实验 3 测试用例

实验 3 测试用例包括必做内容测试用例、分组内容测试用例和代码效率测试用例三部分。

1. 必做内容测试用例

必做内容测试用例分为简单测试用例、复杂测试用例和综合测试用例。简单测试用例和复杂测试用例都基本只包含某一种语言成份，区别在于测试用例复杂程度不同；综合测试用例则包含多种语言成份。

1.1 简单测试用例

此部分共包含 5 个测试用例。

(1) 测试对表达式（赋值表达式和算术表达式）的翻译。

```
int main()
{
    int a = 3, b = 4, c;
    c = a * a + b * b;
    write(c);
    c = (a + c) / b;
    write(c);
    a = a + b;
    b = a - b;
    a = a - b;
    b = ((a - b) * 2 - 1) + (a / b) * (2 * (3 + b));
    write(a);
    write(b);
    return 0;
}
```

输出：25 7 4 13

(2) 测试对分支语句的翻译。

```
int main()
{
    int a = 3, b;
    b = read();
    if(a > b)
        write(a);
    else
        write(b);
    return 0;
}
```

输入：5　　输出：5

(3) 测试对循环语句的翻译。

```
int main()
{
    int i = 10;
    while(i > 0)
    {
        write(i);
        i = i - 1;
    }
    return 0;
}
```

输出：10 9 8 7 6 5 4 3 2 1

(4) 测试对一维数组的翻译。

```
int main()
{
    int a[3];
    a[0] = 1;
    a[1] = 2;
    a[2] = a[0] + a[1];
    write(a[2]);
    return 0;
}
```

输出：3

(5) 测试对函数调用的翻译。

```
int print(int a, int b)
{
    write(a);
    write(b);
    return 0;
}

int main()
{
    int i, j, k;
    i = 2;
    j = 3;
    k = print(i, j);
    return 0;
}
```

输出：2 3

1.2 复杂测试用例

此部分共包含 4 个测试用例。

(1) 测试对分支语句的翻译。此测试用例用于判断以输入的 3 个数为边长能否构成三角形，若能则输出“1”，否则输出“-1”。

```
int main()
{
    int i, j, k;
    i = read();
    j = read();
    k = read();
    if (i <= 0 || j <= 0 || k <= 0)
        write(-1);
    else
    {
        if ((i + j) <= k)
            write(-1);
        else if ((i + k) <= j)
            write(-1);
        else if ((j + k) <= i)
            write(-1);
        else
            write(1);
    }
    return 0;
}
```

① 输入：-1 2 3　　输出：-1

② 输入：3 4 5　　输出：1

(2) 测试对循环语句的翻译。此测试用例用于输出两位数正整数（10-99）中各数字之和大于数字之积的整数（例如整数 18，各数字之和为 1+8，各数字之积为 1×8）。

```
int main()
{
    int s, p, m, n = 10;
    while(n < 100)
    {
        p = 1;
        s = 0;
        m = n;
        while(m != 0)
        {
            p = p * (m - (m / 10) * 10);
            s = s + (m - (m / 10) * 10);
            m = m / 10;
        }
        if(s > p)
            write(n);
        n = n + 1;
    }
    return 0;
}
```

输出：10 11 12 13 14 15 16 17 18 19 20 21 30 31 40 41 50 51 60 61 70 71 80 81 90 91

(3) 测试对一维数组的翻译。此测试用例用于将给定的一维数组中的元素逆序存放。

```
int main()
{
    int a[10];
    int i = 0, j = 10, temp;
    while(i < j)
    {
        a[i] = i;
        i = i + 1;
    }
    i = 0;
    while(i < (j / 2))
    {
        temp = a[i];
        a[i] = a[j - i - 1];
        a[j - i - 1] = temp;
        i = i + 1;
    }
    i = 0;
    while(i < j)
    {
        write(a[i]);
        i = i + 1;
    }
    return 0;
}
```

