数据结构与算法 第六次实验

学号：37220232203808　　姓名：宋浩元

一、 实验目的

1. 详细掌握排序算法的实现方法与原理，实践操作编写实际应用的问题
2. 了解设计各种排序存储、查询、交换数据算法的实现方法与原理，在理解的基础上学习代码编写
3. 学会灵活按照树形选择排序的方式自由编写算法结构代码，学会相关的处理方式

二、 实验内容

6-1 荷兰国旗问题：设有一个仅由红、白、蓝三种颜色的条块组成的序列。试设计一个时间复杂度为 O(n)的算法，使得这些条块按红、白、蓝的顺序排好，即排成荷兰国旗图案。

编写思路：

定义一个结构体 SqList，其中包含一个字符数组 r 和一个整型变量 length。字符数组 r 存储着输入的颜色串行，整型变量 length 存储着输入的串行长度。然后定义一个函数 solve，该函数接受一个结构体变量 L 作为参数，并对 L 中的串行进行排序。

在程序的主函数 main 中，首先读入串行的长度并存储到结构体变量 L 的 length 字段中。然后读入串行的各个元素并存储到 L 中的 r 字段中。最后调用函数 solve 对串行进行排序，并将排序后的串行输出。

```c
#include<stdio.h>
#include<stdlib.h>

// 定义一个最大长度为 100 的字符数组
#define maxsize 100

// 定义一个结构体 SqList，包含一个字符数组 r 和一个整型变量 length
typedef struct {
    char r[maxsize];
    int length;
} SqList;
```

```c
// 函数 solve，接受一个结构体变量 L 作为参数，并对 L 中的序列进行排序
void solve(SqList &L) {
    int i,j,k;
    char temp;

    // 初始化 i、j、k 三个变量的值
    i=k=0;
    j=L.length-1;

    // 当 i 小于 j 时，循环执行以下操作
    while(i<j) {
        // 当 i 小于 j 且 L.r[i] 为 'W' 时，执行 i++ 操作
        while(i<j&&L.r[i]=='W') i++;

        // 当 L.r[i] 为 'R' 时，执行交换操作，将 L.r[i] 与 L.r[k] 交换，然后执行 i++、k++ 操作
        if(L.r[i]=='R') {
            temp=L.r[i];
            L.r[i]=L.r[k];
            L.r[k]=temp;
            i++;
            k++;
        }
        // 当 L.r[i] 为 'B' 时，循环执行以下操作
        else if(L.r[i]=='B') {
// 当 i 小于 j 且 L.r[j] 为 'B' 时，执行 j-- 操作
            while(i<j&&L.r[j]=='B') j--;
// 将 L.r[j] 与 L.r[i] 交换，然后执行 j-- 操作
            temp=L.r[j];
            L.r[j]=L.r[i];
            L.r[i]=temp;
            j--;
        }
    }

}
int main() {
    SqList L;
    int i;
// 读入序列的长度并存储到 L.length 中
    scanf("%d",&L.length);
    getchar();

// 读入序列的各个元素并存储到 L.r 中
    for(i=0; i<L.length; i++) {
        scanf("%c",&L.r[i]);
    }

// 调用函数 solve，对序列进行排序
    solve(L);

// 输出排序后的序列
    for(i=0; i<L.length; i++) {
        printf("%c ",L.r[i]);
    }

    return 0;
}
```

6-2 假设 n 个部门名称的基本数据存储在字符数组 name[N][25]中，5≤n≤N≤20。试设计一个起泡排序算法，将 n 个部门名称按字典序重新排列顺序。

```
//附：随机产生 n 个部门名称
void Names(char A[][25],int n)
{
    srand(time(0));
    int i,j,k;
```

```
        for(i=0;i<n;i++)
        {
            k=2*(rand()%10+3);   //部门字数
            for(j=0;j<k;j++)
                A[i][j]=rand()%30+176; //汉字区
            A[i][j]='\0';
        }
    }
```

编写过程：

```
7   char name[20][25];
```

定义了一个二维字符数组 name，用于存储随机生成的字符串。

```
8   void Names(char A[][25],int n)
9   {
10      srand(time(0));
11      int i,j,k;
12      for(i=0;i<n;i++)
13      {
14          k=2*(rand()%10+3);  //部门字数
15          for(j=0;j<k;j++)
16              A[i][j]=rand()%30+176; //汉字区
17          A[i][j]='\0';
18      }
19  }
```

然后是第一个函数 Names，它接受两个参数：一个二维字符数组 A 和一个整数 n。该函数的功能是生成 n 个随机的汉字字符串，并将这些字符串存储在二维字符数组 A 中。

