

# 编译原理第三次实验测试用例：目录

<b>1 A 组测试用例</b>	<b>2</b>
1.1 A-1 . . . . .	2
1.2 A-2 . . . . .	2
1.3 A-3 . . . . .	3
1.4 A-4 . . . . .	5
1.5 A-5 . . . . .	5
<b>2 B 组测试用例</b>	<b>7</b>
2.1 B-1 . . . . .	7
2.2 B-2 . . . . .	8
2.3 B-3 . . . . .	9
<b>3 C 组测试用例</b>	<b>12</b>
3.1 C-1 . . . . .	12
3.2 C-2 . . . . .	15
<b>4 E 组测试用例</b>	<b>17</b>
4.1 E1-1 . . . . .	17
4.2 E1-2 . . . . .	18
4.3 E1-3 . . . . .	19
4.4 E2-1 . . . . .	21
4.5 E2-2 . . . . .	23
4.6 E2-3 . . . . .	25
<b>5 结束语</b>	<b>27</b>

# 1 A 组测试用例

本组测试用例共 5 个，均为比较简单的程序，简单检查针对赋值/算术语句、分支语句、循环语句、数组表达式和函数调用的翻译。

## 1.1 A-1

输入

```
1 int main() {
2     int answer = 0;
3     int x = 369, y = 258, z = 147;
4     int a = (x + y + z) * (x - y) / (x - z);
5     write(a);
6     a = a + (a * 2) / (z * 3);
7     write(a);
8     answer = x*y + y*z + z*x + a;
9     write(answer);
10    return 0;
11 }
```

程序输入: 无; 预期输出: 387 388 187759

说明：这个测试用例针对赋值与算术语句进行测试。注意，预期输入/输出中每个数字会占一行，这里为了节省空间写在同一行，以空格隔开(下同)。

## 1.2 A-2

输入

```
1 int main() {
2     int a, b, c;
3     int result;
4     a = read();
5     b = read();
6     c = read();
7     result = b * b - 4 * a * c;
8     if(result > 0) {
```

```

9         write(2);
10    }
11    else{
12        if(result == 0) {
13            write(1);
14        }
15        else{
16            write(0);
17        }
18    }
19    return 0;
20 }
```

程序输入: 1 -2 1; 预期输出: 1

程序输入: 2 5 4; 预期输出: 0

程序输入: 1 -7 12; 预期输出: 2

说明: 主要针对分支语句进行测试。

### 1.3 A-3

输入

```

1 int main()
2 {
3     int b[3], c[10];
4     int i = 0, j, t, a[5];
5     while(i < 5)
6     {
7         a[i] = read();
8         i = i + 1;
9     }
10    i = 0;
11    while(i < 4)
12    {
13        j = i + 1;
```

```

14     while(j < 5)
15     {
16         if(a[i] > a[j])
17         {
18             t = a[i];
19             a[i] = a[j];
20             a[j] = t;
21         }
22         j = j + 1;
23     }
24     i = i + 1;
25 }
26 i = 0;
27 while(i < 5)
28 {
29     write(a[i]);
30     i = i + 1;
31 }
32 b = a;
33 c = a;
34 i = 0;
35 while(i < 5) {
36     if(i < 3) {
37         write(b[i]);
38         write(c[i]);
39     }
40     else{
41         write(c[i]);
42     }
43     i = i + 1;
44 }
45 return 0;

```

程序输入: -10 -100 12 33 5; 预期输出: -100 -10 5 12 33 -100 -100 -10 -10 5 5 12 33

说明: 主要测试一维数组。

## 1.4 A-4

输入

```

1 int factorial(int m) {
2     int f_res = 1;
3     while (m > 1) {
4         f_res = f_res * m;
5         m = m - 1;
6     }
7     return f_res;
8 }
9
10 int main() {
11     int n = read();
12     write(factorial(n));
13     return 0;
14 }
```

程序输入: 3; 预期输出: 6

程序输入: 7; 预期输出: 5040

程序输入: 10; 预期输出: 3628800

程序输入: 0; 预期输出: 1

说明: 主要测试循环语句。

## 1.5 A-5

输入

```

1 int main() {
2     int src[3];
3     int dst[3];
```

```
4     int index = 0;
5
6
7     while(index < 3) {
8         src[index] = read();
9         index = index + 1;
10    }
11
12    index = 0;
13
14    while(index < 3) {
15        dst[index] = src[index] * 2;
16        index = index + 1;
17    }
18
19    index = 0;
20
21    while(index < 3) {
22        res = res + src[index] + dst[index];
23        index = index + 1;
24    }
25
26    write(res);
27
28 }
```

程序输入: 1 2 3; 预期输出: 18

程序输入: 3 6 9; 预期输出: 54

说明: 一个测试函数调用的小程序。

## 2 B 组测试用例

本组测试用例共 3 个，较 A 组测试用例复杂，这里不专门针对赋值和算术语句设计测试用例。

### 2.1 B-1

输入

```
1 int factorial(int m) {
2     int f_res = 1;
3     while(m > 1) {
4         f_res = f_res * m;
5         m = m - 1;
6     }
7     return f_res;
8 }
9
10 int cal_combination(int c_base, int c_num) {
11     return factorial(c_base) / (factorial(c_num) * factorial(c_base -
12         c_num));
13 }
14
15 int cal_permutation(int p_base, int p_num) {
16     return factorial(p_base) / factorial(p_base - p_num);
17 }
18
19 int main() {
20     int n = read();
21     int k = read();
22     write(cal_combination(n, k));
23     write(cal_permutation(n, k));
24     return 0;
25 }
```

