

南京林业大学

实验报告

学 院： 信息科学技术学院、人工智能学院

专 业： 计算机科学与技术

课程名称： 数据结构

学 号： 2351610105

学生姓名： 方泽宇

指导教师： 张黎宁

二〇二四年三月——二〇二四年六月

实验一：设计实现抽象数据类型“有理数”

一. 实验内容：

设计实现抽象数据类型“有理数”

设计并上机实现抽象数据类型“有理数”，有理数的基本操作包括：两个有理数的加、减、乘、除等（包括有理数的创建和输出等）。

二. 实验要求：

- (1) 要有能根据用户输入选择不同运算的菜单选择界面。
- (2) 有理数的类型，我们可以构造成一个结构体类型，这个结构体由两个整数构成，分别表示有理数的分子和分母。
- (3) 在初始化或创建一个有理数时，可以给出有理数的分子和分母来创建一个有理数；也可以给出一个小数形式的有理数，来计算对应的分子分母来创建一个有理数（可设置一个允许的计算误差）。
- (4) 以分数形式创建有理数时，要处理分母为零的异常情况。
- (5) 输出不能有类似于 $4/4$ 、 $3/6$ 这样的结果数据。

三. 源程序：

```
#include<stdio.h>
typedef struct
{
    int num,deno;
}RatNum;
int gcd(int a,int b)
{
    return b?gcd(b,a%b):a;
}
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;
}
void create(RatNum *p)
{
    printf("input a num:");
```

```
scanf("%d %d",&(p->num),&(p->deno));
}
void print(RatNum a)
{
    if(a.deno==1) printf("%d",a.num);
    else printf("%d/%d",a.num,a.deno);
}
RatNum approximation(RatNum a)
{
    int factor=gcd(a.num,a.deno);
    a.deno/=factor;
    a.num/=factor;
    if((a.deno<0&& a.num>0) || (a.deno<0&& a.num<0))
    {
        a.num=-a.num;
        a.deno=-a.deno;
    }
    return a;
}
RatNum add(RatNum a,RatNum b)
{
    RatNum res;
    res.num=a.num*b.deno+b.num*a.deno;
    res.deno=a.deno*b.deno;
    return approximation(res);
}
RatNum sub(RatNum a,RatNum b)
{
    RatNum res;
    res.num=a.num*b.deno-b.num*a.deno;
    res.deno=a.deno*b.deno;
    return approximation(res);
}
RatNum muti(RatNum a,RatNum b)
{
    RatNum res;
    res.num=a.num*b.num;
    res.deno=a.deno*b.deno;
    return approximation(res);
}
RatNum div(RatNum a,RatNum b)
{
    RatNum res={0,0};
    if(a.deno==0 || b.deno==0)
    {
```

```
        printf("error\n");
        return res;
    }
    res.num=a.num*b.deno;
    res.deno=a.deno*b.num;
    return approximation(res);
}

int main()
{
    int ch;
    RatNum a,b,res;
    printf("1.add\n2.sub\n3.muti\n4.div\n");
    scanf("%d",&ch);
    printf("give the first number\n");
    create(&a);
    printf("give the second number\n");
    create(&b);
    switch (ch)
    {
        case 1:res=add(a,b);break;
        case 2:res=sub(a,b);break;
        case 3:res=muti(a,b);break;
        case 4:res=div(a,b);break;
        default:break;
    }
    print(res);
    return 0;
}
```

四. 运行结果:

(1) 加法运算

```
1.add
2.sub
3.muti
4.div
1
give the first number
input a num:2 3
give the second number
input a num:4 5
22/15%
fang50253@MacBook-Pro 课程作业 %
```

(2) 减法运算

```
1.add
2.sub
3.muti
4.div
2
give the first number
input a num:1 2
give the second number
input a num:1 2
0%
fang50253@MacBook-Pro 课程作业 %
```

