WinaML

Is Not A Markup Language

Johan Alm & Han Lin Yap
Linköpings universitet
2010-05-08

2010-05-08
Johan Alm (johal003) TDP019 Projekt: Datorspråk
Han Lin Yap (hanya274) Linköpings universitet

Innehållsförteckning

Figur- och tabellförteckning

1. Inledning

Denna rapport är skriven i samband med kursen TDP019 Projekt: Datorspråk på programmet Innovativ programmering (IP) vid institutionen för datavetenskap (IDA) vid Linköpings tekniska högskola (LiTH).

Namnet WINAML är en förkortning av ”WINAML Is Not A Markup Language”.

1.1 Syfte

Vi fick fritt hitta på ett eget programmeringsspråk samt implementera det. Vi skulle bestämma en målgrupp för språket och språket skulle beskrivas med en grammatik. Hur man implementerade språket och vilka verktyg man använde för att göra det var valfritt.

Syftet med WINAML är att göra ett språk som förenklar skrivandet av hemsidor genom att göra en enklare form av HTML med programmeringskonstruktioner. Dessa konstruktioner i WINAML ger en resulterande HTML-kod som kan visas i en webbläsare. WINAML klarar till exempel matematiska beräkningar, villkorsatser, datahantering, återanvändning av kod, beskriva tabeller och har andra verktyg.

Den här dokumentationen kommer ge en grundläggande introduktion till WINAML och dess syntax, konstruktioner och funktionalitet. Dokumentationen består av tre delar. Den första delen är en användarhandledning som beskriver språkets konstruktioner och exempel för nybörjare. Den andra delen är en systemdokumentation som beskriver själva språket mer ingående med grammatiken i BNF, vår kodstandard och instruktioner för installation och användning utav språket. Den tredje och sista delen innehåller våra reflektioner av projektet.

2. Användarhandledning

WINAML är ett högnivåspråk, det är inte ett uppmärkningspråk. Resultatet när man kompilerar en wml-källkodsfil är ett HTML-dokument (Hyper Text Markup Language) som är körbart i vanliga webbläsare.

2.1 Målgrupp

Målgruppen är personer med lite eller ingen programmeringserfarenhet. Bara grundläggande HTML-funktionalitet och programmeringsmetoder finns tillgängliga i WINAML.

2.2 Konstruktioner i språket

WINAML består av flera olika sorters konstruktioner som kan användas och sättas samman på olika sätt för att bilda källkod som kan generera en hemsida som kan visas i en webbläsare.

Kommentering

Man kan göra korta anteckningar i koden. Kommentarer är användbara för programmerare, men ignoreras av kompilatorn och påverkar inte slutresultatet. Kommentarer kan användas för att förtydliga kod, algoritmer och kodstatus. En kommentar startas med ett utrops-tecken följt av ett citat-tecken och avslutas med ett citat-tecken följt av ett utrops-tecken.

Exempel:

```
!” Kommentar ”!
```

Text

Text-taggar som innehåller valfri text som syns i webbläsaren. Varje text-tagg blir en paragraf. En tagg börjar med vänster vinkelparantes, taggens benämning, höger vinkelparantes sedan innehållet och slutar med vänster vinkelparantes följt av höger vinkelparantes.

Exempel på en text-tagg:

```
<text>Detta kommer att synas i webbläsaren<>
```

Aritmetiskt uttryck

Aritmetiska uttryck kan vara en kombination av heltalsiffror, variabler, strängar och operatorer (+-*/) som evalueras efter en viss ordning av regler. Det går inte att blanda tal med text i samma utryck.

Exempel:

```
1+2                         !” Går ”!
”' Hello '” + ”' World '”   !” Går ”!
”'Blanda'” + 8              !” Går ej ”!
```

Variabler

Variabler är en enhet för att spara värden eller data. Man kan hämta, spara och ändra värden på variabler. Variabler har ett nyckelord även kallas för variabelnamn som används

för att identifieras. Variabelnamn måste börja på en liten bokstav som följs av godtyckligt många bokstäver och siffror. Svenska tecknen å, ä och ö kan inte användas i ett variabelnamn. Det finns tre typer av variabler; heltalsiffror, strängar och listor. Listor används i tabeller som beskrivs längre ner.