程序输入: 5 3; 预期输出: 10 60

程序输入: 8 5; 预期输出: 56 6720

说明: 一个计算排列数和组合数的程序。

## 2.2 B-2

输入

```
1 int main() {
2     int num, arr[10], key;
3     int i, res;
4     int left, right;
5     num = read();
6     if(num > 10) {
7         num = 10;
8     }
9
10    i = 0;
11    while(i < num) {
12        arr[i] = read();
13        i = i + 1;
14    }
15
16    key = read();
17
18    left = 0;
19    right = num - 1;
20    while(left < right) {
21        i = (left + right) / 2;
22        if(arr[i] == key) {
23            write(i);
24            return 0;
25        }
26    else {
```

```

27     if(arr[i] > key) {
28         right = i - 1;
29     }
30     else{
31         left = i + 1;
32     }
33 }
34 }
35
36 return 0;
37 }
```

程序输入: 5 -100 -33 0 1 20 10 预期输出: 无输出

程序输入: 5 -123 -45 0 123 456 123 预期输出: 3

说明: 一个二分查找的程序。

### 2.3 B-3

输入

```

1 int main() {
2     int n = 5, arr[5], tmp[5];
3     int i, intv;
4     int s1, e1, curl1, s2, e2, cur2;
5     i = 0;
6     while (i < n) {
7         arr[i] = read();
8         i = i + 1;
9     }
10
11     intv = 1;
12     while (intv < n) {
13         i = 0;
14         while (i <= n - 2 * intv) {
15             s1 = i;
```

```

16     e1 = s1 + intv;
17
18     curl1 = s1;
19
20     s2 = e1;
21
22     e2 = s2 + intv;
23
24     curr2 = s2;
25
26     while (curl1 < e1 && curr2 < e2) {
27
28         if (arr[curl1] < arr[curr2]) {
29
30             tmp[i] = arr[curl1];
31
32             curl1 = curl1 + 1;
33
34             } else {
35
36                 tmp[i] = arr[curr2];
37
38                 curr2 = curr2 + 1;
39
40                 }
41
42
43             i = i + 1;
44
45         }
46
47
48     if (i + intv < n) {
49
50         s1 = i;
51
52         e1 = s1 + intv;
53
54         curl1 = s1;
55
56         s2 = e1;
57
58     }
59
60
61     curl1 = s1;
62
63     curr2 = s2;
64
65     while (curl1 < e1) {
66
67         tmp[i] = arr[curl1];
68
69         curl1 = curl1 + 1;
70
71         i = i + 1;
72
73     }
74
75
76     while (curr2 < e2) {
77
78         tmp[i] = arr[curr2];
79
80         curr2 = curr2 + 1;
81
82         i = i + 1;
83
84     }
85
86
87     i = i + 1;
88
89
90     arr[i] = tmp[i];
91
92     i = i + 1;
93
94
95     curl1 = s1;
96
97     curr2 = s2;
98
99     while (curl1 < e1 && curr2 < e2) {
100
101         tmp[i] = arr[curl1];
102
103         curl1 = curl1 + 1;
104
105         i = i + 1;
106
107     }
108
109
110     curl1 = s1;
111
112     curr2 = s2;
113
114     while (curl1 < e1) {
115
116         tmp[i] = arr[curl1];
117
118         curl1 = curl1 + 1;
119
120         i = i + 1;
121
122     }
123
124
125     while (curr2 < e2) {
126
127         tmp[i] = arr[curr2];
128
129         curr2 = curr2 + 1;
130
131         i = i + 1;
132
133     }
134
135
136     i = i + 1;
137
138
139     arr[i] = tmp[i];
140
141     i = i + 1;
142
143
144     curl1 = s1;
145
146     curr2 = s2;
147
148     while (curl1 < e1 && curr2 < e2) {
149
150         tmp[i] = arr[curl1];
151
152         curl1 = curl1 + 1;
153
154         i = i + 1;
155
156     }
157
158
159     curl1 = s1;
160
161     curr2 = s2;
162
163     while (curl1 < e1) {
164
165         tmp[i] = arr[curl1];
166
167         curl1 = curl1 + 1;
168
169         i = i + 1;
170
171     }
172
173
174     curr2 = s2;
175
176     while (curr2 < e2) {
177
178         tmp[i] = arr[curr2];
179
180         curr2 = curr2 + 1;
181
182         i = i + 1;
183
184     }
185
186
187     i = i + 1;
188
189
190     arr[i] = tmp[i];
191
192     i = i + 1;
193
194
195     curl1 = s1;
196
197     curr2 = s2;
198
199     while (curl1 < e1 && curr2 < e2) {
200
201         tmp[i] = arr[curl1];
202
203         curl1 = curl1 + 1;
204
205         i = i + 1;
206
207     }
208
209
210     curl1 = s1;
211
212     curr2 = s2;
213
214     while (curl1 < e1) {
215
216         tmp[i] = arr[curl1];
217
218         curl1 = curl1 + 1;
219
220         i = i + 1;
221
222     }
223
224
225     curr2 = s2;
226
227     while (curr2 < e2) {
228
229         tmp[i] = arr[curr2];
230
231         curr2 = curr2 + 1;
232
233         i = i + 1;
234
235     }
236
237
238     i = i + 1;
239
240
241     arr[i] = tmp[i];
242
243     i = i + 1;
244
245
246     curl1 = s1;
247
248     curr2 = s2;
249
250     while (curl1 < e1 && curr2 < e2) {
251
252         tmp[i] = arr[curl1];
253
254         curl1 = curl1 + 1;
255
256         i = i + 1;
257
258     }
259
260
261     curl1 = s1;
262
263     curr2 = s2;
264
265     while (curl1 < e1) {
266
267         tmp[i] = arr[curl1];
268
269         curl1 = curl1 + 1;
270
271         i = i + 1;
272
273     }
274
275
276     curr2 = s2;
277
278     while (curr2 < e2) {
279
280         tmp[i] = arr[curr2];
281
282         curr2 = curr2 + 1;
283
284         i = i + 1;
285
286     }
287
288
289     i = i + 1;
290
291
292     arr[i] = tmp[i];
293
294     i = i + 1;
295
296
297     curl1 = s1;
298
299     curr2 = s2;
300
301     while (curl1 < e1 && curr2 < e2) {
302
303         tmp[i] = arr[curl1];
304
305         curl1 = curl1 + 1;
306
307         i = i + 1;
308
309     }
310
311
312     curl1 = s1;
313
314     curr2 = s2;
315
316     while (curl1 < e1) {
