C语言第六章作业-2

顺序查找

> 【问题描述】从键盘输入10个整数，保存到数组中；再输入一个整数x，利用顺序查找法查询整数x在数组中的位置，如果查询到则输出x在数组中的位置，如果未查询到，输出“Not Found”。
> 【输入输出样例1】
> `21 2 36 47 5 65 7 -89 9 100`
> `7`
> `Index is 6` (输出结束不换行)
> 【输入输出样例2】
> `21 2 36 47 5 65 7 -89 9 100`
> `12`
> `Not Found` (输出结束不换行)

解析

- 由于我们输入的数据是无序的，所以最优的方式就是顺序查找
- 实现顺序查找也十分简单：
    - 遍历整个数组直至找到目标数字。找到后返回下标。

示例代码

```c
#include <stdio.h>
int main() {
    int i, x, found = 0, index = -1, nums[11];
    for (i = 0; i < 10; i++) //读入初始数据
        scanf("%d", &nums[i]);
    scanf("%d", &x);//读入寻找的数据
    for (i = 0; i < 10; i++) {//顺序查找
        if (nums[i] == x) {
            found = 1;
            index = i;//找到返回下标
            break;
        }
    }
    if (found)
        printf("Index is %d", index);
    else
        printf("Not Found");
    return 0;
}
```

二分查找

> 【问题描述】二分查找。给定一个有序的数列，查找指定的数值。如果查询到该数值，则返回该数值在数组中的位置。要求：利用数组初始化方法给各数组元素赋值，数组长度为10；输入一个整数x，利用二分查找法查询整数x在数组中的位置，如果查询到则输出x在数组中的位置，如果未查询到，输出“Not Found”。（数组初始值为1 2 3 4 5 6 7 8 9 10）
> 【输入输出样例1】
> `8`
> `Index is 7` （输出结束换行，数值后无空格）
> 【输入输出样例2】
> `12`
> `Not Found` （输出结束换行）

解析

- 由于最开始的数据是有序的，我们可以选用二分查找，在数据有序的情况下它的效率会比顺序查找快上不少。
- 二分查找是一种高效的搜索算法，其核心思想是将搜索范围逐渐缩小至最终找到目标元素或确定目标元素不存在。
    - 有序数组：二分查找要求在有序数组中进行搜索。这是因为有序数组具有一定的结构，可以利用元素之间的相对大小关系来快速缩小搜索范围。
    - 初始化搜索范围：初始时，将整个数组视为搜索范围。设置左边界 `left` 为数组的第一个元素的索引，右边界 `right` 为数组的最后一个元素的索引。
    - 计算中间元素：通过计算左边界和右边界的中间元素的索引 `mid`，可以将搜索范围分为两部分。这里通常使用 `(left + right) / 2` 来计算中间元素的索引。
    - 比较中间元素：将中间元素与目标元素进行比较。
        - 如果中间元素等于目标元素，则找到了目标元素，搜索结束。
        - 如果中间元素大于目标元素，说明目标元素可能在左侧，所以将搜索范围缩小为左半部分。
        - 如果中间元素小于目标元素，说明目标元素可能在右侧，所以将搜索范围缩小为右半部分。
    - 更新搜索范围：根据中间元素与目标元素的比较结果，更新左边界或右边界，以便进一步缩小搜索范围。
    - 重复步骤 3~5：在每次迭代中，都会将搜索范围缩小一半，直到找到目标元素或确定目标元素不存在。循环条件通常是 `left <= right`。
    - 结束条件：当搜索范围缩小到左边界大于右边界时，说明目标元素不在数组中，搜索结束。

示例代码

```c
#include <stdio.h>
int main() {
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int target, left = 0, right = 9, result = -1;
    scanf("%d", &target);
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target){
            result = mid;
            break;
        }
        else if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    if (result != -1)
        printf("Index is %d", result);
    else
        printf("Not Found");
    return 0;
}
```

矩阵中的最大最小值

> 【问题描述】将一个3X2的矩阵（3行2列）的矩阵存入一个3X2的二维数组中，并输出矩阵。同时，找出矩阵中的最大值以及最大值所在的行下标和列下标，输出最大值所在的行下标和列下标及最大值。
> 【输入输出样例】
> `Enter 6 integers:`
> `3 2 10 -9 6 -1`
> `3 2`
> `10 -9`
> `6 -1`
> `max=a[1][0]=10`
> 【样例说明】输入提示符中冒号为英文符号，后面无空格，需换行。输出矩阵时整数按照%4d格式输出。最后输出结束不换行

