链表操作

typedef struct L{

int a;

struct L \*next;

}\*linklist,Lnode;

void InitL(linklist &L)

{

linklist p,pnew;

int i;

printf("请输入链表中%d个元素:\n",MAXSIZE);

L=(linklist)malloc(sizeof(Lnode));

p=L;

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

{

pnew=(linklist)malloc(sizeof(Lnode));

scanf("%d",&pnew->a);

p->next=pnew; p=pnew;}

p->next=NULL;}

void output(linklist L)

{

linklist p;

printf("链表中的元素为:\n");

p=L->next;

while(p)

{

printf("%5d",p->a);

p=p->next;}

printf("\n");}

Int search(int a,linklist L)

{

linklist p;

p=L->next;

printf("请输入要查找的元素：\n");

scanf("%d",&a);

while(p)

{

if(p->a==a)

{ printf("yes\n");

return 1;}

p=p->next;}

printf("no\n");

return 0;}

int Insert(linklist &L)

{

int i,j,k;

linklist p,pnew,s;

printf("请输入要插入的数:\n");

scanf("%d",&i);

printf("要插在第几个数前面\n");

scanf("%d",&j);

if(j>MAXSIZE)

printf("error\n");

p=L;

for(k=0;k<j-1;k++)

p=p->next;

s=p->next;

pnew=(linklist)malloc(sizeof(Lnode));

pnew->a=i;

p->next=pnew; p=pnew;

p->next=s;

return 0;}

int deleteL(linklist L)

{

int i,k,n;

linklist p,s;

printf("要删除第几个数?\n");

scanf("%d",&i);

p=L;

for(k=1;k<i;k++)

p=p->next;

s=p->next;

n=s->a;

p->next=s->next;

return n;}

void merge(linklist &a, linklist &b)

{

linklist p, pa, pb, prea;

if(a == NULL || b == NULL) return;

p = a;

prea = a;

pa = a->next;

pb = b->next;

while(pa && pb)

{

if(pa == pb)

{ pb = NULL;

break;}

else if(pa->a > pb->a)

{

prea->next = pb;

pb = pb->next;

prea = prea->next;

prea->next = pa; }

else

{

if(pa->next == pb->next)

{

pb->next = NULL;

}

prea = pa;

pa = pa->next;}}

prea->next = pa ? pa : pb;

return ;}

括号匹配

#define SIZE 100

#define STACKINCREMENT 10

typedef char ElemType;

typedef struct{

ElemType \* base;

int top;

int stacksize;

}SqStack;

void InitStack(SqStack &S){

S.base=(char\*)malloc(SIZE \* sizeof(char));

if(!S.base) exit(0);

S.top = 0 ;

S.stacksize=SIZE;

}

void Push(SqStack &S,char e){

if(S.top>=S.stacksize)

return ;

S.base=(char \*)malloc((S.stacksize +STACKINCREMENT) \* sizeof (char));

S.base[S.top]=e;

if(!S.base) exit(0);

S.top++;

}

int Pop(SqStack &S,char &e){

if(S.top ==0 )return 0;

e=S.base[S.top];

return 1;

}

int StackEmpty(SqStack &S){

if(S.top==0)

return 1;

return 0;

}

int AusBracket(char \*str){

SqStack S;

char \*p;

char ch;

InitStack(S);

p=str;

while(\*p!='\0'){

if(\*p=='('||\*p=='['||\*p=='{')

Push(S,\*p);

else if(\*p==')'||\*p==']'||\*p=='}'){

if(StackEmpty(S))

return 0;

Pop(S,ch);

if(\*p==')' && ch!='(')

return 1;

if(\*p==']' && ch!='[')

return 1;

if(\*p=='}' && ch!='{')

return 1;

}

p++;

}

if(!StackEmpty(S))

return 0;

return 1;

}

int main(){

char \*c;

c=(char\*)malloc(SIZE\*sizeof(char));

printf("请输入你要匹配的括号类型:\n");

scanf("%s",c);

