【华为OD机考 统一考试机试C卷】手机App防沉迷系统（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

智能手机方便了我们生活的同时，也侵占了我们不少的时间。“手机App防沉迷系统”能够让我们每天合理地规划手机App使用时间，在正确的时间做正确的事。

它的大概原理是这样的：

1. 在一天24小时内，可以注册每个App的允许使用时段
2. 一个时间段只能使用一个App
3. App有优先级，数值越高，优先级越高。注册使用时段时，如果高优先级的App时间和低优先级的时段有冲突，则系统会自动注销低优先级的时段，如果App的优先级相同，则后添加的App不能注册。

请编程实现，根据输入数据注册App，并根据输入的时间点，返回时间点使用的App名称，如果该时间点没有注册任何App，请返回字符串“NA”。

输入描述

第一行表示注册的App数量 N（N ≤ 100）

第二部分包括 N 行，每行表示一条App注册数据

最后一行输入一个时间点，程序即返回该时间点使用的App

> 2
> App1 1 09:00 10:00
> App2 2 11:00 11:30
> 09:30

数据说明如下：

1. N行注册数据以空格分隔，四项数依次表示：App名称、优先级、起始时间、结束时间
2. 优先级1~5，数字越大，优先级越高
3. 时间格式 HH:MM，小时和分钟都是两位，不足两位前面补0
4. 起始时间需小于结束时间，否则注册不上
5. 注册信息中的时间段包含起始时间点，不包含结束时间点

输出描述

输出一个字符串，表示App名称，或NA表示空闲时间

用例2

输入

```
1
App1 1 09:00 10:00
09:30
```

输出

```
App1
```

说明

> App1注册在9点到10点间，9点半可用的应用名是App1

用例2

输入

```
2
App1 1 09:00 10:00
App2 2 09:10 09:30
09:20
```

输出

```
App2
```

说明

> ApP1和App2的时段有冲突，App2优先级高，注册App2之后，App1自动注销，因此输出App2。

用例3

输入

```
2
App1 1 09:00 10:00
App2 2 09:10 09:30
09:50
```

输出

```
NA
```

解题思路

1. **定义App类**：首先，我们定义了一个App类来存储每个App的相关信息，包括名称、优先级、起始时间和结束时间。
2. **时间转换**： 将时间从"小时:分钟"格式的字符串转换为以分钟为单位的整数。
3. **处理注册的App**：创建另一个ArrayList来存储注册的App。对于每个App，我们检查它的时间段是否与已注册App的时间段重叠。如果有重叠，我们比较它们的优先级。如果当前App的优先级高于已注册的App，则注销低优先级的App。否则，跳过当前App。这样确保了在任何给定时间，只有最高优先级的App被注册。
4. **查询特定时间的App**：读取时间，然后遍历注册的App列表，找到在该时间活跃的App（即查询时间在App的起始时间和结束时间之间）。将此App的名称存储在变量 `appAtTime` 中。

总结：核心思想是首先读取App信息并转换时间格式，然后在注册阶段处理时间冲突和优先级问题，最后根据查询时间确定哪个App是活跃的，并输出该App的名称。

C++

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>

// 定义App类，用于存储App的相关信息
class App {
public:
    std::string name; // App名称
    int priority; // App优先级
    int startTime; // App允许使用的起始时间（以分钟为单位）
    int endTime; // App允许使用的结束时间（以分钟为单位）

    // App类的构造函数，用于创建App对象
    App(std::string name, int priority, int startTime, int endTime)
        : name(name), priority(priority), startTime(startTime), endTime(endTime) {}
};

// 时间转换函数，将时间字符串转换为以分钟为单位的整数
int convertTime(const std::string& time) {
    int hours, minutes;
    char colon;
```

