# C语言综合研究与高强度程序设计训练18

仔细研究下面的程序 list，思考、研究相关问题。

实现一个线性表，其存储结构为带头结点的单链表。

线性表是数据的排列关系如下的数据集合:

1. 除第一个元素与最后一个元素外，集合中的每一个元素都有前一个元素与后一个元素:
2. 第一个元素没有前一个元素，最后一个元素没有后一个元素。

下面是两个线性表，表中的元素分别是字符、整数:

a、b、c、d、e、f、g、h

11、22、33、44、55、66、77、88

线性表在计算机中可以用不同的方式存储，线性表在计算机中的存储方式，称为线性表的存储结构。

单链表，是线性表的一种存储结构。

我们可以用如图3.1所描述的单链表来存储线性表List

线性表List:a、b、c、d、e、f、g、h

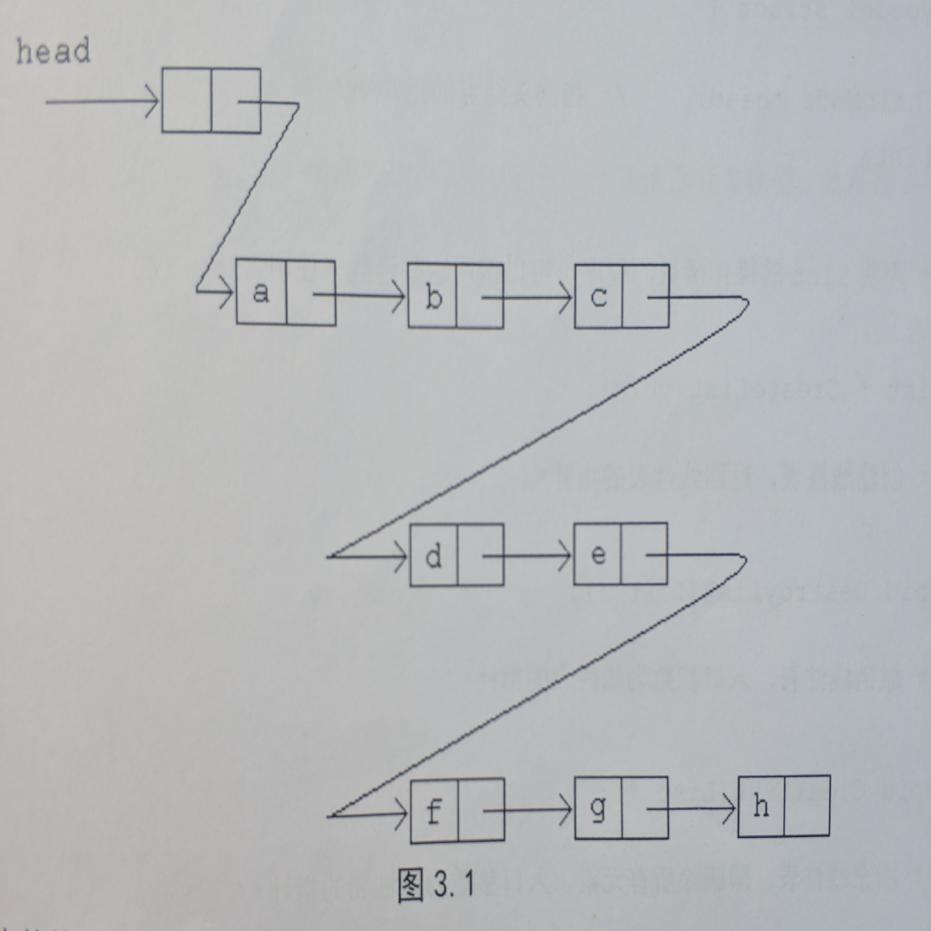
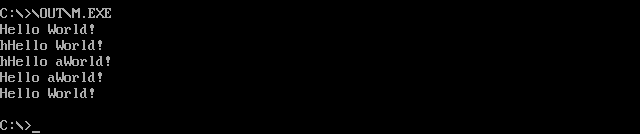
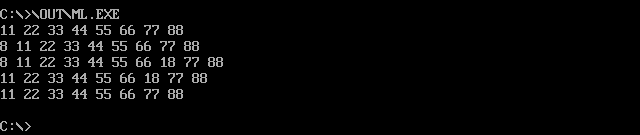
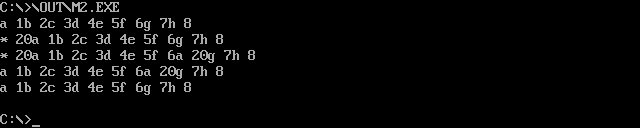
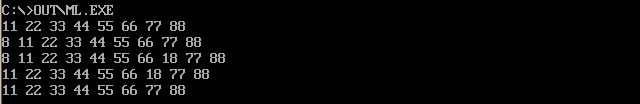


