

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	3
1.3	A-3	5
1.4	A-4	6
1.5	A-5	7
2	B 组测试用例	8
2.1	B-1	8
2.2	B-2	10
2.3	B-3	12
3	C 组测试用例	13
3.1	C-1	14
3.2	C-2	15
4	E 组测试用例	17
4.1	E1-1	17
4.2	E1-2	18
4.3	E1-3	19
4.4	E2-1	21
4.5	E2-2	22
4.6	E2-3	23
5	结束语	25

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

1.1.1 输入

```
1 int main() {
2     int input1;
3     int input2;
4     int initialVal = 100;
5     int intermediateCalc;
6     int reusedVar;
7     int finalResult;
8     int lateAssignVar;
9
10    input1 = 7;
11    input2 = 12;
12
13    intermediateCalc = (input1 + 5) * (input2 - initialVal / 50);
14
15    reusedVar = intermediateCalc + input1;
16    write(reusedVar);
17
18    write(intermediateCalc);
19
20    finalResult = (reusedVar - input2) * 3;
21
22    reusedVar = finalResult / 2;
23    write(reusedVar);
24
25    lateAssignVar = input2 * 10;
26    write(lateAssignVar);
27
28    write(finalResult);
29
30    return 0;
```

```
31 }
```

1.1.2 输出

```
1 >>> Empty
2 [127, 120, 172, 120, 345]
```

1.1.3 说明

主要针对赋值与算术语句进行测试。输入输出中以 »> 起始的行表示程序输入，其后的一行表示程序输出。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（下同）。

1.2 A-2

1.2.1 输入

```
1 int main() {
2     int num1;
3     int num2;
4     int threshold = 10;
5     int category = 0;
6
7     num1 = read();
8     num2 = read();
9
10    if (num1 > threshold && num2 > 0) {
11        write(100);
12        category = 1;
13    } else {
14        if (num1 <= threshold && num2 < 0) {
15            write(200);
16            category = 2;
17        } else {
18            write(250);
19            category = 3;
20        }
21    }
22 }
```

```

23     if (num1 == 0 || num2 == threshold) {
24         write(300);
25         if (category == 1 || category == 3) {
26             write(310);
27         }
28     } else {
29         write(400);
30         if (category == 2) {
31             write(410);
32         }
33     }
34
35     if ((num1 + num2 > 0) && (num1 > threshold || num2 < 0)) {
36         write(500);
37     } else {
38         write(600);
39     }
40
41     return 0;
42 }

```

1.2.2 输出

```

1 >>> 15, 8
2 [100, 400, 500]
3
4 >>> 5, -5
5 [200, 400, 410, 600]
6
7 >>> 8, 10
8 [250, 300, 310, 600]

```

1.2.3 说明

主要针对分支语句进行测试。

1.3 A-3

1.3.1 输入

```
1 int main() {
2     int dataArr[5];
3     int index1;
4     int index2;
5     int val1;
6     int calculatedVal;
7     int temp;
8
9     dataArr[0] = 10;
10    dataArr[4] = 50;
11
12    val1 = read();
13    index1 = read();
14    dataArr[index1] = val1;
15
16    index2 = read();
17
18    temp = dataArr[0] + dataArr[index1];
19    calculatedVal = temp * dataArr[index2];
20
21    write(dataArr[0]);
22    write(dataArr[index1]);
23    write(dataArr[index2]);
24    write(dataArr[4]);
25    write(calculatedVal);
26
27    return 0;
28 }
```

1.3.2 输出

```
1 >>> 25, 1, 0
2 [10, 25, 10, 50, 350]
3
```

```
4 >>> 5, 2, 4
5 [10, 5, 50, 50, 750]
```

1.3.3 说明

主要针对一维数组进行测试。

1.4 A-4

1.4.1 输入

```
1 int main() {
2     int n;
3     int i;
4     int j;
5     int currentVal;
6     int totalSum = 0;
7     int tempSum;
8
9     n = read();
10
11     i = 1;
12     while (i <= n) {
13         j = 1;
14         while (j <= n) {
15             currentVal = i * 10 + j;
16             tempSum = i + j;
17             if ((tempSum / 2) * 2 == tempSum) {
18                 totalSum = totalSum + currentVal;
19             }
20             j = j + 1;
21         }
22         i = i + 1;
23     }
24
25     write(totalSum);
26
27     return 0;
28 }
```

