数据结构实验报告

<table>
<tr><td>学号-姓名</td><td>20030540015 桑龙龙</td><td>实验时间</td><td>2020 年 9 月 19 日</td></tr>
<tr><td>诚信声明</td><td colspan="3">本实验及实验报告所写内容为本人所作，没有抄袭。 桑龙龙</td></tr>
<tr><td>实验题目</td><td colspan="3">实验一 链表的实现及运算
1、 单链表基本运算
2、 单链表上的排序运算
3、 约瑟夫问题
4、 一元多项式相加、减运算器</td></tr>
<tr><td>实验过程中遇到的主要问题</td><td colspan="3">规范编程的问题，使代码的可读性和可维护性得以体现。</td></tr>
<tr><td>实验小结</td><td colspan="3">在编写代码上，发现链表的相关的问题,有一部分可以通用使用的方法,这种相关的方法可以抽象为一个整体,供多个问题进行使用，如链表创建、遍历。
考虑边际相关的问题是很重要的，如果考虑不周全，在后期使用中往往出现问题。</td></tr>
<tr><td>数据结构（自定义数据类型）</td><td colspan="3">问题 1，2，3 中的数据结构


```
struct Node
{
    int data;
    Node* next;
};
```


问题 4 中的数据结构


```
struct Node
{
    float x;        //系数
    int e;          //指数
    Node* next;
};
```

</td></tr>
</table>

主要算法（或算法说明）	问题一、单链表基本运算

```
1.  //问题一、单链表基本运算
2.  #include<stdlib.h>
3.  #include <stdio.h>
4.  using namespace std;
5.  struct Node
6.  {
7.      //链表的结构体定义
8.      int data;
9.      Node* next;
10. };
11.
12.
13. //一、实验思想：
14. //模拟
15.
16.
17. //二、主要函数
18. void head_insert(Node* root, int val);
19. //1、头插法实现函数
20. void traverse(Node* root);
21. //2、遍历函数，对 root 所指向的链表进行变量
22. void insert_element(Node* root, int index, int val);
23. //3、插入函数，对 root 所指向的链表，将值 val 插入到下标为 index 的位置
24. Node* insert_element_help(Node* last, Node* now, int val);
25. //4、插入函数的辅助函数
26. void del_element(Node* root, int index);
27. //5、删除函数，对 root 所指向的链表，删除下标为 index 的节点
28.
29.
30. int main()
31. {
32.     Node* root = (Node*)malloc(sizeof(Node));
33.     //首先生成带头结点 List
34.     root->next = NULL;
35.     int time = 20;
36.     //通过变量 time 控制初始链表的长度
37.     while(time){
38.         //通过头插法进行链表的生成
39.         head_insert(root, rand()%1000);
40.         time--;
41.     }
42.     traverse(root);                 //遍历初始链表
43.     insert_element(root, 2, 100);   //测试 1
```

```
    insert_element(root, -1, 100);  //测试 2
    insert_element(root, 100, 100); //测试 3
    //测试 1：向下标为 2 的节点插入 100
    //测试 2：边界测试，边界测试，向下标为-1 的地方插入
    //测试 3，向下标 100 的地方插入 100
    //超过链表长度，会插在链表尾部

    del_element(root, 1);       //测试 4
    del_element(root, -1);      //测试 5
    del_element(root, 100);     //测试 6
    traverse(root);             //遍历链表
    //测试 4：删除合理下标 1
    //测试 5：删除不合理下标-1
    //测试 6：删除不合理下标 100
    return 0;
}
void head_insert(Node* root, int val)
{
    Node* temp = root->next;
    root->next = (Node*)malloc(sizeof(Node));
    root->next->data = val;
    root->next->next = temp;
}
void traverse(Node* root)
{
    root = root->next;
    printf("\nTraverse begin:\nindex\tval\n");
    if(!root)
    {
        printf("empty list!\n");
        return;
    }
    int cnt = 0;
    while(root)
    {
        printf("%d\t%d\n", cnt, root->data);
        root = root->next;
        cnt++;
    }
}
Node* insert_element_help(Node* last, Node* now, int val)
{
    Node* temp=(Node*)malloc(sizeof(Node));
    temp->data = val;
    last->next = temp;
```

```
89.     temp->next= now;
90.     return temp;
91. }
92. void insert_element(Node* root, int index, int val)
93. {
94.     if(index<0)
95.     {
96.         printf("\ninsert to %d failed,out of boundary\n", index)
    ;
97.         return;
98.     }
99.     Node* last = root;
100.     Node* now = root->next;
101.     Node* temp;
102.     int cnt = 0;
103.     while(now)
104.     {
105.         if(cnt == index)
106.         {
107.             insert_element_help(last, now, val);
108.             printf("\ninsert to %d success\n", index);
109.             return;
110.         }
111.         cnt++;
112.         last = now;
113.         now = now->next;
114.     }
115.     insert_element_help(last, now, val);
116.     printf("\ninsert to %d success,insert to the last\n", index)
    ;
117. }
118. void del_element(Node* root, int index)
119. {
120.     if(index < 0)
121.     {
122.         printf("\ndelete %d failed,out of boundary\n", index);
123.         return;
124.     }
125.     Node* last = root;
126.     Node* now = root->next;
127.     int cnt = 0;
128.     while(now)
129.     {
130.         if(cnt == index)
131.         {
```

```
            printf("\ndelete %d success\n", index);
            last->next = now->next;
            free(now);
            return;
        }
        cnt++;
        last = now;
        now = now->next;
    }
    printf("\ndelete %d failed,out of boundary\n", index);
}
//实验一、单链表基本运算    结束
```

