CSc 451, Spring 2003
Assignment 6
Due: Thursday, March 27 at 18:00

Problem 1. (5 points) `expand.icn`

In this problem you are to use Icon's string scanning facilities to redo the spell-checker word expansion problem from the first examination. Example:

```
% cat expand.1
abbreviate,s,d,\ing,\ion
bar,s,"ed,"ing
calmest
% expand < expand.1
abbreviate
abbreviates
abbreviated
abbreviating
abbreviation
bar
bars
barred
barring
calmest
%
```

You may assume the input is well-formed.

In order to encourage use of Icon's string scanning facilities, the following RESTRICTIONS apply: (1) The only types you may use in your solutions are strings, integers, character sets, and procedures. Note that an implication of this restriction is that you can't use `split`. (2) String subscripting (e.g., `s[3]`) and sectioning to produce a substring (e.g. `s[2:0]`, `s[i:+n]`) may not be used. (3) The string comparison operators, such as `==` and `<<`, may not be used. (4) You may not write procedures such as `charAt(s,i)` to circumvent these rules.

Problem 2. (5 points) `repeated.icn`

Write a program that reads lines on standard input and if the string entirely consists of repetitions of another string, display that string. If there is no repetition, indicate that. Example:

```
% cat repeated.1
xxxxxx
abcabcabc
abcabca
ababaababa
xyz
xyxyx
% repeated < repeated.1
xxxxxx: x
abcabcabc: abc
abcabca: <no repetition>
ababaababa: ababa
xyz: <no repetition>
xyxyx: <no repetition>
%
```

The program should find the shortest string that can be repeated to produce the input string. For example, "`xxxxxx`" can be thought of as two repetitions of "`xxx`" or three repetitions of "`xx`", but the correct result is "`x`".

Assume that each input line contains at least one character.

RESTRICTIONS: Same as problem 3.

Problem 3. (10 points) `calc.icn`

In this problem you are to write a simple five-function line-oriented calculator. Addition, multiplication, subtraction, division, and exponentiation (indicated with "`**`") are supported. Example:

```
% calc
Expression? 2*4
8
Expression? 7/3
2
Expression? 50+70**2
14400
^D
%
```

All operators have the same precedence except assignment, which is lower, and there is no parenthesization; use simple left to right evaluation.

Values may be assigned to variables, whose names consist solely of letters. Uninitialized variables have the value zero. Example:

```
% calc
Expression? x=10
10
Expression? y=20
20
Expression? z=x+y*2
60
Expression? z+q
60
Expression? ABC
0
```

Assume the input contains no whitespace and is well-formed. If there is an assignment, it will be the left-most operator. There will be at most one assignment in an expression. All values will be non-negative integers.

RESTRICTIONS: Same as problem 3, but a table may be used.

Problem 4. (20 points) `patterns.icn`

In this problem you are to write a program that reads lines on standard input and searches for various patterns of numbers, dots, and dashes. One pattern to be recognized is an even number followed by a dash followed by an even number. This pattern is called "E-E". To accommodate the unimaginative, the word "even" (case insensitive) may be specified instead of an even number. Here are some lines that match E-E:

```
4-190
even-2
Even-EVEN
```

The second pattern is O-O, similarly matching odd numbers. The following lines all match O-O:

```
1-1
odd-odd
3-ODD
7-11
```

The third pattern is called "sandwich". It is matched by lines where two even numbers surround an odd number or two odd numbers surround an even number, such as these:

```
1-2-1
3-even-odd
odd-10-odd
5-7-9
```

The fourth pattern is called "x2". It is matched by lines with two "sandwich"s, separated by a dot, such as these:

```
1-2-3.even-odd-even
2-3-4.5-6-7
```

The program should read lines on standard input and print each line followed by the name of the pattern matched by the line, if any. If no pattern is matched, then "<No match>" should be shown.

Example:

```
% cat patterns.1
4-190
even-3
Even-EVEN
Even-EVEN-1
3-ODD
7-11
1-2-1-2
1-2-1
3-even-odd
odd-10-odd
5-7-9
```

```
1-2-3.even-odd-even
2-3-4.5-6-7
1-2-1.1-1-1
odd-odd
%
% patterns <patterns.1

4-190: E-E

even-3: <No match>

Even-EVEN: E-E

Even-EVEN-1: <No match>

3-ODD: O-O

7-11: O-O

1-2-1-2: <No match>

1-2-1: sandwich

3-even-odd: sandwich

odd-10-odd: sandwich

5-7-9: <No match>

1-2-3.even-odd-even: x2

2-3-4.5-6-7: x2

1-2-1.1-1-1: <No match>

odd-odd: O-O
%
```

Note that a blank line is printed before each actual line of output.

RESTRICTIONS: Same as problem 3.