图3.1中的单链表的特性如下

1. 链表中的结点包含一个数据项和（一个指向下一个结点的）指针项
2. 链表有一个头结点，头结点一般不用于存储元素内容，头结点的指针项指向存储线性表第一个元素的结点
3. 线性表为空时，链表中没有存储元素的结点，头结点的指针项为空（可以用0表示空）:线性表不为空时，链表中存储最后一个元素的指针项为空，表示后面再没有元素
4. 线性表的元素从1开始编号，存储结构中，结点可从0开始编号，头结点的编号为0.存储第一个元素的结点编号为1，存储第n个元素的结点编号为n
5. 由一个head指针指向头结点

* 线性表程序 list
* /\* 定义线性表存储的元素类型 \*/  
  typedef char EleType;  
    
  /\* 定义链表结点类型 \*/  
  typedef struct node {  
   EleType data; /\* 数据项 \*/  
   struct node \*next; /\* 指针项 \*/  
  } ChainNode;  
    
  /\* 定义线性表 \*/  
  typedef struct {  
   ChainNode \*head; /\* 指向头结点的指针· \*/  
  } List;  
    
  /\* 声明Lst的操作函数，程序员可以调用这些函数来使用List \*/  
  /\* 创建线性表，返回线性表的指针 \*/  
  List \*CreatList(void);  
  /\* 撤销线性表，入口参数为线性表的指针 \*/  
  void DestoryList(List \*);  
  /\* 清空线性表，即删除所有元素。入口参数为线性表的指针 \*/  
  void ClearList(List \*);  
  /\* 追加元素，在线性表最后一个元素的后面加一个元素，入口参数为线性表的指针，和要加的数据。返回操作是否成功，1表示成功，0表示不线功\*/  
  int ListAppend(List \*, EleType);  
  /\* 加入元素，在线性表编号为的元素的面加一个元素，入口参数为线性表的针，编号和要加入的数据，返回操作是否成功，1表示功，表示不功  
  注意:线性表诸元素的编号一般都是编号，从1开始， \*/  
  int ListInsert(List \*, int, EleType);  
  /\* 删除元素，删除编号为的元素人口参数为线性表的指针、编号，回操作是  
  成功，0表示不成功 \*/  
  int ListDelet(List \*, int);  
  /\* 取元素，取编号为的元素的值，入口参数为线性表指针、编号和要存储读取的值的  
  EleType 型内存单元（变量）的地址，返回操作是否成功，1表示成功，0表示不成功 \*/  
  int GetElement(List \*, int, EleType \*);  
  /\* 取编号为的元素所在的结点的地址，入口参数为线性表的指针、编号、返回结点指针，非0表示成功，0表示不成功。\*/  
  ChainNode \*GetAddr(List \*, int);  
  /\* 遍历，依次访问线性表中的每一个元素，每访问一个元素都用某个函数对这个元素进行某种处理遍历了全部元素返回0，否则返回当前访问元素的编号\*/  
  int TraverseList(List \*, int (\*)(EleType \*));  
  /\* 创建一个存储元素的结点。入口参数为存储在新创建结点中的数据元素的值，返回结点指针，非0表示成功，0表示不成功。\*/  
  ChainNode \*NewChainNode(EleType);  
    
