

You should **submit** an single electronic package that contains the **source** files for your work on the programming questions, along with a **README** file and sample inputs and outputs, following a procedure similar to that used for the previous homework. The rules for using outside resources are also similar to those in that homework.

The main task for this homework is **implementing a compiler for the extended calculator language** of HW02. (The program implemented for HW02 is an *interpreter* for the extended calculator language.)

In more detail, the **input** consists of a valid program in the calculator language of calc.py as discussed in class, extended to support the div and mod operators from HW02. The **output** consists of (only) a well-formatted (using the textbook's conventions) JCoCo assembly language program such that, when that program is executed (say, using the *coco* command), it performs the actions specified by the input calculator program. In particular, the output of such an execution (of the output of this homework's compiler) should be exactly equal to the output of the interpreter from HW02 on the same input.

Unlike the previous homeworks, this homework's program (the compiler) does not have a uniquely specified correct output for a given input, since there are several assembly language programs that are equivalent (in the above sense) to a given calculator language program.

The following sample inputs and outputs illustrate some of these details. (The formatting of the outputs should be improved if possible but the presented form is acceptable.)

Sample Input 1:

```
xyzzzy = -300000 + 5 * 7 - 3
xyzzzy
```

```
STORE_FAST 0
LOAD_FAST 0
LOAD_FAST 0
LOAD_GLOBAL 0
ROT_TWO
CALL_FUNCTION 1
POP_TOP
LOAD_CONST 5
RETURN_VALUE
END
```

Sample Output 1:

```
Function: main/0
Constants: 300000, 0, 5, 7, 3, None
Locals: xyzzzy
Globals: print
BEGIN
  LOAD_CONST 0
  LOAD_CONST 1
  ROT_TWO
  BINARY_SUBTRACT
  LOAD_CONST 2
  LOAD_CONST 3
  BINARY_MULTIPLY
  BINARY_ADD
  LOAD_CONST 4
  BINARY_SUBTRACT
```

Sample Input 2:

```
tri = 1 + 2 + 3 + 4 + 5
pin = 1 * 2 * 3 * 4 * 5
ssq = 1*1 + 2*2 + 3*3 + 4*4 + 5*5
scb = 1*1*1 + 2*2*2 + 3*3*3 + 4*4*4 + 5*5*5
tri
pin
ssq
scb
```

Sample Output 2:

```
Function: main/0
Constants: 1, 2, 3, 4, 5, None
Locals: tri, pin, ssq, scb
Globals: print
```

BEGIN		LOAD_CONST 1
LOAD_CONST 0		LOAD_CONST 1
LOAD_CONST 1		BINARY_MULTIPLY
BINARY_ADD		LOAD_CONST 1
LOAD_CONST 2		BINARY_MULTIPLY
BINARY_ADD		BINARY_ADD
LOAD_CONST 3		LOAD_CONST 2
BINARY_ADD		LOAD_CONST 2
LOAD_CONST 4		BINARY_MULTIPLY
BINARY_ADD		LOAD_CONST 2
STORE_FAST 0		BINARY_MULTIPLY
LOAD_FAST 0		BINARY_ADD
LOAD_CONST 0		LOAD_CONST 3
LOAD_CONST 1		LOAD_CONST 3
BINARY_MULTIPLY		BINARY_MULTIPLY
LOAD_CONST 2		LOAD_CONST 3
BINARY_MULTIPLY		BINARY_MULTIPLY
LOAD_CONST 3		BINARY_ADD
BINARY_MULTIPLY		LOAD_CONST 4
LOAD_CONST 4		LOAD_CONST 4
BINARY_MULTIPLY		BINARY_MULTIPLY
STORE_FAST 1		LOAD_CONST 4
LOAD_FAST 1		BINARY_MULTIPLY
LOAD_CONST 0		BINARY_ADD
LOAD_CONST 0		STORE_FAST 3
BINARY_MULTIPLY		LOAD_FAST 3
LOAD_CONST 1		LOAD_FAST 0
LOAD_CONST 1		LOAD_GLOBAL 0
BINARY_MULTIPLY		ROT_TWO
BINARY_ADD		CALL_FUNCTION 1
LOAD_CONST 2		POP_TOP
LOAD_CONST 2		LOAD_FAST 1
BINARY_MULTIPLY		LOAD_GLOBAL 0
BINARY_ADD		ROT_TWO
LOAD_CONST 3		CALL_FUNCTION 1
LOAD_CONST 3		POP_TOP
BINARY_MULTIPLY		LOAD_FAST 2
BINARY_ADD		LOAD_GLOBAL 0
LOAD_CONST 4		ROT_TWO
LOAD_CONST 4		CALL_FUNCTION 1
BINARY_MULTIPLY		POP_TOP
BINARY_ADD		LOAD_FAST 3
STORE_FAST 2		LOAD_GLOBAL 0
LOAD_FAST 2		ROT_TWO
LOAD_CONST 0		CALL_FUNCTION 1
LOAD_CONST 0		POP_TOP
BINARY_MULTIPLY		LOAD_CONST 5
LOAD_CONST 0		RETURN_VALUE
BINARY_MULTIPLY		END