**1.表ADT**

**数组形式**：

LIST MakeEmpty（LIST PtrL）

{

List PtrL;

PtrL=(List)malloc(sizeof(struct LNode));

PtrL->last=-1;

return PtrL;

}

ElementType FindKth(LIST PtrL,int K)

void Insert(Elementypre x ,int i,LIST PtrL)

{

if(PtrL->last>=MAXSIZE-1){

printf("表满");

if(i<1 || i>=PtrL->last+1){

printf("位置不合法");

return -1;

}

for(int j=Ptrl->last;j>=i;j--){

Ptrl->Data[j+1]=Ptrl->Data[j];

Ptrl->data[i-1]=x;

Ptrl->last++;

}

}

void Delete(int i,LIST PtrL)

{

if(i<1 || i>=PtrL->last+1){

printf("不存在第%d个元素',i);

return -1;

}

for(int j=i;j<=Ptrl->last;j--)

{

Ptrl->Data[j-1]=Ptrl->Data[j];

Ptrl->last--;

}

}

int Find(ElementType x,LIST L )

{

int i=0;

while(i<=L->last && L->Data[i]!=x)

{

i++

}

if(i==L->last)

return -1;

else

return i

}

int Length(LIST L)

数组的创建：

typedef struct LNode \*List；

struct LNode{

ElementType Data[MAXSIZE];

int last;

}PtrL;

**链式存储实现**

typedef struct LNode \*List；

struct LNode{

ElementType Data;

List next;

};

struct Lnode L;

LIST PtrL;

求表长

int Length(List PtrL)

{

List p=Ptrl;

int j=0;

while(p){

p-=p>next;

j=j+1;

}

return j;

}

查找(按序号查找）

ElementType FindKth(LIST L,int K）{

int j=1;

List p=Ptrl;

while(p!=NULL&&j<K){

p=p->nect;

j++;

}

If(j=k) return p;

else return NULL;

按值查找

int Find(ElementType x,LIST L ){

int j=0;

List p=Ptrl;

while(p!=NULL&&p->data==x){

p=p->nect;

}

return p;

void Insert(Elementypre x ,int i,LIST PtrL){

List P,S;

if(i==1){

S=(list)malloc(sizeof(struct Lnode));

S->Data=x;

S->next=PtrL;

return S;

}

P=findKth(i-1,PtrL);

If(P==NULL)

printf(“参数i错”);

return NULL;

else{

S=(list)malloc(sizeof(struct Lnode));

S->Data=x;

S->next=P->next;

P->next=S;

return PtrL;

}

删除操作

void Delete(int i,LIST PtrL){

LIST P,S;

if(i==1){

S=PtrL;

if(PtrL->next==NULL)

PtrL =PtrL->next ;

else return NULL;

free(S);

return PtrL;

}

S=Findth(i-1);

if(S==NULL);

printf(“参数i错误”);

return NULL;

else if(S->next==NULL){

printf(“参数i错误”);

return NULL;

}

else{

P=S->next;

S->next=P-next;

free(P);

return PtrL;

}

**2.堆栈**

**操作集**

Stack CreateStack(int MaxSize)

int IsFull(Stack S，int MaxSize)

void Push(Stack S,int MaxSize)

int IsEmpty(Stack S)

ElementType Pop(Stack S)

定义

#define MaxSize

typedef struct SNode \*Stack

struct SNode{

ElementType Data[MaxSize];

int Top;

}:

**数组实现**

入栈

void Push(Stack PtrS,ElementType item)

{

if(PtrS->Top==MaxSize-1){

printf("堆栈满");return;

}else{

PtrS->Data[++(PtrS->Top)]=item;

}

}

出栈

ElementType Pop(Stack PtrS){

if(PtrS->Top==-1){

printf("堆栈空”);

return ERROR;

}else

return(PtrS->Data[(PtrS->top)--];

**链式表示**

typedef struct SNode \*Stack;

struct SNode{

ElementType Data;

struct SNode \*Next;

};

Stack CreateStack()

{

Stack S;

S=(Stack)malloc(sizeof(struct SNode));

S->Next=NULL;

return S;

}

int IsEmpty(Stack S){

return(S->Next==NULL);

}

void Push(ElementType item,Stack S)

{

struct SNode \*TmpCell;

Tmpcell=(struct SSNode \*)malloc(sizeof(struct SSNode));

Tmpcell->Element =item;

Tmpcell->Next=S->Next;

S->Next=TmpCell;

}

ElementType Pop(Stack S){ %先判别是否为空

struct SNode \*Firstcell;

ElementType TopElem;

if(IsEmpty(S)){

printf("堆栈满");return NULL;

}

Firstcell=S->next;

S->Next=Firstcell->Next;

TopElem = Firstcell->Element;

free(Firstcell);

return(TopElem);

}