学而思

03

Lesson

CSP-J初赛阅读程序及完善程序题技巧分析

阅读程序题

CSP初赛除了基础选择题，还有非常重要的两道大题：阅读程序和完善程序。阅读程序一共有3大题，共40分，是CSP-J/S考试中的一大拉分点。该题首先需要考生认真阅读程序，然后会采用判断题和选择题考查考生对程序的理解深度。

本题所考查的知识点非常多，主要包括计算机语言、多重循环的嵌套、数组的操作等。对于阅读程序题，考生一定要细心，注意循环的判断条件以及程序的输入输出细节，可以按照程序的运行顺序，从主函数还是阅读程序。

阅读程序题

阅读程序题通常考察考生以下几个方面的能力：

- 程序设计语言的掌握情况；
- 程序中基础算法的掌握情况；
- 数学计算的知识与运算能力；
- 细心、耐心的计算思维能力。

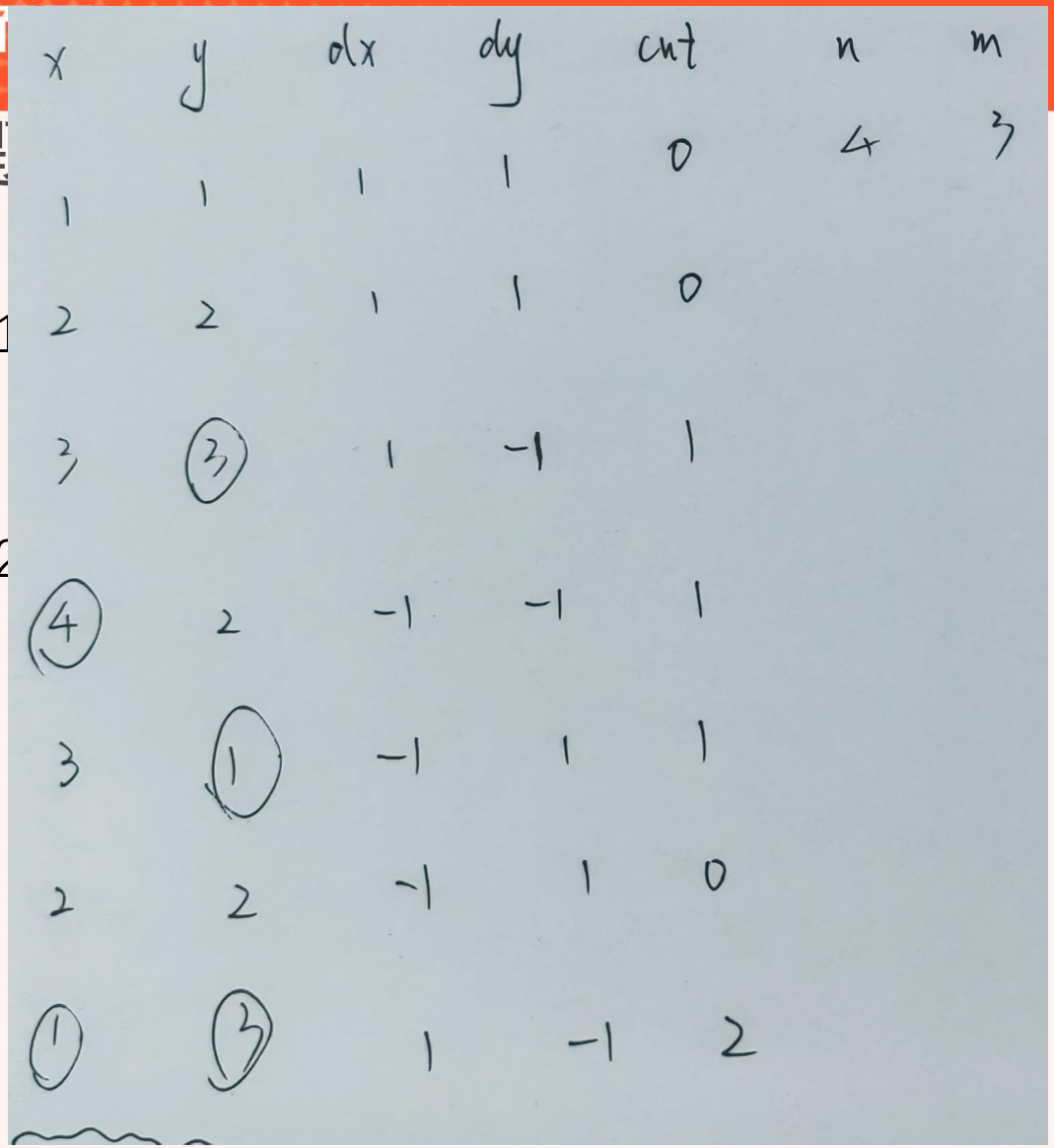
阅读程序题常见解题方法—模拟法

- 在程序阅读题中，如果难以理解题目所要完成的功能，最简单有效的方法就是**模拟法**。
- 所谓模拟法，就是利用人脑模拟程序的执行运算过程。
- 当程序涉及数组、循环、递归等更复杂的语句的时候，通常会有多个相似的变量发生变化，稍不留神就会导致整个程序发生错误，所以通常会**设计表格**来表示各种数据。

例題

(1

(2




```cpp
#include<iostream>
using namespace std;
int main()
{
    int n, m;
    cin>>n>>m;
    int x=1, y=1, dx=1, dy=1;
    int cnt=0;
    while(cnt!=2) {
        cnt=0;
        x=x+dx;
        y=y+dy;
        if(x==1 || x==n) {
            ++cnt;
            dx=-dx;
        }
        if(y==1 || y==m) {
            ++cnt;
            dy=-dy;
        }
    }
    cout<<x<<" "<<y<<endl;
    return 0;
}
```

阅读程序题常见解题方法—猜测验证

模拟法虽然很有用，但如果考生对整个程序的功能不够理解，则会造成求解的速度很慢。

在详细地阅读程序之前，考生可以借助以前学习程序的经验以及程序中变量和函数一些常用的写法提示，大胆地猜测程序的功能，然后再进行验证。

学而思

阅读程序题常见解题方法—猜测验证

(1) 变量的常用含义

序号	变量名	英文单词全拼/含义	序号	变量名	英文单词全拼/含义
1	ans	answer/计算答案	17	index	索引,多用于数组下标
2	ret	return/返回值	18	first	第一,多用于比较
3	res	result/计算结果	19	second	第二,多用于比较
