浙江大学实验报告

专业：混合班
姓名：徐圣泽
学号：3190102721
日期：2020.3.24

课程名称：C程序设计专题　指导老师：翁恺　实验名称：Linked List（链表）

一、实验要求

编写一个存放int的链表，链表用以下的数据类型表示：

```
typedef struct _node Node;

typedef struct {
    Node *head;
    Node *tail;
} List;
```

该库要提供以下API函数：

```
List list_create();//创建一个List，其中的head和tail都是零。
void list_free(List *list);//释放整个链表中全部的结点，list的head和tail置零。
void list_append(List *list, int v);//用v制作新的结点加到链表的最后。
void list_insert(List *list, int v);//用v制作新的结点加到链表的最前面。
void list_set(List *list, int index, int v);//将链表中第index个结点的值置为v，链表结点从零开始编号。
int list_get(List *list, int index);//获得链表中第index个结点的值，第一个结点的index为零。
int list_size(List *list);//给出链表的大小。
int list_find(List *list, int v);//在链表中寻找值为v的结点，返回结点的编号，结点的编号从零开始；如果找不到，返回-1。
void list_remove(List *list, int v);//删除链表中值为v的结点。
void list_iterate(List *list, void (*func)(int v));//遍历链表，依次对每一个结点中的值做func函数。
```

二、实验思路与过程描述

在此次实验中，需要完成`linkedlist.c`和`linkedlist.h`两个文件。

在头文件`linkedlist.h`中，需要声明所有函数名（简单地把所有API函数复制进去即可）并且声明`List`的结构类型。为了防止在同一个编译单元同名结构被重复声明，此时需要用到"标准头文件结构"这个全新知识点的内容，以免被`#include`多次。

在`linkedlist.c`文件中，需要引入库函数，其中包括标准库`#include<stdio.h>`和上述提到的第三方库`#include "linkedlist.h"`，并且要注意不能漏掉`#include<stdlib.h>`（我第一次写的时候就漏掉了，导致了很多麻烦），并在此文件中成所有函数的定义和`Node`的声明。这个文件中大部分函数的定义都曾在W3链表的课程中提及，因此写的时候思路整体比较顺畅（尽管debug时卡了很久）。

三、实验代码解释

(一) `linkedlist.h`

这部分的代码分为几个重点：预处理、声明结构类型、声明函数。关于标准头文件结构的应用是全新的知识点，此前在`linkedlist.h`卡了很长时间，最后发现忘记在结尾加上`#endif`（还是掌握不熟练导致，关于`#endif`的位置仍存在疑惑，发现放置在函数名的前面或后面皆可）。对于结构类型和函数名的声明，都是较为简单的操作。

(二) `linkedlist.c`

在这部分的代码中，首先声明了`Node`：

```
typedef struct _node{
    int value;
    struct _node *next;
}Node;
```

并且补充以下十个函数的定义：

1、`List list_create();`链表的创建

这个函数的主要任务是创建一个链表并初始化链表，是所有函数里难度最低的一个。首先定义`List list`，将`head`和`tail`同时指向NULL，就创造了一个空链表，最后`return list`，注意这里返回的`list`的类型。

2、`void list_free(List *list);`链表的清除

这个函数即链表的清除，跟课堂上所讲到的`clear`函数是异曲同工的。这个函数首先在定义了`Node *p=list->head, *q=NULL`，再用很常见的`for`循环实现了链表的遍历，并`free`了每一个结点，最后将`list->head=NULL,list->tail=NULL;`（这个置零步骤似乎是关键步骤但仍有困惑之处）完成置零。

3、`void list_append(List *list,int v);`在链表最后加入新的结点

这个函数跟课程中所提到的`append_tail`函数差别不大，此处多了一个`list->tail`，但本质差别不大。首先是最关键的一点，给指针分配内存`Node *p=(Node *)malloc(sizeof(Node));`继而`p->value=v;p->next=NULL;`之后分为两种情况进行讨论①`list->head=NULL`（此时为空链表，只需要令`head`和`tail`等于`p`即可）②`list->head!=NULL`（在`tail`后面接上`p`）。

4、`void list_insert(List *list,int v);`在链表最前加入新的结点

这个函数的原理和上一个函数是类似的，前半部分的过程和`append`也是相似的，不过此时`p->next=list->head`。另外，在这个函数中不需要考虑`list->head`是否为`NULL`，直接令`list->head=p`就完成了这部分代码。

5、`void list_set(List *list,int index,int v);`改变链表某位置的值

```
void list_set(List *list, int index, int v){
    int i=0;
    Node*p;
    for(p=list->head;p;p=p->next,i++){
        if(index==i){
            p->value=v;
        }
    }
}
```

在这个函数中，主要方法是定义了一个整形变量`i=0`，再通过`for`循环和`i`的自加，找到`i=index`时的结点，并将`v`赋值给这个结点。

6、`int list_get(List *list,int index);`得到链表某位置的值

这个函数的方法和上述函数的思想是一样的，唯一不同之处在于找到目标结点时，直接返回目标结点处的值`return p->value`。

7、`int list_size(List *list);`给出链表的大小

这个函数的目的是给出链表的大小。定义整形变量`len=0`，遍历整个链表，在每个结点处`len++`，最后得到了链表的长度，`return len`。

最初看到这个函数返回值为`size`时犹豫了一下，后来发现与字符串中关于`size`和`length`的知识点概念混淆了（还是学的不够扎实啊）。

8、`int list_find(List *list,int v);`寻找某结点并返回要求值

```
int list_find(List *list, int v){
    int ret=-1,i=0;
    Node *p=list->head;
    for(p=list->head;p;p=p->next,i++){
        if(p->value==v){
            ret=i;
            break;
        }
    }
    return ret;
}
```

