**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： 数据结构与数据库技术

实验名称： 实验一

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年10月9日

实验报告

## 【实验目的】

1.复习C语言相关内容，熟练掌握结构体、指针、malloc函数的使用。

2.加深对抽象数据类型ADT表的理解。

## 【实验要求】

参照课本p.44-49，及Figure3.6-3.13，编写一个ADT表，要求能判断表是否为空，获取第i个节点的内容，删除，插入。

## 【实验具体内容】

1.实验库函数List.c源代码：

#include "list.h"

#include <stdlib.h>

#include "fatal.h"

/\* Place in the interface file \*/

struct Node

{

    ElementType Element;

    Position Next;

};

List MakeEmpty(List L)

{

    if (L != NULL)

        DeleteList(L);

    L = malloc(sizeof(struct Node));

    if (L == NULL)

        FatalError("Out of memory!");

    L->Next = NULL;

    return L;

}

/\* START: fig3\_8.txt \*/

/\* Return true if L is empty \*/

int IsEmpty(List L)

{

    return L->Next == NULL;

}

/\* END \*/

/\* START: fig3\_9.txt \*/

/\* Return true if P is the last position in list L \*/

/\* Parameter L is unused in this implementation \*/

int IsLast(Position P, List L)

{

    return P->Next == NULL;

}

/\* END \*/

/\* START: fig3\_10.txt \*/

/\* Return Position of X in L; NULL if not found \*/

Position

Find(ElementType X, List L)

{

    Position P;

    /\* 1\*/ P = L->Next;

    /\* 2\*/ while (P != NULL && P->Element != X)

        /\* 3\*/ P = P->Next;

    /\* 4\*/ return P;

}

/\* END \*/

/\* START: fig3\_11.txt \*/

/\* Delete from a list \*/

/\* Cell pointed to by P->Next is wiped out \*/

/\* Assume that the position is legal \*/

/\* Assume use of a header node \*/

void Delete(ElementType X, List L)

{

    Position P, TmpCell;

    P = FindPrevious(X, L);

    if (!IsLast(P, L)) /\* Assumption of header use \*/

    {                  /\* X is found; delete it \*/

        TmpCell = P->Next;

        P->Next = TmpCell->Next; /\* Bypass deleted cell \*/

        free(TmpCell);

    }

}

/\* END \*/

/\* START: fig3\_12.txt \*/

/\* If X is not found, then Next field of returned value is NULL \*/

/\* Assumes a header \*/

Position

FindPrevious(ElementType X, List L)

{

    Position P;

    /\* 1\*/ P = L;

    /\* 2\*/ while (P->Next != NULL && P->Next->Element != X)

        /\* 3\*/ P = P->Next;

    /\* 4\*/ return P;

}

/\* END \*/

/\* START: fig3\_13.txt \*/

/\* Insert (after legal position P) \*/

/\* Header implementation assumed \*/

/\* Parameter L is unused in this implementation \*/

void Insert(ElementType X, List L, Position P)

{

    Position TmpCell;

    /\* 1\*/ TmpCell = malloc(sizeof(struct Node));

    /\* 2\*/ if (TmpCell == NULL)

        /\* 3\*/ FatalError("Out of space!!!");

    /\* 4\*/ TmpCell->Element = X;

    /\* 5\*/ TmpCell->Next = P->Next;

    /\* 6\*/ P->Next = TmpCell;

}

/\* END \*/

#if 0

/\* START: fig3\_14.txt \*/

        /\* Incorrect DeleteList algorithm \*/

        void

        DeleteList( List L )

        {

            Position P;

/\* 1\*/      P = L->Next;  /\* Header assumed \*/

/\* 2\*/      L->Next = NULL;

/\* 3\*/      while( P != NULL )

            {

/\* 4\*/          free( P );

/\* 5\*/          P = P->Next;

            }

        }

/\* END \*/

#endif

/\* START: fig3\_15.txt \*/

/\* Correct DeleteList algorithm \*/

void DeleteList(List L)

