**#ifndef \_\_LIST\_H**

**#define \_\_LIST\_H**

**#include <stdio.h>**

**/\* 该链表不限数据类型：用List中的size记录数据类型的长度。**

**结点数据项指针指向长度为size的空间,参数为数据的地址 \*/**

**/\* 定义线性表存储的元类型 \*/**

**typedef void \*EleType;**

**/\* 定义链表结点类型 \*/**

**typedef struct node{**

**EleType pdata; /\* 数据项指针;(实际上数据存放在pdata指向的内存中) \*/**

**struct node \*next; /\* 指针项 \*/**

**} ChainNode;**

**/\* 定义链表参数：存放链表的头指针，尾指针，及该型链表的数据长度 \*/**

**typedef struct{**

**ChainNode \*head; /\* 指向头结点的指针(头指针) \*/**

**ChainNode \*last; /\* 指向末结点的指针(尾指针) \*/**

**int size; /\* 数据结点的数据项指针指向的内存长度 \*/**

**} List;**

**/\* 说明：**

**声明List的操作函数，通过调用这些函数来使用list 。**

**本链表的头结点不用于存储数据，头结点的指针项指向存储线性表的第一个数据结点。**

**头结点编号为0，它指向的数据结点编号为1，依此后推，最后一个结点的指针项为NULL.**

**\*/**

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

**\* 函数名称：CreateList**

**\* 函数功能：创建线性表：**

**\* 输入参数：int size 数据结点的数据项指针所指向的内存长度**

**\* 输出参数：无**

**\* 返回值： 成功--返回list \* 线性表的指针 ，不成功--返回NULL**

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

**List \*CreateList(int size);**

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

**\* 函数名称： DestroyList**

**\* 函数功能： 撤消线性表(包括清空所有数据结点、头结点、头指针)**

**\* 输入参数： List \* 线性表的指针**

**\* 输出参数： 无**

**\* 返回值： 无**

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

**void DestroyList(List \*);**

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

**\* 函数名称： ClearList**

**\* 函数功能： 清空线性表，即删除所有数据结点**

**\* 输入参数： List \* 线性表的指针**

**\* 输出参数： 无**

**\* 返回值： 无**

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

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

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

**\* 函数名称： ListAppend**

**\* 函数功能： 追加结点，在线性表的最后一个元素的后面加一个数据结点。**

**\* 输入参数： List \* 线性表的指针, EleType 指向要追加数据的指针**

**\* 输出参数： 无**

**\* 返回值： 1--操作成功，0--不成功**

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

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

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

**\* 函数名称： ListInsert**

**\* 函数功能： 指定编号处插入数据结点**

**\* 输入参数： List \* 线性表的指针, int 插入处的编号(>=1), EleType 指向要插入的数据的指针**

**\* 输出参数： 无**

**\* 返回值： 1--操作成功，0--不成功**

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

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

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

**\* 函数名称： ListDelete**

**\* 函数功能： 删除指定编号处的数据结点**

**\* 输入参数： List \* 线性表的指针, int 要删除的数据结点的编号(>=1)**

**\* 输出参数： 无**

**\* 返回值： 1--操作成功，0--不成功**

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

**int ListDelete(List \*, int);**

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

**\* 函数名称： GetElement**

**\* 函数功能： 获取指定编号结点的数据**

**\* 输入参数： List \* 线性表的指针, int 指定结点编号(>=0)**

**\* 输出参数： EleType 读取的数据存入它指向的空间（存放读取数据的指针）**

**\* 返回值： 1--操作成功，0--不成功**

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

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

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

**\* 函数名称： GetAddr**

**\* 函数功能： 获取指定编号结点的地址**

**\* 输入参数： List \* 线性表的指针, int 结点的编号(>=0,为0时对应的是头结点，大于0时对就的是数据结点)**

**\* 输出参数： 无**

**\* 返回值： 成功--返回ChainNode \*结点指针，不成功--NULL**

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

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

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

**\* 函数名称： NewChainNode**

**\* 函数功能： 创建一个存储结点，并初始化该结点的数据。**

**\* 输入参数： List \*线性表的指针, EleType 指向新结点中的要存放数据的指针**

**\* 输出参数： 无**

**\* 返回值： 成功--返回ChainNode \*结点指针，不成功--NULL**

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

**ChainNode \*NewChainNode(List \*, EleType);**

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

**\* 函数名称： TraverseList**

**\* 函数功能： 遍历：依次访问线性表中的每一个数据结点，**

**\* 并对每一个数据结点的数据都用某个函数进行处理**

**\* 输入参数： List \* 线性表的指针,**

**\* int (\*)(EleType)对数据结点指向的数据进行操作的函数指针**

**\* 输出参数： 无**

**\* 返回值： 成功遍历所有数据结点返回0，否则返回当前应访问的数据结点编号(该结点还没被访问)。**

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

