**第三章 栈和队列**

**一、教学基本要求**

1．掌握栈的概念及基本操作。

2．掌握队列的概念及基本操作。

本章目的是介绍栈和队列的逻辑结构定义及在两种存储结构上如柯实现栈和队列的基本运算。要求在掌握栈和队列的特点的基础上，懂得在什么样的情况下能够使用栈或队列。本章重点是掌握栈和队列在两种存储结构上实现的基本运算，难点是循环队列中对边界条件的处理。

重点：掌握栈和队列在两种存储结构上实现的基本运算。

难点：循环队列中对边界条件的处理。

**二、学时安排 总学时：6**

|  |  |
| --- | --- |
| 教 学 内 容 | 学 时 |
| 1．栈 | 2 |
| 2．表达式求值 栈与递归过程 | 2 |
| 3．队列 | 2 |

**三、教学内容**

一、栈

1. 栈的定义 栈是只准在表尾进行插入和删除的线性表，称为LIFO表(即后进先出表)。允许插入和删除的一端叫栈顶，另一端叫栈底。

栈的抽象数据类型：ADT Stack（讲45页的类型）

2.堆栈的基本运算

（1） 入栈 Push(&S, e) 往栈S中插入一个元素e；

（2） 出栈 Pop(&S, &e) 栈顶指针下移；

（3） 取栈顶元素 Gettop(&S, e)，将栈s的栈顶元素赋值给x；

（4） 判栈空 Stackempty(&S) 若栈空，则结果为true,否则为false；

3.栈的工作方式

栈底固定，栈顶可动

4.栈的顺序存储结构

顺序栈：利用一组地址连续的存储单元依次存放自栈底到栈项的数据元素。

typedef struct{

SelemType \*base;

SelemType \*top;

int stacksize;

}SqStack;

base：栈底指针，指向栈底的位置。base=NULL，栈的结构不存在，即没有这个栈，top：栈顶指针，top始终在栈顶元素的下一个位置，其初值指向栈底，top=base 是栈空的标志。

插入时，给top赋值，然后top 增1；

删除时，top-1；然后取top的值。

stacksize：当前已分配的存储空间，以元素为单位。

5.栈的顺序存储结构下操作的实现

【算法3.1】

 Status Push(SqStack &S, SelemType e){

//将e压入栈S

……

\*S.top++=e;//首先做运算S.top=e;　然后进行操作S.top++;

……

}//Push

【算法3.2】

 Status Pop(SqStack &S, SelemType &e){

//若栈不空，用e返回栈顶元素

if (S.top==S.base) return ERROR;

e=\*--top;//首先做—top运算，然后做e=\*top运算。

return OK;

}//Pop

 6.栈的链式存储结构

 和线性表的操作相类似，唯一不同之处在于只能从一个方向插入元素，同时从该方向删除元素。

二、栈的应用

1.数制转换

将十进制数N转换成d进制，例如要将十进制1348转换成8进制，计算过程如48页所示……，这样得到八进制的顺序为4、0、5、2，但是正确答案应该是2504，也就是将原来的顺序反一下。我们可以用数组来实现，取数据的时候从数组尾来取就可以了，同时也可以用栈结构实现，首先产生数据4，然后将它放到栈底，从栈底到栈顶的数据依次为4、0、5、2，这样出栈的顺序刚好为2504。程序如48页所示。

算法思想：将计算过程的d进制数逐位进栈，然后逐位出栈。

2．括号匹配的检验

假设在一个表达式中允许有两种括号，（）和[]，那么它们必须能够匹配，[（[][]）] ，这种匹配方法是合法的，而不能出现（[]]这种情况，括号的匹配检验可以用栈来实现，设置一个栈，这个栈只存放括号，碰到左括号，一律进栈，如果碰到右括号，从栈中弹出一个左括号，如果它们匹配，接着将下面的括号进栈，如果不匹配则报错。直到所有的括号输入完之后结束，如果栈刚好为空，则所有的括号都匹配。（自己写一下这个程序）

