

7-data-types 题解

CPL 助教团队

2022 年 12 月 7 日

目录

1	超级鸡尾酒 (wine.c)	2
2	画长方体 (cuboid.c)	3
3	倒水游戏 (pour.c)	7
4	内存分配器 (buddy.c)	10
5	括号序列 (brackets.c)	12
6	立方体翻转 (rotate.c)	14
7	写在最后	16

教学周历中本周课程的知识点为“基本数据类型”，对应教材章节为 7.1 - 7.6。
本周作业为期中复习题。

本周作业为期中复习题，难度较大。大家可以配合[视频讲解](#)食用本题解。

1 超级鸡尾酒 (wine.c)

本题知识点：整数输入、结构体数组的初始化与访问、排序、整数运算、循环、条件分支。

因为液体的容积可以切割成任意等份，所以一个很自然的想法就是我们不断地往这个容积为 L 的酒杯里面倒入当前价值最高的液体，直到倒入了所有的酒或者是将这个杯子倒满。

为了实现这个想法，我们需要两步：

- 将所有酒按照**价值大小**排序，排序的过程中需要保证酒的总容积跟排序前对应。
- 不断尝试向这个杯子里面倒入**当前价值最大**的酒，直到所有酒都倒完或者是总容积到达 L 。

第一个过程可以采用多种方法完成，其中包括老师上课讲过的归并排序、插入排序等方法；而第二步可以采用递归或是非递归的方法完成，因为第二步有两种结束条件，所以大家实现的时候需要注意边界情况。

```
1  /* wine.c */
2  #include <stdio.h>
3  long long n, L, i, j, k, ans = 0, v[20000], w[20000], idx[20000];
4  int main()
5  {
6      scanf("%d %d", &n, &L);
7      for (i = 1; i <= n; i++)
8          idx[i] = i;
9      for (i = 1; i <= n; i++)
10         scanf("%d", &v[i]);
11     for (i = 1; i <= n; i++)
12         scanf("%d", &w[i]);
13     for (i = 1; i <= n; i++)
14         for (j = i + 1; j <= n; j++)
15             if (v[idx[i]] < v[idx[j]])
16                 {
17                     int t = idx[i];
18                     idx[i] = idx[j];
19                     idx[j] = t;
```

```
20     }
21     i = 1;
22     while (L > 0)
23     {
24         if (L >= w[idx[i]])
25         {
26             L -= w[idx[i]];
27             ans += w[idx[i]] * v[idx[i]];
28         }
29         else
30         {
31             ans += L * v[idx[i]];
32             L = 0;
33         }
34         i++;
35     }
36     printf("%d\n", ans);
37     return 0;
38 }
```

题面中一个疏忽的地方是没有说明输入数据的数据范围，但是这题中 `int` 可以通过，所以无伤大雅。

2 画长方体 (cuboid.c)

本题知识点：整数输入、循环、条件分支、字符（串）输出。

原来是 4-loops 被删减的题目之一。还好删掉了，要不然真的太难了。虽然大家感觉到有点难写，但是其实并没有太多技术含量，大家花一点力气写一写应该也可以写出来。



一种可能比较简单的方法是使用一个二维的 `char` 数组表示地图，然后将长方体分成三个面，分别使用两层的循环去“填”这个二维数组。

```
1  /* cuboid.c */
2  #include <stdio.h>
3  #include <assert.h>
4  #include <string.h>
5
6  #define N 105
7
8  char s[N][N];
9
10 int main() {
11     int T;
12     scanf("%d", &T);
13     for (int n, m, a, b, c; T--; ) {
14         scanf("%d%d%d", &a, &b, &c);
15         n = 2 * c + 1 + 2 * b;
16         m = 2 * a + 1 + 2 * b;
17         for (int i = 1; i <= n; i++)
18             for (int j = 1; j <= m; j++)
19                 s[i][j] = ' ';
20         for (int i = 2 * b + 1; i <= n; i += 2) {
21             for (int j = 1; j <= 2 * a + 1; j += 2) {
22                 s[i][j] = '+';
23                 if (j + 1 <= 2 * a + 1) s[i][j + 1] = '-';
24                 if (i <= n) s[i + 1][j] = '|';
25             }
26         }
27         for (int i = 1; i <= 2 * b; i += 2) {
28             for (int j = 2 * b - i + 1 + 1;
29                  j <= 2 * b - i + 2 * a + 1 + 1; j += 2) {
30                 s[i][j] = '+';
31                 if (j + 1 <= 2 * b - i + 2 * a + 1 + 1)
32                     s[i][j + 1] = '-';
33                 s[i + 1][j - 1] = '/';
34             }
35         }
36         for (int j = 2 * a + 1; j <= m; j += 2) {
37             for (int i = m + 1 - j;
38                  i <= m + 1 - j + 2 * c; i += 2) {
39                 s[i][j] = '+';
```

```

40         if (i + 1 <= m + 1 - j + 2 * c) s[i + 1][j] = '|';
41         if (j + 1 <= m) s[i - 1][j + 1] = '/';
42     }
43 }
44
45 for (int i = 1; i <= n; i++) {
46     for (int j = 1; j <= m; j++) {
47         putchar(s[i][j]);
48     }
49     puts("");
50 }
51 }
52 return 0;
53 }