4	flag	标识,多用于记录状态	20	last	最后,多用于比较
5	done	完成	21	begin	开始,多用于指针位置
6	error	错误,多用于记录错误信息	22	end	结束,多用于指针位置
7	found	找到,多用于 bool 变量	23	start	开始,多用于指针位置
8	success	成功	24	node	节点,多用于链表
9	ok	完成	25	op	操作符,多用于链表
10	num	number/数字	26	min	最小值,多用于比较
11	value	值	27	max	最大值,多用于比较
12	cnt	count/统计	28	avg	平均值,多用于计算
13	target	目标	29	total	总和,多用于统计
14	record	记录	30	preNode	上一个节点,用于链表
15	foo	确实存在东西的普遍替代语	31	curNode	当前节点,用于链表
16	tmp/temp	临时变量	32	nextNode	下一个节点,用于链表

阅读程序题常见解题方法—猜测验证

（2）算法的结构

很多常见的算法都有一些的特殊结构，了解并掌握这些结构不仅能在阅读程序上快速地判断出程序的功能，而且在以后编写程序时，也能通过算法结构快速地写出程序。

阅读程序题常见解题方法—猜测验证

（2）算法的结构

1、二分法

```
while(l <= r)
{
    mid = (l+r) / 2;
    …
    if(…) l = mid + 1;
    else r = mid – 1;
}
```

阅读程序题常见解题方法—猜测验证

（2）算法的结构

2、桶排序

```
for(int i = 1; i <= n; i++) b[a[i]]++;
for(int i = 1; i <= m; i++)
        for(int j = 1; j <= b[i]; j++)
                c[++cnt] = i;
```

阅读程序题常见解题方法—猜测验证

（2）算法的结构

3、连续字符转换成整数

```
num=0;
cin>>c;
while(c>='0'&&c<='9') {
        num=num*10+c-'0’  ;
        cin>>c;
}
```

阅读程序题常见解题方法—猜测验证

（2）算法的结构

4、链表

data[i]

last[next[i]] = last[i];

next[last[i]] = next[i];

阅读程序题常见解题方法—猜测验证

（2）算法的结构

5、分治法

```
solve(l, mid, ...);
solve(mid+1, r, ...);
```

阅读程序题常见解题方法—猜测验证

（2）算法的结构

6、二叉树

left, right, father, depth

l, r, f

dfs(left, k-1, ...);

dfs(k+1, right, ...);