AusBracket(c);

if(AusBracket(c))

printf("成功！\n");

else

printf("错误。\n");

return 0;

}

二叉树遍历

#define size 100

#define addsize 10

typedef struct B{

char data;

struct B \*lchild;

struct B \*rchild;

}BiTNode,\*BiTree;

typedef struct{

BiTree \*base;

int top;

int stacksize;

}St;

typedef struct{

char base;

int front;

int tail;

}Queue;

void InitStack(St &S){

S.base=(BiTree \*)malloc(size\*sizeof(BiTree));

S.top=0;

S.stacksize=size;

}

int StackEmpty(St S){

if(S.top==0)

return 1;

else

return 0;

}

void Push(St &S,BiTree e){

if(S.top>=S.stacksize){

S.base=(BiTree \*)realloc(S.base,(size+addsize)\*sizeof(BiTree));

S.stacksize+=addsize;

}

S.base[S.top++]=e;

}

void Pop(St &S,BiTree &e){

if(S.top==0)

return ;

e=S.base[--S.top];

}

int GetTop(St S,BiTree &e){

if(S.top==0)

return 0;

else{

e=S.base[S.top-1];

return 1;}

}

void CreateBiTree(BiTree &bt){

char c;

c=getchar();

if(c=='#')

bt=NULL;

else{

bt=(BiTree)malloc(sizeof(BiTNode));

bt->data=c;

CreateBiTree(bt->lchild);

CreateBiTree(bt->rchild);

}

}

void PreOrderTraverse(BiTree &bt){

St S;

BiTree p;

if(bt){

InitStack(S);

Push(S,bt);

while(!StackEmpty(S)){

while(GetTop(S,p)&&p){

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

Push(S,p->lchild);

}

Pop(S,p);

if(!StackEmpty(S)){

Pop(S,p);Push(S,p->rchild);

}

}

}

}

/\*void PreOrderTraverse(BiTree &bt){

if(bt){

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

PreOrderTraverse(bt->lchild);

PreOrderTraverse(bt->rchild);

}

}

void InOrderTraverse(BiTree &bt){

St S;

BiTree p;

if(bt){

InitStack(S);

Push(S,bt);

while(!StackEmpty(S)){

while(GetTop(S,p)&&p)

Push(S,p->lchild);

Pop(S,p);

if(!StackEmpty(S)){

Pop(S,p);

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

Push(S,p->rchild);

}

}

}

}\*/

void InOrderTraverse(BiTree &bt){

if(bt)

{

if(bt->lchild)

InOrderTraverse(bt->lchild);

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

if(bt->rchild)

InOrderTraverse(bt->rchild);

}

}

/\*void PostOrderTraverse(BiTree bt)

{

BiTree p,q;

St S;

InitStack(S);

Push(S, NULL);

p=bt;

q=NULL;

while (p || !StackEmpty(S)){

if (p && p!=q) { Push(S, p); p=p->lchild; }

else {

Pop(S,p);

if (!StackEmpty(S))

if (p->rchild && p->rchild!=q){

Push(S,p); p=p->rchild; }

else { printf("%c",p->data); q=p; }

}

}

}\*/

void PostOrderTraverse(BiTree bt)

{

if(bt)

{ if(bt->lchild)

PostOrderTraverse(bt->lchild);

if(bt->rchild)

PostOrderTraverse(bt->rchild);

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

}

}

void LevleOrder(BiTree &bt){

BiTree Queue[size],p;

int front=0,rear=0;

if(bt)

{Queue[rear++]=bt;

while(front!=rear){

p=Queue[front++];

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

if(p->lchild!=NULL)

Queue[rear++]=p->lchild;

if(p->rchild!=NULL)

Queue[rear++]=p->rchild;}

}

}

void yezi(BiTree &bt){

int count=0;

St S;

BiTree p;

printf("叶子结点为:");

if(bt){

InitStack(S);

