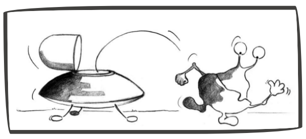
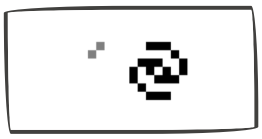



The contest is over, if you want to continue trying, here are the instructions to run the server locally.

Task

As many will remember, in 2006 we discovered documents of an old society, called the Cult of the Bound Variable. With the valuable help from the ICFP community, the Cult's computing device (Universal Machine) was brought back to life, and much interesting information could be recovered. In recent observations from the Pegovka observatory, we discovered something stunning: the Cult of the Bound Variable still exists, and has migrated their civilization to space! It is suspected that Endo helped them move, after he escaped from earth again.

After a couple of months of research, we have been able to decipher most of the received messages, and set up a communication channel. People of the Cult of the Bound variable now use *Interstellar Communication Functional Programs* (ICFP) to communicate. Our findings about ICFP expressions can be found on this page.

Challenges

While we were successful in deciphering the communication language, we unfortunately faced challenges that we cannot solve ourselves, and we once again ask for your help! The communication channel communicates with the School of the Bound Variable, a MOOC where students can follow several courses to learn about various skills necessary for life in space. Each of the courses poses a set of tests, which are scored according to some metric.

An important note: it seems that basic communication utilizes strings only. Therefore, we advice you to implement more advanced ICFP expressions only once you need them, and start with strings.

Communication channel

For you to try out the communication, we have opened up the communication channel with the Cult for you. By sending a HTTP `POST` request to `https://boundvariable.space/communicate`, with the ICFP in the body, your request is sent into the galaxy and the response is returned to you. For identification purposes, you must send the `Authorization` header that you can find on your team page.

We strongly advise that you make the `POST` request directly from your favorite programming language, but we also provide web based communication. That page also shows your communication history.

Furthermore, we found out that sending `S'%4}).$%8` is a great entrypoint for communicating with the school of the Cult of the Bound Variable.

Limits

Communicating with space is an energy consuming process, so for environmental and monetary reasons, we have put limits on the messages. The `POST` body of your request must not exceed 1 MB (`1048576` bytes), and you may send at most `20` messages per minute.

Scoring

The tests for each course are scored according to some criterion explained to you when you enter the course. Your best score for each test is stored. On the scoreboard page we've rendered an overview of the ranks of all teams on each course as well as a global rank. To find the ranks for a course given all the scores of the individual tests of that course, and to find the global rank based on the ranks for the courses, we use the so called Borda count.

In technical terms, the ranklist is an election, where teams are the candidates and the tests are the voters. The better you do for an individual tests, the higher this test will rank you. A more intuitive explanation is that the amount of points you score for each test, is the amount of teams scoring strictly worse than you on that test, and then the ranklist for that course is based on the sum of those points. This method is first used to compute a ranklist per course, and then again using the ranks for the individual courses to find the global rank.

While this may sound abstract, we believe this is all you need to know. And important property of this rating system is that absolute scores don't matter, only the order. It also deals naturally with ties (some tests are just correct/incorrect, there everyone solving it is tied at 1st place), and automatically balances tests with different absolute score ranges. And of course you should just try get the best scores on all tests!

Final code submit

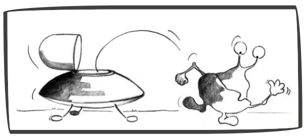
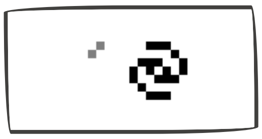
To be considered for prizes, please submit your code near the end of the contest via your team page. The code submission closes 3 hours after the end of the contest. It is not necessary to make a separate submission for the lightning round, but please include a README file indicating which parts are from the first 24 hours.

Meta note

While the introduction refers to various editions of the ICFP programming contest, we want to make explicit that the tasks set for this year are completely new, and knowledge of the earlier challenges is not necessary. However, we referred to them because we enjoyed those contests, so once this year's contest is over we advice everyone to give them a go in case you haven't yet!

To make the contest fair for all timezones, we do not intend to make any changes to the task during the contest. However, we might publish some extra information at the end of the lightning round.

And as last request to contestants: enjoy the contest, but also make sure others can enjoy it! Please do not make attempts to break our server, it should be quite robust, but remember that the ICFPC organizers are volunteers that organize this contest in their free time. And so far we tremendously enjoyed ourselves in the preparations, so we hope to share as much as possible of that with you!

ICFP LANGUAGE

An *Interstellar Communication Functional Program* (ICFP) consists of a list of space-separated *tokens*. A *token* consists of one or more printable ASCII characters, from ASCII code 33 ('`!`') up to and including code 126 ('`~`'). In other words, there are 94 possible characters, and a *token* is a nonempty sequence of such characters.

The first character of a *token* is called the *indicator*, and determines the type of the *token*. The (possibly empty) remainder of the *token* is called *body*. The different *token* types are explained in the next subsections.