Man kan deklarera eller byta värde på en variabel på följande sätt:

```
<<heltal = 8>                                  !” En heltalsvariabel ”!
<<strang = ”'Räksmörgås!'”>                    !” En strängvariabel ”!
<<lista = [[”'Kakor'”, 5][”'Bananer”, 2]]>     !” En listvariabel ”!
```

Variabler kan skrivas ut i text på följande sätt – ett nummertecken följt av ett par klammerparanteser med identifieraren mellan:

```
<text>#{variabel}<>
```

Specialvariabler

Det finns specialvariabler för det färdiga dokumentets titel och språk som lagras i specialvariablerna ”@title” och ”@language”. Specialvariabler hanteras på samma sätt som vanliga variabler med skillnaden att deras namn börjar med ett snabel-a (@).

Standardvärde på ”@title” är ”New WINAML document” och ”@language” är ”sv” där sv är förkortning för svenska.

Boolean

Boolean är en primitiv datatyp som antingen kan vara sann eller falsk och det skrivs som '”true” eller ”false” i WINAML. Med hjälp av operatorer som t.ex. större än, mindre än och likhet så kan man jämföra två aritmetiska uttryck och få ett sant eller falskt värde. Man kan kombinera olika booleaner med ”and” eller ”or”. Booleaner är användbara i villkorssatser som beskrivs längre ner.

Tabell 1. *Tillgängliga boolanska operatorer.*

x < y	x mindre än y
x > y	x större än y
x <= y	x mindre eller lika med y
x >= y	x större eller lika med y
x == y	x lika med y
x != y	x inte lika med y
not x	inte x
x and y	x och y
x or y	x eller y
true	sann
false	falsk

Villkor

Villkorssatser kan utföra olika kodblock beroende på ett eller flera booleanska värden. En vanlig form av villkorsats är ”if-else”.

Exempel på en ”if-else”-sats:

WINAML-kod

```
<<if true>
    <text>Detta kodblock körs ifall villkoret är sant.<>
<<else>
    <text>Detta kodblock körs ifall villkoret är falskt.<>
<<endif>
```

Utdata

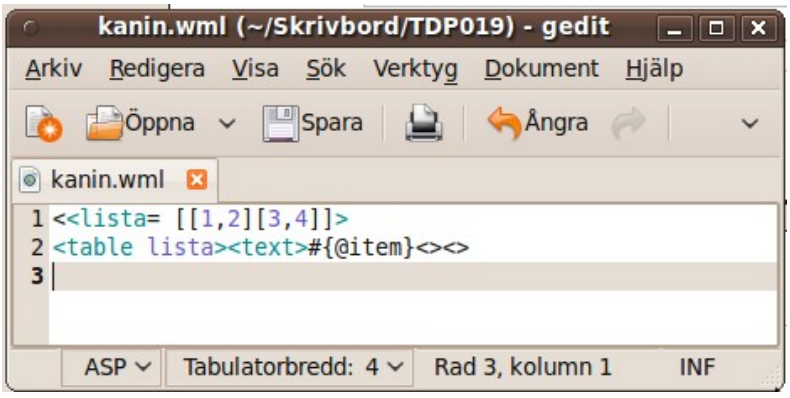
Detta kodblock körs ifall villkoret är sant.

