ICFP Programming Contest 2011: Official Site

FRIDAY, JUNE 17, 2011

Task description - contest starts now!

Welcome to the ACM SIGPLAN ICFP Programming Contest 2011! The task this year is to write a program that plays the card game *Lambda: The Gathering* (LTG for short).

Rules

Each *match* of LTG is played by two programs (player 0 and player 1) in alternate *turns* (turn 1 of player 0, turn 1 of player 1, turn 2 of player 0, turn 2 of player 1, etc.). Each player owns 256 *slots*, numbered from 0 to 255, and a fixed set of *cards*. A slot consists of a *field* and an integer called *vitality*, ranging from -1 to 65535. A field holds a *value*, which is either an integer, ranging from 0 to 65535, or a *function* (in the sense of programming languages). A value is a *valid slot number* if it is an integer greater than or equal to 0, and less than or equal to 255. A slot is *dead* if its vitality is 0 or -1; otherwise the slot is *alive*. At the start of each match, every field is initialized to the identity function (a function that takes an argument x and returns x) and every vitality is initialized to 10000. The cards have fixed values as defined below.

In each turn, the player (called the *proponent*) performs either a *left application* or a *right application*. A left application applies (as a function) a card to the field of one of the proponent's slots. A right application applies (as a function) the field of one of the proponent's slots to a card. In either case, an *error* is raised if the card or the field to be applied is not a function, or if the slot is dead. The other player (called the *opponent*) is able to see which application the proponent has chosen on what card and slot. [Remark: In mathematics and programming languages (as well as in our rules), one applies a function to an argument, not vice versa; that is, "applying f to x" means "f(x)", not "x(f)".] [Remark: As a result of the rules, all function applications in LTG are "call by value": that is, the argument x of function application f(x) is always a value.]

The turn ends when an error is raised, when the number of function applications caused by the left or right application (including itself) exceeds 1000, or when the left or right application returns a value without exceeding this limit. In the last case, the field of the slot used for the left or right application is overwritten with the return value. In the other cases, it is overwritten with the identity function. Effects caused by function applications are *not* rewound and remain even if an error is raised or the application limit is exceeded. Cards are *not* consumed by applications and can be reused as many times as necessary.

A match ends after 100000 turns of each player, or when every slot of a player has become dead after a turn. In either case, a player wins if it has more slots alive than the other player. The players tie if the numbers of slots alive are equal.

Cards

The set of cards are fixed as follows. The effect of a card is undefined in any case not specified below. [Remark: The images are only for amusement and are not part of the official rules.]

Card "I" is the identity function. [Remark: It is called the I combinator and written λx.x in lambda-calculus.]

WHAT'S IN AN IMAGE?

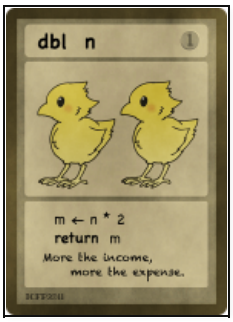
BLOG ARCHIVE

▼ 2011 (10)
▼ June (3)
Task description - contest starts now!
How to prepare an environment for testing your sub...
Contest starting in two weeks
► May (3)
► April (1)
► March (3)

Card "zero" is an integer constant 0.

Card "succ" is a function that takes an argument n and returns n+1 if n<65535 (or returns 65535 if n=65535). It raises an error if n is not an integer.

Card "dbl" is a function that takes an argument n and returns n*2 (n times 2) if n<32768 (or returns 65535 if n>=32768). It raises an error if n is not an integer.



Card "get" is a function that takes an argument i and returns the value of the field of the ith slot of the proponent if the slot is alive. It raises an error if i is not a valid slot number or the slot is dead.

Card "put" is a function that takes an (unused) argument and returns the identity function.

