浙江大学实验报告

姓名：赵泓珏　　**学号**：3190104515　　**日期**：2020.3.8　　**成绩**：_____

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

一、 实验题目要求

该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;
```

在这个库中要求提供以下函数：

```
List list_create();
void list_free(List *list);

void list_append(List *list, int v);
void list_insert(List *list, int v);

void list_set(List *list, int index, int v);
int list_get(List *list, int index);

int list_size(List *list);

int list_find(List *list, int v);
void list_remove(List *list, int v);

void list_iterate(List *list, void (*func)(int v));
```

其中：

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、 实验思路和过程解释

在此次实验中需要两个文件：linkedlist.h 和 linkedlist.c。在头文件 linkedlist.h 中声明 List 的结构类型以便在 main.c 和 linkedlist.c 均可使用，同时将题目中要求实现的 API 函数全部复制粘贴进入头文件。为了防止 linkedlist.h 被重复引用，在一开始便写入合适的预处理语句。在 linkedlist.c 文件中，引入标准库和 linkedlist.h 后，因为题目要求不能在头文件中对链表的基本单位节点进行声明，但是为了在 linkedlist.c 中使用，在此处进行声明。之后按照题目要求，依次对 API 函数的代码进行编写，实现函数功能。

三、 实验代码解释

1. linkedlist.h
    1) 首先写入预处理语句，防止该头文件在其他文件中被重复引用（事实证明的确发生了重复引用，因为我在 main.c 中发现了两次 #include "linkedlist.h"）。
    2) 之后对需要用到的结构类型进行声明，以便于在该文件和其他文件中使用。按照要求只在其中对 List 进行声明，未对 Node 进行声

明，猜测的原因是在 main.c 及判题程序中已经对 Node 进行了声明，如果再在头文件进行声明会报错。

3) 将所有 API 函数的函数体复制粘贴于此，对函数进行声明；

4) #endif 结束头文件的编写。

2. linkedlist.c

引入所需要的库，之后将定义的 List 复制过来同时将其注释化便于之后代码的编写，因为未在头文件中对 Node 进行声明而在之后的函数中也都需要用到，在此处对其声明。之后是每个函数代码的具体解释。

1) List list_create();

这个是所有函数里最好编写的函数，主要到函数要求返回一个 List，因此在函数的一开始先定义 List list，在此之后另其内部的 Node 指针 head 和 tail 同时指向 NULL，返回 list，List list_create()的代码编写结束。

2) void list_free(List *list)

这里涉及到的实际上是链表的清除。第一步定义 Node 指针 p 和 q 为之后的遍历使用，第二步用 for 循环对整个链表进行遍历，让 p->next 的值赋给 q 使 q 指向 p 的下一个，之后 free(p)释放掉 p 所占内存（这里也是因为所有链表的结点都是通过 malloc 函数得到，之后会提到），再令 p=q，进行循环。所有的节点内存都被释放掉后为了彻底抹消这个链表的存在，令链表的头 list->head 和尾 list->tail 全部变成空指针，注意到函数类型为 void 不需返回任何值（全部通过指针操作），void list_free(List *list)的代码编写结束。

3) void list_append(List *list, int v);

此函数要求实现用 v 制作新的结点加到链表的最后。第一步定义 Node 指针 p 并用 malloc 函数为其分配内存，由于要用 v 制作新的节点，将 v 的值赋给 p->value，同时考虑到 p 是加到链表的最后，令 p->next=NULL。之后进行最关键的一步——链表之间的相互连接。在这里考虑到两种情况：如果 head 还没有填入任何内容即仍为空指针，那么 list->head->next=p 便没有任何意义，此时需要先把 p 设置 list->head。同时由于此时链表内部只有一个元素，头即是尾，在这里令 list->tail 也为 p 便于之后的连接。如果 head 已有内容则令之前链表尾 list->tail 中的 next 指向 p，实现连接，同时把 p 设置为新的 list->tail，void list_append(List *list, int v)的代码编写结束。

4) void list_insert(List *list, int v);

该函数要求实现用 v 制作心得结点加到链表的最前面。首先采用与上个函数相同的办法为指针 p 分配内存后令 p->value=v，与上个函数不同的地方在于由于需要把这个结点插到函数最前方，因此在这里直接使 p->next 指向原来的链表头 list->head，此时 p 成了新的链表头，所以令 list->head=p，void list_insert(List *list, int v)的代码编写结束。

5) void list_set(List *list, int index, int v);

该函数要求把第 index 个结点的值置为 v，链表结点从 0 开始。考虑到在链表里实际上没有计数变量，先在一开始定义计数用的变量 i 和之后用于遍历的 Node 指针 p。虽然 while 循环和 for 循环都可以实现第 index 个结点的寻找，但是我在这里选择了 while 循环，自我认为这样会使代码更加通俗易懂。在找到第 index 个结点后，令 p->value=v 实现第 index 个结点值