317
318         tmp[i] = arr[curl1];
319
320         curl1 = curl1 + 1;
321
322         i = i + 1;
323
324     }
325
326
327     curr2 = s2;
328
329     while (curr2 < e2) {
330
331         tmp[i] = arr[curr2];
332
333         curr2 = curr2 + 1;
334
335         i = i + 1;
336
337     }
338
339
340     i = i + 1;
341
342
343     arr[i] = tmp[i];
344
345     i = i + 1;
346
347
348     curl1 = s1;
349
350     curr2 = s2;
351
352     while (curl1 < e1 && curr2 < e2) {
353
354         tmp[i] = arr[curl1];
355
356         curl1 = curl1 + 1;
357
358         i = i + 1;
359
360     }
361
362
363     curl1 = s1;
364
365     curr2 = s2;
366
367     while (curl1 < e1) {
368
369         tmp[i] = arr[curl1];
370
371         curl1 = curl1 + 1;
372
373         i = i + 1;
374
375     }
376
377
378     curr2 = s2;
379
380     while (curr2 < e2) {
381
382         tmp[i] = arr[curr2];
383
384         curr2 = curr2 + 1;
385
386         i = i + 1;
387
388     }
389
390
391     i = i + 1;
392
393
394     arr[i] = tmp[i];
395
396     i = i + 1;
397
398
399     curl1 = s1;
400
401     curr2 = s2;
402
403     while (curl1 < e1 && curr2 < e2) {
404
405         tmp[i] = arr[curl1];
406
407         curl1 = curl1 + 1;
408
409         i = i + 1;
410
411     }
412
413
414     curl1 = s1;
415
416     curr2 = s2;
417
418     while (curl1 < e1) {
419
420         tmp[i] = arr[curl1];
421
422         curl1 = curl1 + 1;
423
424         i = i + 1;
425
426     }
427
428
429     curr2 = s2;
430
431     while (curr2 < e2) {
432
433         tmp[i] = arr[curr2];
434
435         curr2 = curr2 + 1;
436
437         i = i + 1;
438
439     }
440
441
442     i = i + 1;
443
444
445     arr[i] = tmp[i];
446
447     i = i + 1;
448
449
450     curl1 = s1;
451
452     curr2 = s2;
453
454     while (curl1 < e1 && curr2 < e2) {
455
456         tmp[i] = arr[curl1];
457
458         curl1 = curl1 + 1;
459
460         i = i + 1;
461
462     }
463
464
465     curl1 = s1;
466
467     curr2 = s2;
468
469     while (curl1 < e1) {
470
471         tmp[i] = arr[curl1];
472
473         curl1 = curl1 + 1;
474
475         i = i + 1;
476
477     }
478
479
480     curr2 = s2;
481
482     while (curr2 < e2) {
483
484         tmp[i] = arr[curr2];
485
486         curr2 = curr2 + 1;
487
488         i = i + 1;
489
490     }
491
492
493     i = i + 1;
494
495
496     arr[i] = tmp[i];
497
498     i = i + 1;
499
500
501     curl1 = s1;
502
503     curr2 = s2;
504
505     while (curl1 < e1 && curr2 < e2) {
506
507         tmp[i] = arr[curl1];
508
509         curl1 = curl1 + 1;
510
511         i = i + 1;
512
513     }
514
515
516     curl1 = s1;
517
518     curr2 = s2;
519
520     while (curl1 < e1) {
521
522         tmp[i] = arr[curl1];
523
524         curl1 = curl1 + 1;
525
526         i = i + 1;
527
528     }
529
530
531     curr2 = s2;
532
533     while (curr2 < e2) {
534
535         tmp[i] = arr[curr2];
536
537         curr2 = curr2 + 1;
538
539         i = i + 1;
540
541     }
542
543
544     i = i + 1;
545
546
547     arr[i] = tmp[i];
548
549     i = i + 1;
550
551
552     curl1 = s1;
553
554     curr2 = s2;
555
556     while (curl1 < e1 && curr2 < e2) {
557
558         tmp[i] = arr[curl1];
559
560         curl1 = curl1 + 1;
561
562         i = i + 1;
563
564     }
565
566
567     curl1 = s1;
568
569     curr2 = s2;
570
571     while (curl1 < e1) {
572
573         tmp[i] = arr[curl1];
574
575         curl1 = curl1 + 1;
576
577         i = i + 1;
578
579     }
580
581
582     curr2 = s2;
583
584     while (curr2 < e2) {
585
586         tmp[i] = arr[curr2];
587
588         curr2 = curr2 + 1;
589
590         i = i + 1;
591
592     }
593
594
595     i = i + 1;
596
597
598     arr[i] = tmp[i];
599
600     i = i + 1;
601
602
603     curl1 = s1;
604
605     curr2 = s2;
606
607     while (curl1 < e1 && curr2 < e2) {
608
609         tmp[i] = arr[curl1];
610
611         curl1 = curl1 + 1;
612
613         i = i + 1;
614
615     }
616
617
618     curl1 = s1;
619
620     curr2 = s2;
621
622     while (curl1 < e1) {
623
624         tmp[i] = arr[curl1];
625
626         curl1 = curl1 + 1;
627
628         i = i + 1;
629
630     }
631
632
633     curr2 = s2;
634
635     while (curr2 < e2) {
636
637         tmp[i] = arr[curr2];
638
639         curr2 = curr2 + 1;
640
641         i = i + 1;
642
643     }
644
645
646     i = i + 1;
647
648
649     arr[i] = tmp[i];
650
651     i = i + 1;
652
653
654     curl1 = s1;
655
656     curr2 = s2;
657
658     while (curl1 < e1 && curr2 < e2) {
659
660         tmp[i] = arr[curl1];
661
662         curl1 = curl1 + 1;
663
664         i = i + 1;
665
666     }
667
668
669     curl1 = s1;
670
671     curr2 = s2;
672
673     while (curl1 < e1) {
674
675         tmp[i] = arr[curl1];
676
677         curl1 = curl1 + 1;
678
679         i = i + 1;
680
681     }
682
683
684     curr2 = s2;
685
686     while (curr2 < e2) {
687
688         tmp[i] = arr[curr2];
689
690         curr2 = curr2 + 1;
691
692         i = i + 1;
693
694     }
695
696
697     i = i + 1;
698
699
700     arr[i] = tmp[i];
701
702     i = i + 1;
703
704
705     curl1 = s1;
706
707     curr2 = s2;
708