```

你当然也可以把 3 个并列的 for 循环合并到一起。

```

1  /* cuboid.c */
2  #include <stdio.h>
3  char out[200][200];
4
5  int main() {
6      int T;
7      scanf("%d", &T);
8      while (T--) {
9          for (int i = 0; i < 200; i++)
10             for (int j = 0; j < 200; j++)
11                 out[i][j] = ' ';
12         int a, b, c;
13         scanf("%d%d%d", &a, &b, &c);
14         for (int i = 0; i <= 2 * (b + c); i += 2) {
15             for (int j = 0; j < 2 * a; j += 2) {
16                 out[i][j + (2 * b > i ? 2 * b - i : 0)] = '+';
17                 out[i][j + 1 + (2 * b > i ? 2 * b - i : 0)] = '-';
18                 out[i + 1][j + (2 * b > i ? 2 * b - i - 1 : 0)]
19                     = "|/"[2 * b > i];
20             }
21             for (int j = 2 * (a + b + c) - i;
22                 j >= (2 * b > i ? 2 * (a + b) - i : 2 * a);
23                 j -= 2) {

```

```

24         out[i][j] = '+';
25         out[i + 1][j] = " |[i + j != 2 * (a + b + c)];
26         out[i + 1][j - 1] = " /[j > 2 * a];
27     }
28 }
29 for (int i = 0; i <= 2 * (b + c); i++, putchar('\n'))
30     for (int j = 0; j <= 2 * (a + b); j++)
31         putchar(out[i][j]);
32 }
33 return 0;
34 }

```

如果你的技术特别高超，那么你也当然可以不使用数组直接输出。

```

1  /* cuboid.c */
2  #include <stdio.h>
3
4  int main() {
5      int T, a, b, c;
6      scanf("%d", &T);
7      while (T--) {
8          scanf("%d%d%d", &a, &b, &c);
9          for (int i = 0; i < 2 * (b + c) + 1; i++) {
10             for (int j = 0; j < 2 * (a + b) + 1; j++) {
11                 if (i + j < 2 * b || i + j > 2 * (a + b + c))
12                     putchar(' ');
13                 else if (i < 2 * b && i + j <= 2 * (a + b)) {
14                     if ((i & 1) && !(j & 1)) putchar(' ');
15                     else if ((i & 1) && (j & 1)) putchar('/');
16                     else if (!(i & 1) && !(j & 1)) putchar('+');
17                     else if (!(i & 1) && (j & 1)) putchar('-');
18                 }
19                 else if (i >= 2 * b && j <= 2 * a) {
20                     if ((i & 1) && (j & 1)) putchar(' ');
21                     else if ((i & 1) && !(j & 1)) putchar('|');
22                     else if (!(i & 1) && !(j & 1)) putchar('+');
23                     else if (!(i & 1) && (j & 1)) putchar('-');
24                 }
25                 else {
26                     if (!(i & 1) && (j & 1)) putchar(' ');

```

```

27         else if ((i & 1) && !(j & 1)) putchar('|');
28         else if (!(i & 1) && !(j & 1)) putchar('+');
29         else if ((i & 1) && (j & 1)) putchar('/');
30     }
31 }
32 putchar('\n');
33 }
34 }
35 return 0;
36 }

```

CQ 同学还写出了所有助教都没想到的方法，那就是将 $a \times b \times c$ 的长方体分解为若干个 $1 \times 1 \times 1$ 的单位立方体，在合适的位置填充画布。因为前面的立方体恰好遮挡住后面的立方体，所以这种方法写起来也比较简单。代码略。