{

    Position P, Tmp;

    /\* 1\*/ P = L->Next; /\* Header assumed \*/

    /\* 2\*/ L->Next = NULL;

    /\* 3\*/ while (P != NULL)

    {

        /\* 4\*/ Tmp = P->Next;

        /\* 5\*/ free(P);

        /\* 6\*/ P = Tmp;

    }

}

/\* END \*/

Position

Header(List L)

{

    return L;

}

Position

First(List L)

{

    return L->Next;

}

Position

Advance(Position P)

{

    return P->Next;

}

ElementType

Retrieve(Position P)

{

    return P->Element;

}

/\*my functions\*/

List getList()

{

    struct Node \*headNode, \*oneNode;

    int nodeCounts = 0;

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

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

    headNode->Element = 0;

    headNode->Next = oneNode;

    printf("请输入要创建的结点数\n");

    scanf\_s("%d", &nodeCounts);

    if (nodeCounts != 0)

    {

        for (int i = 0; i < nodeCounts; i++)

        {

            struct Node \*nextNode;

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

            printf("请输入第%d个结点的值\n", i + 1);

            scanf\_s("%d", &oneNode->Element);

            oneNode->Next = nextNode;

            if (i == (nodeCounts - 1))

            {

                oneNode->Next = NULL;

            }

            oneNode = nextNode;

        }

    }

    else

    {

        headNode->Next = NULL;

    }

    return (headNode);

}

void showList(struct Node \*p)

{

    if (!IsEmpty(p))

    {

        p = p->Next;

    }

    printf("现在的链表如下：");

    for (;;)

    {

        if (!(p->Next))

        {

            printf("%d\n", p->Element);

            break;

        }

        printf("%d -> ", p->Element);

        p = p->Next;

    }

}

void getElement(int m, struct Node \*p)

{

    p = p->Next;

    for (int i = 1; i < m; i++)

    {

        p = p->Next;

    }

    printf("第%d个结点处的值为：%d\n", m, p->Element);

}

void deleteElement(int m, struct Node \*p)

{

    for (int i = 1; i < m; i++)

    {

        p = p->Next;

    }

    p->Next = (p->Next)->Next;

}

void insertElement(int position, int value, struct Node \*p)

{

    struct Node \*oneNode;

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

    oneNode->Element = value;

    for (int i = 0; i < position; i++)

    {

        p = p->Next;

    }

    oneNode->Next = p->Next;

    p->Next = oneNode;

}

2.实验头文件List.h源代码：

typedef int ElementType;

/\* START: fig3\_6.txt \*/

#ifndef \_List\_H

#define \_List\_H

struct Node;

typedef struct Node \*PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

List MakeEmpty(List L);

int IsEmpty(List L);

int IsLast(Position P, List L);

Position Find(ElementType X, List L);

void Delete(ElementType X, List L);

Position FindPrevious(ElementType X, List L);

void Insert(ElementType X, List L, Position P);

void DeleteList(List L);

Position Header(List L);

Position First(List L);

Position Advance(Position P);

ElementType Retrieve(Position P);

/\*my functions\*/

List getList();

void showList(struct Node \*p);

void getElement(int m, struct Node \*p);

void deleteElement(int m, struct Node \*p);

void insertElement(int position, int value, struct Node \*p);

#endif /\* \_List\_H \*/

/\* END \*/

3.实验主文件main.c源代码：

#include <stdio.h>

#include "fatal.h"

#include "list.h"

int main()

{

    int position, value;

    List myList;

    myList = getList();

    showList(myList);

    printf("检验该表是否为空\n");

    printf("%d\n", IsEmpty(myList));

    printf("请问要找的节点是第几个节点？\n");

    scanf\_s("%d", &position);

    getElement(position, myList);

    printf("请问要删除的节点是第几个节点？\n");

    scanf\_s("%d", &position);

    deleteElement(position, myList);

    showList(myList);

    printf("请问要在第几个节点之后插入一个结点？\n");

    scanf\_s("%d", &position);

    printf("请问要插入的节点的值为多少？\n");

    scanf\_s("%d", &value);

    insertElement(position, value, myList);

