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 i = 1, j = 11, k = 39;
    int result = 0;
    i = (-i) + k + (-(17 * i));
    write(i);
    j = 11 * j - (k + i * i) * i;
    write(j);
    k = i * (i / j) + k;
    write(k);
    result = 4 * i + j / 17 + i * k;
    write(result);
    return 0;
}
```

程序输入: 无; 预期输出: 21 -9959 18 -124

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 x1, y1, x2, y2;
    int u1, v1, u2, v2;
    int l1, r1, t1, b1;
    int l2, r2, t2, b2;
    x1 = read();
```

```
    y1 = read();
    x2 = read();
    y2 = read();
    u1 = read();
    v1 = read();
    u2 = read();
    v2 = read();
    if (x1 == x2 || y1 == y2 || u1 == u2 || v1 == v2) {
        write(-1);
    } else {
        if (x1 < x2) {
            l1 = x1;
            r1 = x2;
        } else {
            l1 = x2;
            r1 = x1;
        }
        if (y1 < y2) {
            t1 = y2;
            b1 = y1;
        } else {
            t1 = y1;
            b1 = y2;
        }
        if (u1 < u2) {
            l2 = u1;
            r2 = u2;
        } else {
            l2 = u2;
            r2 = u1;
        }
        if (v1 < v2) {
```

```
            t2 = v2;
            b2 = v1;
        } else {
            t2 = v1;
            b2 = v2;
        }

        if (l2 >= r1 || r2 <= l1 || b2 >= t1 || t2 <= b1) {
            write(0);
        } else {
            write(1);
        }
    }
    return 0;
}
```

程序输入: 0 0 0 1 1 1 2 2; 预期输出: -1

程序输入: 0 0 1 1 1 1 2 2; 预期输出: 0

程序输入: 0 0 3 3 1 1 -1 4; 预期输出: 1

程序输入: 1 1 -2 3 -1 2 -3 0; 预期输出: 1

程序输入: -2 2 1 -1 0 1 2 3; 预期输出: 1

说明：一个输入两个对角顶点的坐标判断两个矩形的公共面积是否大于 0 的小程序，主要针对分支语句进行测试。

1.3 A-3

输入

```
int main() {
    int k;
    int line = 0, cnt = 0;
    int m = 0, n = 0;
    int numtor = 1, denomtor = 1;
    k = read();

```

```
    if (k <= 0) {
        write(-1);
        return 0;
    }

    while (cnt < k) {
        line = line + 1;
        cnt = cnt + line;
    }
    m = k - (cnt - line);
    n = line;
    write(n);
    write(m);

    cnt = 0;
    while (cnt < m) {
        numtor = numtor * n;
        n = n - 1;
        cnt = cnt + 1;
    }
    while(m > 0) {
        denomtor = denomtor * m;
        m = m - 1;
    }
    write(numtor / denomtor);
    return 0;
}
```

程序输入: 0; 预期输出: -1

程序输入: 1; 预期输出: 1 1 1

程序输入: 5; 预期输出: 3 2 3

程序输入: 17; 预期输出: 6 2 15

程序输入: 25; 预期输出: 7 4 35

说明：计算杨辉三角中第 n 个数所在的行，行中的位置，以及数值的程序，主要测试循环语句。

1.4 A-4

输入

```
int main() {
    int catalan[11], n = 11;
    int i, j;
    catalan[0] = 1;
    catalan[1] = 1;
    i = 2;
    while(i < n) {
        catalan[i] = 0;
        i = i + 1;
    }
    i = 2;
    while (i < n) {
        j = 0;
        while (j < i) {
            catalan[i] = catalan[i] + catalan[j] * catalan[i - j -
                1];
            j = j + 1;
        }
        i = i + 1;
    }
    write(catalan[n - 1]);
    return 0;
}
```