学而思

例题2 阅读程序写结果

输入：abacaba

输出：16

```cpp
#include <iostream>
#include <string>
using namespace std;
string s;
long long magic(int l, int r)
{
    long long ans = 0;
    for(int i = l; i <= r; ++i)
        ans = ans * 4 + s[i] -'a' + 1;
    return ans;
}

int main()
{
    cin>> s;
    int len = s.length();
    int ans = 0;
    for(int l1 = 0; l1 < len; ++l1)
        for(int r1 = l1; r1 < len; ++r1)
        {
            bool bo = true;
            for(int l2 = 0; l2 < len; ++l2)
                for (int r2 = l2; r2 < len; ++r2)
                {
                    if (magic(l1, r1) == magic(l2, r2) && (l1 != l2 || r1 != r2))
                        bo = false;
                }
            if (bo) ans += 1;
        }
    cout << ans << endl;
    return 0;
}
```

完善程序题

在CSP初赛中，完善程序一共有2大题，共30分。完善程序通常会涉及到具体问题的解决和编程，因此具有非常强的整体性。

另一方面，完善程序是分差最大的题型，对于编程基础薄弱的学生，可能面对不完整的程序找不到切入点，导致得分很低；而对于代码能力较强的学生，可能很熟悉相关的问题和代码，得分比较轻松。

对于完善程序题，具体的做题技巧作用是有限的，更需要考生平时多多练习和积累，提高算法的代码实现能力。

完善程序题

完善程序题通常考察考生以下几个方面的能力：

- 算法问题的分析能力；
- 经典代码的编写能力；
- 代码功能的阅读能力；
- 代码细节的实现能力。

完善程序题常见解题方法—模拟验证法

由于大部分程序都是对变量或者数组进行数据操作，因此每一条语句的效果都是可以预见的。

完善程序题通常是在四个选项中选出一个，选项之间区别很小的时候，使用具体的数据来模拟每个选项的效果就成为了准确度最高的一种方法。我们可以用相对简单的数据来验证选项的正确性。

学而思

例题3.（快速排序）快速排序是一个常用且经典的排序算法，它基于分治法，基本思想如下：

1)先从数组中取出一个数作为基准数。

2)将比这个数小的数全放在数组的左边，大于等于这个数的数全放到数组的右边，生成左右两个区间。

3)再对左右区间分别重复一、二步，直到各区间只有一个数。

现在给定一个由n个无序整数组成的数组s，使用快速排序将s数组从小到大排序，试补全程序。

```
#include<iostream>
using namespace std;
const int maxn = 100000;
int n, s[maxn];
void quick_sort(int s[], int l, int r)
{
    if (①) return; //只有一个整数时排序结束
    int i = l, j = r, x = s[l]; //选第一个数为基准数
    while (i < j)
    {
        while(i < j && s[j] >= x)//从右向左找第一个小于 x 的数
            j--;
        if(i < j) {
            s[i] = s[j];
            i++;
        }
        while(i < j && ②) //从左向右找第一个大于等于 x 的数
            i++;
        if(i < j) {
            ③;
            j--;
        }
    }
    s[i] = x;
    quick_sort(s, l, i - 1);
    quick_sort(s, ④);
}
int main()
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> s[i];
    quick_sort(s, ⑤);
    for(int i = 0; i < n; i++)
        cout << s[i] << ' ';
    return 0;
}
```

学而思

例题3.（快速排序）

1)①处应填（ ）。

A. l > r　　B. l >= r

C. l < 0 || r >= n　　D. l < r

2)②处应填（ ）。

A. s[j] < x　　B. s[j] >= x

C. s[i] < x　　D. s[i] >= x

3)③处应填（ ）。

A. s[i] = s[j]　　B. s[j] = s[i]

C. s[i] = x　　D. s[j] = x

4)④处应填（ ）。

A. 0, n - 1　　B. l, r

C. i, r　　D. i + 1, r

5)⑤处应填（ ）。

A. 0, n - 1　　B. 1, n

C. 0, 0　　D. n, n

```
#include<iostream>
using namespace std;
const int maxn = 100000;
int n, s[maxn];
void quick_sort(int s[], int l, int r)
{
    if (①) return; //只有一个整数时排序结束
    int i = l, j = r, x = s[l]; //选第一个数为基准数
    while (i < j)
    {
        while(i < j && s[j] >= x)//从右向左找第一个小于 x 的数
            j--;
        if(i < j) {
            s[i] = s[j];
            i++;
        }
        while(i < j && ②) //从左向右找第一个大于等于 x 的数
            i++;
        if(i < j) {
            ③;
            j--;
        }
    }
    s[i] = x;
    quick_sort(s, l, i - 1);
    quick_sort(s, ④);
}
int main()
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> s[i];
    quick_sort(s, ⑤);
    for(int i = 0; i < n; i++)
        cout << s[i] << ' ';
    return 0;
}
```

