编译原理第三次实验测试用例：目录

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

```
int main(){
        int a, b, c;
        int t = 12;
        int pp = 4, tt = 5;
        a = t * pp;
        b = a + tt - pp;
        c = a + b * pp;
        write(c);
        c = c * (a + b) / 25;
        b = (b + c) - a / c + pp * 12;
        write(c);
        write(b);
        return 0;
}
```

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

```
int main() {
        int price;
        int money;
        price = read();
        money = read();
```

```
        if (price > money) {
                if (price > money * 2) {
                        write(price);
                } else {
                        write(money * 2);
                }
        } else if (price == money) {
                if (money > 100) {
                        write(100);
                } else {
                        write(money);
                }
        } else {
                if (price * 2 < money) {
                        write(money);
                } else {
                        write(price * 2);
                }
        }
        return 0;
}
```

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

```
int main(){
        int a, b;
        int result;
        int start;
        int found = 0;
        a = read();
        b = read();
        if (a > b) {
                start = a;
        } else {
                start = b;
        }
        while(found == 0) {
                if (start == (start / a) * a) {
                        if (start == (start / b) * b) {
                                result = start;
                                found = 1;
                        } else {
                                start = start + 1;
                        }
                } else {
                        start = start + 1;
                }
        }
        write(result);
        return 0;
}
```

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

```
int main() {
        int i[5], t, k, changed, tem;
        t = 0;
        while(t < 5) {
                i[t] = read();
                t = t + 1;
        }
        changed = 1;
        while(changed == 1) {
                changed = 0;
                t = 1;
                while(t < 5) {
                        k = t;

                        while(k > 0 && i[k] < i[k-1]){
                                tem = i[k];
                                i[k] = i[k-1];
                                i[k-1] = tem;
                                changed = 1;
                                k = k - 1;
                        }
                        t = t + 1;

                }
        }
        t = 0;
        while (t < 5) {
                write(i[t]);
```

```
                t = t + 1;
        }
        return 0;
}
```

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

```
int compare(int a, int b) {
        if (a > b) return a;
        if (a < b) return b;
        return 0;
}

int add(int aa, int bb) {
        return aa + bb;
}

int main() {
        int i[10], r[5], n, m;
        n = 0;
        while (n < 10) {
                i[n] = read();
                n = n + 1;
        }
        n = 0;
        m = 0;
        while(n < 10) {
                r[m] = compare(i[n], i[n+1]);
                n = n + 2;
                m = m + 1;
```

```
        }
        n = 0;
        while (n < 5) {
                r[n] = add(r[n], i[n]);
                write(r[n]);
                n = n + 1;
        }
        return 0;
}
```

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

```
int hanoi(int count, int pole1, int pole2, int pole3) {
        if (count == 1){
                write(pole1*1000000000+pole3);
        } else {
                hanoi(count-1,pole1, pole3, pole2);
                write(pole1*1000000000+pole3);
                hanoi(count-1,pole2, pole1, pole3);
        }
        return 0;
}

int main() {
        int n;
```

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

输入：3；输出：

1000000003

1000000002

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;
```

```
                        k = i * 2 + 2;
                        if (j < N) {
                                tem1 = heap[j];
                                if (k < N && heap[k] < heap[j]) {
                                        tem1 = heap[k];
                                        j = k;
                                }
                                if (heap[i] > tem1) {
                                        stop = 0;
                                        heap[j] = heap[i];
                                        heap[i] = tem1;
                                        i = j;
                                }
                        }
                }
                start = start - 1;
        }
        i = 0;
        while (i < N) {
                write(heap[i]);
                i = i + 1;
        }
        return 0;
}
```

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

```
int isPrime(int number) {
        int max = number / 4;
        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;
```

```
        while (ii < M) {
                if (isPrime(ii) == 1) {
                        write(ii);
                }
                if (isRever(ii) == 1) {
                        write(-ii);
                }
                ii = ii + 1;
        }
        return 0;
}
```

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

```
int main() {
        int N = 10;
        int heap[10], result[10];
        int i = 0, start, j, tem1, k, tem2, t;
        int stop = 0;
        while (i < 10) {
                heap[i] = read();
                i = i + 1;
        }
        start = N / 2;
        while(start >= 0) {
                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;
```

```
                        j = t * 2 + 1;
                        k = t * 2 + 2;
                        if (j < start){
                                tem1 = heap[j];
                                if (k < start && heap[k] < heap[j]) {
                                        tem1 = heap[k];
                                        j = k;
                                }
                                if (heap[t] > tem1) {
                                        stop = 0;
                                        heap[j] = heap[t];
                                        heap[t] = tem1;
                                        t = j;
                                }
                        }
                }
        }
        i = 0;
        while (i < N) {
                write(result[i]);
                i = i + 1;
                }
        return 0;
}
```

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

```
int mod(int number2, int m2) {
        int result = number2 - number2 / m2 * m2;
        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;
```

```
        while(n4>0) {
                s4 = mod(n4, 10);
                n4 = (n4 - s4) / 10;
                sum4 = sum4 + power(s4, numDigits4);
        }
        if(sum4 == number4) {
                return 1;
        } else {
                return 0;
        }
}

int main() {
        int count = 0;
        int i = 300;
        while(i < 500) {
                if(isNarcissistic(i) == 1) {
                        write(i);
                        count = count + 1;
                }
                i = i + 1;
        }
        write(count);
        return count;
}
```

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

