# 21线性表16

# 双向链表



这货我们地球人把他称为火车，有了它，全世界每年春季最大规模的物种迁移才会发生在中国！

大家都知道，任何事物出现的初期都显得有些不完善。例如我们的火车刚发明的时候是只有一个“头”的，所以如果它走的线路是如下：

A->B->C->D->E->F->G->H->I->J->K->L->A

假设这会儿火车正停在K处呢，要他送一批货到J处，那么它将走的路线是：

K->L->A->B->C->D->E->F->G->H->I->J

嗯，所以后来我们的火车就都有两个头了。看完这个例子，大家就明白双向链表的必要性了吧。

## 双向链表结点结构

typedef struct DualNode

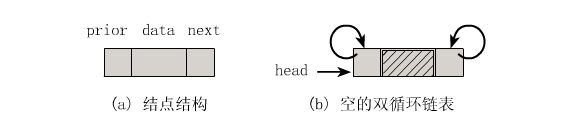
{

ElemType data;

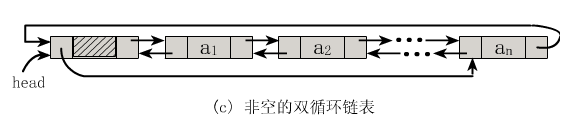
struct DualNode \*prior; //前驱结点

struct DualNode \*next; //后继结点

} DualNode, \*DuLinkList;



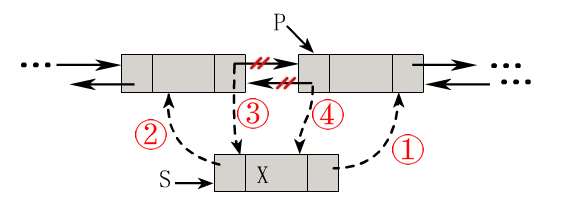
既然单链表可以有循环链表，那么双向链表当然也可以有。



在这里小甲鱼问大家一个问题：由于这是双向链表，那么对于链表中的某一个结点p，它的后继结点的前驱结点是什么？

## 双向链表的插入操作

插入操作其实并不复杂，不过顺序很重要，千万不能写反了。



代码实现：

s->next = p;

s->prior = p->prior;

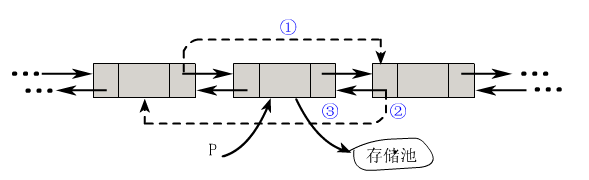
p->prior->next = s;

p->prior = s;

关键在于交换的过程中不要出现矛盾，例如第四步先被执行了，那么p->prior就会提前变成s，使得插入的工作出错。严重性打个比方就是打电话给老婆的时候不小心叫成小三的名字！

## 双向链表的删除操作

如果刚才的插入操作理解了，那么再来理解接下来的删除操作就容易多了。



代码实现：

p->prior->next = p->next;

p->next->prior = p->prior;

free(p);

最后总结一下，双向链表相对于单链表来说，是要更复杂一点，每个结点多了一个prior指针，对于插入和删除操作的顺序大家要格外小心。

不过，双向链表可以有效提高算法的时间性能，说白了就是用空间来换取时间。

力争做一只逆流而上的小甲鱼！

# 22线性表17

## 双向循环链表实践

课堂演示题目：

要求实现用户输入一个数使得26个字母的排列发生变化，例如用户输入3，输出结果：

DEFGHIJKLMNOPQRSTUVWXYZABC

同时需要支持负数，例如用户输入-3，输出结果：

XYZABCDEFGHIJKLMNOPQRSTUVW

小甲鱼带着大家打代码时间：

Caser.c

#include<stdio.h>

#include<stdlib.h>

#define OK 1

#define ERROR 0;

typedef char ElemType;

typedef int Status;

typedef struct DualNode

{

ElemType data;

struct DualNode \*prior;

struct DualNode \*next;

}DualNode,\*DuLinkList;

Status InitList(DuLinkList \*L)

{

DualNode \*p,\*q;

int i;

