NCUSCC 2025秋季考核C语言试题实验报告

目录

1 实验环境搭建

关于造成系统损伤及后续处理的报告

事件经过：

·在联想应用商店下载VMwareWorkstationPlayer 17安装包并安装，选择安装路径至D盘。在安装过程中，曾**错误删除或移动过该软件在C盘创建的部分文件夹**，但软件本身仍可以正常使用。
·试图卸载VMware以使用wsl，发现该软件的**卸载按钮为灰色不可选状态**。按照deepseek的建议，下载Geek Uninstaller对其进行**强制删除**（删除前创建了系统还原点1）。
·删除后在powershell使用wsl --install命令成功，**无法启用“适用windows的Linux子系统”和“Vitual Mechine Platform”功能**，在显示更新进行到100%时回滚；再次尝试从多个不同途径下载并安装各种不同版本的VMware，均**无法正常运行安装指引程序**。
·在powershell运行**sfc /scannow**命令，显示“已修复损坏的文件”。并成功运行**DISM /Online /Cleanup-**

Image /RestoreHealth命令。再次尝试启用上述功能，仍然回滚。
·在“**设备管理器**”找到“VMware Virtual Ethemet Adapter VMnet1”，该网络适配器的图标带有警告标识。卸载该设备。再次尝试启用上述功能，仍然回滚。
·创建系统还原点2，避免因错误删除重要文件导致死机。**在“此电脑”搜索VMware相关文件**，在Program Files文件夹、Users文件夹中找到VMware文件夹（内含大量.sys，.inf，.dll，.cat，.int文件）并删除。在Program Filesx86文件夹中找到VMware安装包并删除。再次尝试启用上述功能失败，仍然回滚。
·使用Autoruns软件搜索VMware有关条目，仅**在service中找到三项并禁用**。再次尝试启用上述功能失败。
·再次运行sfc /scannow命令，显示“windows资源保护找不到任何完整性冲突”。再次尝试启用上述功能，仍然回滚。试图启用系统还原点1，但发现系统在创建还原点2时**自动删除了还原点1**以释放内存。
·在“事件查看器-windows日志-应用程序”查找到来自AppModel-State的警告事件：对程序包Microsoft.MicrosoftPCManager_8wekyb3d8bbwe进行的操作遇到错误：状态位置修复错误，命中错误，修复错误等。 查找到来自winlogon的警告事件：关键通知事件失败。**系统已经出现异常**：网络经常显示“脱机”，键盘的截屏按钮毫无反应。使用“重置此电脑-保留我的文件”选项。
·因为“适用widows的Linux子系统”功能仍然在功能列表中，也为了避免可能的再次伤害，我选择在重置后的系统中使用wsl。然而运行wsl --status命令显示“wsl安装似乎已损坏”；按任意键进行修复后，又无法启用上述功能，因为“一个或多个要求的事务成员不存在”。我判断**难以在短时间内解决此问题**。
·至此已经消耗了过多时间，我选择放弃使用虚拟机。

分析与感想

1.事故原因分析：

- Geek Uninstaller的“强力删除”功能*直接、暴力地结束该软件相关的所有进程，删除安装目录并简单地在注册表中扫描并删除**所有**包含该软件名称或出版商信息的键值*。后续反复尝试安装软件和启用功能的过程可能加剧了系统损坏的扩散。
- 我由此了解了软件安装和卸载的一般流程细节，软件卸载一般会残留哪些文件及服务，怎样清理卸载残留，以及便携化软件和传统软件的区别（以解释为什么装在D盘的bilbil在重置系统后尽管不能在应用列表找到仍可正常使用）。

2.在清理残留过程中了解到：

- 注册表、系统文件、用户文件与配置的层级关系；
- 系统文件的主要分类；
- 打开软件时，注册表和系统文件协同工作的一般过程。

3.在重置系统过程中了解到：

- 注册表的复杂依赖关系，以及系统即将崩溃的细微表现（如快捷键无反应、网络显示脱机等）；

- 了解何时可以尝试使用DISM命令和sfc /scannow命令，以及其上下游关系和局限性（只能修复系统文件，对注册表混乱无能为力）。

4.在检查系统状态过程中了解到：

- 如何查找设备，启用和关闭相关服务；
- 如何查看警告和错误事件以推断问题根源；
- 了解可以使用“干净启动”（禁用所有非微软服务）判断是系统问题还是软件冲突。

2.实现排序算法并生成测试数据

实验目的

分析题目，本实验目的是了解几个点：

1. 不同pivot选择对性能的影响。
2. 不同等级的 gcc 编译优化选项对性能的影响。

具体编写

由于时间紧迫，我选择仅实现快速排序。

1. 首先实现快速排序。

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 交换位置
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