```
    std::istringstream iss(time);
    iss >> hours >> colon >> minutes; // 将时间字符串按照":"分割并转换为小时和分钟
    return hours * 60 + minutes; // 将小时和分钟转换为分钟
}

int main() {
    int n; // 读取App数量
    std::cin >> n;

    std::vector<App> apps; // 创建App列表，用于存储所有App
    for (int i = 0; i < n; i++) {
        // 循环读取每个App的信息，并创建App对象添加到列表中
        std::string appName;
        int appPriority, appStartTime, appEndTime;
        std::string startTimeStr, endTimeStr;
        std::cin >> appName >> appPriority >> startTimeStr >> endTimeStr;
        appStartTime = convertTime(startTimeStr);
        appEndTime = convertTime(endTimeStr);
        apps.emplace_back(appName, appPriority, appStartTime, appEndTime);
    }

    std::string queryTimeStr;
    std::cin >> queryTimeStr;
    int queryTime = convertTime(queryTimeStr); // 读取查询时间，并转换为分钟
    std::string appAtTime = "NA"; // 初始化查询时间对应的App名称为"NA"

    std::vector<App> registeredApps; // 创建已注册App列表
    for (const App& app : apps) {
        if (app.startTime >= app.endTime) continue; // 如果起始时间不小于结束时间，则跳过

        // 遍历已注册的App列表，检查时间冲突
        for (int i = registeredApps.size() - 1; i >= 0; --i) {
            const App& registered = registeredApps[i];
            // 如果存在时间冲突
            if (std::max(app.startTime, registered.startTime) < std::min(app.endTime, registered.endTime)) {
                // 如果当前App的优先级高于已注册App的优先级
                if (app.priority > registered.priority) {
                    registeredApps.erase(registeredApps.begin() + i); // 注销低优先级的App
                } else {
                    continue; // 如果优先级不高，继续检查下一个已注册App
```

```
                }
            }
        }

        // 将当前App添加到已注册App列表中
        registeredApps.push_back(app);
    }

    // 遍历已注册App列表，找到查询时间对应的App
    for (const App& app : registeredApps) {
        if (queryTime >= app.startTime && queryTime < app.endTime) {
            appAtTime = app.name; // 更新查询时间对应的App名称
            break; // 找到后退出循环
        }
    }

    std::cout << appAtTime << std::endl; // 输出查询时间对应的App名称

    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    // 定义App类，用于存储App的相关信息
    static class App {
        String name; // App名称
        int priority; // App优先级
        int startTime; // App允许使用的起始时间（以分钟为单位）
        int endTime; // App允许使用的结束时间（以分钟为单位）

        // App类的构造函数，用于创建App对象
        public App(String name, int priority, int startTime, int endTime) {
            this.name = name;
            this.priority = priority;
            this.startTime = startTime;
            this.endTime = endTime;
```

```
        }
    }

    // 主函数
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); // 创建Scanner对象，用于读取标准输入
        int n = sc.nextInt(); // 读取App数量

        ArrayList<App> apps = new ArrayList<>(); // 创建App列表，用于存储所有App
        for (int i = 0; i < n; i++) {
            // 循环读取每个App的信息，并创建App对象添加到列表中
            String appName = sc.next();
            int appPriority = sc.nextInt();
            int appStartTime = convertTime(sc.next());
            int appEndTime = convertTime(sc.next());
            apps.add(new App(appName, appPriority, appStartTime, appEndTime));
        }

        int queryTime = convertTime(sc.next()); // 读取查询时间，并转换为分钟
        String appAtTime = "NA"; // 初始化查询时间对应的App名称为"NA"

        // 创建已注册App列表
        ArrayList<App> registeredApps = new ArrayList<>();
        for (App app : apps) {
            if (app.startTime >= app.endTime) continue; // 如果起始时间不小于结束时间，则跳过

            // 遍历已注册的App列表，检查时间冲突
            for (int i = registeredApps.size() - 1; i >= 0; i--) {
                App registered = registeredApps.get(i);
                // 如果存在时间冲突
                if (Math.max(app.startTime, registered.startTime) < Math.min(app.endTime, registered.endTime)) {
                    // 如果当前App的优先级高于已注册App的优先级
                    if (app.priority > registered.priority) {
                        registeredApps.remove(i); // 注销低优先级的App
                    } else {
                        continue; // 如果优先级不高，继续检查下一个已注册App
                    }
                }
            }


```

```
            // 将当前App添加到已注册App列表中
            registeredApps.add(app);
        }

        // 遍历已注册App列表，找到查询时间对应的App
        for (App app : registeredApps) {
            if (queryTime >= app.startTime && queryTime < app.endTime) {
                appAtTime = app.name; // 更新查询时间对应的App名称
                break; // 找到后退出循环
            }
        }