输出：9 8 7 6 5 4 3 2 1 0

(4) 测试对函数调用的翻译。此测试用例用于求两个正整数（35 和 14）的最大公约数。

```
int gcd(int m, int n)
{
    if(n == 0)
        return m;
    else
        return gcd(n, (m - (m / n)*n));
}

int main()
{
    int i = 35, j = 14, k;
    k = gcd(i, j);
    write(k);
    return 0;
}
```

输出：7

1.3 综合测试用例

此部分共包含 2 个测试用例。

(1)此测试用例为冒泡法排序程序,其中待排序的数字为若干个连续的偶数(以100为最大值)。输入为待排序数字的个数，输出为排序结果。

```
int sort(int n)
{
    int a[100], i = 0, j = 0;
    while (i < n)
    {
        a[i] = 100 - i * 2;
        i = i + 1;
    }
    i = 0;
    while (i < n)
    {
        j = 0;
        while (j < i)
        {
            if (a[i] < a[j])
            {
                int t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
            j = j + 1;
        }
        i = i + 1;
    }
    i = 0;
    while (i < n)
    {
        write(a[i]);
        i = i + 1;
    }
    return 0;
}

int main()
{
    int m;
    m = read();
    if (m >= 100) write(-1);
    else sort(m);
    return 0;
}
```

输入：0　　输出：(空)

输入：10　　输出：82 84 86 88 90 92 94 96 98 100

输入：120　　输出：-1

(2) 此测试用例为约瑟夫环问题：编号为 1, 2, 3,…，n 的 n 个人按顺时针方向依次排列成环（1 为 n 在顺时针方向的下一个人）， 选一个正整数 m 作为报数的值，从第 1 个人开始按顺时针报数，报到 m 时停止，报 m 的人出列，从他在顺时针方向的下一个人开始重新从 1 报数，如此下去，直到所有人全部出列为止。测试用例中 n 和 m 的取值分别为 21 和 5。

```
int josef(int m)
{
    int a[21];
    int i = 0, j, k = 0;
    while(i < 21)
    {
        a[i] = i + 1;
        i = i + 1;
    }
    i = 0;
    while(i < 21)
    {
        j = 1;
        while(j < m)
        {
            while(a[k] == 0)
            {
                k = (k + 1)-((k + 1) / 21 * 21);
            }
            j = j + 1;
            k = (k + 1)-((k + 1) / 21 * 21);
        }
        while(a[k] == 0)
        {
            k = (k + 1)-((k + 1) / 21 * 21);
        }
        write(a[k]);
        a[k] = 0;
        i = i + 1;
    }
    return 0;
}

int main()
{
    josef(5);
    return 0;
}
```

输出：5 10 15 20 4 11 17 2 9 18 6 14 3 16 8 1 21 7 13 19 12

2 分组内容测试用例

每个分组包含两个测试用例,第一个测试用例针对所有同学，第二个测试用例只针对需要完成相应分组的同学。

2.1 分组 3.1 测试用例

(1) 此测试用例针对所有同学

```
struct Complex
{
    int real, image;
};

int main()
{
    struct Complex c;
    c.real = 3;
    c.image = 4;
    write(c.real);
    write(c.image);
    return 0;
}
```

对于需要完成分组 3.1 的同学，此测试用例应输出“3 4”；对于不需要完成分组 3.1 的同学，应输出“Can not translate the code:”的响应提示，否则倒扣分。