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

**#endif**

**/\* 函数功能：创建线性表：\*/**

**List \*CreateList(int size)**

**{**

**List \*p = NULL;**

**EleType pdata = NULL; /\* ???? \*/**

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

**if (p == NULL) {return NULL;}**

**p->size = size; /\* 记录该线性表结点数据项的长度 \*/**

**p->head = NewChainNode(p, pdata);**

**memset(p->head->pdata, 0, size); /\* 头结点的数据项指向的数据清零。\*/**

**if (p->head == NULL){free(p); return NULL;}**

**p->last = p->head; /\* 此时链表为空，只有头结点，没有数据结点 \*/**

**return p;**

**}**

**/\* 函数功能： 撤消线性表(包括清空所有数据结点、头结点、头指针) \*/**

**void DestroyList(List \*lp)**

**{**

**ClearList(lp); /\* 清空线性表的所有数据结点 \*/**

**free(lp->head); /\* 释放头结点 \*/**

**free(lp); /\* 释放线性表 \*/**

**}**

**/\* 函数功能：清空线性表的所有数据结点，即删除所有数据结点 \*/**

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

**{**

**while (ListDelete(lp, 1)); /\* 每次删除第一个数据结点 \*/**

**}**

**/\* 函数功能：追加结点，在线性表的最后一个元素的后面加一个数据结点。 \*/**

**int ListAppend(List \*lp, EleType pdata)**

**{**

**ChainNode \*newp = NULL;**

**newp = NewChainNode(lp, pdata);**

**if (newp == NULL){return 0;}**

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

**lp->last = newp; /\* 尾指针后移 \*/**

**return 1;**

**}**

**/\* 函数功能：指定编号处插入数据结点 \*/**

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

**{**

**ChainNode \*p = NULL;**

**ChainNode \*newp = NULL;**

**p = GetAddr(lp, n-1); /\* 获取插入结点的前一个结点的地址 \*/**

**if (p == NULL){return 0;}**

**newp = NewChainNode(lp, pdata);**

**if (newp == NULL){return 0;}**

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

**if(p == lp->last) {lp->last = newp;} /\* 在链表末插入数据结点，尾指针后移 \*/**

**p->next = newp;**

**return 0;**

**}**

**/\* 函数功能：删除指定编号处的数据结点 \*/**

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

**{**

**ChainNode \*p = NULL;**

**ChainNode \*p1 = NULL;**

**if (lp->head->next == NULL){return 0;} /\* 没有数据结点(即头结点指针项为NULL) \*/**

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

**if (p == NULL){return 0;}**

**p1 = p->next; /\* 记录要删除的结点 \*/**

**p->next = p->next->next; /\* 将要删除的结点移出链表 \*/**

**if(p->next->next == NULL) /\* 如果删除的是尾结点，则尾指针前移 \*/**

**{**

**lp->last = p;**

**}**

**free(p1->pdata); /\* 释放数据结点数据项指向的空间 \*/**

**free(p1); /\* 释放要删除的结点的空间 \*/**

**return 1;**

**}**

**/\* 函数功能：获取指定编号结点的数据 \*/**

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

**{**

**ChainNode \*p = NULL;**

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

**if (p == NULL){return 0;}**

**memcpy(pdata, p->pdata, lp->size); /\* 将数据项的数据拷贝到data指向的内存中 \*/**

**return 1;**

**}**

**/\* 函数功能：获取指定编号结点的地址 \*/**

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

**{**

**ChainNode \*p = NULL;**

**if (n < 0){return 0;}**

**p = lp->head;**

**while (p!=NULL && n>0)**

**{**

**p = p->next;**

**n--;**

**}**

**return p;**

**}**

**/\* 函数功能： 创建一个存储结点，并初始化该结点的数据。\*/**

**ChainNode \*NewChainNode(List \*lp, EleType pdata)**

**{**

**ChainNode \*p = NULL;**

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

**if (p == NULL){return 0;}**

**p->pdata = (EleType \*)malloc(lp->size);**

**if (p->pdata == NULL){free(p); return 0;}**

**memcpy( p->pdata, pdata, lp->size); /\* 用pdata指向内存的数据初始化数据项指向的内存 \*/**

**p->next = NULL;**

**return p;**

**}**

**/\* 函数功能：遍历：依次访问线性表中的每一个数据结点，**

**并对每一个数据结点的数据都用某个函数进行处理 \*/**

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

**{**

**ChainNode \*p = NULL;**

**int num = 0;**

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

**{**

**if( f(p->pdata) == 0) {return num+1;} /\* 操作失败，返回下一个编号 \*/**

**num++;**

**}**

**return 0;**

**}**

**测试代码：**

#include <stdio.h> //**测试二**

#include "t\_list.h"