709     while (curl1 < e1 && curr2 < e2) {
710
711         tmp[i] = arr[curl1];
712
713         curl1 = curl1 + 1;
714
715         i = i + 1;
716
717     }
718
719
720     curl1 = s1;
721
722     curr2 = s2;
723
724     while (curl1 < e1) {
725
726         tmp[i] = arr[curl1];
727
728         curl1 = curl1 + 1;
729
730         i = i + 1;
731
732     }
733
734
735     curr2 = s2;
736
737     while (curr2 < e2) {
738
739         tmp[i] = arr[curr2];
740
741         curr2 = curr2 + 1;
742
743         i = i + 1;
744
745     }
746
747
748     i = i + 1;
749
750
751     arr[i] = tmp[i];
752
753     i = i + 1;
754
755
756     curl1 = s1;
757
758     curr2 = s2;
759
760     while (curl1 < e1 && curr2 < e2) {
761
762         tmp[i] = arr[curl1];
763
764         curl1 = curl1 + 1;
765
766         i = i + 1;
767
768     }
769
770
771     curl1 = s1;
772
773     curr2 = s2;
774
775     while (curl1 < e1) {
776
777         tmp[i] = arr[curl1];
778
779         curl1 = curl1 + 1;
780
781         i = i + 1;
782
783     }
784
785
786     curr2 = s2;
787
788     while (curr2 < e2) {
789
790         tmp[i] = arr[curr2];
791
792         curr2 = curr2 + 1;
793
794         i = i + 1;
795
796     }
797
798
799     i = i + 1;
800
801
802     arr[i] = tmp[i];
803
804     i = i + 1;
805
806
807     curl1 = s1;
808
809     curr2 = s2;
810
811     while (curl1 < e1 && curr2 < e2) {
812
813         tmp[i] = arr[curl1];
814
815         curl1 = curl1 + 1;
816
817         i = i + 1;
818
819     }
820
821
822     curl1 = s1;
823
824     curr2 = s2;
825
826     while (curl1 < e1) {
827
828         tmp[i] = arr[curl1];
829
830         curl1 = curl1 + 1;
831
832         i = i + 1;
833
834     }
835
836
837     curr2 = s2;
838
839     while (curr2 < e2) {
840
841         tmp[i] = arr[curr2];
842
843         curr2 = curr2 + 1;
844
845         i = i + 1;
846
847     }
848
849
850     i = i + 1;
851
852
853     arr[i] = tmp[i];
854
855     i = i + 1;
856
857
858     curl1 = s1;
859
860     curr2 = s2;
861
862     while (curl1 < e1 && curr2 < e2) {
863
864         tmp[i] = arr[curl1];
865
866         curl1 = curl1 + 1;
867
868         i = i + 1;
869
870     }
871
872
873     curl1 = s1;
874
875     curr2 = s2;
876
877     while (curl1 < e1) {
878
879         tmp[i] = arr[curl1];
880
881         curl1 = curl1 + 1;
882
883         i = i + 1;
884
885     }
886
887
888     curr2 = s2;
889
890     while (curr2 < e2) {
891
892         tmp[i] = arr[curr2];
893
894         curr2 = curr2 + 1;
895
896         i = i + 1;
897
898     }
899
900
901     i = i + 1;
902
903
904     arr[i] = tmp[i];
905
906     i = i + 1;
907
908
909     curl1 = s1;
910
911     curr2 = s2;
912
913     while (curl1 < e1 && curr2 < e2) {
914
915         tmp[i] = arr[curl1];
916
917         curl1 = curl1 + 1;
918
919         i = i + 1;
920
921     }
922
923
924     curl1 = s1;
925
926     curr2 = s2;
927
928     while (curl1 < e1) {
929
930         tmp[i] = arr[curl1];
931
932         curl1 = curl1 + 1;
933
934         i = i + 1;
935
936     }
937
938
939     curr2 = s2;
940
941     while (curr2 < e2) {
942
943         tmp[i] = arr[curr2];
944
945         curr2 = curr2 + 1;
946
947         i = i + 1;
948
949     }
950
951
952     i = i + 1;
953
954
955     arr[i] = tmp[i];
956
957     i = i + 1;
958
959
960     curl1 = s1;
961
962     curr2 = s2;
963
964     while (curl1 < e1 && curr2 < e2) {
965
966         tmp[i] = arr[curl1];
967
968         curl1 = curl1 + 1;
969
970         i = i + 1;
971
972     }
973
974
975     curl1 = s1;
976
977     curr2 = s2;
978
979     while (curl1 < e1) {
980
981         tmp[i] = arr[curl1];
982
983         curl1 = curl1 + 1;
984
985         i = i + 1;
986
987     }
988
989
990     curr2 = s2;
991
992     while (curr2 < e2) {
993
994         tmp[i] = arr[curr2];
995
996         curr2 = curr2 + 1;
997
998         i = i + 1;
999
1000    }
1001
1002
1003    i = i + 1;
1004
1005
1006    arr[i] = tmp[i];
1007
1008    i = i + 1;
1009
1010
1011    curl1 = s1;
1012
1013    curr2 = s2;
1014
1015    while (curl1 < e1 && curr2 < e2) {
1016
1017        tmp[i] = arr[curl1];
1018
1019        curl1 = curl1 + 1;
1020
1021        i = i + 1;
1022
1023    }
1024
1025
1026    curl1 = s1;
1027
1028    curr2 = s2;
1029
1030    while (curl1 < e1) {
1031
1032        tmp[i] = arr[curl1];
1033
1034        curl1 = curl1 + 1;
1035
1036        i = i + 1;
1037
1038    }
1039
1040
1041    curr2 = s2;
1042
1043    while (curr2 < e2) {
1044
1045        tmp[i] = arr[curr2];
1046
1047        curr2 = curr2 + 1;
1048
1049        i = i + 1;
1050
1051    }
1052
1053
1054    i = i + 1;
1055
1056
1057    arr[i] = tmp[i];
1058
1059    i = i + 1;
1060
1061
1062    curl1 = s1;
1063
1064    curr2 = s2;
1065
1066    while (curl1 < e1 && curr2 < e2) {
1067
1068        tmp[i] = arr[curl1];
1069
1070        curl1 = curl1 + 1;
1071
1072        i = i + 1;
1073
1074    }
1075
1076
1077    curl1 = s1;
1078
1079    curr2 = s2;
1080
1081    while (curl1 < e1) {
1082
1083        tmp[i] = arr[curl1];
1084
1085        curl1 = curl1 + 1;
1086
1087        i = i + 1;
1088
1089    }
1090
1091
1092    curr2 = s2;
1093