(3) 乘法运算

```
1.add
2.sub
3.muti
4.div
3
give the first number
input a num:1 3
give the second number
input a num:7 2
7/6%
fang50253@MacBook-Pro 课程作业 %
```

(4) 除法运算

```
2.sub
3.muti
4.div
4
give the first number
input a num:3 7
give the second number
input a num:7 0
error
_
```

五. 实验小结:

- (1) 在本次实习过程中, 我们要通过多种运行的数据来寻找程序可能存在的问题, 并通过修改程序来解决他们。比如对于一个有理数(分数), 其分母不能为 0, 一个被除数也不可以为 0。
- (2) 本次实验采用一个结构体类型来保存有理数, 保证了有理数的存储精度不会因为数据类型的精度而丢失。
- (3) 代码实现了有理数的基本运算, 包括加法、减法、乘法和除法, 这些操作都是基于有理数的分子和分母的基本运算。使用了 approximation 函数来简化结果, 确保结果为最简分数形式。

实验二：顺序表基本操作的实现

一. 实验内容：

实现顺序表的初始化，输出顺序表中各元素的值，在顺序表中插入数据元素，删除数据元素，求顺序表的长度，顺序表的逆置，顺序表的按值从小到大排序，合并有序的两个顺序表等操作。

二. 实验要求：

- (1) 要有能根据用户输入选择不同运算的菜单选择界面。
- (2) 顺序表中数据元素的类型统一抽象表示为 ElemType 类型，具体类型不限，可以是整型、实型、字符型、或者是自己构造的一种结构体类型。
- (3) 实现课后习题有关顺序表运算算法