\*L=(DuLinkList)malloc(sizeof(DualNode));

if(!(\*L))

{

return ERROR;

}

(\*L)->next=(\*L)->prior=NULL;

p=(\*L);

for(i=0;i<26;i++)

{

q=(DualNode \*)malloc(sizeof(DualNode));

if(!q)

{

return ERROR;

}

q->data='A'+i;

q->prior=p;

q->next=p->next;

p->next=q;

p=q;

}

p->next=(\*L)->next;

(\*L)->next->prior=p;

return OK;

}

void caser(DuLinkList \*L,int i)

{

if(i>0)

{

do

{

(\*L)=(\*L)->next;

}while(--i);

}

if(i<0)

{

i=i-1;

(\*L)=(\*L)->next;

do

{

(\*L)=(\*L)->prior;

}while(++i);

}

}

int main()

{

DuLinkList L;

int i,n;

InitList(&L);

printf("请输入一个整数:\n");

scanf("%d",&n);

printf("\n");

caser(&L,n);

for(i=0;i<26;i++)

{

L=L->next;

printf("%c",L->data);

}

printf("\n");

return 0;

}

## 课后作业

Vigenere（维吉尼亚）加密：

当输入明文，自动生成随机密匙匹配明文中每个字母并移位加密。

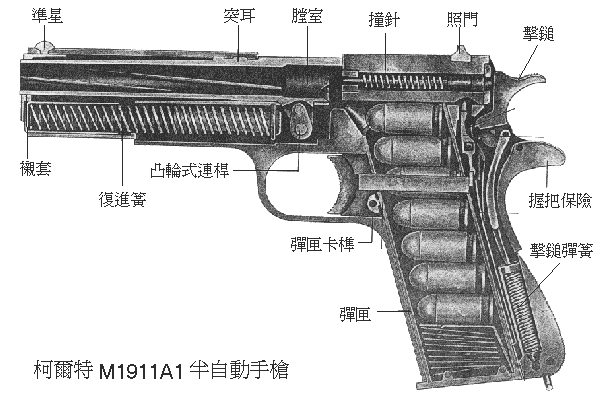
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 明文 | I | L | O | V | E | F | I | S | H | C |
| 随机密匙 | 3 | 15 | 23 | 2 | 52 | 1 | 33 | 49 | 13 | 19 |
| 密文 | L | A | L | X | E | G | P | P | U | V |