1.4.2 输出

```
1 >>> 2
2 [33]
3
4 >>> 3
5 [110]
```

1.4.3 说明

主要针对循环语句进行测试。

1.5 A-5

1.5.1 输入

```
1 int processHelper(int val) {
2     if (val < 0) {
3         return val * 2;
4     }
5     return val + 10;
6 }
7
8 int recursiveWithHelperCall(int n) {
9     int helperVal;
10    int recursiveVal;
11
12    if (n <= 0) {
13        return 0;
14    } else {
15        helperVal = processHelper(n);
16        recursiveVal = recursiveWithHelperCall(n - 2);
17        return helperVal + recursiveVal;
18    }
19 }
20
21 int main() {
```

```

22     int input;
23     int finalResult;
24
25     input = read();
26
27     finalResult = recursiveWithHelperCall(input);
28
29     write(finalResult);
30
31     return 0;
32 }

```

1.5.2 输出

```

1 >>> 4
2 [26]
3
4 >>> 5
5 [39]
6
7 >>> 0
8 [0]

```

1.5.3 说明

主要针对函数调用进行测试。

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

2.1.1 输入

```

1 int calculateBase(int inputParam) {
2     int baseRes;
3     if (inputParam > 50) {

```



```

4         baseRes = inputParam / 2;
5         return baseRes;
6     } else {
7         baseRes = inputParam + 10;
8         return baseRes;
9     }
10 }
11
12 int determineIndex(int baseIn, int inputOther) {
13     int indexRes = (baseIn + inputOther) / 5;
14     if (indexRes < 0) { return 0; }
15     if (indexRes > 4) { return 4; }
16     return indexRes;
17 }
18
19 int main() {
20     int dataArray[5];
21     int val1;
22     int val2;
23     int baseOut;
24     int targetIdx;
25     int originalVal;
26     int modifiedVal;
27
28     dataArray[0]=1; dataArray[1]=2; dataArray[2]=3; dataArray[3]=4;
29     dataArray[4]=5;
30
31     val1 = read();
32     val2 = read();
33
34     baseOut = calculateBase(val1);
35     targetIdx = determineIndex(baseOut, val2);
36     originalVal = dataArray[targetIdx];
37
38     if (baseOut > 30 && val2 > 0) {
39         dataArray[targetIdx] = baseOut;
40     } else {

```

```

40         dataArray[targetIdx] = originalVal * 2;
41     }
42     modifiedVal = dataArray[targetIdx];
43
44     write(targetIdx);
45     write(originalVal);
46     write(modifiedVal);
47
48     return 0;
49 }

```

2.1.2 输出

```

1 >>> 60, 10
2 [4, 5, 10]
3
4 >>> 45, 5
5 [4, 5, 55]
6
7 >>> 20, -15
8 [3, 4, 8]

```

2.1.3 说明

2.2 B-2

2.2.1 输入

```

1 int processGridValue(int rowIdx, int colIdx, int factorParam) {
2     int gridResult = rowIdx * rowIdx + colIdx * factorParam;
3     if ((gridResult / 10) * 10 == gridResult) {
4         return gridResult / 10;
5     } else {
6         return gridResult + 5;
7     }
8 }
9
10 int main() {

```

```

11     int gridSize = 4;
12     int rowLoop = 0;
13     int colLoop = 0;
14     int procFactor;
15     int gridProcessedVal;
16     int gridTotalSum = 0;
17     int gridConditionCount = 0;
18
19     procFactor = read();
20
21     while (rowLoop < gridSize) {
22         colLoop = 0;
23         while (colLoop < gridSize) {
24             gridProcessedVal = processGridValue(rowLoop, colLoop,
25                 procFactor);
26             gridTotalSum = gridTotalSum + gridProcessedVal;
27             if (gridProcessedVal > 25) {
28                 gridConditionCount = gridConditionCount + 1;
29             }
30             colLoop = colLoop + 1;
31         }
32         rowLoop = rowLoop + 1;
33     }
34
35     write(gridTotalSum);
36     write(gridConditionCount);
37
38     return 0;

```

2.2.2 输出

```

1 >>> 3
2 [175, 0]
3
4 >>> 10
5 [302, 6]