Tabell

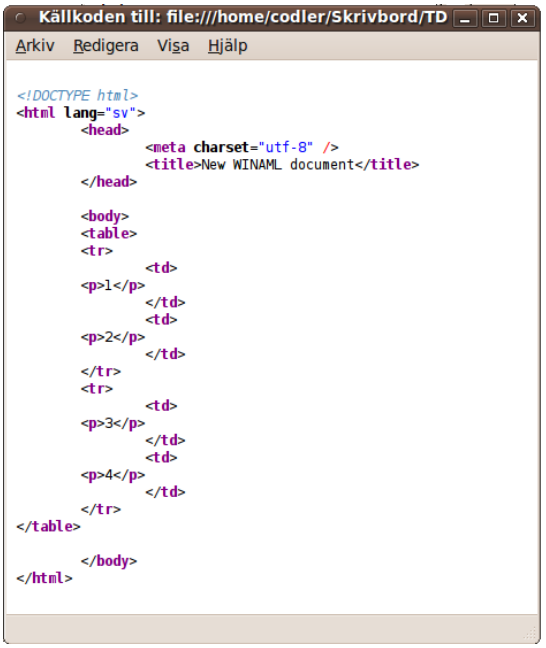
Tabeller består av rader och kolumner med data från en listad variabel. När man skriver ut en tabell så skrivs ”@item” innanför tabell för att identifiera innehållet i listad variabel.

Exempel:

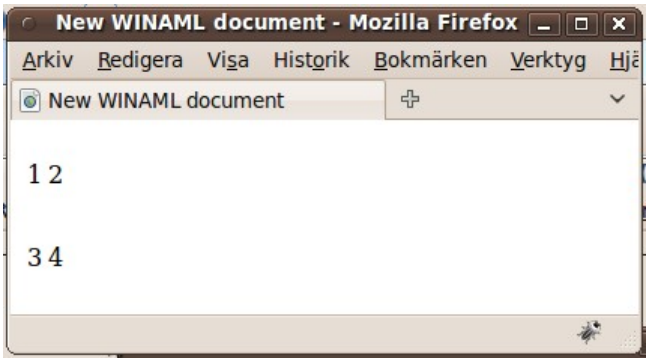
```
<<lista= [[1,2][3,4]]>
<table lista>
        <text>#{@item}<>
<>
```



Figur 1. *Källkod i WINAML*



Figur 2. *Den genererade HTML-koden.*



Figur 3. *Det skapande HTML-dokumentet presenterad i en webbläsare.*

Importera andra wml-filer

Att importera en wml-fil till en annan wml-fil kan vara användbart om man vill kunna återanvända kod eller vill separera till flera mindre källkodsfiler.

Man importerar andra wml-filer genom att skriva följade:

```
<<include "'filnamnet.wml'">
```

Variablerna följer med vid importering som i följande exempel:

Fil1.wml

```
<<variabel = 8>
```

Fil2.wml

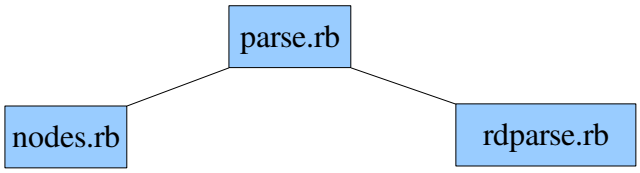
```
<<include "'Fil1.wml'">
<text>#{variabel}<>
```

3. Systemdokumentation

Första delen av systemdokumentation beskriver kodstandarden vi använde vid implementationen av språket i Ruby. Den andra delen av systemdokumentationen innehåller instruktioner om hur man installerar och använder WINAML.

WINAML består av följande Ruby-moduler, ”rdparse.rb”, ”parse.rb” och ”nodes.rb”. Modulen ”rdparse.rb” är ett verktyg för att tolka domänspecifika språk. Modulen ”parse.rb” är själva huvudfilen för vårt språk där vi har programmerat in regler för hur vårt språk ska läsas. Modulen ”nodes.rb” innehåller alla klasser för uppbyggnaden av ett

syntaxträd som sedan evalueras.



