# CSc 451, Spring 2003
# Final Examination Solutions

Problem 1: (3 points)

*In some cases it would be nice to have a convenient syntax to initialize some number of table entries when a table is created. Here is a proposed addition to Icon:*

> *If the argument to `table()` is a list of N elements where N >= 2 and even, then the elements are assumed to be [key1, value1, key2, value2, ...] and those key/value pairs are added to the table.*

The key issue is that such an addition would create an ambiguity: There's no way to tell if, for example, a two-element list is a default value or a key/value pair. One way to accommodate an initializing list of key/value pairs would be to have it as an optional second argument to `table()`.

Problem 2: (2 points)

*Consider the idea that `+` be used instead of '`||`' for string concatenation, and instead of '`|||`' for list concatenation. For example, `"xyz" + "123"` would produce the string `"xyz123"` and `[1,2] + [3,4]` would produce the list `[1,2,3,4]`. Cite one advantage and one disadvantage of this broader meaning for the `+` operator. Ignore the problem of invalidating some existing code.*

Two advantages among several: (1) A single operator is easier to remember. (2) Polymorphic routines using + can concatenate strings and lists.

The big disadvantage is that the result of an expression such as 1 + "2" is debatable and requires non-trivial specification. Another problem among several is that the plus sign is so strongly associated with addition that some would take issue with the fact that concatenation is not commutative.

Problem 3: (2 points)

*Given this Unicon procedure,*

```
procedure f(s1:split, s2:2)
     return *s1*s2
end
```

*What does `f("just testing")` return?*

It returns 4.

Problem 4: (3 points)

*Consider this statement:*

```
if (c == "x") | (c == "X") then ...
```

*Show three different ways to perform the same comparison that are more concise.*

Here are three:

```
if c == !"xX" then...
if c == ("x" | "X") then...
if map(c) == "x" then...
```

Problem 5: (4 points)

Write a procedure `span(L)` that suspends the smallest value in `L` and then suspends the largest value in `L`. Assume that `L` is a list of integers. `span(L)` fails if L is empty.

```
procedure span(L)
    L := sort(L)
    suspend L[1|-1]
end
```

Problem 6: (15 points)

Write a program `lcount` that accepts one or more file names as command line arguments and prints the number of lines that each file contains. If no command line arguments are specified, lines on standard input are counted instead.

```
procedure main(a)
    if *a = 0 then
        put(a, &input)

    R := []
    longest := ""

    every f := !a do {
        if f === &input then
            fname := "<stdin>"
        else {
            fname := f
            f := open(f) | stop(fname, ": Can't open")
            }
        c := create !f
        while @c
        close(f)
        put(R, fname, *c)
        if *fname > *longest then
            longest := fname
        }
    while fname := get(R) do
        write(left(fname,*longest), " ", right(get(R),3),
           " lines")
end
```

Problem 7: (8 points)

Characters in a URL can be specified using a two-digit hexadecimal code preceded by a percent sign. For example, instead of "`http://www.google.com`" one might use "`http://www%2egoogle%2eco%6d`".

Write a procedure `urlhex(s)` that returns a copy of the string `s` with any such hexadecimal specifications converted to the corresponding characters.

```
procedure urlhex(s)
    s ? {
         r := ""
          while r ||:= tab(upto('%')) do {
              move(1)
               code := integer("16r"||move(2)) | fail
               r ||:= char(code)
               }
        return r || tab(0)
        }
end
```

Problem 8: (12 points)

In this problem you are to write a graphical program that repeatedly sweeps out a circle and then erases it. It takes one minute to sweep out the circle and one minute to erase it. There should be an update once per second.

```
procedure main()
    WOpen("size=200,200")

    c := create |WAttrib("reverse="||("on"|"off"))
    ticks := 1
    repeat {
        every 1 to 60 do {
            WDelay(1000)
            FillCircle(100, 100, 100, -&pi/2,
                                          (2*&pi)*(ticks/60.0))
            ticks +:= 1
            ticks %:= 60
            while *Pending() > 0 do
                Event() === "q" & exit()
            }
        @c
        }
end
```