Card "S" is a function that takes an argument f and returns another function, which (when applied) will take another argument g and return yet another function, which (when applied) will take yet another argument x, apply f to x obtaining a return value h (or raise an error if f is not a function), apply g to x obtaining another return value y (or raise an error if g is not a function), apply h to y obtaining yet another return value z (or raise an error if h is not a function), and return z. [Remark: The first function is called the S combinator and written λf.λg.λx.fx(gx) in lambda-calculus.]

Card "K" is a function that takes an argument x and returns another function, which (when applied) will take another (unused) argument y and return x. [Remark: The first function is called the K combinator and written λx.λy.x in lambda-calculus.]

Card "inc" is a function that takes an argument i, and

- increases by 1 the vitality v of the ith slot of the proponent if v>0 and v<65535,
- does nothing if v=65535 or v<=0, or
- raises an error if i is not a valid slot number,

and returns the identity function.

Card "dec" is a function that takes an argument i, and

- decreases by 1 the vitality v of the (255-i)th slot of the opponent if v>0,
- does nothing if v<=0, or
- raises an error if i is not a valid slot number,

and returns the identity function.

Card "attack" is a function that takes an argument i and returns another function, which (when applied) will take another argument j and return yet another function, which (when applied) will take yet another argument n, decrease by n the vitality v of the ith slot of the proponent (or raise an error if i is not a valid slot number, n is not an integer, or n is greater than v), and

- decrease by n*9/10 (n times 9 divided by 10, with the remainder discarded) the vitality w of the (255-j)th slot of the opponent if it is alive (w is set to 0 if it would become less than 0 by this decrease),
- do nothing if the slot is dead, or
- raise an error if j is not a valid slot number,

and return the identity function.

Card "help" is a function that takes an argument i and returns another function, which (when applied) will take another argument j and return yet another function, which (when applied) will take yet another argument n, decrease by n the vitality v of the ith slot of the proponent (or raise an error if i is not a valid slot number, n is not an integer, or n is greater than v), and

- increase by n*11/10 (n times 11 divided by 10, with the remainder discarded) the vitality w of the jth slot of the proponent if it is alive (w is set to 65535 if it would become greater than 65535 by this increase),
- do nothing if the slot is dead, or
- raise an error if j is not a valid slot number,

and return the identity function.

Card "copy" is a function that takes an argument i, and returns the value of the field of the ith slot of the opponent. It raises an error if i is not a valid slot number. Note that the slot is ith, not (255-i)th.

Card "revive" is a function that takes an argument i, sets to 1 the vitality v of the ith slot of the proponent if v<=0 (or does nothing if v>0), and returns the identity function. It raises an error if i is not a valid slot number.

Card "zombie" is a function that takes an argument i and returns another function, which (when applied) will take another argument x, and

- overwrite with x the field of the (255-i)th slot of the opponent if the slot is dead, or
- raise an error if i is not a valid slot number or the slot is alive,

and set the vitality of the slot to -1, and return the identity function.

Immediately before each turn of a player, the field of every slot of the player with vitality -1 is automatically applied (as a function) to the identity function (even though the slot is dead) in increasing order of slot number. For each of these automatic applications, an error is raised if the field is not a function or the number of function applications caused by the application exceeds 1000. An error caused by each automatic application does not affect any other automatic applications. After each automatic application, the field of the slot used for the application is overwritten with the identity function, and the vitality of the slot is reset to 0.

In addition, during the above automatic applications, parts of the effects of 4 cards change from their previous descriptions as follows (the other parts do not change from the original):

- Card "inc" decreases the v (in the previous description of this card) by 1 if v>0, or does nothing if v<=0.
- Card "dec" increases the v by 1 if v>0 and v<65535, or does nothing if v<=0 or v=65535.
- The third function in card "attack" increases the w by n*9/10 if w>0 (w is set to 65535 if it would become greater than 65535 by this increase), or does nothing if w<=0.
- The third function in card "help" decreases the w by n*11/10 if w>0 (w is set to 0 if it would become less than 0 by this decrease), or does nothing if w<=0.

