ФГБОУ ВО “Чувашский государственный университет им.

И.Н. Ульянова”

Кафедра вычислительной техники

Лабораторная работа №1

Стеки. Очереди. Деки.

Вариант 9.

Выполнил: студент гр. Ивт-41-22

Иванов В.С.

Проверил: кандидат технических наук

Павлов Л.А.

***Цель работы*** – изучить основные способы реализации стеков, очередей и деков, выполняемые над ними операции; получить практические навыки программирования задач с использованием стеков, очередей, деков.

Подготовка к работе

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:

*1)Empty*(*S*) – проверка на наличие элементов (проверка пустоты стека):

Массив

**if** t=0

**then** {True}

**else** {False}

Односвязный список

**if** t=NULL

**then**{True}

**else**{False}

2)Push(S, x) – в вершину стека 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

*3)Pop*(*S*) – удаляет элемент из вершины стека *S* и возвращает его значение (не определена для пустого стека);

Массив

**if** t = 0

**then** {Стек пуст}

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

t = t-1}

Односвязный список

**if** t = NULL

**then**{стек пуст}

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

l := t

t := t.next

dispose(l)}

4) *StackTop*(*S*) – возвращает 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 = NULL

**then**{стек пуст}

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

2. Реализовать очередь с использованием последовательного (массив) и связного (динамические списковые структуры) распределений элементов. Для каждого способа представления разработать алгоритмы, реализующие выполнение следующих операций над очередями:

1) *EmptyQueue*(*Q*) – проверка на наличие элементов (проверка пустоты очереди):

Массив

**if** r=f

**then** {True}

**else** {False}

Односвязный список

**if** f=r

**then** {True}

**else** {False}

2) *InsQueue*(*Q*, *x*) – 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 = NULL

3) *DelQueue*(*Q*) – удаляет элемент из начала очереди *Q* и возвращает его значение (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)

3. Реализовать дек с использованием последовательного (массив) и связного (динамические списковые структуры) распределений элементов. Для каждого способа представления разработать алгоритмы, реализующие выполнение следующих операций над деками:

1) *Empty*(*D*) – проверка на наличие элементов (проверка пустоты дека):

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

**then**{True}

**else**{False}

2) *PushBack*(*D*, *x*) – помещает элемент *x* в конец дека *D*:

**if** h=t

**then**{Дек переполнен}

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

d(t) := x}

3) *PushFront*(*D*, *x*) – помещает элемент *x* в начало дека *D*:

**if** h = t

**then**{Дек переполнен}

**else**{

h := h-1

d(h) := x }

4) *PopBack*(*D*) – удаляет элемент из конца дека *D* и возвращает его значение (не определена для пустого дека):

**if** h = 0

**then** {Дек пуст}

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

t := t-1}

5) *PopFront* (*D*) – удаляет элемент из начала дека *D* и возвращает его значение (не определена для пустого дека):

4. 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 *Push*1, *Push*2, *Pop*1, *Pop*2, *Empty*1 и *Empty*2, манипулирующие обоими стеками, учитывая, что стеки растут навстречу друг другу.

Push 1

t1 := t1 + 1

if t1 > m1

then{Переполнение стека 1}

else S(t1) := x

Push 2

t2 := m-1

m2 := m-m1

if t2 > m or t2 = m1

then{Переполнение стека 2}

else S(t2) := x

Pop1

if t1 = 0

then {Стек 1 пуст}

else {x = S(t1)

t1 = t1-1}

Pop2

if t2 = m

then {Стек 2 пуст}

else {x := S(t2)

t2 := t2 + 1}

5. Реализовать очередь на базе двух стеков. Определить время работы (асимптотику) 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).

6. Реализовать стек на базе двух очередей. Определить время работы (асимптотику) 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

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).

7. Разработать алгоритмы и программы решения задач в соответствии с заданными вариантами (***из каждого блока по 1 задаче***), используя одно из представлений стеков, очередей или деков.

Блок 1.

Вычисление значения арифметического выражения (без переменных), записанного в префиксной форме. Например, выражение \*+123 будет равно 9.

Алгоритм:

if (Char == '+' || Char == '-' || Char == '\*' || Char == '/') {

if (Char == '+') {

return operand1+operand2;

}

if (Char == '-') {

return operand1 - operand2;

}

if (Char == '\*') {

return operand1 \* operand2;

}

if (Char == '/') {

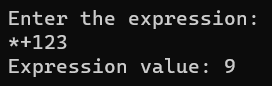
return operand1 / operand2;

}

}

return Char - '0';

Результат выполнения:

****

**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(koloda\_A[0]!=-1 || koloda\_B[0] != -1) {

if ((koloda\_A[0] > koloda\_B[0]) || (koloda\_A[0] == 0 && koloda\_B[0] == 9)) {

koloda\_A.push\_back(koloda\_A[0]);

koloda\_A.push\_back(koloda\_B[0]);

koloda\_A.erase(koloda\_A.begin());

koloda\_B.erase(koloda\_B.begin());

koloda\_B.push\_back(-1);

koloda\_B.push\_back(-1);

auto a = partition(koloda\_B.begin(), koloda\_B.end(), [](int num) {

return num != -1;

});

auto b = partition(koloda\_A.begin(), koloda\_A.end(), [](int num) {

return num != -1;

});

}

else {

koloda\_B.push\_back(koloda\_A[0]);

koloda\_B.push\_back(koloda\_B[0]);

koloda\_A.erase(koloda\_A.begin());

koloda\_B.erase(koloda\_B.begin());

koloda\_A.push\_back(-1);

koloda\_A.push\_back(-1);

auto a = partition(koloda\_A.begin(), koloda\_A.end(), [](int num) {

return num != -1;

});

auto b = partition(koloda\_B.begin(), koloda\_B.end(), [](int num) {

return num != -1;

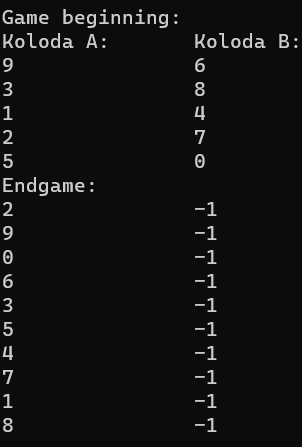
});

}

if (koloda\_A[0] == -1 || koloda\_B[0] == -1) break;

}\

Пример работы:

****

Вывод: изучил основные способы реализации стеков, очередей и деков, выполняемые над ними операции; получил практические навыки программирования задач с использованием стеков, очередей, деков.