Benchmarking Version 8 of Icon

Ralph E. Griswold

Department of Computer Science, The University of Arizona

Benchmarks of Icon programs provide interesting comparisons of the performance of different computer systems [1].

A suite of representative Version 8 Icon programs has been assembled to provide uniform benchmarks over the range of computers on which Icon has been implemented. Tools are provided so that testing is largely automatic.

The benchmark programs do not require any ''optional'' features, such as co-expressions, and they work with the same regions sizes on implementations of Icon with either fixed or expandable memory regions. Input and output normally are suppressed to avoid factors like disk speed from affecting the results.

The benchmark programs, taken from the Icon program library [2], are:

concord.icn	Simple word concordance; string analysis and synthesis with table manipulation.
deal.icn	Randomly selected bridge hands; string synthesis with mapping.
ipxref.icn	Icon program cross reference; string analysis and synthesis with list manipulation.
queens.icn:	Solutions to the non-attacking n-queens problem; goal-directed evaluation and string synthesis.
rsg.icn:	Random sentence generation; string synthesis with list and table manipulation.

The procedures that are used to support benchmarking are listed in Appendix A. A Makefile for running the benchmarks is listed in Appendix B.

The benchmark suite is available in a variety of formats for different computer systems. It includes a form for reporting results to the Icon Project [3].

References

1. R. E. Griswold and M. T. Griswold, *Icon Newsletter 31*, Nov. 1989.
2. R. E. Griswold, *The Icon Program Library*, The Univ. of Arizona Tech. Rep. 90-7, 1990.
3. R. E. Griswold, *Version 8 Icon Benchmark Report*, The Univ. of Arizona Icon Project Document IPD116, 1989.

Appendix A — Support Procedures

```
############################################################################
#
#  Support procedures for Icon benchmarking.
#
############################################################################
#
#      The code to be times is bracketed by calls to Init__(name)
#  and Term__(), where name is used for tagging the results.
#  The typical usage is:
#
#        procedure main()
#           [declarations]
#           Init__(name)
#                 .
#                 .
#                 .
#           Term__()
#        end
#
#      If the environment variable OUTPUT is set, program output is
#  not suppressed.
#
############################################################################

global Save__, Saves__, Name__

# List information before running.
#
procedure Init__(prog)
   Name__ := prog                       # program name
   Signature__()                        # initial information
   Regions__()
   Time__()
   if getenv("OUTPUT") then { # if OUTPUT is set, allow output
      write("*** Benchmarking with output ***")
      return
      }
   Save__ := write                      # turn off output
   Saves__ := writes
   write := writes := 1
   return
end
```

```
# List information at termination.

procedure Term__()
   if not getenv("OUTPUT") then {      # if OUTPUT is not set, restore output
      write := Save__
      writes := Saves__
      }
                                       # final information
   write(Name__," elapsed time = ",Time__())
   Regions__()
   Storage__()
   Collections__()
   return
end

# List garbage collections performed.
#
procedure Collections__()
   static labels
   local collections

   initial labels := ["total","static","string","block"]

   collections := []
   every put(collections,&collections)
   write("collections")
   every i := 1 to *labels do
      write(labels[i],right(collections[i],8))
   return
end

# List region sizes.
#
procedure Regions__()
   static labels
   local regions

   initial labels := ["static","string","block"]

   regions := []
   every put(regions,&regions)
   write("regions")
   every i := 1 to *labels do
      write(labels[i],right(regions[i],8))
   return
end
```

```
# List relveant implementation information
#
procedure Signature__()
    write(&version)
    write(&host)
    every write(&features)
    return
end

# List storage used.
#
procedure Storage__()
    static labels
    local storage

    initial labels := ["static", "string", "block"]

    storage := []
    every put(storage, &storage)
    write("storage")
    every i := 1 to *labels do
        write(labels[i], right(storage[i],8))
    return
end

# List elapsed time.
#
procedure Time__()
    static lasttime

    initial lasttime := &time
    return &time - lasttime
end
```

Appendix B — Makefile for Benchmarking

```
############################################################################
#
#  Makefile for Version 8 Icon benchmarking.
#
############################################################################
#
#      In order for benchmark results to be compared meaningfully with
#  those from other systems, the string and block regions must be set to
#  65,000 bytes. This is the normal default.
#
#      To run the benchmarks, use
#
#        make benchmark
#
#  This creates .out files with benchmark results and lists the timings.
#
#      On systems where timing varies with load or other factors, use
#
#        make rerun
#
#  which reruns the benchmarks and appends the results to the .out files.
#
############################################################################
#
#      Program output normally is suppressed. To get program output, set
#  the environment variable OUTPUT.  The "expected" output (modulo
#  timing differences), is in files .std for comparison.  (These files
#  are not included with all disributions because of their large size.)
#
############################################################################

SHELL=/bin/sh

what:
                @echo "What do you want to make?"

benchmark:      # do the whole thing
                make translate compile run check

translate:      # create ucode files for linking
                icont -s -c post
                icont -s -c options
                icont -s -c shuffle

compile:        # compile the benchmark programs
                icont -s concord
                icont -s deal
                icont -s ipxref
                icont -s queens
                icont -s rsg
```

```
run:    # run the programs
        echo Running concord ...
        iconx concord <concord.dat >concord.out
        echo Running deal ...
        iconx deal -h 500 >deal.out
        echo Running ipxref ...
        iconx ipxref <ipxref.icn >ipxref.out
        echo Running queens ...
        iconx queens -n9 >queens.out
        echo Running rsg ...
        iconx rsg <rsg.dat >rsg.out

rerun:  # rerun the benchmarks
        echo Running concord ...
        iconx concord <concord.dat >>concord.out
        echo Running deal ...
        iconx deal -h 500 >>deal.out
        echo Running ipxref ...
        iconx ipxref <ipxref.icn >>ipxref.out
        echo Running queens ...
        iconx queens -n9 >>queens.out
        echo Running rsg ...
        iconx rsg <rsg.dat >>rsg.out

check:
        grep elapsed *.out
```