Figur 4. *Språkets moduler*

Rdparse delar upp källkoden i form av i tokens med hjälp av regulära uttryck. Dessa tokens matchas sedan ihop beroende på vår grammatik och används för att att bilda ett syntaxträd av nodklasser. Efter att den har byggt ihop trädet så evalueras noderna och genererar strängar av HTML-kod som läggs in i ett HTML-dokument med samma namn som källkodsfilen fast med en annan filändelse.

WINAML-kod

```
<text>Hello World<>
```

Utdata (HTML)

```
<!DOCTYPE html>
<html lang="sv">
        <head>
                <meta charset="utf-8" />
                <title>New WINAML document</title>
        </head>
        <body>
        <p>Hello World</p>
        </body>
</html>
```

3.1 Kodstandard

Indentering

Vi har valt att använda två mellanrum för indentering, eftersom vi tycker det ger hög läsbarhet samt skrivbarhet.

Radlängd
Vi har använt max 79 tecken per rad.

Kommentarer
Våra kommentarer är hela meningar på engelska. Vårt mål har varit att göra läsbar kod som behöver så få kommentarförklaringar som möjligt. Mer komplicerad kod som t.ex. några av våra långa reguljära uttryck har vi valt att göra korta förklarande kommentarer på.
Exempel:

```
/(\s)(?!((?!"').)*"')(?!((?!<text>).)*<>)/ #Filters out all non-string or non-text-tag whitespaces.
```

Strängar
För strängar har vi främst använt singel citat-tecken.
Exempel:

```
'text'
```

Filnamn
Våra filnamn består endast av små tecken.
Exempel:
”nodes.rb”

Klassnamn

Namnen på våra klasser börjar på stor bokstav och det är små bokstäver i början av orden om namnet innehåller flera ord förutom det första ordet. Det används understreck mellan orden.
Exempel:

```
class Array_nodes
```

Variabelnamn
Namnen på våra variabler följer Ruby:s kodstandard för variabler vilket innebär små bokstäver med understreck mellan olika ord. Lokala metodvariabler har ett snabel-a framför namnet medan globala variabler har ett dollartecken framför sitt namn.
Exempel:

```
@local_variable
$global_variable
```

Funktionsnamn
Namnen på våra funktioner följer samma standard som för våra variabelnamn, små bokstäver med understreck mellan ord.
Exempel:

```
def function_name
```

3.2 Nodklasser

Parsern skapar ett träd av nodobjekt av klasserna nedan. Utifrån det färdiga trädet genereras HTML-kod när noderna evalueras. Varje nodklass har ett unikt syfte. WINAML använder följande nodklasser:

- **Constant_node** – sparar ett värde av text eller nummer.
- **Expression_node** – beräknar ett uttryck med två operander och en operator.
- **Variable_node** – hämtar ett variabelvärde ur en global variabelhashtabell.
- **Special_node** – hämtar en specialvariabel, t.ex. ”@item”, ur en global variabelhashtabell.
- **Array_node** – genererar HTML-kod för en tabell.
- **Array_box_node** – hämtar ett värde på en tabellrad ur en listvariabel.
- **Array_join_node** – slår samman Array_box_node:s eller Array_join_node:s.
- **Assign_node** – sparar variabeldata till en global variabelhashtabell. Hashnyckeln är variabelnamnet och hashvärdet är variabeldatan.
- **Condition_node** – får in ett booleansk värde och en eller två noder. Beroende på booleanens värde så evalueras en eller ingen av noderna.
- **Join_node** – slår samman olika sorters noder förutom Array_node.

3.3 Användning

WINAML kräver att Ruby finns installerat för att fungera korrekt. Du behöver även filerna ”rdparse.rb”, ”parse.rb” och ”nodes.rb”.

3.3.1 Installation - Linux

För att installera Ruby i Linux (Debian/Ubuntu) skriv följande i terminalen:

```
$ sudo apt-get install ruby-full
```

Kopiera filerna ”rdparse.rb”, ”parse.rb” och ”nodes.rb” till din projektmapp. Gå till projektmappen i terminalen och skriv:

```
$ ruby parse.rb
```

Skriv in sökvägen till din wml-fil som ska kompileras.