程序输入: 无; 预期输出: 16796

说明：计算第 11 个卡塔兰数，主要测试一维数组。

1.5 A-5

输入

```
int add(int ai, int aj) {
    return ai + aj;
}

int sub(int si, int sj) {
    return si - sj;
}

int mul(int mi, int mj) {
    return mi * mj;
}

int main() {
    int i, j, k, l, res;
    i = read();
    j = read();
    k = read();
    l = read();
    res = add(mul(sub(i, j), sub(i, j)), add(k, l));
    write(res);
    return 0;
}
```

程序输入: 0 0 0 0; 预期输出: 0

程序输入: 1 2 3 4; 预期输出: 8

程序输入: 1 19 17 29; 预期输出: 370

程序输入: 39 -1 0 40; 预期输出: 1640

程序输入: 9 11 15 27; 预期输出: 46

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 mod(int i, int j) {
    return i - i / j * j;
}

int quick_power_mod(int x, int y, int k) {
    int res = 1;
    if (x <= 0 || y <= 0 || k <= 0) {
        return -1;
    } else {
        x = mod(x, k);
        while (y != 0) {
            if (mod(y, 2) == 1) {
                res = mod(res * x, k);
            }
            y = y / 2;
            x = mod(x * x, k);
        }
        return res;
    }
}

int main() {
    int input[3], cnt = 0;
    while (cnt < 3) {
        input[cnt] = read();
```

```
26          cnt = cnt + 1;
27      }
28      write(quick_power_mod(input[0], input[1], input[2]));
29      return 0;
30  }
```

程序输入: -1 4 5; 预期输出: -1

程序输入: 8 500 5; 预期输出: 1

说明：一个计算大数幂的模 mod(pow(x, y), k) 的程序。

2.2 B-2

输入

```
1   int main() {
2       int cnum = 3, charges[3];
3       int amount = 100, dp[101];
4       int valid = 1;
5       int i = 0, j = 0;
6       while (i < cnum) {
7           charges[i] = read();
8           if (charges[i] <= 0) {
9               valid = 0;
10          }
11          i = i + 1;
12      }
13      if (valid == 0) {
14          write(-1);
15          return 0;
16      }
17
18      dp[0] = 0;
19      i = 1;
20      while (i < amount + 1) {
21          dp[i] = amount + 1;
```

```
        i = i + 1;
    }

    i = 0;
    while (i < cnum) {
        int chg = charges[i];
        j = chg;
        while (j < amount + 1) {
            if (dp[j - chg] + 1 < dp[j]) {
                dp[j] = dp[j - chg] + 1;
            }
            j = j + 1;
        }
        i = i + 1;
    }

    if (dp[amount] > amount) {
        write(-1);
    } else {
        write(dp[amount]);
    }
    return 0;
}
```

程序输入: 49 99 101; 预期输出: -1

程序输入: 7 2 5; 预期输出: 15

程序输入: 3 11 5; 预期输出: 12

说明：一个用动态规划解目标为 100 的“最少零钱兑换”问题的程序。

2.3 B-3

输入

```
int main() {
    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;
16              e1 = s1 + intv;
17              cur1 = s1;
18              s2 = e1;
19              e2 = s2 + intv;
20              cur2 = s2;
21              while (cur1 < e1 && cur2 < e2) {
22                  if (arr[cur1] < arr[cur2]) {
23                      tmp[i] = arr[cur1];
24                      cur1 = cur1 + 1;
25                  } else {
26                      tmp[i] = arr[cur2];
27                      cur2 = cur2 + 1;
28                  }
29                  i = i + 1;
30              }
31              while (cur1 < e1) {
32                  tmp[i] = arr[cur1];
33                  cur1 = cur1 + 1;
34                  i = i + 1;
```

```
35              }
36              while (cur2 < e2) {
37                  tmp[i] = arr[cur2];
38                  cur2 = cur2 + 1;
39                  i = i + 1;
40              }
41          }
42
43          if (i + intv < n) {
44              s1 = i;
45              e1 = s1 + intv;
46              cur1 = s1;
47              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) {
```

