* + - 1. **CreateList函数中，为什么定义了一个EleType \*data作为头结点的数据，而不是EleType data?**

答：EleType data=0; 如果EleType 为结构体类型则就会出错，因为结构体变量不是那样赋值的。于是我改成EleType data;这样就可以用了，但是会出现警告，所以就诞生了这样的设计EleType \*data=0;就是说任何类型的指针都可以那样赋值。这样就解决了所有问题。

* + - 1. **将头结点去掉，实现链表，体会为什么要有头结点。**

头节点去掉后的程序为：

**typedef char EleType;**

**typedef struct node{**

**EleType data;**

**struct node \*next;**

**}ChainNode;**

**typedef struct {**

**ChainNode \*head;**

**ChainNode \*tail;**

**}List;**

**ChainNode \*GetAddr(List \*lp,int n);**

**ChainNode \*NewChainNode(EleType data);**

**void DestoryList(List \*lp);**

**void ClearList(List \*lp);**

**int InsertList(List \*lp,int n,EleType data);**

**int AppendList(List \*lp,EleType data);**

**int DeleteList(List \*lp,int n);**

**int TraverseList(List \*lp,int (\*f)(EleType \*));**

**int GetElement(List \*lp,int n,EleType \*data);**

**List \*CreateList()/\*EleType \*data\*/**

**{**

**List \*lp;**

**lp=(List \*)malloc(sizeof(List));**

**if(!lp)return 0;**

**/\* lp->head=NewChainNode(\*data);**

**if(!lp->head){free(lp);return 0;}**

**lp->tail=lp->head; \*/**

**lp->head=0;**

**lp->tail=0;**

**return lp;**

**}**

**void DestoryList(List \*lp)**

**{**

**ClearList(lp);**

**free(lp);**

**}**

**void ClearList(List \*lp)**

**{**

**while(DeleteList(lp,1));**

**}**

**int InsertList(List \*lp,int n,EleType data)**

**{**

**ChainNode \*p;**

**ChainNode \*newp;**

**newp=NewChainNode(data);**

**if(!newp) return 0;**

**if(n==1)**

**{**

**newp->next=lp->head;**

**lp->head=newp;**

**}**

**else**

**{**

**p=GetAddr(lp,n-1);**

**if(!p)return 0;**

**newp->next=p->next;**

**p->next=newp;**

**}**

**return 1;**

**}**

**int AppendList(List \*lp,EleType data)**

**{**

**ChainNode \*newp;**

**newp=NewChainNode(data);**

**if(!newp) return 0;**

**if(!lp->head)**

**{**

**lp->head=newp;**

**lp->tail=newp;**

**}**

**else**

**{**

**lp->tail->next=newp;**

**lp->tail=newp;**

**}**

**return 1;**

**}**

**int DeleteList(List \*lp,int n)**

**{**

**ChainNode \*p;**

**ChainNode \*p1;**

**if(!lp->head) return 0;**

**if(n==1)**

**{**

**p1=lp->head;**

**lp->head=lp->head->next;**

**free(p1);**

**}**

**else**

**{**

**p=GetAddr(lp,n-1);**

**if(!(p&&p->next))return 0;**

**p1=p->next;**

**p->next=p->next->next;**

**free(p1);**

**}**

**return 1;**

**}**

**int GetElement(List \*lp,int n,EleType \*data)**

**{**

**ChainNode \*p;**

**p=GetAddr(lp,n);**

**if(!p)return 0;**

**\*data=p->data;**

**return 1;**

**}**

**int TraverseList(List \*lp,int (\*f)(EleType \*))**

**{**

**ChainNode \*p;**

**int a=0;**

**for(p=lp->head;p;p=p->next)**

**{**

**if(!f(&(p->data)))return a+1;**

**a++;**

**}**

**return 0;**

**}**

**ChainNode \*NewChainNode(EleType data)**

**{**

**ChainNode \*p;**

**p=(ChainNode \*)malloc(sizeof(ChainNode));**

**if(!p)return 0;**

**p->data=data;**

**p->next=0;**

**return p;**

**}**

**ChainNode \*GetAddr(List \*lp,int n)**

**{**

**ChainNode \*p;**

**int a=1;**

**if(n<1)return 0;**

**p=lp->head;**

**while(p&&a<n)**

**{**

**p=p->next;**

**a++;**

**}**

**return p;**

**}**

最大的体会是没有头结点的时候，你得考虑多种情况，然而在有头结点的时候无需考虑。

