****

**Galaxy Programming Language**

GX is a syntax friendly and easy to understand language programming language without any ambiguity. One of the difference from other language is that GX using . (dot) for line breaking. GX License v1.0.

**Syntax and Semantics of Galaxy**

- The default file extension for Galaxy files is ".gx".  
- GX statements end with a dot (.).  
- GX is a case sensitive language.

- There is only one data type: digit(0-9).

- There is only one way to name variables: char(a-z & A-Z) . So in total you can have 52 variables.

- There is two built-in functions: print(var or const) and scan(var)

- An equal sign (=) is used to assign, arithmetic operators ( + - \* / ) is used to compute values to variables. Computed variables can’t go further from 9.

- There is no declaration of variables, therefore implicitly specifying each data type for variables is not needed too.

- If condition syntax (Which actually works):

if(<expr>) then <stmt> {elseif <stmt>}\* {else <stmt>}?

- Inline if condition syntax (This one works too):

(<expr> then <stmt> else <stmt>)

- While loop syntax:

while(<expr>) start <stmt> end

- For loop syntax:

for(<assignment>. <expr>. <assignment>) start <stmt> end

- Function syntax:

<char>(<paremeters>) start <stmt> end  
- Gx does not have a main function to start, it just starts to read from top to bottom, so it isn’t interpreted language.

- Single line comments starts with **#,** multi line comments starts with #\*and ends with **\*#**.

**Instructions to Operate**

Makefile handles everything for you. Here are some of useful commands:

- **make fullBuild**

Compiles lex and yacc and links them.

**- make runExample**

Runs an example prepared by us.

- **make clean**

Cleans all compiled binaries, headers etc.

**BNF Form of Galaxy**

<program> ::= <stmt\_list>

<consts> ::= const <assignment> | <line\_breaker>

<stmt\_list> ::= empty

| <consts> <stmt\_list>

| <function> <stmt\_list>

| <statement> <stmt\_list>

| <comment> <stmt\_list>

<function> ::= <char> ( <parameters> ) start {<statement>}\* end

<line-breaker> ::= <..>

<parameters> ::= <expr>

| <parameters> , <expr>

| empty

<comment> ::= <SL\_COMM>

| <ML\_COMM\_START>

<expr> → <expr> or <and-expression>

| <and-expression>

| <equality-expression>

| <relational-expression>

| <additive-expression>

| <multiplicative-expression>

| <expression-with-head>

| <expression-with-tail>

<and-expression> ::= <equality-expression>

| <and-expression> and <equality-expression>

<equality-expression> ::= <relational-expression>

| <equality-expression> == <relational-expression>

| <equality-expression> != <relational-expression>

<relational-expression> ::= <additive-expression>

| <relational-expression> < <additive-expression>

| <relational-expression> > <additive-expression>

| <relational-expression> <= <additive-expression>

| <relational-expression> >= <additive-expression>

<additive-expression> ::= <multiplicative-expression>

| <additive-expression> + <multiplicative-expression>

| <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <expression-with-head>

| <multiplicative-expression> \* <expression-with-head>

| <multiplicative-expression> / <expression-with-head>

| <multiplicative-expression> % <expression-with-head>

<expression-with-head> ::= <expression-with-tail>

| ++ <vars-and-consts>

| -- <vars-and-consts>

<expression-with-tail> ::= <expression\_with\_parenthesis>

| <expression-with-tail> ++

| <expression-with-tail> --

<expression-with-parenthesis> ::= not(<expr>)

| (<expr>)

| <vars-and-consts>

<vars-and-consts> ::= <char>

| <integer>

| ( <assignment> )

<conditional-expression> ::= (<expr> then <expr> else <expr>)

<expr\_or\_conditional\_expression> ::= <expr>

| <onditional-expression>

<assignment> ::= <vars-and-consts> = <expr\_or\_conditional\_expression>

| <vars-and-consts> \*= <conditional-expression>

| <vars-and-consts> /= <conditional-expression>

| <vars-and-consts> %= <conditional-expression>

| <vars-and-consts> += <conditional-expression>

| <vars-and-consts> -= <conditional-expression>

<statement> ::= <line-breaker>

| <assignment> <line-breaker>

| <condition>

| CONTINUE <line\_breaker>

| BREAK <line\_breaker>

| <lup>

| <expression\_with\_head> <line\_breaker>

| PRINT (<vars\_and\_consts>) <line\_breaker>

| SCAN (CHAR) <line\_breaker>

<function\_expr> :: = (<expr>) then

<function\_else> :: = else

<expr\_and\_statement\_zero\_or\_more>:

| ELSEIF expr\_and\_statement <expr\_and\_statement\_zero\_or\_more>

<condition> : IF <function\_expr> <statements> <expr\_and\_statement\_zero\_or\_more> END

| IF <function\_expr> <statements> < expr\_and\_statement\_zero\_or\_more> <function\_else> <statements> END

<for\_assignment> : <expression\_with\_head> | <assignment>

<lup> ::= while ( <expr> ) start <statement> end

| for ( <assignment><line-breaker> <expr> <line-breaker> <for\_assignment> ) start <statements> end

**Contributors**

Eray Orçunus

Gökmen Özçelik

Caner Türkmen