第四章：栈

1.栈的相关概念介绍

1.顺序栈

```
#include <stdio.h>
#include <stblid.h>

#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define MAXSIZE 20 /* 存储空间初始分配量 */

typedef int Status;
typedef int SElemType; /* SElemType类型根据实际情况而
定，这里假设为int */

/* 顺序栈结构 */
typedef struct
{
        SElemType data[MAXSIZE];
        int top; /* 用于栈顶指针 */
}SqStack;

Status visit(SElemType c)
{
        printf("%d ",c);
        return OK;
}

/*  构造一个空栈S */
```

```
Status InitStack(SqStack *S)
{
        /* S.data=(SElemType
*)malloc(MAXSIZE*sizeof(SElemType)); */
        S->top=-1;
        return OK;
}

/* 把S置为空栈 */
Status ClearStack(SqStack *S)
{
        S->top=-1;
        return OK;
}

/* 若栈S为空栈，则返回TRUE，否则返回FALSE */
Status StackEmpty(SqStack S)
{
        if (S.top==-1)
                return TRUE;
        else
                return FALSE;
}

/* 返回S的元素个数，即栈的长度 */
int StackLength(SqStack S)
{
        return S.top+1;
}

/* 若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR
*/
Status GetTop(SqStack S,SElemType *e)
{
        if (S.top==-1)
                return ERROR;
```

```
        else
                *e=S.data[S.top];
        return OK;
}

/* 插入元素e为新的栈顶元素 */
Status Push(SqStack *S,SElemType e)
{
        if(S->top == MAXSIZE -1) /* 栈满 */
        {
                return ERROR;
        }
        S->top++;               /* 栈顶指针增加一 */
        S->data[S->top]=e;  /* 将新插入元素赋值给栈顶空间
*/
        return OK;
}

/* 若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返
回ERROR */
Status Pop(SqStack *S,SElemType *e)
{
        if(S->top==-1)
                return ERROR;
        *e=S->data[S->top]; /* 将要删除的栈顶元素赋值给e
*/
        S->top--;               /* 栈顶指针减一 */
        return OK;
}

/* 从栈底到栈顶依次对栈中每个元素显示 */
Status StackTraverse(SqStack S)
{
        int i;
        i=0;
        while(i<=S.top)
```

```
        {
                visit(S.data[i++]);
        }
        printf("\n");
        return OK;
}

int main()
{
        int j;
        SqStack s;
        int e;
        if(InitStack(&s)==OK)
                for(j=1;j<=10;j++)
                        Push(&s,j);
        printf("栈中元素依次为: ");
        StackTraverse(s);
        Pop(&s,&e);
        printf("弹出的栈顶元素 e=%d\n",e);
        printf("栈空否: %d(1:空 0:
否)\n",StackEmpty(s));
        GetTop(s,&e);
        printf("栈顶元素 e=%d 栈的长度
为%d\n",e,StackLength(s));
        ClearStack(&s);
        printf("清空栈后，栈空否: %d(1:空 0:
否)\n",StackEmpty(s));

        return 0;
}
```

2.两栈共享空间

```
#define OK 1
#define ERROR 0
```

```c
#define TRUE 1
#define FALSE 0
#define MAXSIZE 20 /* 存储空间初始分配量 */

typedef int Status;

typedef int SElemType; /* SElemType类型根据实际情况而定，这里假设为int */


/* 两栈共享空间结构 */
typedef struct
{
        SElemType data[MAXSIZE];
        int top1;   /* 栈1栈顶指针 */
        int top2;   /* 栈2栈顶指针 */
}SqDoubleStack;


Status visit(SElemType c)
{
        printf("%d ",c);
        return OK;
}

/*  构造一个空栈S */
Status InitStack(SqDoubleStack *S)
{
        S->top1=-1;
        S->top2=MAXSIZE;
        return OK;
}

/* 把S置为空栈 */
Status ClearStack(SqDoubleStack *S)
{
```

```
        S->top1=-1;
        S->top2=MAXSIZE;
        return OK;
}

/* 若栈S为空栈，则返回TRUE，否则返回FALSE */
Status StackEmpty(SqDoubleStack S)
{
        if (S.top1==-1 && S.top2==MAXSIZE)
                return TRUE;
        else
                return FALSE;
}

/* 返回S的元素个数，即栈的长度 */
int StackLength(SqDoubleStack S)
{
        return (S.top1+1)+(MAXSIZE-S.top2);
}

/* 插入元素e为新的栈顶元素 */
Status Push(SqDoubleStack *S,SElemType e,int
stackNumber)
{
        if (S->top1+1==S->top2) /* 栈已满，不能再push新
元素了 */
                return ERROR;
        if (stackNumber==1)         /* 栈1有元素进栈
*/
                S->data[++S->top1]=e; /* 若是栈1则先
top1+1后给数组元素赋值。 */
        else if (stackNumber==2)    /* 栈2有元素进栈
*/
                S->data[--S->top2]=e; /* 若是栈2则先
top2-1后给数组元素赋值。 */
        return OK;
```

```
}

/* 若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR */
Status Pop(SqDoubleStack *S,SElemType *e,int stackNumber)
{
	if (stackNumber==1)
	{
		if (S->top1==-1)
			return ERROR; /* 说明栈1已经是空栈，溢出 */
		*e=S->data[S->top1--]; /* 将栈1的栈顶元素出栈 */
	}
	else if (stackNumber==2)
	{
		if (S->top2==MAXSIZE)
			return ERROR; /* 说明栈2已经是空栈，溢出 */
		*e=S->data[S->top2++]; /* 将栈2的栈顶元素出栈 */
	}
	return OK;
}

Status StackTraverse(SqDoubleStack S)
{
	int i;
	i=0;
	while(i<=S.top1)
	{
		visit(S.data[i++]);
	}
	i=S.top2;
	while(i<MAXSIZE)
```

```
        {
                visit(S.data[i++]);
        }
        printf("\n");
        return OK;
}

int main()
{
        int j;
        SqDoubleStack s;
        int e;
        if(InitStack(&s)==OK)
        {
                for(j=1;j<=5;j++)
                        Push(&s,j,1);
                for(j=MAXSIZE;j>=MAXSIZE-2;j--)
                        Push(&s,j,2);
        }

        printf("栈中元素依次为: ");
        StackTraverse(s);

        printf("当前栈中元素有: %d \n",StackLength(s));

        Pop(&s,&e,2);
        printf("弹出的栈顶元素 e=%d\n",e);
        printf("栈空否: %d(1:空 0:
否)\n",StackEmpty(s));

        for(j=6;j<=MAXSIZE-2;j++)
                Push(&s,j,1);

        printf("栈中元素依次为: ");
        StackTraverse(s);
```

```
        printf("栈满否：%d(1:否 0:
满)\n",Push(&s,100,1));


        ClearStack(&s);
        printf("清空栈后，栈空否：%d(1:空 0:
否)\n",StackEmpty(s));

        return 0;
}
```