这个函数的重点与前几个类似，依然是通过`p=p->next`实现链表的遍历，定义整型变量`i`用以记录目标结点位于链表的何处，如果找到目标结点则返回结点位置`i`，反之则返回`-1`。

9、`void list_remove(List *list,int v);`删除特定结点

```
void list_remove(List *list, int v){
    Node *p=list->head,*q=NULL;
    for(p=list->head,q=NULL;p;q=p,p=p->next){
        if(p->value==v){
            if(q){
                if(p==list->tail){
                    list->tail=q;
                    list->tail->next=NULL;
                    p=NULL;
                    free(p);
                    break;
                }else if(p==list->head){
                    list->head=p->next;
                    p=NULL;
                    free(p);
                    break;
                }else{
                    q->next=p->next;
                    p=q->next;
                    free(p);
                    break;
                }
```

```
            }else{
                list->head=p->next;
                free(p);
                p=NULL;
                break;
            }
        }
    }
}
```

这个函数的定义是十个函数中最复杂的，所讨论的情况也是最多的。本函数要求删除链表中值为`v`的结点，因此需要释放该节点的内存并且在删除结点后使链表仍然连续。需要先定义`Node *p=list->head,*q=NULL`,之后对链表进行遍历，判断条件为`p`是否为`NULL`，在每次循环后做操作`q=p,p=p->next`。在这个过程中，不断寻找目标结点，在找到后需要判断以下三种情况并分别作出不同反应：

①`p==list->head`，此时要删除的结点为链表头，此时只需将原先`head`的`next`令为现在的`head`，再释放`p`的内存空间，跳出循环。

②`p==list->tail`，此时要删除的结点为链表尾，此时`q`为`p`的前一个，因此令`q`为新的链表尾，并且使`list->tail->next=NULL`，释放`p`的内存空间，跳出循环。

③`p!=list->head&&p!=list->tail`，此时要删除的结点不是链表头也不是链表尾，因此删除`p`结点只需要将`p`原先的前一个结点和后一个结点连接起来，而此时`p`的前一个结点为`q`即`q->next=p->next`，再释放`p`的内存空间，跳出循环。

10、`void list_iterate(List *list,void(*func)(int v));`对链表各结点值做函数

这个函数的重点仍然是遍历函数，并以此对每个结点中的值做`func`函数。此函数中，直接调用函数`func`即可。

四、实验体会和心得

在完成这个专题的学习之后，最大的收获是初步掌握了大程序结构的相关知识，了解了多个源代码文件的编写、头文件和声明的相关知识点。

在代码的编写过程中，遇到的第一个麻烦是关于`list`的类型问题，这里的`list`是一个指针，最开始的时候混淆了`.`和`->`的用法，访问数据的方式选择错误，虽然之后的修改还算容易，但确实造成了不小的麻烦。

在写函数部分的过程中，我发现在我的代码中一定需要在`void list_free(List *list)`的最后加上`list->tail=NULL;list->head=NULL;`否则无法通过，这跟此函数前部分的代码编写也有关系，且题目中说到需要将`head`和`tail`置零。

之后的几个函数较为简单，`append`重点考虑了链表是否为空的情形，`insert`函数非常直观。`set`和`get`函数是非常相似的，但在写这两个函数的过程中，遇到了`i`和`index`的比较问题，导致循环结束后的结点位置不对，最后在`for`里套了一个`if`判断语句就非常“愉快”地解决了。

问题最大的函数是`remove`，在解决了其余所有问题之后，终于从“运行时错误”变成了“部分错误”，页面显示在`remove last`的地方出现了问题。最开始我仿照上课的`for`循环方式找到目标结点并删除，但漏掉了`p==list->tail`的情况，于是总卡在这个测试点过不了，最后寻求了马同学的帮助才恍然大悟，补上了这种情况的判断。

在书写函数的过程中我同样也遇到了`malloc`使用错误的情形，分配内存的语句始终不太熟悉，于是在程序组某个`malloc`的语句出了小错误（比如漏了个*），于是就迎来了最痛苦的之处——上传压缩包后看到页面显示"segmentation fault"，没有多余的提示，只能从头开始找。

还有就是在写`linkedlist.h`的过程中，对于`Node`声明的位置和标准头文件结构的问题，稍有不当就会造成重复定义或者没有定义第一次使用的错误。

比较遗憾的地方在于本想借此次专题作业的机会学习VSCode的使用，但最后还是因为对DevC++熟悉并且操作比较习惯而“真香”了。还有就是关于“定义”和“声明“的概念总是会混淆，说不清楚两者到底谁是谁。

不过还是非常欣慰地看到，经过这个大程序的编写，上课时掌握比较欠缺和认识比较模糊的一些知识点都在自主学习后清晰了很多，各个函数背后的原理也比初学链表的时候明白得更加透彻，也终于有了一点点写“程序”的感觉。看到”答案正确“的时候，真是很感动呐。写实验报告的同时也让自己对代码有了一个整体和局部的回顾过程，得以反思自己为何会犯错。另外，十分感谢陆TH同学推荐了Typora软件，写出来的报告格式十分好看。希望自己能够通过大程序和小程序的编写，不断进步！