1094    while (curr2 < e2) {
1095
1096        tmp[i] = arr[curr2];
1097
1098        curr2 = curr2 + 1;
1099
1100        i = i + 1;
1101
1102    }
1103
1104
1105    i = i + 1;
1106
1107
1108    arr[i] = tmp[i];
1109
1110    i = i + 1;
1111
1112
1113    curl1 = s1;
1114
1115    curr2 = s2;
1116
1117    while (curl1 < e1 && curr2 < e2) {
1118
1119        tmp[i] = arr[curl1];
1120
1121        curl1 = curl1 + 1;
1122
1123        i = i + 1;
1124
1125    }
1126
1127
1128    curl1 = s1;
1129
1130    curr2 = s2;
1131
1132    while (curl1 < e1) {
1133
1134        tmp[i] = arr[curl1];
1135
1136        curl1 = curl1 + 1;
1137
1138        i = i + 1;
1139
1140    }
1141
1142
1143    curr2 = s2;
1144
1145    while (curr2 < e2) {
1146
1147        tmp[i] = arr[curr2];
1148
1149        curr2 = curr2 + 1;
1150
1151        i = i + 1;
1152
1153    }
1154
1155
1156    i = i + 1;
1157
1158
1159    arr[i] = tmp[i];
1160
1161    i = i + 1;
1162
1163
1164    curl1 = s1;
1165
1166    curr2 = s2;
1167
1168    while (curl1 < e1 && curr2 < e2) {
1169
1170        tmp[i] = arr[curl1];
1171
1172        curl1 = curl1 + 1;
1173
1174        i = i + 1;
1175
1176    }
1177
1178
1179    curl1 = s1;
1180
1181    curr2 = s2;
1182
1183    while (curl1 < e1) {
1184
1185        tmp[i] = arr[curl1];
1186
1187        curl1 = curl1 + 1;
1188
1189        i = i + 1;
1190
1191    }
1192
1193
1194    curr2 = s2;
1195
1196    while (curr2 < e2) {
1197
1198        tmp[i] = arr[curr2];
1199
1200        curr2 = curr2 + 1;
1201
1202        i = i + 1;
1203
1204    }
1205
1206
1207    i = i + 1;
1208
1209
1210    arr[i] = tmp[i];
1211
1212    i = i + 1;
1213
1214
1215    curl1 = s1;
1216
1217    curr2 = s2;
1218
1219    while (curl1 < e1 && curr2 < e2) {
1220
1221        tmp[i] = arr[curl1];
1222
1223        curl1 = curl1 + 1;
1224
1225        i = i + 1;
1226
1227    }
1228
1229
1230    curl1 = s1;
1231
1232    curr2 = s2;
1233
1234    while (curl1 < e1) {
1235
1236        tmp[i] = arr[curl1];
1237
1238        curl1 = curl1 + 1;
1239
1240        i = i + 1;
1241
1242    }
1243
1244
1245    curr2 = s2;
1246
1247    while (curr2 < e2) {
1248
1249        tmp[i] = arr[curr2];
1250
1251        curr2 = curr2 + 1;
1252
1253        i = i + 1;
1254
1255    }
1256
1257
1258    i = i + 1;
1259
1260
1261    arr[i] = tmp[i];
1262
1263    i = i + 1;
1264
1265
1266    curl1 = s1;
1267
1268    curr2 = s2;
1269
1270    while (curl1 < e1 && curr2 < e2) {
1271
1272        tmp[i] = arr[curl1];
1273
1274        curl1 = curl1 + 1;
1275
1276        i = i + 1;
1277
1278    }
1279
1280
1281    curl1 = s1;
1282
1283    curr2 = s2;
1284
1285    while (curl1 < e1) {
1286
1287        tmp[i] = arr[curl1];
1288
1289        curl1 = curl1 + 1;
1290
1291        i = i + 1;
1292
1293    }
1294
1295
1296    curr2 = s2;
1297
1298    while (curr2 < e2) {
1299
1300        tmp[i] = arr[curr2];
1301
1302        curr2 = curr2 + 1;
1303
1304        i = i + 1;
1305
1306    }
1307
1308
1309    i = i + 1;
1310
1311
1312    arr[i] = tmp[i];
1313
1314    i = i + 1;
1315
1316
1317    curl1 = s1;
1318
1319    curr2 = s2;
1320
1321    while (curl1 < e1 && curr2 < e2) {
1322
1323        tmp[i] = arr[curl1];
1324
1325        curl1 = curl1 + 1;
1326
1327        i = i + 1;
1328
1329    }
1330
1331
1332    curl1 = s1;
1333
1334    curr2 = s2;
1335
1336    while (curl1 < e1) {
1337
1338        tmp[i] = arr[curl1];
1339
1340        curl1 = curl1 + 1;
1341
1342        i = i + 1;
1343
1344    }
1345
1346
1347    curr2 = s2;
1348
1349    while (curr2 < e2) {
1350
1351        tmp[i] = arr[curr2];
1352
1353        curr2 = curr2 + 1;
1354
1355        i = i + 1;
1356
1357    }
1358
1359
1360    i = i + 1;
1361
1362
1363    arr[i] = tmp[i];
1364
1365    i = i + 1;
1366
1367
1368    curl1 = s1;
1369
1370    curr2 = s2;
1371
1372    while (curl1 < e1 && curr2 < e2) {
1373
1374        tmp[i] = arr[curl1];
1375
1376        curl1 = curl1 + 1;
1377
1378        i = i + 1;
1379
1380    }
1381
1382
1383    curl1 = s1;
1384
1385    curr2 = s2;
1386
1387    while (curl1 < e1) {
1388
1389        tmp[i] = arr[curl1];
1390
1391        curl1 = curl1 + 1;
1392
1393        i = i + 1;
1394
1395    }
1396
1397
1398    curr2 = s2;
1399
1400    while (curr2 < e2) {
1401
1402        tmp[i] = arr[curr2];
1403
1404        curr2 = curr2 + 1;
1405
1406        i = i + 1;
1407
1408    }
1409
1410
1411    i
```

```

48     e2 = n;
49
50     cur2 = s2;
51
52     while (curl < e1 && cur2 < e2) {
53
54         if (arr[curl] < arr[cur2]) {
55
56             tmp[i] = arr[curl];
57
58             curl = curl + 1;
59
60             i = i + 1;
61
62         } else {
63
64             tmp[i] = arr[cur2];
65
66             cur2 = cur2 + 1;
67
68             i = i + 1;
69
70         }
71
72     } else {
73
74         while (i < n) {
75
76             tmp[i] = arr[i];
77
78             i = i + 1;
79
80         }
81
82     }
83
84     i = 0;
85
86     while (i < n) {
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

80         arr[i] = tmp[i];
81
82     i = i + 1;
83
84 }
85
86 i = 0;
87 while (i < n) {
88     write(arr[i]);
89     i = i + 1;
90 }
91 return 0;
92 }
```