  /\* 定义各操作函数 \*/  
  List \*CreatList(void) {  
   List \*p;  
   EleType \*data = 0;  
   p = (List \*)malloc(sizeof(List));  
   if (!p) {  
   return 0;  
   }  
   p->head = NewChainNode(\*data);  
   if (!p->head) {  
   free(p);  
   return 0;  
   }  
   return p;  
  }  
  void DestoryList(List \*lp) {  
   ClearList(lp);  
   free(lp->head);  
   free(lp);  
  }  
  void ClearList(List \*lp) {  
   while (ListDelet(lp, 1))  
   ;  
  }  
  int ListAppend(List \*lp, EleType data) {  
   ChainNode \*p;  
   ChainNode \*newp;  
    
   newp = NewChainNode(data);  
   if (!newp)  
   return 0;  
   for (p = lp->head; p->next; p = p->next)  
   ;  
   p->next = newp;  
   return 1;  
  }  
  int ListInsert(List \*lp, int n, EleType data) {  
   ChainNode \*p;  
   ChainNode \*newp;  
    
   p = GetAddr(lp, n - 1);  
   if (!p)  
   return 0;  
   newp = NewChainNode(data);  
   if (!newp)  
   return 0;  
   newp->next = p->next;  
   p->next = newp;  
    
   return 1;  
  }  
  int ListDelet(List \*lp, int n) {  
   ChainNode \*p;  
   ChainNode \*pl;  
    
   if (!lp->head->next)  
   return 0;  
    
   p = GetAddr(lp, n - 1);  
   if (!(p && p->next))  
   return 0;  
    
   pl = p->next;  
   p->next = p->next->next;  
    
   free(pl);  
    
   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->next; 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;  
    
   if (n < 0)  
   return 0;  
    
   p = lp->head;  
   a = 0;  
    
   while (p && a < n) {  
   p = p->next;  
   a++;  
   }  
    
   return p;  
  }
* m.c 对 list.h进行测试
* #include "list.h"  
    
  char a[20] = "Hello World!";  
  void showlist(List \*);  
  int putelement(EleType \*);  
    
  main() {  
   List \*lp;  
   int n;  
    
   lp = CreatList();  
   if (!lp) {  
   printf("Creat?\n");  
   return;  
   }  
    
   for (n = 0; a[n]; n++) {  
   ListAppend(lp, a[n]);  
   }  
   showlist(lp);  
    
   ListInsert(lp, 1, 'h');  
   showlist(lp);  
   ListInsert(lp, 8, 'a');  
   showlist(lp);  
    
   ListDelet(lp, 1);  
   showlist(lp);  
   ListDelet(lp, 7);  
   showlist(lp);  
  }  
    
  void showlist(List \*lp) {  
   TraverseList(lp, putelement);  
   printf("\n");  
  }  
    
  int putelement(EleType \*data) {  
   printf("%c", \*data);  
   return 1;  
  }
* 
* 将list.h的第一个语句 typedef char EleType改为 typedef int EleType，然后用下面的程序  
  ml.c进行测试。
  + ml.c
  + #include "list.h"  
      
    char a[8] = {11, 22, 33, 44, 55, 66, 77, 88};  
    void showlist(List \*);  
    int putelement(EleType \*);  
      
    main() {  
     List \*lp;  
     int n;  
      
     lp = CreatList();  
     if (!lp) {  
     printf("Creat?\n");  
     return;  
     }  
      
     for (n = 0; n < 8; n++) {  
     ListAppend(lp, a[n]);  
     }  
     showlist(lp);  
      