Exempel:

```
[WINAML] fil.wml
```

En ”fil.html” har skapats i projektmappen.

3.3.2 Installation - Windows

För att installera Ruby i Windows gå till http://www.ruby-lang.org/en/downloads/ och ladda hem Ruby X.X.X one-click installer.

Kopiera filerna ”rdparse.rb”, ”parse.rb” och ”nodes.rb” till din projektmapp. Kör ”parse.rb”.

Skriv in sökvägen till din wml-fil som ska kompileras.

Exempel:

```
[WINAML] fil.wml
```

En ”fil.html” har skapats i projektmappen.

4. Erfarenheter och reflektioner

Under konstruktionen av vårt språk fick vi stor nytta av våra erfarenheter ifrån att tidigare ha använt ”rdparse.rb” i kursen TDP007, Konstruktion av datorspråk.

Ett problem vi hade i början var att endast filtrera bort tomma mellanrum som inte fanns i text-taggar eller i strängar. Men efter lite efterforskning om mer avancerade reguljära uttryck så hittade vi en bra lösning till problemet. Lösningen blev följande reguljära uttryck:

```
/(\s)(?!((?!'").)*'")(?!((?!<text>).)*<>)/
```

och jämfört med vad vi hade i början.

```
/\s+/
```

Vi hade ett litet problem med prioriteten av beräkningar i aritmetiska uttryck och booleanska uttryck. Vi löste detta genom att dela upp vissa regler till flera olika regler med olika prioriteter. Till exempel fick de aritmetiska operatorerna multiplikation och

division en egen regel som har högre prioritet än regeln för addition och subtraktion.

Ett annat stort problem vi stötte på var att falska grenar evaluerades även när dom inte skulle evalueras.

Till exempel blev en variabel felaktigt tilldelad värdet 3 i följande villkorsuttryck.

WINAML-kod

```
<<variabel = 1>
<<if false>
        <<variabel = 3>
<<else>
        <text>Vår variabeln har värdet: #{variabeln}. Den bör ha värdet 1.<>
<<endif>
```

Utdata:

Vår variabel har värdet 3. Den bör ha värdet 1.

Problemet var att när parsern gick igenom wml-koden så evaluerade den all kod i villkorssatserna, även de grenar som inte skulle evalueras.

Vi löste detta med att bygga upp ett träd med nodklasser som endast evaluerade de rätta grenarna i villkorsatser. Ingenting i wml-koden parsas förrän hela trädet med nodklasser har byggts upp, till skillnad från innan vi använde nodklasser.

Ett annat svårt problem var när vi skulle matcha ”@item” i ”rdparse.rb” så hittades den aldrig ordet ”@item”. Det visade sig att vi hade glömt ett komma-tecken så att det blev en inparameter för lite.

```
match('#', '{', '@' 'item','}')          # Observera att ett kommatecken saknas.
match('#', '{', '@', 'item','}')
```

Referenser

Tryckta referenser

McConnel, Steve (2004). *Code Complete, Second Edition.* Microsoft Press.

Elektroniska referenser

[www] <http://www.ida.liu.se/~TDP019/ > hämtad den 2010-05-24. TDP019 Projekt: Datorspråk.

[www] <http://www.ida.liu.se/~TDP007/> hämtad den 2010-05-24. TDP007 Konstruktion av datorspråk.

[www] <http://www.ida.liu.se/~TDP007/material/examples/rdparse.rb> hämtad den 2010-05-24. TDP007 Konstruktion av datorspråk, Kursmaterial.

[www] <http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form> hämtad den 2010-05-24. Backus–Naur Form.

Bilaga 1 Exempel

Kommentering

WINAML-kod

!" Det här är en kommentar "!