typedef struct STU{char a; int b;} Stu;

Stu a[8] = {{'a', 1}, {'b', 2}, {'c', 3}, {'d', 4},

{'e', 5}, {'f', 6}, {'g', 7}, {'h', 8}};

void showlist(List \*);

int putelement(EleType);

int main(void)

{

List \*stu\_lp = NULL;

int n = 0;

Stu data;

stu\_lp = CreateList(sizeof(Stu));

if (stu\_lp == NULL)

{

printf("Create?\n");

return 1;

}

for (n=0; n<8; n++)

{

ListAppend(stu\_lp, &a[n]);

showlist(stu\_lp);

}

data.a = '\*';

data.b = 8;

ListInsert(stu\_lp, 1, &data);showlist(stu\_lp);

data.a = 'a';

data.b = 20;

ListInsert(stu\_lp, 8, &data);showlist(stu\_lp);

ListDelete(stu\_lp, 1);showlist(stu\_lp);

ListDelete(stu\_lp, 7);showlist(stu\_lp);

return 0;

}

void showlist(List \*lp)

{

TraverseList(lp, putelement);

printf("\n");

}

int putelement(EleType data)

{

printf("%c %d ", ((Stu \*)data)->a, ((Stu \*)data)->b);

return 1;

}

#include <stdio.h> //**测试一**

#include "t\_list.h"

char a[20] = "hello world!";

void showlist(List \*lp)

{ TraverseList(lp, putelement);

printf("\n");

}

int putelement(EleType pdata)

{ printf("%c", \*(char \*)pdata);

return 1;

}

int main(void)

{

List \*ch\_lp = NULL;

int n = 0;

char ch = 'a';

ch\_lp = CreateList(sizeof(char));

if (ch\_lp == NULL)

{

printf("Create?\n");

return 1;

}

for (n=0; a[n]; n++)

{

ListAppend(ch\_lp, &a[n]);

showlist(ch\_lp);

}

ListInsert(ch\_lp, 1, &a[0]);

showlist(ch\_lp);

ListInsert(ch\_lp, 8, &ch);

showlist(ch\_lp);

ListDelete(ch\_lp, 1);

showlist(ch\_lp);

ListDelete(ch\_lp, 7);

showlist(ch\_lp);

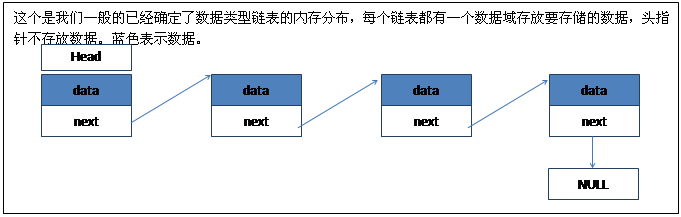
return 0;

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*以下是别人的\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

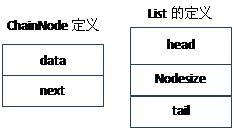
**可接受任意类型的数据的C语言链表**

     今天在宣讲会上讲了一下我所写的那个可以接收任意类型的参数类型的链表，感觉还是没有讲清楚，再次整理一下 ，同样的道理我们可以构造出其它的可以接受任意数据类型的结构，如：可接收任意数据类型的栈，可接收任意数据类型的队列，可接收任意数据类型的树·················

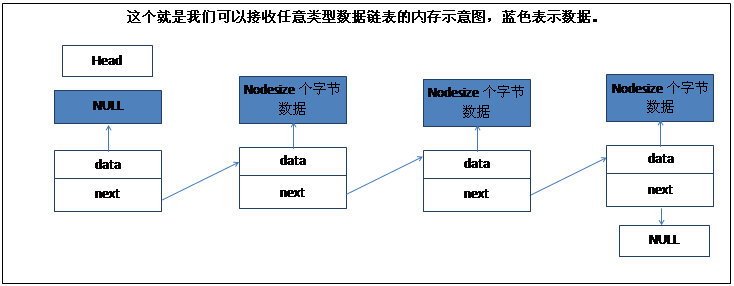


       现在我们要在链表中存储任意类型的数据(也就是说数据所占字节数是在使用链表的时候确定)，既然要能存储任意类型的数据，那么我们的链表的节点和链表的定义就要做一些修改了。

