Name:________________________________________

# CSc 451, Spring 2003
# Examination #2
# April 17, 2003

## READ THIS FIRST

**Fill in your name above. Do not turn this page until you are told to begin.**

DO NOT use your own paper. If you run out of room, write on the back of the page.

If you have a question that can be safely resolved by making an assumption, simply write down that assumption and proceed. Examples:

> "Assuming `reverse(s)` reverses a string"
> "Assuming `*t` returns the number of keys in table `t`"

If you have a question you wish to ask, raise your hand and the instructor will come to you. DO NOT leave your seat.

You may use all elements of Icon.

You may use Icon procedures that have appeared on the slides, or been presented in class, or been mentioned in e-mail, or that have appeared on an assignment this semester, or that are mentioned in this exam.

There are no deductions for poor style. Anything that works and meets all problem-specific restrictions will be worth full credit, but try to keep your solutions brief to save time.

You need not include any explanation in an answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it. If you're completely puzzled on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that will certainly earn no credit.

This is a **sixty** minute exam with a total of 100 points and 5 possible points of extra credit. There are seven regular problems and an extra credit section with three questions.

When you have completed the exam, enter your name on the exam sign-out log and then hand your exam to the instructor.

**When told to begin, double-check that your name is at the top of this page,** and then put your initials in the lower right hand corner of each page, being sure to check that you have all ten pages.

Problem 1: (20 points)

Write a program that opens a 300 x 300 window and permits the user to draw circles of varying size and color, as follows:

(1) The program starts in "placement mode". A left-click draws a circle of radius 1 centered at the position of the click and puts the program into "sizing mode".

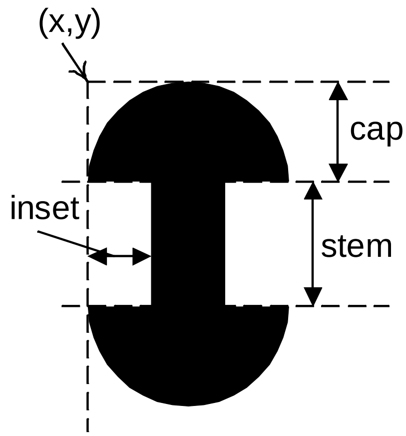
(2) In sizing mode, typing a "`+`" causes the radius of the circle to grow by one; a "`-`" minus causes it to shrink by one. Thus the user can type a series of `+`'s and `-`'s to adjust the size of the circle. Typing a "`.`" (period) permanently sets the size and position of the circle and returns the program to placement mode. Other events are ignored in sizing mode. Don't worry about the radius possibly going negative.

(3) In placement mode right-clicks cycle the current color through red, green, and blue repeatedly. The initial color is red.

(4) A "`q`" in placement mode exits the program.

Problem 2: (20 points)

Write a procedure `FillGizmo(x, y, cap, stem, inset)` that draws the following gizmo at the coordinates (`x`, `y`).




The gizmo is symmetrical around both the X and Y axes. The rounded top and bottom elements are semicircles with radius `cap`. Note that the width of the figure is `cap*2`.

Draw the figure in the current foreground color.

Your implementation may change window attributes, but be sure it restores them to their original values before it returns.

Don't be concerned with off-by-one-pixel errors. Note that there is no `FillSemiCircle`() routine in the library.

Problem 3: (12 points)

**RESTRICTIONS FOR THIS PROBLEM: (1) You may not 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.**

Write a procedure `format(fmt, v[])` that does simple `printf`-like formatting of the values in `v` based on the specifications in the string `fmt`. It returns the resulting string.

`format` has three formatting specifiers:

| | |
|---|---|
| `%v` | Do no conversion on the value; simply concatenate it to the result in progress |
| `%i` | Call `image()` for the value and concatenate the result. |
| `%I` | Call Image() for the value and concatenate the result. |

Examples:

```
][ format("x = %v, y = %v, z = %v", 10, 20, 30);
   r := "x = 10, y = 20, z = 30"  (string)

][ format("%v%v%v", 1, 200, 4000);
   r := "12004000"  (string)

][ format("list(10) = %i, table(0) = %i", list(10), table(0));
   r := "list(10) = list_22(10), table(0) = table_5(0)"  (string)

][ write(format("L = %i (%I)", [1,2,3], [1,2,3]));
L = list_25(3) (L6:[
  1,
  2,
  3])
   r := "L = list_25(3) (L6:[\n  1,\n  2,\n  3])"  (string)

][ format("10 < 20 is %v", 10 < 20);
   r := "10 < 20 is 20"  (string)

][ format("10 > 20 is %v", 10 > 20);
Failure
```

Note that `format` returns a string result; it does no output.