3 倒水游戏 (pour.c)

本题知识点：整数输入、递归、循环、指针 / 数组的操作。

(2021 年)

(前一周出题前，蚂老师办公室)

蚂老师：这周题目可以简单一点。

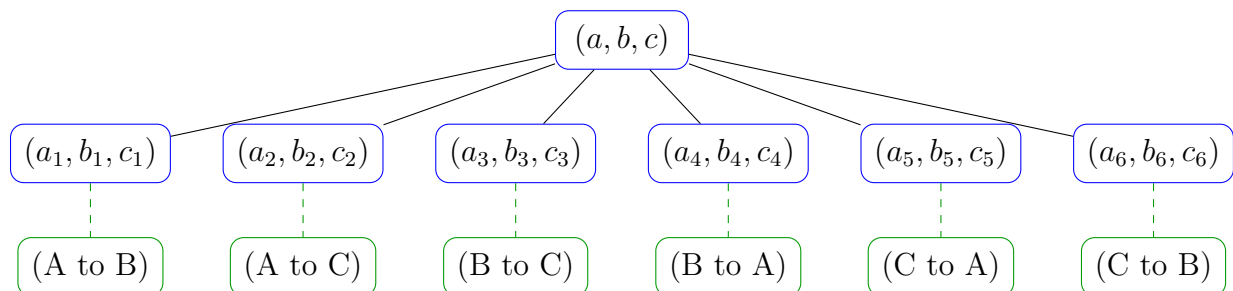
(题目基本上出好了)

蚂老师：不行，好像有点太简单了，要搞一个题目难难他们。

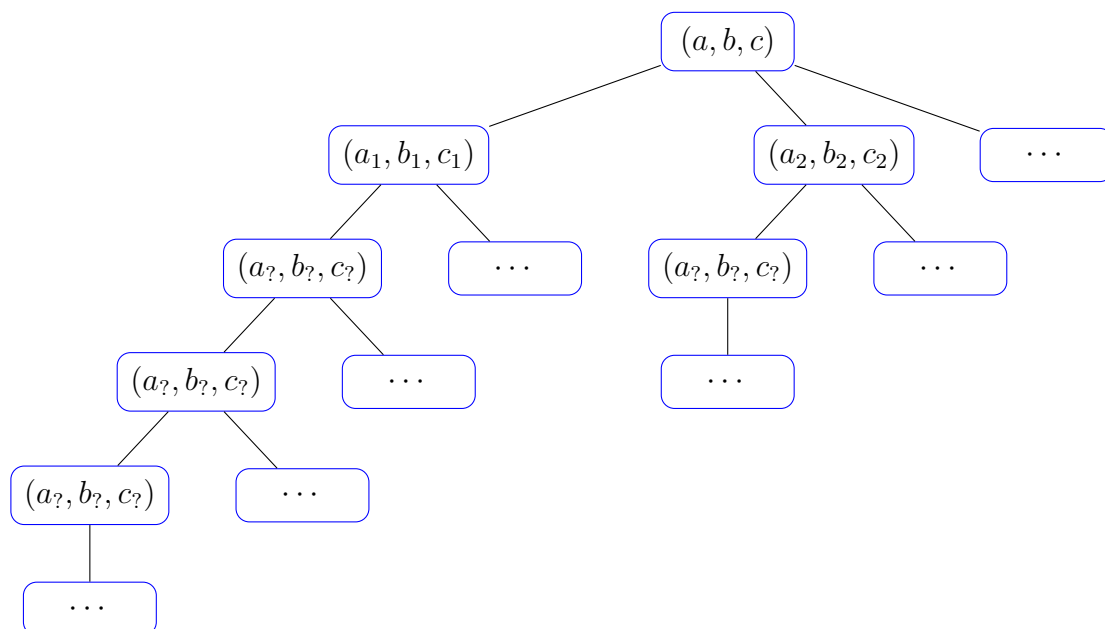
(此题诞生)

这个题面看上去就非常递归，为什么？大家可以回忆了一下上周的那个数字分解，我们采用的是一种“尝试”的思想，再仔细想想，我们其实在做一件“枚举全部可能”的工作。当你遇到一道题的时候，如果你对“如何用数学知识解决这道题”没什么想法，而这道题的数据范围又特别小，那么**枚举全部可能**就成为了一种行之有效的手段，这一类的递归本质上都是一种非常有效的枚举手段。