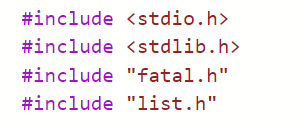
    showList(myList);

    getchar();

}

## 【实验记录与说明分析】

除去课本上的代码，在myfunction内添加了一些自己的函数，通过调用头文件，即可将自己编写的库函数引入



下面对几个函数的实现原理进行说明

（1）getList()函数

1. List getList()
2. {
3. **struct** Node\* headNode, \* oneNode;
4. **int** nodeCounts = 0;
5. headNode = (**struct** Node\*)malloc(**sizeof**(**struct** Node));
6. oneNode = (**struct** Node\*)malloc(**sizeof**(**struct** Node));
7. headNode->Element = 0;
8. headNode->Next = oneNode;
9. printf("请输入要创建的结点数\n");
10. scanf\_s("%d", &nodeCounts);
11. **if** (nodeCounts != 0)
12. {
13. **for** (**int** i = 0; i < nodeCounts; i++)
14. {
15. **struct** Node\* nextNode;
16. nextNode = (**struct** Node\*)malloc(**sizeof**(**struct** Node));
17. printf("请输入第%d个结点的值\n", i + 1);
18. scanf\_s("%d", &oneNode->Element);
19. oneNode->Next = nextNode;
20. **if** (i == (nodeCounts - 1))
21. {
22. oneNode->Next = NULL;
23. }
24. oneNode = nextNode;
25. }
26. }
27. **else**
28. {
29. headNode->Next = NULL;
30. }
31. **return** (headNode);
32. }

该函数先使用malloc函数构建一个头节点，若为空表，则令头节点的Next指针为空。令表头的内容Element为空，每新建一个节点，令前一节点的Next指向这个新的节点，并将内容写入Element，直至最后一个节点，最后一个节点的Next指向NULL即可，再让函数返回头节点的头指针，即可得到一个链表。

（2）showList()函数

1. **void** showList(**struct** Node\* p)
2. {
3. **if** (!IsEmpty(p))
4. {
5. p = p->Next;
6. }
7. printf("现在的链表如下：");
8. **for** (;;)
9. {
10. **if** (!(p->Next))
11. {
12. printf("%d\n", p->Element);
13. **break**;
14. }
15. printf("%d -> ", p->Element);
16. p = p->Next;
17. }
18. }

该函数首先对头节点判断，若表不为空才进行显示，显示直到遇到一个节点的Next为NULL，意味着已经到达最后一个节点，即已完成显示。

（3）getElement()函数

1. **void** deleteElement(**int** m, **struct** Node\* p)
2. {
3. **struct** Node\* oneNode;
4. oneNode = (**struct** Node\*)malloc(**sizeof**(**struct** Node));
5. **for** (**int** i = 1; i < m; i++)
6. {
7. p = p->Next;
8. }
9. p->Next = (p->Next)->Next;
10. oneNode->Next = NULL;
11. oneNode = NULL;
12. }

该函数使用的是遍历的方法，其实现原理较为简单，不再赘述

（4）deleteElement()函数

1. **void** deleteElement(**int** m, **struct** Node\* p)
2. {
3. **for** (**int** i = 1; i < m; i++)
4. {
5. p = p->Next;
6. }
7. p->Next = (p->Next)->Next;
8. }