        System.out.println(appAtTime); // 输出查询时间对应的App名称
    }

    // 时间转换函数，将时间字符串转换为以分钟为单位的整数
    private static int convertTime(String time) {
        String[] parts = time.split(":"); // 将时间字符串按照":"分割
        return Integer.parseInt(parts[0]) * 60 + Integer.parseInt(parts[1]); // 将小时和分钟转换为分钟
    }
}
```

javaScript

```
const readline = require('readline');

// 定义App类，用于存储App的相关信息
class App {
  constructor(name, priority, startTime, endTime) {
    this.name = name; // App名称
    this.priority = priority; // App优先级
    this.startTime = startTime; // App允许使用的起始时间（以分钟为单位）
    this.endTime = endTime; // App允许使用的结束时间（以分钟为单位）
  }
}

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
```

```
});

// 用于存储输入行的数组
const lines = [];
// 读取输入
rl.on('line', (line) => {
  lines.push(line);
}).on('close', () => {
  // 当输入完成后开始处理数据
  processInput(lines);
});

// 处理输入数据的函数
function processInput(lines) {
  const n = parseInt(lines.shift()); // 读取App数量
  const apps = []; // 创建App列表，用于存储所有App

  for (let i = 0; i < n; i++) {
    // 循环读取每个App的信息，并创建App对象添加到列表中
    const [appName, appPriority, appStartTime, appEndTime] = lines.shift().split(' ');
    apps.push(new App(appName, parseInt(appPriority), convertTime(appStartTime), convertTime(appEndTime)));
  }

  const queryTime = convertTime(lines.shift()); // 读取查询时间，并转换为分钟
  let appAtTime = "NA"; // 初始化查询时间对应的App名称为"NA"

  // 创建已注册App列表
  const registeredApps = [];
  for (const app of apps) {
    if (app.startTime >= app.endTime) continue; // 如果起始时间不小于结束时间，则跳过

    // 遍历已注册的App列表，检查时间冲突
    for (let i = registeredApps.length - 1; i >= 0; i--) {
      const registered = registeredApps[i];
      // 如果存在时间冲突
      if (Math.max(app.startTime, registered.startTime) < Math.min(app.endTime, registered.endTime)) {
        // 如果当前App的优先级高于已注册App的优先级
        if (app.priority > registered.priority) {
          registeredApps.splice(i, 1); // 注销低优先级的App
        } else {
```

```
            continue; // 如果优先级不高，继续检查下一个已注册App
          }
        }
      }

      // 将当前App添加到已注册App列表中
      registeredApps.push(app);
    }

    // 遍历已注册App列表，找到查询时间对应的App
    for (const app of registeredApps) {
      if (queryTime >= app.startTime && queryTime < app.endTime) {
        appAtTime = app.name; // 更新查询时间对应的App名称
        break; // 找到后退出循环
      }
    }

    console.log(appAtTime); // 输出查询时间对应的App名称
}

// 时间转换函数，将时间字符串转换为以分钟为单位的整数
function convertTime(time) {
    const [hours, minutes] = time.split(':').map(Number); // 将时间字符串按照":"分割并转换为数字
    return hours * 60 + minutes; // 将小时和分钟转换为分钟
}
```

Python

```
class App:
    """定义App类，用于存储App的相关信息"""

    def __init__(self, name, priority, start_time, end_time):
        self.name = name  # App名称
        self.priority = priority  # App优先级
        self.start_time = start_time  # App允许使用的起始时间（以分钟为单位）
        self.end_time = end_time  # App允许使用的结束时间（以分钟为单位）

def convert_time(time_str):
    """
    时间转换函数，将时间字符串转换为以分钟为单位的整数
```

```
    :param time_str: 时间字符串，格式为"小时:分钟"
    :return: 转换后的分钟数
    """
    hours, minutes = map(int, time_str.split(":"))  # 将时间字符串按照":"分割并转换为整数
    return hours * 60 + minutes  # 将小时和分钟转换为分钟

def main():
    n = int(input())  # 读取App数量
    apps = []  # 创建App列表，用于存储所有App

    for _ in range(n):
        # 循环读取每个App的信息，并创建App对象添加到列表中
        app_name, app_priority, app_start_time, app_end_time = input().split()
        app_priority = int(app_priority)
        app_start_time = convert_time(app_start_time)
        app_end_time = convert_time(app_end_time)
        apps.append(App(app_name, app_priority, app_start_time, app_end_time))

    query_time = convert_time(input())  # 读取查询时间，并转换为分钟
    app_at_time = "NA"  # 初始化查询时间对应的App名称为"NA"