* + - 1. **DestoryList()与ClearList()的返回值为什么为设为空？**

答：整个函数中失败有两个，一是可能申请空间失败，比如malloc函数，二是用户输入不对导致失败，如GetAddr这个函数n<0则会返回0表示得到指定地址失败，其他函数用到malloc和GetAddr这两个函数就有可能发生失败。所以需要有个返回值来表达函数成功与否。while(ListDelete(lp,1));这个是我们自己写的，用户不能改变，那么GetAddr这个函数就不会失败。不会失败就可以设置返回值为空了。 **4.ListDelete()中可否可删除判断链表为空的if语句？这条语句的意义何在？**

答：可以删除，但是这条语句意义在于当你链表确实为空的时候大大增加程序的效率。

**5.实现通用链表**

#define eletype void \*

/\* 1是char

2是int

3是long

4是struct \*/

typedef struct node{

eletype data;

int type;

struct node \*next;

}chainnode;

typedef struct {

chainnode \*head;

}list;

chainnode \*get\_addr(list \*lp,int n);

list \*create\_list();

void destory\_list(list \*lp);

void clear\_list(list \*lp);

int append\_list(list \*lp,eletype data,int type,int size);

int insert\_list(list \*lp,eletype data,int n,int type,int size);

int delete\_list(list \*lp,int n);

int get\_element(list \*lp,int n,eletype \*data);

chainnode \*new\_chainnode(eletype data,int type,int size);

int traverse\_list(list \*lp,int (\*f)(eletype \*data,int type));

list \*create\_list()

{

list \*lp;

eletype \*data=0;

lp=(list \*)malloc(sizeof(list));

if(!lp) return 0;

lp->head=new\_chainnode(\*data,0,0);

if(!lp->head){free(lp);return 0;}

return lp;

}

void destory\_list(list \*lp)

{

clear\_list(lp);

free(lp->head);

free(lp);

}

void clear\_list(list \*lp)

{

while(delete\_list(lp,1));

}

int append\_list(list \*lp,eletype data,int type,int size)

{

chainnode \*newp;

chainnode \*p;

newp=new\_chainnode(data,type,size);

if(!newp)return 0;

for(p=lp->head;p->next;p=p->next);

p->next=newp;

return 1;

}

int insert\_list(list \*lp,eletype data,int n,int type,int size)

{

chainnode \*p;

chainnode \*newp;

newp=new\_chainnode(data,type,size);

if(!newp)return 0;

p=get\_addr(lp,n-1);

if(!p)return 0;

newp->next=p->next;

p->next=newp;

return 1;

}

int delete\_list(list \*lp,int n)

{

chainnode \*p1;

chainnode \*p2;

if(!lp->head->next) return 0;

p1=get\_addr(lp,n-1);

if(!(p1&&p1->next)) return 0;

p2=p1->next;

p1->next=p1->next->next;

free(p2);

return 1;

}

int get\_element(list \*lp,int n,eletype \*data)

{

chainnode \*p;

p=get\_addr(lp,n);

if(!p)return 0;

\*data=p->data;

return 1;

}

chainnode \*new\_chainnode(eletype data,int type,int size)

{

chainnode \*p;

eletype data1;

p=(chainnode \*)malloc(sizeof(chainnode));

data1=(eletype)malloc(size);

if(!p) return 0;

memcpy(data1,data,size);

p->data=data1;

p->type=type;

p->next=0;

return p;