```
int process(int x) {
        int y = 3;
        y = 11 * 3 - 2 + 5;
        y = x * 321 * 2 + x * y - x + y * x + y * y + x + x - 23 +
           45;
        y = y / 3 + 14 * 24 - x * 12 / 4  - 20 * 3 + y / 12 *24 + 12
           * 3 + 3 / 2;
        y = x + 4 * 6 + 3 / 2;
        return y;
}

int main () {
        int a = 5 / 2 + 14 -3, b = 7 * 5 / 2 + 3, c = 4 + 5 + 6 - 1 /
            2;
        int d = a + b + c;
        int e = a * b + c / 2;
        int f = a - b - c;
        int g1 = 42, i = 0;
        int g, h;
        f = a + b + c + 1000 * 2 - f;
        while (a + b < f) {
                g1 = g1 + i * 12 + 4 +5 +7 / 3;
                g = process(f) + 2 * a  - f + c * d;
                i = i + i;
                i = i + i;
                i = i + i;
                i = i + i;
                i = i + i;
                h = i + 3;
                h = h - 1;
```

```
                h = h + 3;
                h = h - 3 * 2;
                if (process(a) == process(a + 3 - 2 -1)) {
                        f = f - 2 + 1;
                }
                a = a + 2 + 1;
        }
        h = g1 - 3 * 4;
        while (h < g1) {
                f = 15 * 4 - 2 + a;
                g = g1 -12;
                h = h + 1;
                g = g1;
                i = a + b;
                c = a + b;
        }
        write(f);
        a = a + b;
        b = a + b;
        c = a + b;
        f = a + b;
        g = a + b;
        write(c+f+g);
        return 0;

}
```

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

```
struct Food{
        int name;
        int price;
};

int main() {
        struct Food burger;
        struct Food cola;
        burger.name = 1;
        burger.price = 2;
        cola.name = 3;
        cola.price = 4;
        write(cola.name + burger.price);
        return 0;
}
```

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

```
struct Product{
        int  type[10];
        int name[10];
```

```
};

struct Cola{
        int price[10];
        int sold[10];
};

int main(){
        struct Product things;
        struct Cola cc;
        int i = 0, j =12, result = 0;
        while (i < 5) {
                things.type[i] = j;
                things.name[i] = j * j;
                j = j - 1;
                cc.price[i] = j;
                cc.sold[i] = j / 3;
                i = i + 1;
        }
        i = 0;
        while (i < 5) {
                result = result + things.type[i] * cc.sold[4-i];
                i = i + 1;
        }
        write(result);
        return 0;
}
```

输入：无；输出：127

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。

5.3 E1-3

输入

```
struct Student{
        int name[10];
        int grade[10];
};

struct Class{
        struct Student students;
        int average;
};

int isStudentInClass(struct Class ccc, struct Student stu) {
        int ii = 0;
        while (ii < 10) {
                if (ccc.students.name[ii] == stu.name[0]) {
                        return ccc.students.grade[ii];
                }
                ii = ii + 1;
        }
        return 0;
}

int main() {
        struct Class cl;
        int i = 0, result;
        struct Student ss;
        ss.name[0] = read();
        ss.grade[0] = 0;
        cl.average = 0;
        while (i < 10) {
                cl.students.name[i] = i;
```

```
                cl.students.grade[i] = i * i;
                i = i + 1;
        }
        write(isStudentInClass(cl,ss));
        return 0;
}
```

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

```
int main() {
        int map[10][10];
        int t[10],i = 0, j = 0;
        while (i < 10){
                j = 0;
                while(j < 10) {
                        map[i][j] = i + j;
                        j = j + 1;
                }
                i = i + 1;
        }
        i = 0;
        while (i < 10) {
                j = 0;
                t[i] = 0;
                while (j < 10)  {
                        t[i] = t[i] + map[i][j];
                        j = j + 1;
```

```
                }
                i = i + 1;
        }
        i = 0;
        while (i < 10) {
                write(t[i]);
                i = i + 1;
        }
        return 0;
}
```

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

```
int print(int array[4]) {
        int ii = 0;
        while (ii < 4) {
                write(array[ii]);
                ii = ii + 1;
        }
        return 0;
}

int product(int a[4], int b[4]) {
        int result = 0;
        int i = 0;
        while (i < 4) {
                result = result + a[i] * b[i];
                i = i  +1;
        }
```

```
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;
```

```
                                } else {
                                        board[i1][j1] = 0;
                                        tem = tem * 10;
                                }
                                j1 = j1 + 1;
                        }
                        write(tem);
                        i1 = i1 + 1;
                }
        }
        return 0;
}

int dfs(int p[10], int r[10], int ld[10], int rd[10], int current,
   int target, int c[1]){
        int j = 0, nld[10], nrd[10], k;
        if (current == target) {
                c[0] = c[0] + 1;
                display(p,c,target);
                return 0;
        }
        while (j < target) {
                if (r[j] == 1 && ld[j] == 1 && rd[j] == 1 ) {
                        p[current] = j;
                        r[j] = 0;
                        k = 0;
                        while (k< target - 1){
                                nld[k] = ld[k + 1];
                                k = k + 1;
                        }
                        nld[target -1] = 1;
                        if (j != 0) {
```

```
                                nld[j - 1] = 0;
                        }
                        k = target-1;
                        while (k > 0){
                                nrd[k] = rd[k-1];
                                k = k - 1;
                        }
                        nrd[0] = 1;
                        if (j != target -1){
                                nrd[j + 1] = 0;
                        }
                        dfs(p, r, nld, nrd, current + 1, target, c);
                        r[j] = 1;
                }
                j =  j + 1;
        }
        return 0;
}
int main() {
        int place[10], N, count[1];
        int row[10], ldiag[10], rdiag[10] ,i = 0;
        N = read();
        if (N == 0 || N > 10) { return 0;}
        while(i < N) {
                row[i] = 1;
                ldiag[i] = 1;
                rdiag[i] = 1;
                i = i + 1;
        }
        count[0] = 0;
        dfs(place,row,ldiag,rdiag,0,N,count);
        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。