[Remark: For an informational purpose only, executables for interactive LTG plays are provided at:

http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.linux32.gz
http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.linux64.gz
http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.win32.exe.zip
http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.macosx-10.4-ppc.gz (also for 10.5-ppc)
http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.macosx-10.5-intel.gz
http://www.kb.ecei.tohoku.ac.jp/icfpc/ltg.macosx-10.6-intel.gz

It is forbidden to use any of these executables as any part of your submission (such programs would be disqualified). Note also that these executables are not part of the official rules. The contest organizers do not provide any guarantee or support for them, although reports of bugs and problems are welcome if any. A simple example session (with alternate turns by two players) is shown below (details of the informational output may differ depending on versions).

```
$ ./ltg
Lambda: The Gathering version $Date:: 2011-06-15 18:44:34 +0900#$
Usage: ./ltg <options> only
         (one player; continues forever)
       ./ltg <options> alt
         (two players; continues forever)
       ./ltg <options> match prog1 prog2
         (similar to official match)

  -non-blocking <true|false>  Non-blocking I/O on pipes to prog1
and prog2
                              (default: false on Win32, true othe
rwise;
                               does not work on Win32)
  -silent <true|false>  No output; exit status only (default: fal
se)
  -print-fields <true|false>  Print fields (true, default) or not
 (false)
  -help  Display this list of options
  --help  Display this list of options
$ ./ltg alt
Lambda: The Gathering version $Date:: 2011-06-15 18:44:34 +0900#$
###### turn 0
*** player 0's turn, with slots:
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 0 applied slot 0 to card zero
*** player 1's turn, with slots:
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
inc
player 1 applied slot 0 to card inc
###### turn 2
*** player 0's turn, with slots:
0={10000,zero}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
succ
slot no?
0
player 0 applied card succ to slot 0
*** player 1's turn, with slots:
0={10000,inc}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 1 applied slot 0 to card zero
###### turn 4
*** player 0's turn, with slots:
0={10000,1}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
succ
slot no?
0
player 0 applied card succ to slot 0
*** player 1's turn, with slots:
0={10001,I}
```

```
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
dec
player 1 applied slot 0 to card dec
###### turn 6
*** player 0's turn, with slots:
0={10000,2}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
dbl
slot no?
0
player 0 applied card dbl to slot 0
*** player 1's turn, with slots:
0={10001,dec}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 1 applied slot 0 to card zero
###### turn 8
*** player 0's turn, with slots:
0={10000,4}
255={9999,I}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
inc
slot no?
0
player 0 applied card inc to slot 0
*** player 1's turn, with slots:
0={10001,I}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
succ
slot no?
0
player 1 applied card succ to slot 0
Exception: Native.Error
slot 0 reset to I
###### turn 10
*** player 0's turn, with slots:
4={10001,I}
255={9999,I}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
```