依然是采用尝试一步的方法进行枚举。考虑第一步怎么倒水，一共有 6 种可能，分别为：



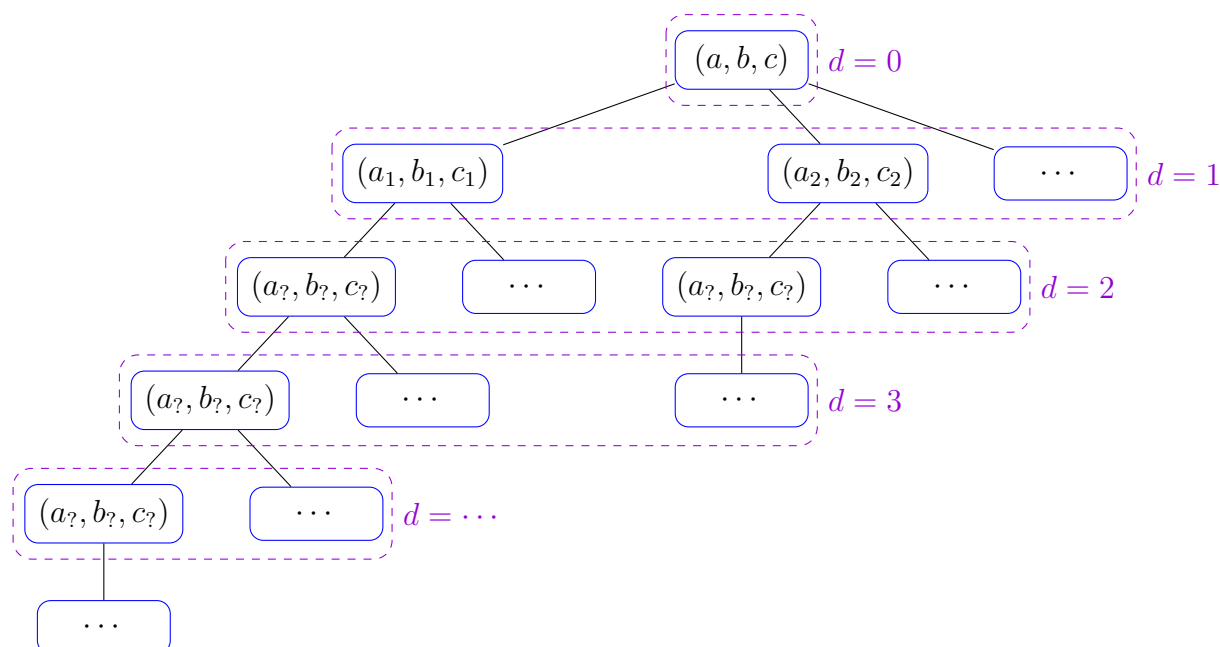
容易发现，我们希望得到的一个枚举的工作实际上是把一种水的状态不断“扩展”为六种不同的、新的水的状态，如果我们对每一种新出现的状态都执行这样的操作的话，那么我们就成功枚举了所有情况。



因为 6 这个数字实在是有一点点大，所以上图进行了很多的缩略，如果我们将上面得到的每一个新状态都不断地向下扩展的话，我们就成功枚举了每一个状态。如果这些状态中有一些是我们的目标状态的话，我们就知道了答案应该为 Yes。

容易发现，一个状态会扩展出 6 个状态，这个增长速度是非常非常快的，当我们向下扩展到第 d 层时，就已经有了 $\sum_{i < d} 6^i$ 个不同的状态，这是一个指数级别的增长速度。当 d 很大的时候，需要运算的数据量已经非常恐怖，哪怕是计算机也承受不了。

