CISC 223 002 (Winter 22)
Assignment #4: Formal Languages Specification (20 Points)

Student Name/ID: Lucas Papadatos/20233257

Solutions are due before 2 PM on **Sunday March 27, 2022**.

- The assignments are graded according to the correctness, preciseness and legibility of the solutions. All handwritten parts included as figures in the .tex file should be clear and legible. This assignment is marked out of 20 possible marks.
- Please submit your solutions as PDF on onQ before the deadline.
- The assignment must be based on individual work. Copying solutions from other students is a violation of academic integrity. See the course onQ site for more information.

1. (5 marks) What should the pre-condition `P` be in each of the following ten correctness statements for the statement to be an instance of Hoare's axiom scheme? All variables are of type `int`.

   (a) `P { x = 2; } x == 1`
   `2 == 1`

   (b) `P { x = 2; } x > 2`
   `2 > 2`

   (c) `P { x = y + z; } x < y + z`
   `y + z < y + z`

   (d) `P { x = x + y + z; } z > x*x + 2`
   `z > (x + y + z)*(x + y + z) + 2`

   (e) `P { x = x + y + z; } y*y > z + 5`
   `y*y > z + 5`

   (f) `P { x = y + z; } Exists(w = 0; w < 10) x + w == 50`
   `Exists(w = 0; w < 10) y + z + w == 50`

   (g) `P { x = y + z; } ForAll(z = 1; z < 100) x + 2*z > w + 2`
   `ForAll(v = 1; v < 100) (y + z) + 2*v > w + 2`

   (h) `P { x = y + z; } ForAll(x = 1; x < z) x + y + 2 < 100`
   `ForAll(x = 1; x < z) x + y + 2 < 100`

(i) `P { x = y + z; } ForAll(y = 1; y < n) Exists(z = 1; z < n) x*y <= 3*z+w`
`ForAll(u = 1; u < n) Exists(v = 1; v < n) (y + z)*u <= 3*v+w`

(j) `P { x = y + z; } ForAll(y = 1; y < n) Exists(x = 1; x < n) x*y <= 3*z+w`
`ForAll(y = 1; y < n) Exists(x = 1; x < n) x*y <= 3*z+w`

2. Verify the validity of the following two correctness statements (a) and (b) by adding all the intermediate assertions, that is, give the proof tableau showing the validity of the correctness statement. All variables are of type int. Also state any mathematical facts used.

(a) (2.5 marks)

```
ASSERT( x >= 2 && y == 0 && z < 0 )
// y == 0 implies y <= 0
// x >= 2 and y == 0 implies x + y >= 1
ASSERT( y <= 0 && x + y >= 1 )
ASSERT( 3 - y >= 3 && y - 3 + x >= -2 )
ASSERT( x - y + 3 - x >= 3 && y - 3 + x >= -2 )
z = 3 - x;
ASSERT( x - y + z >= 3 && y - z >= -2 )
ASSERT( x - (y - z) >= 3 && (y - z) >= -2 )
y = y - z;
ASSERT( x - y >= 3 && y >= -2 )
x = x - y;
ASSERT( x >= 3 && y >= -2 )
```

(b) (2.5 marks)

```
ASSERT( true )
if (x >= y) x = y - 1; else y = y + 1;
z = y + 1;
ASSERT( x < y < z )
```

if:

```
ASSERT( true && x >= y )
// ASSERT( true ) implies always true
ASSERT( y - 1 < y < y + 1 )
x = y - 1;
ASSERT( x < y < y + 1 )
z = y + 1;
ASSERT( x < y < z )
```

else:

```
ASSERT( true && x < y )
ASSERT( x < y + 1 < y + 2 )
// x < y + 1 implies x < y < y + 1
```

```
    y = y + 1;
    ASSERT( x < y < y + 1 )
    z = y + 1;
    ASSERT( x < y < z )
```

3. (5 marks) Write and verify a program that computes the sum of the cubes of the first `n` positive integers. Below is the specification.

   Declarative interface:

   ```
   const int n; /* the program will compute the sum of the cubes of the first
   n positive integers */
   int sum; /* the sum of the cubes is stored in this variable */
   ```

   Below are given the pre- and post-condition:

   ```
   ASSERT( n >= 1 )
   sum = 0;
   s = 0;
   while (s < n) {
       s = s + 1;
       sum = sum + (s*s*s);
   }
   ASSERT( sum == ∑_{i=1}^{n} i*i*i)
   ```

   The program should use a **while-loop and standard arithmetic operations**. Select a loop invariant and **give a complete proof tableau** for the program (including all the intermediate assertions). Also make an argument for termination.

   ```
   Invariant:  sum == ∑_{i=1}^{s} i*i*i && 0 <= s <= n
   Proof Tableau:
   ASSERT( n >= 1 )
   // n >= 1 implies n > 0
   ASSERT( 0 == 0 && 0 <= 0 <= n )
   sum = 0;
   ASSERT( sum == 0 && 0 <= 0 <= n )
   s = 0;
   ASSERT( I )
   while (s < n) {
       ASSERT( I && s < n )
       // s < n implies s + 1 <= n
       ASSERT( sum + ((s+1)*(s+1)*(s+1)) == ∑_{i=1}^{s+1} i*i*i && 0 <= s + 1 <= n )
       s = s + 1;
       ASSERT( sum + (s*s*s) == ∑_{i=1}^{s} i*i*i && 0 <= s <= n )
       sum = sum + (s*s*s);
       ASSERT( I )
   }
   ```

```
ASSERT( I && s >= n )
ASSERT( sum == ∑_{i=1}^{n} i*i*i )
Termination:  Since 0 <= s <= n holds after each iteration of the loop and
the loop increments s, the loop must terminate after a finite number of iterations.
```

4. (5 marks) Verify the below code using the suggested invariant. That is, give a complete proof tableau by adding all the intermediate assertions. Also make an argument for termination. All variables are of type int.

```
ASSERT(num >= 0 && den > 0)
quot = 0; rem = num;
while (rem >= den)
INVAR( num == quot * den + rem && 0 <= rem && 0 < den)
{ rem = rem - den;
quot++;
}
ASSERT( num == quot * den + rem && 0 <= rem < den)
```

Note: This is Exercise 3.12 in the textbook and a discussion of the code appears on page 69.

```
Proof Tableau:
ASSERT( num >= 0 && den > 0 )
// num >= 0 implies 0 <= num
// den > 0 implies 0 < den
ASSERT( num == num && 0 <= num && 0 < den )
quot = 0;
ASSERT( num == quot * den + num && 0 <= num && 0 < den )
rem = num;
ASSERT( I )
while (rem >= den)
INVAR( num == quot * den + rem && 0 <= rem && 0 < den )
{
  ASSERT( I && rem >= den )
  // rem >= den implies rem - den >= 0
  // num == quot * den + rem implies num == quot * den + rem
  ASSERT( num == quot * den + rem && 0 <= rem-den && 0 < den )
  ASSERT( num == (quot+1) * den + (rem-den) && 0 <= rem-den && 0 < den )
  rem = rem - den;
  ASSERT( num == (quot+1) * den + rem && 0 <= rem && 0 < den )
  quot++;
  ASSERT( I )
}
ASSERT( I && rem < den )
ASSERT( num == quot * den + rem && 0 <= rem < den )
```

Termination: Since rem >= den holds after each iteration of the loop and the loop decrements rem by subtracting den, the loop must terminate after a finite number of iterations.