建议：当然你的随机密匙生成后不能丢掉，丢掉了就很难把明文还原来了，建议把随机密匙和密文加密存储在一起。

# 23栈和队列1

## 栈和队列

线性表的特例



## 栈的定义

栈是一种重要的线性结构，可以这样讲，栈是前面讲过的线性表的一种具体形式。

就像我们刚才的例子，栈这种后进先出的数据结构应用是非常广泛的。在生活中，例如我们的浏览器，每点击一次“后退”都是退回到最近的一次浏览网页。

例如我们Word，Photoshop等的“撤销”功能也是如此。再例如我们C语言的函数，也是利用栈的基本原理实现的。

官方定义：栈（Stack）是一个后进先出（Last in first out,LIFO）的线性表，它要求只在表尾进行删除和插入操作。

小甲鱼定义：所谓的栈，其实也就是一个特殊的线性表（顺序表、链表），但是它在操作上有一些特殊的要求和限制：

栈的元素必须“后进先出”。

栈的操作只能在这个线性表的表尾进行。

注：对于栈来说，这个表尾称为栈的栈顶（top），相应的表头称为栈底（bottom）。

## 栈的插入和删除操作

栈的插入操作（Push），叫做进栈，也称为压栈，入栈。类似子弹放入弹夹的动作。

栈的删除操作（Pop），叫做出栈，也称为弹栈。如同弹夹中的子弹出夹。

请看动画演示：

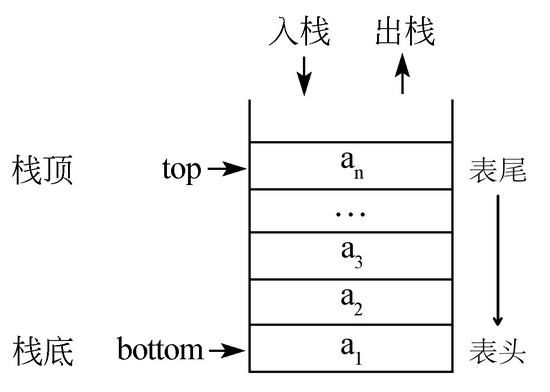
## 栈的顺序存储结构

因为栈的本质是一个线性表，线性表有两种存储形式，那么栈也有分为栈的顺序存储结构和栈的链式存储结构。

最开始栈中不含有任何数据，叫做空栈，此时栈顶就是栈底。然后数据从栈顶进入，栈顶栈底分离，整个栈的当前容量变大。数据出栈时从栈顶弹出，栈顶下移，整个栈的当前容量变小。

妈呀，说啥呢？

请结合下图发挥想象力理解并记住：



typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

这里定义了一个顺序存储的栈，它包含了三个元素：base,top,stackSize。其中base是指向栈底的指针变量，top是指向栈顶的指针变量，stackSize指示栈的当前可使用的最大容量。

## 创建一个栈

#define STACK\_INIT\_SIZE 100

initStack(sqStack \*s)

{

s->base = (ElemType \*)malloc( STACK\_INIT\_SIZE \* sizeof(ElemType) );

if( !s->base )

exit(0);

s->top = s->base; // 最开始，栈顶就是栈底

s->stackSize = STACK\_INIT\_SIZE;

}

入栈操作

* 入栈操作又叫压栈操作，就是向栈中存放数据。
* 入栈操作要在栈顶进行，每次向栈中压入一个数据，top指针就要+1，直到栈满为止。
* 请看代码挺小甲鱼详细解释：Push.c

出栈操作

* 出栈操作就是在栈顶取出数据，栈顶指针随之下移的操作。
* 每当从栈内弹出一个数据，栈的当前容量就-1。
* 代码清单：

Pop(sqStack \*s, ElemType \*e)

{

if( s->top == s->base ) // 栈已空空是也

return;

\*e = \*--(s->top);

}

v

# 24栈和队列2

## 疑问解释

上节课我们讲解栈的结构，我们是这样声明的：

typedef int ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

有些朋友提出了疑问：怎么没有data元素存放数据？怎么会有两个ElemType元素？

其实如果小甲鱼按照套路出牌，我们完全可以这样子声明：

typedef int ElemType;

typedef struct

{

ElemType data[MAXSIZE];

int top; // 用于标注栈顶的位置

int stackSize;

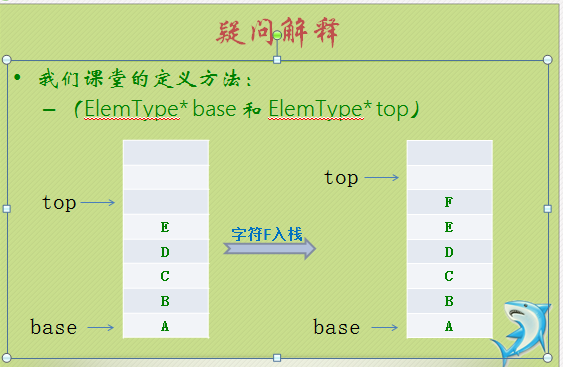
}

但是呢，小甲鱼要考虑到大家希望学习到的是“不变应万变”的能力，所以，这次有些朋友没反应过来^\_^

好吧，如果理解的朋友接下来的两分钟视频可以撸过，小甲鱼画两个图给可能还不理解的朋友解释下，因为比较重要，所以大家不要嫌弃我罗嗦哈，就两分钟T\_T

我们课堂的定义方法：

（ElemType\* base 和 ElemType\* top）





## 栈的其他操作

上节课我们介绍了柯尔特半自动手枪，介绍了栈相关的基本操作：入栈和出栈。

除了以上介绍外，对栈还有一些其他的操作，例如清空一个栈，销毁一个栈，计算栈的当前容量等。

我们这节课的安排就这些！

哦还有，利用栈的原理，给大家讲解一道例题。

题目：利用栈的特点，将用户输入的二进制数转换为十进制数。（大家可以先思考）

## 清空一个栈

所谓清空一个栈，就是将栈中的元素全部作废，但栈本身物理空间并不发生改变（不是销毁）。

因此我们只要将s->top的内容赋值为s->base即可，这样s->base等于s->top，也就表明这个栈是空的了。这个原理跟高级格式化只是但单纯地清空文件列表而没有覆盖硬盘的原理是一样的。