程序输入: 5 4 3 2 1 预期输出: 1 2 3 4 5

程序输入: 10 -3 29 100 2 预期输出: -3 2 10 29 100

说明: 非递归版本的归并排序。

### 3 C 组测试用例

本组测试用例共 2 个, 是较经典的问题。

#### 3.1 C-1

输入

```

1 int factorial(int m) {
2
3     int f_res = 1;
4
5     while (m > 1) {
6
7         f_res = f_res * m;
8
9         m = m - 1;
10    }
11
12    return f_res;
13 }
```

```

10  int cal_combination(int c_base, int c_num) {
11      return factorial(c_base) / (factorial(c_num) * factorial(c_base -
12          c_num));
13
14  int cal_permutation(int p_base, int p_num) {
15      return factorial(p_base) / factorial(p_base - p_num);
16  }
17
18  int isqrt(int n) {
19      int i = 0;
20      while (i < n) {
21          if (i * i <= n && (i + 1) * (i + 1) > n) {
22              return i;
23          }
24          i = i + 1;
25      }
26      return -1;
27  }
28
29  int mod(int k1, int k2) {
30      if (k1 < 0 || k2 <= 0) {
31          return -1;
32      } else {
33          return k1 - k1 / k2 * k2;
34      }
35  }
36
37  int is_prime(int l) {
38      int j = 2;
39      int end = isqrt(l);
40      while (j <= end) {