Another example session (with unilateral turns by a solitary player, which is not part of the official rules) follows, again for an informational purpose only. It shows how to compose (as functions) the field of a slot with a card, as well as how to apply (as a function) the field of a slot to that of another.

```
$ ./ltg only
Lambda: The Gathering version $Date:: 2011-06-15 18:44:34 +0900#$
###### turn 0
*** player 0's turn, with slots:
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
```

```
help
player 0 applied slot 0 to card help
###### turn 1
*** player 0's turn, with slots:
0={10000,help}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 0 applied slot 0 to card zero
###### turn 2
*** player 0's turn, with slots:
0={10000,help(zero)}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
K
slot no?
0
player 0 applied card K to slot 0
###### turn 3
*** player 0's turn, with slots:
0={10000,K(help(zero))}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
S
slot no?
0
player 0 applied card S to slot 0
###### turn 4
*** player 0's turn, with slots:
0={10000,S(K(help(zero)))}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
succ
player 0 applied slot 0 to card succ
###### turn 5
*** player 0's turn, with slots:
0={10000,S(K(help(zero)))(succ)}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 0 applied slot 0 to card zero
###### turn 6
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
1
card name?
zero
player 0 applied slot 1 to card zero
###### turn 7
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,zero}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
```

```
succ
slot no?
1
player 0 applied card succ to slot 1
###### turn 8
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,1}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
dbl
slot no?
1
player 0 applied card dbl to slot 1
###### turn 9
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,2}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
dbl
slot no?
1
player 0 applied card dbl to slot 1
###### turn 10
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,4}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
dbl
slot no?
1
player 0 applied card dbl to slot 1
###### turn 11
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,8}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
dbl
slot no?
1
player 0 applied card dbl to slot 1
###### turn 12
*** player 0's turn, with slots:
0={10000,help(zero)(1)}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
K
slot no?
0
player 0 applied card K to slot 0
###### turn 13
*** player 0's turn, with slots:
0={10000,K(help(zero)(1))}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
S
slot no?
0
player 0 applied card S to slot 0
```

```
###### turn 14
*** player 0's turn, with slots:
0={10000,S(K(help(zero)(1)))}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
get
player 0 applied slot 0 to card get
###### turn 15
*** player 0's turn, with slots:
0={10000,S(K(help(zero)(1)))(get)}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
K
slot no?
0
player 0 applied card K to slot 0
###### turn 16
*** player 0's turn, with slots:
0={10000,K(S(K(help(zero)(1)))(get))}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
1
card name?
S
slot no?
0
player 0 applied card S to slot 0
###### turn 17
*** player 0's turn, with slots:
0={10000,S(K(S(K(help(zero)(1)))(get)))}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
succ
player 0 applied slot 0 to card succ
###### turn 18
*** player 0's turn, with slots:
0={10000,S(K(S(K(help(zero)(1)))(get)))(succ)}
1={10000,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 0 applied slot 0 to card zero
###### turn 19
*** player 0's turn, with slots:
0={9984,I}
1={10017,16}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
```

The last example shows how to implement an infinite loop.

```
$ ./ltg only
Lambda: The Gathering version $Date:: 2011-06-15 18:44:34 +0900#$
###### turn 0
*** player 0's turn, with slots:
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
```

```
2
slot no?
0
card name?
S
player 0 applied slot 0 to card S
###### turn 1
*** player 0's turn, with slots:
0={10000,S}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
get
player 0 applied slot 0 to card get
###### turn 2
*** player 0's turn, with slots:
0={10000,S(get)}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
I
player 0 applied slot 0 to card I
###### turn 3
*** player 0's turn, with slots:
0={10000,S(get)(I)}
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
2
slot no?
0
card name?
zero
player 0 applied slot 0 to card zero
Exception: Native.AppLimitExceeded
slot 0 reset to I
###### turn 4
*** player 0's turn, with slots:
(slots {10000,I} are omitted)
(1) apply card to slot, or (2) apply slot to card?
```

It is known that (closed) *lambda-terms* can be translated into function applications of S, K, and I (and I can also be implemented by S and K). A lambda-term M is either a variable x, a function application M(N) (often written MN without the parentheses), or a function λx.N that takes an argument x, evaluates another lambda-term N and returns its value. A translation T from (closed) lambda-terms to S, K, and I can be given as follows:

```
T(λx.M) = A(x,T(M))
T(MN) = T(M) T(N)
T(x) = x

where

A(x,x) = I
A(x,y) = K y   (when y is different from x)
A(x,MN) = S (A(x,M)) (A(x,N))
```

Lambda-terms are closed when they have no "undefined variables". For example, the lambda-term "λx.λy.x" is closed, while "λy.x" is not. For more details on lambda-calculus, please see the literature and/or search the WWW (though these are not required for participating in this contest).]

Submission

Player programs must be submitted via:

https://spreadsheets.google.com/spreadsheet/viewform?formkey=dDRFOXgxQ3g2M3I3X1VTV3FBREdULWc6MQ
(or http://goo.gl/FTP9F)

Each submission must be a single .tar.gz compressed archive file containing (at least)

two executable files "./install" and "./run", and a "./src/" directory (all without the quotes). The "install" file will be executed just once with no human interaction after the submission is expanded into the home directory of a unique general user. The execution environment has been announced at:

http://www.icfpcontest.org/2011/05/judges-machine-and-system-environment.html
(or http://goo.gl/2DBgy)

Please try to minimize the installation procedure: for example, avoid compiling your program from source code and include a precompiled binary as far as possible.

