北京師範大學
BEIJING NORMAL UNIVERSITY

《数据结构》上机实验报告

第 1 次上机

学号：202011140104
姓名：李馨
学院：物理学系
专业：物理学
教师：郑新
日期：2022.9.9

一、实验要求

1. 上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2. 独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3. 上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

二、实验过程

a）问题一：利用顺序表的操作，实现以下的函数。

i. 从顺序表中删除具有最小值的元素并由函数返回被删元素的值。空出的位置由最后一个元素填补，若顺序表为空则显示出错信息并退出运行。
ii. 从顺序表中删除第 i 个元素并由函数返回被删元素的值。如果 i 不合理或顺序表为空则显示出错信息并退出运行。
iii. 向顺序表中第 i 个位置插入一个新的元素 x。如果 i 不合理则显示出错信息并退出运行。
iv. 从顺序表中删除具有给定值 x 的所有元素。
v. 从顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素。如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。
vi. 从有序顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素。如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。
vii. 将两个有序顺序表合并成一个新的有序顺序表并由函数返回结果顺序表。
viii. 从顺序表中删除所有其值重复的元素，使表中所有元素的值均不相同。

b）问题一：问题分析与关键代码

i. **最小值删除**

从顺序表中删除具有最小值的元素并由函数返回被删元素的值。空出的位置由最后一个元素填补，若顺序表为空则显示出错信息并退出运行。

1. 对顺序表进行逐个扫描，储存最小值及其索引，然后返回最小值，并根据索引删除该元素即可。
2. 扫描并储存最小值及其索引：

```
DataType min = L.data[0]; // 储存最小值
int MinIndex = 0;           // 储存最小值的索引
for (int i = 1; i < L.n; i++) {
    if(L.data[i] < min) {
        min = L.data[i];
        MinIndex = i;
    }
}
```

ii. 删除元素

从顺序表中删除第 i 个元素并由函数返回被删元素的值。如果 i 不合理或顺序表为空则显示出错信息并退出运行。

1. 第 i 个元素的索引是 i-1，让该元素之后的元素全部前移即可。
2. 注意定义的顺序表中有元素个数的数据，删除后需要减一。
3. 该部分主要代码：

```
DataType ReturnValue = L.data[i-1]; // 被删元素的值，用于函数返回变量
for (int j = i-1; j < L.n - 1; j++) {
    L.data[j] = L.data[j+1];        // 依次将后面元素前移
}
L.n--;                              // 当前表元素个数-1
```

iii. 插入元素

向顺序表中第 i 个位置插入一个新的元素 x。如果 i 不合理则显示出错信息并退出运行。

1. 将该元素插入位置（索引为 i-1）之后的所有元素后移，然后在索引 i-1 处插入该元素。
2. 注意后移时要从最后一个元素开始，逆序进行。顺序进行后移事实上把后一个元素覆盖了，造成元素丢失。
3. 该部分主要代码：

```
// 后续元素后移
for (int j = L.n-1; j>=i-1; j--) L.data[j+1] = L.data[j];
L.data[i-1] = x;    // 在第 i 个位置插入 x
L.n++;              // 元素个数+1
```

iv. 删除给定值所有元素

从顺序表中删除具有给定值 x 的所有元素。

1. 对顺序表进行逐个扫描，如果其元素的值等于给定值，进行删除操作。
2. 删除操作的实现我直接调用了删除元素的 remove 函数。
3. 在扫描循环中，调用后顺序表元素个数-1，扫描指针（数组索引）需要回退一个，否则会跳过一个元素没有扫描。
4. 该部分主要代码：

```
for (int i = 0; i < L.n; i++) {
    if(L.data[i] == x) {
        remove(L, i+1); // 用 remove 函数移除给定值元素
        i--;            // 索引回退 1 个，否则下次循环会跳过一个元素
    }
}
```

v. 删除给定区间值的元素

从顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素。如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。

1. 对顺序表进行逐个扫描，如果其元素的值在给定区间内，进行删除操作。
2. 与上一个问题代码相同，只改变删除条件即可。
3. 该部分主要代码：

```
for (int i = 0; i < L.n; i++) {
    if(L.data[i] > s && L.data[i] < t) {
        remove(L, i+1);     // 用 remove 函数移除满足条件的元素
```

```
            i--;                // 回退
        }
    }
```

vi. 在有序顺序表中删除给定区间值的元素

从有序顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素。如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。