解析

- 循环嵌套的输入和判断就OK。
- 这里是建议读入和寻找最大值同时进行，或者输出与寻找最大值同时进行。

示例代码

```c
#include <stdio.h>
int main() {
    int matrix[3][2];
    int i, j, max_row = 0, max_col = 0, max_value;
    printf("Enter 6 integers:\n");
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            scanf("%d", &matrix[i][j]);
    max_value = matrix[0][0];
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 2; j++) {
            printf("%4d", matrix[i][j]);
            if (matrix[i][j] > max_value) {
                max_value = matrix[i][j];
                max_row = i;
                max_col = j;
            }
        }
        printf("\n");
    }
    printf("max=a[%d][%d]=%d", max_row, max_col, max_value);
    return 0;
}
```

上三角与下三角输出

> 【问题描述】编程实现输出矩阵上/下三角的数值。要求：输入一个正整数n和n阶矩阵的数值，打印输出矩阵、下三角和上三角的数值。
> 【样例说明】输入提示符中冒号为英文符号，后面无空格。输出矩阵时整数按照%4d格式输出。

解析

- 上三角：列数小于等于行数
- 下三角：列数大于等于行数

示例代码

```c
#include <stdio.h>
int main() {
    int n, i, j,  matrix[100][100];
    printf("Input n:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &matrix[i][j]);
    for (i = 0; i < n; i++) {//输出矩阵
        for (j = 0; j < n; j++)
            printf("%4d", matrix[i][j]);
        printf("\n");
    }
    for (i = 0; i < n; i++) {//输出上三角
        for (j = 0; j < n; j++) {
            if (j <= i)
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
    for (i = 0; i < n; i++) {//输出下三角
        for (j = 0; j < n; j++) {
            if (j >= i)
                printf("%4d", matrix[i][j]);
            else
                printf("%4c",' ');
        }
        if (i == n-1)
            break;
        printf("\n");
    }
    return 0;
}
```

简单排序

> 【问题描述】输入一个正整数n(1 <n≤10)，再输入n个整数，从大到小排序后输出。
> 【输入形式】从键盘输入一个正整数n和n个整数。
> 【输入输出样例】`Input n:5  Input 5 integers:23 12 -9 8 3  After sorted:  23  12  8  3  -9`
> 【样例说明】每个整数按照 `%4d` 格式输出，输出结束没有换行符。提示符后冒号为英文字符，无空格

解析

- 我们选择一个排序方法就OK

示例代码

```c
#include <stdio.h>
int main() {
    int n, a[11];
    printf("Input n:");
    scanf("%d", &n);
    printf("Input %d integers:", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    // 使用选择排序算法进行降序排序
    for (int i = 0; i < n - 1; i++) {
        int maxIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] > arr[maxIndex])
                maxIndex = j;
        }
        // 交换最大值与当前位置的元素
        int temp = arr[i];
        arr[i] = arr[maxIndex];
        arr[maxIndex] = temp;
    }
    printf("After sorted:");
    for (int i = 0; i < n; i++)
        printf("%4d", arr[i]);
    return 0;
}

```

统计数字个数

> 【问题描述】输入以回车符结束的字符串（少于80个字符），统计其中数字字符的个数。
> 【输入输出样例】 `Enter a string:1212shanghai345` `count=7`
> 【样例说明】提示符后冒号为英文字符，无空格；“=”等号两边无空格，输出结束无换行符。

解析

- 我们其实可以使用 `ctype.h` 库中的 `isdigit()` 函数，不用纠结具体的原理（其实就是ASCII码）
- 可以使用 `while()` 循环读入直至换行，也可以全行读入后再判断。

示例代码

```c
#include <stdio.h>
#include <ctype.h>
int main() {
    char str[81], count = 0, i = 0;
    printf("Enter a string:");
    fgets(str, sizeof(str), stdin);//读入整行数据
    while (str[i] != '\0') {
        if (isdigit(str[i]))//判断是否为数字
            count++;
        i++;
    }
    printf("count=%d", count);
    return 0;
}
```

