**《程序设计基础课程设计》第2周实验报告**

班级：1803012

姓名：杨煜

学号：18030100204

所做题目：一、数据结构—— 二选一

**第3题**

**1.原始题目及要求；**

**（1）复杂数据结构-动态链表**

要求：链表是一种重要的数据结构，需要动态的进行存储分配，要求通过函数分别实现动态链表的建立、结点的插入、结点的删除以及链表的输出。

**2.题目的分析**

（1）涉及知识点

内存管理、结构体定义、指针运用、函数

（2）题目功能理解

为了体现链表的结点，需要用结构体实现独立的数据结构。链表实现的重点就在于链表的插入与删除，这些可以通过指针，动态内存申请与释放等完成。同时考虑到实现的方便性，选择使用一个空节点（首结点）表示一个链表。

**3. 题目的总体设计：设计思路、算法描述**

（1）程序模块：

（A）初始化模块

初始化链表,即分配头结点。

（B）插入模块

在链表的某一结点后插入一个结点。

（C）删除模块

删除链表的某一结点后的一个结点。

（D）遍历链表模块

遍历链表并输出所有结点的值。

（E）销毁模块

销毁链表。

（2）模块调用关系

各模块相互独立，无相互调用关系。

（3）输入输出数据说明

输入数据：功能选择，功能所需的位置和值。

输出数据：操作结果，链表打印结果。

（4）总体流程

1）显示功能选择界面，提示用户选择功能，转2。

2）用户输入功能序号，用户选择功能。输入若为‘1’，则转3,；若为‘2’则转4

若为‘3’，则转5；若为‘4’，则转6；若为其他，则提示输入错误，转1。

3）用户输入位置和值，显示插入结果，转1。

4）用户输入位置，显示删除结果，转1。

5）打印出现在链表的内容，转1。

6）销毁链表，结束运行。

**4. 各功能模块/函数的设计说明：**

（1）linked\_list linked\_list\_initialize( )

函数名称: linked\_list\_initialize

函数功能: 初始化链表,即分配头结点

参数: 无

返回值: linked\_list( 表示分配结果 )

执行流程：

1）分配首结点地址空间，若分配成功，转2，否则返回NULL。

2）使得新结点的下一个结点为NULL，转3）

3）返回结点地址。

可能结果：

1）分配失败，返回NULL

2）分配成功，返回链表首地址

（2）bool linked\_list\_insert( linked\_list obj ,size\_t loc, int val )

函数名称: linked\_list\_insert

函数功能: 在链表的某一结点后插入一个结点

参数: linked\_list obj( 链表头结点 ) ,size\_t loc( 要插入的位置的前一个结点的位

置 ), int val( 新结点的值 )

返回值: bool( 表示操作是否成功 )

执行流程：

1）寻找指定的位置，寻找失败，返回false，否则转2。

2）申请新的地址空间，若申请失败，返回false，否则转3。

3）将新结点赋值后插入指定位置，返回true。

可能结果：

1）true，表示插入成功；

2）false，表示插入失败

（3）bool linked\_list\_delete( linked\_list obj ,size\_t loc )

函数名称: linked\_list\_delete

函数功能: 删除链表的某一结点后的一个结点

参数: linked\_list obj( 链表头结点 ) ,

size\_t loc( 要删除的位置的前一个结点的位置 )

返回值: bool( 表示操作是否成功 )

执行流程：

1）寻找指定的位置，寻找失败，返回false，否则转2。

2）若指定位置之后没有元素，返回false，否则转3。

3）改变指针指向并删除元素，返回true。

可能结果：

1）true，表示删除成功。

2）false，表示删除失败。

（4）void linked\_list\_print( linked\_list obj )

函数名称: linked\_list\_printr

函数功能: 遍历链表并输出所有结点的值

参数: linked\_list obj( 链表头结点 )

返回值: void

执行流程：

1）从除头结点外的第一个结点开始打印，直至没有下一个结点。

可能结果：

1）完成输出

（5）void linked\_list\_destory( linked\_list obj )

函数名称: linked\_list\_destory

函数功能: 销毁链表

参数: linked\_list obj( 链表头结点 )

返回值:void

执行流程：

1）每次均删除头结点后的首个结点，直至除头结点外没有结点，转2.

