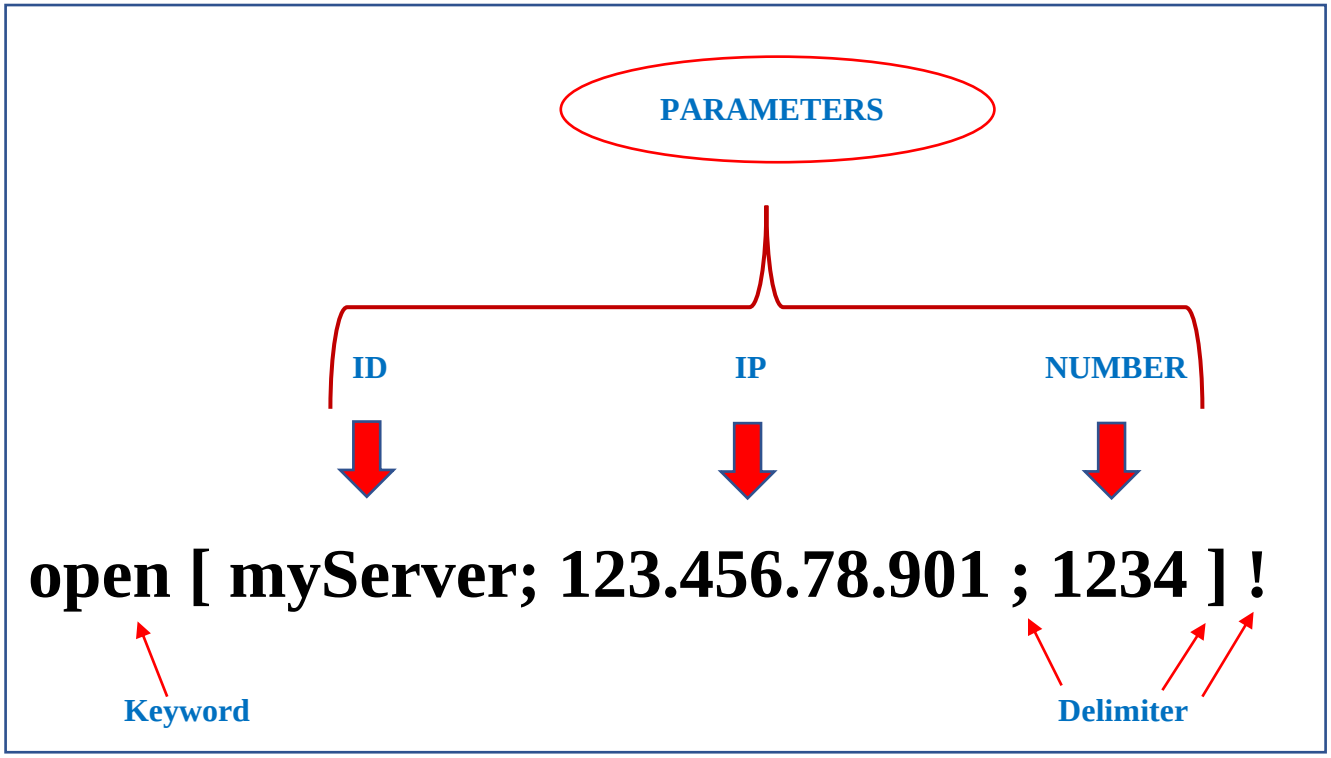
AmpleNET

AmpleNET is a new programming language intended to simplify the communication between devices. It provides simple functionality to create local servers without worrying about all the networking process. To create your servers and start a connection, just provide a name, IP address, and port for each of them. If you're not too familiar with these things, just run the default command and give it a name, the rest will be done automatically. After creating your servers, connect them and start sending messages. It's that easy! Below is the syntax of the language, along with some examples and information you might find useful.

Syntax:



Most of AmpleNET's instructions follow the same syntax. They all start with a keyword specifying the action wanted, followed by the parameters required. The parameters must be inside brackets, each separated by semicolons. Finally, all instructions end with an exclamation mark. The commands included in this language are:

- *Open:* which receives an ID, IP, and NUMBER as parameters. This instruction creates a server with the name specified and binds it to the given IP and port.
- *Default:* receives only an ID, and then opens a new server with computer's IP and random port.