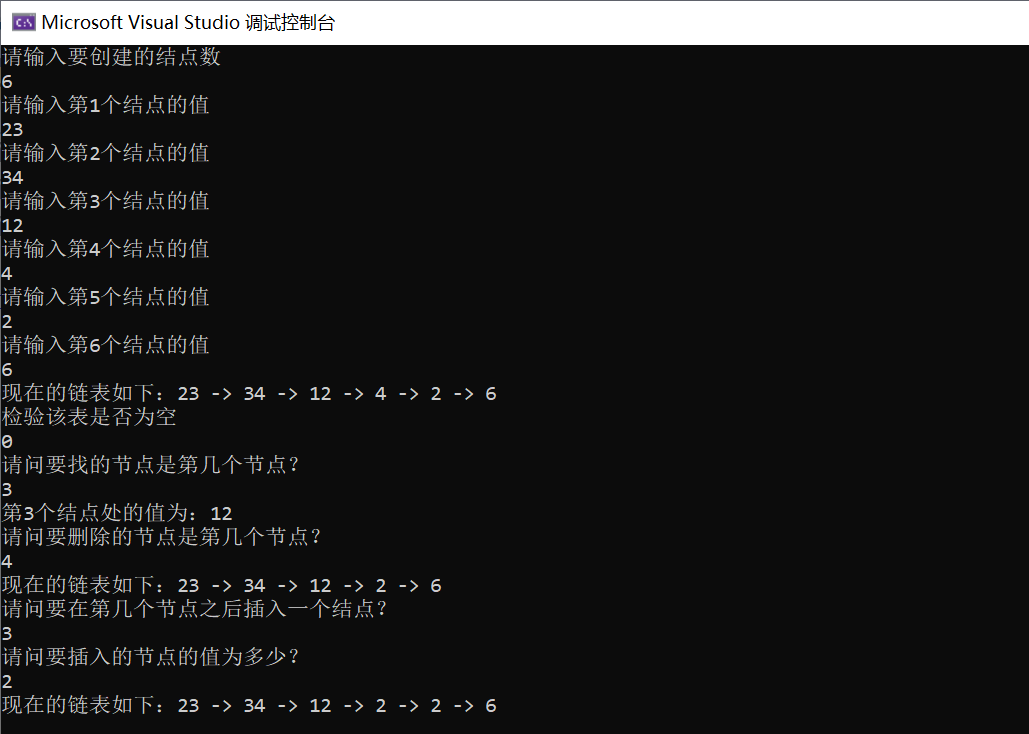
将要删除的节点前一个结点指向删除节点的下一个节点即可。

（5）insetElement()函数

1. **void** insertElement(**int** position, **int** value, **struct** Node\* p)
2. {
3. **struct** Node\* oneNode;
4. oneNode = (**struct** Node\*)malloc(**sizeof**(**struct** Node));
5. oneNode->Element = value;
6. **for** (**int** i = 0; i < position; i++)
7. {
8. p = p->Next;
9. }
10. oneNode->Next = p->Next;
11. p->Next = oneNode;
12. }

新建一个节点，将其插入即可。

下面是运行结果：



可以直观的看出，程序能够实现相应功能。

首先，第一部操作创建了一个6个结点的链表，要注意的是这个链表是包含一个隐藏着的头结点的。

下面将会要求输入六个结点的值，输入后使用showList()函数将链表展示。

然后调用IsEmpty()函数检验链表是否为空，若为空则返回0，不为空则返回1。从图中结果可以看出，返回的是1，这是符合预期的。

下面作寻找结点的操作，寻找第三个结点，成功返回了值12。

下面作删除节点的操作，删除第四个节点后得到了新的链表，可见也可实现功能。

最后作插入操作，插入默认在某节点后作插入，插入后的链表也是符合预期的。

## 【实验心得】

本实验使用的是gcc配合VSCODE进行编译运行，所以如果在VS中使用Clang编译可能会出现问题，需要修改一些include、声明，且printf与scanf需要改成安全形式，程序的内容是没有问题的。

本次对C中的结构体、指针有了更深的了解，让我意识到，其实结构体+指针已经非常类似于java、python等面向对象编程中的类了。本次实验对链表有了直观的理解，它的删除、插入操作非常灵活，与找到一个数的时间复杂度是一致的，而且其存放不需要连续的空间，这决定了它在大数据量的数据库中的广泛应用。

这一次的数据库实验，让我真切地感到了ADT链表的作用之大与神奇，也感到了自己能写出一个链表的成就感。