Text – Hello World

WINAML-kod

<text>Hello World<>

Utdata

Hello World

Aritmetiskt uttryck - Beräkning

WINAML-kod

<text>#{3+6}<>

*<text>#{3*6}<>*

*<text>#{3*6/3-2}<>*

Utdata

9

18

4

Variabler - Sätta och hämta

WINAML-kod

<<var1 = "' Sätter in värde till variabel'">

<<var2 = var1 + "' och slår samman till en annan variabel.'">

<text>#{var2}<>

Utdata

Sätter in värde till variabel och slår samman till en annan variabel.

Villkor

WINAML-kod

```
<<antal_frukter = 5>
<<if (antal_frukter > 3)>
        <text>Det finns fler än 3 frukter<>
<<endif>

<<antal_butiker = 10>
<<if (antal_frukter > antal_butiker)>
        <text>Det finns fler frukter än antal butiker<>
<<else> !" Denna villkor kommer att uppfyllas "!
        <text>Det finns ej fler frukter än antal butiker<>
        <text>Det finns #{antal_butiker} butiker<>
<<endif>
```

Utdata

```
Det finns fler än 3 frukter
Det finns ej fler frukter än antal butiker
Det finns 10 butiker
```

Tabell

WINAML-kod

```
<<saker = [["'Glass'",5]["'Kaka'",2]]>
<table saker><text>#{@item}<><>
```

Utdata

```
Glass   5
Kaka    2
```

Importera andra wml-filer

Fil1.wml

WINAML-kod

<text>Fil1.wml har blivit importerad av fil2.wml<>

Fil2.wml

WINAML-kod

<text>Fil2.wml importerar fil1.wml<>

<<include "'Fil1.wml'">

Utdata

Fil2.wml importerar fil1.wml

Fil1.wml har blivit importerad av fil2.wml

Bilaga 2 Grammatik – BNF

<start> ::= <setvars> <start> | <setvars> |
<getvars> <start> | <getvars> |
<tags> | <tags> <start> |
<condition> | <condition> <start>

<condition> ::= '<<if' <booleans> '>' <start> '<<else>' <start> '<<endif>' |
'<<if' <booleans> '>' <start> '<<endif>'

<booleans> ::= <boolean> |
<boolean> 'or' <booleans> |
<boolean> 'and' <booleans>

<boolean> ::= '(' <booleans> ')' |
<expr> |
'false' |
'true' |
<expr> <operator> <expr> |
'not(' <booleans> ')' |
'not' <boolean>

<operator> ::= '==' |
'!=' |
'<=' |
'>=' |
'>' |
'<'

<tags> ::= '<table' <expr> '>' <start> '<>' |
'<text><>' |
'<text>' <anytext> '>'

<anytext> ::= <token> <anytext> |
<token> '<' |
<getvars> <anytext> |
<getvars> '<'

<string> ::= <token> <string> |
<token> '\" |
<getvars> <string> |
<getvars> '\"

<token> ::= [0-9]+ |
[A-z0-9]+

<getvars> ::= '#{' <expr> '}' |/(\s)(?!((?!"").)*"")(?!((?!<text>).)*<>)/
'#{@' <var> '}' |
'#{@item}'

<setvars> ::= '<<' <var> '=[' <array> ']>' |
'<<' <var> '=' <expr> '>' |
'<<@' <var> '=' <expr> '>'

<array> ::= '[' <box> ']' |
'[' <box> ']' <array>

```
<box> ::=       <expr> |
                <expr> ',' <box>

<var> ::=       [a-zA-Z][a-zA-Z0-9]*

<expr> ::=      <term> |
                <expr> '-' <term> |
                <expr> '+' <term>

<term> ::=      <atom> |
                <term> '/' <atom> |
                <term> '*' <atom>

<atom> ::=      '(' <expr> ')' |
                '"\" <string> '"' |
                <var> |
                [0-9]+
```