首先调用了 srand 函数，使用当前时间作为随机数种子，保证了每次生成的随机数不同。

然后使用一个外层循环遍历 n 个字符串，对于每一个字符串，使用一个循环。然后，内层循环使用 rand 函数生成一个随机数，将其乘以 2 后再加上 3，得到了当前字符串的字数。然后使用内层循环依次生成字符串的每一个字符，将生成的随机数加上 176 后转换成字符，这样就得到了一个随机的汉字字符。最后，将生成的字符串的末尾添加一个空字符，表示字符串的结束。

```
20  void NameSort(char name[][25], int n) {
21    // 循环 n-1 次
22    for (int i = 0; i < n - 1; i++) {
23      // 循环 n-1-i 次，每次把最大的数放到最后
24      for (int j = 0; j < n - 1 - i; j++) {
25        // 如果前一个数大于后一个数，就交换它们
26        if (strcmp(name[j], name[j + 1]) > 0) {
27          char temp[25];
28          strcpy(temp, name[j]);
29          strcpy(name[j], name[j + 1]);
30          strcpy(name[j + 1], temp);
31        }
32      }
33    }
34  }
```

接下来是第二个函数 NameSort，它接受两个参数：一个二维字符数组 name 和一个整数 n。该函数的功能是将 name 数组中的字符串按字典序排序。
该函数使用了冒泡排序的思想，使用一个外层循环遍历前 n - 1 个字符串，内层循环遍历前 n - 1 - i 个字符串，每次比较相邻的两个字符串，如果前一个字符串大于后一个字符串，就交换它们的位置。

```
35 int main(){
36     Names(name,20);
37     cout << "排序前：" << endl;
38     for(int i=0;i<20;i++)cout << name[i] << endl;
39     NameSort(name,20);
40     cout << "\n\n排序后：" << endl;
41     for(int i=0;i<20;i++)cout << name[i] << endl;
42 }
```

最后是主函数 main，它调用了前面定义的两个函数。首先调用 Names 函数生成 20 个随机的汉字字符串，然后使用 cout 输出这些字符串。接着调用 NameSort 函数对字符串进行排序，最后再使用 cout 输出排序后的字符串。

6-3 设计基于顺序表存储结构的树形选择排序算法。

编写思路：

1. 将数组中的所有元素构建成一颗二叉树，这个过程叫建树。
2. 从树的顶部（根节点）取出最小的元素，将它与数组最后一个元素交换位置，再将树的大小减 1。
3. 重复这个过程直到树的大小为 0，这样就可以完成排序。

代码：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fixtree(int* tree, int treeSize, int index) {
5      // 左子节点的索引
6      int left = index * 2 + 1;
7      // 右子节点的索引
8      int right = index * 2 + 2;
9      // 最小值的索引，初始化为当前节点
10     int min = index;
11     // 如果左子节点存在，并且比当前最小值小，更新最小值的索引
12     if (left < treeSize && tree[left] < tree[min]) {
13         min = left;
14     }
15     // 如果右子节点存在，并且比当前最小值小，更新最小值的索引
16     if (right < treeSize && tree[right] < tree[min]) {
17         min = right;
18     }
19     // 如果最小值的索引发生变化，说明子节点中有比堆顶元素更小的
20     if (min != index) {
21         // 交换顶元素和最小值
22         int temp = tree[index];
23         tree[index] = tree[min];
24         tree[min] = temp;
25         fixtree(tree, treeSize, min);
26     }
27 }
```

在这段程序中，函数 fixtree 用来修复二叉树，使之满足堆的性质。当一个节点的值发生变化时，它可能会破坏堆的性质，因此需要调用该函数将它重新调整为一个合法的堆。

```
// 建立树
void buildtree(int* tree, int treeSize) {
    // 从最后一个非叶子节点开始，向前依次处理
    for (int i = treeSize / 2 - 1; i >= 0; i--) {
        fixtree(tree, treeSize, i);
    }
}
```

函数 buildtree 用来建树，它首先对数组中的所有元素构建一颗二叉树，然后调用函数 fixtree 依次修复每一个节点。

```
// 树形选择排序
void treeSelectSort(int* array, int size) {
    // 建立树
    buildtree(array, size);
    // 循环取出顶元素，并将树大小减 1
    for (int i = size - 1; i >= 0; i--) {
        // 将顶元素与数组最后一个元素交换
        int temp = array[i];
        array[i] = array[0];
        array[0] = temp;
        // 将树的大小减 1
        size--;
        // 继续处理树
        fixtree(array, size, 0);
    }
}
```

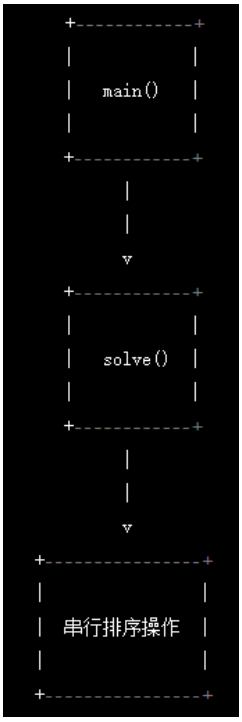
函数 treeSelectSort 是主要的排序函数，它调用函数 buildtree 建树，然后循环取出树的顶元素，并将树的大小减 1，最后调用函数 fixtree 修复树。