接下来考虑这个步数 k 的限制，如果我们给上图分分层，我们就可以发现当前倒了 d 次水相当于当前已经扩展到了第 d 层，所以当当前枚举的层数越界的时候我们让程序直接返回结束即可。



```
1 /* pour.c */
2 #include <stdio.h>
3
4 int k, va, vb, vc, a0, b0, c0, ans = 0;
```



```
5
6 void pour(int* c1, int* c2, int v1, int v2) {
7     /* pour c1 into c2 */
8     int sum = (*c1) + (*c2);
9     if (sum > v2) sum = v2;
10    *c1 = 0, *c2 = sum;
11 }
12
13 void solve(int stp, int a, int b, int c) {
14     if (stp > k) return;
15     if (a == a0 && b == b0 && c == c0) {
16         ans = 1;
17         return;
18     }
19     if (ans) return;
20     int oa = a, ob = b, oc = c;
21
22     a = oa, b = ob, c = oc;
23     pour(&a, &b, va, vb);
24     solve(stp + 1, a, b, c);
25
26     a = oa, b = ob, c = oc;
27     pour(&b, &a, vb, va);
28     solve(stp + 1, a, b, c);
29
30     a = oa, b = ob, c = oc;
31     pour(&a, &c, va, vc);
32     solve(stp + 1, a, b, c);
33
34     a = oa, b = ob, c = oc;
35     pour(&c, &a, vc, va);
36     solve(stp + 1, a, b, c);
37
38     a = oa, b = ob, c = oc;
39     pour(&b, &c, vb, vc);
40     solve(stp + 1, a, b, c);
41
42     a = oa, b = ob, c = oc;
43     pour(&c, &b, vc, vb);
```

```

44     solve(stp + 1, a, b, c);
45 }
46
47 int main() {
48     int a, b, c;
49     scanf("%d%d%d%d%d%d%d%d", &k, &va, &vb, &vc,
50         &a, &b, &c, &a0, &b0, &c0);
51     solve(0, a, b, c);
52     puts(ans ? "Yes" : "No");
53     return 0;
54 }

```

4 内存分配器 (buddy.c)

本题知识点：整数和字符（串）的输入，循环，条件分支，数组模拟链表。

强烈建议结合视频提示阅读本题题解，本题题解将会结合视频里给出的代码框架讲解。

在视频里面，我们已经讲解了“操作”类型的题目如何进行读入以及使用函数来进行一些分解和抽象。我们在本题中的任务只剩下了填写我们留下的 query、fit_size、get_best、allocate 函数。

不妨从简单的入手，对于 query，我们可以直接利用已有的信息输出。

```

1 void query() {
2     printf("%d\n", tot);
3     for (int i = 1; i <= tot; i++) {
4         printf("%d", id[i]);
5         printf(i != tot ? " " : "\n");
6     }
7 }

```

只需要回头看看 id、tot 等变量的定义就可以完成 query。

然后就是 fit_size 了，这个函数只要求出“恰好”能够放得下 m 的 2^k 的结果就行，我们可以从小到大遍历尝试。

```

1 int fit_size(int m) {
2     for (int i = 0; i <= n; i++) {
3         if ((1 << i) >= m) return i;
4     }
5     /* will not reach here. */
6 }

```

如果数据完全符合要求的话，这个函数在循环过程中一定会返回。另外一点可能需要强调的就是浮点数的精度误差问题。pow 在本题中是可接受的，但 log 和 log2 在本题中是不可接受的。

然后是 get_best，按照我们给出的定义，这个函数将会按照分裂的规则返回当前“最好”的空间节点，这一块既有可能直接用于分配也有可能用于分裂。

```
1 int get_best(int m) {
2     for (int i = 1; i <= tot; i++) {
3         if (id[i] != 0) continue;
4         if (siz[i] == m) return i;
5     }
6     for (int i = 1; i <= tot; i++) {
7         if (id[i] != 0) continue;
8         if (siz[i] > m) return i;
9     }
10    /* will not reach here */
11 }
```

上面两个 for 循环分别对应了寻找大小恰好的块和寻找需要分裂的块两个步骤。但其实下面的写法也是正确的。

```
1 int get_best(int m) {
2     for (int i = 1; i <= tot; i++) {
3         if (id[i] != 0) continue;
4         if (siz[i] >= m) return i;
5     }
6     /* will not reach here */
7 }
```

我个人 (czh) 认为这样的写法的正确性并不显然，但是确实是可以证明的，这里留作一个小小的 bonus，感兴趣的同学可以来证明一下。

那么最后就还剩下 allocate 函数啦。

```
1 void allocate(int cur_id, int m) {
2     while (1) {
3         int pos = get_best(m);
4         if (siz[pos] == m) {
5             id[pos] = cur_id;
6             break;
7         } else {
8             for (int i = tot + 1; i > pos + 1; i--) {
9                 id[i] = id[i - 1];
10                siz[i] = siz[i - 1];

```

```
11         }
12         ++tot;
13         --siz[pos];
14         siz[pos + 1] = siz[pos];
15         id[pos + 1] = 0;
16     }
17 }
18 }
```