三. 源程序：

```
#include<stdio.h>
#define ElemType int
#define ERROR 1
#define maxsize 110
typedef struct
{
    ElemType elem[maxsize]; // 静态申请顺序表所需要的空间
    int last;
}SeqList;
SeqList List;
void Init_List(SeqList *L);
void Visit(SeqList *L);
void Create(SeqList *L,int n);
void Insert_List(SeqList *L,int arr,ElemType x);
int Seq_Length(SeqList *L);
void Reserve_List(SeqList *L,int l,int r);
void Sort_List(SeqList *L,int l,int r);
void Merge_List(SeqList *a,SeqList *b,SeqList *c);
void swap(ElemType *a,ElemType *b);
void Merge_List(SeqList *a,SeqList *b,SeqList *c);
void Delete_List(SeqList *L,int arr);
void Merge_Sort_List(SeqList *a,SeqList *b,SeqList *c);
void swap(ElemType *a,ElemType *b)
{
    ElemType tmp=*a;
    *a=*b;
```

```
*b=tmp;
}

void Init_List(SeqList *L)// 初始化操作
{
    L->last=-1;
}

void Visit(SeqList *L)// 按顺序输出顺序表
{
    if(L->last==-1)
    {
        printf("Empty_array\n");
        return;
    }
    for(int i=0;i<=L->last;++i) printf("%d ",L->elem[i]);
    printf("\n");
}

void Create(SeqList *L,int n)// 创建一个单链表, 并使用尾插法输入 n 个数字
{
    if(n>maxsize)
    {
        printf("Failed to create the array,the num is out of the size\n");
    }
    Init_List(L);
    // 初始化链表L
    for(int i=0;i<n;++i)
    {
        // 依次输入数据, 并将其添加至顺序表尾部
        ElemType input;
        scanf("%d",&input);
        Insert_List(L,i-1,input);
    }
}

void Insert_List(SeqList *L,int arr,ElemType x)// 在顺序表L 中, 索引号arr 的数字之后插入数据元素x
{
    // 第一步, 移动索引号为arr+1 到last 到元素, 使其索引号+1
    if(L->last==maxsize)
    {
        // 处理静态顺序表满的异常
        printf("Error,the array is full\n");
        return;
    }
    else ++L->last;
    for(int i=L->last;i>=arr+1;--i) L->elem[i+1]=L->elem[i];
    // 第二步, 将x 添加到顺序表的arr 结点当中
```

```
L->elem[arr+1]=x;
}
int Seq_Length(SeqList *L)// 返回顺序表L 的长度
{
    return L->last+1;
}
void Reserve_List(SeqList *L,int l,int r)// 对顺序表L 中, 索引号为l 到r 的项进行逆序操作
{
    if(l>r) printf("Error,the num L is larger than R\n");
    while(l<r)
    {
        swap(&(L->elem[l]),&(L->elem[r]));
        ++l,--r;
    }
}
void Sort_List(SeqList *L,int l,int r)// 将顺序表中, 对索引号l 到r 的项从小到大排序
{
    if(l>=r) return;
    int arr=L->elem[l+r>>1];
    int i=l-1,j=r+1;
    while(i<j)
    {
        do ++i; while(L->elem[i]<arr);
        do --j; while(L->elem[j]>arr);
        if(i<j) swap(&(L->elem[i]),&(L->elem[j]));
    }
    Sort_List(L,l,j);
    Sort_List(L,j+1,r);
}
void Merge_List(SeqList *a,SeqList *b,SeqList *c)// 将a 和b 两个顺序表合并, 并将答案存入 c
中
{
    int i=0,j=0,k=0;
    for(int i=0;i<=a->last;++i,++k) Insert_List(c,k-1,a->elem[i]);
    for(int i=0;i<=b->last;++i,++k) Insert_List(c,k-1,b->elem[i]);
}
void Merge_Sort_List(SeqList *a,SeqList *b,SeqList *c)// 合并两个有序的顺序表
{
    int i=0,j=0,k=0;
    while(i<=a->last&& j<=b->last)
    {
        if(a->elem[i]<b->elem[j])
        {
            Insert_List(c,k-1,a->elem[i]);
            ++i;
        }
    }
}
```



```
    }
    else
    {
        Insert_List(c,k-1,b->elem[j]);
        ++j;
    }
    ++k;
}
while(i<=a->last)
{
    Insert_List(c,k-1,a->elem[i]);
    ++i,++k;
}
while(j<=b->last)
{
    Insert_List(c,k-1,b->elem[j]);
    ++j,++k;
}
}
void Delete_List(SeqList *L,int arr)// 删除顺序表L 中索引号为arr 的元素
{
    for(int i=arr+1;i<=L->last;++i) L->elem[i-1]=L->elem[i];
    --L->last;
}
void fun()
{
    //system(CLS);
    printf("1. 在顺序表中插入数组元素\n");
    printf("2. 删除数组中的元素\n");
    printf("3. 求顺序表的长度\n");
    printf("4. 顺序表的逆序\n");
    printf("5. 将顺序表中的值从小到大排序\n");
    printf("6. 合并两个顺序表\n");
    printf("7. 合并两个有序的顺序表\n");
    printf("8. 输出 L\n");
    printf("9. 使用尾插法一次性向 List 输入数据\n");
    int ch;
    scanf("%d",&ch);
    if(ch==1)
    {
        printf("2 个参数: 在顺序表 L 中, 索引号 arr 的数字之后插入数据元素 x\n");
        int arr;
        ElemType x;
        scanf("%d%d",&arr,&x);
        Insert_List(&List,arr,x);
    }
}
```

```
}
else if(ch==2)
{
    printf("1 个参数: 在顺序表 L 中, 删除索引号为 arr 的元素\n");
    int arr;
    scanf("%d",&arr);
    Delete_List(&List,arr);
}
else if(ch==3)
{
    printf("0 个参数\n");
    printf("顺序表 L 的长度为: %d\n",Seq_Length(&List));
}
else if(ch==4)
{
    printf("2 个参数: 需要逆序部分索引号的左边界和右边界\n");
    int l,r;
    scanf("%d%d",&l,&r);
    Reserve_List(&List,l,r);
}
else if(ch==5)
{
    printf("2 个参数: 需要排序的索引号左边界和右边界\n");
    int l,r;
    scanf("%d%d",&l,&r);
    Sort_List(&List,l,r);
}
else if(ch==6)
{
    SeqList a,b;
    printf("4 个参数: 请先输入顺序表 a 的长度, 再依次输入 a 中的值; 再输入顺序表 b 的长度, 再依次输入 b 中的值\n");
    int len;
    scanf("%d",&len);
    Create(&a,len);
    scanf("%d",&len);
    Create(&b,len);
    Init_List(&List);
    Merge_List(&a,&b,&List);
}
else if(ch==7)
{
    SeqList a,b;
    printf("4 个参数: 请先输入顺序表 a 的长度, 再依次输入 a 中的值; 再输入顺序表 b 的长度, 再依次输入 b 中的值\n");
```

```
    int len;
    scanf("%d",&len);
    Create(&a,len);
    scanf("%d",&len);
    Create(&b,len);
    Init_List(&List);
    Merge_Sort_List(&a,&b,&List);
}
else if(ch==8)
{
    Visit(&List);
}
else if(ch==9)
{
    printf("2个参数: 输入需要创建顺序表的表长, 以及顺序表的每个数字\n");
    int n;
    scanf("%d",&n);
    Create(&List,n);
}
}
int main()
{
    Init_List(&List);
    while(1) fun();
    return 0;
}
```

