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

1.1.1 输入

```
int main() {
    int input1;
    int input2;
    int initialVal = 100;
    int intermediateCalc;
    int reusedVar;
    int finalResult;
    int lateAssignVar;

    input1 = 7;
    input2 = 12;

    intermediateCalc = (input1 + 5) * (input2 - initialVal / 50);

    reusedVar = intermediateCalc + input1;
    write(reusedVar);

    write(intermediateCalc);

    finalResult = (reusedVar - input2) * 3;

    reusedVar = finalResult / 2;
    write(reusedVar);

    lateAssignVar = input2 * 10;
    write(lateAssignVar);

    write(finalResult);

    return 0;
```

```
}
```

1.1.2 输出

```
>>> Empty
[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

```
int main() {
    int num1;
    int num2;
    int threshold = 10;
    int category = 0;

    num1 = read();
    num2 = read();

    if (num1 > threshold && num2 > 0) {
        write(100);
        category = 1;
    } else {
        if (num1 <= threshold && num2 < 0) {
            write(200);
            category = 2;
        } else {
            write(250);
            category = 3;
        }
    }

```

```
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

```
int main() {
    int dataArr[5];
    int index1;
    int index2;
    int val1;
    int calculatedVal;
    int temp;

    dataArr[0] = 10;
    dataArr[4] = 50;

    val1 = read();
    index1 = read();
    dataArr[index1] = val1;

    index2 = read();

    temp = dataArr[0] + dataArr[index1];
    calculatedVal = temp * dataArr[index2];

    write(dataArr[0]);
    write(dataArr[index1]);
    write(dataArr[index2]);
    write(dataArr[4]);
    write(calculatedVal);

    return 0;
}
```

1.3.2 输出

```
>>> 1, 2, 3, 4, 5
[1, 2, 3, 1, 99, 3, 4, 5]

```

```
>>> 11, 22, 33, 44, 55
[11, 22, 33, 11, 99, 33, 44, 55]
```

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

```
int main() {
    int n;
    int i;
    int j;
    int currentVal;
    int totalSum = 0;
    int tempSum;

    n = read();

    i = 1;
    while (i <= n) {
        j = 1;
        while (j <= n) {
            currentVal = i * 10 + j;
            tempSum = i + j;
            if ((tempSum / 2) * 2 == tempSum) {
                totalSum = totalSum + currentVal;
            }
            j = j + 1;
        }
        i = i + 1;
    }

    write(totalSum);

    return 0;
}
```

1.4.2 输出

```
>>> 2
[33]

>>> 3
[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

```
int processHelper(int val) {
    if (val < 0) {
         return val * 2;
    }
    return val + 10;
}

int recursiveWithHelperCall(int n) {
    int helperVal;
    int recursiveVal;

    if (n <= 0) {
        return 0;
    } else {
        helperVal = processHelper(n);
        recursiveVal = recursiveWithHelperCall(n - 2);
        return helperVal + recursiveVal;
    }
}

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  >>> 5
2  [0]
3
4  >>> 6
5  [1]
```

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 {
```

```
        baseRes = inputParam + 10;
        return baseRes;
    }
}

int determineIndex(int baseIn, int inputOther) {
    int indexRes = (baseIn + inputOther) / 5;
    if (indexRes < 0) { return 0; }
    if (indexRes > 4) { return 4; }
    return indexRes;
}

int main() {
    int dataArray[5];
    int val1;
    int val2;
    int baseOut;
    int targetIdx;
    int originalVal;
    int modifiedVal;

    dataArray[0]=1; dataArray[1]=2; dataArray[2]=3; dataArray[3]=4;
       dataArray[4]=5;

    val1 = read();
    val2 = read();

    baseOut = calculateBase(val1);
    targetIdx = determineIndex(baseOut, val2);
    originalVal = dataArray[targetIdx];

    if (baseOut > 30 && val2 > 0) {
        dataArray[targetIdx] = baseOut;
    } else {
        dataArray[targetIdx] = originalVal * 2;
    }
    modifiedVal = dataArray[targetIdx];
```

```
    write(targetIdx);
    write(originalVal);
    write(modifiedVal);

    return 0;
}
```

2.1.2 输出

```
>>> 60, 10
[4, 5, 10]

>>> 45, 5
[4, 5, 55]

>>> 20, -15
[3, 4, 8]
```

2.1.3 说明

2.2 B-2

2.2.1 输入

```
int processGridValue(int rowIdx, int colIdx, int factorParam) {
    int gridResult = rowIdx * rowIdx + colIdx * factorParam;
    if ((gridResult / 10) * 10 == gridResult) {
        return gridResult / 10;
    } else {
        return gridResult + 5;
    }
}

int main() {
    int gridSize = 4;
    int rowLoop = 0;
    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,
                    procFactor);
25              gridTotalSum = gridTotalSum + gridProcessedVal;
26              if (gridProcessedVal > 25) {
27                  gridConditionCount = gridConditionCount + 1;
28              }
29              colLoop = colLoop + 1;
30          }
31          rowLoop = rowLoop + 1;
32      }
33
34      write(gridTotalSum);
35      write(gridConditionCount);
36
37      return 0;
38  }
```

