**第3章 栈和队列**

**栈和队列是操作受限制的线性表。**

**§3.1栈（stack）**

**3.1.1栈的逻辑结构**

**1．栈的定义**

**一种后进先出（last in first out）或先进后出（first in last out)的结构称为栈（stack）的线性表。**

**日常生活中的栈例：**

**●一摞盘子的码放和取出：先放入的盘子后取出，后放入的盘子先取出；**

**●穿脱衣服：先穿的衣服后脱，后穿的衣服先脱;**

**●子弹夹上膛和射击：先装的子弹后出膛，后装的子弹先出膛等。**

**栈是限定仅在表尾进行插入和删除的线性表。表尾称为栈顶（top），表头称为栈底（bottom），不含任何元素的栈称为空栈。**

**出（退）栈 入（进）栈**

a1

栈顶top

……

an

栈底Bottom

**可以看出：栈底没有前驱，栈顶没有后继，栈中其它元素有一个直接的前驱和一个直接的后继，所以符合线性的规定。**

**2．栈的基本操作**

**●栈的空与不空情形**

**（1）栈的初始化，构造一个空栈，top=-1。**

**top🡪**

**（2）判断栈是否为空？为空返回1(true)，否则返回0(false)。**

**top🡪**

**（3）栈中有1个元素top=0,函数返回0(false)，表示不是空栈**

A

top

**（4）栈中有3个元素，top=2**

A

top

B

C

**（5）以下情形计作栈中有两个元素（元素“C”不计入在栈中），top=1**

A

top

B

C

**●入栈（进栈）：在栈顶压入一个新元素，该元素即为栈增加的元素，同时它成为栈顶元素，栈顶指针top++。**

**●出栈（退栈）：若栈非空，在栈顶删除一个栈顶元素。此时栈减少了一个元素，栈顶指针top--。**

**●取栈顶元素：取栈顶元素的值（与出栈不同），栈顶元素（还在栈中）不变，栈顶指针不变。**

**3.1.2 栈的顺序存储**

**本质上栈就是运算受限制的线性表。**

**存储结构也有两种：顺序存储结构（称为顺序栈）和链式存储结构（称为链栈）。**

**1．顺序栈的存储**

**定义：**

**#define MAXSIZE 100**

**typedef int ElemType;**

**typedef struct Node**

**{**

**ElemType data[MAXSIZE];**

**int top;**

**}SeqStack;**

**●从定义可以看出，顺序栈的元素存储用的是一个数组，数组是一片连续的存储单元。其中MAXSIZE是顺序栈的最大容量。**

**2．顺序栈的操作**

**（1）顺序栈的初始化**

**void SeqStackInit(SeqStack S)**

**{**

**S.top=-1; //表示为空栈**

**}**

**（2）判空栈**

**int SeqStackEmpty(SeqStack S)**

**{**

**if(S.top==-1)return 1;**

**//空栈返回1，非空返回0**

**else return 0;**

**}**

**（3）入栈**

**void SeqStackPush(SeqStack S,ElemType x)**

**{**

**if(S.top==MAXSIZE)**

**{printf(“栈上溢!\n”);exit(0);}**

**S.data[++S.top]=x; //++top,将x加入栈顶**

**}**

**（4）出栈**

**ElemType SeqStackPop(SeqStack S)**

**{**

**if(SeqStackEmpty(S))**

**{printf(“栈下溢！\n”);exit(0);}**

**else return S.data[S.top--];**

**}**

**（5）取栈顶元素**

**ElemType SeqStackGetTop(SeqStack S)**

**{**

**if(SeqStackEmpty(s)==1)**

**{printf(“栈空！\n);exit(0);}**

**else return S.data[S.top];**

**// 返回栈顶元素的值，栈指针不变**

**}**

**●对栈操作的说明：**

**●上溢：顺序栈满（条件是top=MAXSIZE-1）时不能再有元素入栈，否则出现溢出（称为上溢）。**

**●下溢：顺序栈空（条件是top=-1）时不能再有元素出栈，否则出现溢出（称为下溢）。**

**●顺序栈操作一个明显的不足在于存储空间的静态分配方式，存储分配的数量过小，操作容易溢出，过大又会造成浪费。**

**为解决这种情况，可以使用双向栈（操作在和处进行）。**

**如：**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **…** |  |  |  |  |  | **…** |  |  |

**链栈可以解决顺序栈明显不足的情况。**

**3.1.3 链栈的存储结构**

**1．链栈的存储**

**定义：**

**typedef int ElemType;**

**typedef struct node**

**{**

**ElemType data;**

**struct Node \*next;**

**}LStackNode,\*LinkedStack;**

**LStackNode \*top;**

**或LinkedStack top;**

**●从定义可以看出，链栈就是一个单链表。**

**●因为栈的入栈和出栈操作只能在栈顶进行，显然将链表的头部做栈顶是最方便的，不用像单链表那样为了方便附加一个头结点。**

3

top

13

40