四. 运行结果:

1. 使用尾插法建表(一次性输入数据)

```
1 warning generated.
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
9
2个参数: 输入需要创建顺序表的表长, 以及顺序表的每个数字
5 1 2 3 4 5
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
8
1 2 3 4 5
```

2. 删除指定索引号的元素

```
1 2 3 4 5
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
2
1个参数：在顺序表L中，删除索引号为arr的元素
2
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
8
1 2 4 5
4.在顺序表中插入数组元素
```

3. 求顺序表的表长

```
1 2 4 5
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
3
0个参数
顺序表L的长度为：4
4.在顺序表中插入数组元素
```

5. 将顺序表中的值从小到大排序

```
2417 327 2389 2389 291
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
5
2个参数：需要排序的索引号左边界和右边界
0 4
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
8
291 327 2389 2389 2417
4.在顺序表中插入数组元素
```

6. 合并两个有序顺序表

```

7
4个参数：请先输入顺序表a的长度，再依次输入a中的值；再输入顺序表b的长度，再依次输入b中的值
4 2 4 6 8
5 3 7 9 12 15
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
8
2 3 4 6 7 8 9 12 15

```

7. 顺序表的逆置

```

2 3 4 6 7 8 9 12 15
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
4
2个参数：需要逆序部分索引号的左边界和右边界
0 8
1.在顺序表中插入数组元素
2.删除数组中的元素
3.求顺序表的长度
4.顺序表的逆序
5.将顺序表中的值从小到大排序
6.合并两个顺序表
7.合并两个有序的顺序表
8.输出L
9.使用尾插法一次性向List输入数据
8
15 12 9 8 7 6 4 3 2

```

五. 实验小结：

- (1) 顺序表结构体：SeqList 包含一个静态数组 elem 和一个指向最后一个元素的 last 指针。
- (2) 初始化顺序表：Init_List 函数将顺序表的 last 指针设置为 -1，表示顺序表为空。
- (3) 插入元素：Insert_List 在指定位置插入元素，先右移元素再插入新元素。
- (4) 删除元素：Delete_List 删除指定位置的元素，并将后续元素左移以填补空缺。
- (5) 获取长度：Seq_Length 返回顺序表的当前长度，即 last + 1。
- (6) 逆序操作：Reserve_List 将顺序表中指定区间的元素顺序反转。
- (7) 排序操作：Sort_List 使用快速排序算法对指定区间的元素从小到大排序。
- (8) 顺序表合并：Merge_List 将两个顺序表合并为一个顺序表，不要求输入顺序表有序。
- (9) 合并有序顺序表：Merge_Sort_List 合并两个有序顺序表，保证合并后的顺序表仍然有序。