问题二、单链表上的排序运算

```
1.  //问题二、单链表上的排序运算
2.  #include<stdlib.h>
3.  #include <stdio.h>
4.  #include <algorithm>
5.  using namespace std;
6.  int arr[10000]; //排序辅助数组
7.  struct Node
8.  {
9.      //链表的结构体定义
10.     int data;
11.     Node* next;
12. };
13.
14.
15. //一、实验思想:
16. //将链表中的值存入到数组中，在数组中进行排序
17. //并重新建立链表
18.
19.
20. //二、主要函数
21. void head_insert(Node* root, int val);
22. //1、头插法实现函数
23. void traverse(Node* root);
24. //2、遍历函数，对 root 所指向的链表进行变量
25. Node* List_sort(Node* root);
26. //3、排序函数，具体实现见下方函数说明
27.
28.
29. int main()
30. {
31.     Node* root = (Node*)malloc(sizeof(Node));
32.     //首先生成带头结点 List
33.     root->next = NULL;
34.     int time = 20;
35.     //通过变量 time 控制初始链表的长度
36.     while(time)
37.     {
38.         //通过头插法进行链表的生成
39.         head_insert(root, rand()%1000);
40.         time--;
41.     }
42.     traverse(root);          //遍历生成的无序链表
43.     root = List_sort(root); //对链表排序
```

```
    traverse(root);          //对排序后的升序链表进行遍历
    return 0;
}
void head_insert(Node* root,int val)
{
    Node* temp = root->next;
    root->next = (Node*)malloc(sizeof(Node));
    root->next->data = val;
    root->next->next = temp;
}
void traverse(Node* root)
{
    root = root->next;
    printf("\nTraverse begin:\nindex\tval\n");
    if(!root){
        printf("empty list!\n");
        return;
    }
    int cnt = 0;
    while(root){
        printf("%d\t%d\n", cnt, root->data);
        root = root->next;
        cnt++;
    }
}
Node* List_sort(Node* root)
{
    //第一部分:
    //将链表中的数据存储到 arr 数组中
    int cnt = 0;
    root = root->next;
    while(root)
    {
        arr[cnt++] = root->data;
        root = root->next;
    }
    //第二部分:
    //利用 C++ sort 函数对 arr 进行排序
    sort(arr,arr+cnt,[](const int& a,const int& b){
        return a > b;
    });

    //第三部分:
    //对数组中排序好的数据进行单链表的创建
    Node* temp = (Node*)malloc(sizeof(Node));
```

```
    temp->next = NULL;
    for(int i = 0; i < cnt; i++)
    {
        head_insert(temp, arr[i]);
    }
    //第四部分:
    //返回有序链表的头结点
    return temp;
}
//问题二、单链表上的排序运算 结束
```

问题三、约瑟夫问题

```
1.  //问题三、约瑟夫问题
2.  #include<stdlib.h>
3.  #include <stdio.h>
4.  using namespace std;
5.  int password[50];
6.  struct Node
7.  {
8.      //链表的结构体定义
9.      int data;
10.     Node* next;
11. };
12.
13.
14. //一、实验思想:
15. //1、首先创建一个带头结点的环形链表
16. //2、模拟约瑟夫问题，并在此过程逐渐删除节点
17. //3、当环形链表只剩下头结点时，退出模拟
18.
19.
20. //二、主要函数
21. void head_insert(Node* root,int val);
22. //1、头插法实现函数
23. Node* find_last(Node* root);
24. //2、寻找链表 root 的最后一个节点并返回其指针
25. void yuesefu(Node* root, Node* last, int m);
26. //3、约瑟夫问题模拟函数
27.
28.
29. int main()
30. {
31.     int n = 7;
32.     //人数初始化
33.     int m = 20;
34.     //初始化报数
35.     Node* root = (Node*)malloc(sizeof(Node));
36.     root->next=NULL;
37.     //首先生成带头结点 List
38.     root->data=-1;
39.     //data=-1 表示为头结点
40.     password[1] = 3, password[2] = 1, password[3] = 7;
41.     password[4] = 2, password[5] = 4;
42.     password[6] = 8, password[7] = 4;
43.     //初始化密码
```

```
44.
45.     for(int i = n; i >= 1; i--){
46.         //通过头插法进行链表的生成
47.         head_insert(root, i);
48.     }
49.     Node* last = find_last(root);
50.     //找到最后一个节点
51.     last->next = root;
52.     //构建环形链表，使最后一个节点的 next 指向 root
53.     yuesefu(root, last, m);
54.     //模拟
55.     return 0;
56. }
57. void head_insert(Node* root,int val)
58. {
59.     Node* temp = root->next;
60.     root->next = (Node*)malloc(sizeof(Node));
61.     root->next->data = val;
62.     root->next->next = temp;
63. }
64. Node* find_last(Node* root)
65. {
66.     Node* last=root;
67.     root = root->next;
68.     while(root)
69.     {
70.         last = root;
71.         root = root->next;
72.     }
73.     return last;
74. }
75. void yuesefu(Node* root, Node* last, int m){
76.     Node* now = root;
77.     int cnt = 0;//报数器
78.     while(root)
79.     {
80.         if(root->data == -1)
81.         {
82.             //遇到头结点，不算入循环
83.             if(root == root->next) return;
84.             //如果只剩下头结点本身，退出该函数
85.             last = root;
86.             root = root->next;
87.             continue;
88.         }
```

```
89.         cnt++;
90.         if(cnt == m)
91.         {
92.             //报到 m 的人出圈
93.             printf("%d out\n", root->data);
94.             m = password[root->data];//重置 m
95.             cnt = 0;                 //重置报数器
96.             last->next = root->next;
97.             root = root->next;
98.         }else
99.         {
100.             last = root;
101.             root = root->next;
102.          }
103.     }
104. }
105.
106. //问题三、约瑟夫问题 结束
```