Push(S,bt);

while(!StackEmpty(S)){

while(GetTop(S,p)&&p){

if(!p->lchild&&!p->rchild)

{ count++;

printf("%c",p->data);}

Push(S,p->lchild);

}

Pop(S,p);

if(!StackEmpty(S)){

Pop(S,p);Push(S,p->rchild);

}

}

}

printf("\n");

printf("请输出叶子结点的数目:\n");

printf("%d\n",count);

}

邻接表

#define MAX 20

#define addsize 10

typedef int VertexType;

typedef struct ArcNode{

int adjvex;//邻接点

struct ArcNode \*nextarc;

//int \*info;

}ArcNode,\*ArcPtr;

typedef struct VNode{

VertexType data;

ArcNode \*firstarc;

int tag;

}VNode,AdjList[MAX];

typedef struct{

int vexnum,arcnum;//顶点数和边数

AdjList vertices;//顶点向量

}ALGraph;

//栈的有关函数

typedef struct{

ArcPtr \*base;

int top;

int stacksize;

}St;

typedef struct{

int \*base;

int top;

int stacksize;

}St1;

void InitStack(St &S){

S.base=(ArcPtr\*)malloc(MAX\*sizeof(ArcPtr));

S.top=0;

S.stacksize=MAX;

}

void InitStack1(St1 &S){

S.base=(int \*)malloc(MAX\*sizeof(int));

S.top=0;

S.stacksize=MAX;

}

int StackEmpty(St S){

if(S.top==0)

return 1;

else

return 0;

}

int StackEmpty1(St1 S){

if(S.top==0)

return 1;

else

return 0;

}

void Push(St &S,ArcPtr p){

if(S.top>=S.stacksize){

S.base=(ArcPtr \*)realloc(S.base,(MAX+addsize)\*sizeof(ArcPtr));

S.stacksize+=addsize;

}

S.base[S.top++]=p;

}

void Push1(St1 &S,int p){

if(S.top>=S.stacksize){

S.base=(int \*)realloc(S.base,(MAX+addsize)\*sizeof(int));

S.stacksize+=addsize;

}

S.base[S.top++]=p;

}

void Pop(St &S,ArcPtr &p){

if(S.top==0)

return ;

p=S.base[--S.top];

}

void Pop1(St1 &S,int &p){

if(S.top==0)

return ;

p=S.base[--S.top];

}

//队列的有关函数

typedef struct {

ArcNode \*base; //初始化动态分配空间

int front;

int rear;

} SqQueue;

void InitQueue(SqQueue &Q){

//构造一个空队列

Q.base = (ArcNode \*)malloc(MAX \* sizeof(ArcNode));

if ( ! Q.base) exit(0);

Q.front = Q.rear = 0;

}//InitQueue

int QueueEmpty(SqQueue Q)

{//判断队列是否是空队列

if (Q.rear == Q.front ) return 1;

return 0;

}

void EnQueue(SqQueue &Q, ArcNode e){

//插入元素e为Q的新的队尾元素

if ((Q.rear+1)%MAX == Q.front)

exit(0);

Q.base[Q.rear] = e;

Q.rear = (Q.rear+1) % MAX;

}

void DeQueue(SqQueue &Q, ArcNode &e)

{// 删除Q的队头元素, 并用e返回其值

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

e = Q.base[Q.front];

Q.front = (Q.front+1) % MAX;

}

int LocateVex(ALGraph G,int v){

int i;

for( i=0; i<G.vexnum; i++)

if(G.vertices[i].data==v)

return i;

}

//创建一个图

void CreateAdjList(ALGraph &G){

//根据输入的有向图G的顶点数及边数，建立图G的邻接表

int i,j,k;

char v1,v2;

ArcNode \*s;

ArcNode \*p;

printf("请输入图的顶点数和弧数:\n");

scanf("%d%d",&G.vexnum, &G.arcnum);

printf("输入边表:\n");

for( i=0; i<G.vexnum; i++) //初始化头结点

{

scanf("%d",&G.vertices[i].data);