- (10) 遍历输出: `Visit` 按顺序输出顺序表中的所有元素, 如果为空则提示。
- (11) 顺序表创建: `Create` 函数使用尾插法根据输入的元素个数创建顺序表。
- (12) 主菜单功能: `fun` 函数提供一个用户交互的菜单, 执行各种顺序表操作。

实验三：单链表基本操作的实现

一. 实验内容：

构建线性表的链式存储结构，采用动态分配方式实现单链表的初始化，数据的插入，删除，输出单链表内中各元素，求单链表的长度，实现单链表中数据结点的按值排序，实现单链表的逆置，合并两个有序的单链表（有序的 a 表和有序的 b 表合并之后的结果保存在 a 表中）等操作。

二. 实验要求：

- (1) 要有能根据用户的输入来选择不同运算的菜单界面。
- (2) 单链表中数据元素的类型统一抽象表示为 ElemType 类型，具体类型不限，可以是整型、实型、字符型、或者是自己构造的一种结构体类型。
- (3) 实现课后习题有关单链表运算算法

三. 源程序：

```
#include<stdio.h>
#include<stdlib.h>
#define ElemType int
//#define CLS "cls"//win
#define CLS "clear"//macos/linux
typedef struct Node
{
    ElemType data;
    struct Node *next;
}Node,*LinkList;
Node *List;
void Init_List(Node *L);
void Insert_List(Node *L,int arr,ElemType x);
void Delete_List(Node *L,int arr);
void Visit(Node *L);
int List_Length(Node *L);
void List_Bubble_Sort(Node *L);
void List_Reserve(Node *L);
void Merge_List(Node *a,Node *b);
void swap(ElemType *a,ElemType *b);
void swap(ElemType *a,ElemType *b)
{
    ElemType tmp=*a;
    *a=*b;
    *b=tmp;
}
void Init_List(Node *L)//单链表的初始化
{
```

```
L->next=NULL;
}
void Insert_List(Node *L,int arr,ElemType x)//向单链表L的第arr位后面插入x
{
    Node *p=L;
    int cnt=-1;
    while(p->next!=NULL&&arr!=-1)
    {
        p=p->next;
        ++cnt;
        if(cnt==arr) break;
    }
    if(cnt<arr)
    {
        printf("Insert_Out_of_Array\n");
        return;
    }
    Node *new_node=(Node*)malloc(sizeof(Node));
    new_node->data=x;
    new_node->next=p->next;
    p->next=new_node;
}
void Delete_List(Node *L,int arr)//删除单链表L中索引号为arr的结点
{
    Node *p=L;
    int cnt=0;
    while(p->next!=NULL&&arr!=0)
    {
        p=p->next;
        ++cnt;
        if(cnt==arr) break;
    }
    if(p->next==NULL)
    {
        printf("Delete_Out_of_Array\n");
        return;
    }
    Node *tmp=p->next;
    p->next=p->next->next;
    free(tmp);
}
void Visit(Node *L)//输出单链表中的所有元素
{
    Node *p=L->next;
    while(p!=NULL)
```



```
{
    printf("%d ", p->data);
    p=p->next;
}
printf("\n");
}

int List_Length(Node *L) // 返回1 以L 为头节点的链表的长度
{
    Node *p=L;
    int cnt=0;
    while(p->next!=NULL)
    {
        p=p->next;
        ++cnt;
    }
    return cnt;
}

void List_Bubble_Sort(Node *L) // 对链表L 进行冒泡排序
{
    Node *p=L;
    int change=1;
    while(change)
    {
        change=0;
        p=L;
        while(p->next->next!=NULL)
        {
            p=p->next;
            if(p->data>p->next->data)
            {
                swap(&(p->data), &(p->next->data));
                change=1;
            }
        }
    }
}

void List_Reserve(Node *L) // 实现单链表的逆置
{
    if(!L->next || !(L->next->next)) return;
    Node *p=L->next->next; L->next->next=NULL;
    while(p)
    {
        Node *temp=p->next;
        p->next=L->next; L->next=p;
        p=temp;
    }
}
```

```
    }
}

void Merge_List(Node *a, Node *b) // 合并两个有序的单链表 (有序的a 表和有序的b 表合并之后的结果保
存在a 表中)
{
    Node *pre=a, *p=a->next, *q=b->next;
    while(p&&q)
    {
        if(p->data<q->data) {pre->next=p; pre=p; p=p->next;}
        else {pre->next=q; pre=q; q=q->next;}
    }
    pre->next=p?p:q;
    free(b);
}

void Init_Insert(Node *L, int n)
{
    Init_List(L);
    for(int i=1; i<=n; ++i)
    {
        int x;
        scanf("%d", &x);
        Insert_List(L, i-2, x);
    }
}

void fun()
{
    printf("1. 初始化链表 L, 并使用尾插法添加 n 个元素\n");
    printf("2. 在链表 L 的索引号 arr 后面插入一个数字 x\n");
    printf("3. 删除链表 L 中索引号为 arr 的项\n");
    printf("4. 输出单链表 L 中所有的项\n");
    printf("5. 输出单链表 L 的长度\n");
    printf("6. 将单链表 L 中的值按照升序排序\n");
    printf("7. 将单链表中的值逆序\n");
    printf("8. 合并两个有序的单链表\n");
    int ch;
    scanf("%d", &ch);
    if(ch==1)
    {
        printf("2 个参数: 需要添加的元素个数, 所有元素\n");
        int n;
        scanf("%d", &n);
        Init_Insert(List, n);
    }
    else if(ch==2)
    {