```

```

41         if (mod(l, j) == 0) {
42             return 0;
43         }
44         j = j + 1;
45     }
46     return 1;
47 }
48
49
50 int main() {
51     int base = read();
52     int key = read();
53     int com = cal_combination(base, key);
54     int per = cal_permutation(base, key);
55     int index = 1;
56     while(index <= com) {
57         if(is_prime(index)) {
58             write(index);
59         }
60         index = index + 1;
61     }
62
63     index = 1;
64     while(index <= per) {
65         if(is_prime(index)) {
66             write(index);
67         }
68         index = index + 1;
69     }
70
71     return 0;
72 }
```

程序输入: 3 2 预期输出: 1 2 3 1 2 3 5

程序输入: 5 3 预期输出: 1 2 3 5 7 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59

说明: 计算排列数和组合数并判断 1 排列数或组合数之间哪些是素数。

### 3.2 C-2

输入

```
1 int mod(int x,int y)
2 {
3     return x -(x / y) * y;
4 }
5
6 int gcd(int c, int d)
7 {
8     if(c==0)
9         return d;
10    return gcd(mod(d, c),c);
11 }
12
13 int lcm(int e, int f)
14 {
15     return e * f / (gcd(e, f));
16 }
17
18 int main()
19 {
20     int n, i, g;
21     int tmp, sum = 0;
22     int a[50];
23     int b[50];
24     n = read();
25     i = 0;
26     while(i < n)
```

```

27     {
28         a[i] = read();
29         b[i] = read();
30         i = i + 1;
31     }
32     tmp = b[0];
33     i = 1;
34     while(i < n)
35     {
36         tmp = lcm(tmp, b[i]);
37         i = i + 1;
38     }
39     i = 0;
40     while(i < n)
41     {
42         sum = sum + a[i] * (tmp / b[i]);
43         i = i + 1;
44     }
45     g = gcd(sum, tmp);
46     sum = sum / g;
47     tmp = tmp / g;
48     if(tmp == 1)
49         write(sum);
50     else
51     {
52         write(sum);
53         write(tmp);
54     }
55     return 0;
56 }
```

程序输入: 5 2 5 4 15 1 30 2 60 8 3; 预期输出: 17 5

说明: 分式相加, 输入是分式的个数以及每个分式的分子和分母, 输出是结果的最简分式

的分子和分母。

## 4 E 组测试用例

本组测试用例共 6 个，针对不同分组进行测试。

E1 组针对 3.1 分组测试结构体的翻译，E2 组针对 3.2 分组测试一维数组作为参数和高维数组的翻译。每组 3 个测试用例。

### 4.1 E1-1

输入

```
1 struct Rectangle {
2     int r_length;
3     int r_width;
4     int r_area;
5 };
6
7 struct Triangle {
8     int t_width;
9     int t_height;
10    int t_area;
11 };
12
13 int main() {
14     struct Rectangle r;
15     struct Triangle t;
16     r.r_length = 100;
17     r.r_width = 25;
18     t.t_height = 36;
19     t.t_width = 24;
20     r.r_area = r.r_length * r.r_width;
21     t.t_area = t.t_height * t.t_width / 2;
22     write(r.r_area);
23     write(t.t_area);
```

```
24     return 0;  
25 }
```

程序输入: 无; 预期输出: 2500 432

说明: 测试对于简单结构体的翻译, 不涉及与数组的交互和结构体作为函数参数调用。针对 3.1 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

## 4.2 E1-2

输入

```
1 struct Good {  
2     int price;  
3     int number;  
4 };  
5  
6 int main(){  
7     struct Good goods[10];  
8     int cnt = 0;  
9     int sum = 0;  
10    while(cnt < 10){  
11        goods[cnt].price = cnt + 101;  
12        goods[cnt].number = cnt + 1;  
13        cnt = cnt + 1;  
14    }  
15  
16    cnt = 0;  
17    while(cnt < 10){  
18        sum = sum + goods[cnt].price * goods[cnt].number;  
19        cnt = cnt + 1;  
20    }  
21    write(sum);  
22    return 0;  
23 }
```

程序输入: 无; 预期输出: 5885

说明: 针对 3.1 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

### 4.3 E1-3

输入

```
1  struct Car {  
2      int loc_x;  
3      int loc_y;  
4      int speed;  
5  };  
6  
7  struct Street {  
8      int distanceSum;  
9      int speedAverage;  
10     struct Car cars[3];  
11  };  
12  
13  
14  int distance(struct Car car1, struct Car car2){  
15      int dis_x = 0;  
16      int dis_y = 0;  
17      if(car1.loc_x > car2.loc_x){  
18          dis_x = car1.loc_x - car2.loc_x;  
19      }  
20      else{  
21          dis_x = car2.loc_x - car1.loc_x;  
22      }  
23  
24      if(car1.loc_y > car2.loc_y){  
25          dis_y = car1.loc_y - car2.loc_y;  
26      }  
27      else{
```

```

28         dis_y = car2.loc_y - car1.loc_y;
29     }
30
31     return dis_x + dis_y;
32
33
34     int sum_distance(struct Street street2) {
35
36         int sum = 0;
37
38         sum = sum + distance(street2.cars[0], street2.cars[1]);
39         sum = sum + distance(street2.cars[1], street2.cars[2]);
40         sum = sum + distance(street2.cars[2], street2.cars[0]);
41
42         return sum;
43
44
45     int ave_speed(struct Car car3, struct Car car4, struct Car car5) {
46
47         return (car3.speed + car4.speed + car5.speed) / 3;
48
49
50     int initStreet(struct Street street1) {
51
52         int index = 0;
53
54         street1.distanceSum = 0;
55
56         street1.speedAverage = 0;
57
58         while(index < 3) {
59
60             street1.cars[index].loc_x = 30 * index + 10;
61
62             street1.cars[index].loc_y = 20 * (index + 1) - 15;
63
64             street1.cars[index].speed = (index + 1) * 100 - 50;
65
66             index = index + 1;
67
68         }
69
70         return 0;
71
72     }
73
74
75     int main() {
76
77         struct Street myStreet;