```

2.2.3 说明

2.3 B-3

2.3.1 输入

```
1  int getNextValue(int idxParam, int oldValParam, int modParam) {
2      int valueModifier;
3      if ((idxParam / 2) * 2 == idxParam) {
4          valueModifier = idxParam + 5;
5      } else {
6          valueModifier = idxParam * 3;
7      }
8      return oldValParam + valueModifier;
9  }
10
11 int checkComplexCondition(int valueToCheck, int checkThresholdParam)
12 {
13     if (valueToCheck > 50 && checkThresholdParam < 5) { return 1; }
14     if (valueToCheck < 10 && checkThresholdParam > 0) { return 1; }
15     return 0;
16 }
17
18 int main() {
19     int mainArray[5];
20     int loopCounter = 0;
21     int valueModifierIn;
22     int complexCheckValIn;
23     int conditionMetCounter = 0;
24     int loopExecLimit;
25
26     mainArray[0]=5; mainArray[1]=8; mainArray[2]=3; mainArray[3]=12;
27     mainArray[4]=7;
28     valueModifierIn = read();
29     complexCheckValIn = read();
30     loopExecLimit = read();
31     if (loopExecLimit > 5) { loopExecLimit = 5; }
32     if (loopExecLimit < 0) { loopExecLimit = 0; }
```

```

32     while (loopCounter < loopExecLimit) {
33         mainArray[loopCounter] = getNextValue(loopCounter, mainArray[
            loopCounter], valueModifierIn);
34         if (checkComplexCondition(mainArray[loopCounter],
            complexCheckValIn) == 1) {
35             conditionMetCounter = conditionMetCounter + 1;
36         }
37         loopCounter = loopCounter + 1;
38     }
39
40     write(mainArray[0]);
41     if (loopExecLimit > 0) {
42         write(mainArray[loopExecLimit-1]);
43     } else {
44         write(mainArray[0]);
45     }
46     write(conditionMetCounter);
47     write(loopCounter);
48
49     return 0;
50 }

```

2.3.2 输出

```

1 >>> 10, 3, 4
2 [10, 21, 0, 4]
3
4 >>> 2, 8, 5
5 [10, 16, 0, 5]

```

2.3.3 说明

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

3.1.1 输入

```
1 int calculateInnerLoopLimit(int totalSize, int outerLoopIndex) {
2     int limitResult;
3     limitResult = totalSize - outerLoopIndex - 1;
4     return limitResult;
5 }
6
7 int main() {
8     int sortArrayData[5];
9     int arrayDataSize = 5;
10    int outerIdx = 0;
11    int innerIdx = 0;
12    int swapHolder;
13    int outerLimit;
14    int innerLimit;
15
16    while (outerIdx < arrayDataSize) {
17        sortArrayData[outerIdx] = read();
18        outerIdx = outerIdx + 1;
19    }
20
21    outerIdx = 0;
22    outerLimit = arrayDataSize - 1;
23    while (outerIdx < outerLimit) {
24        innerIdx = 0;
25        innerLimit = calculateInnerLoopLimit(arrayDataSize, outerIdx)
26        ;
27        while (innerIdx < innerLimit) {
28            if (sortArrayData[innerIdx] > sortArrayData[innerIdx +
29            1]) {
30                swapHolder = sortArrayData[innerIdx];
31                sortArrayData[innerIdx] = sortArrayData[innerIdx +
32                1];
33                sortArrayData[innerIdx + 1] = swapHolder;
34            }
35        }
36        outerIdx = outerIdx + 1;
37    }
38}
```

```

32         innerIdx = innerIdx + 1;
33     }
34     outerIdx = outerIdx + 1;
35 }
36
37 outerIdx = 0;
38 while (outerIdx < arrayDataSize) {
39     write(sortArrayData[outerIdx]);
40     outerIdx = outerIdx + 1;
41 }
42
43 return 0;
44 }

```

3.1.2 输出

```

1 >>> 5, 4, 3, 2, 1
2 [1, 2, 3, 4, 5]
3
4 >>> 10, 2, 8, 1, 6
5 [1, 2, 6, 8, 10]

```

3.1.3 说明

3.2 C-2

3.2.1 输入

```

1 int isDivisibleBy(int numParam, int divisorParam) {
2     int quotient;
3     if (divisorParam <= 0) { return 0; }
4     if (divisorParam == 1) { return 1; }
5     quotient = numParam / divisorParam;
6     if (quotient * divisorParam == numParam) {
7         return 1;
8     } else {
9         return 0;
10    }