```

```
    printf("2 个参数: 在索引号 arr 后面添加元素 x\n");
    int arr,x;
    scanf("%d%d",&arr,&x);
    Insert_List(List,arr,x);
}
else if(ch==3)
{
    printf("1 个参数: 删除元素的索引号 arr\n");
    int arr;
    scanf("%d",&arr);
    Delete_List(List,arr);
}
else if(ch==4)
{
    printf("无参数\n");
    Visit(List);
}
else if(ch==5)
{
    printf("无参数\n");
    printf("List 的长度为%d\n",List_Length(List));
}
else if(ch==6)
{
    printf("无参数\n");
    List_Bubble_Sort(List);
}
else if(ch==7)
{
    printf("无参数\n");
    List_Reserve(List);
}
else if(ch==8)
{
    printf("2 个参数: 需要和 List 归并的升序数组 b 的长度, 以及他的所有元素\n");
    int n;
    scanf("%d",&n);
    Node *b=malloc(sizeof(Node));
    Init_Insert(b,n);
    Merge_List(List,b);
}
}

int main()
{
    List=malloc(sizeof(Node));
```

```

while(1) fun();
return 0;
}

```

四. 运行结果:

1. 创建链表, 并添加元素 1 15 6 9 10

```

1
2个参数: 需要添加的元素个数, 所有元素
5 1 15 6 9 10
1.初始化链表L, 并使用尾插法添加n个元素
2.在链表L的索引号arr后面插入一个数字x
3.删除链表L中索引号为arr的项
4.输出单链表L中所有的项
5.输出单链表L的长度
6.将单链表L中的值按照升序排序
7.将单链表中的值逆序
8.合并两个有序的单链表
4
无参数
1 15 6 9 10

```

2. 在 arr=2 的后面添加一个元素 18

```

2 2 18
2个参数: 在索引号arr后面添加元素x
1.初始化链表L, 并使用尾插法添加n个元素
2.在链表L的索引号arr后面插入一个数字x
3.删除链表L中索引号为arr的项
4.输出单链表L中所有的项
5.输出单链表L的长度
6.将单链表L中的值按照升序排序
7.将单链表中的值逆序
8.合并两个有序的单链表
4
无参数
1 15 6 18 9 10