       下图是节点和链表的定义，data是一个ElemType类型的数据，而ElemType是被我们定义成了一个void \*，也就是一个空指针。head和tail都是一个ChainNode类型的指针，作为链表的头结点和尾巴节点，头结点本书的数据域是不存放数据的。Nodesize表示要存储的数据类型的字节大小。



这个时候data这个指针才是真正指向了我们存放在链表里面的数据，



**实现代码的讲解：**

首先我们初始化一个链表mylist

myElemType是我们自己定义的任意类型的数据

       List \* mylist;

       mylist = CreateList( sizeof(myElemType) );

       然后我们想要在链表上面附加myElemType类型的数据，可以直接调用ListAppend(mylist ,a);

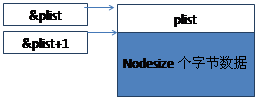
     //a是一个myElemType类型数据，

       这里的ListAppend第二个参数之所以能接受任意参数是因为它的声明形式是

       i**nt ListAppend(List \* plist,...)**

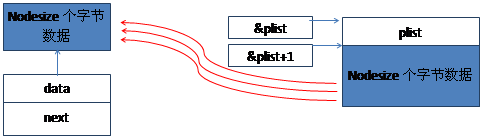
       我们在声明的时候已经指示了，除第一个参数外，后面的参数可以是任意类型的，

       下面分析一下在执行这条语句执行后参数在栈空间里面的存储形式，首先入栈的是plist这个指针，存储这个指针位置是下面就是我们1个myElemType类型的数据（大小为Nodesize，初始化链表的时候确定）。



**下面以ListAppend为列子，分析的思想：**

           已经知道了节点的内存示意（图一）和调用**ListAppend**之后栈空间的内存分布（图二）



        图一                                                                                    图二

             那么首先我们就是要为节点本身分配一款内存（存储两个指针next和data），然后我们还要为data这个指针分配额一块内存，用来存储数据，也就是上面图一蓝色的部分。然后再将此时已经在栈里面的数据通过memcpy函数拷贝到节点指针data所指向的内存（图中的三个红色箭头）。关于memcpy函数:

**void \*memcpy( void \****dest***, const void \****src***, size\_t***count***);**

       就是从指针*src*所指向地址拷贝*count个字节*到另一个指针所*dest*指向的地址，拷贝个数由参数size确定，我们这里当然可以传入我们在初始化链表的时候给予的数据字节大小的plist->Nodesize;

实际操作如下的：

ListAppend函数代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/lilien1010/article/details/7786164)

1. **int** ListAppend(List \* plist,...)
2. {
3. ChainNode \* newpt = 0;
4. **void** \* data;
5. **void** \* pos;
6. pos = &plist + 1;           //这个时候pos指向的实际就是在栈里面的一个myElemType类型数据的首地址
7. **if**( !(plist && plist->head) )        **return** 0;   //防止指针为空
9. data = (**void** \*)malloc(plist->Nodesize);      //为直接的数据域指针分配空间，大小是（当前Nodesize）
10. **if**( !data )     **return** 0;                   //防止指针为空
12. memcpy(data,pos,plist->Nodesize);        //将栈里面的一个myElemType类型数据拷贝到data指针所指向的内存
14. newpt =  NewChainNode(data);
15. **if**( !newpt )        **return** 0;               //防止指针为空
17. plist->tail->next = newpt;                //将新建的节点添加到链表最后面
18. plist->tail = newpt;
19. **return** 1;
20. }

 代码如下

list.h

**[cpp]** [view plaincopy](http://blog.csdn.net/lilien1010/article/details/7786164)

1. **typedef**  **void** \* ElemType;