Problem 5. (40 points) `xmlparse.icn`

Write a program, `xmlparse`, that parses a well-formed XML document and prints a dump of the entire document. Here is a simple XML document with information about months of the year (in `months.xml`):

```
<months>
    <month name="January">
        <days>31</days>
    </month>
    <month name = 'February' >
        <days>28</days>
        <days>29</days>
    </month>
    <month name="July">
        <days>31</days>
        <holiday day="4" name="Independence Day"/>
    </month>
</months>
```

`xmlparse` accepts a single command-line argument that is the name of the file to dump:

```
% xmlparse months.xml
Element: months
    Element: month
    Attributes:
        name="January"
        Element: days
            Char Data: "31"
    Element: month
    Attributes:
        name="February"
        Element: days
            Char Data: "28"
        Element: days
            Char Data: "29"
    Element: month
    Attributes:
        name="July"
        Element: days
            Char Data: "31"
        Element: holiday
        Attributes:
            day="4"
            name="Independence Day"
```

Another example:

```
% cat text.xml
<doc>
This is a <i>document</i>.  Here is
some text in <b>boldface</b>.  A
footnote<fnref/> might appear here.
<fndef n="1">Footnote text</fndef>
<!--
<fndef n="1">Old footnote</fndef>
-->
</doc>
%
% xmlparse text.xml
Element: doc
    Char Data: "This is a"
    Element: i
        Char Data: "document"
    Char Data: ".  Here is\nsome text in"
    Element: b
        Char Data: "boldface"
    Char Data: ".  A\nfootnote"
    Element: fnref
    Char Data: "might appear here."
    Element: fndef
    Attributes:
        n="1"
        Char Data: "Footnote text"
```

You may assume that the XML documents presented as input are well-formed according to the XML supplemental material presented in class.

Trim leading and trailing whitespace (blanks, tabs, newlines) from character data. Display character data using the `image()` function.

You may find Icon records to be useful in this problem.

Problem 6. (10 points) `rslen.icn`

Write a program that reads Icon expressions, one per line, from standard input and prints each expression followed by the length of the expression's result sequence. Example:

```
% cat rslen.1
1 to 10
!&lcase
seq()
10
1|2|3
i := 0 & repl(!&lcase, i +:= 1)
s := "rotate" & i:= 1 to *s & s[i:0]||s[1:i]
s := "am"|"pm" & (12|(1 to 11))||":"||(0 to 5)||(0 to 9)||s
s1 := "butterball" & s2 := "biscuit" & !s1 == !~s2
%
% rslen < rslen.1
1 to 10: 10 results
!&lcase: 26 results
seq(): >1000 results
10: 1 results
1|2|3: 3 results
i := 0 & repl(!&lcase, i +:= 1): 26 results
s := "rotate" & i:= 1 to *s & s[i:0]||s[1:i]: 6 results
s := "am"|"pm" & (12|(1 to 11))||":"||(0 to 5)||(0 to 9)||s:
>1000 results
s1:= "butterball" & s2 := "biscuit" & !s1 == !~s2: 5 results
```

By default, if an expression generates more than 1000 results, the text "> 1000 results" is shown. That limit can be changed by specifying a command line argument:

```
% rslen 10
1 to 10
1 to 11
^D
1 to 10: 10 results
1 to 11: >10 results
%
```

Unfortunately, there is no facility in Icon to compile expressions at run-time. To solve this problem you'll need to have `rslen` generate, compile, and execute an Icon program that performs the desired computation.

Use "`xrslen.icn`" for the name of the program generated by `rslen`. With that in mind, here's how you might use the `system()` function to compile and execute `xrslen.icn`:

```
system("icont -s xrslen.icn -x")
```

Use the `remove()` function to delete the files `xrslen.icn` and `xrslen` when execution is complete. You may assume that the input expressions do no I/O.

Problem 7. (5 points) `Altends.icn`

Write a PDCO named `Altends(e)` that generates the results of the expression `e` in this way: first result, last result, second result, next-to-last result, etc.

Example:

```
][ .every Altends{1 to 3};
   1  (integer)
   3  (integer)
   2  (integer)
][ .every Altends{!"abcd"};
   "a"  (string)
   "d"  (string)
   "b"  (string)
   "c"  (string)
][ .every Altends{Altends{1|5|2|4|3}};
   1  (integer)
   2  (integer)
   3  (integer)
   4  (integer)
   5  (integer)
][ .every Altends{1 < 0};
][
```

If the result sequence of `e` is infinite, `Altends(e)` never produces a result.

DO NOT include a main procedure in `Altends.icn`. Note that the file name is capitalized.

Miscellaneous

Aside from code appearing in the class handouts, in the texts, or in messages from the instructor you may not seek, study, refer to, use, incorporate, etc., any existing solutions or portions thereof for any problem.

Keep in mind that an undeclared variable is treated as a local UNLESS it has the same name as a built-in procedure, in which case it is treated as a global variable. For larger programs it is strongly recommended that locals be declared and that programs be compiled with the `-u` options, which flags uses of undeclared variables. See slide 49, "Scope Rules—a hazard", for more details.

Reference Versions

Reference versions of all programs and some files with test data can be found in `/home/cs451/a6`. For each problem your solution must produce exactly the same output as the reference versions. Use `diff` to compare the output of your versions to the reference versions.

Notify the instructor if the behavior of a reference version seems to contradict or extend a problem specification.

Deliverables

Use `turnin` with the tag `451_6` to submit your solutions for grading. The deliverables for this assignment are `expand.icn`, `repeated.icn`, `calc.icn`, `patterns.icn`, `xmlparse.icn`, `rslen.icn`, and `Altends.icn` (procedure only).