# Лекция 4

## Очередь, стек, дек

Всем перечисленным структурам свойственна операция "выборка" (чтение с удалением).

**Очередью** называется структура данных, добавление записи в которую производится в конец цепочки, а выборка осуществляется из начала цепочки. Очередь работает по принципу

FIFO (First-In, First-Out) –поступивший первым, обслуживается первым.

**Стеком** называется структура данных, добавление записи в которую и выборка записи из которой производится из начала цепочки (вершины стека). Стек работает по принципу

LIFO (Last-In, First-Out) – поступивший последним, обслуживается первым.

**Деком** называется структура данных, у которой добавление и выборка записей осуществляется как в начале, так и в конце цепочки. Дек является обобщением структур данных очередь и стек.

***Логические описания.***

Пусть записи принадлежат типуInf;

S – стек;S'- непустой стек;

О – очередь;О' – непустая очередь.

Тогда можно определить стек и очередь следующим образом:

Стек

S=(пусто|S');

S'=(Inf i,S)

Oчередь

О=(пусто|О');

О'=(Inf i,O |O,Inf i).

Это **рекурсивные** определения структур стек и очередь. Рекурсивное определение предполагает использование рекурсивных алгоритмов при реализации операций.

Операции над **деком** включают все операции, перечисленные в таблице, учитывая, что S и O есть частные случаи дека.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя операции** | **Функциональ-ные спецификации** | **Аргументы** | **Результат** | **Описание** |
| Создать | S  O |  | пустой s  пустая o | Создается пустой стек или очередь |
| Пусто? | SBoolean  OBoolean | s  o | истина или ложь | Пуст ли стек, или очередь ? |
| Первый | ST  OT | (t, s)  (t, o) | t  t | Определение элемента подлежащего обслуживанию |
| Выборка | SS  OO | s  o | s,t  o,t | Из стека или очереди выбран первый элемент |
| Добавление | TSS  TOO | t, s  t, o | (t, s)  (t,o) | Добавлен новый элемент |

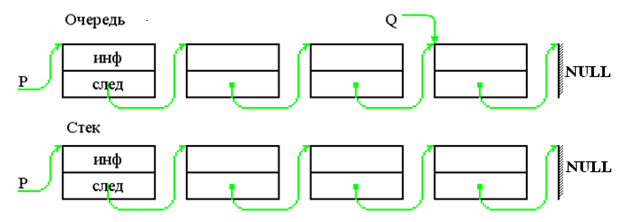
***Задача.***

Представить структуры очереди, стека и дека с помощью массивов и реализовать операции.

При динамическом представлении структур записи можно представить так:

Struct Elem {Inf инф; Elem\*след).

Для удобства реализаций стек задается одним указателем на голову, а очередь – двумя (на первый и последний элемент)



***Задачи***

Выборка(Pelem p,Inf i) – из структуры p (очередь или стек) выбрать в переменную i данное.

Пусть p не пуст, тогда

i=p->инф; p1=p; p=p->след; delete(p1)

Распечатать информационные части стека p.

|  |  |
| --- | --- |
| Традиционное решение | Рекурсивное решение |
| Распечатать(Elem p)  {  q=p;  while (p!=NULL)  {  печ(p->инф);  p=p->след;  }  } | Распечатать(Elem p)  {  if (p!=NULL)  {  печ(p->инф);  Распечатать(p->след);  }  } |

1. Распечатать информационные части стека в обратном порядке.

2. Превратить стек p в очередь p,q.

3. Перевернуть файл (используем стек p).

#include <iostream>

using namespace std;

class Elem

{

public:

int x;// Это информационная часть

Elem\* next; // Это указатель на следующий элемент в цепочке

Elem(int x) // Конструктор с одним параметром – значение инф. части

{

this->x=x; // this - системный указатель на сконструированный объект

next=NULL;

}

};

class Stack

{

public:

Elem\* head;

Stack() {head=NULL;}// Конструктор создает пустой стек

void print() // метод печатает стек

{

Elem\* el=head;

while(el)

{

cout<<el->x<<" ";

el=el->next;

}

cout<<endl;

}

int getCount()// метод подсчитывает количество элементов в стеке (Лишнее!!!)

