

编译原理第三次实验测试用例：目录

1	A 组测试用例	2
1.1	A-1	2
1.2	A-2	2
1.3	A-3	3
1.4	A-4	5
1.5	A-5	6
2	B 组测试用例	7
2.1	B-1	7
2.2	B-2	8
2.3	B-3	9
3	C 组测试用例	11
3.1	C-1	11
3.2	C-2	13
4	D 组测试用例	15
4.1	D-1	16
5	E 组测试用例	18
5.1	E1-1	18
5.2	E1-2	18
5.3	E1-3	20
5.4	E2-1	21
5.5	E2-2	22
5.6	E2-3	23
6	结束语	26

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-算数语句、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 a, b, c;  
3     int t = 12;  
4     int pp = 4, tt = 5;  
5     a = t * pp;  
6     b = a + tt - pp;  
7     c = a + b * pp;  
8     write(c);  
9     c = c * (a + b) / 25;  
10    b = (b + c) - a / c + pp * 12;  
11    write(c);  
12    write(b);  
13    return 0;  
14 }
```

程序输入：无；预期输出：244 946 1043

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 price;  
3     int money;  
4     price = read();  
5     money = read();
```

```

6      if (price > money) {
7          if (price > money * 2) {
8              write(price);
9          } else {
10             write(money * 2);
11         }
12     } else if (price == money) {
13         if (money > 100) {
14             write(100);
15         } else {
16             write(money);
17         }
18     } else {
19         if (price * 2 < money) {
20             write(money);
21         } else {
22             write(price * 2);
23         }
24     }
25     return 0;
26 }

```

输入：50 20；输出：50

输入：50 26；输出：52

输入：50 50；输出：50

输入：1000 1000；输出：100

输入：16 70；输出：70

输入：16 17；输出：32

说明：这个测试用例主要针对分支语句进行测试的小程序。注意，程序输入以空格隔开，每次输入一个数（下同）。

1.3 A-3

输入

```

1  int main(){
2      int a, b;
3      int result;
4      int start;
5      int found = 0;
6      a = read();
7      b = read();
8      if (a > b) {
9          start = a;
10     } else {
11         start = b;
12     }
13     while(found == 0) {
14         if (start == (start / a) * a) {
15             if (start == (start / b) * b) {
16                 result = start;
17                 found = 1;
18             } else {
19                 start = start + 1;
20             }
21         } else {
22             start = start + 1;
23         }
24     }
25     write(result);
26     return 0;
27 }

```

输入: 5 5; 输出: 4

输入: 5 6; 输出: 30

输入: 6 8; 输出: 24

输入: 1 25; 输出: 25

说明：这个测试用例主要针对循环语句进行测试，求 a 和 b 的最小公倍数。

1.4 A-4

输入

```
1 int main() {
2     int i[5], t, k, changed, tem;
3     t = 0;
4     while(t < 5) {
5         i[t] = read();
6         t = t + 1;
7     }
8     changed = 1;
9     while(changed == 1) {
10        changed = 0;
11        t = 1;
12        while(t < 5) {
13            k = t;
14
15            while(k > 0 && i[k] < i[k-1]){
16                tem = i[k];
17                i[k] = i[k-1];
18                i[k-1] = tem;
19                changed = 1;
20                k = k - 1;
21            }
22            t = t + 1;
23
24        }
25    }
26    t = 0;
27    while (t < 5) {
28        write(i[t]);
```

```

29         t = t + 1;
30     }
31     return 0;
32 }

```

输入：35 25 12 14 12；输出：12 12 14 25 35

说明：这个测试用例主要针对一维数组进行测试，实现升序冒泡排序。

1.5 A-5

输入

```

1  int compare(int a, int b) {
2      if (a > b) return a;
3      if (a < b) return b;
4      return 0;
5  }
6
7  int add(int aa, int bb) {
8      return aa + bb;
9  }
10
11 int main() {
12     int i[10], r[5], n, m;
13     n = 0;
14     while (n < 10) {
15         i[n] = read();
16         n = n + 1;
17     }
18     n = 0;
19     m = 0;
20     while (n < 10) {
21         r[m] = compare(i[n], i[n+1]);
22         n = n + 2;
23         m = m + 1;