4. **typedef** **struct** node
5. {
6. ElemType data;
7. **struct** node \* next;
9. }ChainNode;
11. **typedef** **struct**
12. {
13. ChainNode \*head;
15. **int** Nodesize;
17. ChainNode \*tail;
18. }List;
20. List \* CreateList(**int**); /\* 创建链表 \*/
22. **void** DestoryList(List\*);    /\* 销毁链表 \*/
24. **void** ClearList(List\*);      /\* 清空链表 \*/
26. **int** ListAppend(List\*,...);      /\* 追加元素 \*/
28. **int** ListInsert(List\*,**int**,...);      /\* 加入元素 \*/
30. **int** ListDelete(List \*,**int**);     /\* 删除第几个元素 \*/
32. **int** GetElem(List\*,**int**,ElemType);  /\* 取得第几个元素的值用第三个参数返回 \*/
34. ChainNode \* GetAddr(List \*, **int**);   /\* 取得编号为N的元素所在地址 \*/
36. **int** TraverseList(List\*,**int** (\*)(ElemType )); /\* 遍历访问，反问某个节点元素用函数处理 \*/
38. ChainNode \* NewChainNode( ElemType);
40. List \*CreateList(**int** size )
41. {
42. List \* pt = 0;
43. ElemType data = 0;
44. pt=(List\*)malloc( **sizeof**(List) );
46. **if**( !pt )
47. **return** 0;
48. pt->head = NewChainNode(data );
49. **if**( ! pt->head )
50. {
51. free(pt);
52. **return** 0;
53. }
55. pt->Nodesize = size;
57. pt->tail = pt->head;
58. **return** pt;
59. }
60. /\*==================\*/
61. **void** DestoryList(List \* plist)
62. {
63. ClearList(plist);
65. free(plist->head);
66. plist->head = 0;
68. free(plist);
69. plist = 0;
71. }
73. /\*==================\*/
75. **int** ListAppend(List \* plist,...)
76. {
77. ChainNode \* newpt = 0;
78. **void** \* data;
79. **void** \* pos;
80. pos = &plist + 1;
82. **if**( !(plist && plist->head) )
83. **return** 0;
85. data = (**void** \*)malloc(plist->Nodesize);
86. **if**( !data )
87. **return** 0;
89. memcpy(data,pos,plist->Nodesize);
90. newpt =  NewChainNode(data);
92. **if**( !newpt )
93. **return** 0;
95. plist->tail->next = newpt;
96. plist->tail = newpt;
97. **return** 1;
98. }
100. /\*==================\*/
101. **int** ListInsert(List \* plist, **int** n ,...)
102. {
103. ChainNode \* pt = 0;
104. ChainNode \* newpt = 0;
105. **void** \* data;
106. **void** \*pos = &plist + 2;
107. pt = GetAddr( plist, n-1 );
108. **if**( !(pt) )
109. **return** 0;
110. data = (**void**\*)malloc(plist->Nodesize);
112. **if**( !data )
113. **return** 0;
114. memcpy(data,pos,plist->Nodesize);
116. newpt = NewChainNode(data);
117. **if**( !newpt )
118. **return** 0;
120. **if** ( pt->next == plist->tail )
121. plist->tail = newpt;
123. newpt->next = pt->next;
125. pt->next = newpt;
127. **return** 1;
129. }
131. /\*==================\*/
132. **int** GetElem(List \* plist,**int** n,ElemType data)
133. {
134. ChainNode \* pt = 0;

137. **if**( !data )
138. **return** 0;
140. pt = GetAddr(plist,n);
141. **if**( ! pt )
142. **return** 0;
144. memcpy(data, pt->data ,plist->Nodesize);
146. **return** 1;
147. }
149. /\*==================\*/
150. **int** TraverseList(List\* plist,**int** (\*f)(ElemType ))
151. {
152. ChainNode \* pt = 0;
153. **int** a=0;
155. /\*\*/
156. **if**( !(plist && plist->head) )
157. **return** 0;
158. **for**( a = 0 ,pt = plist->head->next; pt ; pt = pt->next )
159. {
160. **if**( ! f( (pt->data)) )
161. **return** a+1;
162. a++;
163. }
164. **return** 0;
165. }
167. /\*==================\*/
168. **void** ClearList(List \* plist)
169. {
170. **while** ( ListDelete(plist,1) );
171. }
173. /\*==================\*/
174. **int** ListDelete( List \* plist,  **int** n )
175. {
176. ChainNode \* pt =0;
177. ChainNode \* pf=0;
179. **if**( !plist->head->next )
180. **return** 0;
182. pt = GetAddr( plist,n-1 );
184. **if** ( pt->next == plist->tail )
185. plist->tail = pt;
187. **if**( !(pt && pt->next ))
188. **return** 0;
190. pf = pt->next;
191. pt->next = pt->next->next;
193. free(pf->data);
194. free(pf);
196. **return** 1;
198. }
200. ChainNode \* GetAddr(List \* plist,**int** n)
201. {
202. ChainNode \* pt = 0;
203. **int** a = 0;
205. **if**( n < 0)   **return** 0;
207. pt = plist->head;
209. **while**( pt && a < n )
210. {
211. pt = pt->next;
212. a++;
213. }
214. **return** pt;
215. }
217. /\*==================\*/
218. ChainNode \* NewChainNode(ElemType data)
219. {
220. ChainNode \* pChain=0;
221. pChain = ( ChainNode \* )malloc( **sizeof**(ChainNode) );
223. **if**( ! pChain )  **return** 0;
225. pChain->data=data;
226. pChain->next=0;
228. **return** pChain;
229. }

