Problem 1. timer.icn

Implement a Timer class that allows one to measure the execution time of Icon code. It has these methods:

Timer(name:string)
   Create a timer with the name specified.

start()  Start the timer running, possibly continuing accumulation of time
          on a timer that was previously stopped.

stop()   Stop the timer.

reset()  Reset the timer.

elapsed() Return the elapsed time in milliseconds

string() Print the name of the timer and the current elapsed time.

Timer times CPU time, which can be accessed with &time.

Here's a program that uses two Timers to take a look at the execution of speed of integer versus real arithmetic:

procedure main()
   ti := Timer("int + int")
   ti.start()
   i := 0
   every 1 to 1000000 do
      i +=: 1
   ti.stop()

   tr := Timer("real + real")
   tr.start()
   i := 0.0
   every 1 to 1000000 do
      i +=: 1.0
   tr.stop()

   write(ti.string())
   write(tr.string())
end

Execution:

   int + int: 1061ms
   real + real: 1292ms
Problem 2. String.icn

In this problem you are to implement an "abstract" class called String and four concrete subclasses.

String has these methods:

- length()  Return the length of the String.
- value()    Return a string (the built-in type) that contains the characters represented by the String.
- char()     Generate the characters in String.
- at(n)      Return the nth character in String.

The first subclass of String is PalString, which represents a palindromic version of the initializing value:

```plaintext
ps := PalString("oops");
r := ...

ps.value();
r := "oopsspoo"  (string)

ps.length();
r := 8    (integer)

every write(ps.at(1 to ps.length()));
 oo
 p s
 s p
 oo
Failure

every write(ps.char() \ 5);
 oo
 p s
 s
Failure
```
The second subclass of String is ReplString, which represents a string that is replicated some number of times. Example:

```plaintext
s1 := ReplString("abc", 2);
```

```plaintext
s1.length();
  r := 6  (integer)
```

```plaintext
s1.value();
  r := "abcabc"  (string)
```

```plaintext
every write(s1.char());
a
b
c
a
b
c
```

Failure

Another example:

```plaintext
s2 := ReplString(&lcase, 1000000000000000); 
```

```plaintext
s2.length();
  r := 26000000000000000  (integer)
```

```plaintext
s2.at(26000000000000000);
  r := "z"  (string)
```

```plaintext
every write(s2.char() \ 3);
a
b
c
```

Note that calling s2.value() would be problematic!

The third subclass is IspString, which represents an "interspersed" string. It is best described with an example:

```plaintext
s1 := IspString("abc",".");
```

```plaintext
s1.value();
  r := "a.b.c"  (string)
```

The constructor specifies that the string "." is to be interspersed between the characters of the string "abc".

The interspersing string may be more than one character long:

```plaintext
s2 := IspString("abcd", "<>");
```

```plaintext
s2.value();
```
r := "a<->b<->c<->d"  (string)

][ s2.length();
 r := 13  (integer)

][ every write(s2.char() \ 5);
 a
 <
 -
 >
 b
 Failure

Another example:

][ n := (s3 := IspString(repl("x", 1000000),
 repl(&ucase, 1000000))).length();
 r := 25999975000000  (integer)

][ s3.at(1);
 r := "x"  (string)

][ s3.at(2);
 r := "A"  (string)

][ s3.at(27);
 r := "Z"  (string)

][ s3.at(28);
 r := "A"  (string)

][ s3.at(26000002);
 r := "x"  (string)

][ s3.at(28000000000);
 r := "M"  (string)

][ s3.at(28000000001);
 r := "N"  (string)

The fourth class is RandomString, which represents a random sequence of characters with a specified length and consisting of characters from a specified character set. Example:

][ rs := RandomString('ATCG', 10).value();
 r := "CGGGGGATCC"  (string)

][ rs := RandomString('ATCG', 10).value();
 r := "TTAGCGCTTC"  (string)

][ rs := RandomString('ATCG', 10);
}
] rs.length();
r := 10 (integer)

][ rs.at(1);
  r := "A" (string)
]

][ rs.at(5);
  r := "C" (string)
]

][ rs.at(10);
  r := "C" (string)
]

][ rs.value();
  r := "AGCCCAGAGC" (string)
]

][ rs.at(2);
  r := "G" (string)
]

][ rs2 := RandomString('ATCG', 4000000000);
]

][ rs2.length();
  r := 4000000000 (integer)
]

][ rs2.at(rs2.length());
  r := "A" (string)

Note that once a character is produced with at() or value(), its value is permanently fixed, i.e, a random choice is made (at most) once for each character. Subsequent references to a character or the whole string produce the same value.

Restriction: With the exception of PalString, your implementation should be able to handle very long strings, such as the examples shown above. You may limit your solution to only handling length() and at() in cases where very long strings are represented.

Miscellaneous

Note that this is a practice assignment to simply give you some experience with Unicon; it will not be graded. The specification is less rigorous than prior assignments and behavior in many odd cases in not specified; handle those cases in any way you'd like.

Reference Versions

It's not simple to hide the implementation of classes in Unicon, and to spare you the temptation of poking around in the instructor's solution, no reference versions are supplied.

Deliverables

None.