电子科技大学信息与软件工程学院

实 验 报 告

学　　号　2016220304022

姓　　名　罗悦

（实验）　课程名称　数据结构与算法

理论教师　陈安龙

实验教师　陈安龙

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告（1）

学生姓名：罗悦　　**学 号：**2016220304022　　**指导教师：**陈安龙

实验地点：清水河科技实验大楼　　**实验时间：**2017.4.5

一、实验室名称：学校实验中心软件实验室 506

二、实验项目名称：编程实现线性表的合并

三、实验学时：4

四、实验原理：

在链式存储结构中，存储数据结构的存储空间可以不连续，各数据结点的存储顺序与数据元素之间的逻辑关系可以不一致，而数据元素之间的逻辑关系是由指针域来确定的。链式存储方式即可以用于表示线性结构，也可用于表示非线性结构。一般来说，在线性表的链式存储结构中，各数据结点的存储符号是不连续的，并且各结点在存储空间中的位置关系与逻辑关系也不一致。对于线性链表，可以从头指针开始，沿各结点的指针扫描到链表中的所有结点。

线性表的链接存储中，为了方便在表头插入和删除结点的操作，经常在表头结点（存储第一个元素的结点）的前面增加一个结点，称之为头结点或表头附加结点。这样原来的表头指针由指向第一个元素的结点改为指向头结点，头结点的数据域为空，头结点的指针域指向第一个元素的结点。

五、实验目的：

本实验通过定义单向链表的数据结构，设计创建链表、插入结点、遍历结点等基本算法，使学生掌握线性链表的基本特征和算法，并能熟练编写 C 程序，培养理论联系实际和自主学习的能力，提高程序设计水平。

六、实验内容：

使用数据结构 typedef struct node {

```
    Elemtype    data;
    struct    node    *next;
} ListNode, *ListPtr;
typedef struct stuInfo {
    int     stuID;
    char    stuName[10];        /*学生姓名*/
    int      Age    /*年龄*/
 } ElemType
```

实现带头结点的单向链表的创建、删除链表、插入结点等操作，可每个学生的学号互不相同，学号不同而姓名相同则为不同的学生，每个学生的学号在合并后的链表中不重复，如果出现重复，则删除年龄较小结点。最后打印输出合并后的链表元素，验证结果的正确性。