```
int main(void) {
    int array[] = {5, 2, 9, 4, 7, 6, 1, 3, 8};
    int size = sizeof(array) / sizeof(int);
    // 调用树形选择排序算法
    treeSelectSort(array, size);
    // 输出排序后的结果
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}
```

最后，在 main 函数中，调用函数 treeSelectSort 对数组进行排序，然后输出排序后的结果。

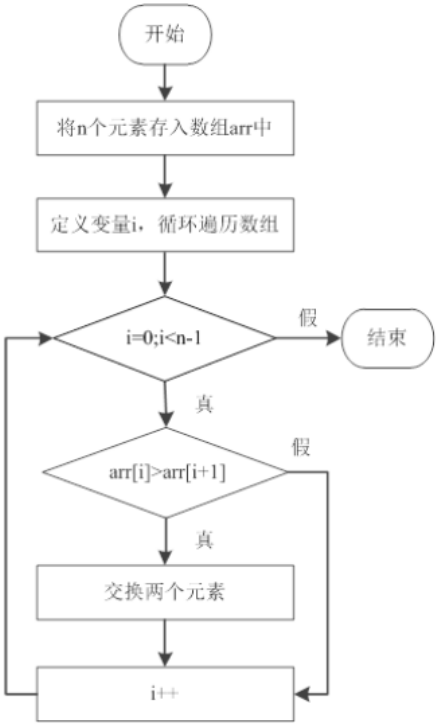
三、 主要算法流程图

6-1 算法流程



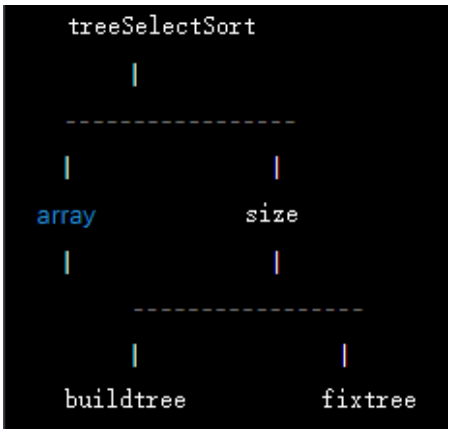
在 main 函数中，程序会先读入串行的长度和各个元素，然后调用 solve 函数对串行进行排序。在 solve 函数中，程序会执行串行排序操作。

6-2 算法流程



在 6-2 中，我使用了冒泡排序法对 20 个汉字字符串进行排序。

6-3 算法流程



上述流程实现了树形选择排序算法，它的具体文字流程如下：

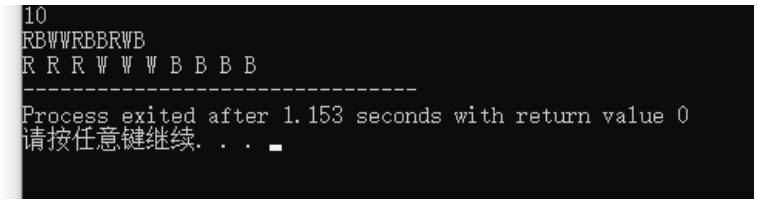
1. 创建二叉树：首先，调用函数 buildtree 将数组中的所有元素构建成一颗二叉树。在这个过程中，会依次调用函数 fixtree 修复每一个节点，使之满足堆的性质。
2. 取出树的顶部元素：然后，调用函数 treeSelectSort 开始排序。在这个函数中，会循环取出树的顶部元素，并将树的大小减 1。