G.vertices[i].tag=0;

G.vertices[i].firstarc = NULL;

}

for( k=0; k<G.arcnum;k++){

printf("输入一条弧:\n");//构造邻接表

scanf( "%d",&v1); //输入一条边

scanf("%d",&v2);

i = LocateVex( G, v1); j = LocateVex( G, v2); //确定v1和v2在G中的位置

s = (ArcNode\*)malloc(sizeof(ArcNode));

s->adjvex = j; s->nextarc = NULL;

p = G.vertices[i].firstarc;

if( !p)

G.vertices[i].firstarc = s;

else {

while(p->nextarc)

p = p->nextarc;

p->nextarc = s; }

}

}//CreateAdjMatrix

//递归深度搜索

void DFS(ALGraph &G, int v){

//从第v个顶点出发递归地深度优先遍历图G

ArcNode \*w;

G.vertices[v].tag=1; printf("%d ",G.vertices[v].data);//访问第v个顶点

while(1){

for(w = G.vertices[v].firstarc;w;w = w->nextarc){

if(G.vertices[w->adjvex].tag==0)

DFS(G,w->adjvex);

}

for(v=0;v<G.vexnum&& G.vertices[v].tag==1;v++);

if(v==G.vexnum)

return ;

else{

G.vertices[v].tag= 1; printf(" %d ",G.vertices[v].data); //访问第v个顶点

}

}

}

/\*

//非递归深度搜索

void DFS(ALGraph &G, int v){

ArcNode \*p;

St S;

//从第v个顶点出发深度优先遍历图G

InitStack(S);

G.vertices[v].tag= 1; printf("%d ",G.vertices[v].data); //访问第v个顶点

p=G.vertices[v].firstarc;

while(1){

while( p || !StackEmpty(S))

{ while(p)

if (G.vertices[p->adjvex].tag==1)

p = p->nextarc;

else

{ printf("%3d",p->adjvex); G.vertices[p->adjvex].tag=1;

Push(S, p); p = G.vertices[p->adjvex].firstarc;

}

if ( !StackEmpty(S))

{ Pop( S, p); p = p->nextarc; }

}//while

for(v=0;v<G.vexnum&& G.vertices[v].tag==1;v++);

if(v==G.vexnum)

return ;

else{

G.vertices[v].tag= 1; printf(" %d ",G.vertices[v].data); //访问第v个顶点

p=G.vertices[v].firstarc;}

}

}\*/

//广度优先搜索

void BFSTraverse(ALGraph G){

//对图G作广度优先遍历

int v;

ArcNode \*w;

SqQueue Q;

InitQueue(Q); //初始化队列

for (v=0; v<G.vexnum; v++)

if( G.vertices[v].tag==0)

{ G.vertices[v].tag= 1; printf("%3d ",G.vertices[v].data); //访问第v个顶点

EnQueue( Q, \*G.vertices[v].firstarc);

while(!QueueEmpty(Q))

{ DeQueue(Q, \*G.vertices[v].firstarc);

for(w = G.vertices[v].firstarc; w; w = w->nextarc)

if( G.vertices[w->adjvex].tag==0)

{ G.vertices[w->adjvex].tag=1; printf("%3d",G.vertices[w->adjvex].data);

EnQueue( Q, \*w);

}//if }//while}//if}//BFSTraverse

void FindInDegree(ALGraph G,int indegree[]){

int i;

ArcNode \*p;

for(i=0;i<G.vexnum;i++)

indegree[i]=0;

for(i=0;i<G.vexnum;i++){

p=G.vertices[i].firstarc;

while(p){

indegree[p->adjvex]++;

p=p->nextarc;}}}

//拓扑排序

void TopologicalSort(ALGraph G){

St1 S;

int indegree[MAX];

int i,count,k;

ArcNode \*p;

FindInDegree( G, indegree) ; //对各顶点求入度

InitStack1(S) ;

for( i = 0; i <G.vexnum;i++ ) //将入度为0的顶点压入栈

if( !indegree[i] ) Push1( S, i );

count = 0; while( !StackEmpty1( S ))