问题四、一元多项式相加、减运算器

```
1.  //问题四、一元多项式相加、减运算器
2.  #include<stdlib.h>
3.  #include <stdio.h>
4.  using namespace std;
5.  struct Node
6.  {
7.      //多项式链表节点的定义
8.      float x;        //系数
9.      int e;          //指数
10.     Node* next;
11. };
12.
13.
14. //一、实验思想:
15. //1、首先创建两个带头结点的链表分别表示两个多项式 a，b
16. //2、对两个链表进行合并如 a±b 就将 a 和 b 合并到新链表中
17.
18.
19. //二、主要函数
20. void insert(Node* root, float x, int e, float typ);
21. //1、插入函数，将(x,e)表示的一项合并到多项式链表 root 中
22. //其中 typ=1.0 或-1.0，表示是相加或相减
23. Node* input();
24. //2、输入函数，通过标准输入构建两个多项式
25. Node* op(Node* a, Node* b, float typ);
26. //3、相加相减运算器，如果 typ=1.0 表示多项式 a+b-
27. //如果 typ=-1.0 表示多项式 a-b
28. void traverse(Node* root);
29. //4、遍历函数
30.
31.
32. int main()
33. {
34.     Node* a = input();  //创建 a 多项式
35.     traverse(a);        //遍历 a 多项式
36.     Node* b = input();  //创建 b 多项式
37.     traverse(b);        //遍历 b 多项式
38.     Node* add = op(a, b, 1.0);
39.     //add = a + b，多项式 a+b
40.     traverse(add);      //遍历(a+b)
41.     Node* sub = op(a, b, -1.0);
42.     //sub = a - b,多项式(a-b)
43.     traverse(sub);      //遍历(a-b)
```

```
    return 0;
}
void insert(Node* root, float x, int e, float typ)
{
    Node* last=root;
    root = root->next;
    while(root)
    {
        if(root->e == e)
        {
            //如果 root 中存在指数相同的节点，那么直接相加即可
            root->x += typ * x;
            return;
        }
        last=root;
        root=root->next;
    }
    //root 中不存在指数为 e 的节点，在多项式末尾创建节点即可
    Node* temp=(Node*)malloc(sizeof(Node));
    temp->next = NULL;
    temp->e = e;
    temp->x = x;
    last->next=temp;
}
Node* input()
{
    //创建多项式链表
    //输入格式
    //第一行输入多项式项数 n
    //随后 n 行，每行形如"x e"，表示系数与指数
    Node* a = (Node*)malloc(sizeof(Node));
    a->next = NULL;
    int n, e;
    float x;
    scanf("%d\n", &n);
    for(int i = 0;i < n; i++)
    {
        scanf("%f %d\n", &x, &e);
        insert(a, x, e, 1.0);
    }
    return a;
}
Node* op(Node* a, Node* b, float typ){
    Node* root= (Node*)malloc(sizeof(Node));
    root->next=NULL;
```

```
    for(Node* i=a->next; i; i = i->next)
    {
        insert(root, i->x, i->e, 1.0);
    }
    for(Node* i=b->next; i; i = i->next)
    {
        insert(root, i->x, i->e, typ);
    }
    return root;
}
void traverse(Node* root)
{
    root = root->next;
    printf("\nTraverse begin:\ncoefficient\texponent\n");
    if(!root)
    {
        printf("empty list!\n");
        return;
    }
    while(root)
    {
        if(root->x != 0.0)printf("%f\tx**%d\n", root->x, root->e
);
        root = root->next;
    }
}
//问题四、一元多项式相加、减运算器 结束
```