（1）设计学生信息结点的数据结构；

（2）用 C 语言实现创建升序链表的函数，每个结点的学号不同，按照学号升序排列；

（3）用 C 语言实现结点的插入的函数，插入后仍然为升序；

（4）编程实现两个单向链表合并，合并后仍然升序；

（5）编程实现合并后链表逆序排列的算法；

（6）打印输出合并后的链表元素。

七、实验器材（设备、元器件）：

PC 机一台，装有 C 语言集成开发环境。

八、数据结构与程序：

```
#include<stdio.h>
#include<stdlib.h>
typedef struct stu{
    int stuID;
    char stuName[21];
    int Age;
}ElemType;

typedef struct node{
    ElemType data;
    struct node *next;
```

```
}ListNode,*LinkedList;

void LinkedListInit(LinkedList *L){
     *L=(ListNode*)malloc(sizeof(ListNode));
        if(!L){
        printf("申请空间失败！");
        exit(0);
          }
        (*L)->next=NULL;
}

void LinkedListout(LinkedList L){
     L=L->next;
     printf("\nID\tName\tAge\n");
    while(L!=NULL){
        printf("%d\t",L->data.stuID);
        printf("%s\t",L->data.stuName);
        printf("%d\n",L->data.Age);
         L=L->next;

    }
}

ElemType read(){
     ElemType e;
     scanf("%d",&e.stuID);
     scanf("%s",e.stuName);
     scanf("%d",&e.Age);
     return e;
}

int  leave(LinkedList *l,LinkedList e){
    LinkedList p,pre;
    pre=*l;
    p=(*l)->next;
    while(p!=NULL){
        if(p->data.stuID==e->data.stuID){
            pre->next=p->next;
            free(p);
            p=NULL;
            return 0;
        }
        else p=p->next;
    }
```

```
}

int length(LinkedList l){
    LinkedList p;
    int i=0;
    LinkedListInit(&p);
    p=l->next;
    while(p!=NULL){
        p=p->next;
        i++;
    }
    return i;
}

void LinkedList1(LinkedList *L){
      int  i,n;
      printf("请输入学生总人数：\n");
      scanf("%d",&n);
      *L=(ListNode*)malloc(sizeof(ListNode));
        if(!*L){
        printf("申请空间失败！");
        exit(0);
          }
      LinkedList  p,tail;
      (*L)->next=NULL;
      tail=*L;
      printf("请按学号、姓名、年龄的顺序输入你要创建链表的数据：\n");
      for(i=0;i<n;i++){
        p=(ListNode*)malloc(sizeof(ListNode));
        if(!p){
        printf("申请空间失败！");
        exit(0);
          }
         p->data=read();
         p->next=NULL;
         tail->next=p;
         tail=p;
      }

}


void LinkedListInsert(LinkedList *L){
      LinkedList pre,q,p;
```

```
      pre=*L;
      p=(*L)->next;
     q=(ListNode*)malloc(sizeof(ListNode));
     LinkedListInit(&q);
      printf("请输入你要插入链表 L3 的学生的学号、名字和年龄：\n");
      q->data=read();
         while(p&&(q->data.stuID>p->data.stuID)){
        p=p->next;
        pre=pre->next;
     }
     q->next=p;
     pre->next=q;

}

void   LinkedListreverse(LinkedList *lc) {
     int i=length(*lc);
    LinkedList l,p1,p2,p3,pre,pre1,pre2;
    LinkedListInit(&l);
    while(i>=2){
    pre=*lc;
    pre1=*lc;
    pre2=*lc;
    p1=(*lc)->next;
    p2=(*lc)->next;
    p2=p2->next;
    pre2=pre2->next;
    if(p1->data.stuID>p2->data.stuID){
        p3=p2;
        p2=p2->next;pre=pre2;pre2=pre2->next;

    }
    else {
        p3=p1;
        p2=p2->next;pre2=pre2->next;
        pre=pre1;
    }
    while(p2){
        if(p2->data.stuID>p3->data.stuID)
        p2=p2->next;
        else{
            p3=p2;
            p2=p2->next;pre=pre2;pre2=pre2->next;
```

```
        }
    }

    pre->next=p3->next;
     leave(lc,p3);
     p3->next=NULL;
    p3->next=l->next;
    l->next=p3;
    i--;
    }
    pre=(*lc)->next;
    pre->next=l->next;
    l->next=pre;

}


void LinkedListAsd(LinkedList *lc){
    int i=length(*lc);
    LinkedList l,p1,p2,p3,pre,pre1,pre2;
    LinkedListInit(&l);
    while(i>=2){
    pre=*lc;
    pre1=*lc;
    pre2=*lc;
    p1=(*lc)->next;
    p2=(*lc)->next;
    p2=p2->next;
    pre2=pre2->next;
    if(p1->data.stuID<p2->data.stuID){
        p3=p2;
        p2=p2->next;pre=pre2;pre2=pre2->next;

    }
    else {
        p3=p1;
        p2=p2->next;pre2=pre2->next;
        pre=pre1;
    }
    while(p2){
        if(p2->data.stuID<p3->data.stuID)
        p2=p2->next;
        else{
            p3=p2;
```

```
            p2=p2->next;pre=pre2;pre2=pre2->next;

        }
    }

    pre->next=p3->next;
     leave(lc,p3);
     p3->next=NULL;
    p3->next=l->next;
    l->next=p3;
    i--;
    }
    pre=(*lc)->next;
    pre->next=l->next;
    l->next=pre;

}

void Listmerge(LinkedList *la, LinkedList *lb, LinkedList *lc){
    LinkedList pa,pb,pc;
    pa=(*la)->next;
    pb=(*lb)->next;
    *lc=*la;
    pc=*lc;
    while((pa!=NULL)&&(pb!=NULL)){
        if(pa->data.stuID<pb->data.stuID){
            pc->next=pa;
            pc=pa;
            pa=pa->next;
        }
        else if(pa->data.stuID>pb->data.stuID){
            pc->next=pb;
            pc=pb;
            pb=pb->next;
        }
        else if(pa->data.stuID==pb->data.stuID){
            if(pa->data.Age<=pb->data.Age){
                pa=pa->next;
                pc->next=pb;
                pc=pb;
                pb=pb->next;
            }
            else{
                pb=pb->next;
```

```
            pc->next=pa;
            pc=pa;
            pa=pa->next;
        }
    }
}
if(pa)pc->next=pa;
else pc->next=pb;

}

int main(){
 LinkedList L1,L2,L3;
 LinkedListInit(&L1);
 LinkedListInit(&L2);
 LinkedListInit(&L3);
 printf("请创建链表 L1!");
 LinkedList1(&L1);
 printf("\nL1 按学号升序排列为：");
 LinkedListAsd(&L1);
 LinkedListout(L1);
 printf("\nL2 请创建链表 L2!");
 LinkedList1(&L2);
 printf("\n 按学号升序排列为：");
 LinkedListAsd(&L2);
 LinkedListout(L2);
 printf("\nL1 与 L2 保序合并后的链表 L3 为：");
 Listmerge(&L1,&L2,&L3);
   LinkedListout(L3);
 LinkedListInsert(&L3);
 printf("\n 插入新元素后的 L3 为：");
 LinkedListout(L3);
 printf("L3 按学号逆序排列为：");
LinkedListreverse(&L3);
 LinkedListout(L3);
printf("实验完成！");
return 0;
}
```