回文判断

> 【问题描述】输入一个字符串，判断是否是回文。
> 【输入输出样例1】
> `Enter a string:ILoveevoLI`
> `It is a palindrome`
> 【输入输出样例2】
> `Enter a string:I Love Shanghai`
> `It is not a palindrome`
> 【样例说明】提示符后冒号为英文字符，无空格。输出结束无换行符。

解析

- **初始化：** 定义两个变量，一个从字符串的开头向后移动（例如 `start`），另一个从字符串的末尾向前移动（例如 `end`）。这些变量的初始值分别是 `0` 和字符串长度减 `1`。
- **比较字符：** 比较 `start` 和 `end` 处的字符是否相等。
- **移动索引：** 如果比较相等，将 `start` 增加 `1`，将 `end` 减少 `1`，然后继续比较下一对字符。
- **继续比较：** 重复步骤 `2` 和步骤 `3`，直到 `start` 大于等于 `end`。
- **返回结果：** 如果 `start` 大于等于 `end`，说明整个字符串是回文，返回 "是回文" 结果；否则，返回 "不是回文" 结果。

示例代码

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

int main() {
    char input[100];
    printf("Enter a string:");
    gets(input);//不使用printf("%s",input)的原因是输入数据里面可能包涵空格
    int start = 0;
    int end = strlen(input) - 1;
    int isPalindrome = 1; // 假设是回文
    while (start < end) {
        if (input[start] != input[end]) {
            isPalindrome = 0; // 不是回文
            break;
        }
        start++;
        end--;
    }

    if (isPalindrome)
        printf("It is a palindrome");
    else
        printf("It is not a palindrome");
    return 0;
}

```

依次减少输出

【问题描述】输入一个字符串，按以下格式输出。
【输入输出样例】

```
 Enter a string:Shanghai
1:Shanghai
2: hanghai
3:  anghai
4:   nghai
5:    ghai
6:     hai
7:      ai
8:       i
```

【样例说明】提示符后冒号为英文字符，无空格。输出结束有换行符。

解析

- 每一次输出把字符串完整输出就好，输出完后把第i个数据用空格占位

示例代码

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char str[100];
    printf("Enter a string:");
    gets(str);
    int len = strlen(str);
    for (int i = 0; i < len; i++){
        printf("%d:%s\n",i+1,str);
        str[i]=' ';//将第i个数据替换为空格
    }
    return 0;
}
```

定点插入

> 【问题描述】输入一个字符串，在指定下标位置处插入字符。
> 【输入输出样例】
> `Enter a string:lov`
> `3`
> `e`
> `love`
> 【样例说明】提示符后冒号为英文字符，无空格。输出字符串结果使用puts函数。

解析

- 在`a[n]`处插入一个数据，`a[n]`以前的数据不动，将`a[n]`后面面数据和它本身向后移动一位就行
    - `a[n+1] = a[n]`

示例代码

```c
#include <stdio.h>
#include <string.h>
int main() {
    char ch, str[100];
    int index;
    printf("Enter a string:");
    fgets(str, sizeof(str), stdin);
    scanf("%d", &index);
    scanf(" %c", &ch);
    int length = strlen(str);
    // 移动字符以腾出空间来插入新字符
    for (int i = length; i >= index; i--)
        str[i + 1] = str[i];
    // 插入新字符
    str[index] = ch;
    puts(str);
    return 0;
}
```

删除指定的字符

> 【问题描述】输入一个字符串，删除指定的字符。
> 【输入输出样例】
> `Enter a string:I love ShangHai`
> `e`
> `I lov ShangHai`
> 【样例说明】提示符后冒号为英文字符，无空格。输出结束有换行符。

解析

- 输入要删除的字符后，遍历原有字符串，存在就是删除，不存在就不做处理
- 删除的逻辑：
    - 将找到的字符位置赋值为数组后一个的值，依次类推
    - 在字符之前的数据不做处理

示例代码

```c
#include <stdio.h>
#include <string.h>
int main() {
    char ch, str[80];
    printf("Enter a string:");
    fgets(str, sizeof(str), stdin);
    scanf(" %c", &ch);
    int length = strlen(str);
    int i, j;
    for (i = j = 0; i < length; i++) {
        if (str[i] != ch)
            str[j++] = str[i];
    }
    str[j] = '\0';
    printf("%s\n", str);
    return 0;
}

```