Enhance your compiler with these ideas.
Show me your working solutions as you complete them.
Open book, notes, computer, Internet, and calculator.
No Human help, including humans through the Internet.

1. Give me a copy of your completed compiler code, so I can run my test suite on it.
   Remove any print statements from your code other than the output, so my automated test
cases will work better. If you email it, remove all Debug and Release directories, zip it,
and rename the .zip file to .zpi. Or you can copy it to my USB drive.

2. Support using keywords “true” and “false”. Assign “true” to be the integer 1 and “false” to be
   the integer 0. Thus:
   ```
   if ((3<5) == true)
```
   is the same as:
   ```
   if ((3<5) == 1)
```
   which should pass in both cases.

3. For Interpreting only, support the “shift arithmetic right” operator >> as it works in
   C++. It’s priority must be set to between addition/subtraction and the relational operators.

   How does it work?
   X >> 1 will shift the bits of X 1 position to the right, which basically divides by 2.
   X >> 3 will shift the bits of X 3 positions to the right, which basically divides by 8.
   X >> Y will shift Y positions to the right.

   Examples:
   ```
   int x;
   x = 24;
   int y;
   y = 2;
   cout << (x >> y); // 6
   cout << (x >> y-1); // 12, not 5. -1 happens first
   cout << (x >> y==2); // 0, not 12. ==2 happens after
   ```

4. For Coding and Executing, support the “shift arithmetic right” operator.

   Hint: Follow the pattern of PopPopMulPush, except instead of using MUL_EAX_EBX1
   and MUL_EAX_EBX2, use:
   ```
   const unsigned char SAR_EAX_ECX1 = 0xD3;
   const unsigned char SAR_EAX_ECX2 = 0xF8;
   ```

   Note these shift the bits of the EAX register by what is in the ECX register, so instead of
   popping into EBX as you did with PopPopMulPush, pop into ECX, using:
   ```
   const unsigned char POP_ECX = 0x59;
   ```