- pivot选择只需改动partition函数。

选择最后一个元素

```
// 分区进行
int partition(int arr[], int low, int high) {
```

三数取中

```
int partition_median(int arr[], int low, int high) {
    int mid = low + (high - low) / 2;

    // 确保arr[low] <= arr[mid] <= arr[high]
    if (arr[low] > arr[mid]) swap(&arr[low], &arr[mid]);
    if (arr[low] > arr[high]) swap(&arr[low], &arr[high]);
    if (arr[mid] > arr[high]) swap(&arr[mid], &arr[high]);

    swap(&arr[mid], &arr[high]);
```

后面一模一样：

```
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

- 递归和非递归版本只需改动quickSort函数部分。

递归版本

```
// 快速排序主函数
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

非递归版本

```
// 非递归快速排序（使用栈模拟递归）
void quickSort_iterative(int arr[], int low, int high) {
    // 创建栈来存储边界
    int stack[high - low + 1];
    int top = -1;

    // 初始边界入栈
    stack[++top] = low;
    stack[++top] = high;

    while (top >= 0) {
        // 出栈获取边界
        high = stack[top--];
        low = stack[top--];

        // 分区操作（可以用原来的partition函数）
        int pi = partition(arr, low, high);

        // 如果左边有元素，压入左边界
        if (pi - 1 > low) {
            stack[++top] = low;
            stack[++top] = pi - 1;
        }

        // 如果右边有元素，压入右边界
        if (pi + 1 < high) {
            stack[++top] = pi + 1;
            stack[++top] = high;
        }
    }
}
```

将后面main函数中的quickSort函数名改成quickSort_iterative即可。
2. 接下来生成测试数据并保存到一个文件内。

```c
void generate_test_data(int size, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("错误：无法创建文件 %s\n", filename);
        return;
    }

    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        fprintf(file, "%d\n", rand() % 10000);
    }

    fclose(file);
    printf("已生成 %d 条测试数据到 %s\n", size, filename);
}
```

3. 从该文件中读取数据，并放到一个数组里。

```c
int read_data_from_file(const char* filename, int arr[], int max_size) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("错误：无法打开文件 %s\n", filename);
        return 0;
    }

    int count = 0;
    while (count < max_size && fscanf(file, "%d", &arr[count]) == 1) {
        count++;
    }

    fclose(file);
    return count;
}
```

4. 检查数组是否已按顺序排好

```c
int is_sorted(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            return 0; // 未排序
        }
    }
    return 1; // 已排序
}

// 打印数组前20个元素（用于预览）
void print_array_preview(int arr[], int n) {
    printf("前20个元素: ");
    int limit = n < 20 ? n : 20;
    for (int i = 0; i < limit; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

5. 运行主函数。

```c
int main() {
    printf("=== 快速排序测试程序 ===\n");
}
// 配置测试参数
    const char* filename = "test_data.txt";
    int data_size = 100000;
    int* data = (int*)malloc(data_size * sizeof(int));

    if (data == NULL) {
        printf("错误：内存分配失败！\n");
        return 1;
    }

    // 生成测试数据
    printf("\n生成测试数据...\n");
    generate_test_data(data_size, filename);

    // 读取数据
    printf("读取数据...\n");
    int actual_size = read_data_from_file(filename, data, data_size);
    printf("   成功读取 %d 条数据\n", actual_size);
    print_array_preview(data, actual_size);

    // 执行排序并计时
    printf("开始快速排序...\n");
    clock_t start_time = clock();
    quickSort(data, 0, actual_size - 1);
    clock_t end_time = clock();

    // 计算用时
    double time_used = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

    // 验证结果
    printf("验证排序结果...\n");
    if (is_sorted(data, actual_size)) {
        printf("排序正确\n");
    } else {
        printf("排序错误\n");
    }

    printf("性能统计：\n");
    printf("   - 数据量：%d 条\n", actual_size);
    printf("   - 排序用时：%.3f 秒\n", time_used);
```

```
    print_array_preview(data, actual_size);

    free(data);
    printf("\n=== 程序执行完毕 ===\n");
    return 0;
}
```

由于时间没有安排好以及环境问题，无法完成之后的要求。非常抱歉。感谢您看到这里。

3.参考资料与特别鸣谢

- 参考资料
  Vmware下载安装教程
  VMware卸载清理工具推荐
  在 Windows 10 和 11 上安全清理注册表的 3 种方法
  错误代码0x8007371b
  【初探数据结构】详解三大经典排序算法（选择/堆/冒泡）
- 特别鸣谢
  Deepseek1.3.1(98)
  bilbil
  愿意看到这里的你