Problem 9: (20 points)

In this problem you are to implement four classes in Unicon...

```
class Eater(capacity, current_amt, consumed)
    method eat(x)
        if type(x) == "string" then {
            put(consumed, x)
            amt := *x
            }
        else {
            amt := x.units()
            put(consumed, x.what())
            }
        current_amt +:= amt
        until current_amt <= capacity do {
            write("Burp!")
            current_amt -:= capacity
            }
    end
    method eaten()
        every write(!consumed)
    end
    initially (cap: integer)
        capacity := cap
        current_amt := 0
        consumed := []
end

class Food(_what, _units)
    method units()
        return _units
    end
    method what()
        return _what
    end
    initially(w:string, u:integer)
        _what := w
        _units := u
end

class Burger: Food()
    initially
        self$Food.initially("Burger", 25)
end

class Fries: Food()
    initially
        self$Food.initially("Fries", 15)
end
```

Problem 10: (3 points)

Comment intelligently on this statement: *In Icon, variables are implicitly declared upon their first use. For example, if the variable* `x` *has not been previously used, the statement* `x := 3` *declares that the type of the variable* `x` *is an integer, just like '*`int x = 3;`*' in Java.*

The problem with the statement is that `x := 3` does not declare a type for `x`—variables in Icon have no type!

Problem 11: (4 points each; 28 points total; answer 7 of 13)

**Proposed change**: *Don't allow the names of built-in Icon functions, such as* `pos`*, to be used as variables.*

Question: Unintentional assignments to built-in functions can certainly cause some headaches but in some cases it's very handy. Present a set of language additions/modifications that simultaneously and conveniently accommodates these three different schools of thought: (1) I like things the way they are. (2) Never let me assign to the name of a built-in procedure. (3) Don't let me assign to a built-in unless I specifically indicate I want to.

> This could be achieved by adding a file-scope declaration such as "`immutable procedures`" and a syntax to force assignment, such as `write !:= x`.
>
> Some of you may have noticed that Unicon warns about assignments to built-ins.

**Proposed change**: `write()` *should return the full string it printed, not just the last argument.*

Question: What would be a negative impact of this change, and why?

> That would require a lot of strings to be built that would be never be used. In other words, it would create a lot of memory throughput.

**Proposed change**: *Icon has too many operators. I cannot think of any other language with so many operators to remember. At some point it's more useful to have functions than to make the programmer use a cheat-sheet.*

Question: Specify four operators that it would be good to replace with functions and suggest a function name for each that's easier to remember than the symbolic operator.

> Here are four that seem reasonable to me:
>
> ```
> @    activate
> ^    refresh
> ?    random
> %    mod
> ```
>
> Commonly used operators such as * were not favorably viewed when grading.

**Proposed change**: *When you try to assign a value outside a list's boundaries, the list should automatically expand itself and fill any gaps created with nulls. For example (`L[3] := "c"` should create a list L of `[null, null, "c"]`).*

Question: For the expression `L[N]`, what's a value of N that would pose a problem with the above rule, and why? (Hint: For full credit you must think of an issue that's at least as good as the one the instructor has in mind.)

> That's a good question! When I first wrote it I had negative numbers in mind, and that's the answer that got full credit, but perhaps there is a reasonable interpretation for a negative N.

**Proposed change**: *The `repeat` and `until-do` loops should not be included in the language. Although they are considered "syntactic sugar", their functionality can be modeled using the regular while loop with minor changes.*

Question: Present a strong argument that `until-do` is a worthwhile element of the language.

> It's easier to reason about
>
> ```
> until f() do ...
> ```
>
> than
>
> ```
> while not f() do ...
> ```

**Proposed change**: *Unicode support, i.e., support for 16-bit characters should be added to Icon. Characters shouldn't be limited to the English character set.*

Question: What element of the language would be most severely impacted, in terms of both speed and space, by adding Unicode support. Why?