Assume that:

> There will be at least as many values as formatting specifiers, i.e., there won't be a call like `format("%v")`.
>
> The formatting specification is well-formed.
>
> Values corresponding to `%v` specifiers can be converted to a string, i.e., there won't be a call like `format("%v", [])`.

There is space for your solution on the next page.

(SPACE FOR SOLUTION FOR PROBLEM 3)

**DON’T FORGET THE RESTRICTIONS!**

Problem 4: (8 points)

**RESTRICTIONS FOR THIS PROBLEM : Same as Problem 3.**

Write a program `vc` that reads lines on standard input and prints those lines that contain more vowels than consonants.

Example:

```
% cat vc.1
Would
you
believe
an abalone
wrote this?
% vc < vc.1
you
believe
an abalone
%
```

Problem 5: (25 points)

**RESTRICTIONS FOR THIS PROBLEM : Same as Problem 3.**

Write a procedure `dollars(s)` that converts a string specifying a sum of money into a corresponding `real` value.

The sum may be specified in three ways:

(1) One or two digits followed by a "c" (always lower case), such as "`8c`" or "`99c`".

(2) A dollar sign, one or more digits, a period, and then two digits.

(3) English such as "`one dollar`" or "`three bucks`", up to twenty dollars. Grammatically incorrect forms such as "one bucks" or "seven dollar" are not allowed. The units will be either "buck(s)" or "dollar(s)". Only lower case is permitted.

If the specification is valid, a real is returned. If the specification is not valid the procedure fails.

Examples of valid conversions:

```
][ dollars("8c");
   r := 0.08  (real)

][ dollars("99c");
   r := 0.99  (real)

][ dollars("$1.25");
   r := 1.25  (real)

][ dollars("$19985.99");
   r := 19985.99  (real)

][ dollars("one buck");
   r := 1.0  (real)

][ dollars("ten bucks");
   r := 10.0  (real)

][ dollars("nineteen dollars");
   r := 19.0  (real)
```

Here are calls that fail:

```
dollars("10cx")       # trailing "x"
dollars("  $2.34")    # leading blanks
dollars("100c")       # more than two digits with "c"
dollars("$.25")       # no digits before decimal point
dollars("$1.2")       # should be two digits after decimal point
dollars("$100.000")   # should be two digits after decimal point
dollars("ten buck")   # should be "ten bucks"
dollars("one dollars") # should be "one dollar"
dollars("10")         # no unit specified
```

No whitespace may appear except between the number and the unit in the english specification.

**IMPORTANT**: You may assume the presence of a procedure `english(n)` that converts the integer `n` to English. For example, `english(1)` returns "`one`"; `english(20)` returns "`twenty`".

**DON'T FORGET THE RESTRICTIONS!**

Problem 6: (9 points)

Write a PDCO `Longest{`*`expr1, expr2, ..., exprN`*`}` that returns the argument with the longest result sequence. In the case of a tie, one of the tying expressions is produced, but which one is not specified. The call `Longest{}` (i.e, with no arguments) fails. You may assume that no expression has an infinite result sequence.

Examples:

```
][ c := Longest{1 to 3, !"abcde", 1 < 0};
   r := co-expression_29(0)  (co-expression)

][ while write(@c);
a
b
c
d
e
Failure

][ c := Longest{1 < 0, 2};
   r := co-expression_39(0)  (co-expression)

][ while write(@c);
2
Failure

][ c := Longest{3 = 4, !"", [] === []};
   r := co-expression_45(0)  (co-expression)

][ write(@c);
Failure
```

Hint: The instructor's solution uses `sortf(L, sort_by_posn)`.

Problem 7: (2 points each; 6 points total)

(a) What is the fundamental difference between an additive color model and a subtractive color model?

(b) Name a subtractive color model and the colors it uses.

(c) What "color" would be produced by the setting `Fg("#bababa")`?

**EXTRA CREDIT SECTION**

(a) (1 point) What is the shortest Icon program that will successfully compile and execute without error? (Hint: It easily fits in the space provided.)

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

(c) (3 points) An Icon programmer homesick for Java wants to produce output with `System.out.println()` instead of `write()`. Create a file `java.icn` that provides the necessary elements to meet the needs of this misdirected individual until professional help can be obtained.

Here is a sample program that should work:

```
link java
procedure main()
    System.out.println("Numbers:")
        every System.out.println(1 to 10)
end
```

If your solution needs it, you may require that the user call `java_init()` before using `System.out.println()`.