{

int count=0;

Elem\* el=head;

while(el)

{

count++;

el=el->next;

}

return count;

}

void add(int x) // добавить

{

Elem\* el=new Elem(x);

el->next=head;

head=el;

}

bool isEmpty(){return (head==NULL);} // пуст ли стек?

Elem\* get(){} // Выборка

Elem\* readHead(){} // чтение из головы

};

int main()

{

Stack st;

cout<<st.isEmpty() <<endl;

st.add(17);

st.print();

st.add(34);

st.print();

cout<<st.isEmpty() <<endl;

cout<<st.getCount()<<endl;

system ("pause");

}

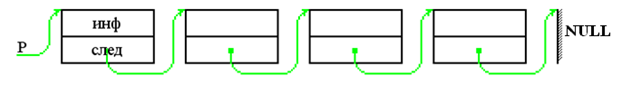
**Лабораторная работа №4а**. Построить динамическую модель стека (или очереди)

# Дек

**Деком** называется структура данных, у которой добавление и выборка записей осуществляется как в начале, так и в конце цепочки. Дек является обобщением структур данных очередь и стек.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя операции** | **Функциональ-ные спецификации** | **Аргументы** | **Результат** | **Описание** |
| Создать | D |  | пустой d | Создается пустой дек |
| Пусто? | DBoolean | d | истина или ложь | Пуст ли дек? |
| Выборка из начала | DD,t | d | d,t | Из дека выбран первый элемент |
| Выборка из конца | DD,t | d | d,t | Из дека выбран последний элемент |
| Добавление в начало | TDD | t, d | d | Добавлен новый элемент |
| Добавлениев конец | TDD | t, d | d | Добавлен новый элемент |

1. Единственный указатель на начало структуры.
2. Операции – Создать, Пуста ли структура?, Добавить в начало, Добавить в конец, Удалить из начала, Удалить из конца.
3. Вспомогательные операции – Получить указатель на последний элемент, Получить указатели на последний и предпоследний элементы.



p – указатель на начало дека

#include<iostream>

using namespace std;

class Elem

{

public:

int x;// Это информационная часть

Elem\* next; // Это указатель на следующий элемент в цепочке

Elem(int x) // Конструктор с параметром - информационная часть

{

this->x=x; // this - системный указатель на сконструированный объект

next=NULL;

}

};

classDack

{

public:

Elem\* head;

Dack() {head=NULL;}// Конструктор создает пустой дек

void print() // метод распечатывает информационные части дека

{

Elem\* el=head;

while(el)

{

cout<<el->x<<" ";

el=el->next;

}

cout<<endl;

}

voidadd\_b(int x) {

Elem\* el=new Elem(x);

el->next=head;

head=el;

}// добавить в начало

void add\_f(intx) {}// добавить в конец

bool isEmpty(){return (head==NULL);} // пустлидек?

Elem\* get\_b(){} // Выборка из начала

Elem\* get\_f(){} // Выборка с конца

};

//Получить указатель на последний элемент

Elem\* last(Elem\* p)

{

if(p==NULL) return NULL;

Elem\*q;

q=p;

do

q=q->next;

while(q->next!=NULL);

return q;

}

//Получить указатель на предпоследний элемент

Elem\* predlast(Elem\* p)

{

if(p==NULL) return NULL;

if(p->next==NULL) return NULL;

Elem\*q,\*q1;

q=p;

q1=q->next;

do

{

q=q->next;

q1=q1->next;

}

while(q1->next!=NULL);

return q;

}

int main()

{

Dack dack;

dack.add\_b(44);

dack.add\_b(22);

dack.add\_b(11);

Elem\* q=last(dack.head);

Elem\* predLast=predlast(dack.head);

cout<<q->x<<" "<<predLast->x<<endl;

system("pause");

return 0;

}

**Лабораторная работа №4в**. Построить динамическую модель дека

## 

# Лекция 5

## Обратная польская (постфиксная) запись

Привычное определение арифметического выражения:

1. Арифметическая константа или инициализированная арифметическая переменная является арифметическим выражением.
2. Если А и В – арифметические выражения, то А#В и (А#В) – арифметические выражения.

Привычная форма записи арифметического выражения

((a+b\*(c-e)/(d+f))/(a\*b))/c

Арифметическое выражение в форме обратной польской записи (ПФ):

1. Арифметическая константа или инициализированная арифметическая переменная является арифметическим выражением ПФ.
2. Если А и В – арифметические выражения ПФ, то АВ # – арифметическое выражение ПФ.