1. 有序顺序表中，给定区间的元素在表中是连续的。通过扫描找到这一系列元素的头和尾的地址，然后让这一系列元素之后的元素全部前移，达到删除的效果。
2. 如果使用 remove 函数一次删除一个，则每次删除都需从头扫描一遍顺序表，时间复杂度达到 $O(n^2)$，这是不必要的。由于是有序顺序表，扫描一次即可确认所有满足要求的元素的地址。
3. 对于“所有元素都小于 s”和“大于 s 的元素都比 t 小”的两种情况要特别处理。
4. 删除部分主要代码，其中 i、j 分别是区间值内第一个元素和最后一个元素在数组中的索引：

```
    int d = j - i + 1;// 删除元素的个数，也是删除之后，其后的元素要前移的位置大小
    for (int k = i; k + d < L.n; k++) {
        /* k = i: 从位置 i 处开始将其后第 d 个元素前移至此位置
         * k + d < L.n: 最后一次循环时，原表位置 n-1 处移至新表位置 n-1-d 处（即新表表尾） */
        L.data[k] = L.data[k+d];            // 依次将被删区间之后的元素前移
    }
    L.n -= d;       // 顺序表元素个数减少 d 个
```

vii. 合并有序顺序表

将两个有序顺序表合并成一个新的有序顺序表并由函数返回结果顺序表。

1. 如果允许对原顺序表进行改变，则这一合并过程可以通过不断比较两个顺序表的首元、摘取（存到结果表里，然后从原表删除）较小首元到结果表里来实现。
2. 但还是尽量不改变原顺序表比较好，所以上述过程只取不摘，扫描指针指到的元素插入到结果表中后不进行删除，直接指向下一个元素。
3. 当至少一个顺序表没有被扫描完时，每次循环只会有两种操作，要么是把第一个表的扫描指针指向的元素插入结果表并前进一个地址，要么是第二个。因此，在 if 的条件设置中只需考虑全面其中一种情况，另一种写 else 即可。
4. 该部分主要代码：

```
int i1 = 0, i2 = 0;
// 将 List1, List2 从位置索引 i1, i2 处截断，其之前的元素不再考虑，考虑剩余的
while (i1 < List1.n || i2 < List2.n) {
// 这保证了 List1 和 List2 至少有一个有“剩余”元素
    if(i1 < List1.n && (i2 == List2.n || List1.data[i1] < List2.data[i2])) {
    // “List1 仍有剩余元素”且“其首元小于 List2 剩余元素首元，或 List2 没有剩余元素”
        insert(ResultList, ResultList.n + 1, List1.data[i1]);
        // 这时将 List1 的“首元”插入结果表尾
        i1 += 1;    // 取 List1 的下一个“首元”
    }
    else {
         insert(ResultList, ResultList.n + 1, List2.data[i2]);
        // 这时将 List1 的“首元”插入结果表尾
        i2 += 1;    // 取 List2 的下一个“首元”
```

```
        }
    }
```

viii. **删除重复元素**

从顺序表中删除所有其值重复的元素，使表中所有元素的值均不相同。

1. 从第一个元素开始，对每一个元素，寻找其后面有没有与其值相同的元素，如有则进行删除操作。这需要两层循环，时间复杂度 $O(n^2)$。
2. 主要部分代码：

```
    for (int i = 0; i < L.n; i++) {     // 扫描元素，被删除的元素不会再被扫描
        for (int j = i + 1; j < L.n; j++) { // 将 i 元素与之后的所有元素进行对比
            if (L.data[i] == L.data[j]) {
                // 如果 j 元素与 i 元素相同，删除 j 元素，即第 j+1 个元素
                remove(L, j+1);
                j--;   // 删除后索引 j 回退一个，否则下次循环就跳过了一个元素
            }
        }
    }
```

c) 问题一：运行结果

i. 建立整数顺序表

TestList1 = [-1, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1],

TestList2 = [2, 4, 6, 8, 9, 9, 10, 12].

ii. **removeMin**, **Insert**, **Remove**, **removeAll** 的测试，注释为预期的输出结果：

```
    printf("%d\n",removeMin(TestList1));    // -1
    printIntList(TestList1);
    // [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    insert(TestList1, 2, 100);
    printIntList(TestList1);
    // [1, 100, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    remove(TestList1, 2);
    printIntList(TestList1);
    // [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    removeAll(TestList2, 9);
    printIntList(TestList2);
    // [2, 4, 6, 8, 10, 12]
```

