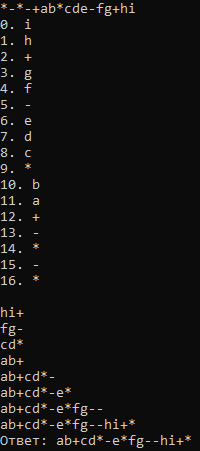
b.push(a\_);

}

}

cout << "Ответ: " << b.stack\_top();

Результаты выполнения алгоритма:



**Блок 2.**

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает. Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту. Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды). Смоделировать игру и определить, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

Алгоритм:

while (!a.empty() || !b.empty()) {

stack a\_, b\_, \_a, \_b;

if ((a.stack\_top() > b.stack\_top()) || (a.stack\_top() == 0 && b.stack\_top() == 9)) {

printf("Выйграла колода A\n");

a\_.push(b.stack\_top()); b.pop();

a\_.push(a.stack\_top()); a.pop();

while(!(a.empty()))

\_a.push(a.pop());

while (!(\_a.empty()))

a\_.push(\_a.pop());

cout << "Стек А: \t Стек B:\n";

a = a\_;

a.print\_2stack(a, b);

}

else if ((b.stack\_top() > a.stack\_top()) || (b.stack\_top() == 0 && a.stack\_top() == 9)) {

printf("Выйграла колода B\n");

b\_.push(a.stack\_top()); a.pop();

b\_.push(b.stack\_top()); b.pop();

while (!(b.empty()))

\_b.push(b.pop());

while (!(\_b.empty()))

b\_.push(\_b.pop());

cout << "Стек А: \t Стек B:\n";

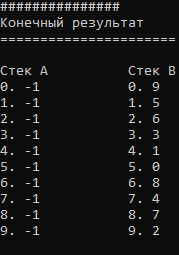
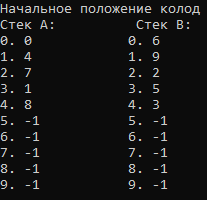
b = b\_;

a.print\_2stack(a, b);

}

cout << "###############" << endl;

if ((a.empty()) || (b.empty())) break;



Вывод: Я ознакомился со способами реализации стеков и очередей и выполняемых над ними операций; получил практические навыки программирования задач с использованием стеков и очередей.