完善程序题常见解题方法—模仿相似代码

在我们编写程序的时候，常常会出现相似度很高的代码或者语句，它们通常是对不同的对象做相同或者相似的处理。这种现象常常出现在枚举、分治或者树结构相关的程序上。当我们需要补全语句的时候，参考与它相似的段落往往会给我们带来很多提示和启发。

学而思

例题4. （二叉查找）

二叉查找树具有如下性质：**每个节点的值都大于其左子树上所有节点的值、小于其右子树上所有节点的值。**

试判断一棵树是否为二叉查找树。

输入的第一行包含一个整数n，表示这棵树有n个顶点，编号分别为1, 2, … , n，其中编号为1的为根结点。之后的第i行有三个数value, lch, rch，分别表示该节点关键字的值、左子节点的编号、右子节点的编号（-100,000 ≤ value ≤ 100,000, 1 ≤ lch, rch ≤ n）；如果不存在左子节点或右子节点，则用0代替。输出1表示这棵树是二叉查找树，输出0则表示不是，试补全程序。

```
#include <iostream>
using namespace std;
const int SIZE = 100, INFINITE = 1000000;
struct node {
    int lch; // Left child
    int rch; // right child
    int value;
};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound)
{
    if (root == 0)
        return 1;
    int cur = ①;
    if ((cur > lower_bound) && (cur < upper_bound) &&
        (is_bst( ② , lower_bound, cur) == 1) &&
        (is_bst(a[root].rch, ③ , ④ ) == 1))
          return 1;
    return 0;
}
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i].value >> a[i].lch >> a[i].rch;
    cout << is_bst( ⑤ , -INFINITE, INFINITE) << endl;
    return 0;
}
```

例题4. （二叉查找）

1) ①处应填（ ）。

A. a[root].value　B. a[root].lch

C. a[root].rch　　D. root

2)②处应填（ ）。

A. a[root].value　B. a[root].lch

C. a[root].rch　　D. root

3)③处应填（ ）。

A. lower_bound　　B. upper_bound

C. cur　　D. root

4)④处应填（ ）。

A. lower_bound　　B. upper_bound

C. cur　　D. root

5)⑤处应填（ ）。

A. 0　　B. 1

C. n　　D. n-1

```
#include <iostream>
using namespace std;
const int SIZE = 100, INFINITE = 1000000;
struct node {
    int lch; // Left child
    int rch; // right child
    int value;
};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound)
{
    if (root == 0)
        return 1;
    int cur = ①;
    if ((cur > lower_bound) && (cur < upper_bound) &&
        (is_bst( ② , lower_bound, cur) == 1) &&
        (is_bst(a[root].rch, ③ , ④ ) == 1))
          return 1;
    return 0;
}
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i].value >> a[i].lch >> a[i].rch;
    cout << is_bst( ⑤ , -INFINITE, INFINITE) << endl;
    return 0;
}
```