这个函数剩下的部分在于如何完成“区间后移”，当然有很多种方法，我认为最简单的方法就是从后往前遍历（想一想，为什么是从后往前而不是从前往后），然后“一路”把相应数组的值复制过来。在后移完成之后，我们需要谨慎地处理 `pos` 和 `pos + 1` 两个位置的 `id` 和 `siz`，尤其是不要忘记将 `id[pos + 1]` 设为 0。

由于本题的核心代码已经全部给出，此处不再给出完整的标程。

5 括号序列 (brackets.c)

本题知识点：整数和字符串的输入、栈的模拟、字符串输出、字符串处理（使用栈）。

在阅读本题的题解之前，你需要先了解“栈”这种抽象的结构和它支持的 `push` 与 `pop` 两种核心操作。（我该怎么了解这些知识？）

括号序列的匹配是栈这种模型最经典的用法之一。我们发现在匹配的过程中，如果一个括号序列合法，那么每一个右括号都是和离它最近的那个左括号配对的。

于是使用一个栈，栈中元素为当前还没有配对的左括号。我们对整个括号序列从左到右依次扫描，如果：

- 遇到一个左括号，则将这个左括号入栈。
- 遇到一个右括号，则将栈顶（考虑一下，为什么是栈顶）的元素出栈，表示栈顶左括号与当前右括号匹配成功。

容易发现，如果匹配成功，最终栈中一定为空；但是最终栈中为空一定能保证得到了一个合法的括号序列吗？我们还需要保证所有匹配的过程是合法的，在此题中具体包括：

- 栈顶出栈的括号与当前的括号形状相符合（比如“（”不能和“]”匹配）。
- 匹配到一个右括号时，当前栈顶不能为空。

大家也可以推出其他正确的条件。还是那句话，正确的条件都能在数学上证明等价。

```
1 /* brackets.c */
2 #include <stdio.h>
3 #include <string.h>
```

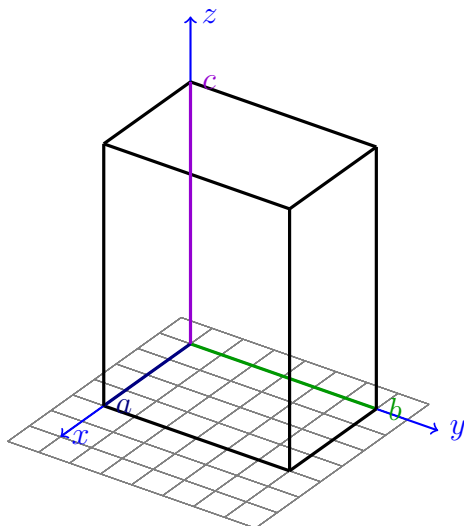
```
4 #include <stdbool.h>
5
6 #define N (100000 + 50)
7
8 int T, top;
9 bool vis[N];
10 char s[N], stk[N];
11
12 int main() {
13     scanf("%d", &T);
14     for (int n; T--; ) {
15         scanf("%s", s);
16         n = strlen(s), top = 0;
17         bool ok = true;
18         for (int i = 0; i < n; i++) {
19             char ch = s[i];
20             if (ch == '(' || ch == '[' || ch == '{') {
21                 stk[++top] = ch;
22             } else {
23                 if (!top) {
24                     ok = false;
25                     break;
26                 }
27                 if ((ch == ')' && stk[top] != '(')
28                     || (ch == ']' && stk[top] != '[')
29                     || (ch == '}' && stk[top] != '{')) {
30                     ok = false;
31                     break;
32                 }
33                 --top;
34             }
35         }
36         ok &= (top == 0);
37         puts(ok ? "True" : "False");
38     }
39     return 0;
40 }
```

6 立方体翻转 (rotate.c)