```
                tmp[i] = arr[cur2];
                cur2 = cur2 + 1;
                i = i + 1;
            }
        } else {
            while (i < n) {
                tmp[i] = arr[i];
                i = i + 1;
            }
        }

        i = 0;
        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: 1 5 4 2 3; 预期输出: 1 2 3 4 5

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 fact(int m) {
    if (m <= 0) {
        return 1;
    } else {
        return fact(m - 1) * m;
    }
}

int isqrt(int n) {
    int i = 0;
    while (i < n) {
        if (i * i <= n && (i + 1) * (i + 1) > n) {
            return i;
        }
        i = i + 1;
    }
    return -1;
}

int mod(int k1, int k2) {
    if (k1 < 0 || k2 <= 0) {
        return -1;
    } else {
        return k1 - k1 / k2 * k2;
    }
}

int is_prime(int l) {
    int j = 2;
    int end = isqrt(l);
```

```
    while (j <= end) {
        if (mod(l, j) == 0) {
            return 0;
        }
        j = j + 1;
    }
    return 1;
}

int main() {
    int c = 2;
    int d = read();
    while (c < fact(d)) {
        if (is_prime(c)) {
            write(c);
        }
        c = c + 1;
    }
    return 0;
}
```

程序输入: 0; 预期输出: 无

程序输入: 3; 预期输出: 2 3 5

程序输入: 4; 预期输出: 2 3 5 7 11 13 17 19 23

说明：打印从 1 到输入数字的阶乘中，所有的素数。

3.2 C-2

输入

```
int bit_and(int aop1, int aop2) {
    if (aop1 == 0) {
        return 0;
    } else {
        return aop2;
```

```
    }
}

int bit_or(int oop1, int oop2) {
    if (oop1 == 0) {
        return oop2;
    } else {
        return 1;
    }
}

int bit_not(int nop) {
    if (nop == 0) {
        return 1;
    } else {
        return 0;
    }
}

int mod(int mop1, int mop2) {
   return mop1 - mop1 / mop2 * mop2;
}

int and(int m, int n) {
    int isize = 32;
    int am[32];
    int an[32];
    int res[32];
    int i = 0;
    int mn = 0;
    if (m <= 0 || n <= 0) {
        return 0;
```

```
38      }
39
40      while (i < isize) {
41          am[i] = 0;
42          an[i] = 0;
43          res[i] = 0;
44          i = i + 1;
45      }
46
47      i = 0;
48      while (i < isize) {
49          am[i] = mod(m, 2);
50          an[i] = mod(n, 2);
51          m = m / 2;
52          n = n / 2;
53          i = i + 1;
54      }
55
56      i = 0;
57      while (i < isize) {
58          res[i] = bit_and(am[i], an[i]);
59          i = i + 1;
60      }
61
62      i = isize - 1;
63      while (i >= 0) {
64          mn = mn * 2 + res[i];
65          i = i - 1;
66      }
67      return mn;
68  }
69
```