3、行编辑程序

行编辑程序的基本思想是用户输入的数据不直接送给程序，而送到缓冲，因为如果输入数据敲错，可以修改。

可以规定如果需要删除刚输入的一个字，则输入＃，而如果需要删入刚输入的行，则输入＠，计算机怎么才能记住最近输入的字符呢，可以用栈表示缓冲区，程序如50页所示（讲）。

4.迷宫求解不讲，有兴趣下去自己看。

作业：1.设有两个栈S1，S2，都采用顺序栈方式，并且共享一个存储区[0…..maxsize-1],为了充分利用空间，减少溢出的可能，可采用栈顶相向，迎面增长的存储方式，试设计进栈出栈操作。（36）。

栈的应用

1.表达式求值

表达式求值指计算机怎么识别出一个表达式，对它作出正确的解释，并计算出它的值。

例如：4＋2×3-10÷5,计算机怎么能够先算出2×3，然后计算出

4＋6……

首先，我们这里的算符有7种，＋、－、＊、/、（、）、＃。在表达式的开始和结束放一个＃，也就是说两个＃中间表示一个完整的表达式。

算符间的优先关系如表3－1所示。

算符的优先关系有三种<、=、> 。

其中加减算符优先权低于乘除，而乘除优先级低于加减。

算法的思想：首先这个算法假设表达式输入不会出现语法错误。设置两个工作栈：运算符栈OPTR和操作数栈OPND。操作数栈也存放表达式的运算结果。首先置操作数栈为空栈，置运算符栈的栈底为表达式的起始符#。依次读入表达式中的每个字符，若是操作数则进OPND栈；若是运算符，应进行判断：

1. 若读入的是“＃”，且运算符栈顶的符号也是“＃”时，则表达式处理结束。从操作数栈弹出表达式结果。

2.若读入的运算符的优先级大于运算符栈顶的优先级，则进运算符栈，继续扫描下一符号；否则从操作数栈顶弹出两个操作数，从运算符栈弹出一个运算符，生成运算指令，把结果送入操作数栈。继续处理刚才读入的符号。看53页程序

首先初始化两个栈，其中操作符栈的栈底元素为“＃”。

然后输入字符，

如果所输入的字符是“＃”且操作符栈底也是“＃”，则返回操作数栈的栈底，（最后一句程序）

如果输入的字符是数字，则该数字进操作数栈，继续输入字符。如果输入字符是操作符，则将该操作符和操作符栈的栈顶元素进行优先级比较，比较结果可有三种情况，＜，＝，＞；

对于第一种情况，栈顶元素优先权低，则接着输入。

对于第二种情况，则只能是两个括号相遇或两个井号相遇，比如计算（3＋7），“＋”优先级大于“）”，所以先进行3＋7的运算，然后得的结果是（11），所以只需将“（”弹出即可，同时接着输入字符。

第三种情况，栈顶元素优先权高，必须弹出两个操作数，同时弹出一个操作符，进行运算，将运算结果再进操作数栈。

讲程序。…………………………………………

2、栈与递归的实现

递归函数：一个通过直接或间接调用自己的函数叫做递归函数。例如：1求n!。

这个函数可以用递归函数表示，也可以用非递归函数表示用递归表示如下：

【算法3.3】

fact( int n) {

int result ;

if (n<0) printf(“error”); break;

else if(n==0) result=1;

else result=n\*fact(n-1);

return result;

}//其中fact函数直接调用了自己，所以这个函数是递归函数。

还有汉诺塔问题，汉诺塔问题假设有三个塔座x,y,z,见55页图3.5，（详细讲解规则）然后讲解程序，这是一个递归函数，在函数的执行过程中，不断地调用自己。

通常函数的调用由栈来实现。比如图3－6（C）中的函数调用，其中＊＊＊＊＊调用＊＊＊＊＊，＊＊＊调用　＊＊＊，函数调用过程中必须做三件事（1）保存现场，（2）分配存储区，（3）控制转移。（详细讲解C程序）。