```

```

24     }
25     n = 0;
26     while (n < 5) {
27         r[n] = add(r[n], i[n]);
28         write(r[n]);
29         n = n + 1;
30     }
31     return 0;
32 }

```

输入：1 2 3 4 5 5 4 3 2 1；输出：3 6 3 8 7

说明：这个测试用例主要针对函数的调用进行简单测试。

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 hanoi(int count, int pole1, int pole2, int pole3) {
2     if (count == 1){
3         write(pole1*1000000000+pole3);
4     } else {
5         hanoi(count-1,pole1, pole3, pole2);
6         write(pole1*1000000000+pole3);
7         hanoi(count-1,pole2, pole1, pole3);
8     }
9     return 0;
10 }
11
12 int main() {
13     int n;

```

```

14         n = read();
15         hanoi(n, 1, 2, 3);
16         return 0;
17     }

```

输入：3；输出：

```

1000000003
3000000002
1000000003
2000000001
2000000003
1000000003

```

说明：Hanoi 塔，考察复杂的函数调用和递归；每行两端表示将左端编号的环移到后侧编号。

2.2 B-2

输入

```

1  int main() {
2      int N = 10;
3      int heap[10];
4      int i = 0, start, j, tem1, k, tem2;
5      int stop = 0;
6      while (i < 10) {
7          heap[i] = read();
8          i = i + 1;
9      }
10     start = N / 2;
11     while(start >= 0) {
12         stop = 0;
13         i = start;
14         while (stop == 0) {
15             stop = 1;
16             j = i * 2 + 1;

```



```

17         k = i * 2 + 2;
18         if (j < N) {
19             tem1 = heap[j];
20             if (k < N && heap[k] < heap[j]) {
21                 tem1 = heap[k];
22                 j = k;
23             }
24             if (heap[i] > tem1) {
25                 stop = 0;
26                 heap[j] = heap[i];
27                 heap[i] = tem1;
28                 i = j;
29             }
30         }
31     }
32     start = start - 1;
33 }
34 i = 0;
35 while (i < N) {
36     write(heap[i]);
37     i = i + 1;
38 }
39 return 0;
40 }

```

输入: 35 4 78 96 35 1 247 89 50 12; 输出: 1 4 35 50 12 78 247 89 96 35

说明: 建立一个小根堆。

2.3 B-3

输入

```

1 int isPrime(int number) {
2     int max = number / 4;
3     int i = 2;

```

```

4         while(i < max) {
5             if (number == number / i * i) {
6                 return 0;
7             }
8             i = i + 1;
9         }
10        return 1;
11    }
12
13    int isRever(int num) {
14        int n = num;
15        int array[10];
16        int bit = 0, j = 0;
17        while (n != 0) {
18            array[bit] = n - n / 10 * 10;
19            n = n / 10;
20            bit = bit + 1;
21        }
22        bit = bit - 1;
23        while (j != bit) {
24            if (array[j] != array[bit]){
25                return 0;
26            }
27            j = j + 1;
28            bit = bit -1;
29        }
30        return 1;
31    }
32
33    int main(){
34        int N = 100, M = 110;
35        int ii = N;

```

```

36     while (ii < M) {
37         if (isPrime(ii) == 1) {
38             write(ii);
39         }
40         if (isRever(ii) == 1) {
41             write(-ii);
42         }
43         ii = ii + 1;
44     }
45     return 0;
46 }

```

输入：null；输出：101 -101 103 107 109

说明：找到 100 和 110 之间的所有素数（正数输出）和回文数（负数输出）。

3 C 组测试用例

本组测试用例共 2 个，是经典问题。

3.1 C-1

输入

