第三章 栈与队列

第3章 栈与队列

一. 概念

1. 栈
2. 队列
3. 循环队列

二. 方法

4. ★栈的性质，用栈来生成序列，栈的实现
5. 队列的性质，用队列生成序列
6. ★队列的实现
7. ★利用栈来消除递归
8. 栈的灵活应用，例如表达式求值 (中缀表达式转后缀表达式的算法、后缀表达式求值算法)

1.栈

概念

后进先出(LIFO)的线性表（限制访问端口），也叫下推表

实现方式

顺序栈——大小确定
-使用向量实现，是顺序表的简化版 -关键在于确定栈顶端
-上溢、下溢问题

链式栈——大小可以不定
-用单链表存储

2.队列

概念

先进先出(FIFO)的线性表，只能访问队头和队尾

实现方式

顺序队列——长度确定
-用向量存储元素，front待出队的位置，rear待入队的位置
-上溢出、下溢出、假溢出
-假溢出要用循环队列解决

链式队列——长度可以不定
-单链表实现

3.循环队列

4.栈的性质，用栈来生成序列，栈的实现

栈的性质

后进先出是栈的最主要性质

用栈来生成序列

火车进出栈问题——构造进栈出栈序列，利用一一映射找出所有不合法序列，n个元素可以生成的序列个数为Catalan数 $\frac{1}{n+1}C_{2n}^n$

栈的实现

```
//顺序栈的实现
template <class T>
class arrStack{
private:
    int maxSize;
    int top;// 栈顶位置，应小于maxSize
    T*st;
public:
    arrStack(int size) {
        maxSize = size;
        top = -1;
        st = new T[maxSize];
    }
    arrStack() {
        top = -1;
    }
    ~arrStack() {
        delete [] st;
    }
    void clear() {
        top = -1;
    }
    void push(T val){
        if(top==maxSize)//上溢出
            return;
        st[++top]=val;
    }
    void pop(){
        if(top==-1)//下溢出
            return;
        top--;
```

```
    }
    T& top(){
        return st[top];
    }
    bool empty(){
        return top==-1;
    }
};
```

```
//链式栈的实现
template<class T>
class listNode{
public:
    T val;
    listNode*next;
    listNode(T v=-1,listNode*n=nullptr):
        val(v),next(n){}
};
template <class T>
class lnkStack{
private:
    listNode<T>*top;//指向栈顶的指针
    int curSize;
public:
    lnkStack(){
        top=nullptr;
        curSize=0;
    }
    void clear(){
        listNode<T>*tmp=top;
        while(tmp){
            listNode<T>*p=tmp;
            tmp=tmp->next;
            delete p;
        }
        top=nullptr;
        curSize=0;
    }
    ~lnkStack(){
        clear();
    }
    void push(T val){
        curSize++;
        listNode<T>*p=new listNode<T>(val,top);
        top=p;
    }
    void pop(){
        if(curSize==0)
            return;
        curSize--;
        listNode<T>*p=top;
```

```
            top=top->next;
            delete p;
        }
        bool empty(){
            return top==nullptr;
        }
    };
```

5.队列的性质，用队列生成序列

6.队列的实现

```
//顺序（循环）队列的实现
template<class T>
class queue{
private:
    int maxSize;
    int front,rear;
    T*arr;
public:
    queue(int size){
        maxSize=size+1;//size是队列最大长度，maxSize是数组长
                       //要留一个空位
        rear=0,front=1;
        arr=new T[maxSize];
    }
    ~queue(){
        delete []arr;
    }
    void clear(){
        delete []arr;
        maxSize=0;
        rear=0,front=1;//初始时数组第0位留作空位
    }
    void enqueue(T val){
        if(front==(rear+2)%2)//满
            return;
        rear=(rear+1)%maxSize;
        arr[rear]=val;
    }
    void dequeue(){
        if(front==(rear+1)%maxSize)//空
            return;
        front=(front+1)%maxSize;
    }
};
```

```
//链式队列的实现
template<class T>
class listNode{
public:
    T val;
    listNode*next;
};
template<class T>
class queue{
private:
    int curSize;
    listNode<T> *front,*rear;
public:
    queue(){
        curSize=0;
        front=rear=nullptr;
    }
    ~queue(){
        listNode<T>*p=front;
        while(p){
            listNode<T>*q=p;
            p=p->next;
            delete q;
        }
    }
    void clear(){
        listNode<T>*p=front;
        while(p){
            listNode<T>*q=p;
            p=p->next;
            delete q;
        }
        font=rear=nullptr;
        curSize=0;
    }
    void enqueue(T val){
        curSize++;
        listNode*tmp;
        tmp->val=val;
        rear->next=tmp;
        rear=rear->next;
    }
    void dequeue(){
        curSize--;
        listNode*tmp=front;
        front=front->next;
        tmp->next=nullptr;
        delete tmp;
    }
};
```

7.利用栈来消除递归

a.设置工作栈

b.设置t+2个label

c.增加非递归入口

d.替换第i(i=1,2,...,t)个递归规则

e.所有递归出口处增加语句：goto labelt+1;

f.标号为t+1的语句的格式

g.改写循环和嵌套中的递归

h.优化处理

8.栈的灵活应用

括号匹配
前缀、中缀、后缀表达式的计算
递归转非递归
单调栈