使用顺序表和链表结合的方法对输入数据进行存储。

一、对于一行的输入，使用顺序表进行存储。

//规定容量不足时的添加标准以及初始行最大容量

#define ADD x

#define LEN y

struct List

{

//记录一行输入的字符串

Elemtype \* str;

//记录当前Elemtype数

int len;

//初始一行最大Elemtype数(可扩容)

int curlen;

//一行最大Elemtype数

int maxlen;

} \*L;

//初始化顺序表

void init\_list(struct List \*L)

{

L -> str = (Elemtype\*) malloc (addlen \* sizeof(Elemtype));

if(L -> str == NULL)

{

printf(“init fail\n”);

exit(-1);

}

L -> len = 0;

L -> curlen = LEN;

}

//顺序表插入(位置为光标悬浮处)

void insert\_list (struct List \*L ,int i, Elemtype data)

{

Elemtype \*neww, \*loc , \*tmp;

//当容量不够时对顺序表扩容并保存原数据

if(L -> len > L -> curlen)

{

if(L -> len + ADD < L -> maxlen)

{

neww = (Elemtype\*) realloc (L -> str, (L -> curlen + ADD) \* sizeof(Elemtype));

L -> str = neww;

L -> curlen += ADD;

}

else

{

printf(“expand fail\n”);

exit(-1);

}

}

//目标指针指向要插入的目标地址

loc = &(L -> str[i-1]);

//后续元素依次向后

for(tmp = L -> str + L -> len - 1; tmp >= loc; tmp--)

\*(tmp + 1) = \*tmp;

\*loc = data;

L -> len++;

}

//顺序表删除

void delete\_list(struct List \*L, int i)

{

int \*loc, \*tmp;

tmp = L -> str + L -> len – 1;

for(loc = &(L -> str[i – 1]); loc <= tmp; loc++)

\*(loc – 1) = \*loc;

L -> len--;

}

二、对于行与行之间的连接，使用链表，把记录一行数据的顺序表作为链表中的节点。

struct Head

{

//记录行数

int num;

//指向第一行

struct node \*first;

} \*head;

struct Node

{

//以行为节点，指针指向某一行的顺序表

struct list \* row;

//指向下一行

struct node \* next;

} \*node;

//初始化链表

void init\_head(struct Head \*head)

{

head = (struct Head\*) malloc (sizeof(struct Head));

Head -> num = 0;

Head -> first = NULL;

}

//添加行(位置为光标悬浮行数)

void add\_row(struct Head \*head, int j)

{

struct Node \*tmp, \*p;

p = head -> first;

tmp = (struct Node\*) malloc (sizeof(struct Node));

insert\_list(tmp -> row, i, data);

//第1行输入数据

if(j == 1)

{

tmp -> next = head -> first;

head -> first = tmp;

}

//第1行之后输入数据

else

{

for(int t = 2; t < j; t++)

p = p -> next;

tmp -> next = p -> next;

p -> next = tmp;

}

head -> num++;

}

//删除行

void delete\_row(struct Head \*head, int j)

{

struct Node \*tmp, \*p;

tmp = head -> first;

//删除第1行

if(j == 1)

{

head -> first = tmp -> next;

free(tmp);  
 }

//删除第2行之后

else

{

for(int i = 2; i < j; i++)

tmp = tmp -> next;

p = temp -> next;

tmp -> next = p -> next;

free(p);

}

head -> num--;

}