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

```
int getNextValue(int idxParam, int oldValParam, int modParam) {
    int valueModifier;
    if ((idxParam / 2) * 2 == idxParam) {
        valueModifier = idxParam + 5;
    } else {
        valueModifier = idxParam * 3;
    }
    return oldValParam + valueModifier;
}

int checkComplexCondition(int valueToCheck, int checkThresholdParam)
    {
    if (valueToCheck > 50 && checkThresholdParam < 5) { return 1; }
    if (valueToCheck < 10 && checkThresholdParam > 0) { return 1; }
    return 0;
}

int main() {
    int mainArray[5];
    int loopCounter = 0;
    int valueModifierIn;
    int complexCheckValIn;
    int conditionMetCounter = 0;
    int loopExecLimit;

    mainArray[0]=5; mainArray[1]=8; mainArray[2]=3; mainArray[3]=12;
       mainArray[4]=7;
    valueModifierIn = read();
    complexCheckValIn = read();
    loopExecLimit = read();
    if (loopExecLimit > 5) { loopExecLimit = 5; }
    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

```
int calculateInnerLoopLimit(int totalSize, int outerLoopIndex) {
    int limitResult;
    limitResult = totalSize - outerLoopIndex - 1;
    return limitResult;
}

int main() {
    int sortArrayData[5];
    int arrayDataSize = 5;
    int outerIdx = 0;
    int innerIdx = 0;
    int swapHolder;
    int outerLimit;
    int innerLimit;

    while (outerIdx < arrayDataSize) {
        sortArrayData[outerIdx] = read();
        outerIdx = outerIdx + 1;
    }

    outerIdx = 0;
    outerLimit = arrayDataSize - 1;
    while (outerIdx < outerLimit) {
        innerIdx = 0;
        innerLimit = calculateInnerLoopLimit(arrayDataSize, outerIdx)
           ;
        while (innerIdx < innerLimit) {
            if (sortArrayData[innerIdx] > sortArrayData[innerIdx +
               1]) {
                swapHolder = sortArrayData[innerIdx];
                sortArrayData[innerIdx] = sortArrayData[innerIdx +
                   1];
                sortArrayData[innerIdx + 1] = swapHolder;
            }
```

```
            innerIdx = innerIdx + 1;
        }
        outerIdx = outerIdx + 1;
    }

    outerIdx = 0;
    while (outerIdx < arrayDataSize) {
        write(sortArrayData[outerIdx]);
        outerIdx = outerIdx + 1;
    }

    return 0;
}
```

3.1.2 输出

```
>>> 5, 4, 3, 2, 1
[1, 2, 3, 4, 5]

>>> 10, 2, 8, 1, 6
[1, 2, 6, 8, 10]
```

3.1.3 说明

3.2 C-2

3.2.1 输入

```
int isDivisibleBy(int numParam, int divisorParam) {
    int quotient;
    if (divisorParam <= 0) { return 0; }
    if (divisorParam == 1) { return 1; }
    quotient = numParam / divisorParam;
    if (quotient * divisorParam == numParam) {
        return 1;
    } else {
        return 0;
    }
```

```
}

int isPrimeFunc(int numberParam) {
    int divisorVar = 2;
    int checkLimit;

    if (numberParam <= 1) {
        return 0;
    }

    checkLimit = numberParam / 2 + 1;

    while (divisorVar < checkLimit) {
        if (isDivisibleBy(numberParam, divisorVar) == 1) {
            return 0;
        }
        divisorVar = divisorVar + 1;
    }

    return 1;
}

int main() {
    int numToCheck;
    int primeResult;

    numToCheck = read();
    primeResult = isPrimeFunc(numToCheck);
    write(primeResult);

    return 0;
}
```

3.2.2 输出

```
>>> 7
[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
```

```
    tempValX = startPoint.pointX;
    tempValY = startPoint.pointY;

    calculatedDist = tempValX * tempValX + tempValY * tempValY;

    write(tempValX);
    write(tempValY);
    write(calculatedDist);

    return 0;
}
```

4.1.2 输出

```
>>> Empty
[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

```
struct Vector2D {
  int vectorDx;
  int vectorDy;
};

int processVector(struct Vector2D vecParam) {
    int resultVal;
    int tempDx;
    int tempDy;

    tempDx = vecParam.vectorDx;
    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;
```

```
    int colorB;
};

struct PixelData {
    int pixelX;
    int pixelY;
    struct ColorRGB pixelColorValue;
};

int analyzePixel(struct PixelData pixelParam) {
    int analysisCode = 0;
    int redValue;
    int xPos;

    redValue = pixelParam.pixelColorValue.colorR;
    xPos = pixelParam.pixelX;

    if (redValue > 100) {
        analysisCode = analysisCode + 10;
    }
    if (xPos < 50) {
        analysisCode = analysisCode + 1;
    }
    return analysisCode;
}

int main() {
    struct PixelData screenPixel;
    int analysisResult;

    screenPixel.pixelColorValue.colorR = read();
    screenPixel.pixelColorValue.colorG = 100;
    screenPixel.pixelColorValue.colorB = 50;

    screenPixel.pixelX = read();
    screenPixel.pixelY = 80;

```

```
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;
```

```
    int skipValue;
    int finalComp;

    iDim = 0;
    while (iDim < 4) {
        lineVector[iDim] = iDim * 3 + 1;
        iDim = iDim + 1;
    }

    iDim = 0;
    while (iDim < 2) {
        jDim = 0;
        while (jDim < 3) {
            kDim = 0;
            while (kDim < 2) {
                cubeStorage[iDim][jDim][kDim] = iDim * 100 + jDim *
                    10 + kDim;
                kDim = kDim + 1;
            }
            jDim = jDim + 1;
        }
        iDim = iDim + 1;
    }

    skipValue = read();
    lineProcResult = processFixed1DArrayWithCheck(lineVector,
       skipValue);

    iDim = 0;
    while (iDim < 2) {
        jDim = 0;
        while (jDim < 3) {
            kDim = 0;
            while (kDim < 2) {
                int currentCubeVal;
                currentCubeVal = cubeStorage[iDim][jDim][kDim];
                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。