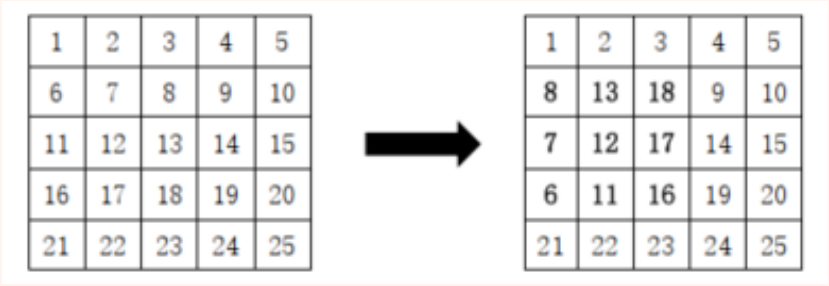
完善程序题常见解题方法—相关变量

对于大部分程序员来说，一个变量的使用场景常常是相同的，在多变量之间进行分辨的时候，如果能明确每个变量的意义，就能更快地排除错误选项，完成题目。

学而思

例题5.（矩阵变换）

给定一个n行n列的矩阵，矩阵初始由1到n^2的正整数按照从左往右，从上到下的顺序依次填入。接下来对这个矩阵进行m次变换，每次变换需要输入四个整数x, y, r, z，表示把以第x行第y列为中心的边长为2r+1的方阵按照顺时针或者逆时针方向旋转90°，其中z=0表示顺时针，z=1表示逆时针。例如当n=5, m=1, x=3, y=2, r=1, z=1时，矩阵变换如下图所示：



编写程序求m次变换后得到的矩阵，试补全程序。

```
#include<cstdio>
using namespace std;
const int maxn = 505;
int n, m, d[maxn][maxn], f[maxn][maxn];
int main()
{
    scanf("%d%d", &n, &m);
    int tot = 0, x, y, r, z;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            d[i][j] = ①;
    for(int k = 1; k <= m; k++)
    {
        scanf("%d%d%d%d", &x, &y, &r, &z);
        if(z == 0) {
            for(int i = x - r; i <= x + r; i++)
                for(int j = y - r; j <= y + r; j++)
                    f[x - y + j][y + x - i] = d[i][j];
        }
        else {
            for(int i = x - r; i <= x + r; i++)
                for(int j = y - r; j <= y + r; j++)
                    f[②][③] = d[i][j];
        }
        for(int i = x - r; i <= x + r; i++)
            for(int j = ④; j++)
                ⑤;
    }
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
            printf("%d ", d[i][j]);
        printf("\n");
    }
    return 0;
}
```

学而思

例题5. （矩阵变换）

1)①处应填（ ）。

A. ++tot　　B. i

C. tot++　　D. j

2)②处应填（ ）。

A. x – y + j　　B. x + y - j

C. y – x + i　　D. y + x - i

3)③处应填（ ）。

A. x – y + j　　B. x + y - j

C. y – x + i　　D. y + x - i

4)④处应填（ ）。

A. y - r; j <= y + r　　B. 1; j <= y

C. y; j <= n　　D. 1; j <= n

5)⑤处应填（ ）。

A. d[i][j] = f[j][i]　　B. d[i][j] = f[i][j]

C. f[i][j] = d[j][i]　　D. f[i][j] = d[i][j]

```
1  #include<cstdio>
2  using namespace std;
3  const int maxn = 505;
4  int n, m, d[maxn][maxn], f[maxn][maxn];
5  int main()
6  {
7     scanf("%d%d", &n, &m);
8     int tot = 0, x, y, r, z;
9     for (int i = 1; i <= n; i++)
10       for (int j = 1; j <= n; j++)
11          d[i][j] = ①;
12    for(int k = 1; k <= m; k++)
13    {
14       scanf("%d%d%d%d", &x, &y, &r, &z);
15       if(z == 0) {
16          for(int i = x - r; i <= x + r; i++)
17             for(int j = y - r; j <= y + r; j++)
18                f[x - y + j][y + x - i] = d[i][j];
19       }
20       else {
21          for(int i = x - r; i <= x + r; i++)
22             for(int j = y - r; j <= y + r; j++)
23                f[②][③] = d[i][j];
24       }
25       for(int i = x - r; i <= x + r; i++)
26          for(int j = ④; j++)
27             ⑤;
28    }
29    for(int i = 1; i <= n; i++)
30    {
31       for(int j = 1; j <= n; j++)
32          printf("%d ", d[i][j]);
33       printf("\n");
34    }
35    return 0;
37 }
```

完善程序题常见解题方法—经典算法实现

完善程序题通常会先给一个应用编程题，然后需要编写程序解决。大部分题目都会涉及到一个或多个具体的算法，而很多算法是有明显的代码编写特色的。因此，利用算法本身约定俗成的写法，可以帮助我们完成题目。