3. 修复树：每次取出树的顶部元素，都会导致树的顶部（根节点）发生变化，因此需要调用函数 fixtree 修复树。这样才能继续取出树的顶部元素，并将树的大小减 1。
4. 重复步骤 2 和 3，直到树的大小为 0，排序结束。

最后，在 main 函数中，调用函数 treeSelectSort 对数组进行排序，然后输出排序后的结果。

四、 实验结果：

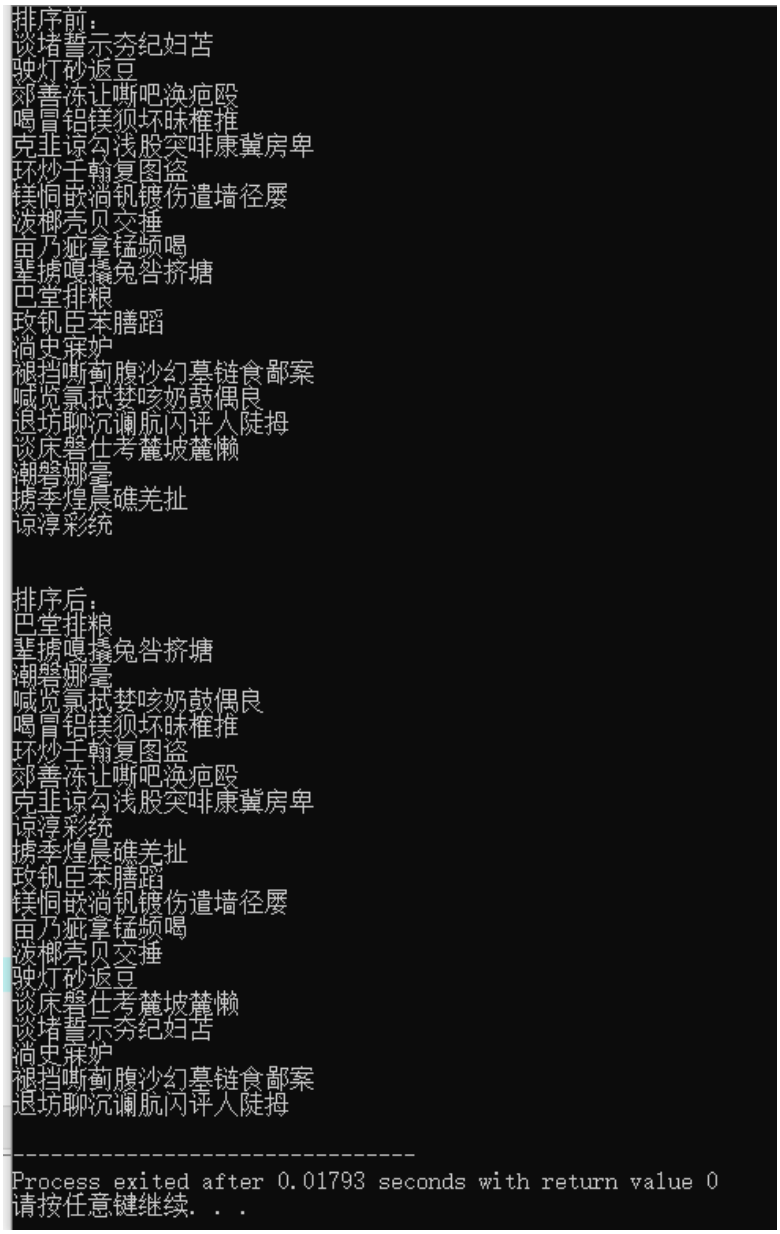
（结合截图说明算法的输入输出）

1、关于 6-1 的输入与输出：



在实际运行中，我创建了 RBWWRBBRWB 的序列，程序成功地读取了序列并按照了“R-W-B”的顺序排序完毕。

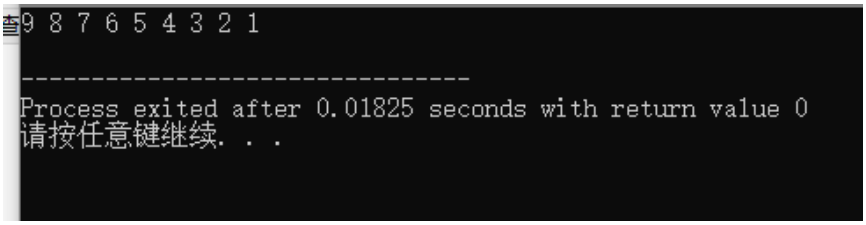
2、关于 6-2 的输入与输出



在实际运行中，程序随机生成了 20 个随机的汉字字符串，然后将这 20 个字符串按字典序排序后输出。

3、关于 6-3 的输入与输出

```
int array[] = {5, 2, 9, 4, 7, 6, 1, 3, 8};
```



可以看出程序成功的将数组中的元素按照降序排序了。

五、 实验小结（即总结本次实验所得到的经验与启发等）：

在本次实验中，我尝试具体运用了冒泡排序、树形选择排序，并在实际问题中分析问题、解决了问题，在实体机的实验中我能够更深刻地理解对这一部分算法的执行方式与特点，并且在编写代码的过程中，我通过不断的调试去寻找语句之间的问题和不足，在潜移默化中提高了我的代码编写能力，这是一次完成效果良好的实验！