Bilaga 3 Programkod

Parse.rb

```
require 'rdparse'
require 'nodes'

class Winaml

 def init_html()
  # Set default values for html document.
  @html_output = {
   'html_begin' => "
<!DOCTYPE html>
<html lang=\"#{$vars['@language']}\">
      <head>
           <meta charset=\"utf-8\" />
           <title>#{$vars['@title']}</title>
      </head>

      <body>\r\n",
   'html_end' => "\r\n\t</body>\r\n</html>"
  }

  return @html_output
 end

 def initialize

  @winaml_parser = Parser.new("winamlParser") do
   # Filters out all comments.
   token(/!"(.*?)"!/)
   # Escape '<' to html.
   token(/\</) { |m| '&lt;' }
   # Escape '>' to html.
   token(/\>/) { |m| '&gt;' }
   # Filters out spaces except in text-tags or strings.
   token(/(\s)(?!((?!"').)*'")(?!((?!<text>).)*<>)/)
   # Match words.
   token(/[a-zA-Z][a-zA-Z0-9]*/) { |m| m }
   # Match integers.
   token(/\d+/) {|m| m.to_i }
   # Match rest, one sign.
   token(/./) {|m| m }

   start :program do
```

```
  match(:condition, :program) { |a,b| Join_node.new(a, b) }
  match(:condition) { |a| a }
  match(:tags, :program) { |a,b| Join_node.new(a, b) }
  match(:tags) { |a| a }
  match(:getvars, :program) { |a,b| Join_node.new(a, b) }
  match(:getvars) { |a| a }
  match(:setvars,:program) { |a,b| Join_node.new(a, b) }
  match(:setvars) { |a| a }
end

# if-statement
# eg. <<if true> X <endif>
rule :condition do
  match('<', '<', 'if', :booleans, '>', :program, '<', '<', 'endif', '>'){
    |_,_,_,a,_,b,_,_,_,_| Condition_node.new(a, b)}
  match('<', '<', 'if', :booleans, '>', :program, '<', '<', 'else', '>',
        :program, '<', '<', 'endif', '>') {
    |_,_,_,a,_,b,_,_,_,_,c,_,_,_,_| Condition_node.new(a, b, c) }
end

# Boolean expressions.
rule :booleans do
  match(:boolean, 'and', :booleans) { |a,_,b|
    Expression_node.new(:evaland, a, b) }
  match(:boolean, 'or', :booleans) { |a,_,b|
    Expression_node.new(:evalor, a, b) }
  match(:boolean) { |a| a }
  #match('(',:booleans,')') { |_,a,_| a }
end

rule :boolean do
  match('not', :boolean) { |_,a| Expression_node.new(:evalnot, a, a) }
  match('not', '(', :booleans, ')') { |_,_,a,_|
    Expression_node.new(:evalnot, a, a) }
  match(:expr, :operator, :expr) { |a,b,c| Expression_node.new(b, a, c) }
  match('true') { |a| Constant_node.new(true)}
  match('false') { |a| Constant_node.new(false)}
  match(:expr) { |a| a}
  match('(',:booleans,')') { |_,a,_| a }
end

rule :operator do
  match('=', '=') { |_,_| :== }
  match('!', '=') { |_,_| :!= } #Ruby1.9 needed to use ":!=" as a symbol.
  match('>') {|_| :> }
  match('<') {|_| :< }
```

```
  match('>', '=') {|_| :>= }
  match('<', '=') {|_| :<= }
end

rule :tags do
  # Text tag.
  # eg. <text>Hello<>
  match('<','text','>', :anytext, '>') { |_,_,_,a,_|
    Join_node.new(Join_node.new(Constant_node.new("\t<p>"), a),
                  Constant_node.new("</p>\r\n"))
  }
  match('<','text','>', '<', '>') { |_,_,_,_,_|
    Constant_node.new("\t<p></p>\r\n")
  }

  # Table tag.
  # eg. <table data> X <>
  match('<','table', :expr,'>', :program,'<', '>') { |_,_,a,_,b,_,_|