```

```

11 }
12
13 int isPrimeFunc(int numberParam) {
14     int divisorVar = 2;
15     int checkLimit;
16
17     if (numberParam <= 1) {
18         return 0;
19     }
20
21     checkLimit = numberParam / 2 + 1;
22
23     while (divisorVar < checkLimit) {
24         if (isDivisibleBy(numberParam, divisorVar) == 1) {
25             return 0;
26         }
27         divisorVar = divisorVar + 1;
28     }
29
30     return 1;
31 }
32
33 int main() {
34     int numToCheck;
35     int primeResult;
36
37     numToCheck = read();
38     primeResult = isPrimeFunc(numToCheck);
39     write(primeResult);
40
41     return 0;
42 }

```

3.2.2 输出

```

1 >>> 7
2 [1]

```



```
3
4 >>> 10
5 [0]
6
7 >>> 13
8 [1]
9
10 >>> 1
11 [0]
12
13 >>> 2
14 [1]
```

3.2.3 说明

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

4.1.1 输入

```
1 struct Point2D {
2     int pointX;
3     int pointY;
4 };
5
6 int main() {
7     struct Point2D startPoint;
8     int tempValX;
9     int tempValY;
10    int calculatedDist;
11
12    startPoint.pointX = 3;
13    startPoint.pointY = 5;
14
```

```

15     tempValX = startPoint.pointX;
16     tempValY = startPoint.pointY;
17
18     calculatedDist = tempValX * tempValX + tempValY * tempValY;
19
20     write(tempValX);
21     write(tempValY);
22     write(calculatedDist);
23
24     return 0;
25 }

```

4.1.2 输出

```

1 >>> Empty
2 [3, 5, 34]

```

4.1.3 说明

测试对于简单结构体的翻译。输入输出仅针对选做要求 4.1 的同学，其余同学需要提示无法翻译不输出中间代码。

4.2 E1-2

4.2.1 输入

```

1 struct Vector2D {
2     int vectorDx;
3     int vectorDy;
4 };
5
6 int processVector(struct Vector2D vecParam) {
7     int resultVal;
8     int tempDx;
9     int tempDy;
10
11     tempDx = vecParam.vectorDx;
12     tempDy = vecParam.vectorDy;

```

```

13     resultVal = tempDx + tempDy * 2;
14     return resultVal;
15 }
16
17 int main() {
18     struct Vector2D motionVector;
19     int processedResult;
20
21     motionVector.vectorDx = read();
22     motionVector.vectorDy = read();
23
24     processedResult = processVector(motionVector);
25
26     write(processedResult);
27
28     return 0;
29 }

```

4.2.2 输出

```

1 >>> 10, 5
2 [20]
3
4 >>> 3, -2
5 [-1]

```

4.2.3 说明

测试将结构体作为函数参数。输入输出仅针对选做要求 4.1 的同学，其余同学需要提示无法翻译不输出中间代码。

4.3 E1-3

4.3.1 输入

```

1 struct ColorRGB {
2     int colorR;
3     int colorG;

```

```

4     int colorB;
5 };
6
7 struct PixelData {
8     int pixelX;
9     int pixelY;
10    struct ColorRGB pixelColorValue;
11 };
12
13 int analyzePixel(struct PixelData pixelParam) {
14     int analysisCode = 0;
15     int redValue;
16     int xPos;
17
18     redValue = pixelParam.pixelColorValue.colorR;
19     xPos = pixelParam.pixelX;
20
21     if (redValue > 100) {
22         analysisCode = analysisCode + 10;
23     }
24     if (xPos < 50) {
25         analysisCode = analysisCode + 1;
26     }
27     return analysisCode;
28 }
29
30 int main() {
31     struct PixelData screenPixel;
32     int analysisResult;
33
34     screenPixel.pixelColorValue.colorR = read();
35     screenPixel.pixelColorValue.colorG = 100;
36     screenPixel.pixelColorValue.colorB = 50;
37
38     screenPixel.pixelX = read();
39     screenPixel.pixelY = 80;
40

```

```

41     analysisResult = analyzePixel(screenPixel);
42
43     write(analysisResult);
44
45     return 0;
46 }

```

4.3.2 输出

```

1 >>> 150, 30
2 [11]
3
4 >>> 80, 70
5 [0]
6
7 >>> 200, 90
8 [10]