```

```

60     initStreet(myStreet);
61     write(sum_distance(myStreet));
62     write(ave_speed(myStreet.cars[0], myStreet.cars[1], myStreet.cars
63         [2]));
64     return 0;
}

```

程序输入: 无; 预期输出: 200 150

说明: 测试对于较复杂的结构体及其作为函数参数进行函数的调用。针对 3.1 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

#### 4.4 E2-1

```

1  int main() {
2      int mat[4][4];
3      int i, j, k;
4      i = 0;
5      while(i < 4) {
6          j = 0;
7          while(j < 4) {
8              mat[i][j] = 999;
9              j = j + 1;
10         }
11         i = i + 1;
12     }
13
14     i = 0;
15     while(i < 4) {
16         mat[i][i] = 0;
17         i = i + 1;
18     }
19     mat[0][1] = 3;
20     mat[0][3] = 5;
21     mat[1][0] = 2;
}

```

```

22     mat[1][3] = 4;
23     mat[2][1] = 1;
24     mat[3][2] = 2;
25
26     k = 0;
27     while(k < 4) {
28         i = 0;
29         while(i < 4) {
30             j = 0;
31             while(j < 4) {
32                 if(mat[i][k] + mat[k][j] < mat[i][j]) {
33                     mat[i][j] = mat[i][k] + mat[k][j];
34                 }
35                 j = j + 1;
36             }
37             i = i + 1;
38         }
39         k = k + 1;
40     }
41
42     write(mat[0][3]);
43     write(mat[1][2]);
44     write(mat[2][1]);
45     write(mat[3][0]);
46
47     return 0;
48 }
```

程序输入: 无; 预期输出: 5 6 1 5

说明: Floyd Washer 多源最短路算法, 测试对于简单高维数组的翻译, 不涉及数组作为函数参数。针对 3.2 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

## 4.5 E2-2

输入

```
1
2 int initW(int w[6]) {
3     w[0] = 0;
4     w[1] = 1;
5     w[2] = 2;
6     w[3] = 5;
7     w[4] = 6;
8     w[5] = 7;
9     return 0;
10 }
11
12 int initV(int v[6]) {
13     v[0] = 0;
14     v[1] = 1;
15     v[2] = 6;
16     v[3] = 18;
17     v[4] = 22;
18     v[5] = 28;
19     return 0;
20 }
21
22 int main() {
23     int weight[6];
24     int value[6];
25     int resutls[6][12];
26
27     int r_i;
28     int r_j;
29     int ki, kj;
```

```

31     initW(weight);
32     initV(value);
33
34     r_i = 0;
35     while(r_i < 6) {
36         r_j = 0;
37         while(r_j < 12) {
38             resutls[r_i][r_j] = 0;
39             r_j = r_j + 1;
40         }
41         r_i = r_i + 1;
42     }
43
44     ki = 1;
45     while(ki < 6) {
46         kj = 1;
47         while(kj < 12) {
48             if(kj < weight[ki]) {
49                 resutls[ki][kj] = resutls[ki-1][kj];
50             }
51             else{
52                 if(resutls[ki-1][kj] > value[ki] + resutls[ki-1][kj - weight
53 [ki]]) {
54                     resutls[ki][kj] = resutls[ki-1][kj];
55                 }
56                 else{
57                     resutls[ki][kj] = value[ki] + resutls[ki-1][kj-weight[ki
58 ]];
59                 }
60             }
61             kj = kj + 1;
62         }
63     }

```

```

61     ki = ki + 1;
62 }
63
64 write(resutls[5][11]);
65 return 0;
66 }
```

程序输入: 无预期输出: 40

说明: 01 背包问题, 测试对于数组作为函数参数的翻译。针对 3.2 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

## 4.6 E2-3

输入

```

1 int Swap(int a[8], int l, int h)
2 {
3     int temp;
4     temp = a[l];
5     a[l] = a[h];
6     a[h] = temp;
7     return 0;
8 }
9
10 int Partition(int b[8], int low, int high)
11 {
12     int base = b[low];
13     while (low < high)
14     {
15         while (low < high && b[high] >= base)
16         {
17             high = high - 1;
18         }
19         Swap(b, low, high);
20         while (low < high && b[low] <= base)
```

```

21     {
22         low = low + 1;
23     }
24     Swap(b, low, high);
25 }
26 return low;
27 }
28
29 int QuickSort(int c[8], int low1, int high1)
30 {
31     if(low1 < high1)
32     {
33         int base1 = Partition(c, low1, high1);
34         QuickSort(c, low1, base1 - 1);
35         QuickSort(c, base1 + 1, high1);
36     }
37     return 0;
38 }
39
40 int main()
41 {
42     int n = 8;
43     int arr[8];
44     int i = 0;
45     while(i < n)
46     {
47         arr[i] = read();
48         i = i + 1;
49     }
50     QuickSort(arr, 0, n-1);
51     i = 0;
52     while(i < n)

```

```
53     {
54         write(arr[i]);
55         i = i + 1;
56     }
57     return 0;
58 }
```

程序输入: 23 5 19 23 6 6 2 35; 预期输出: 2 5 6 6 19 23 23 35

说明: 快速排序, 测试对于较复杂的数组操作的翻译, 针对 3.2 分组, 其他分组同学需要提示无法翻译且不输出中间代码。

## 5 结束语

如果对本测试用例有任何疑议, 可以写邮件与[张灵毓](#)助教联系, 注意同时抄送给[许老师](#)。