```

1  int main() {
2      int N = 10;
3      int heap[10], result[10];
4      int i = 0, start, j, tem1, k, tem2, t;
5      int stop = 0;
6      while (i < 10) {
7          heap[i] = read();
8          i = i + 1;
9      }
10     start = N / 2;
11     while(start >= 0) {
12         stop = 0;

```

```

13         i = start;
14         while (stop == 0) {
15             stop = 1;
16             j = i * 2 + 1;
17             k = i * 2 + 2;
18             if (j < N) {
19                 tem1 = heap[j];
20                 if (k < N && heap[k] < heap[j]) {
21                     tem1 = heap[k];
22                     j = k;
23                 }
24                 if (heap[i] > tem1) {
25                     stop = 0;
26                     heap[j] = heap[i];
27                     heap[i] = tem1;
28                     i = j;
29                 }
30             }
31         }
32         start = start - 1;
33     }
34     start = 10;
35     i = 0;
36     while (i < N) {
37         result[i] = heap[0];
38         i = i + 1;
39         heap[0] = heap[start-1];
40         stop = 0;
41         start = start - 1;
42         t = 0;
43         while (stop == 0) {
44             stop = 1;

```

```

45         j = t * 2 + 1;
46         k = t * 2 + 2;
47         if (j < start){
48             tem1 = heap[j];
49             if (k < start && heap[k] < heap[j]) {
50                 tem1 = heap[k];
51                 j = k;
52             }
53             if (heap[t] > tem1) {
54                 stop = 0;
55                 heap[j] = heap[t];
56                 heap[t] = tem1;
57                 t = j;
58             }
59         }
60     }
61 }
62 i = 0;
63 while (i < N) {
64     write(result[i]);
65     i = i + 1;
66 }
67 return 0;
68 }

```

输入: 5 6 7 9 10 5 3 4 2 6; 输出: 2 3 4 5 5 6 6 7 9 10

说明: 堆排序。

3.2 C-2

输入

```

1 int mod(int number2, int m2) {
2     int result = number2 - number2 / m2 * m2;
3     int result2 = result;

```

```

4         return result;
5     }
6
7     int power(int base1, int p1) {
8         int ret1 = 1 + p1 - p1;
9         while(p1 > (ret1 - ret1 + 90 - 89 + 1 - 2)) {
10             ret1 = ret1 * base1;
11             p1 = 2 * 1 * p1 - 1 * p1 - 1;
12         }
13         return ret1;
14     }
15
16     int getNumDigits(int number3) {
17         int ret3 = 0;
18         if(number3 < 0) {
19             return -1;
20         }
21         while(number3 > 0) {
22             number3 = number3 / 10;
23             ret3 = ret3 + 2;
24             ret3 = ret3 + 2;
25             ret3 = ret3 - 3;
26         }
27
28         return ret3;
29     }
30
31     int isNarcissistic(int number4) {
32         int numDigits4 = getNumDigits(1 + number4 - 1);
33         int sum4 = 0;
34         int n4 = number4;
35         int s4;

```

```

36     while(n4>0) {
37         s4 = mod(n4, 10);
38         n4 = (n4 - s4) / 10;
39         sum4 = sum4 + power(s4, numDigits4);
40     }
41     if(sum4 == number4) {
42         return 1;
43     } else {
44         return 0;
45     }
46 }
47
48 int main() {
49     int count = 0;
50     int i = 300;
51     while(i < 500) {
52         if(isNarcissistic(i) == 1) {
53             write(i);
54             count = count + 1;
55         }
56         i = i + 1;
57     }
58     write(count);
59     return count;
60 }

```

输入：无；输出：370 371 407 3

说明：找到 300 和 500 之间的所有水仙数，并输出个数。

4 D 组测试用例

本组测试用例共 1 个，主要用于测试中间代码的优化。

4.1 D-1

输入

