第5章 阅读程序和完善程序

5.1 阅读程序和完善程序概述

从本章开始，本书分析考试的后两道大题：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。该题首先让考生阅读程序，然后利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，程序从主函数main开始顺序执行，要注意循环的判断条件以及程序的输入和输出细节。

完善程序一共2有大题，共30分。完善程序题将程序的核心部分空出一些空格，让考生从四个选项中选出正确的答案补充完全整个程序。该题型一般都会告诉考生程序要完成的功能，考查考生对于各类算法的设计思路、实现过程等具体步骤。要想补充完善程序，首要点就是弄清楚程序实现的思路和方法，然后具体了解程序实现的细节。

5.2 常用解题方法

阅读程序题和完善程序题主要考查考生以下几个方面的能力：

- 程序设计语言的掌握情况；
- 程序中基本算法的掌握情况；
- 数学的知识面及运算能力；
- 细心、耐心的计算思维。

该部分的解题也是有规律可循的，本书主要总结了以下两种解题方法。

5.2.1 模拟法

在CSP-J/S的阅读程序题中，当考生无法得知程序所要完成的功能时，最简单且最有效的方法就是“模拟法”。所谓“模拟法”就是利用人脑模拟程序的执行过程，只要题目不是很复杂，这种方法就比较奏效。但在模拟过程中，要特别注意各个变量变化时书写的认真和整洁，因为一个变量的计算错误就会引起整个程序结果的错误。

当程序涉及数组、循环、双重循环、递归调用等稍微复杂的语句时，这些过程模拟往往涉及较多的变量变化，稍不留神就可能会导致整个程序的错误。为了让整个过程整洁有序，从而快速发现程序的运行规律，设计表格以进一步模拟是非常有必要的。

案例 5-1　模拟法演示(1)

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, u, i, num;
    cin>>a>>b>>u;
    num =0;
    for (i =a; i <=b; i++)
    if ((i * i %u) ==1)
        num++;
    cout<<num<<endl;
    return 0;
}
```

【分析】 该题通过观察很难找出功能说明,但程序并不复杂,可以直接使用模拟法。模拟法的一般思路为:首先将程序涉及的变量和表达式列举出来,然后从 main 函数的第一句开始,利用人脑“模拟”程序的执行,一边执行一边记录变量和表达式值的变化,直至找到规律或程序执行结束。该题的模拟过程如表 5-1 所示。

表 5-1　案例 5-1 模拟过程

序号	a	b	u	num	i	i * i % u	备　注
1	3	10	8	0			初始化
2				1	3	1	开始循环
3				1	4	0	
4				2	5	1	
5				2	6	4	
6				3	7	1	
7				3	8	0	
8				4	9	1	
9				4	10	4	结束循环

案例 5-2　模拟法演示(2)

```
#include<cstdio>
using namespace std;
int n;
int a[100];
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%d",&a[i]);
```