Постфиксная форма записи того же арифметического выражения

abce - \* df + / + ab \* / c /

Как получить обратную польскую запись?

**Рекурсивное решение:** Если арифметическое выражение является арифметической константой или инициализированной арифметической переменной, то по определению это и есть арифметическое выражение ПФ. Если арифметическое выражение имеет вид А#В или (А#В), то нужно записать ПФ для А, далее приписать ПФ для В, а затем записать знак операции #.

((a+b\*(c-e)/(d+f))/(a\*b))/c

(((a+((b\*(c-e))/(d+f)))/(a\*b))/c)

(((a+((b\*(c-e))/(d+f)))/(a\*b))/c)

((a+((b\*(c-e))/(d+f)))/(a\*b))c/

((a+((b\*(c-e))/(d+f)))/(a\*b))c/

((a+((b\*(c-e))/(d+f)))/(a\*b))c/

(a+((b\*(c-e))/(d+f)))(a\*b)/c/

(a+((b\*(c-e))/(d+f)))(a\*b)/c/

(a+((b\*(c-e))/(d+f)))(a\*b)/c/

a((b\*(c-e))/(d+f))+ab\*/c/

a((b\*(c-e))/(d+f))+ab\*/c/

a((b\*(c-e))/(d+f))+ab\*/c/

a(b\*(c-e))(d+f)/+ab\*/c/

a(b\*(c-e))(d+f)/+ab\*/c/

a(b\*(c-e))(d+f)/+ab\*/c/

ab(c-e)\*df+/+ab\*/c/

ab(c-e)\*df+/+ab\*/c/

ab(c-e)\*df+/+ab\*/c/

abce-\*df+/+ab\*/c/

abce-\*df+/+ab\*/c/

***Задача.***

Вычислить значение арифметического выражения, записанного в виде обратной польской записи.

***Решение.***

Создаем пустой стек. Читаем арифметическое выражение ПФ слева направо. Если встречается константа или имя инициализированной переменной, то в стек вталкивается соответствующее значение. Если встречается знак арифметической операции, то из стека выбираются два значения, над ними выполняется операция (первое значение – правый операнд, второе значение – левый операнд), результат операции вталкивается в стек.

**Докажите**, что в результате такой обработки правильного выражения ПФ в вершине стека будет лежать значение арифметического выражения, других записей в стеке не будет.

Пусть a=1, b=2, c=3, d=4, e=5, f=6, тогда алгоритм вычисления выражения

abce - \* df + / + ab \* / c / реализуется так:

|  |  |  |
| --- | --- | --- |
| **Номер действия** | **Содержание стека** | **Обратная польская запись** |
|  |  | 1,2,3,5,-,\*,4,6,+,/,+,1,2,\*,/,3,/ |
|  | 1 | 2,3,5,-,\*,4,6,+,/,+,1,2,\*,/,3,/ |
|  | 1,2 | 3,5,-,\*,4,6,+,/,+,1,2,\*,/,3,/ |
|  | 1,2,3 | 5,-,\*,4,6,+,/,+,1,2,\*,/,3,/ |
| 1 | 1,2,3,5 | -,\*,4,6,+,/,+,1,2,\*,/,3,/ |
| 2 | 1,2,-2 | \*,4,6,+,/,+,1,2,\*,/,3,/ |
|  | 1,-4 | 4,6,+,/,+,1,2,\*,/,3,/ |
|  | 1,-4,4 | 6,+,/,+,1,2,\*,/,3,/ |
| 3 | 1,-4,4,6 | +,/,+,1,2,\*,/,3,/ |
| 4 | 1,-4,10 | /,+,1,2,\*,/,3,/ |
| 5 | 1,-0.4 | +,1,2,\*,/,3,/ |
|  | 0.6 | 1,2,\*,/,3,/ |
|  | 0.6,1 | 2,\*,/,3,/ |
| 6 | 0.6,1,2 | \*,/,3,/ |
| 7 | 0.6,2 | /,3,/ |
|  | 0.3 | 3,/ |
| 8 | 0.3,3 | / |
|  | 0.1 |  |