代码清单：

ClearStack(sqStack \*s){

s->top = s->base;

}

## 销毁一个栈

* 销毁一个栈与清空一个栈不同，销毁一个栈是要释放掉该栈所占据的物理内存空间，因此不要把销毁一个栈与清空一个栈这两种操作混淆。
* 代码清单：
* XXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXOOXXO

DestroyStack(sqStack \*s){

int i, len;

len = s->stackSize;

for( i=0; i < len; i++ ){

free( s->base );

s->base++;

}

s->base = s->top = NULL;

s->stackSize = 0;

}

## 计算栈的当前容量

* 计算栈的当前容量也就是计算栈中元素的个数，因此只要返回s.top-s.base即可。
* 注意，栈的最大容量是指该栈占据内存空间的大小，其值是s.stackSize，它与栈的当前容量不是一个概念哦。
* 代码清单：

int StackLen(sqStack s)

{

return(s.top – s.base); // 初学者需要重点讲解

}

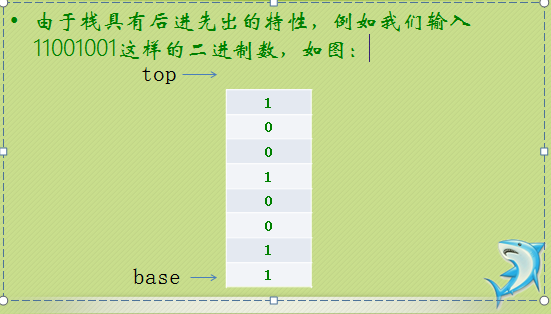
## 实例分析

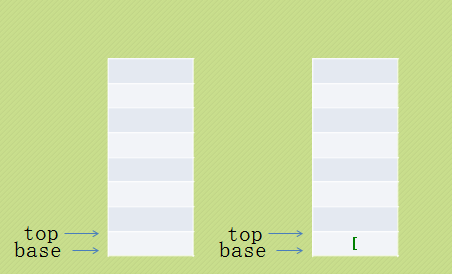
题目：利用栈的数据结构特点，将二进制转换为十进制数。

分析：地球人都知道，二进制数是计算机数据的存储形式，它是由一串0和1组成的，每个二进制数转换成相应的十进制数方法如下：

(XnXn-1……X3X2X1)2 = X1\*2^0+X2\*2^1+…+Xn\*2^(n-1)

一个二进制数要转换为相应的十进制数，就是从最低位起用每一位去乘以对应位的积，也就是说用第n位去乘以2^(n-1)，然后全部加起来。





# 25进制转换

实例分析

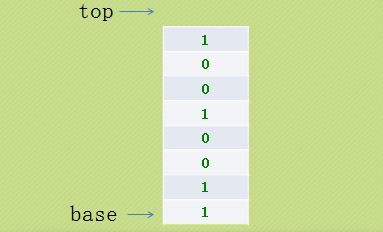
题目：利用栈的数据结构特点，将二进制转换为十进制数。

分析：地球人都知道，二进制数是计算机数据的存储形式，它是由一串0和1组成的，每个二进制数转换成相应的十进制数方法如下：

(XnXn-1……X3X2X1)2 = X1\*2^0+X2\*2^1+…+Xn\*2^(n-1)

一个二进制数要转换为相应的十进制数，就是从最低位起用每一位去乘以对应位的积，也就是说用第n位去乘以2^(n-1)，然后全部加起来。

由于栈具有后进先出的特性，例如我们输入11001001这样的二进制数，如图：



Bin2Dec.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define STACK\_INIT\_SIZE 20

#define STACKINCREMENT 10

typedef char ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

void InitStack(sqStack \*s)

{

s->base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

s->top = s->base;

s->stackSize = STACK\_INIT\_SIZE;

}

void Push(sqStack \*s, ElemType e)