```
9       }
10      int ans=1;
11      for(int i=1;i<=n;++i){
12          if(i>1&&a[i]<a[i-1])
13              ans=i;
14          while(ans<n && a[i]>=a[ans+1])
15              ++ans;
16          printf("%d ",ans);
17      }
18      printf("\n");
19      return 0;
20  }
```

• 判断题

(1) 第 16 行输出 ans 时，ans 的值一定大于 i。　　(　　)

(2) 程序输出的 ans 小于或等于 n。　　(　　)

(3) 若将 12 行的“<”改为“！=”，则程序的输出结果不会改变。　　(　　)

(4) 当程序执行到第 16 行时，若 ans-i>2，则 a[i+1]<=a[i]。　　(　　)

• 选择题

(5) 若输入的 a 数组是一个严格单调递增的数列，则此程序的时间复杂度是(　　)。

A. O(log n)　　B. $O(n^2)$　　C. O(nlog n)　　D. O(n)

(6) 最坏情况下，此程序的时间复杂度是(　　)。

A. $O(n^2)$　　B. O(log n)　　C. O(n)　　D. O(nlog n)

【分析】 阅读该题，可以发现程序里面就是数组的操作，直观上看不出程序的功能，这时需要利用一组数据验证程序的功能，选择数据时，如果题目中有数据，则可以直接选择题目中的数据；如果题目中没有数据，则根据题目构造一组数据。

本题没有直接给出测试数据，但可以构造一组数据：3 2 1 3。下面根据这组数据对题目进行具体分析，如表 5-2 和表 5-3 所示。

(1) 数据初始化(6～10 行)

表 5-2　案例 5-2 数据初始化

变量	n	a[1]	a[2]	a[3]	ans
变量值	3	2	1	3	1

(2) 数据处理(11～17 行)

表 5-3　案例 5-2 数据处理

变　量	i	ans	a[i]	a[i－1]	a[ans＋1]	输出
变量值	1	1	2		1	
		2	2		3	2
	2	2	1	2		

续表

变　量	i	ans	a[i]	a[i−1]	a[ans+1]	输出
		2	1		3	2
	3	2	3	1	3	
		3				3

通过数据分析可以得出：整个程序的含义是找到每个 a[i]后第一个大于 a[i]的位置(注：如果存在，则输出位置序号，序号范围为 0～n−1)，如果不存在大于 a[i]的数，则输出位置 n。

判断题(1)解析：

如果 12 行的 if 成立，而 14 行的 while 不成立，则 ans 的值与 i 相等。

参考答案：错误。

判断题(2)解析：

从 15 行看，ans<n 才会执行 ans++，如果 ans==n，则会退出 while 循环，所以不会超过 n。

参考答案：正确

判断题(3)解析：

将“<”改成“!=”，多出了一些比较操作，最后结果不受影响。

参考答案：正确

判断题(4)解析：

解析：由 ans-i>2 可见，ans 是第一个大于 a[i]的，所以从 a[i+1]到 a[ans-1]都不会超过 a[i]。

参考答案：正确

选择题(5)解析：

解析：如果输入数据为单调递增，则 12 行的 if 就不会成立，也就是 ans 只增不减，所以时间复杂度为 O(n)。

参考答案：D

选择题(6)解析：

解析：最坏情况下，12 行的 if 总是成立(a 单调降)，此时 14 行也会一直运行到 ans=n，时间复杂度为 n+(n−1)+…+1=$O(n^2)$。

参考答案：A

5.2.2　先猜测、后验证

模拟法虽然奏效，但如果考生对整个程序要完成的功能不理解，则会造成求解的速度很慢。如果考生知道了程序的功能，那么对阅读程序的效率的提高很大。所以，在阅读程序中，考生可以借助以前阅读程序的功底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，这就要求学员在平时学习时一方面要及时总结经典或者

常用的功能代码段，另一方面要按照程序设计的经典写法进行书写，以提升自己程序的可读性。

1. 变量的常用含义

搞懂或者猜出变量的含义对于程序的理解至关重要。在变量的定义上，程序员喜欢使用英语单词或者英语单词的缩写、简写等方式表达变量含义，从而逐渐形成了一些固定的使用习惯，例如：sum 表示累加求和的结果，count 表示累加计数等。表 5-4 列举了一些常用的变量使用习惯和含义。

表 5-4　程序中常用的变量及其含义

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

2. 算法的结构

很多常用算法都有一些基本的结构，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，而且在今后编写程序上也能根据算法结构快速地写出程序。这里列举几例，其他结构需要考生在学习过程中整理。

（1）二分算法

```
while(l<=r)
{
    mid=(l+r)/2
    …
    if(…)
```

```
        l=mid+1;
    else
        r=mid-1;
}
```

这里的 l 代表 left，r 代表 right，mid 代表 middle。

(2) 数据链表

```
data[next[i]]
```

这里的 data 数组存储数据域，next 数组存储指针域。

(3) 变量交换位置

```
if(…){
    t =a[i];
    a[i] =a[j];
    a[j] =t;
}
```

当满足某一条件时，交换数组中 a[i]和 a[j]的位置。

(4) 将连续数字字符转换成整数

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

(5) 辗转相除求最大公约数

```
while(b!=0){
    temp=b;
    b=a%b;
    a=temp;
}
return a;
```

返回值 a 就是数值 a 和 b 的最大公约数。

下面我们利用该方法实现一个题目。

案例 5-3　猜测验证法演示(1)

```
1   #include <iostream>
2   using namespace std;
3   const int maxn=10000;
4   int n;
5   int a[maxn];
6   int b[maxn];
7   int f(int l,int r,int depth){
```

