中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/10/9	完成日期	2021/10/10

1. 实验题目

1. Loops in the Linked List
2. Delete Duplicate
3. Remove for single link list with head node
4. Insert for single link list with head node

2. 实验目的

通过本次实验，熟悉链表的遍历，节点的插入与删除，完成：

1．判断一个链表是否有环

2．删除链表中重复的节点

3．根据索引和头节点删除带头节点的单链表中的任意节点

4．根据索引和头节点在带头节点的单链表中的任意位置插入节点

3.程序设计

1) Loops in the Linked List

设计思路：

由于无环的单链表一直遍历下去最终为 NULL，而有环的单链表会无穷地遍历，故设计一快一慢两个指针：通过循环，快指针 fast 每次走两步，慢指针 slow 每次走一步，如果 slow 追上 fast（fast==slow），说明有环，返回 true；如果 fast==NULL 或 fast->next==NULL，说明链表遍历完了，无环，跳出循环，返回 false。

代码：

```
bool check(node *head){
   if(head==NULL || head->next==NULL) return false ;
   // 当 head 为 NULL 或只有 1 个不成环的节点时都不成环
    node* slow = head ;
    node* fast = head->next;
   while(fast!=NULL && fast->next!=NULL){
   // 有环 fast 就不会走到 NULL，直至 slow==fast；没环就会走到 NULL，跳出循环
        if(slow == fast ) return true ;
        slow = slow->next ;
        fast = fast->next->next ;
    }
    return false ;
}
```

2）Delete Duplicate

设计思路：

从头节点 head 开始遍历，如果头节点为空或只有一个头节点都直接返回；令 p=head ,当 p！=NULL 时循环：

如果 p->data==p->next->data,删除节点 p->next；否则令 p=p->next

如此便删除了重复的节点，且具有稳定性，保留的节点都是先出现的节点。

代码：

```
void delete_duplicate(LinkList &head){
   if(head==NULL || head->next ==NULL)  return ;
   LinkList p = head ;
   while(p->next!=NULL){
      if(p->data==p->next->data){ //当 p==p->next 时删除节点 p->next
         LinkList temp = p->next ;
         p->next = p->next->next ;
         delete temp ;
      }
      else p=p->next ;
   }
}
```

3）Remove for single link list with head node

设计思路：

首先设计一个 ListNode *getNode(ListNode *head,int pos)函数，函数根据头节点和索引获取从实际节点开始的第 pos 个节点的指针。

我们再用 getNode 函数得到第 pos-1 个节点的指针 p_pre，根据该指针得到第 pos 个节点指针（该节点要 delete）temp，和第 pos+1 个节点指针 p_next，使 p_pre->next=p_next,并删去 temp，这便实现了删去第 pos 个实际节点。

代码：

```
ListNode *getNode(ListNode *head,int pos){
//返回从实际节点开始的第 pos 个节点
    ListNode*p = head ;
    for(int i = 0 ; i < pos; ++i ){
      p =p ->next ;
    }
    return  p ;
}

void List::remove(int pos){ // 删除第 pos 个节点
```

```
    ListNode * p_pre = getNode(head,pos-1) ;
    ListNode * temp = p_pre->next ;
    ListNode * p_next = temp->next ;
    p_pre->next = p_next ;
    delete temp ;
}
```

4）Insert for single link list with head node

设计思路：

首先设计一个 ListNode *getNode(ListNode *head,int pos)函数，函数根据头节点和索引获取从实际节点开始的第 pos 个节点的指针（同 3）题）。

我们再用 getNode 函数得到第 pos-1 个节点的指针 p_pre，根据该指针得到第 pos 个节点指针 p_next，将新增节点插入 p_pre 和 p_next 之间，这便实现了在第 pos 个实际节点前新增一个节点。

代码：

```
ListNode *getNode(ListNode *head,int pos){
//返回从实际节点开始的第 pos 个节点
    ListNode*p = head ;
    for(int i = 0 ; i < pos; ++i ){
      p =p ->next ;
    }
    return  p ;
}

void List::insert(int toadd, int pos){
//在第 pos-1 个节点和第 pos 个节点间新增 toadd 节点
   ListNode* _add =  new ListNode ;
   _add->data = toadd ;
   ListNode * p_pre = getNode(head,pos-1) ;
   ListNode * p_next = p_pre->next ;
   p_pre->next = _add ;
   _add->next = p_next ;
}
```

4.程序运行与测试

本次实验不用写程序运行与测试。

5. 实验总结与心得

本次实验让我熟悉了链表中节点的删除与插入操作，尤其是在带有头节点的单链表中，由于头节点的存在，使在任一实际节点前插入新节点及删除任一实际节点的操作都变得统一，不用再去区分情况，使得对第一个节点的删除或在第一个节点前插入元素的操作与其他节点相同。

附录、提交文件清单