{

if( s->top - s->base >= s->stackSize )

{

s->base = (ElemType \*)realloc(s->base, (s->stackSize + STACKINCREMENT) \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

}

\*(s->top) = e;

s->top++;

}

void Pop(sqStack \*s, ElemType \*e)

{

if( s->top == s->base )

{

return;

}

\*e = \*--(s->top);//指针访问属性使用箭头

}

int StackLen(sqStack s)

{

return (s.top - s.base);//元素差，结构访问属性用点

}

int main()

{

ElemType c;

sqStack s;

int len, i, sum = 0;

InitStack(&s);

printf("请输入二进制数，输入#符号表示结束！\n");

scanf("%c", &c);

while( c != '#' )

{

Push(&s, c);

scanf("%c", &c);

}

getchar(); // 把'\n'从缓冲区去掉

len = StackLen(s);

printf("栈的当前容量是: %d\n", len);

for( i=0; i < len; i++ )

{

Pop(&s, &c);

sum = sum + (c-48) \* pow(2, i);

}

printf("转化为十进制数是: %d\n", sum);

return 0;

}

从二进制到八进制

* 地球人都知道，我们学习编程常常会接触到不同进制的数，而最多的就是二进制、八进制、十进制、十六进制。
* 鱼C人还知道，二进制是计算机唯一认识的，十进制是人们通常使用的。
* 那么，有没有谁知道八进制和十六进制呢？为什么没有三进制、四进制、五六七进制呢？
* (⊙v⊙)嗯，我们仔细观察二进制跟十六进制的对应关系：

Bin2Oct.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* 二进制/八进制转换器 \*\*/

/\*\* By www.fishc.com 小甲鱼 \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define STACK\_INIT\_SIZE 20

#define STACKINCREMENT 10

typedef char ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

// 函数功能：初始化栈

// 参数\*s：栈的地址

void InitStack(sqStack \*s)

{

s->base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

s->top = s->base;

s->stackSize = STACK\_INIT\_SIZE;

}

// 函数功能：入栈操作

// 参数\*s：栈的地址

// 参数e：待压入栈的元素

void Push(sqStack \*s, ElemType e)

{

if( s->top - s->base >= s->stackSize )

{

s->base = (ElemType \*)realloc(s->base, (s->stackSize + STACKINCREMENT) \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

s->top = s->base + s->stackSize;

s->stackSize = s->stackSize + STACKINCREMENT;

}

\*(s->top) = e;

s->top++;

}

// 函数功能：弹栈操作

// 参数\*s：栈的地址

// 参数e：存放从栈里弹出的数据

void Pop(sqStack \*s, ElemType \*e)

{

if( s->top == s->base )

{

return;

}

\*e = \*--(s->top);

}

// 函数功能：计算栈s的当前长度

// 参数s：栈

int StackLen(sqStack s)

{

return (s.top - s.base);

}

int main()

{

ElemType c;

sqStack s1;

sqStack s2;

int len, i, j, sum = 0;

InitStack(&s1); // 初始化栈s1，用来存放二进制输入

printf("请输入二进制数，输入‘#’号表示结束！\n\n");

scanf("%c", &c);

while( c != '#' )

{

if( c=='0' || c=='1' ) // 检查输入是否二进制

Push(&s1, c);

scanf("%c", &c);

}

getchar(); // 把'\n'从缓冲区去掉

len = StackLen(s1);

InitStack(&s2); // 初始化栈s2，用来存放转换的八进制

for( i=0; i < len; i+=3 )

{

for( j=0; j < 3; j++ )

{

Pop( &s1, &c ); // 取出栈顶元素

sum = sum + (c-48) \* pow(2, j);

if( s1.base == s1.top )

{

break;

}

}

Push( &s2, sum+48 );

sum = 0;

}

printf("\n转化为八进制数是: ");

while( s2.base != s2.top )

{

Pop( &s2, &c );

printf("%c", c);

}

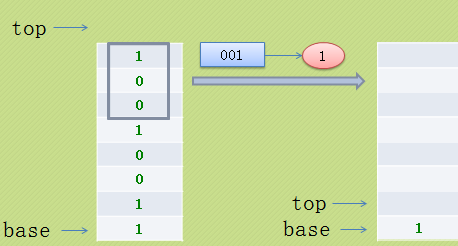
printf("(O)\n");