```
8     if(l>r)
9         return 0;
10  int min=maxn,mink;
11  for(int i=l;i<=r;++i){
12      if(min>a[i]){
13          min=a[i];
14          mink=i;
15      }
16  }
17  int lres=f(l,mink-1,depth+1);
18  int rres=f(mink+1,r,depth+1);
19  return lres+rres+depth*b[mink];
20  }
21  int main(){
22    cin >>n;
23    for(int i=0;i<n;++i)
24      cin>>a[i];
25    for(int i=0;i<n;++i)
26      cin>>b[i];
27    cout<<f(0,n-1,1)<<endl;
28    return 0;
29  }
```

- 判断题

(1) 如果 a 数组有重复的数字，则程序运行时会发生错误。　　(　　)

(2) 如果 b 数组全为 0，则输出为 0。　　(　　)

- 选择题

(3) 当 n＝100 时，最坏情况下，与第 12 行的比较运算执行的次数最接近的是(　　)。

A. 5000　　B. 600　　C. 6　　D. 100

(4) 当 n＝100 时，最好情况下，与第 12 行的比较运算执行的次数最接近的是(　　)。

A. 100　　B. 6　　C. 5000　　D. 600

(5) 当 n＝10 时，若 b 数组满足：对任意 0≤i＜n 都有 b[i]＝i＋1，那么输出最大为(　　)。

A. 386　　B. 383　　D. 385　　C. 384

(6) 当 n＝100 时，若 b 数组满足：对任意 0≤i＜n 都有 b[i]＝1，那么输出最小为(　　)。

A. 582　　B. 580　　C. 579　　D. 581

【分析】 阅读该题，可以发现里面有函数的递归调用，根据函数 f(int l，int r，int depth)参数中 l、r、depth 这三个单词，很容易联想到二叉树中的左子树 left、右子树 right 和树的深度 depth。由此可大胆推测这是一个有关二叉树的题目。

再根据 min 变量代表的为最小值，可以推算出 mink 为数组 a 中从 l 到 r 位置中最小数值的序号。

最后根据 17、18、19 三行，可以推算出从 l 到 mink－1 构成 lres 为左子树，从 mink＋1 到 r 构成 rres 为右子树，返回值 lres＋rres＋depth＊b[mink]代表一棵根节点最小的二叉树

各节点深度 depth 与对应数组 b 值的加权和。

有了这个分析之后，再根据结果判断题目就会简单很多。下面是对题目的具体解析。

判断题(1)解析：

对于该题，最直接的方法就是采用验证法。构造一个数组 a[]={1,1,1}，代入程序运行一下，会发现程序不会出现错误，而是会优先选择最左侧的那个 1 进行计算。

参考答案： 错误

判断题(2)解析：

解析： 根据返回值的计算公式 lres+rres+depth * b[mink]，这里的 lres 和 rres 分别代表左子树和右子树的值，其计算需要递归计算，真正影响计算结果的是 depth * b[mink]，若数组 b 全为 0，则该项结果必为 0，从而加权和的最后结果显然为 0。

参考答案： 正确

选择题(3)解析：

最坏情况一般都是极端情况，该题很容易发现最坏情况为如下数组：

$$a[]=\{1,2,3,4,5,6,7,\cdots,100\}$$

这种情况下，程序所构造的二叉树的每个节点仅有一个子节点，而程序将递归 100 层。第 12 行的比较运算，第 i 层需要进行 100−i+1 次，所以总执行次数为 100+99+98+…+1≈5000。

参考答案： A

选择题(4)解析：

跟最坏情况相反，所谓的最佳情况就是在构造二叉树时，每一构造都尽可能均分其左右子树，这样就可以保证二叉树的深度最小。假设根节点深度为 1，则含 n=100 个节点的树的深度最小为 logn≈7，对于每一层节点，第 12 行的比较运算，程序总共执行约 n 次，因此总执行次数约为 nlogn≈600。

参考答案： D

选择题(5)解析：

首先计算出数组 b 的值为

$$b[]=\{1,2,3,4,5,6,7,8,9,10\}$$

要使输出的值最大，就要使得 depth * b[mink]最大，也就是程序所构造的二叉树的深度 depth 应尽可能地大，这也就是第 3 题中的最坏情况。定义根节点深度为 1，则含 10 个节点的二叉树的最大深度为 10，因此输出最大值为 1×1+2×2+3×3…+10×10=385。

参考答案： D

选择题(6)解析：

首先计算出数组 b 的值为

$$b[]=\{1,1,1,1,1,1,1,1,1,1\}$$

该题要使输出值最小，就要使得 depth * b[mink]最小，由于数组 b 的值都一样，所以只要使得 depth 最小就行了。而相同节点，完全二叉树的 depth 最小，所以本题要构造一棵完全二叉树，该树中深度为 1 的节点共 1 个，深度为 2 的节点共 2 个，深度为 3 的节点共 4 个……深度为 6 的节点共 32 个，剩余 37 个节点的深度为 7，因此输出最小值为 1×1+2×2+3×4…+6×32+7×37=580。

参考答案： B