uselist.c

**[cpp]** [view plaincopy](http://blog.csdn.net/lilien1010/article/details/7786164)

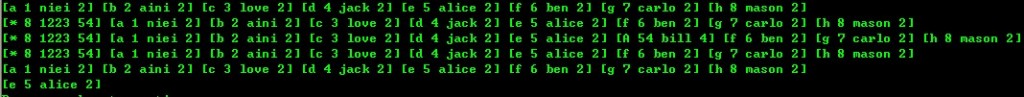
1. #include "list.h"
2. /\*提供两种数据测试\*/
3. **typedef**   **struct** {**char** ch ; **int** id; **char** name[10]; **int** r;} myElemType;
5. /\*
6. typedef  char myElemType;
7. \*/
9. myElemType a[20] ={{'a',1,"niei",2},{'b',2,"aini",2},{'c',3,"love",2},{'d',4,"jack",2},{'e',5,"alice",2},{'f',6,"ben",2},{'g',7,"carlo",2},{'h',8,"mason",2}};
11. /\*
12. myElemType a[20]="Hello world!";
13. \*/
15. **void** showList(List\* );
17. **int** putElem( myElemType \*);
18. **void** main()
19. {
20. List \* mylist;
21. **int** n=0;
22. myElemType data;
23. myElemType data2;
25. myElemType\* pdata;
26. mylist = CreateList( **sizeof**(myElemType) );
28. **if**( ! mylist)
29. {
30. printf("error");
31. **return**;
32. }
33. **for**( n = 0 ;n < 8 ;n++)
34. ListAppend(mylist ,a[n]);
36. showList( mylist);
38. data.ch = '\*';
39. data.id = 8;
40. strcpy(data.name , "1223");
41. data.r = 2;

44. /\*  a[0]='E';
45. a[1]='r';
46. a[2]='r';
47. a[3]='o';
48. a[4]='r';
49. \*/  ListInsert(mylist,1,data);
50. showList( mylist);
52. /\*\*/    data2.ch = 'A';
53. data2.id = 54;
54. strcpy(data2.name , "bill");
55. data2.r = 4;
56. ListInsert(mylist,7,data2);
57. showList( mylist);
59. ListDelete(mylist,7);
60. showList( mylist);

63. ListDelete(mylist,1);
64. showList( mylist);
66. **if** (GetElem(mylist,5,&data2) )
67. printf("[%c %d %s %d] ",data2.ch,data2.id,data2.name,data2.r);

70. ClearList(mylist);
71. showList( mylist);
73. DestoryList(mylist);
74. mylist = 0;
76. showList( mylist);
78. }
79. /\*==================\*/
80. **void** showList(List\* plist)
81. {
82. **if**( !plist  )
83. **return**;
84. TraverseList(plist,(**int**(\*)(**void** \*))putElem);
85. printf("\n");
86. }
87. /\*输出字符\*/
88. /\*
89. int putElem(myElemType \*data)
90. {
92. if( ! ( data) )
93. return 0;
95. printf("%c",\*data);
96. return 1;
97. }
98. \*/
99. /\*输出结构体\*/
100. /\*\*/
101. **int** putElem(myElemType \*data)
102. {
104. **if**( ! ( data) )
105. **return** 0;
106. printf("[%c %d %s %d] ",(data)->ch,(data)->id,(data)->name,(data)->r);
108. **return** 1;
109. }

最后的运行结果，一个[ ] 表示一个结构体所有成员



**比较：**

**体会两种方式的异同，两种实现方式各有优异，分析两种实现方法的优缺点。**

**两种方式在本质上是一样的，只不过第一种方式用的是传址，第二种方式用的是传值。**