Lab 12

本次内容：

接收命令行参数的C程序

作业：
1. 作息时间
2. You
3. 字符串匹配
4. 翻转数组
5. 最长字符串
6. 使用指针的栈
7. 解析URL字符串

提交截止时间：12/8 23:59

命令行参数

还记得我们在一次作业 上帝掷骰子之前 中根据输入的两个物理量 `nu` 和 `t` 计算了黑体的辐射率。当时的要求是使用 `scanf` 读取输入，但是这种方式不够灵活，不方便批处理，也不方便和其它程序集成。

记得 `ls` 和 `cd` 这些命令并不是 `scanf` 那种运行然后等待输入的交互逻辑。在运行 `ls` 和 `cd` 的同时我们就用命令行直接传入了参数。这里我们介绍C语言程序接收命令行参数的方法:

```c
int main(int argc, char* argv[]) {}
```

与我们之前见过的 `main` 函数不同，上面这个 `main` 函数接收两个参数输入。其中：

- `argc` 表示命令行参数的数量，包括程序名称本身。 `argc` 至少为1，因为程序名称本身也算作一个参数。
- `argv` 参数是一个指向指针的数组，每个指针指向一个以空格分隔的命令行参数字符串。 `argv[0]` 是程序名称本身的字符串。

下面是使用命令行参数改写的计算普朗克黑体辐射率的程序，其中 `atof` 是将字符串转换成浮点数的函数：

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

double I(double nu, double t) {
    double term1 = nu * nu * nu;
    double term2 = exp(nu / t) - 1;
    return term1 / term2;
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        fprintf(stderr, "usage: %s <nu> <t>", argv[0]);
        exit(1);
    }

    double nu = atof(argv[1]);
    double t = atof(argv[2]);
    printf("%f\n", I(nu, t));
}
```

现在可以直接这样使用程序

```
gcc planck.c -o planck -lm
./planck 1.1 0.5
0.165856
```

而且可以方便地批处理，这次lab压缩包里面有两个脚本 `planck.ps1` 和 `planck.sh` ， 分别是 `Windows` 平台和其它平台的批处理脚本文件，运行可以很方便地输出不同 `nu` 下的辐射率：

`Windows`：

```
./planck.ps1
```

`Linux` ， `Mac`：

```
bash planck.sh
```

作业

1. 作息时间

在大家都还是高中生的时候，想必作息是非常规律的。上了大学后，很多同学都喜欢熬夜，这毕竟不是好的习惯。现在用这道判断睡觉时间的题自勉：

题目输入格式： `hh::mm::ss` ，表示一个合法的24小时制的时刻， `hh` ， `mm` ， `ss` 都为整数，多组输入，每组输入一行。当输入 `EOF` 时退出程序（阅读材料， `Windows` 下 `Ctrl+Z` ， `Linux` 和 `Mac` 下 `Ctrl+D` 表示 `EOF` ）

程序输出：对于每组输入，如果该时刻在 `23:30:00` 到次日 `07:30:00` 之间（包含这两个时刻），则输出字符串 `sleep` ，否则输出字符串 `wake up` 。

2. You

输入 n 和 n 个空格分割的字符串，输出其中单词 You 的个数（不分大小写）

例如：

```
4
zee You yOU thee
cnt: 2
```

3. 字符串匹配

期中考试最后一道题是字符串匹配，这次我们把这道题再出成作业：

实现函数

```
int string_match(const char* str, const char* p)
```

函数返回字符串 p 首次在字符串 str 出现的位置
例如，当 str 为"aaa example"，p 为"example"时，函数输出4，表示 str 从下标4开始与字符 p 匹配。如果不能匹配则返回-1

参考测试

```
if (string_match("abcd", "") != 0) {
    fputs("test 1 failed", stderr);
    exit(1);
}

if (string_match("abcd", "bcde") != -1) {
    fputs("test 2 failed", stderr);
    exit(1);
}

if (string_match("here is an simple example", "example") != 18) {
    fputs("test 3 failed", stderr);
    exit(1);
}

if (string_match("here is an simple exampl", "example") != -1) {
    fputs("test 4 failed", stderr);
    exit(1);
}
puts("success");
```

4. 翻转数组

请实现翻转数组的函数:

```
void reverse(int* array, int length)
```

参考测试：

```c
int arr[] = {1, 2, 3, 4, 5};
int length = sizeof(arr) / sizeof(arr[0]);

reverseArray(arr, length);
for (int i = 0; i < length; i++) {
    if (arr[i] != length - i) {
        fputs("error", stderr);
        exit(1);
    }
}
printf("success\n");
```

5. 最长字符串

设计一个函数 `longest_str` ，它接受多个字符串作为参数，并返回其中最长的字符串。

```c
const char* longest_str(const char* strs[], int cnt)
```

参考测试：

```c
const char* strings[] = {"Hello", "OpenAI", "Assistant", "GPT-3.5"};
int count = sizeof(strings) / sizeof(strings[0]);

const char* longest = longest_str(strings, count);
printf("longest: %s\n", longest);
```

6. Stack Once More

我们之前实现过栈这种数据结构，现在请采用指针再实现一次栈，这次的要求会更加明确：

- 使用指针表示栈顶位置
- 实现 `void push(int value)`
- 实现 `void pop()` ，注意这里 `pop()` 的返回类型是 `void`
- 实现 `int is_empty()`
- 实现 `int is_full()`
- 实现 `void reset()` 清空栈

```c
const int MAX_SIZE = 100;
int stack[MAX_SIZE];
int* top = NULL;
```

下面给出一个测试用例作为参考：

```
reset();
if (!is_empty() || top != NULL) {
    fputs("stack not empty", stderr);
    exit(1);
}

for (int i = 0; i < MAX_SIZE; i += 1) {
    push(i);
    if (top == NULL || *top != i) {
        fputs("unexpected value", stderr);
        exit(1);
    }
}

if (!is_full()) {
    fputs("stack not full", stderr);
    exit(1);
}

for (int i = 0; i < MAX_SIZE; i += 1) {
    if (top == NULL || *top != MAX_SIZE - i - 1) {
        fputs("unexpected value", stderr);
        exit(1);
    }
    pop();
}

fputs("success", stdout);
```

7. 解析URL字符串

URL(Uniform Resource Locator)也就是我们平时网上冲浪使用的网址，以必应搜索为例：

```
https://cn.bing.com
```

URL一般包含：

- 协议类型：例如 `https`
- 域名：`cn.bing.com` ， 用 `://` 与协议类型分开

有些时候会包含端口号形成类似这样的URL：`https://cn.bing.com:443` 。端口号放在最后，用 `:` 分割。端口号一般会省略，如果协议是 `https` 则端口号默认为443，如果协议是 `http` 则端口号默认为80

在这个作业中，你需要实现解析URL字符串的功能，从输入的URL字符串分别提取协议类型、域名和端口号。

```
void parse_url(const char* url, char* protocol, char* domain, *port)
```

以下为参考测试示例：

```
char protocol[20];
char domain[50];
char port[10];

parse_url("https://cn.bing.com", protocol, domain, port);
if (strcmp(protocol, "https") != 0
    || strcmp(domain, "cn.bing.com") != 0
    || strcmp(port, "443") != 0
) {
    fputs("test case 1 parse error", stderr);
    exit(1);
}

parse_url("http://cn.bing.com", protocol, domain, port);
if (strcmp(protocol, "http") != 0
    || strcmp(domain, "cn.bing.com") != 0
    || strcmp(port, "80") != 0
) {
    fputs("test case 2 parse error", stderr);
    exit(1);
}

parse_url("http://cn.bing.com:233", protocol, domain, port);
if (strcmp(protocol, "http") != 0
    || strcmp(domain, "cn.bing.com") != 0
    || strcmp(port, "233") != 0
) {
    fputs("test case 3 parse error", stderr);
    exit(1);
}
puts("success");
```