```
int main() {
    int x = read();
    int y = read();
    write(and(x, y));
    return 0;
}
```

程序输入: 0 1; 预期输出: 0

程序输入: 3 10; 预期输出: 2

程序输入: 100 111; 预期输出: 100

说明：计算输入两个整数的逻辑与。

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 fact(int i1) {
    if (i1 <= 0) {
        return 1;
    } else {
        return i1 * fact(i1 - 1);
    }
}

int isqrt(int i2) {
    int c1 = 0;
    while (c1 < i2) {
        if (c1 * c1 <= i2 && (c1 + 1) * (c1 + 1) > i2) {
            return c1;
        }
        c1 = c1 + 1;
```

```
    }
    return -1;
}

int mod(int i3, int i4) {
    return i3 - i3 / i4 * i4;
}

int main() {
    int a = 1331;
    int b = 1217;
    int c = -22121;
    int d = 5;
    int i = b * 7 / a + (1990 + 9 * 10) / (b + 23);
    int j = (2000 - 1) * 10  / (b + 2 * 10 + 3);
    int k = 0;
    int l = 0;
    int m = 0;
    int arr[1000];
    while (k < fact(isqrt(isqrt(b)))) {
        arr[k] = fact(mod(k, 4));
        a = k + k / 4 * 4;
        a = a + k / 4 * 4;
        a = a + k / 4 * 4;
        a = a + k / 4 * 4;
        while (c < d * d * d) {
            c = mod(a, 10) + 10 + c + 1 + i - j;
        }
        c = c + fact(mod(isqrt(c), 10));
        k = k + 1;
    }

```

```
48      k = 0;
49      while(k < 10) {
50          l = 0;
51          while (l < 10) {
52              if (k == 0 && l == 0) {
53                  m = 1;
54              } else {
55                  m = 0;
56              }
57              while (m < 10) {
58                  d = d + arr[k * 10 * 10 + l * 10 + m] - arr[k * 10 *
                        10 + l * 10 + m - 1];
59                  m = m + 1;
60              }
61              l = l + 1;
62          }
63          k = k + 1;
64      }
65      k = d + c;
66      write(k);
67      return 0;
68  }
```

程序输入: 无; 预期输出: 363410

说明:

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 Giant {
    int id;
    int age;
    int height;
    int weight;
};

int main() {
    struct Giant g;
    int bmi = 0;
    g.id = 0;
    g.age = 20;
    g.height = 2;
    g.weight = 90;
    bmi = g.weight / (g.height * g.height);
    write(bmi);
    return 0;
}
```

程序输入: 无; 预期输出: 22

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 Giant {
    int id;
    int age;
    int height;
    int weight;
```

```
6       int bmi;
7   };
8
9   int cal_bmi(struct Giant g) {
10      g.bmi = g.weight / (g.height * g.height);
11      return 0;
12  }
13
14  int main() {
15      int i = 0;
16      int n = 10;
17      struct Giant giants[10];
18      while (i < n) {
19          giants[i].id = i;
20          giants[i].age = 20 + i;
21          giants[i].height = 2 + i;
22          giants[i].weight = 90 + i * i * i * i;
23          i = i + 1;
24      }
25
26      i = 0;
27      while (i < n) {
28          cal_bmi(giants[i]);
29          i = i + 1;
30      }
31
32      i = 0;
33      while (i < n) {
34          write(giants[i].bmi);
35          i = i + 1;
36      }
37      return 0;
```

```
}
```

程序输入: 无; 预期输出: 22 10 6 6 9 14 21 30 41 54

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 Giant {
    int id;
    int age;
    int height;
    int weight;
    int bmi;
};

struct Family {
    struct Giant giants[5];
    int avg_bmi;
};

int cal_bmi(struct Giant g) {
    g.bmi = g.weight / (g.height * g.height);
    return 0;
}

int cal_avg_bmi(struct Family f) {
    int c = 0;
    int sum = 0;
    while (c < 5) {
        cal_bmi(f.giants[c]);
        sum = sum + f.giants[c].bmi;
        c = c + 1;
    }
```

```
    f.avg_bmi = sum / 5;
    return 0;
}

int main() {
    int i, j;
    struct Family gf[10];
    i = 0;
    while (i < 10) {
        j = 0;
        while (j < 5) {
            gf[i].giants[j].id = i / 2 + j;
            gf[i].giants[j].age = i / 2 + 20 + j;
            gf[i].giants[j].height = i / 2 + 2 + j;
            gf[i].giants[j].weight = i / 2 + 90 + j * j * j * j;
            j = j + 1;
        }
        cal_avg_bmi(gf[i]);
        i = i + 1;
    }