The "run" file will be executed for each match with a command-line argument "0" (without the quotes, ditto below) or "1", meaning whether it is executed as player 0 or player 1 of the match. For each turn of the player, "run" must write to standard output either (as determined by the player) of:

- the digit "1" (meaning left application) and the line feed character '\n', followed by a case-sensitive string (meaning a card name) and '\n', again followed by a decimal integer from 0 to 255 (meaning a slot number) and '\n', and
- the digit "2" (meaning right application) and the line feed character '\n', followed by a slot number and '\n', again followed by a card name and '\n'.

For each turn of the opponent, "run" must read from standard input either (as determined by the opponent) of:

- the digit "1" (meaning left application) and the line feed character '\n', followed by a card name and '\n', again followed by a slot number and '\n', and
- the digit "2" (meaning right application) and the line feed character '\n', followed by a slot number and '\n', again followed by a card name and '\n'.

The input and output must be in ASCII code. Each output by the player must be performed within 60 seconds (in real time) after the last output from the opponent. In addition, for each match, the total CPU usage, the maximum resident set size, and the maximum disk usage of "run" must not exceed 10000 seconds, 512 MB, and 1 GB, respectively, including any descendant (child, grandchild, grand-grandchild, etc.) processes. Any files created by "run" during a match must be deleted by itself before or when it terminates. "run" must not communicate any information across matches. Submissions that are found to violate any of the above requirements would lose the match and/or be disqualified from the entire contest. [Remark: The judges also reserve the right to disqualify any submissions that we find are cheating and/or harmful; use common sense.]

[Remark: Although not mandatory, it is strongly urged that submissions are tested in advance on Debian squeeze for amd64 (which supports x86-64 including Intel 64). Details of the environment are described in:

http://www.icfpcontest.org/2011/05/judges-machine-and-system-environment.html
(or http://goo.gl/2DBgy)

If necessary, use some virtual machine software such as VirtualBox and VMware Player. See our unofficial information:

http://www.icfpcontest.org/2011/06/how-to-prepare-environment-for-testing.html
(or http://goo.gl/whrS7)

In case you can only produce 32-bit executables, please remember:

- Statically linked 32-bit Linux executables should run with no problem on the 64-bit environment;
- Dynamically linked 32-bit libraries should also be fine as long as they are included in the ia32-libs package of Debian; and
- other dynamically linked 32-bit libraries (and any other files required at runtime but unavailable on Debian squeeze for amd64) can be included in the submission itself.

]

The "src" directory must contain all the source code written for the submission. It does not have to include publicly available common software such as standard compilers and libraries. (The judges will not try to compile the source code, though winning programs will be examined more carefully.)

[Remark: Here is a (very) simple example of the "install" script:

```
#!/bin/sh
exit 0
```

with an example of the "run" program:

```
#!/bin/sh
opp()
{
  read lr
  case $lr in
    1) read card; read slot;;
    2) read slot; read card;;
  esac
}
if [ $1 = "1" ]; then
  opp
fi
while [ true ]; do
  echo "1"
  echo "I"
  echo "0"
  opp
done
```

]

Additionally, the archive may include an optional "./README" file, explaining the most interesting aspect(s) of the submission. This file will be used for determining the discretionary judges' prize.

Evaluation

Submitted programs are evaluated in two rounds. In the first round, each program plays as player 0 against n randomly chosen distinct programs as player 1. The number n will be at least 15 and determined by the judges considering the number of submissions and the actual time required for the matches. In each match, 6 (resp. 2) points are given to player 0 if it wins against player 1 within (resp. after) 100000 turns, and 1 point is given to player 0 in case of a tie (in this round, no point is given to player 1). The programs are ranked by the total points.