{ Pop1( S, i ); printf( "i=%d,data=%d\n",i, G.vertices[i].data ); count++;

for( p = G.vertices[i].firstarc; p; p = p->nextarc )

{ k = p->adjvex;

if ( ! (--indegree[k] )) Push1( S, k );

}//for

}//while

if( count < G.vexnum ) { printf("该图有回路\n"); exit(0); }

}// TopologicalSort

邻接矩阵

#define MAX 20

#define addsize 10

typedef int AdjMatrix[MAX][MAX];

typedef char VertexType;

typedef struct{

int vexnum, arcnum;

VertexType vexs[MAX];

AdjMatrix arcs;

}MGraph;

int LocateVex(MGraph G,char v){

int i;

for( i=0; i<G.vexnum; i++)

if(G.vexs[i]==v)

return i;

}

void CreateAdjMatrix(MGraph &G){

int i,j,k;

char v1,v2;

printf("请输入无向图的顶点数和弧数:\n");

scanf("%d%d",&G.vexnum, &G.arcnum);

printf("请输入无向图的顶点向量:\n");

for(i=0; i<G.vexnum; i++)

{ getchar();

scanf("%c",&G.vexs[i]);}

for( i=0; i<G.vexnum; i++)

for(j=0; j<G.vexnum;j++)

G.arcs[i][j] = 0;

getchar();

for( k=0; k<G.arcnum; k++){

printf("请输入无向图中的边:\n");

scanf("%c",&v1); getchar();

scanf("%c",&v2);

getchar();

i = LocateVex(G,v1);

j = LocateVex(G,v2);

G.arcs[i][j] =1;

G.arcs[j][i] =1;

}

}

int FirstAdjVex( MGraph G,int v )

{

int i;

for( i = 0 ; i < G.vexnum ; i++ )

if( G.arcs[v][i] == 1 )

return i;

return -1;

}

int NextAdjVex( MGraph G,int v,int w )

{

int i;

for( i = w+1 ; i < G.vexnum ; i ++ )

if( G.arcs[v][i] == 1 )

return i;

return -1;

}

void DFS( MGraph G,int v,int visited[])

{

int w;

visited[v] = 1;

printf("%3c",G.vexs[v]);

for( w = FirstAdjVex( G,v ) ; w >= 0 ; w = NextAdjVex( G,v,w) )

if( !visited[w] )

DFS( G,w,visited);

}

/\*void DFSTraverse( MGraph &G )

{

St S;

int v,w;

int visited[MAX];

InitStack( S );

for( v = 0 ; v <= G.vexnum ; v ++ )

visited[v] = 0;

v = 0;

visited[v] = 1;

printf("%3c",G.vexs[v]);

w = FirstAdjVex( G,v );

while(1){

while( w >= 0 || !StackEmpty( S ) )

{

while( w >= 0 ){

if( visited[ w ] )

w = NextAdjVex( G,v,w );

else

{

printf("%3c",G.vexs[w]);

visited[w] = 1;

Push( S,w );

w = FirstAdjVex( G,v );

}}

if( !StackEmpty( S ) )

{

Pop( S,w );

w = NextAdjVex( G,w,v);

}

}

for(v=0;v<G.vexnum&& visited[v]== 1;v++);

if(v==G.vexnum)

return ;

else{

visited[v] = 1; printf(" %3c ",G.vexs[v]); //访问第v个顶点

w = FirstAdjVex( G,v );

}

}

}\*/

void Output(MGraph G){

int i,j;

printf("该图的邻接矩阵为:\n");

for( i=0; i<G.vexnum; i++)

{ printf("%c",G.vexs[i]);

for(j=0; j<G.vexnum;j++)

printf("%3d",G.arcs[i][j]);

printf("\n");