> Character sets would balloon from eight 32-bit words to 2048 32-bit words and there would be a consequent increase in processing time.

**Proposed change**: *The `*` operator shouldn't show the number of results already generated by a co-expression, instead it should be the total number of results that the co-expression will generate.*

Question: Cite a co-expression with a finite result sequence for which the number of results it will produce can not be predicted.

```
create read()
```

**Proposed change**: *Icon permits the use of the `insert`, `delete` and `member` functions for sets and tables. `delete` should be supported for lists as well.*

Question: With the idea of being able to delete from lists in mind, what's an additional aspect of the behavior of `delete` that would need to be specified?

> Whether `delete(L,x)` should delete all occurrences of x or just one.

**Proposed change**: *Any type determination that can be made at compile-time should be made to improve runtime performance.*

Question: Give a specific example of how compile-time type determination can be used to reduce execution time.

> Given the expression `x + 5`, if we know that `x` is always an integer, we can simply fetch the value and add five to it. As is, we must first check the type of `x` and then take appropriate action.

**Proposed change**: *`member()`, `insert()`, and `delete()` should work on both sets and csets.*

Question: That might create a pitfall for a newcomer to Icon. What is it?

> Because csets have value semantics a function can't change the value of a cset—it can only produce a new cset. But the newcomer might expect this:

```
cs := 'abcd'
delete(cs, 'a')
```

> to delete the '`a`' from `cs`.

**Proposed change**: *You should be able to specify an initializing value for a global variable. For example, `global whitespace := ' \t'`*

Question: Depending on the implementation of this feature, a potential inconsistency could be created. What is that potential inconsistency?

> Global declarations in two files might initialize the same variable to different values.

**Proposed change**: *Give lists and sets value semantics.*

Question: What's meant by that statement? (Be sure to include an example involving some code.)

Lists, for example, would be treated as values. A comparison such as `L1 === L2` would succeed if the two lists are congruent and have identical values in corresponding positions. As is, that comparison simply tests to see whether `L1` and `L2` reference the same object in memory.

**Proposed change**: *Lists should be invocable. The invocation `[proc, a, b](c, d)` would be equivalent to `proc(a, b, c, d)`, and `[p]()` would be like `p()`, for example.*

Question: Assuming the presence of list invocation as described above, provide an implementation of `partial` that would produce a suitable value for `replx`.

```
procedure partial(a[])
    return a
end
```

(The cost of that question was to understand it...)

## EXTRA CREDIT SECTION

(1 point) List the last names of ten other students in this class. Use of phonetic spelling is acceptable.

Almost everybody got this question on their third, and final, attempt. One student named all fifteen others.

(1 point) To the best of the instructor's knowledge, the last time a class at the University of Arizona covered Icon's graphics facilities was in the mid-1990s. In general, students then had a very positive response to the graphics facilities. The response this semester, particularly to VIB and vidgets, was not so warm. Why?

The graphics programming landscape was relatively bleak at that time.

(1 point) Write a routine `which(x)` that returns "`string`" or "`list`", depending on whether `x` is a string or list. You may not use `type()` or `[Ii]mage()`. You may assume that `x` is either a string or a list.

```
procedure which(x)
    return if x === copy(x) then "string"
                            else "list"
end
```

Another central test is '`if x[1:1] === "" then...`'.

(1 point) The text list vidget provides no way to indicate that there was a double-click on a item. Describe a technique that would allow a program to respond to a double-click on a text list item. The text list vidget must be used as-is. (If you tried to do this on your project but it didn't work out, briefly describe what you tried.)

Have code in the callback routine that measures the time between clicks.

(1 point) Write a SNOBOL4 program that reads lines from standard input and prints the first and last line on standard output. Recall that referencing the variable `INPUT` causes a line to be read and that assigning a value to the variable `OUTPUT` causes a line to be written.

```
      OUTPUT = INPUT
LOOP  LAST = INPUT :S(LOOP)
      OUTPUT = LAST
END
```