实际运行结果与预期相符：

```
    printf("%d\n",removeMin(TestList1));    // -1
    printIntList(TestList1);                // [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    insert(TestList1, 2, 100);
    printIntList(TestList1);                // [1, 100, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    remove(TestList1, 2);
    printIntList(TestList1);                // [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]

    removeAll(TestList2, 9);
    printIntList(TestList2);                // [2, 4, 6, 8, 10, 12]
```

问题 终端 JUPYTER

```
-1
[1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[1, 100, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[2, 4, 6, 8, 10, 12]
```

此时 TestList1 = [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1], TestList2 = [2, 4, 6, 8, 10, 12].

iii. **removeBetween**, **removeSortedBetween**, **removeRepeated**, **mergeList** 的测试，注释为预期的输出结果（沿用之前被处理过的 TestList1 和 TestList2）：

```
    removeBetween(TestList1, 2, 6);
    printIntList(TestList1);
    // [1, 2, 6, 2, 7, 8, 1]

    removeSortedBetween(TestList2, 2, 11);
    printIntList(TestList2);
    // [2, 12]

    removeRepeated(TestList1);
    printIntList(TestList1);
    // [1, 2, 6, 7, 8]

    printIntList(mergeList(TestList1, TestList2));
    // [1, 2, 2, 6, 7, 8, 12]
```

实际运行结果与预期相符：

```
    removeBetween(TestList1, 2, 6);
    printIntList(TestList1);                // [1, 2, 6, 2, 7, 8, 1]

    removeSortedBetween(TestList2, 2, 11);
    printIntList(TestList2);                // [2, 12]

    removeRepeated(TestList1);
    printIntList(TestList1);                // [1, 2, 6, 7, 8]

    printIntList(mergeList(TestList1, TestList2)); // [1, 2, 2, 6, 7, 8, 12]

    return 0;
}
```

问题 终端 JUPYTER

```
-1
[1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[1, 100, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 7, 8, 1]
[2, 4, 6, 8, 10, 12]
[1, 2, 6, 2, 7, 8, 1]
[2, 12]
[1, 2, 6, 7, 8]
[1, 2, 2, 6, 7, 8, 12]
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Micros
t-brge4xxn.ojv"
```

d) 问题二：编写一组程序，用**带头结点的有序单链表**实现一个集合，放在“SetList.h”中。要求实现这个集合的结构和相关的操作。

至少应包括初始化，用尾插法建立集合，查找给定元素是否在集合内，将新元素插入集合中，删除集合中指定元素，求两个集合的并、交、差、输出等操作的实现。要求设计一个主程序，首先定义 3 个用有序单链表实现的 A、B、C 并初始化为空集合，其元素的数据类型为整型；再依次读入若干整数创建一个集合 A 和 B，并检查集合上的查找、插入、删除等操作的实现，存入 C 中并输出它。

e) 问题二：问题分析与关键代码

i. **初始化、查找元素、新元素插入、删除元素**

1. 这些代码在教材上都有实现，因此简略带过：
   a) 初始化时注意因为是有头结点的单链表，故而不是 `S = NULL;`
   而是

```
S = (SetNode *) malloc (sizeof(SetNode));
S->link = NULL;
```

   b) 注意查找时利用有序性，插入和删除时不改变有序性。要定位一个元素应在的位置，要找到有序单链表中最后一个小于它的结点，扫描指针 p 及其前驱 pr 应经历这样的循环：

```
SetNode *p = S->link, *pr = S;
while (p != NULL && p->data < x) {
    pr = p;
    p = p->link;
}
```

ii. **尾插法建立集合**

1. 事实上插入元素后为了保持集合的的有序性，并不是在单链表尾部插入。相较插入，对其主要代码增加输入语句和直到输入 endTag 才结束的循环即可。每次循环后要对扫描指针及其前驱进行初始化。
2. 主要代码：

```
void insertRear(LinkedSet& S, DataType endTag) {
    DataType val;
    SetNode *p = S->link, *pr = S;          // p 是扫描指针，pr 是 p 的前驱
    scanf("%d", &val); // 读入一数据
    while (val != endTag) {
        while (p != NULL && p->data < val) { // 先进行定位查找
            pr = p;
            p = p->link;
        }
        if (p != NULL && p->data == val) {
            // 如果集合中已存在此元素，读入下一数据，并直接进行下一次循环
            p = S->link, pr = S;
            scanf("%d", &val);
            continue;
        }
        SetNode *q = (SetNode *) malloc (sizeof(SetNode));
```

```
        q->data = val; q->link = p; pr->link = q;   // 新元素链入
        p = S->link, pr = S;
        scanf("%d", &val);          // 读入下一数据
    }
}
```

