CSE 3311 SOFTWARE DESIGN
REPORT 2 SPECIFICATION

PRZEMYSLAW PAWLUK

Due: Thursday, October 19, 5:30pm
Where: In class
NOTE: If the class has begun your report is late

1. Main Points

The explicit specification is the union of this document plus the program text you are given. Be sure to consider all the implicit specifications, for example and without limitation, be sure to read and follow all the guidelines from the links on reports and academic honesty from the WWW home page for the course.

1.1. Learning objectives.

- Reading and writing assertions
- Reading and understanding contracts
- Verifying the correctness of algorithms

1.2. **To hand in.** Hand in, in class, a report containing the following items as a package in the given order.

(1) Cover page printed from the course web pages
(2) Your report consisting of your solutions to the tasks in Section 2. For reports done in pairs, include an appendix describing the contributions of the two team members.
(3) Electronic submission: **There is no electronic submission for this report.**

Date: October 2, 2012.

2. Tasks

2.1. **Greatest common divisor (GDC) contract.** Using mathematical notation, annotate the best possible require, ensure, invariant and variant assertions for the following function. gcd, that computes the greatest common divisor of two positive integers. The greatest common divisor (GDC) of two positive integers, a and b, denoted by $gcd(a,b)$ is the largest natural number that divides both a and b. Some examples include $gcd(9,6) = 3$, $gcd(16,5) = 1$. Euclids algorithm computes the gcd of two numbers as follows:

Step A:
Write $a = q * b + r$ where $0 <= r <= b$.
StepB:
If $r > 0$,then set $a = b$, $b = r$ and *goto Step A*. Otherwise last b is the GDC.
The equation $a = q*b+r$ implies that $gcd(a,b) = gcd(b,r)$ and hence the process works.
Here are several iterations:
$a = q1 * b + r1$ where $0 <= r1 <= b$
$b = q2 * r1 + r2$ where $0 <= r2 < r1$
$r1 = q3 * r2 + r3$ where $0 <= r3 < r2$

The algorithm:

```
gcd (a, b: INTEGER): INTEGER
      -- Greatest common divisor of a and b.
      require ???
      local x, y, remainder: INTEGER
      do
            from x := a ; y := b; remainder := x \\ y -- remainder of x divided by y
            invariant ???
            until remainder = 0
                  loop
                        x := y
                        y := remainder
                        remainder := x \\ y
                        variant ???
                  end
      Result := y
      ensure ???
end
```

2.2. **Cumulative sum contract.** Using mathematical notation, annotate the best possible require, ensure, invariant and variant assertions for the following function, cumulative_sum, that creates an array that contains the cumulative sum of the first n integers in the array in.

```
cumulative_sum(in : ARRAY[INTEGER], n : INTEGER) : ARRAY[INTEGER]
      require ???
      local  j : INTEGER
      do
            create Result.make(1, n)
            from Result[1] := in[1] ; j := 1
            invariant ???
            until j = n - 1 do
                  j := j + 1
                  Result[j] := Result[j-1] + in[j]
            variant ???
            end
         ensure ???
end
```

2.3. **Separate even-odd contract.** Using mathematical notation, annotate the best possible require, ensure, invariant and variant assertions for the following function, separate_even_odd, that rearranges the elements of the array such that the lower part contains the even integers in the original array and the upper part contains the odd integers the order of the even integers and the odd integers does not have to be the same as in the original array. The returned result is the split point index.

```
separate_even_odd(in : ARRAY[INTEGER]) : INTEGER
   require ???
   local max_even : INTEGER ; min_odd : INTEGER do
            from min_odd := in.upper + 1 ; max_even := in.lower - 1
            invariant ???
            until max_even = min_odd - 1 do
                  max_even := max_even + 1
                  if max_even \= min_odd then
                        if odd(in[max_even]) then
                              min_odd := min_odd - 1
                              swap(in[max_even], in[min_odd])
                              if odd(in[max_even]) then
                                    max_even := max_even - 1
                              end
                          end
                        end
                        variant ???
                  end Result := min_odd
      ensure ???
 end
```

2.4. **Verify double_half algorithm is correct.** From the given contract, verify the algorithm double_half is correct.

```
double_half (in : ARRAY[REAL])
      require in \= void
      local k : INTEGER do
            from k = in.lower
      invariant
```

$\forall j : in.lower...k-1 | odd(in[j]) \bullet in'[j] = in[j] * 2$
$\wedge\, \forall j : in.lower...k-1 | even(in[j]) \bullet in'[j] = in[j]/2$

```
until k > in.upper loop
   if even(in[k]) then
      in[k] := in[k]/2
   else
      in[k] := in[k]*2
   end
   k:=k+1
end
ensure
```

$\forall j : in.lower...in.upper | odd(in[j]) \bullet in'[j] = in[j] * 2$
$\wedge\, \forall j : in.lower...in.upper | even(in[j]) \bullet in'[j] = in[j]/2$

```
end
```

3. Grading scheme

The grade for the report is partitioned into the following parts.

(1) Overall presentation 10%
(2) Greatest common divisor (GCD) contract 20%
(3) Cumulative sum contract 20%
(4) Separate even-odd contract 20%
(5) Verify double_half algorithm 30%