```
1 int process(int x) {
2     int y = 3;
3     y = 11 * 3 - 2 + 5;
4     y = x * 321 * 2 + x * y - x + y * x + y * y + x + x - 23 +
        45;
5     y = y / 3 + 14 * 24 - x * 12 / 4 - 20 * 3 + y / 12 * 24 + 12
        * 3 + 3 / 2;
6     y = x + 4 * 6 + 3 / 2;
7     return y;
8 }
9
10 int main () {
11     int a = 5 / 2 + 14 - 3, b = 7 * 5 / 2 + 3, c = 4 + 5 + 6 - 1 /
        2;
12     int d = a + b + c;
13     int e = a * b + c / 2;
14     int f = a - b - c;
15     int g1 = 42, i = 0;
16     int g, h;
17     f = a + b + c + 1000 * 2 - f;
18     while (a + b < f) {
19         g1 = g1 + i * 12 + 4 + 5 + 7 / 3;
20         g = process(f) + 2 * a - f + c * d;
21         i = i + i;
22         i = i + i;
23         i = i + i;
24         i = i + i;
25         i = i + i;
26         h = i + 3;
27         h = h - 1;
```



```

28         h = h + 3;
29         h = h - 3 * 2;
30         if (process(a) == process(a + 3 - 2 -1)) {
31             f = f - 2 + 1;
32         }
33         a = a + 2 + 1;
34     }
35     h = g1 - 3 * 4;
36     while (h < g1) {
37         f = 15 * 4 - 2 + a;
38         g = g1 -12;
39         h = h + 1;
40         g = g1;
41         i = a + b;
42         c = a + b;
43     }
44     write(f);
45     a = a + b;
46     b = a + b;
47     c = a + b;
48     f = a + b;
49     g = a + b;
50     write(c+f+g);
51     return 0;
52
53 }

```

输入：无；输出：1601 9438

说明：程序中有多个可优化点，包括常量折叠，公共子表达式等。首先需要保证中间代码的正确性，要能准确输出最后的结果，才能参加后面的效率竞赛。

5 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。

5.1 E1-1

输入

```
1 struct Food{
2     int name;
3     int price;
4 };
5
6 int main() {
7     struct Food burger;
8     struct Food cola;
9     burger.name = 1;
10    burger.price = 2;
11    cola.name = 3;
12    cola.price = 4;
13    write(cola.name + burger.price);
14    return 0;
15 }
```

输入：无；输出：5

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。

5.2 E1-2

输入

```
1 struct Product{
2     int type;
3     int name;
```

```

4 };
5
6 struct Cola{
7     int price;
8     int sold;
9 };
10
11 int main(){
12     struct Product things[5];
13     struct Cola cc[5];
14     int i = 0, j =12, result = 0;
15     while (i < 5) {
16         things[i].type = j;
17         things[i].name = j * j;
18         j = j - 1;
19         cc[i].price = j;
20         cc[i].sold = j / 3;
21         i = i + 1;
22     }
23     i = 0;
24     while (i < 5) {
25         result = result + things[i].type * cc[4-i].sold;
26         i = i + 1;
27     }
28     write(result);
29     return 0;
30 }

```

输入：无；输出：127

说明：测试对于结构体作为数组的类型。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。

5.3 E1-3

输入

```
1 struct Student{
2     int name;
3     int grade;
4 };
5
6 struct Class{
7     struct Student students[10];
8     int average;
9 };
10
11 int isStudentInClass(struct Class ccc, struct Student stu) {
12     int ii = 0;
13     while (ii < 10) {
14         if (ccc.students[ii].name == stu.name) {
15             return ccc.students[ii].grade;
16         }
17         ii = ii + 1;
18     }
19     return 0;
20 }
21
22 int main() {
23     struct Class cl;
24     int i = 0, result;
25     struct Student ss;
26     ss.name = read();
27     ss.grade = 0;
28     cl.average = 0;
29     while (i < 10) {
30         cl.students[i].name = i;
```

```

31         cl.students[i].grade = i * i;
32         i = i + 1;
33     }
34     write(isStudentInClass(cl,ss));
35     return 0;
36 }

```

输入：3；输出：9

输入：12；输出：0

输入：5；输出：25

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。

5.4 E2-1

输入