Booleans

`indicator = T` and an empty *body* represents the constant `true`, and
`indicator = F` and an empty *body* represents the constant `false`.

Integers

`indicator = I`, requires a non-empty *body*.

The *body* is interpreted as a base-94 number, e.g. the digits are the 94 printable ASCII characters with the exclamation mark representing `0`, double quotes `1`, etc. For example, `I/6` represent the number `1337`.

Strings

`indicator = S`

The Cult of the Bound variable seems to use a system similar to ASCII to encode characters, but ordered slightly differently. Specifically, ASCII codes 33 to 126 from the *body* can be translated to human readable text by converting them according to the following order:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

Here `<space>` denotes a single space character, and `<newline>` a single newline character. For example, `SB%,,/}Q/2,$_` represents the string "Hello World!".

Unary operators

`indicator = U`, requires a *body* of exactly 1 character long, and should be followed by an ICFP which can be parsed from the tokens following it.

Character	Meaning	Example
`-`	Integer negation	`U- I$` -> `-3`
`!`	Boolean not	`U! T` -> `false`
`#`	string-to-int: interpret a string as a base-94 number	`U# S4%34` -> `15818151`
`$`	int-to-string: inverse of the above	`U$ I4%34` -> `test`

The `->` symbol in this table should be read as "will evaluate to", see Evaluation.

Binary operators

`indicator = B`, requires a *body* of exactly 1 character long, and should be followed by two ICFPs (let's call them `x` and `y`).

Character	Meaning	Example
`+`	Integer addition	`B+ I# I$` -> `5`

Character	Meaning	Example
`-`	Integer subtraction	`B- I$ I#` -> `1`
`*`	Integer multiplication	`B* I$ I#` -> `6`
`/`	Integer division (truncated towards zero)	`B/ U- I( I#` -> `-3`
`%`	Integer modulo	`B% U- I( I#` -> `-1`
`<`	Integer comparison	`B< I$ I#` -> `false`
`>`	Integer comparison	`B> I$ I#` -> `true`
`=`	Equality comparison, works for int, bool and string	`B= I$ I#` -> `false`
`|`	Boolean or	`B| T F` -> `true`
`&`	Boolean and	`B& T F` -> `false`
`.`	String concatenation	`B. S4% S34` -> `"test"`
`T`	Take first `x` chars of string `y`	`BT I$ S4%34` -> `"tes"`
`D`	Drop first `x` chars of string `y`	`BD I$ S4%34` -> `"t"`
`$`	Apply term `x` to `y` (see Lambda abstractions)	

If

`indicator = ?` with an empty *body*, followed by three ICFPs: the first should evaluate to a boolean, if it's true then the second is evaluated for the result, else the third. For example:

```
? B> I# I$ S9%3 S./
```

evaluates to `no`.

Lambda abstractions

`indicator = L` is a lambda abstraction, where the *body* should be interpreted as a base-94 number in the same way as integers, which is the variable number, and it takes one ICFP as argument. `indicator = v` is a variable, with again a *body* being the base-94 variable number.

When the first argument of the binary application operator `$` evaluates to a lambda abstraction, the second argument of the application is assigned to that variable. For example, the ICFP

```
B$ B$ L# L$ v# B. SB%,,/ S}Q/2,$_ IK
```

represents the program (e.g. in Haskell-style)

```
((\v2 -> \v3 -> v2) ("Hello" . " World!")) 42
```

which would evaluate to the string `"Hello World!"`.

Evaluation

The most prevalent ICFP messaging software, Macroware Insight, evaluates ICFP messages using a call-by-name strategy. This means that the binary application operator is non-strict; the second argument is substituted in the place of the binding variable (using capture-avoiding substitution). If an argument is not used in the body of the lambda abstraction, such as `v3` in the above example, it is never evaluated. When a variable is used several times, the expression is evaluated multiple times.

For example, evaluation would take the following steps:

```
B$ L# B$ L" B+ v" v" B* I$ I# v8
```

```
B$ L" B+ v" v" B* I$ I#
B+ B* I$ I# B* I$ I#
B+ I' B* I$ I#
B+ I' I'
I-
```

Limits

As communication with Earth is complicated, the Cult seems to have put some restrictions on their Macroware Insight software. Specifically, message processing is aborted when exceeding `10_000_000` beta reductions. Built-in operators are strict (except for `B$`, of course) and do not count towards the limit of beta reductions. Contestants' messages therefore must stay within these limits.

For example, the following term, which evaluates to `16`, uses `109` beta reductions during evaluation:

```
B$ B$ L" B$ L# B$ v" B$ v# v# L# B$ v" B$ v# v# L" L# ? B= v#
```

Researchers expect that the limit on the amount beta reductions is the only limit that contestants may run into, but there seem to also be some (unknown) limits on memory usage and total runtime.

Unknown operators

The above set of language constructs are all that researchers have discovered, and it is conjectured that the Cult will never use anything else in their communication towards Earth. However, it is unknown whether more language constructs exist.