iii. 集合的并

1. 与“合并有序顺序表”大致思路是相同的，只不过不再是顺序表而是单链表，将数组索引的语句相应翻译成指针即可。
2. 主要代码：

```
    SetNode *pa = LA->link, *pb = LB->link; // 分别是LA和LB的扫描指针
    while (pa != NULL || pb != NULL) {
        // 事实上if和else if的条件应该互为补集，else if可写为else，但为清晰起见还是写明
        if (pa != NULL && (pb == NULL || pa->data < pb->data)) {
            // LA没被扫描完，且“LA的扫描元素小于LB的扫描元素或LB已经被扫描完”
            addMember(LC, pa->data);    // 将此时LA的扫描元素插入LC
            pa = pa->link;              // LA继续扫描下一元素
        }
        else if (pb != NULL && (pb->data <= pa->data || pa == NULL)) {
            addMember(LC, pb->data);
            // addMember已经考虑了数据相等时不插入的情况
            pb = pb->link;              // LB继续扫描下一元素
        }
    }
```

iv. 集合的交和差

1. 对其中一个集合的元素依次进行扫描，每扫描到一个元素，将之数据与另一集合的每个元素对比。对于交集，相同的元素插入初始为空的结果集合中；对于差集，插入的是在另一集合查找不到的元素。
2. 针对另一集合的扫描指针在每次外层循环结束前需要初始化。
3. 差集时对内层扫描循环的处理：

```
    while (pb != NULL) {
        // 扫描LB，如果LB中有与LA此时扫描元素相同的元素，则退出循环
        if (pa->data == pb->data) break;
        pb = pb->link;      // 全部都不相同才会最终到NULL
    }
    if (pb == NULL) addMember(LC, pa->data); // 都不相同，将该LA元素插入LC
```

v. 输出集合

1. 让扫描指针扫描集合，读取集合每个元素的数据打印即可。
2. 我选取的是(elem1, elme2, ...)的打印格式。针对最后一个元素（即 p->link == NULL 的元素）进行条件判断，不要打逗号和空格。
3. 代码：

```
void printSet(LinkedSet S) {
    SetNode *p = S->link;     // 扫描指针，从首元结点（非头结点）开始打印
    printf("(");
```

```
    while (p != NULL) {
        printf("%d",p->data);
        if (p->link != NULL) printf(", ");
        p = p->link;
    }
    printf(")\n");
}
```