2）释放头结点内存空间。

**5. 源程序：**

（1）linked\_list\_test.c

#include<stdio.h>

#include<stdbool.h>

#include "linked\_list.h"

int main(){

int choice;

linked\_list obj = linked\_list\_initialize();

bool is\_end = false;

while( !is\_end ){

printf("请选择功能\n");

printf("1.插入节点\n2.删除节点\n3.遍历链表\n4.结束程序\n");

scanf("%d", &choice );

switch( choice ){

case 1:{

int loc, val;

printf("请输入要在哪个位置后插入节点\n");

scanf("%d", &loc );

printf("请输入节点节点值\n");

scanf("%d", &val );

if( linked\_list\_insert(obj, loc, val ) ){

printf("插入成功\n");

}

else{

printf("插入失败\n");

}

break;

};

case 2:{

int loc;

printf("请输入要在哪个位置后删除节点\n");

scanf("%d", &loc );

if( linked\_list\_delete(obj, loc) ){

printf("删除成功\n");

}

else{

printf("删除失败\n");

}

break;

};

case 3:{

printf("链表打印结果如下\n");

linked\_list\_print( obj);

break;

}

case 4:{

printf("退出程序\n");

is\_end = true;

break;

};

default:{

printf("不存在此选项,请重新输入\n");

break;

};

}

}

linked\_list\_destory( obj );

return 0;

}

（2）big\_int.h

#ifndef \_\_LINKED\_LIST\_H

#define \_\_LINKED\_LIST\_H

#include<stdbool.h>

#include<malloc.h>

/\*

\* 链表从下标0开始计算下标,但下标为0的结点是用于代表链表的头结点,

\* 其值并无实际意义, 故有效的链表结点下标大于等于1

\*/

typedef struct linked\_list\_node{

int val;//节点值

struct linked\_list\_node\* next;//下一节点

}linked\_list\_node, \*linked\_list;// 定义链表的结点

/\*

\* 函数名称: linked\_list\_initialize

\* 函数功能: 初始化链表,即分配头结点

\* 参数: 无

\* 返回值: linked\_list( 表示分配结果 )

\*/

linked\_list linked\_list\_initialize( ){

linked\_list temp =(linked\_list)malloc( sizeof( linked\_list\_node ) );//

if( !temp ){

return NULL;

}

temp->next = NULL;

return temp;

}

/\*

\* 函数名称: linked\_list\_insert

\* 函数功能: 在链表的某一结点后插入一个结点

\* 参数: linked\_list obj( 链表头结点 ) ,size\_t loc( 要插入的位置的前一个结点的位置 ),

\* int val( 新结点的值 )

\* 返回值: bool( 表示操作是否成功 )

\*/

bool linked\_list\_insert( linked\_list obj ,size\_t loc, int val ){

linked\_list\_node\* cur = obj;

size\_t cnt=0;

while( cur && cnt<loc ){

cur = cur->next;

++cnt;

}

if( cnt<loc || !cur ){

return false;

}

linked\_list temp =(linked\_list)malloc( sizeof( linked\_list\_node ) );

if( !temp ){

return false;

}

temp->val = val;

temp->next = cur->next;

cur->next = temp;

return true;

}

/\*

\* 函数名称: linked\_list\_delete

\* 函数功能: 删除链表的某一结点后的一个结点

\* 参数: linked\_list obj( 链表头结点 ) ,size\_t loc( 要删除的位置的前一个结点的位置 ),

\* 返回值: bool( 表示操作是否成功 )

\*/

bool linked\_list\_delete( linked\_list obj ,size\_t loc ){

linked\_list\_node\* cur = obj;

size\_t cnt=0;

while( cur && cnt<loc ){

cur = cur->next;

++cnt;

}

if( cnt<loc || ( cur && !cur->next ) || !cur ){

return false;

}

linked\_list\_node\* temp = cur->next->next;

free( cur->next );

cur->next = temp;

return true;

}

/\*

\* 函数名称: linked\_list\_print

\* 函数功能: 遍历链表并输出所有结点的值

\* 参数: linked\_list obj( 链表头结点 )

\* 返回值: void

\*/

void linked\_list\_print( linked\_list obj ){

linked\_list\_node\* cur = obj->next;

while( cur ){

printf("%d ",cur->val);

cur = cur->next;

}

printf("\n");

return;

}

/\*

\* 函数名称: linked\_list\_destory

\* 函数功能: 销毁链表

\* 参数: linked\_list obj( 链表头结点 )

\* 返回值: void

\*/

void linked\_list\_destory( linked\_list obj ){

linked\_list\_node\* temp;

while( temp = obj->next ){

obj->next = obj->next->next;

free( temp );

}

free( obj );

return;

}

#endif

**6. 测试数据（输入、输出）：**

见最后一张图

**7. 小结：**

本次实验圆满完成，实验结果符合实验要求。

实验过程中，我复习到了c语言中内存管理、结构体定义、指针运用、函数等重要知识，收货颇丰。