学而思

例题6.（最大公约数之和）（2018年）

下列程序想要求解整数n的所有约数两两之间最大公约数的和对10007求余后的值，试补全程序。（第一空2分，其余3分）

举例来说，4的所有约数是1,2,4。1和2的最大公约数为1；2和4的最大公约数为2；1和4的最大公约数为1。于是答案为1 + 2 + 1 = 4。要求getDivisor函数的复杂度为O(√n)，gcd函数的复杂度为O(log max(a,b))。

```
#include <iostream>
using namespace std;
const int N = 110000, P = 10007;
int n;
int a[N], len;
int ans;
void getDivisor() {
  len = 0;
  for (int i = 1; (1) <= n; ++i)
    if (n % i == 0) {
      a[++len] = i;
      if ( (2) != i) a[++len] = n / i;
    }
}

int gcd(int a, int b) {
  if (b == 0) {
    (3) ;
  }
  return gcd(b, (4) );
}

int main() {
  cin >> n;
  getDivisor();
  ans = 0;
  for (int i = 1; i <= len; ++i) {
    for (int j = i + 1; j <= len; ++j) {
      ans = ( (5) ) % P;
    }
  }
  cout << ans << endl;
  return 0;
}
```

学而思

例题7.（郊游活动）（NOIP2016）

有n名同学参加学校组织的郊游活动，已知学校给这n名同学的郊游总经费为A元，与此同时，第i位同学自己携带了Mi元。为了方便郊游，活动地点提供B（≥n）辆自行车供人租用，租用第j辆自行车的价格为Cj元，每位同学可以使用自己携带的钱或者学校的郊游经费，为了方便账务管理，每位同学只能为自己租用自行车，且不会借钱给他人，他们想知道最多有多少位同学能够租用到自行车。（第四、五空2.5分，其余3分。）

本题采用二分法。对于区间[1，r]，我们取中间点mid并判断租用到自行车的人数能否达到mid。判断的过程是利用贪心算法实现的。

```
#include<iostream>
using namespace std;
#define MAXN 1000000
int n, B, A, M[MAXN], C[MAXN], l, r, ans, mid;
bool check(int nn) {
    int count = 0, i, j;
    i = ____(1)____;
    j = 1;
    while(i <= n) {
        if(____(2)____)
            count += C[j] - M[i];
        i++;
        j++;
    }
    return ____(3)____;
}
void sort(int a[], int l, int r) {
    int i = l, j = r, x = a[(l + r) / 2], y;
    while (i <= j) {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j){
            y=a[i]; a[i]=a[j]; a[j]=y;
            i++; j--;
        }
    }
    if(i < r) sort(a, i, r);
    if(l < j) sort(a, l, j);
}
int main() {
    int i;
    cin >> n >> B >> A;
    for(i=1; i<=n; i++)
        cin >> M[i];
    for(i=1; i<=B; i++)
        cin >> C[i];
    sort(M, 1, n);
    sort(C, 1, B);
    l = 0;
    r = n;
    while(l <= r) {
        mid = (l+r) / 2;
        if(____(4)____){
            ans = mid;
            l = mid + 1;
        }else
            r = ____(5)____;
    }
    cout<<ans<<endl;
    return 0;
}
```

完善程序题常见解题方法—参考文字提示

由于完善程序题的整体难度很高，阅读陌生代码对考生综合素质要求很高，而且很多题目写法不止一种。出题人为了保证考生能顺利理解代码，往往会在说明部分解释算法，或者在代码关键位置添加注释。而这些关键的文字信息，可以成为我们做出题目的重要助力。

例题8.（序列重排）

全局数组变量a定义如下：

```
const int SIZE=100;
int a[SIZE], n;
```

它记录着一个长度为n的序列a[1], a[2], …, a[n]。

现在需要一个函数，以整数p（1≤p≤n)为参数，实现如下功能：将序列a的前p个数与后n-p个数对调，且不改变这p个数（或n p个数）之间的相对位置。例如，长度为5的序列1, 2, 3, 4, 5，当p=2时重排结果为3, 4, 5, 1, 2。

有一种朴素的算法可以实现这一需求，其时间复杂度为O(n)、空间复杂度为O(n)：

有一种朴素的算法可以实现这一需求，其时间复杂度为$O(n)$、空间复杂度为$O(n)$：

```
void swap1(int p) {
    int i, j, b[SIZE];
    for(i=1; i<=p; i++)
        b[    ①    ] = a[i];      //（3分）
    for(i=p+1; i<=n; i++)
        b[i-p] =    ②    ;       //（3分）
  for(i=1; i<=    ③    ; i++)     //（2分）
        a[i] = b[i];
}
```