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

程序输入: 无; 预期输出: 10 10 6 6 3 3 2 2 1 1

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

```
int main() {
    int i;
    int j;
    int n = 5;
    int m[5][5];
    int rs[5][5];
    int sum;

    i = 0;
    while (i < n) {
        j = 0;
        while (j < n) {
            m[i][j] = i * i + j;
            j = j + 1;
        }
        i = i + 1;
    }

    i = 0;
    while (i < n) {
        j = 0;
        while (j < n) {
            m[j][i] = m[i][j];
            j = j + 1;
        }
        i = i + 1;
    }

    sum = 0;
    i = 0;
    while (i < n) {
```

```
32          sum = sum + m[0][i];
33          i = i + 1;
34      }
35      write(sum);
36      return 0;
37  }
```

程序输入: 无; 预期输出: 10

说明：矩阵的转置。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 dot_product(int u[3], int v[3]) {
2       return u[0] * v[0] + u[1] * v[1] + u[2] * v[2];
3   }
4
5   int cross_product(int x[3], int y[3], int z[3]) {
6       z[0] = x[1] * y[2] - x[2] * y[1];
7       z[1] = x[2] * y[0] - x[0] * y[2];
8       z[2] = x[0] * y[1] - x[1] * y[0];
9       return 0;
10  }
11
12  int main() {
13      int i;
14      int j;
15      int ops[2][3];
16      int res[3];
17      i = 0;
18      while (i < 2) {
19          j = 0;
20          while (j < 3) {
```

```
            ops[i][j] = (i + 1) * (i + 1) + (j + 1) * (j + 1);
            j = j + 1;
        }
        i = i + 1;
    }
    cross_product(ops[0], ops[1], res);
    write(dot_product(res, res));
    return 0;
}
```

程序输入: 无; 预期输出: 882

说明：三维向量的叉积和点积。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

```
int merge(int a[6], int l, int m, int r)
{
    int la[10], ra[10];
    int i, j, k;
    int n1 = m - (l - 1);
    int n2 = r - m;

    i = 0;
    while (i < n1) {
        la[i] = a[l + i];
        i = i + 1;
    }
    j = 0;
    while (j < n2) {
        ra[j] = a[m + 1 + j];
        j = j + 1;
    }
```

```
18
19      i = 0;
20      j = 0;
21      k = l;
22
23      while (i < n1 && j < n2) {
24          if (la[i] <= ra[j]) {
25              a[k] = la[i];
26              i = i + 1;
27          }
28          else {
29              a[k] = ra[j];
30              j = j + 1;
31          }
32          k = k + 1;
33      }
34
35      while (i < n1) {
36          a[k] = la[i];
37          i = i + 1;
38          k = k + 1;
39      }
40
41      while (j < n2) {
42          a[k] = ra[j];
43          j = j + 1;
44          k = k + 1;
45      }
46      return 0;
47  }
48
49  int merge_sort(int arr[6], int start, int end){
```

```
50      int mid = 0;
51      if (start >= end) {
52          return 0;
53      }
54      mid = start + (end - start) / 2;
55      merge_sort(arr, start, mid);
56      merge_sort(arr, mid + 1, end);
57      merge(arr, start, mid, end);
58      return 0;
59  }
60
61  int main()
62  {
63      int n = 6;
64      int array[6];
65      int c = 0;
66      while (c < n) {
67          array[c] = read();
68          c = c + 1;
69      }
70
71      merge_sort(array, 0, n - 1);
72
73      c = 0;
74      while (c < n) {
75          write(array[c]);
76          c = c + 1;
77      }
78      return 0;
79  }
```

程序输入: 4 -1 32 2 10 111; 预期输出: -1 2 4 10 32 111

说明：递归版本的归并排序。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。

6 结束语

如果对本测试用例有任何疑议，可以写邮件与李聪助教或屈道涵助教联系，注意同时抄送给许老师。