}

int traverse\_list(list \*lp,int (\*f)(eletype \*data,int type))

{

chainnode \*p;

int a=0;

for(p=lp->head->next;p;p=p->next)

{

if(!f(&(p->data),p->type)) return a+1;

a++;

}

return 0;

}

chainnode \*get\_addr(list \*lp,int n)

{

chainnode \*p;

int a;

if(n<0) return 0;

p=lp->head;

a=0;

while(p&&a<n)

{

p=p->next;

a++;

}

return p;

}

大致思路是chainnode结构体三个成员，data是来放置数据首地址，type这个是用来记录这个数据的类型，主要方便最后的打印，next还是不变，指向下一个节点。创建新节点时需要传递的参数有数据的地址，数据的大小，类型。然后为新节点申请空间，再申请一个数据大小的空间，把数据原有地址处的数据复制到新开辟的空间，这个空间的首地址给新节点成员data,类型给 type,next=0;

**6.尾指针实现**

typedef char EleType;

typedef struct node

{

EleType data;

struct node \* next;

}ChainNode;

typedef struct

{

ChainNode \* head;

ChainNode \* tail;

}List;

List\* CreateList(void);

void DestroyList(List\*);

void ClearList(List\*);

int ListInsert(List\*, int, EleType);

int ListDelete(List\*, int);

int ListAppend(List\*, EleType);

int TraverseList(List\*, int (\*)(EleType \*));

int GetElement(List\*, int, EleType\*);

ChainNode\* newChainNode(EleType);

ChainNode\* GetAddr(List\* list, int pos);

List\* CreateList(void)

{

List \*lp;

EleType \*data=0;

lp=(List \*)malloc(sizeof(List));

if(!lp)return 0;

lp->head=newChainNode(\*data);

if(!lp->head){free(lp);return 0;}

lp->tail=lp->head;

return lp;

}

void DestroyList(List \*lp)

{

ClearList(lp);

free(lp->head);

free(lp);

}

void ClearList(List \*lp)

{

while(ListDelete(lp,1));

}

int ListInsert(List \*lp, int n, EleType data)

{

ChainNode \*p;

ChainNode \*newp;

newp=newChainNode(data);

if(!newp)return 0;

p=GetAddr(lp,n-1);

if(!p)return 0;

newp->next=p->next;

p->next=newp;

if(!newp->next) lp->tail=newp;

return 1;

}

int ListDelete(List \*lp, int n)

{

ChainNode \*p;

ChainNode \*p1;

if(!lp->head->next) return 0;

p=GetAddr(lp,n-1);

if(!(p&&p->next))return 0;

p1=p->next;

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

if(!p->next)lp->tail=p;

free(p1);

return 1;

}

int ListAppend(List \*lp, EleType data)

{

ChainNode \*newp;

newp=newChainNode(data);

if(!newp)return 0;

lp->tail->next=newp;

lp->tail=newp;

return 1;

}

int TraverseList(List \*lp, int (\*f)(EleType \*))

{

ChainNode \*p;

int a=0;

for(p=lp->head->next;p;p=p->next)

{

if(!f(&(p->data))) return a+1;

a++;

}

return 0;

}

int GetElement(List \*lp, int n, EleType \*data)

{

ChainNode \*p;

p=GetAddr(lp,n);

if(!p)return 0;

\*data=p->data;

return 1;

}

ChainNode\* newChainNode(EleType data)

{

ChainNode \*p;

p=(ChainNode \*)malloc(sizeof(ChainNode));

if(!p)return 0;

p->data=data;

p->next=0;

return p;

}

ChainNode\* GetAddr(List \*lp , int pos)

{

ChainNode \*p;

int a=0;

if(pos<0) return 0;

p=lp->head;

while(p&&a<pos)

{

p=p->next;

a++;

}

return p;

}