LINKÖPINGS UNIVERSITET

TDP019 Projekt: Datorspråk

Språkspecifikation

Albin Dahlén, albda746@student.liu.se
Filip Ingvarsson, filin764@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2022
Version 1.2

2023-02-27

Innehåll

Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Språkspecifikation 2	23-02-22
1.2	Språkspecifikation 3	23-02-27

1 Introduktion

2 Grammatik

```
<program> ::= <stmt_list>

<stmt_list> ::= <stmt><stmt_list>
              | <stmt>

<stmt> ::= <var_declaration>
         | <conditional>
         | <func_declaration>
         | <class_declaration>
         | <assign_stmt>
         | <loop>
         | <expr>
         | "return" <expr>

<var_declaration> ::= "const" <type_specifier> <identifier> <var_declaration_tail>
                    | <type_specifier> <identifier> <var_declaration_tail>
                    | "const" <array_type_specifier> <identifier> <array_declaration_tail>
                    | <array_type_specifier> <identifier> <array_declaration_tail>

<var_declaration_tail> ::= "=" <expr>
                         | empty

<array_declaration_tail> ::= "=" <array_literal>
                           | empty

<array_literal> ::= "[" <expr> <array_literal_tail> "]"
                  | "[" "]"

<array_literal_tail> ::= "," <expr>
                       | empty

<class_declaration> ::= "Class" <identifier> "{" <class_members> "}"

<class_members> ::= <var_declaration> <class_members>
                  | <func_declaration> <class_members>
                  | empty

<func_declaration> ::= "func" <func_specifier> <identifier> "(" <func_params> ")" "{" <stmt_list> "}"

<func_params> ::= <type_specifier> <identifier> <opt_func_params>
                | empty

<opt_func_params> ::= "," <type_specifier> <identifier> <opt_func_params>
                    | empty

<func_specifier> ::= "void" | <type_specifier>
```

```
<type_specifier> ::= "int" | "float" | "bool" | "string" | <array_type_specifier>

<array_type_specifier> ::= <type_specifier> "[]"

<conditional> ::= "if" <expr> "{" <stmt_list> "}" <elsif_block> <else_block>

<elsif_block> ::= "elsif" <expr> "{" <stmt_list> "}" <elsif_block>
                | empty

<else_block> ::= "else" "{" <stmt_list> "}"
               | empty

<loop> ::= "while" <expr> "{" <stmt_list> "}"
         | "for" <var_declaration> "," <logical_expr> "," <primary_expr> "{" <stmt_list> "}"

<assign_stmt> ::= <identifier> "=" <expr>

<func_call> ::= <identifier> "(" <func_call_params> ")"

<func_call_params> ::= <expr> <opt_func_call_params>
                     | empty

<opt_func_call_params> ::= "," <expr> <opt_func_call_params>
                         | empty

<expr> ::= <logical_expr>
         | <func_call>
         | <primary_expr>

<logical_expr> ::= <logical_and_expr> | <logical_or_expr>

<logical_and_expr> ::= <comparison_expr> { "&&" <comparison_expr> }

<logical_or_expr> ::= <logical_and_expr> { "||" <logical_and_expr> }

<comparison_expr> ::= <additive_expr> { <logical_comparator> <additive_expr> }

<additive_expr> ::= <multiplication_expr> { ("+" | "-") <multiplication_expr> }

<multiplication_expr> ::= <unary_expr> { ("*" | "/") <unary_expr> }

<unary_expr> ::= ("+" | "-") <primary_expr>

<primary_expr> ::= <identifier>
                 | <array_access>
                 | <numeric_literal>
                 | <boolean_literal>
                 | "(" <expr> ")"
                 | <class_instance>
```

```
<class_instance> ::= <identifier> "." "new" "(" ")"
                   | <identifier> "." "new" "(" <func_call_params> ")"

<array_access> ::= <identifier> "[" <expr> "]"

<identifier> ::= /[a-zA-Z_][a-zA-Z0-9_]*/

<numeric_literal> ::= /[0-9]+ ( "." [0-9]+ )?/

<boolean_literal> ::= "true" | "false"

<logical_comparator> ::= "<" | ">" | "<=" | ">=" | "==" | "!="
```

3 Typning

Typningen ska vara statisk, där variabler deklareras genom att specificera vilken typ det är.