    # 创建已注册App列表
    registered_apps = []
    for app in apps:
        if app.start_time >= app.end_time:
            continue  # 如果起始时间不小于结束时间，则跳过

        # 遍历已注册的App列表，检查时间冲突
        for i in range(len(registered_apps) - 1, -1, -1):
            registered = registered_apps[i]
            # 如果存在时间冲突
            if max(app.start_time, registered.start_time) < min(app.end_time, registered.end_time):
                # 如果当前App的优先级高于已注册App的优先级
                if app.priority > registered.priority:
                    registered_apps.pop(i)  # 注销低优先级的App
                else:
                    continue  # 如果优先级不高，继续检查下一个已注册App

        # 将当前App添加到已注册App列表中
        registered_apps.append(app)
```

```
    # 遍历已注册App列表，找到查询时间对应的App
    for app in registered_apps:
        if query_time >= app.start_time and query_time < app.end_time:
            app_at_time = app.name  # 更新查询时间对应的App名称
            break  # 找到后退出循环

    print(app_at_time)  # 输出查询时间对应的App名称

if __name__ == "__main__":
    main()
```

C语言

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

// 定义App结构体，用于存储App的相关信息
typedef struct {
    char name[30]; // App名称
    int priority;  // App优先级
    int startTime; // App允许使用的起始时间（以分钟为单位）
    int endTime;   // App允许使用的结束时间（以分钟为单位）
} App;

// 时间转换函数，将时间字符串转换为以分钟为单位的整数
int convertTime(char *time) {
    int hours, minutes;
    sscanf(time, "%d:%d", &hours, &minutes); // 将时间字符串按照":"分割并转换为整数
    return hours * 60 + minutes; // 将小时和分钟转换为分钟
}

int main() {
    int n; // 读取App数量
    scanf("%d", &n);

    App *apps = (App *)malloc(n * sizeof(App)); // 创建App数组，用于存储所有App

    for (int i = 0; i < n; i++) {
```

```
        // 循环读取每个App的信息，并创建App对象添加到数组中
        scanf("%s %d", apps[i].name, &apps[i].priority);
        char startTimeStr[6], endTimeStr[6];
        scanf("%s %s", startTimeStr, endTimeStr);
        apps[i].startTime = convertTime(startTimeStr);
        apps[i].endTime = convertTime(endTimeStr);
    }

    char queryTimeStr[6];
    scanf("%s", queryTimeStr);
    int queryTime = convertTime(queryTimeStr); // 读取查询时间，并转换为分钟
    char *appAtTime = "NA"; // 初始化查询时间对应的App名称为"NA"

    // 创建已注册App数组和计数器
    App *registeredApps = (App *)malloc(n * sizeof(App));
    int registeredCount = 0;

    for (int i = 0; i < n; i++) {
        if (apps[i].startTime >= apps[i].endTime) continue; // 如果起始时间不小于结束时间，则跳过

        // 遍历已注册的App数组，检查时间冲突
        for (int j = registeredCount - 1; j >= 0; j--) {
            // 如果存在时间冲突
            if (apps[i].startTime < registeredApps[j].endTime && apps[i].endTime > registeredApps[j].startTime) {
                // 如果当前App的优先级高于已注册App的优先级
                if (apps[i].priority > registeredApps[j].priority) {
                    // 注销低优先级的App
                    for (int k = j; k < registeredCount - 1; k++) {
                        registeredApps[k] = registeredApps[k + 1];
                    }
                    registeredCount--; // 减少已注册App的计数
                } else {
                    goto continue_outer; // 如果优先级不高，继续检查下一个已注册App
                }
            }
        }

        // 将当前App添加到已注册App数组中
        registeredApps[registeredCount++] = apps[i];

```

```
        continue_outer: ; // 标签用于跳过当前循环
    }

    // 遍历已注册App数组，找到查询时间对应的App
    for (int i = 0; i < registeredCount; i++) {
        if (queryTime >= registeredApps[i].startTime && queryTime < registeredApps[i].endTime) {
            appAtTime = registeredApps[i].name; // 更新查询时间对应的App名称
            break; // 找到后退出循环
        }
    }

    printf("%s\n", appAtTime); // 输出查询时间对应的App名称

    // 释放动态分配的内存
    free(apps);
    free(registeredApps);

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师