(2) 此测试用例针对需要完成分组 3.1 的同学。此测试用例为给定直角坐标系中 4 个点的坐标：1(0,0)，2(3,3)，3(6,6)，4(9,9)，求与点 5(4,5)距离最近的点。

```
struct Node
{
    int id;
    int p[2];
};

int distance (struct Node n1, struct Node n2)
{
    int dis;
    dis = (n1.p[0] - n2.p[0]) * (n1.p[0] - n2.p[0])
          + (n1.p[1] - n2.p[1]) * (n1.p[1] - n2.p[1]);
    return dis;
}

int main()
{
    struct Node n[4];
    struct Node obj;
    int i = 0, j = 1, minDis, minId, cntDis;
    while (i < 4)
```

```
    {
        n[i].id = i + 1;
        n[i].p[0] = i * 3;
        n[i].p[1] = i * 3;
        i = i + 1;
    }
    obj.id = 5;
    obj.p[0] = 4;
    obj.p[1] = 5;
    minDis = distance(obj, n[0]);
    minId = 0;
    while(j < 4)
    {
        cntDis = distance(obj, n[j]);
        if (cntDis < minDis)
        {
            minDis = cntDis;
            minId = n[j].id;
        }
        j = j + 1;
    }
    write(minId);
    return 0;
}
```

输出：2

2.2 分组 3.2 测试用例

(1) 此测试用例针对所有同学

```
int main()
{
    int a[2][2];
    a[0][0] = 0;
    a[0][1] = 1;
    a[1][0] = 2;
    a[1][1] = 3;
    write(a[1][1]);
    return 0;
}
```

对于需要完成分组 3.2 的同学，此测试用例应输出“3”；对于不需要完成分组 3.2 的同学，应输出“Can not translate the code:”的相应提示，否则倒扣分。

(2) 此测试用例针对需要完成分组 3.2 的同学。此测试用例用于计算给定的两个矩阵的乘积。

```
int product (int m1[4], int m2[4])
{
    int sum = 0, i = 0;
    while(i < 4)
    {
        sum = sum + m1[i] * m2[i];
        i = i + 1;
    }
    return sum;
}

int main()
{
    int m3[2][4], m4[4][2], p[2][2], temp1[4], temp2[4];
    int j = 0, k = 0, l = 0;
    while(j < 2)
    {
        while(k < 4)
        {
            m3[j][k] = j + 1;
            m4[k][j] = k + 1;
            k = k + 1;
        }
        k = 0;
        j = j + 1;
    }
    j = k = 0;
    while(j < 2)
    {
        while(l < 4)
        {
            temp1[l] = m3[j][l];
            l = l + 1;
        }
        l = 0;
        while(k < 2)
        {
            while(l < 4)
            {
                temp2[l] = m4[l][k];
                l = l + 1;
            }
            l = 0;
            p[j][k] = product(temp1, temp2);
            write(p[j][k]);
            k = k + 1;
        }
        k = 0;
        j = j + 1;
    }
    return 0;
}
```

输出：10 10 20 20

3 代码效率测试用例

(1) 此测试用例包含很多公共子表达式、无用赋值、可折叠常数以及循环内部的冗余计算，为代码优化提供了相当大的空间。

```
int main()
{
    int a = 3, b = 4, i = 0, sum = 0;
    int c, d, e;
    int array[150];

    d = a * a;
    e = b * b;
    if ( -1 || d + e > 0)
        i = (a * a - b * b + d * e) / (a - 8);
    c = - ( 7 - a * (1+2) - (36/(10-1)) * b + d * e);

    i = c - c;
    while (i < 100+50)
    {
        array[i] = 1;
        i = i + 1;
    }

    i = array[0] + d - 1 - d;

    while (i < -1 * (c - b * b + a * a))
    {
        int k = a + 4 - b;
        int j = d, l = k - 1;
        while (j > a + b - 10 + 3)
        {
            int f = 0;
            array[i/10] = array[i/10] + (j*3);
            if (l == 2) array[i/10] = array[i/10] - (2 + j*3);
            if (i > 100)
                f = j / 2 - 1;
            else
                array[i/10] = array[i/10] + j;
            f = f - 1;
            j = j / 2 - 1;
        }

        if (b + sum < d || e)
            i = i + 1;
    }

    i=0;
```

```
    while (i < 150)
    {
        sum = sum + array[i];
        i = i + 1;
    }
    write(sum);

    return 0;
}
```

输出：830