CS100 Recitation 5

GKxx

目录

- Homework 2 讲评
- C 收尾

Homework 2

1. 区间和查询

给一个序列 $a_0, \cdots, a_{n-1}$ ， q 次询问，每次查询一段区间和。

1. 区间和查询

给一个序列 $a_0, \cdots, a_{n-1}$ ， q 次询问，每次查询一段区间和。

令 $s_i = \sum_{j=0}^{i-1} a_j$ ，则 $\sum_{i=l}^{r-1} a_i = s_r - s_l$ 。

s_i 可以递推得来： $s_i = s_{i-1} + a_{i-1}, s_0 = 0$ 。

```
for (int i = 0; i != n; ++i)
    s[i + 1] = s[i] + a[i];
int q;
scanf("%d", &q);
while (q--) {
  int l, r;
  scanf("%d%d", &l, &r);
  printf("%lld\n", s[r] - s[l]);
}
```

1. 区间和查询

a 和 s 这两个序列怎么存储？

- 题目说了 $n \leqslant 10^6$ ，所以开 10^6 的数组就行了。
- 巨大数组开全局，或者加 `static` ，否则会 stack overflow 。
- $|a_i| \leqslant 10^6$ ，所以求和之后可能会超过一般 `int` 的表示范围，要用 `long long` 。

为什么要用动态内存？

- 这里用动态内存唯一的好处就是稍微省一点空间： n 是多少就开多少。
- 但是 n 并不是一开始就知道的，所以你只能两倍两倍地扩容，非常麻烦。
- 如果你用动态内存却开 10^6 ，那就真的是行为艺术了。

2. 一元二次方程

输入一元方程 $ax^2 + bx + c = 0$ 的三个系数，按要求输出方程的解。

- （本应该）唯一的坑点：输出 `x\in\mathbb{R}` 时，`'\'` 需要转义
- 实际出现的坑点：如何判断两个根谁大谁小？
    - 初中老师流下了伤心的泪水

注意：题目保证了输入的 a, b, c 都是**整数**。**不要随便用浮点数代替整数**

3. 命令行参数

先复习一下命令行参数：

```
int main(int argc, char **argv) {
  printf("argc == %d\n", argc);
  for (int i = 0; i != argc; ++i)
    puts(argv[i]);
}
```

运行 `./my_program shanghai rainbow chamber singers` ，输出：

```
5
./my_program
shanghai
rainbow
chamber
singers
```

3. 命令行参数

遍历每个字符串，匹配，输出相应的内容

如何匹配？

- 完整地匹配 `-Wall`，`-Wpedantic` 等
- 匹配前缀 `-std=`
- 匹配后缀 `.cpp`，`.C`，`.hxx` 等

3. 命令行参数

遍历每个字符串，匹配，输出相应的内容

如何匹配？

- 完整地匹配 `-Wall`, `-Wpedantic` 等：直接 `strcmp`
- 匹配前缀 `-std=` ：`strncmp`
- 匹配后缀 `.cpp`, `.C`, `.hxx` 等：从后往前找 `.` ，然后 `strncmp`

```
int pos = len;
for (int j = len - 1; j >= 0; --j)
  if (argv[i][j] == '.') {
    pos = j;
    break;
  }
```

3. 命令行参数

直接地表达你的意图

- “匹配”的含义是“相等”。 `a` 是 `b` 的前缀意味着 `strcmp(a, b) <= 0` ，反之不然。
- `a` 是 `b` 的后缀意味着 `a` 在 `b` 中出现了，反之不然。

查找、匹配、遍历等操作都是**只读**的，不应该修改被操作的字符串。

这些操作也足够简单，不应该对字符串进行任何拷贝。

Homework 3 的若干要求

第二题

- 字符串读进来存好之后就不可以拷贝了。
- 输出必须由一次 `printf` 搞定。

Homework 3 的若干要求

第三题

- 前面 6 个字符处理的函数，只能由一条 `return` 语句完成，不能写其它任何内容。
- 后面 5 个字符串处理的函数，每个都只能扫字符串至多一遍。

典型的错误：

```
int len = hw3_strlen(src);
for (int i = 0; i < len; ++i)
  // ...
```

C 收尾

一些字符串问题

下面的代码有什么问题？

```
for (int i = 0; i < strlen(s); ++i) {
  do_something_with(s[i]);
}
```

一些字符串问题

下面的代码有什么问题？

```
for (int i = 0; i < strlen(s); ++i) {
  do_something_with(s[i]);
}
```

