CS3101: Quiz 1. **UWO, January 29, 2015.**

Student ID number:
Student Last Name:

Guidelines. The quiz consists of two exercises and is a closed book test. All answers should be written in the *answer boxes*. No justifications for the answers are needed, unless explicitly required. You are expected to do this quiz on your own without assistance from anyone else in the class. If possible, please avoid pencils and use pens with **dark ink**. Thank you.

Exercise 1: multiple choice questions In each case, **zero, one or more answers** may be correct; indicate **all correct answers**.

(1) Consider the following `Julia` function `f`:

```
function f(u,v)
   n=length(u)
   [u[i] - v[i] for i=1:n]
end
```

which assumes that `u` and `v` are vectors of equal length.

(a) the function call `f(u,v)` prints "Hello World"
(b) the function call `f(u,v)` does not return anything
(c) the function call `f(u,v)` returns the difference $u - v$ of the vectors `u` and `v`
(d) the function call `f(u,v)` returns the difference $u - v$ of the vectors `u` and `v` provided that their coefficients are integer numbers.

Answer: [c]

(2) Assume that in a `Julia` session, one enters the following command line:

```
reduce(+,[1,2,3,4,5,6,7,8,9,10])
```

Which of the following results will be displayed:

(a) `1+2+3+4+5+8+9+10`

(b) `55`
(c) `[55]`

Answer: [b]

(3) Consider the following `Julia` session:

```
n = @parallel (+) for i=1:10
            i
end
```

After executing the above, the value of `n` is:

(a) 10
(b) 55
(c) the number of processors involved in this `Julia` session
(d) a remote reference

Answer: [b]

(4) Assume that in a Julia session, one enters successively the following three command lines:

```
addprocs(1)
r = @spawnat 2 2+3
fetch(r)
```

What is the value returned after entering the third command like?

(a) `2`
(b) `5`
(c) `3`

Answer: [b]

(5) Consider the following `Julia` session where two methods are proposed for computing the square of a random matrix.

```
#method 1
A = rand(1000,1000)
Bref = @spawn A^2
```

```
fetch(Bref)

# method 2
Bref = @spawn rand(1000,1000)^2
fetch(Bref)
```

- (a) In the first method, a random matrix is constructed locally, then sent to another processor where it is squared.
- (b) In the first method, a random matrix is both constructed and squared on another processor.
- (c) In the second method, a random matrix is constructed locally, then sent to another processor where it is squared.
- (d) In the second method, a random matrix is both constructed and squared on another processor.

Answer: [a,d]

(6) Assume that in a `Julia` session, one enters the following command line:

```
da = @parallel [2 * i for i = 1:10]
```

Which of the following statements are true?

- (a) `da` is an integer variable whose value is the sum $(2\cdot 1)+\cdots+(2\cdot 10) = 110$
- (b) `da` is the array of 10 values [2 4 6 8 10 12 14 16 18 20]
- (c) `da` is the distributed array of 10 values [2 4 6 8 10 12 14 16 18 20]

Answer: [c]

(7) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers `2` and `3`, own `da`: Worker `2` owns the first 5 elements and Worker `3` owns the last ones. What does the following command return?

```
[(@spawnat p sum(localpart(da))) for p=procs(da)]
```

(a) An array of two remote references
(b) `[2 3]`

Answer: [a]

(8) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers `2` and `3`, own `da`: Worker `2` owns the first 5 elements and Worker `3` owns the last ones. What does the following command return?

```
map(fetch, { (@spawnat p sum(localpart(da))) for p=procs(da) })
```

(a) A 2-element array with entries `30` and `80`
(b) A 2-element array with remote references as entries

Answer: [a]

(9) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers `2` and `3`, own `da`: Worker `2` owns the first 5 elements and Worker `3` owns the last ones. What does the following command return?

```
reduce(+, map(fetch, { (@spawnat p sum(localpart(da))) for p=procs
```

(a) `2`
(b) `110`

Answer: [b]

(10) Consider the following `Julia` functions defined in the same `Julia` session, which is using 4 workers in addition to the master.

```
@everywhere function fib(n)
                 if (n < 2) then
                     return n
                 else return fib(n-1) + fib(n-2)
                 end
              end

@everywhere function fib_parallel(n)
          if (n < 40) then
              return fib(n)
          else
              x = @spawn fib_parallel(n-1)
              y = fib_parallel(n-2)
              return fetch(x) + y
          end
       end
```

What is the value `fib_parallel(4)`?

(a) 3
(b) 5
(c) 8

Answer: [a]

Exercise 2: writing a parallel `Julia` function

The *Padovan sequence* is the sequence of integers $P(n)$ defined by the initial values and recurrence relation below:

$$P(1) = P(2) = P(3) = 1 \text{ and } P(n) = P(n-2) + P(n-3). \qquad (1)$$

The first few values of $P(n)$ are $1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151, 200, 265$.

1. Write a serial `Julia` function, that on an input positive integer `n`, computes the n-th Padovan number $P(n)$.

```
function Padovan(n)
   if (n < 4) then
      return 1
   else
      return Padovan(n-2) + Padovan(n-3)
   end
end
```

2. Write a parallel `Julia` function, that on an input positive integer `n`, computes the n-th Padovan number $P(n)$. The parallel constructs to be used are `@spawn` and `fetch`.

```
@everywhere function Padovan(n)
     if (n < 4) then
        return 1
     else
        // spawn task Padovan(n-2) to another processor
        x = @spawn Padovan(n-2)
        y =  Padovan(n-3)
        //fetch value "x" from the processor who execute this task
        return fetch(x) + y
     end
end
```

3. Write a `Julia` task `t`, that generates the sequence of the Padovan numbers. In other words, calling `consume(t)` repeatedly, produces the Padovan numbers successively.

```
function Padovan()
   p1 = 1
   produce(p1)
   p2 = 1
   produce(p2)
   p3 = 1
   produce(p3)
   while true
      p4 = p1 + p2
      produce(p4)
      p1 = p2
      p2 = p3
      p3 = p4
   end
end

s = Task(Padovan)

[consume(s) for i=1:22]
```