}

}

void DFSTraverse(MGraph G){

//对图G作深度优先遍历

int visited[MAX];

int v;

for (v=0; v<G.vexnum; v++) //访问标志数组初始化

visited[v] = 0;

for (v=0; v<G.vexnum; v++) //对尚未访问的顶点调用DFS

if(!visited[v]) DFS(G, v,visited);

}

void BFSTraverse( MGraph G )

{int v,w;

int visited[MAX];

SqQueue Q;

for( v = 0 ; v < G.vexnum ; v ++ )

visited[v] = 0;

InitQueue( Q );

for( v = 0 ; v <= G.vexnum ; v ++ )

if( !visited[v] )

{

visited[v] = 1;

printf("%3c",G.vexs[v]);

EnQueue( Q,G.vexs[v]);

while( !QueueEmpty( Q ) )

{

DeQueue( Q,G.vexs[v]);

for( w = FirstAdjVex( G,v ) ; w >= 0 ; w = NextAdjVex( G,v,w ) )

if( !visited[w] )

{

visited[w] = 1;

printf("%3c",G.vexs[w]);

EnQueue( Q,G.vexs[w] );}}}}

排序

typedef struct{

int num;

char a[20];

int score;

}Student;

void Input(Student stu[],int n){

int i;

for(i=1;i<=n;i++){

printf("请输入学生的学号，姓名，分数:\n");

scanf("%d",&stu[i].num);

getchar();

scanf("%s",&stu[i].a);

scanf("%d",&stu[i].score);

}

}

void InsertSort(Student stu[],int n){

//对待排序序列L进行直接插入排序

int i,j;

for(i=2; i<=n; i++)

if (stu[i].score<stu[i-1].score){

stu[0]=stu[i];

stu[i]=stu[i-1];

for(j = i-2; stu[i].score < stu[j].score; j--)

stu[i] = stu[j]; // 记录后移

stu[j+1] = stu[0]; //记录插入

}

} // InsertSort

void Maopaosort(Student stu[],int n){

int i,j;

Student temp;

for (j=1; j<=n-1; j++)

for (i =1; i<=n-j; i++){

if (stu[i].score >stu[i + 1].score){

temp = stu[i];

stu[i] = stu[i + 1];

stu[i + 1] = temp;

}

}

}

void Output(Student stu[],int n){

int i;

int j=1;

for(i=n;i>=1;i--){

printf("%3d %s%3d%3d\n",stu[i].num,stu[i].a,stu[i].score,j++);

}

}

int Partition(Student stu[], int low,int high){

//对顺序表L进行一趟快速排序，返回枢轴记录所在的位置

int key;

stu[0] = stu[low]; // 用子表的第一记录作枢轴记录

key = stu[low].score;

while(low<high){

while(low<high && stu[high].score>=key) --high;

if(low<high)

stu[low++] = stu[high]; // 将比pivotkey 小的记录移到低端

while(low<high &&stu[low].score<=key) ++low;

if(low<high)

stu[high--] =stu[low]; // 将比pivotkey 大的记录移到高端

}

stu[low] = stu[0];

return low;

}

void Qsort(Student stu[],int low,int high){

//对顺序表L的子序列L.r[low..high]作快速排序

int pivotloc;

if(low<high){

pivotloc = Partition(stu, low, high);

Qsort(stu, low, pivotloc-1);

Qsort(stu, pivotloc+1, high);

}

} // QSort

void QuickSort(Student stu[],int n)

{//对顺序表L作快速排序

Qsort(stu, 1, n);

} // QuickSort

void SelectSort(Student stu[],int n){

//对顺序表L进行简单选择排序

int i,k,j;

Student temp;

for ( i=1; i<n; ++i)

{ k=i;

//选择关键字最小的记录

for (j = i+1;j <= n; ++j)

if (stu[k].score >stu[j].score) k=j;

//最小记录与第i个记录互换

if (i!=k){

temp=stu[i];

stu[i]=stu[k];

stu[k]=temp;