```

4.3.3 说明

测试对于较复杂的结构体的使用。输入输出仅针对选做要求 4.1 的同学，其余同学需要提示无法翻译且不输出中间代码。

4.4 E2-1

4.4.1 输入

```

1 int main() {
2     int matrixGrid[2][3];
3     int fixedRow = 1;
4     int fixedCol = 2;
5     int valueAccessed;
6
7     matrixGrid[0][1] = 15;
8     matrixGrid[1][0] = 25;
9     matrixGrid[1][2] = matrixGrid[0][1] + matrixGrid[1][0];
10
11     valueAccessed = matrixGrid[fixedRow][fixedCol];

```

```
12
13     write(valueAccessed);
14     write(matrixGrid[0][1]);
15
16     return 0;
17 }
```

4.4.2 输出

```
1 >>> Empty
2 [40, 15]
```

4.4.3 说明

测试高维数组变量的使用。输入输出仅针对选做要求 4.2 的同学，其余同学需要提示无法翻译且不输出中间代码。

4.5 E2-2

4.5.1 输入

```
1 int incrementElemFunc(int targetVector[4], int indexFixed) {
2     targetVector[indexFixed] = targetVector[indexFixed] + 1;
3     return targetVector[indexFixed];
4 }
5
6 int main() {
7     int dataList[4];
8     int modifyFixedIdx = 1;
9     int valModified;
10
11     dataList[0]=10; dataList[1]=30; dataList[2]=50; dataList[3]=70;
12
13     valModified = incrementElemFunc(dataList, modifyFixedIdx);
14
15     write(valModified);
16     write(dataList[0]);
17 }
```

```
18     return 0;
19 }
```

4.5.2 输出

```
1 >>> Empty
2 [31, 10]
```

4.5.3 说明

测试一维数组的传参。输入输出仅针对选做要求 4.2 的同学，其余同学需要提示无法翻译且不输出中间代码。

4.6 E2-3

4.6.1 输入

```
1 int processFixed1DArrayWithCheck(int arrayParam[4], int checkVal) {
2     int sum = 0;
3     int i = 0;
4
5     while (i < 4) {
6         if (arrayParam[i] != checkVal) {
7             sum = sum + arrayParam[i];
8         }
9         i = i + 1;
10    }
11    return sum;
12 }
13
14 int main() {
15     int cubeStorage[2][3][2];
16     int lineVector[4];
17     int iDim = 0;
18     int jDim = 0;
19     int kDim = 0;
20     int lineProcResult;
21     int cubeEvenSum = 0;
```

```

22     int skipValue;
23     int finalComp;
24
25     iDim = 0;
26     while (iDim < 4) {
27         lineVector[iDim] = iDim * 3 + 1;
28         iDim = iDim + 1;
29     }
30
31     iDim = 0;
32     while (iDim < 2) {
33         jDim = 0;
34         while (jDim < 3) {
35             kDim = 0;
36             while (kDim < 2) {
37                 cubeStorage[iDim][jDim][kDim] = iDim * 100 + jDim *
38                     10 + kDim;
39                 kDim = kDim + 1;
40             }
41             jDim = jDim + 1;
42             iDim = iDim + 1;
43         }
44
45         skipValue = read();
46         lineProcResult = processFixed1DArrayWithCheck(lineVector,
47             skipValue);
48
49         iDim = 0;
50         while (iDim < 2) {
51             jDim = 0;
52             while (jDim < 3) {
53                 kDim = 0;
54                 while (kDim < 2) {
55                     int currentCubeVal;
56                     currentCubeVal = cubeStorage[iDim][jDim][kDim];
57                     if ((currentCubeVal / 2) * 2 == currentCubeVal) {

```



```

57         cubeEvenSum = cubeEvenSum + currentCubeVal;
58     }
59     kDim = kDim + 1;
60 }
61 jDim = jDim + 1;
62 }
63 iDim = iDim + 1;
64 }
65
66 if (lineProcResult > cubeEvenSum) {
67     finalComp = 1;
68 } else {
69     finalComp = 0;
70 }
71
72 write(lineProcResult);
73 write(cubeEvenSum);
74 write(finalComp);
75
76 return 0;
77 }

```

4.6.2 输出

```

1 >>> 4
2 [18, 360, 0]
3
4 >>> 10
5 [12, 360, 0]

```

4.6.3 说明

测试更复杂情况下对数组的使用。输入输出仅针对选做要求 4.2 的同学，其余同学需要提示无法翻译且不输出中间代码。

5 结束语

若对本文档有任何疑议，可写邮件与孙伟杰助教联系，注意同时抄送给许畅老师。