编译器报 warning ：

```
a.c:6:21: warning: comparison of integer expressions of different signedness:
‘int’ and ‘size_t’ {aka ‘long unsigned int’} [-Wsign-compare]
    6 |   for (int i = 0; i < strlen(s); ++i)
      |                     ^
```

一些字符串问题

下面的代码有什么问题？

```
for (int i = 0; i < strlen(s); ++i) {
  do_something_with(s[i]);
}
```

`strlen(s)` 返回值类型为 `size_t` ，一种（通常非常大）的无符号整数类型。

当 `int` 和 `size_t` 互相比较时， `int` **值会被隐式转换成** `size_t` **类型**，所以编译器认为这里不安全。

- 如果 `int x = -1` ，表达式 `x < strlen(s)` 几乎肯定是 false 。

一些字符串问题

下面的代码有什么问题？

```
for (size_t i = 0; i < strlen(s); ++i) {
  do_something_with(s[i]);
}
```

`strlen(s)` 会遍历整个字符串。它没有任何更神奇的办法求出 `s` 的长度。

假如 `do_something_with(s[i])` 的时间复杂度是 $O(1)$，上面的代码时间复杂度就是 $O\left(n^2\right)$，其中 n 是 `s` 的长度。

正确办法：

```
for (size_t i = 0, len = strlen(s); i < len; ++i) {
  do_something_with(s[i]);
}
```

一些字符串问题

理解下面的代码：

- `const char *s = "hello";`
- `char a[] = "hello";`
- `char *strs[] = {"hello", "world"};`
- `printf(NULL);`
- `strcmp(s1, s2);`

`enum`

用来自定义一些**字面值**

```
enum Color {
  red, green, blue
};
enum Result {
  accepted, wrong_answer, runtime_error, time_exceeded, memory_exceeded
};
```

每个 enum item 会被关联到一个 `int` ，默认情况下按照 `0`, `1`, `2`, ... 排下去。

`enum`

和 `struct` 一样， `enum` 也定义了一个特殊的类型。

```
void print_color(enum Color c) {
  switch (c) {
    case red:
      puts("red");
      break;
    case green:
      puts("green");
      break;
    case blue:
      puts("blue");
      break;
  }
}
```

`union`

`union` 和 `struct` 一样，也有一系列**成员**。不同的是， `union` 是这些成员的“叠加态”，而不是笛卡尔积。

$$\texttt{sizeof(union X)} \geqslant \max_{\texttt{member}\in\texttt{X}} \texttt{sizeof(member)}.$$

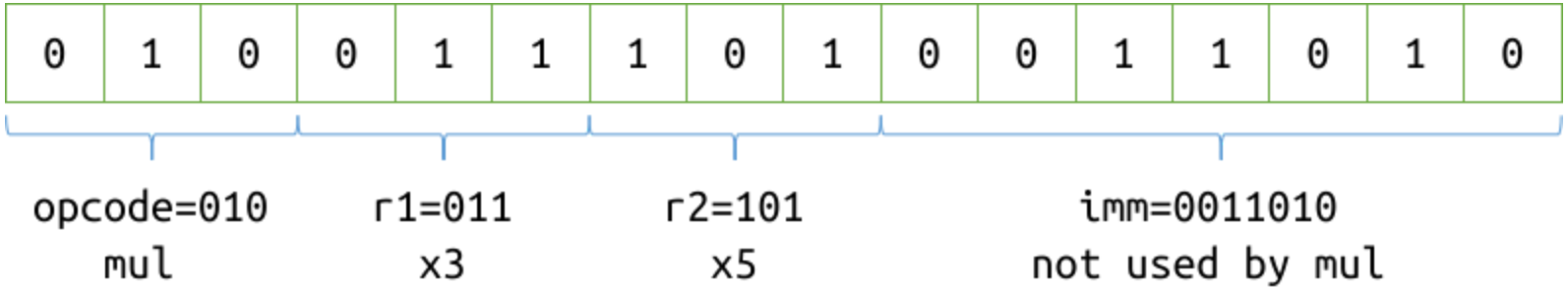
$$\texttt{sizeof(struct X)} \geqslant \sum_{\texttt{member}\in\texttt{X}} \texttt{sizeof(member)}.$$

使用 `union` 要非常小心。

bitfield

注意：以下代码的行为是 implementation-dependent 的

```
struct Instruction { // 很有可能是反过来的
  unsigned opcode : 3;
  unsigned r1 : 3;
  unsigned r2 : 3;
  unsigned imm : 7;
};
```




```
unsigned x; scanf("%x", &x);
struct Instruction i= *(struct Instruction *)&x;
// 现在可以直接使用 i.opcode, i.r1, i.r2, i.imm
```