}

}

}

void HeapAdjust(Student stu[], int s, int m){

//调整H.r[s..m]成一个大顶堆

Student rc;

int j;

rc=stu[s];

for(j=2\*s; j<=m; j\*=2 )

{ if( j<m && stu[j].score < stu[j+1].score) j++;

if( rc.score>= stu[j].score) break;

stu[s] = stu[j]; s=j;

}

stu[s] = rc;

}

void HeapSort(Student stu[],int n){

//对顺序表H进行堆排序

//初始堆建立

int i;

Student temp;

for ( i = n/2; i > 0; --i)

HeapAdjust(stu, i, n);

//堆的输出与调整

for ( i = n; i > 1; --i)

{ temp = stu[1];

stu[1] = stu[i];

stu[i] = temp;

HeapAdjust(stu, 1, i-1);

}

}

void Merge( Student s[], Student t[], int i, int m, int n )

{//将有序的SR[i..m]和SR[m+1..n]归并为有序的TR[i..n]

int j,k;

j = m+1; k = i;

while( i <= m && j <= n )

if( s[i].score <= s[j].score) t[k++] = s[i++];

else t[k++] = s[j++];

while( i <= m) t[ k++] = s[i++];

while( j <= n) t[ k++] = s[j++];

}//Merge

void MSort( Student stu[], Student t1[], int s, int t)

{//将SR[s..t]归并排序为TR1[s..t]。

Student t2[20];

int m;

if( s == t ) t1[s] = stu[s];

else

{ m = ( s+t )/2; //将SR[s..t]平分为SR[s..m]和SR[m+1..t]

MSort( stu, t2, s, m ); //将SR[s..m]归并为有序的TR2[s..m]

MSort( stu, t2, m+1, t); //将SR[m+1..t]归并为有序的TR2[m+1..t]

Merge( t2,t1,s,m,t); //将TR2[s..m]和TR2[m+1..t]归并到TR1[s..t]

}

}//MSort

查找

#define MAX 20

typedef int ElemType;

typedef struct{

ElemType \*elem;

int length;

} SSTable;

typedef struct{

int head;

int tail;

int maxnum;

}Block;

void InitLink(SSTable &L){

L.elem=(ElemType \*)malloc(MAX\*sizeof(ElemType));

L.length=0;

}

void EnList(SSTable &L,int i,int n){

L.elem[i]=n;

L.length++;

}

void CreateList( SSTable &L){

int num,n,i;

srand(time( 0) );

InitLink(L);

printf("请由输入表中元素个数:\n");

scanf("%d",&num);

printf("随机产生表中元素:\n");

for(i=1;i<=num;i++){

n=rand()%90+10;

printf("%3d",n);

EnList(L,i,n);

}

printf("\n");

}

int Search\_Seq(SSTable L, int key){

//在顺序ST中查找关键字等于key的数据元素。

int i;

L.elem[0]=key;

for ( i =1;i<=L.length; i++ )

if(L.elem[i]==key){

printf("查找成功！\n");

return i;

}//返回0则查找失败

return 0;

}

void sort(SSTable &L){

int i,j,temp;

for (j=1; j<=L.length-1; j++)

for (i =1; i<=L.length-j; i++){

if (L.elem[i] >L.elem[i + 1]){

temp = L.elem[i];

L.elem[i] = L.elem[i + 1];

L.elem[i + 1] = temp;

}

}

for (i=1; i<=L.length; i++){

printf("%3d", L.elem[i]);

}

printf("\n");

}

int Search\_Bin(SSTable L, int key,int low,int high)

{//在有序表ST中折半查找关键字为key的元素

int mid;

while (low <= high)

{ mid = (low+high)/2;

if ( key == L.elem[mid]) return mid;

else if (key < L.elem[mid]) high = mid-1;

else low = mid+1;

}

return 0;

}

void CreateList1( SSTable &L){

int num;

InitLink(L);

printf("请由输入表中元素个数:\n");

scanf("%d",&num);

printf("请输入输入整组数无序，但局部有序的一组整数:\n");

int i,n;

for(i=1;i<=num;i++){

scanf("%d",&n);