**2．链栈的基本操作**

**（1）链栈初始化**

**void LinkedStackInit(LinkedStack top)**

**{**

**top=NULL;//链栈top=0,顺序栈top=-1**

**}**

**(2)判空栈**

**int LinkedStackEmpty(LinkedStack top)**

**{**

**if(top==NULL)return 1;**

**else return 0; //空栈返回1，非空返回0(NULL)**

**}**

**(3)入（进）栈**

**只能作插入头结点的操作**

**入栈前：**

66

top

2

88

**准备入栈：**

10

s

**入栈后：**

10

top

66

2

88

**void LinkedStackPush(LinkedStack top,**

**ElemType x)**

**{**

**LinkedStack s;**

**s=(LinkedStack)malloc(sizeof(LStackNode));**

**if(s==NULL){printf(“申请失败！”)；exit（0）；}**

**s🡪data=x;**

**s🡪next=top;**

**top=s;**

**}**

**(4)出（退）栈**

**只能作删除头结点的操作**

**出栈前：**

q

66

top

2

88

**出栈后：**

2

top

88

66

q

**ElemType LinkedStackPop(LinkedStack top)**

**{**

**LinkedStack q; ElemType y；**

**if(top==NULL){printf(“栈下溢！\n”);exit(0);}**

**y=top🡪data;**

**q=top;**

**top=top🡪next;**

**free(q);**

**return(y);**

**}**

**3.栈的使用-子程序的嵌套调用**

主程序A

.

.

call B

b:

.

.

end

b

子程序B

.

.

call C

c:

.

.

return

子程序C

.

.

.

.

.

.

return

b

c

b

**例3-1将十进制数转换为二进制：每次将余数入进栈中。**

**如：将13化为二进制。**

**入栈：**

**1%2=1**

**3%2=1**

**6%2=0**

**13%2=1**

**出栈：1101**

**第1步：13/2=6，余1进栈**

**第2步：6/2=3，余0进栈**

**第3步：3/2=1，余1进栈**

**第4步：1/2=0，余1进栈停止**

**例p3-1.c将十进制数化成二进制数（顺序栈的使用）**

**#define MAXSIZE 100**

**#include<stdio.h>**

**typedef int DataType;**

**typedef struct node**

**{**

**DataType data[MAXSIZE];**

**int top;**

**}SeqStack;//定义一个顺序栈**

**void InitStack(SeqStack \*s)//构造空栈**

**{**

**s🡪top=0;**

**}**

**int IsEmpty(SeqStack \*s)//判栈空**

**{**

**return s🡪top==0;//栈空时返回1，非空时为0**

**}**

**void Push(SeqStack \*s,DataType x)//入栈**

**{**

**if(s🡪top==MAXSIZE)printf("栈上溢出\n");**

**else s🡪data[++s🡪top]=x;**

**}**

**DataType Pop(SeqStack \*s)//出栈**

**{**

**if(IsEmpty(s))**

**{**

**printf("栈下溢出\n");**

**return (int)NULL;**

**}**

**else return s🡪data[s🡪top--];**

**//返回栈顶元素值，top--**

**}**

**void Conversion(int n)//十化二**

**{**

**int v;**

**SeqStack s;**

**InitStack(&s);**

**while(n!=0)**

**{**

**Push(&s,n%2);**

**n/=2;**

**}**

**while(!IsEmpty(&s))**

**{**

**v=Pop(&s);**

**printf("%d",v);**

**}**

**}**

**void main()**

**{**

**int n;**

**printf("输入一个十进制整数：");**

**scanf("%d",&n);**

**printf("%d---->",n);**

**Conversion(n);**

**printf("\n");**

**}**

**例p3-2.c 递归实现十翻二（栈的内部使用）**

**如：100**

**入栈：**

**1**

**3**

**6**

**12**

**25**

**50**

**100**

**出栈：1100100**

**void bin(int n)**

**{**

**if(n/2>0)bin(n/2);**

**printf(“%d”,n%2);**

**}**

**void main()**

**{**

**int n;**

**printf("输入一个十进制整数：");**

**scanf("%d",&n);**

**printf("%d---->",n);**

**bin(n);**

**printf("\n");**

**}**