3.链栈

```
#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define MAXSIZE 20 /* 存储空间初始分配量 */

typedef int Status;
typedef int SElemType; /* SElemType类型根据实际情况而
定，这里假设为int */


/* 链栈结构 */
typedef struct StackNode
{
        SElemType data;
        struct StackNode *next;
}StackNode,*LinkStackPtr;


typedef struct
{
```

```
        LinkStackPtr top;
        int count;
}LinkStack;

Status visit(SElemType c)
{
        printf("%d ",c);
        return OK;
}

/*  构造一个空栈S */
Status InitStack(LinkStack *S)
{
        S->top =
(LinkStackPtr)malloc(sizeof(StackNode));
        if(!S->top)
                return ERROR;
        S->top=NULL;
        S->count=0;
        return OK;
}

/* 把S置为空栈 */
Status ClearStack(LinkStack *S)
{
        LinkStackPtr p,q;
        p=S->top;
        while(p)
        {
                q=p;
                p=p->next;
                free(q);
        }
        S->count=0;
        return OK;
}
```

```
/* 若栈S为空栈，则返回TRUE，否则返回FALSE */
Status StackEmpty(LinkStack S)
{
        if (S.count==0)
                return TRUE;
        else
                return FALSE;
}

/* 返回S的元素个数，即栈的长度 */
int StackLength(LinkStack S)
{
        return S.count;
}

/* 若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR
*/
Status GetTop(LinkStack S,SElemType *e)
{
        if (S.top==NULL)
                return ERROR;
        else
                *e=S.top->data;
        return OK;
}

/* 插入元素e为新的栈顶元素 */
Status Push(LinkStack *S,SElemType e)
{
        LinkStackPtr s=
(LinkStackPtr)malloc(sizeof(StackNode));
        s->data=e;
        s->next=S->top; /* 把当前的栈顶元素赋值给新结点的
直接后继，见图中① */
```

```
        S->top=s;          /* 将新的结点s赋值给栈顶指针，见图中② */
        S->count++;
        return OK;
}

/* 若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR */
Status Pop(LinkStack *S,SElemType *e)
{
        LinkStackPtr p;
        if(StackEmpty(*S))
                return ERROR;
        *e=S->top->data;
        p=S->top;                    /* 将栈顶结点赋值给p，见图中③ */
        S->top=S->top->next;     /* 使得栈顶指针下移一位，指向后一结点，见图中④ */
        free(p);                     /* 释放结点p */

        S->count--;
        return OK;
}

Status StackTraverse(LinkStack S)
{
        LinkStackPtr p;
        p=S.top;
        while(p)
        {
                visit(p->data);
                p=p->next;
        }
        printf("\n");
        return OK;
}
```

```
int main()
{
        int j;
        LinkStack s;
        int e;
        if(InitStack(&s)==OK)
                for(j=1;j<=10;j++)
                        Push(&s,j);
        printf("栈中元素依次为: ");
        StackTraverse(s);
        Pop(&s,&e);
        printf("弹出的栈顶元素 e=%d\n",e);
        printf("栈空否: %d(1:空 0:
否)\n",StackEmpty(s));
        GetTop(s,&e);
        printf("栈顶元素 e=%d 栈的长度
为%d\n",e,StackLength(s));
        ClearStack(&s);
        printf("清空栈后，栈空否: %d(1:空 0:
否)\n",StackEmpty(s));
        return 0;
}
```

4.斐波那契函数

```
int Fbi(int i)  /* 斐波那契的递归函数 */
{
    if( i < 2 )
        return i == 0 ? 0 : 1;
    return Fbi(i - 1) + Fbi(i - 2);  /* 这里Fbi就是函
数自己，等于在调用自己 */
}
```

```c
int main()
{
    int i;
    int a[40];
    printf("迭代显示斐波那契数列: \n");
    a[0]=0;
    a[1]=1;
    printf("%d ",a[0]);
    printf("%d ",a[1]);
    for(i = 2;i < 40;i++)
    {
        a[i] = a[i-1] + a[i-2];
        printf("%d ",a[i]);
    }
    printf("\n");

    printf("递归显示斐波那契数列: \n");
    for(i = 0;i < 40;i++)
        printf("%d ", Fbi(i));
    return 0;
}
```