EnList(L,i,n);

}

}

int Max(SSTable L,int h,int t){

int max,i;

max=L.elem[h];

for(i=h;i<=t;i++)

if(max<L.elem[i])

max=L.elem[i];

return max;

}

int Blocksearch(SSTable &L){

Block a[MAX];

int i,j,k,n,m,x;

printf("要分为几块？\n");

scanf("%d",&n);

if(n>L.length)

printf("无法分块\n");

j=1;

if(L.length%n==0)

for(i=1;i<=n;i++){

k=L.length/n;

a[i].head=j;

a[i].tail=j+k-1;

a[i].maxnum=Max(L,j,j+k-1);

j=j+k;

}

else{

m=L.length/n;

for(i=1;i<=n-1;i++){

k=m+1;

a[i].head=j;

a[i].tail=j+k-1;

a[i].maxnum=Max(L,j,j+k-1);

j=j+k;

}

k=L.length-(n-1)\*k;

a[i].head=j;

a[i].tail=j+k-1;

a[i].maxnum=Max(L,j,j+k-1);

}

printf("下面进行分块查找，请输入要查找的数:\n");

scanf("%d",&x);

for(i=1;i<=n&&x>a[i].maxnum;i++);

if(i>n)

return 0;

j=a[i].head; k=a[i].tail;

for(i=j;i<=k;i++)

if(L.elem[i]==x)

return i;

return 0;

}

void main(){

SSTable L,L2;

int key;

int low,high;

int i,j;

int x;

CreateList(L);

printf("下面进行顺序查找，请输入要查找的数:\n");

scanf("%d",&key);

i=Search\_Seq(L, key);

if(i==0)

printf("查找失败!\n");

else

printf("该数为表的第%d个数\n",i);

printf("冒泡排序后为:\n");

sort(L);

low=1; high = L.length;

printf("下面进行折半查找，请输入要查找的数:\n");

scanf("%d",&key);

j=Search\_Bin(L, key, low, high);

if(j==0)

printf("查找失败!\n");

else

printf("该数为表的第%d个数\n",j);

CreateList1( L2);

i=Blocksearch( L2);

if(i==0)

printf("查找失败!\n");

else

printf("该数为表的第%d个数\n",i);

}

折半递归

void Search\_Bin(SSTable L, int key,int low,int high){

int mid;

mid=(low+high)/2;

if (key == L.elem[mid]){

printf("查找成功！\n");

return ;

}

if (low>high){

printf("查找失败！\n");

return ;

}

if (key < L.elem[mid])

Search\_Bin( L, key,low,mid-1);

if (key > L.elem[mid])

Search\_Bin(L,key,mid+1,high);

}

进制转换

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 100

typedef struct

{

char data[MAX\_SIZE];

int top;

}stack;

int isfull(stack s)

{

return (++s.top==MAX\_SIZE);

}

int isempty(stack s)

{

return s.top==-1;

}

void push(stack \* s,char data)

{

if(isfull(\*s))

{

printf("栈已满,不能执行操作!\n");

exit(1);

}

s->data[++s->top]=data;

}

void pop(stack \*s)

{

if(isempty(\*s))

{

printf("栈已空,不能执行操作!\n");

exit(1);

}

s->top--;

}

char getpop(stack \*s)

{

if(isempty(\*s))

{

printf("栈已空,不能执行操作!\n");

exit(1);

}

return s->data[s->top];

}

int main()

{

int num,n;

stack s;

char ch;

s.top=-1;

printf("输入转换的数据和对应的进制:");

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

printf("%d对应的%d进制为:",num,n);

while(num!=0)

{

if(num%n>=10)

ch=((num%n)-10)+'a';

else

ch=((num%n)+'0');

push(&s,ch);

num=num/n;

}

while(!isempty(s))

{

putchar(getpop(&s));

pop(&s);

}

printf("\n");

return 0;