ETL
(EASY TO LEARN)

Språkspecifikation v3

Det som vi tänkte skapa är ett nytt språk som har **Imperativt programspråk** som grund och koden skall vara baserad på **Ruby** samt tänkte vi välja **rdparse.rb** som parser.

Språket skall passa de nybörjare som har inga tidigare förkunskaper inom programmering. Språket skall innehålla de vanliga satserna såsom **(villkor**, **tilldelning**, **repetition)** och funktioner samt vissa datatyper som **integer** och **strängar**.
Språket skall kunna hantera sedvanliga **aritmetiska**(**+**, **-**, ***** , **/**) och **logiska** uttryck(**not**, **or**, **and**).

Betyg 3: Aritmetik med rätta prioriteter och associativiteten, kontrollstrukturer (åtm både någon sorts iteration och val), funktioner/procedurer med parametrar och scopehantering,

- **Ett utkast av språkets grammatik**

  <PROGRAM> ::= start program <STATEMENTS> end program

  <STATEMENTS> ::= <STATEMENTS> <STATEMENT>

  | <STATEMENT>

  <STATEMENT> ::= <RETURN>

  | <FUNCTION>

  | <FUNCTION_CALL>

  | <BREAK>

  | <PRINT>

  | <IF_BLOCK>

| <WHILE_LOOP>

| <ASSIGN>

<PRINT> ::= write <STRING_ADDING>

| write <STRING_EXPR>

| write <EXPR>

<ID> ::= /[a-z]+[a-z0-9_]*/

<FUNCTION> ::= define <ID> (<PARAMETER_NAME>) <STATEMENTS> enddef

| define <ID> ( ) <STATEMENTS> enddef

<FUNCTION_CALL> ::= <ID> ( )

| <ID> (<PARAMETERS>)

<RETURN> ::= return <EXPR>

| return <STRING_EXPR>

<PARAMETER_NAME> ::= <ID>

| <PARAMETER_NAME> , <ID>

<PARAMETERS> ::= <PARAMETER>

| <PARAMETERS> , <PARAMETER>

<PARAMETER> ::= <EXPR>

| <ID>

<WHILE_LOOP> ::= while (<BOOL_EXPR>) <STATEMENTS> endwhile

<TERM> ::= <TERM> * <ATOM>

| <TERM> / <ATOM>

| <FUNCTION_CALL>

| <ATOM>

<EXPR> ::= <EXPR> + <TERM>

| <EXPR> - <TERM>

| <TERM>

STRING_ADDING ::= <STRING_ADDING> plus <STRING_EXPR>

| <STRING_EXPR> plus <STRING_EXPR>

<IF_BOX> ::= <IF_RULE>

<IF_RULE> ::= <IF> endif

| <IF> <ELSE> endif

| <IF> <ELSE_IF> <ELSE> endif

<IF> ::= if (<BOOL_EXPR>) then <STATEMENTS>

<ELSE> ::= otherwise <STATEMENTS>

<ELSE_IF>::= else if (<BOOL_EXPR>) then <STATEMENTS>

<ASSIGN> ::= <ID> = <EXPR>

| <ID = <BOOL_EXPR>

| <ID> = <STRING_EXPR>

| <ID> = <STRING_ADDING>

<BOOL_WORD> ::= <EXPR> less than <EXPR>

| <EXPR> greater than <EXPR>

| <EXPR> less than or equal to <EXPR>

| <EXPR> greater than or equal to <EXPR>

| <EXPR> not equal to <EXPR>

| <EXPR> equal <EXPR>

<BOOL_SYMBOL> ::= <EXPR> < <EXPR>

| <EXPR> > <EXPR>

| <EXPR> <= <EXPR>

| <EXPR> >= <EXPR>

| <EXPR> != <EXPR>

| <EXPR> == <EXPR>

<BOOL_EXPR> ::= <BOOL_WORD>

| <BOOL_SYMBOL>

| <BOOL_EXPR> and <BOOL_EXPR>

| <BOOL_EXPR> or <BOOL_EXPR>

| not <BOOL_EXPR>

<STRING_EXPR> ::= <ID>

| /"[^\"]*"/

| /'[^\']*'/

| <FUNCTION_CALL>

<ATOM> ::= <FUNCTION_CALL>

| <ID>

| <Float>

| <Integer>

- **Statisk vs dynamisk typning?**

  Vi tycker att vi ska ha en dynamisk typning där variabeln kommer ha sin type från själva värdet på det. Är det inom (“”) så är det sträng, är det siffror så blir det en (integer) osv.

- **När en funktion anropas med komplexa datatyper såsom strängar eller listor, blir det en kopia eller referens?**

  Vi tycker att det kommer bli en kopia på de komplexa datatyper liksom strängar eller listor eftersom vi tycker att det kommer bli mer säkert med en kopia samt detta kommer göra språket lättare att använda då man behöver inte oroa sig för de variabler man ger till en funktion.

- **Hur funkar scopet: Vid anrop av funktion? I blocket tillhörande en for-loop? Delar funktioner och variabler scope? Osv**

  Vi bestämde oss att ha statisk scoping, dvs vid anrop av en funktion så kommer programmet kolla först det globala scopet och inte bryr sig om vad variabeln innehåller i själva funktions kroppen. Exempel:

```
1.  b = 5
2.
3.  foo()
4.  a = b + 5
5.  return a
6.  end
7.
8.  bar()
9.  b = 2
10. return foo()
11. end
12.
13. foo() // returns 10
14. bar() // returns 10
```

- **Mer detaljerade kodexempel**

start program

a = 5
b = 8

define myfunc(a, b)

if (a < b) then
write “a is smaller than b”

else if (b < a) then
write "b is smaller than a"

otherwise
write “b is bigger than a”
endif

return "the end"

enddef

myfunc(a, b)

while (y < 5)
write “y is less than five”
endwhile

x = "Hej "
y = "på "

write x plus y plus "dig!" ## Hej på dig!
write x plus y plus “dig!” plus 2 ##felmeddelande (titta Andra relevanta detaljer)

*f = 5 * 5 ##term*
*f * 2 ##term * atom*
5 / 5

g = 1 + 2 ##expr + expr
g - 3 ##term - expr

end program

- **Andra relevanta detaljer**

(write “Hej ” plus 2) Kommer skicka fel meddelande till användaren.
Då kommer vi fixa fel hantering för detta eftersom man inte kan plussa integer med strängar i språket.