     ListInsert(lp, 1, 8);  
     showlist(lp);  
     ListInsert(lp, 8, 18);  
     showlist(lp);  
      
     ListDelet(lp, 1);  
     showlist(lp);  
     ListDelet(lp, 7);  
     showlist(lp);  
    }  
      
    void showlist(List \*lp) {  
     TraverseList(lp, putelement);  
     printf("\n");  
    }  
      
    int putelement(EleType \*data) {  
     printf("%d ", \*data);  
     return 1;  
    }
  + 
* 将list.h的第一个语句 typedef char EleType改为typedef struct{char s;int b;} EleType然后用下面的程序m2.c进行测试。
  + m2.c
  + #include "list.h"  
      
    EleType a[8] = {'a', 1, 'b', 2, 'c', 3, 'd', 4, 'e', 5, 'f', 6, 'g', 7, 'h', 8};  
    void showlist(List \*);  
    int putelement(EleType \*);  
      
    main() {  
     List \*lp;  
     int n;  
     EleType data;  
      
     lp = CreatList();  
     if (!lp) {  
     printf("Creat?\n");  
     return;  
     }  
      
     for (n = 0; n < 8; n++) {  
     ListAppend(lp, a[n]);  
     }  
     showlist(lp);  
      
     data.a = '\*';  
     data.b = 20;  
     ListInsert(lp, 1, data);  
     showlist(lp);  
     data.a = 'a';  
     data.b = 20;  
     ListInsert(lp, 8, data);  
     showlist(lp);  
      
     ListDelet(lp, 1);  
     showlist(lp);  
     ListDelet(lp, 7);  
     showlist(lp);  
    }  
      
    void showlist(List \*lp) {  
     TraverseList(lp, putelement);  
     printf("\n");  
    }  
      
    int putelement(EleType \*data) {  
     printf("%c %d", data->a, data->b);  
     return 1;  
    }
  + 

## 思考研究

* 1. List只有一个数据项“ChainNode＊head”，为什么还要定义这个数据类型？
     1. 方便理解与操作
  2. 改进list.h中线性表的实现方法，用一个tail指针指向链表的最后一个结点，提高  
     ListAppend的速度。用m.c对其进行测试。
     + list.h
     + /\* 定义线性表存储的元素类型 \*/  
       /\* typedef char EleType; \*/  
       typedef int EleType;  
       /\* typedef struct {  
        char a;  
        char b;  
       } EleType; \*/  
       /\* 定义链表结点类型 \*/  
       typedef struct node {  
        EleType data; /\* 数据项 \*/  
        struct node \*next; /\* 指针项 \*/  
       } ChainNode;  
         
       /\* 定义线性表 \*/  
       typedef struct {  
        ChainNode \*head; /\* 指向头结点的指针· \*/  
        ChainNode \*tail; /\* 指向为结点的指针 \*/  
       } List;  
         