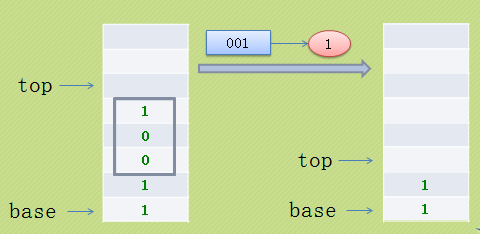
return 0;

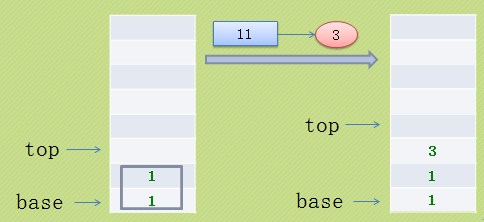
}

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 10 | 2 |
| 11 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

* 可见一个字节（8bit）刚好用两个十六进制数可以表示完整，也大大的节省了显示空间。
* 那八进制呢？因为早期的计算机系统都是三的倍数，所以用八进制比较方便。
* 我们发现了，在进行二进制到八进制的转换时，要将二进制数的每三位转换成一个八进制数来表示，然后按顺序输出即可。
* 对于文字描述不好理解的概念，我们就只能：
* No pic you say a j8!







Bin2Hex.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* 二进制/十六进制转换器 \*\*/

/\*\* By www.fishc.com 小甲鱼 \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define STACK\_INIT\_SIZE 20

#define STACKINCREMENT 10

typedef char ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

// 函数功能：初始化栈

// 参数\*s：栈的地址

void InitStack(sqStack \*s)

{

s->base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

s->top = s->base;

s->stackSize = STACK\_INIT\_SIZE;

}

// 函数功能：入栈操作

// 参数\*s：栈的地址

// 参数e：待压入栈的元素

void Push(sqStack \*s, ElemType e)

{

if( s->top - s->base >= s->stackSize )

{

s->base = (ElemType \*)realloc(s->base, (s->stackSize + STACKINCREMENT) \* sizeof(ElemType));

if( !s->base )

{

exit(0);

}

s->top = s->base + s->stackSize;

s->stackSize = s->stackSize + STACKINCREMENT;

}

\*(s->top) = e;

s->top++;

}

// 函数功能：弹栈操作

// 参数\*s：栈的地址

// 参数e：存放从栈里弹出的数据

void Pop(sqStack \*s, ElemType \*e)

{

if( s->top == s->base )

{

return;

}

\*e = \*--(s->top);

}

// 函数功能：计算栈s的当前长度

// 参数s：栈

int StackLen(sqStack s)

{

return (s.top - s.base);

}

int main()

{

ElemType c;

sqStack s1;

sqStack s2;

int len, i, j, sum = 0;

InitStack(&s1); // 初始化栈s1，用来存放二进制输入

printf("请输入二进制数，输入‘#’号表示结束！\n\n");

scanf("%c", &c);

while( c != '#' )

{

if( c=='0' || c=='1' ) // 检查输入是否二进制

Push(&s1, c);

scanf("%c", &c);

}

getchar(); // 把'\n'从缓冲区去掉

len = StackLen(s1);

InitStack(&s2); // 初始化栈s2，用来存放转换的八进制

for( i=0; i < len; i+=4 )

{

for( j=0; j < 4; j++ )

{

Pop( &s1, &c ); // 取出栈顶元素

sum = sum + (c-48) \* pow(2, j);

if( s1.base == s1.top )

{

break;

}

}

switch( sum )

{

case 10: sum = 'A'; break;

case 11: sum = 'B'; break;

case 12: sum = 'C'; break;

case 13: sum = 'D'; break;

case 14: sum = 'E'; break;

case 15: sum = 'F'; break;

default: sum += 48;

}

Push( &s2, sum );

sum = 0;

}

printf("\n转化为十六进制数是: ");

while( s2.base != s2.top )

{

Pop( &s2, &c );

printf("%c", c);

}

printf("(H)\n");

return 0;

}