的改变，void list_set(List *list, int index, int v)的代码编写结束。

6) int list_get(List *list, int index);

该函数要求获得链表中第 index 个结点的值。思路同上个函数，通过遍历得到第 index 个结点的值，之后直接返回该结点出的值，int list_get(List *list, int index)的代码编写结束。

7) int list_size(List *list);

该函数要求给出链表大小。考虑到链表里并没有自己的计数系统，首先引入计数变量 cnt 并赋初值为 0，同时定义 Node 的指针 p 用于遍历整个链表。之后用 for 循环遍历链表，用 cnt 计数，最后 cnt 的值即为链表中结点的个数，即链表的大小。返回 cnt 的值，int list_size(List *list)的代码编写结束。

8) int list_find(List *list, int v);

该函数要求在链表中寻找值为 v 的结点，返回结点的编号，如果找不到，返回-1。从第 5 个函数开始到该函数，所有函数的编写的核心都在于遍历链表。在该函数中首先定义整型变量 isfound 和 index 分别为 0，同时定义 Node 的指针 p 用于之后遍历。在用 for 循环进行遍历时，一方面用 index 记录当前所到的结点个数，结点每移动一次就使 index 加一；另一方面寻找 v 的值，只要找到 v 的值就使 isfound 等于 1 并立即结束循环，使 index 停止增长。循环结束后如果 isfound==1 就返回 index 的值，否则就返回-1。这是一种常见的判断方式，引入一个 flag 表示状态，只要找到就令其为 1，若没找到就使其为 0。理论上虽然应该采用单一出口，但是我才疏学浅，只能做到这种地步。int list_find(List *list, int v)的代码编写结束。

9) void list_remove(List *list, int v);

该函数要求删除链表中值为 v 的结点。首先考虑到所有的结点所占据的内存空间都由 malloc()函数得来，如果要删除肯定要释放被删结点的内存，所以要用到 free()函数。但是不能直接 free 需要删除的结点，必须要使被删结点的前后连接起来，所以在一开始定义两个 Node 指针 p 和 q。之后对链表进行遍历（遍历仍然是这个函数的核心之一），首先令 q=NULL，p=list->head，判断退出的条件为 p 是否为空指，。每进行一次循环，就令 q=p，p=p->next，使得遍历向前推进同时确保 q 和 p 为相邻的一对结点。寻找到值为 v 的结点。在找到以后有三种情况需要考虑:

- 需要删除的结点为 list->head。此时重点在于链表头的更换。q 仍然为 NULL，所以也就不存在把需要被删除结点的前后结点进行连接。需要做的只是使得链表头 list->head 向后推进一个，使 list->head 的下一个成为新的链表头以确保链表的完整性。
- 需要删除的结点既不是链表头也不是链表尾而位于链表中间。在这里只需要把需要被删除结点前后进行连接即可。p 为需要删除的节点，q 为 p 的上一个，p->next 为 p 的下一个，只需要使 q->next=p->next 便可实现前后的连接。
- 需要删除的结点为 list->tail。此时重点在于链表尾的更换。q 为 p 的上一个，此时 q 成为新的链表尾，所以令 list->tail=q；同时注意到链表尾中的 next 应该为空指针，所以令 q->next=NULL。

在完成对上述三种情形的判断后即有关更换后，释放 p 占据的内存，由于没有说明是要删除所有值为 v 的结点，所以在这里使用 break 退出循

环，同时在这里采用 break 也防止代码出现问题。如果说需要实现删除链表中所有值 v 的结点，我们可以对该函数中的代码进行循环，为了判断是否已经全部删除，可以采用第八个函数中采用的方法，即引入状态变量 flag 去表示是否已经删除所有值为 v 的结点并以此判断何时退出循环。具体代码已经在 3.11 的实验课中写出，因此不再赘述。void list_remove(List *list, int v)的全部代码编写结束。

10) void list_iterate(List *list, void (*func)(int v));

该函数要求遍历链表，依次对每一个结点中的值做 func 函数。遍历的具体过程已经在前面的函数里被反复使用，在此不再赘述。在遍历过程中同时 func(p->value)即可。至此所有函数的代码编写结束。

四、 实验体会和心得

纵观这十个函数，除了一开始的创建链表和在链表头或尾插入结点的函数之外，其他所有的函数都涉及到了一个最基本的步骤——遍历。链表不同于数组，数组本身就有自己的一套计数系统，遍历起来自然也十分简单，而链表本身就和数组有一定的相似性，为了实现未知个数数据的储存在这里我们牺牲了实际对链表内部内容进行操作的便利性，但是这是值得的。我目前所接触到有关链表的知识还十分有限，还希望之后能够进一步学习。

同时，在此次实验中我也体会到了代码严密的逻辑。实际上在我第一次完成所有代码进行测试时，只有第一个函数运行成功，其他的全部没有显示。在之后的调试过程中，我发现导致这个的原因是我在编写有些函数的代码时虽然本身内部逻辑没有问题，但是如果放到整个 linkedlist.c 来看，实际上有很多无法衔接的地方，破坏了链表的结构，导致代码无法正常运行。这也给我之后的

代码编写提供了经验。

我是在 3 月 8 日，也就是第一次讲链表的前一天完成全部实验内容的，此次实验中我所运用到的有关链表的知识都是在寒假所学。在完成这个实验的过程中，我也多次去查找了有关的资料。通过自己的的探究和摸索，我完成了所有实验的内容，也进一步加深了自己对有关链表内容的理解。我也更加坚定地相信，既然计算机的所有内容都是由人想出来的，那么终有一天我也能理解有关计算机的全部内容。虽然我现在的水平并不是很高，但是我希望能在之后的学习实践中不断提高我自己的水平。这就是我在此次实验中的心得体会和感想。