- *External:* receives ID and NUMBER as parameters. This opens a different type of server for external communication in the same network. Once created, it will receive messages from client server created in another computer.
- *Client:* receives ID, IP and NUMBER. It opens special type of server called client which is used in the external communication. The IP and port given must be the ones pertaining to the external server created in the other computer.
- *Connect:* accepts two server IDs as parameters. It connects both servers given and prepares them for interaction.
- *Send:* just like connect, it accepts two server IDs as well as a MESSAGE in its parameters. It then sends the given message from the first server to the second.

As for comments, they are accepted in AmpleNET. These must be between parenthesis for the language to identify and not read as part of the program.

Local communication:

Example program with *open*: (IP has been hidden for privacy purposes.)

```
1  open[s1;[redacted];4567]!
2  open[s2;[redacted];4789]!
3  (this is a comment)
4  connect[s1;s2]!
5  send[s1;s2;<Hello, we are connected.>]!
6  send[s2;s1;<Yes, this is fun.>]!
7  send[s1;s2;<That was fun while it lasted, goodbye.>]!
```

Example output: (IP has been hidden for privacy purposes.)

```
Connection from s1 (1[redacted] 4567) established with s2 [redacted], 4789)
Message sent from s1 [redacted], 4567)to s2 (1[redacted] 4789) :  Hello, we are connected.
Message sent from s2 [redacted], 4789) to s1 ([redacted], 4567):  Yes, this is fun.
Message sent from s1 [redacted], 4567)to s2 (1[redacted] 4789) :  That was fun while it lasted, goodbye.

Process finished with exit code 0
```

As can be seen in the example above, *open* is used to create two servers named “s1” and “s2”. *Connect* is then used to join these two servers, and finally *send* is used to push information from one server to the other. Note that the messages beings sent are between the compare symbols, which is a requirement of the program.

Example program with *default*:

```
1    (Using default command as parameter of connect)
2    connect[default[s1];default[s2]]!
3    send[s1;s2;<I have sent a message>]!
4    send[s2;s1;<I can see such message>]!
5    send[s1;s2;<Sending message back>]!
```

The example above runs the same commands as before. However, instead of the *open* command, *default* is used to create the servers. Also, the default function is given as parameter in the connect command, reducing the program even further by applying functionality.

External communication:

Due to AmpleNET's early release, external communication is only possible if both computers are connected to the same network. To do so, run the external command in one computer, and the client command in another as shown below:

Host computer: external[name;port]!

Client computer: client[name;host-IP;host-port]!

Once done, the host computer will receive messages sent from the client. Stay tuned for AmpleNET 2.0, which will allow communication through different networks.

Parsing rules:

Included are the grammatical rules that govern the process of the language. Rule 1 is the main process which most of the test cases included follow. As for rules 2-5, they were included to allow console to accept each instruction individually, instead of expecting a full script with all instructions.

```
Rule 0     S' -> process
Rule 1     process -> create EXCLAMATION create EXCLAMATION join EXCLAMATION
                      communication
Rule 2     process -> join EXCLAMATION communication
Rule 3     process -> create EXCLAMATION
Rule 4     process -> join EXCLAMATION
Rule 5     process -> communication
Rule 6     create -> DEFAULT LB ID RB
Rule 7     create -> EXTERNAL LB ID SEMICOLON NUMBER RB
Rule 8     create -> OPEN LB ID SEMICOLON IP SEMICOLON NUMBER RB
Rule 9     create -> CLIENT LB ID SEMICOLON IP SEMICOLON NUMBER RB
Rule 10    join -> CONNECT LB ID SEMICOLON ID RB
```

```
Rule 11    join -> CONNECT LB create SEMICOLON create RB
Rule 12    communication -> talk EXCLAMATION communication
Rule 13    communication -> talk EXCLAMATION
Rule 14    talk -> SEND LB ID SEMICOLON ID SEMICOLON MESSAGE RB
```

Contributors:

Everson Rodriguez Muñiz

Geidel X. Soliván Amaro

Yamil J. Hernandez Bencomo