       /\* 声明Lst的操作函数，程序员可以调用这些函数来使用List \*/  
       /\* 创建线性表，返回线性表的指针 \*/  
       List \*CreatList(void);  
       /\* 撤销线性表，入口参数为线性表的指针 \*/  
       void DestoryList(List \*);  
       /\* 清空线性表，即删除所有元素。入口参数为线性表的指针 \*/  
       void ClearList(List \*);  
       /\* 追加元素，在线性表最后一个元素的后面加一个元素，入口参数为线性表的指针，和要加的数据。返回操作是否成功，1表示成功，0表示不线功\*/  
       int ListAppend(List \*, EleType);  
       /\* 加入元素，在线性表编号为的元素的面加一个元素，入口参数为线性表的针，编号和要加入的数据，返回操作是否成功，1表示功，表示不功  
       注意:线性表诸元素的编号一般都是编号，从1开始， \*/  
       int ListInsert(List \*, int, EleType);  
       /\* 删除元素，删除编号为的元素人口参数为线性表的指针、编号，回操作是  
       成功，0表示不成功 \*/  
       int ListDelet(List \*, int);  
       /\* 取元素，取编号为的元素的值，入口参数为线性表指针、编号和要存储读取的值的  
       EleType 型内存单元（变量）的地址，返回操作是否成功，1表示成功，0表示不成功 \*/  
       int GetElement(List \*, int, EleType \*);  
       /\* 取编号为的元素所在的结点的地址，入口参数为线性表的指针、编号、返回结点指针，非0表示成功，0表示不成功。\*/  
       ChainNode \*GetAddr(List \*, int);  
       /\* 遍历，依次访问线性表中的每一个元素，每访问一个元素都用某个函数对这个元素进行某种处理遍历了全部元素返回0，否则返回当前访问元素的编号\*/  
       int TraverseList(List \*, int (\*)(EleType \*));  
       /\* 创建一个存储元素的结点。入口参数为存储在新创建结点中的数据元素的值，返回结点指针，非0表示成功，0表示不成功。\*/  
       ChainNode \*NewChainNode(EleType);  
         
       /\* 定义各操作函数 \*/  
       List \*CreatList(void) {  
        List \*p = 0;  
        EleType \*data = 0;  
        p = (List \*)malloc(sizeof(List));  
        if (!p) {  
        return 0;  
        }  
        p->head = NewChainNode(\*data);  
        if (!p->head) {  
        free(p);  
        return 0;  
        }  
        p->tail = p->head;  
        return p;  
       }  
       void DestoryList(List \*lp) {  
        ClearList(lp);  
        free(lp->head);  
        lp->head = 0;  
        free(lp);  
        lp = 0;  
       }  
       void ClearList(List \*lp) {  
        while (ListDelet(lp, 1))  
        ;  
       }  
       int ListAppend(List \*lp, EleType data) {  
        ChainNode \*newp;  
         
        newp = NewChainNode(data);  
        if (!newp)  
        return 0;  
         
        newp->next = lp->tail->next;  
        lp->tail->next = newp;  
        lp->tail = newp;  
         
        return 1;  
       }  
       int ListInsert(List \*lp, int n, EleType data) {  
        ChainNode \*p;  
        ChainNode \*newp;  
         
        p = GetAddr(lp, n - 1);  
        if (!p)  
        return 0;  
        newp = NewChainNode(data);  
        if (!newp)  
        return 0;  
        newp->next = p->next;  
        p->next = newp;  
         
        return 1;  
       }  
       int ListDelet(List \*lp, int n) {  
        ChainNode \*p;  
        ChainNode \*pl;  
         
        if (!lp->head->next)  
        return 0;  
         
        p = GetAddr(lp, n - 1);  
         
        if (p->next == lp->tail)  
        lp->tail = p;  
         
        if (!(p && p->next))  
        return 0;  
         
        pl = p->next;  
        p->next = p->next->next;  
         
        free(pl);  
         
        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->next; 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;  
         
        if (n < 0)  
        return 0;  
         
        p = lp->head;  
        a = 0;  
         
        while (p && a < n) {  
        p = p->next;  
        a++;  
        }  
         
        return p;  
       }
     + ml.c
     + #include "list1.h"  
         
       int a[8] = {11, 22, 33, 44, 55, 66, 77, 88};  
       void showlist(List \*);  
       int putelement(EleType \*);  
         
       main() {  
        List \*lp;  
        int n;  
         
        lp = CreatList();  
        if (!lp) {  
        printf("Creat?\n");  
        return;  
        }  
         
        for (n = 0; n < 8; n++) {  
        ListAppend(lp, a[n]);  
        }  
        showlist(lp);  
         
        ListInsert(lp, 1, 8);  
        showlist(lp);  
        ListInsert(lp, 8, 18);  
        showlist(lp);  
         