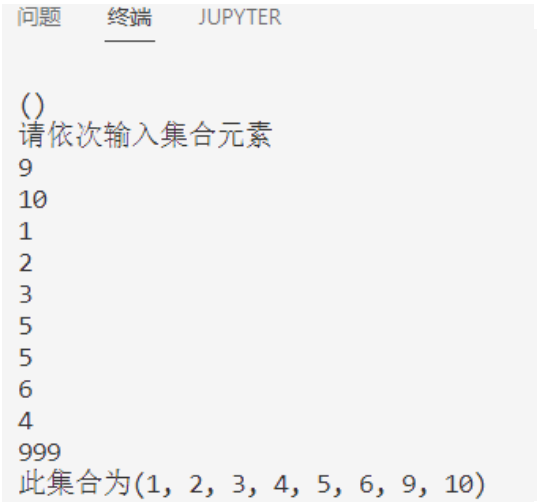
f) 问题二：运行结果

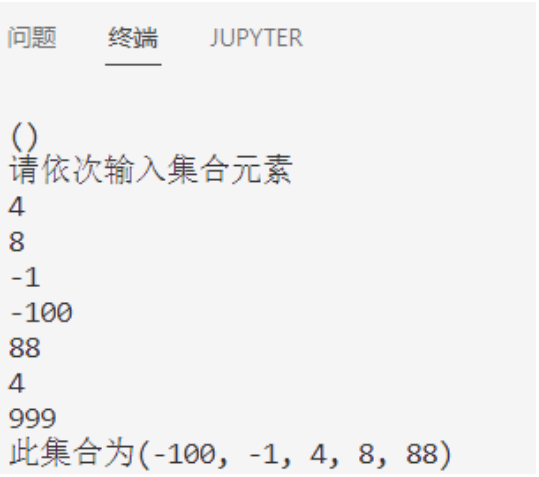
i. **initSet, insertRear, printSet** 的测试，其中建立集合时以 999 为 endTag。

```
// initSet, insertRear, printSet 的测试
LinkedSet S;
initSet(S); printSet(S);
printf("请依次输入集合元素\n"); insertRear(S, 999);
printf("此集合为"); printSet(S);
```

在 **initSet(S); printSet(S);** 后应输出 **()** ，即打印空集。

在执行 **insertRear(S, 999);** 后依次输入集合元素（无论大小是否有序或者是否有重复）和 endTag，程序会排列好这个有序集合并输出：





ii. 之后的测试中，初始设定集合

A = (-2, 1, 2, 4, 5, 7, 8, 9, 11),

B = (-2, -1, 1, 3, 5, 6, 7, 9, 10),

C = ().

iii. **Contains, addMember, delMember** 的测试，注释为预期结果：

```
printf("%d 是 A 集合中的元素\n", Contains(A, 2)->data);
addMember(A, 100); printf("A = A + {100} = "); printSet(A);
// (-2, 1, 2, 4, 5, 7, 8, 9, 11, 100)
delMember(B, 5); printf("B = B - {5} = "); printSet(B);
// (-2, -1, 1, 3, 6, 7, 9, 10)
```

其中 Contains 返回 A 集合中元素“2”的地址。运行结果与预期结果一致：

```
问题    终端    JUPYTER
A = (-2, 1, 2, 4, 5, 7, 8, 9, 11)
B = (-2, -1, 1, 3, 5, 6, 7, 9, 10)
· 属于、增加和删除元素测试
2是A集合中的元素
A = A + {100} = (-2, 1, 2, 4, 5, 7, 8, 9, 11, 100)
B = B - {5} = (-2, -1, 1, 3, 6, 7, 9, 10)
```

运行后得到的集合沿用至下一测试，

A = (-2, 1, 2, 4, 5, 7, 8, 9, 11, 100),

B = (-2, -1, 1, 3, 6, 7, 9, 10),

C = ().

iv. **Merge**, **Intersect**, **Diff** 的测试，C 为储存结果的集合，每次输出后释放并初始化。注释为预期结果：

```
Merge(A, B, C); printf("A + B = "); printSet(C);
// (-2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 100)
free(C); initSet(C);

Intersect(A, B, C); printf("A ^ B = "); printSet(C);
// (-2, 1, 7, 9)
free(C); initSet(C);

Diff(A, B, C); printf("A - B = "); printSet(C);
// (2, 4, 5, 8, 11, 100)
free(C);
```

运行结果与预期结果一致：

```
问题    终端    JUPYTER
· 并、交、差集测试
A + B = (-2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 100)
A ^ B = (-2, 1, 7, 9)
A - B = (2, 4, 5, 8, 11, 100)
[1] + Done                    "/usr/bin/gdb" --interpr
```

三、总结（实验中遇到的问题、取得的经验、感想等）

①本次实验让我印象较深的是合并有序顺序表/求有序单链表集合的并集的问题，这两个问题本质是相似的，只不过一个是利用顺序表实现，另一个是使用链表实现。这加深了我对顺序表和链表的联系和区别的感受和理解。比如，链表扫描指针的 p = p->link; 与 数组索引的 i++; 的相通；判断到达表尾，链表是 p->link == NULL; ，顺序表则是 I == L.n;。它们各自独特的特点在这个问题上并没有过多体现，程序设计上基本没有差异。

②比较印象深刻的还有头结点的作用。在链表操作中反复需要用到扫描指针 p 和其前驱 pr，如果没有头结点，针对不同问题空表时需要做各种特殊处理，与通常非空表情况的操作不统一。有了头结点后，初始时就可以直接 p = S->link; pr = S; 而无需特别处理空表的情况。这是非常便利的。

③本次上机实验主要还是通过实践加深了我对顺序表和链表各方面的理解、把握和印象。