Exempel:
int a = 2
float b = 2
string c = “Hej”

4 Variabler

Variabler deklareras genom att specificera vilken typ variabeln är följt av en identifierare och eventuellt en tilldelning.

```
Exempel:
int a
float b = 3.4
```

Varaibler kan även bli satta till konstanter med nyckelordet “const” föjt av en vanlig variabel deklaration. Skillnaden här är att en konstant variabel måste tilldelas sitt värde när den skapas.

```
Exempel:
const int a = 4
const float b = 3.4
```

5 Array / Lista

Arrayer skapas genom att skriva typ specificerare följt av “[]” och sedan identifieraren. Arrayer kan precis som vanliga variabler deklareras utan att tilldelas. Vill man tilldela en array ett värde när den deklareraras gör man detta genom att ange en kommaseparerad lista inom “[]”. Värde måste dock vara samma typ som arrayen är.

```
Exempel:
int[] a
int[] b = []
int[] c = [3, 5, 6]
```

För att komma åt ett värde i listan gör man det med “[]” operator där man specificerar ett index.

```
Exempel:
int[] c = [3, 5, 6]
c[1]
```

I ovan exempel hade vi då fått ut värdet på index 1 vilket är 5.

6 Styrstrukturer

```
Exempel if-sats:
if (cond) {

}

Exempel if-else-sats:
if (cond) {

} else {

}

Exempel if-elsif-elsesats:
if (cond) {

} elseif cond {

} else {

}
```

7 Iteratorer

7.1 For-loop

For-loopar börjar med nyckelorder for. Därefter kan en variabel initieras men det är inte nödvändigt. Därefter kommer ett sanningssats/sanningvärde som måste finnas med. Slutligen kan det finnas en inkremering som läggs på den initierade variabeln ifall den finns. Kodblocket som sedan körs i loopen insluts med .

Exempel:

```
for (variabel initiering, true/false-statement, inkremering){
  statements
}

for int i=1, i < 10, i++{
  statements
}
```

7.2 While-loop

while-loopar börjar med nyckelordet 'while' följt av ett true/false-statement. Kodblocket som sedan körs i loopen insluts med {}.
Exempel:

```
while true/false-statement {
    statement
}

while working == true {
    take_break
}
```

7.3 Foreach-loop

foreach-loopar börjar med nyckelordet 'foreach' följt av en identifierare som används senare i kodblocket för varje objekt i behållare. Därefter kommer nyckelordet in och sedan behållaren som ska loopas igenom.

```
foreach identifier in container{
  "identifier used in statement"
}

foreach apple in tree{
  apple.check_size()
}
```

8 Funktioner

Funktioner defineras genom att skriva nyckelordet *func* följt av vilket returtyp funktionen ska ha. Efter returtypen följer funktionsnamnet och en komma separerad parameterlista omgiven av paranteser där varje parameters typ anges innan parameterns namn. Funktionskroppen omslutes av en start- och slutmåsvinge. I slutet av funktionskroppen måste ett nyckelordet *return* finnas om det inte är en void funktion där nyckelordet *void* har skrivit i funktions deklarationen.

```
Exempel på en funktion:
func string name(string param1, int param2...) {
    statements
    return "This is a string"
}
```

8.1 Parameteröverföring

Alla parametrar skickas alltid som referenser och vill man ha en kopia får man anropa funktionen *copy* på objektet.

```
Exempel funktionsanrop med parameteröverföring:
func_name(param1, param2)

Exempel funktionsanrop med parameteröverföring som kopior:
object.func_name(param1.copy(), param2.copy())
```

9 Objektorienterat

Språket ska vara objektorienterat med arv där funktioner kan specificeras som public, protected och private om de finns definierade i klasser. Finns de inte i klasser kommer de att defaulta till publik.

```
Exempel publik funktion:
func string name(string param1, int param2...) {
```

```
    statements
    return "This is a string"
}

Exempel publik funktion:
public func int name(int param1, int param2...) {
    statements
    return 45
}

Exempel protected funktion:
protected func string name(string param1, int param2...) {
    statements
    return "This is a string"
}

Exempel privat funktion:
private func string name(string param1, int param2...) {
    statements
    return "This is a string"
}
```

9.1 Klasser

```
Exempel klass deklaration:
Class Name {

}

Exempel klass deklaration med arv:
Class Name -> Parent {

}
```

För att instansiera en klass skriver man klassnamn följt av ".new()". Inuti paranteserna kan parametrar skickas med.

```
Exempel:
Person.new()
Person.new(10, 45)
```

För att komma åt medlemsfunktioner i en klass använder man klassinstansen följt av "." och ett funktionsanrop till den funktionen.

```
Exempel:
Person p = Person.new()
p.getAge()
```

10 Scope

Språket ska ha ett statiskt scope vilket betyder att när exempelvis vi letar efter en variabel och går upp till föräldrascopet, kommer vi gå till det scope som är föräldern till där funktionen är definierad. Vid en

funktions anropskedja kommer vi då inte att gå upp genom hela kedjan utan vi kommer hoppa upp till funktionens föräldra scope.

10.1 Variablers livslängd

En variabel där i samband med sitt scope. Definieras en variabel i exempelvis en loop, funktion eller if-sats är variabel bara tillgänglig inom kodblocket.