例题8.（序列重排）

全局数组变量a定义如下：

```
const int SIZE=100;
int a[SIZE], n;
```

它记录着一个长度为n的序列a[1], a[2], …, a[n]。

现在需要一个函数，以整数p（1≤p≤n)为参数，实现如下功能：将序列a的前p个数与后n-p个数对调，且不改变这p个数（或n p个数）之间的相对位置。例如，长度为5的序列1, 2, 3, 4, 5，当p=2时重排结果为3, 4, 5, 1, 2。

有一种朴素的算法可以实现这一需求，其时间复杂度为O(n)、空间复杂度为O(n)：

我们也可以用时间换空间，使用时间复杂度为$O(n^2)$、空间复杂度为$O(1)$的算法：

```
void swap2(int p) {
    int i, j, temp;
    for(i=p+1; i<=n; i++) {
      temp = a[i];
      for(j=i; j >= ④ ; j--) //（3分）
            a[j] = a[j-1];
      ⑤ = temp;      //（3分）
    }
}
```

完善程序题常见解题方法—特殊数据代入

有时候我们观察选项，会发现一些选项相似度过高，如果选择特殊数据，很多选项都会得到一样的结果，那么就能同时排除多个错误选项。对于两个高度相似的选项，也可以设计一个特殊数据，使两者结果不同，然后通过模拟找出正确的结果，就可以锁定正确答案了。

学而思

例题9.【CSP-J2019】

1) ①处应填（ ）

A. `n % 2`　　B. `0`　　C. `t`　　D. `1`

2) ②处应填（ ）

A. `x - step, y - step`　　B. `x, y - step`

C. `x - step, y`　　D. `x, y`

3) ③处应填（ ）

A. `x - step, y - step`　　B. `x + step, y + step`

C. `x - step, y`　　D. `x, y - step`

4) ④处应填（ ）

A. `n - 1, n % 2`　　B. `n, 0`

C. `n, n % 2`　　D. `n - 1, 0`

5) ⑤处应填（ ）

A. `1 << (n + 1)`　　B. `1 << n`

C. `n + 1`　　D. `1 << (n - 1)`

```
1  #include <cstdio>
2  using namespace std;
3  int n;
4  const int max_size = 1 << 10;
5
6  int res[max_size][max_size];
7
8  void recursive(int x, int y, int n, int t) {
9    if (n == 0) {
10     res[x][y] = ①;
11     return;
12   }
13   int step = 1 << (n - 1);
14   recursive(②, n - 1, t);
15   recursive(x, y + step, n - 1, t);
16   recursive(x + step, y, n - 1, t);
17   recursive(③, n - 1, !t);
18 }
19
20 int main() {
21   scanf("%d", &n);
22   recursive(0, 0, ④);
23   int size = ⑤;
24   for (int i = 0; i < size; ++i) {
25     for (int j = 0; j < size; ++j)
26       printf("%d", res[i][j]);
27     puts("");
28   }
29   return 0;
30 }
```

学而思

例题9.【CSP-J2019】

（矩阵变幻）有一个奇幻的矩阵，在不停的变幻，其变幻方式为：数字0变成矩阵$\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}$，数字1变成矩阵$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$。最初该矩阵只有一个元素0，变幻 n 次后，矩阵会变成什么样？

例如，矩阵最初为：[0]；矩阵变幻 1 次后：$\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}$；矩阵变幻 2 次后：

$$\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & 1\\ 0 & 0 & 1 & 1\\ 0 & 1 & 1 & 0\end{bmatrix}。$$

输入一行一个不超过 10 的正整数 n。输出变幻 n 次后的矩阵。

试补全程序。

提示：

“<<”表示二进制左移运算符，例如$(11)_2 << 2 = (1100)_2$；

而“^”表示二进制异或运算符，它将两个参与运算的数中的每个对应的二进制位一一进行比较，若两个二进制位相同，则运算结果的对应二进制位为 0，反之为 1。

```
#include <cstdio>
using namespace std;
int n;
const int max_size = 1 << 10;

int res[max_size][max_size];

void recursive(int x, int y, int n, int t) {
  if (n == 0) {
    res[x][y] = ①;
    return;
  }
  int step = 1 << (n - 1);
  recursive(②, n - 1, t);
  recursive(x, y + step, n - 1, t);
  recursive(x + step, y, n - 1, t);
  recursive(③, n - 1, !t);
}

int main() {
  scanf("%d", &n);
  recursive(0, 0, ④);
  int size = ⑤;
  for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j)
      printf("%d", res[i][j]);
    puts("");
  }
  return 0;
}
```