队列

一、队列

1.队列的定义

队列是允许在一端进行插入而在另一端进行删除的线性表。允许插入的一端称为队尾，允许删除的一端称为队头。队列也称为先进先出表（FIFO）。

队列的抽象数据类型：ADT Queue

｛基本对象；

基本关系；

基本操作；｝

2.队列的基本操作

（1）入队 Enqueue(&Q, e) 在队列Q中插入元素e。

（2）出队Dequeue(&Q, &e) 若队列Q不空，则删除其队头元素。

（3）取队头元素Gethead(&Q, &e) 若队列Q不空，用e返回队头元素。

（4）判队列是否为空Queueempty(Q) 若队列Q为空，返回true，否则，返回false。

（5）遍历队列queueraverse(q,visit())对队列的每个数据元素调用函数visit().

(6)initquery();

(7)destroyquery();

(8)clearquery();

3.队列的链式表示和实现

链队列：用链表示的队列,每个结点的结构和单链表类似。

结点存储结构：

typedef struct QNode{

QelemType data ;

struct QNode \*next ;

}QNode, \*QueuePtr;

每个结点包括两部分，数据域和指针域。

 链表结构：

typedef struct{

QueuePtr \*front ;

QueuePtr \*rear ;

}每个链队列至少包括出众部分，队头指针和队尾指针。

front：队头指针，指向头结点

rear：队尾指针，指向尾结点。

空链队列：队头指针和队尾指针均指向头结点。

 4.队列操作在链式存储结构下的实现

【算法3.4】

（1）构造一个空队列initqueue(linkqueue &q)

{

q.front=q.rear= (QueuePtr) malloc(sizeof(qnode));

if(!q.front)exit(overflow);

q.front->next=null;

}

【算法3.5】

(2)销毁队列

｛

while(q.front)

{

q.rear=q.front->next;

free(q.front);;

q.front=q.rear;

}

return ok;

｝

【算法3.6】

（3）    入队Enqueue(&Q, e){

p=(QueuePtr) malloc(sizeof(node)); /\*生成结点\*/

p->data=e;

p->next=NUll;

Q.rear->next=p;

Q.rear=p}

【算法3.7】

（4）    出队 Dequeue(&Q, &e){

if (Q.front==Q.rear) return Error;

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

//删除队中最后一个元素的情况

free(p);}

二、循环队列—队列的顺序表示和实现

typedef struct{

QelemType \*base;

int front;//指向队头元素

 int rear; //指向队尾元素的下一个位置。

}SqQueue;

当插入一个元素时队尾声指针加1，当删除一个元素时，队头元素加1。

 二、循环队列的基本操作的算法描述

【算法3.8】

(1)status initqueue(sqqueue &q)

{//初始化循环链表

q.base=(elemtype\*)malloc(sizeof(elemtype));

if(!q.base)exit(overflow);

q.front=q.rear=0;

return ok;

}

【算法3.9】

(2) int queuelength(sqqueue q)

{//返回q的元素个数

return(q.rear-q.front+maxsize)%maxsize;

}//第一个maxsize为了防止出现负数。

【算法3.10】

(3) enqueue(sqqueue &q qelemtype e){

if((q.rear+1)%maxsize==q.front) return error;

q.base[q.rear]=e;

q.rear=(q.rear+1)%maxsize;

return ok;

}

【算法3.11】

status dequeue(sqqueue &q,qelemtype &e)

{//若队列不空，删除q的队头元素，并用e 返回其值

if(q.rear==q.front) return error;

e=q.base[q.front];

q.front=(q.front+1)%maxsize;

return ok;

}

作业：1.用标志位方式设计出在循环队列中进行插入和删除运算的算法。

　2.习题集。