        ListDelet(lp, 1);  
        showlist(lp);  
        ListDelet(lp, 7);  
        showlist(lp);  
       }  
         
       void showlist(List \*lp) {  
        TraverseList(lp, putelement);  
        printf("\n");  
       }  
         
       int putelement(EleType \*data) {  
        printf("%d ", \*data);  
        return 1;  
       }
     + 结果验证
     + 
  3. 是怎样的设计保证了线性表实现方法的改动，没有引起使用线性表的程序（比如:m.c）  
     的改动？
     1. 通过使用头文件使得定义和使用分开
     2. 通过使用 typedef来修改链表节点中数据的类型

1. 为什么要使用一个头结点？
   1. 方便操作
2. 设计、实现一个不限元素数据类型的通用链表。
   * list.c
   * /\* 定义线性表存储的元素类型 \*/  
     /\* typedef char EleType; \*/  
     /\* typedef int EleType; \*/  
     /\* typedef struct {  
      char a;  
      char b;  
     } EleType; \*/  
     typedef void \*EleType;  
     /\* 定义链表结点类型 \*/  
     typedef struct node {  
      EleType data; /\* 数据项 \*/  
      struct node \*next; /\* 指针项 \*/  
     } ChainNode;  
       
     /\* 定义线性表 \*/  
     typedef struct {  
      ChainNode \*head; /\* 指向头结点的指针· \*/  
      ChainNode \*tail; /\* 指向为结点的指针 \*/  
      int size; /\* 数据类型大小 \*/  
     } List;  
       
     /\* 声明Lst的操作函数，程序员可以调用这些函数来使用List \*/  
     /\* 创建线性表，返回线性表的指针 \*/  
     List \*CreatList(int);  
     /\* 撤销线性表，入口参数为线性表的指针 \*/  
     void DestoryList(List \*);  
     /\* 清空线性表，即删除所有元素。入口参数为线性表的指针 \*/  
     void ClearList(List \*);  
     /\* 追加元素，在线性表最后一个元素的后面加一个元素，入口参数为线性表的指针，和要加的数据。返回操作是否成功，1表示成功，0表示不线功\*/  
     int ListAppend(List \*, EleType);  
     /\* 加入元素，在线性表编号为的元素的面加一个元素，入口参数为线性表的针，编号和要加入的数据，返回操作是否成功，1表示功，表示不功  
     注意:线性表诸元素的编号一般都是编号，从1开始， \*/  
     int ListInsert(List \*, int, EleType);  
     /\* 删除元素，删除编号为的元素人口参数为线性表的指针、编号，回操作是  
     成功，0表示不成功 \*/  
     int ListDelet(List \*, int);  
     /\* 取元素，取编号为的元素的值，入口参数为线性表指针、编号和要存储读取的值的  
     EleType 型内存单元（变量）的地址，返回操作是否成功，1表示成功，0表示不成功 \*/  
     int GetElement(List \*, int, EleType \*);  
     /\* 取编号为的元素所在的结点的地址，入口参数为线性表的指针、编号、返回结点指针，非0表示成功，0表示不成功。\*/  
     ChainNode \*GetAddr(List \*, int);  
     /\* 遍历，依次访问线性表中的每一个元素，每访问一个元素都用某个函数对这个元素进行某种处理遍历了全部元素返回0，否则返回当前访问元素的编号\*/  
     int TraverseList(List \*, int (\*)(EleType));  
     /\* 创建一个存储元素的结点。入口参数为存储在新创建结点中的数据元素的值，返回结点指针，非0表示成功，0表示不成功。\*/  
     ChainNode \*NewChainNode(EleType);  
       