```

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

```

```

19         }
20         i = i + 1;
21     }
22     i = 0;
23     while (i < 10) {
24         write(t[i]);
25         i = i + 1;
26     }
27     return 0;
28 }

```

输入：无；输出：45 55 65 75 85 95 105 115 125

说明：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。

5.5 E2-2

输入

```

1  int print(int array[4]) {
2      int ii = 0;
3      while (ii < 4) {
4          write(array[ii]);
5          ii = ii + 1;
6      }
7      return 0;
8  }
9
10 int product(int a[4], int b[4]) {
11     int result = 0;
12     int i = 0;
13     while (i < 4) {
14         result = result + a[i] * b[i];
15         i = i + 1;
16     }

```

```

17         return result;
18     }
19
20 int main() {
21     int p[4], pp[4];
22     int j = 0;
23     while (j < 4) {
24         p[j] = read();
25         pp[j] = read();
26         j = j + 1;
27     }
28     j = 0;
29     print(p);
30     write(product(p,pp));
31     return 0;
32 }

```

输入：1 2 3 4 5 6 7 8；输出：1 3 5 7 100

说明：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。

5.6 E2-3

输入

```

1 int display(int chess[10], int number[1], int sum){
2     int board[10][10], i1 = 0, j1 = 0, tem = 1;
3     if (number[0] == 1) {
4         while (i1 < sum) {
5             j1 = 0;
6             tem = 1;
7             while (j1 < sum) {
8                 if (j1 == chess[i1]) {
9                     board[i1][j1] = 1;
10                    tem = tem * 10 + 1;

```

```

11         } else {
12             board[i1][j1] = 0;
13             tem = tem * 10;
14         }
15         j1 = j1 + 1;
16     }
17     write(tem);
18     i1 = i1 + 1;
19 }
20 }
21 return 0;
22 }
23
24 int dfs(int p[10], int r[10], int ld[10], int rd[10], int current,
25         int target, int c[1]){
26     int j = 0, nld[10], nrd[10], k;
27     if (current == target) {
28         c[0] = c[0] + 1;
29         display(p,c,target);
30         return 0;
31     }
32     while (j < target) {
33         if (r[j] == 1 && ld[j] == 1 && rd[j] == 1 ) {
34             p[current] = j;
35             r[j] = 0;
36             k = 0;
37             while (k< target - 1){
38                 nld[k] = ld[k + 1];
39                 k = k + 1;
40             }
41             nld[target -1] = 1;
42             if (j != 0) {

```



```

42         nld[j - 1] = 0;
43     }
44     k = target-1;
45     while (k > 0){
46         nrd[k] = rd[k-1];
47         k = k - 1;
48     }
49     nrd[0] = 1;
50     if (j != target -1){
51         nrd[j + 1] = 0;
52     }
53     dfs(p, r, nld, nrd, current + 1, target, c);
54     r[j] = 1;
55 }
56 j = j + 1;
57 }
58 return 0;
59 }
60 int main() {
61     int place[10], N, count[1];
62     int row[10], ldiag[10], rdiag[10] ,i = 0;
63     N = read();
64     if (N == 0 || N > 10) { return 0;}
65     while(i < N) {
66         row[i] = 1;
67         ldiag[i] = 1;
68         rdiag[i] = 1;
69         i = i + 1;
70     }
71     count[0] = 0;
72     dfs(place,row,ldiag,rdiag,0,N,count);
73     write(count[0]);

```

```
74     return 0;
75 }
```

输入：8；输出：110000000 100001000 100000001 100000100 100100000 100000010 101000000
100010000 92

说明：测试对于较复杂的数组操作的翻译，是一个八皇后问题，输出第一个搜索到的摆放方案（每行 1 开头，之后八位代表摆放，1 代表放置皇后），并输出总共的解法数目。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。

6 结束语

如果对本测试用例有任何疑议，可以写邮件与王珏助教联系，注意同时抄送给许老师，本学期编译原理实验到此结束，祝愿大家都能取得好的成绩。