```

3. 删除 arr=4 的项

```

3
1个参数: 删除元素的索引号arr
4
1.初始化链表L, 并使用尾插法添加n个元素
2.在链表L的索引号arr后面插入一个数字x
3.删除链表L中索引号为arr的项
4.输出单链表L中所有的项
5.输出单链表L的长度
6.将单链表L中的值按照升序排序
7.将单链表中的值逆序
8.合并两个有序的单链表
4
无参数
1 15 6 18 10

```

4. 输出单链表 L 现在的长度

```
5
无参数
List 的长度为 5
```

5. 将单链表中的值逆序

```
7
无参数
1. 初始化链表 L, 并使用尾插法添加 n 个元素
2. 在链表 L 的索引号 arr 后面插入一个数字 x
3. 删除链表 L 中索引号为 arr 的项
4. 输出单链表 L 中所有的项
5. 输出单链表 L 的长度
6. 将单链表 L 中的值按照升序排序
7. 将单链表中的值逆序
8. 合并两个有序的单链表
4
无参数
10 18 6 15 1
```

6. 将单链表中的值, 按照升序排序

```
6
无参数
1. 初始化链表 L, 并使用尾插法添加 n 个元素
2. 在链表 L 的索引号 arr 后面插入一个数字 x
3. 删除链表 L 中索引号为 arr 的项
4. 输出单链表 L 中所有的项
5. 输出单链表 L 的长度
6. 将单链表 L 中的值按照升序排序
7. 将单链表中的值逆序
8. 合并两个有序的单链表
4
无参数
1 6 10 15 18
```

7. 合并两个有序的单链表, 并将合并结果存储在 List 中

```
8
2 个参数: 需要和 List 归并的升序数组 b 的长度, 以及他的所有元素
5 2 7 9 20 21
1. 初始化链表 L, 并使用尾插法添加 n 个元素
2. 在链表 L 的索引号 arr 后面插入一个数字 x
3. 删除链表 L 中索引号为 arr 的项
4. 输出单链表 L 中所有的项
5. 输出单链表 L 的长度
6. 将单链表 L 中的值按照升序排序
7. 将单链表中的值逆序
8. 合并两个有序的单链表
4
无参数
1 2 6 7 9 10 15 18 20 21
```

五. 实验小结:

1. 节点结构定义与全局变量: 定义了存储整数的链表节点结构 Node 和一个全局链表指针 List。
2. 基本操作函数: 提供了链表的初始化、插入、删除、访问、长度计算、排序、逆置、合并等操作函数。

3. swap 函数：用于交换两个元素的值，辅助排序操作。
4. 链表初始化 (Init_List)：将链表的头节点的 next 指针初始化为 NULL。
5. 元素插入 (Insert_List)：在链表指定位置后插入一个新的元素，如果位置超出范围则打印错误信息。
6. 元素删除 (Delete_List)：删除链表中指定索引的节点，如果索引无效则打印错误信息。
7. 访问链表 (Visit)：遍历并输出链表中的所有元素。
8. 链表长度计算 (List_Length)：返回链表的长度。
9. 冒泡排序 (List_Bubble_Sort)：对链表元素进行升序排序，使用冒泡排序算法。
10. 链表逆置 (List_Reserve)：将链表中的元素顺序反转。
11. 链表合并 (Merge_List)：将两个有序链表合并成一个新的有序链表，并将结果存放在第一个链表中。
12. 初始化插入 (Init_Insert)：初始化链表并按顺序插入多个元素。
13. 用户交互函数 (fun)：提供一个文本菜单，用户可以选择对链表进行的操作，输入相应参数以执行功能。

实验四：

一．实验内容：

xxx

二．实验要求：

xxx

三．源程序：

xxx

四．运行结果：

截图

五．实验小结：

- (1) xxx
- (2) xxx
- (3) xxx