In the second round, the best 30 (or more, in case of ties) programs in the first round play against each of the other 29 (or more) programs to determine the 1st and 2nd places. In each match, 6 (resp. 2) points are given to a program if it wins against another program within (resp. after) 100000 turns, and 1 point is given to both programs in case of a tie. The programs are ranked by the total points from the second round (points from the first round are not included). [Remark: In case of ties for the 1st and/or 2nd places, the prize(s) will be divided equally. Details of this division are at the discretion of the judges.]

Judges' prize will be determined discretionarily on the basis of the optional README files and the submissions themselves.

Enjoy!

Questions about this task description can be posted as (moderated) comments to this post or e-mailed to icfpc2011-blogger AT kb.ecei.tohoku.ac.jp. Please use the latter if your question includes spoiler information.

[Remark: Use #icfpc2011 for Twitter and #icfp-contest (on chat.freenode.net) for IRC, though the contest organizers will not always watch them (so do not ask questions there). Do not publish or communicate spoilers anywhere, either (again, please use common sense). Thank you!]

[Remark: An ***unofficial*** duel server is available at http://kokako.kb.ecei.tohoku.ac.jp/. It is unofficial in the sense that uploading there does not mean a submission for the official contest, that details may differ from the official rules, and that its availability and correctness are not guaranteed (though reports of bugs and problems are welcome - please comment to this post or e-mail to icfpc2011-blogger AT kb.ecei.tohoku.ac.jp).

Posted by ICFPC2011 Organizers at 9:00 AM

11 comments:

Matthias Görgens said...

Cool problem! Thanks for coming up with it.

June 17, 2011 9:15 AM

Kragen Javier Sitaker said...

It says that a turn ends when an error is raised, and errors can be raised by automatic applications of slots with -1 vitality, but do not affect other automatic applications. Do these errors end the turn — i.e., can you prevent your opponent from playing by zombieing their slots with crashers?

June 17, 2011 10:20 AM

ICFPC2011 Organizers said...

No, as the task description says, the automatic applications of zombies happen _before_ a turn and do not prevent it.

June 17, 2011 11:21 AM

Anonymous said...

It's not clear how we can see what the opponent has performed during their turn.

June 17, 2011 11:24 AM

Anonymous said...

I know it says that card images are just for amusement, but it looks like the image for "put" card does not correspond to description.
The image says the put x y = return y, but description says that put takes only one argument (unused) and returns identity function, i.e. image should say put x = return I

June 17, 2011 11:30 AM

ICFPC2011 Organizers said...

Please read the section "Submission".

June 17, 2011 11:32 AM

ICFPC2011 Organizers said...

> I know it says that card images are just for amusement, but it looks like the image for "put" card does not correspond to description.

You are right. The task description is correct. We'll fix the image to avoid confusion. Thanks!

June 17, 2011 12:02 PM

Anonymous said...

I'd like to clarify `attack i j n' and `help i j n'. Each of these involves changes to some slot `i' and another slot `j'. If `i' is not a valid slot number, do the changes to slot `j' still take place before the exception is raised? If `j' is not a valid slot number, do the changes to slot `i' still take place before the exception is raised? Thanks.

June 17, 2011 12:06 PM

ICFPC2011 Organizers said...

Please read the task description carefully. The order of such effects are exactly described.

June 17, 2011 12:13 PM

Anonymous said...

It would help a lot if unofficial server could show logs. Currently my submission loses by invalid output at turn 4000+ and I can't figure out what's wrong, because all it does now is spamming the same command. :(

June 17, 2011 12:17 PM

ICFPC2011 Organizers said...

We cannot publish the original logs for the obvious reason (they reveal the strategy of other players). We will think about better output perhaps.

June 17, 2011 12:26 PM

Post a Comment

Comment as: Select profile...

Post Comment Preview

Home Older Post

Subscribe to: Post Comments (Atom)