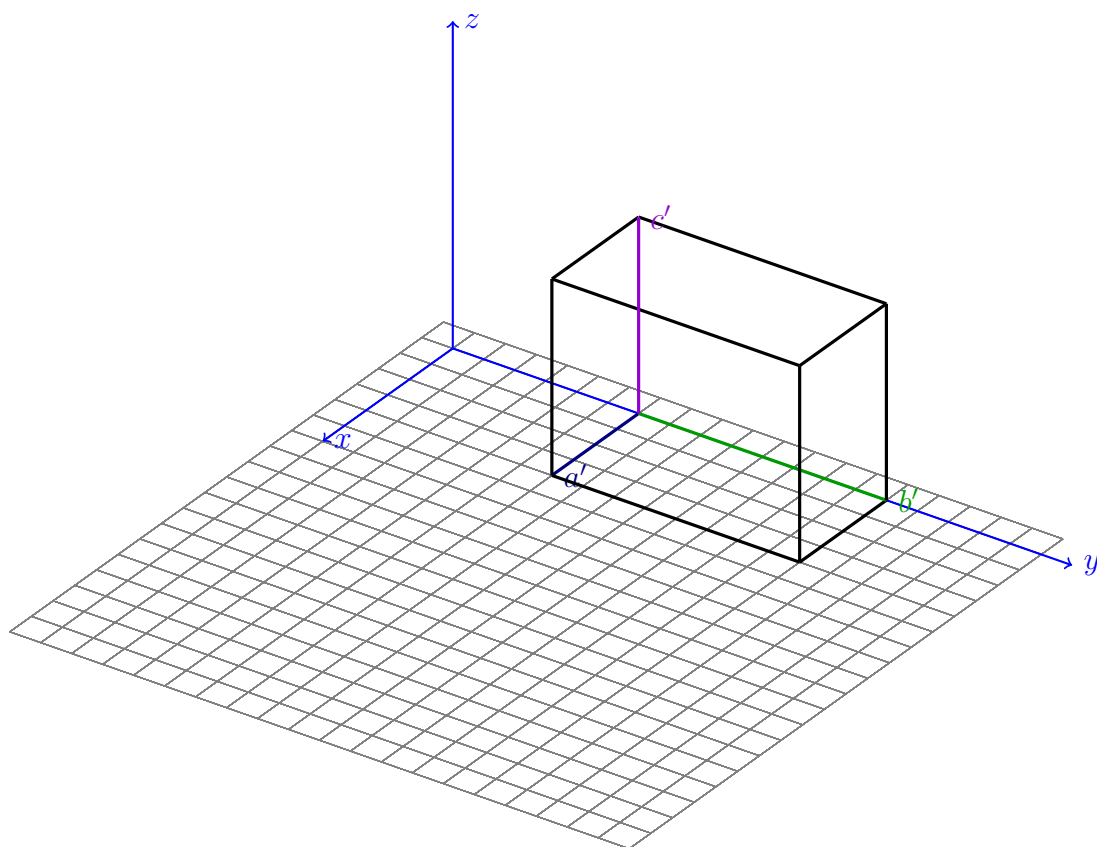
本题知识点：整数输入、循环、条件分支、整数输出。

这道题需要大家使用合理的方法在存储立方体的信息，否则就很容易写出很长的代码。

一种比较简单的方法应该是记录当前的最靠“里”的点的位置 (x, y) 和当前的三条棱的长度。



比如在上图中，立方体看上去一次都没有翻转过，即 $(x, y) = (0, 0)$ ，是原点坐标。我们可以考虑一次翻转之后变成了什么样，比如上图经过了一次 D 操作（向右翻转）之后。



我们可以发现，在上一步操作中， $a' = a, b' = c, c' = b$ (b, c 互换)，而 (x, y) 的坐标也相应变化。类似的，我们也可以找出其他几种操作的规律，这样我们程序的实现就十分简单了。

```
1  /* rotate.c */
2  #include <stdio.h>
3  #include <string.h>
4  #include <assert.h>
5
6  #define N 1005
7
8  int n;
9  char s[N];
10
11 void swap(int *a, int *b) {
12     int c = *a;
13     *a = *b;
14     *b = c;
15 }
16
17 int main() {
18     int a, b, c;
19     scanf("%d%d%d", &a, &b, &c);
20     scanf("%s", s + 1);
21     n = strlen(s + 1);
22     int x = 0, y = 0;
23     for (int i = 1; i <= n; i++) {
24         char ch = s[i];
25         switch (ch) {
26             case 'S': x += a; swap(&a, &c); break;
27             case 'W': x -= c; swap(&a, &c); break;
28             case 'A': y -= c; swap(&b, &c); break;
29             case 'D': y += b; swap(&b, &c); break;
30             default: assert(0);
31         }
32     }
33     printf("%d %d %d %d\n", x, x + a, y, y + b);
34     return 0;
35 }
```

7 写在最后

题解晚了很不好意思，大家一起咕咕咕。

由于写的比较赶，所以很有可能有各种错误或者遗漏的地方，非常欢迎大家指出！

感谢大家的理解和包容，祝我们都能顺利度过考试周。