     /\* 定义各操作函数 \*/  
     List \*CreatList(int size) {  
      List \*p = 0;  
      EleType data = 0;  
       
      p = (List \*)malloc(sizeof(List));  
      if (!p) {  
      return 0;  
      }  
        
      p->head = NewChainNode(data);  
      if (!p->head) {  
      free(p);  
      return 0;  
      }  
      p->head->next = 0;  
      p->tail = p->head;  
      p->size = size;  
      return p;  
     }  
     void DestoryList(List \*lp) {  
      ClearList(lp);  
      free(lp->head);  
      lp->head = 0;  
      free(lp);  
      lp = 0;  
     }  
     void ClearList(List \*lp) {  
      while (ListDelet(lp, 1))  
      ;  
     }  
     int ListAppend(List \*lp, EleType data) {  
      ChainNode \*newp;  
      EleType elem;  
       
      elem = (void \*)malloc(lp->size);  
      if (!elem)  
      return 0;  
      memcpy(elem, data, lp->size);  
       
      newp = NewChainNode(elem);  
      if (!newp)  
      return 0;  
       
      newp->next = lp->tail->next;  
      lp->tail->next = newp;  
      lp->tail = newp;  
       
      return 1;  
     }  
     int ListInsert(List \*lp, int n, EleType data) {  
      ChainNode \*p;  
      ChainNode \*newp;  
      EleType elem;  
       
      p = GetAddr(lp, n - 1);  
      if (!p)  
      return 0;  
       
      elem = (void \*)malloc(lp->size);  
      if (!elem)  
      return 0;  
      memcpy(elem, data, lp->size);  
      newp = NewChainNode(elem);  
       
      if (!newp)  
      return 0;  
      newp->next = p->next;  
      p->next = newp;  
       
      return 1;  
     }  
     int ListDelet(List \*lp, int n) {  
      ChainNode \*p;  
      ChainNode \*pl;  
       
      if (!lp->head->next)  
      return 0;  
       
      p = GetAddr(lp, n - 1);  
       
      if (p->next == lp->tail)  
      lp->tail = p;  
       
      if (!(p && p->next))  
      return 0;  
       
      pl = p->next;  
      p->next = p->next->next;  
       
      free(pl->data);  
      free(pl);  
       
      return 1;  
     }  
       
     int GetElement(List \*lp, int n, EleType \*data) {  
      ChainNode \*p;  
       
      if (!data)  
      return 0;  
       
      p = GetAddr(lp, n);  
      if (!p)  
      return 0;  
       
      memcpy(data, p->data, lp->size);  
       
      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;  
     }  
     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;  
       
      if (n < 0)  
      return 0;  
       
      p = lp->head;  
      a = 0;  
       
      while (p && a < n) {  
      p = p->next;  
      a++;  
      }  
       
      return p;  
     }
   * ml.c
   * #include "list5.h"  
     typedef struct student {  
      int number;  
      int math;  
      int c;  
      int java;  
     } stu;  
     typedef stu MyElementType;  
     MyElementType sa[4] = {1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4};  
     void showlist(List \*);  
     int putelement(MyElementType \*);  
       
     main() {  
      List \*lp;  
      int n;  
      stu a = {5, 5, 5, 5};  
      stu b = {8, 8, 8, 8};  
       
      lp = CreatList(sizeof(MyElementType));  
      if (!lp) {  
      printf("Creat?\n");  
      return;  
      }  
       
      for (n = 0; n < 4; n++) {  
      ListAppend(lp, &(sa[n]));  
      }  
      showlist(lp);  
       
      ListInsert(lp, 1, &a);  
      showlist(lp);  
      ListInsert(lp, 4, &b);  
      showlist(lp);  
       
      ListDelet(lp, 1);  
      showlist(lp);  
      ListDelet(lp, 3);  
      showlist(lp);  
     }  
       
     void showlist(List \*lp) {  
      TraverseList(lp, (int (\*)(void \*))putelement);  
      printf("\n");  
     }  
       
     int putelement(MyElementType \*element) {  
      printf("%d %d %d %d -- ", element->number, element->math, element->c,  
      element->java);  
      return 1;  
     }
   * 结果验证
   * 