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

```
int main(){
  int answer = 0;
  int x = 369, y = 258, z = 147;
  int a = (x + y + z) * (x - y) / (x - z);
  write(a);
  a = a + (a * 2) / (z * 3);
  write(a);
  answer = x*y + y*z + z*x + a;
  write(answer);
  return 0;
}
```

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

```
int main(){
  int a, b, c;
  int result;
  a = read();
  b = read();
  c = read();
  result = b * b - 4 * a * c;
  if(result > 0){
```

```
      write(2);
    }
    else{
      if(result == 0){
        write(1);
      }
      else{
        write(0);
      }
    }
    return 0;
  }
```

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

```
  int main()
{
    int b[3], c[10];
    int i = 0, j, t, a[5];
    while(i < 5)
    {
        a[i] = read();
        i = i + 1;
    }
    i = 0;
    while(i < 4)
    {
        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;
```

```
46 }
```

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  iint 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];
```

```
    int index = 0;
    int res = 0;

    while(index < 3){
      src[index] = read();
      index = index + 1;
    }

    index = 0;
    while(index < 3){
      dst[index] = src[index] * 2;
      index = index + 1;
    }

    index = 0;
    while(index < 3){
      res = res + src[index] + dst[index];
      index = index + 1;
    }

    write(res);

    return 0;

}
```

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

```
int factorial(int m){
  int f_res = 1;
  while(m > 1){
    f_res = f_res * m;
    m = m - 1;
  }
  return f_res;
}

int cal_combiniation(int c_base, int c_num){
  return factorial(c_base) / (factorial(c_num) * factorial(c_base -
      c_num));
}

int cal_permutation(int p_base, int p_num){
  return factorial(p_base) / factorial(p_base - p_num);
}

int main() {
  int n = read();
  int k = read();
  write(cal_combiniation(n, k));
  write(cal_permutation(n, k));
  return 0;
}
```

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

```
int main(){
int num, arr[10], key;
int i, res;
int left, right;
num = read();
if(num > 10){
  num = 10;
}

i = 0;
while(i < num){
  arr[i] = read();
  i = i + 1;
}

key = read();

left = 0;
right = num - 1;
while(left < right){
  i = (left + right) / 2;
  if(arr[i] == key){
    write(i);
    return 0;
  }
  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, cur1, 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;
```

```
            e1 = s1 + intv;
            cur1 = s1;
            s2 = e1;
            e2 = s2 + intv;
            cur2 = s2;
            while (cur1 < e1 && cur2 < e2) {
                if (arr[cur1] < arr[cur2]) {
                    tmp[i] = arr[cur1];
                    cur1 = cur1 + 1;
                } else {
                    tmp[i] = arr[cur2];
                    cur2 = cur2 + 1;
                }
                i = i + 1;
            }
            while (cur1 < e1) {
                tmp[i] = arr[cur1];
                cur1 = cur1 + 1;
                i = i + 1;
            }
            while (cur2 < e2) {
                tmp[i] = arr[cur2];
                cur2 = cur2 + 1;
                i = i + 1;
            }
        }

        if (i + intv < n) {
            s1 = i;
            e1 = s1 + intv;
            cur1 = s1;
            s2 = e1;
```

```
48              e2 = n;
49              cur2 = s2;
50              while (cur1 < e1 && cur2 < e2) {
51                  if (arr[cur1] < arr[cur2]) {
52                      tmp[i] = arr[cur1];
53                      cur1 = cur1 + 1;
54                      i = i + 1;
55                  } else {
56                      tmp[i] = arr[cur2];
57                      cur2 = cur2 + 1;
58                      i = i + 1;
59                  }
60              }
61              while (cur1 < e1) {
62                  tmp[i] = arr[cur1];
63                  cur1 = cur1 + 1;
64                  i = i + 1;
65              }
66              while (cur2 < e2) {
67                  tmp[i] = arr[cur2];
68                  cur2 = cur2 + 1;
69                  i = i + 1;
70              }
71          } else {
72              while (i < n) {
73                  tmp[i] = arr[i];
74                  i = i + 1;
75              }
76          }
77
78          i = 0;
79          while (i < n) {
```

```
            arr[i] = tmp[i];
            i = i + 1;
        }
        intv = intv * 2;
    }

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

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

```
int factorial(int m){
  int f_res = 1;
  while(m > 1){
    f_res = f_res * m;
    m = m - 1;
  }
  return f_res;
}

```

```
10  int cal_combiniation(int c_base, int c_num){
11    return factorial(c_base) / (factorial(c_num) * factorial(c_base -
          c_num));
12  }
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_combiniation(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

```
int mod(int x,int y)
{
    return x -(x / y) * y;
}

int gcd(int c, int d)
{
    if(c==0)
        return d;
    return gcd(mod(d, c),c);
}

int lcm(int e, int f)
{
    return e * f / (gcd(e, f));
}

int main()
{
    int n, i, g;
    int tmp, sum = 0;
    int a[50];
    int b[50];
    n = read();
    i = 0;
    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

```
struct Rectangle {
  int r_length;
  int r_width;
  int r_area;
};

struct Triangle {
  int t_width;
  int t_height;
  int t_area;
};

int main() {
  struct Rectangle r;
  struct Triangle t;
  r.r_length = 100;
  r.r_width = 25;
  t.t_height = 36;
  t.t_width = 24;
  r.r_area = r.r_length * r.r_width;
  t.t_area = t.t_height * t.t_width / 2;
  write(r.r_area);
  write(t.t_area);
```

```
    return 0;
}
```

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

```
struct Good {
  int price;
  int number;
};

int main(){
  struct Good goods[10];
  int cnt = 0;
  int sum = 0;
  while(cnt < 10){
    goods[cnt].price = cnt + 101;
    goods[cnt].number = cnt + 1;
    cnt = cnt + 1;
  }

  cnt = 0;
  while(cnt < 10){
    sum = sum + goods[cnt].price * goods[cnt].number;
    cnt = cnt + 1;
  }
  write(sum);
  return 0;
}
```

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

```
struct Car {
  int loc_x;
  int loc_y;
  int speed;
};

struct Street {
  int distanceSum;
  int speedAverage;
  struct Car cars[3];
};


int distance(struct Car car1, struct Car car2){
  int dis_x = 0;
  int dis_y = 0;
  if(car1.loc_x > car2.loc_x){
    dis_x = car1.loc_x - car2.loc_x;
  }
  else{
    dis_x = car2.loc_x - car1.loc_x;
  }

  if(car1.loc_y > car2.loc_y){
    dis_y = car1.loc_y - car2.loc_y;
  }
  else{
```

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

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

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;
```

```
    mat[1][3] = 4;
    mat[2][1] = 1;
    mat[3][2] = 2;

    k = 0;
    while(k < 4){
      i = 0;
      while(i < 4){
        j = 0;
        while(j < 4){
          if(mat[i][k] + mat[k][j] < mat[i][j]){
            mat[i][j] = mat[i][k] + mat[k][j];
          }
          j = j + 1;
        }
        i = i + 1;
      }
      k = k + 1;
    }

    write(mat[0][3]);
    write(mat[1][2]);
    write(mat[2][1]);
    write(mat[3][0]);

    return 0;
  }
```

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;
30
```

```
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
              [ki]]){
53            resutls[ki][kj] = resutls[ki-1][kj];
54          }
55          else{
56            resutls[ki][kj] = value[ki] + resutls[ki-1][kj-weight[ki
                ]];
57          }
58        }
59        kj = kj + 1;
60      }
```

```
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。