九、程序运行结果：

图 1-1

图 1-1 先输入链表 L1 的学生人数，再录入数据，并且以学号升序依次排列输出。

图 1-2

图 1-2 先输入链表 L2 的数据，并且同样以学号的升序输出。并且将 L1 与 L2 保序合并程链表 L3 并且输出。

图 1-3

图 1-3 在合并后的链表 L3 中插入新元素，依然按升序排列。再将链表 L3 按学号大小逆序排列。实验完成。

十、实验结论：

此次实验成功的解决了学生信息的存储和按学号升序排列的续剧结构。并且实现了节点的删除和插入操作。也完成了对两个链表的保序合并。和对一个链表的逆序操作。实验中遇到了许多困难，但都靠巧妙的运用指针和数组得到解决。

十一、总结及心得体会：

为了解决顺序表带来的插入和删除操作的开销，这里运用线性表的链式存储结构。这种存储方式可以允许不连续存储，这样在插入和删除时就不需要部分或是全部移动表。各数据结点的存储顺序与数据元素之间的逻辑关系可以不一致，而数据元素之间的逻辑关系是由指针域来确定的。所以可以从头结点开始，沿各结点的指针扫描到链表中的所有结点。

首先为了方便在表头插入和删除结点的操作，经常在表头结点（存储第一个元素的结点）的前面增加一个结点，称之为头结点或表头附加结点。这样原来的表头指针由指向第一个元素的结点改为指向头结点，头结点的数据域为空，头结点的指针域指向第一个元素的结点。

此实验应多次运用指针，搜索找寻原链表中所需要的部分，再进行删除或插入操作，或定义新的链表依次取出原链表的学号较小的节点，并存入新的链表中，这样就完成了对原链表按学号升序排列的操作。对于两个链表的合并，仍然运用指针进行操作，依次比较两链表的节点的学号大小并且录入新的链表中。

总之采用动态存储分配，不会造成内存浪费和溢出；另外，链表执行插 入和删除操作十分方便，修改指针即可，不需要移动大量元素。且不要求连续空间，空间利用效率高。

此次实验让我更加熟练的掌握了指针和数组的运用，能清楚的分析出节点含义和转换删除或是插入的操作。但如果不认真仔细的编写程序却很容易出现许多小错误，如漏写或多写了一些符号，或是没有定义变量，或是形参和实参的类型的不同或是指针的越界现象等等...但只要细心改好每一个错误，仔细检查好每一个程序，就会使自己的能力得到一个很大的提升。