    Array_node.new(a, b)
  }
end

# The text in text tags.
rule :anytext do
  match(:getvars, '<') { |a,_| a }
  match(:getvars, :anytext) { |a,b| Join_node.new(a, b) }
  match(:token, '<') {|a,_| a }
  match(:token, :anytext) { |a,b| Join_node.new(a, b) }
end

# String datatype.
rule :string do
  match(:getvars, '\'') { |a,_| a }
  match(:getvars, :string) { |a,b| Join_node.new(a, b) }
  match(:token, '\'') {|a,_| a }
  match(:token, :string) { |a,b| Join_node.new(a, b) }
end

rule :token do
  match(String) { |a| Constant_node.new(a) }
  match(Integer) { |a| Constant_node.new(a.to_s) }
end

# Evaluate WINAML programming code.
rule :getvars do
  match('#', '{', '@', 'item','}') { |_,_,_,_,_|
```

```
    Special_node.new('item') }
  match('#', '{', '@', :var,'}') { |_,_,_,a,_| a.value = "@" + a.value;
    Variable_node.new(a) }
  match('#', '{',:expr,'}') { |_,_,a,_| a }

end

rule :setvars do
  # Declare or assign a special variable to an expression.
  match('<', '<', '@', :var,'=',:expr,'>') { |_,_,_,a,_,b,_| a.value =
    "@" + a.value; Assign_node.new(a, b) }

  # Declare or assign a variable to an array.
  match('<', '<', :var,'=','[',:array,']', '>') { |_,_,a,_,_,b,_,_|
    Assign_node.new(a, b) }

  # Declare or assign a variable to an expression.
  match('<', '<', :var,'=',:expr,'>') { |_,_,a,_,b,_|
    Assign_node.new(a, b) }

end

# Array datatype.
rule :array do
  match('[', :box, ']', :array) { |_,a,_,b|
    Array_join_node.new(Array_box_node.new(a), b) }
  match('[', :box, ']') { |_,a,_| Array_box_node.new(a) }
end

rule :box do
  match(:expr, ',', :box) { |a,_,b|
    Array_join_node.new(Array_box_node.new(a) ,b) }
  match(:expr) { |a| Array_box_node.new(a) }
end

# Acceptable variablename.
rule :var do
  match(/[a-zA-Z][a-zA-Z0-9]*/) { |a| Constant_node.new(a) }
end

# Arithmetic expressions.
rule :expr do
  match(:expr, '+', :term) { |a, _, b| Expression_node.new(:+, a, b) }
  match(:expr, '-', :term) { |a, _, b| Expression_node.new(:-, a, b) }
  match(:term)
end
```

```
    rule :term do
      match(:term, '*', :atom) { |a, _, b| Expression_node.new(:*, a, b) }
      match(:term, '/', :atom) { |a, _, b| Expression_node.new(:/, a, b) }
      match(:atom)
    end

    rule :atom do
      # Match an Integer, variable, WINAML-string or an (expression).
      match(Integer) { |a| Constant_node.new(a) }
      match(:var) { |a| Variable_node.new(a) }
      match('"',/"/,:string,'"') { |_,_,a,_| a }
      match('(', :expr, ')') { |_, a, _| a }
    end
  end
end

# Exit WINAML.
def done(str)
  ["quit","exit","bye",""].include?(str.chomp)
end

# WINAML "include"-function.
def include_file(data)
  found = false
  data = data.gsub(/<<include "'(.*?)'">/) {|m| found = true;
    File.new(m.sub("<<include \"", '').sub("'\">", '')).read()}
  if (found) then
    data = include_file(data)
  end
  data
end

#For testing purposes.
def test(str)
  $vars = {}
  log(false)
  parsed_data = @winaml_parser.parse(str)
  return parsed_data.eval().to_s
end

def parse()
  log(false)
  # user input
  print "[WINAML] "
  str = gets
```

```
    str = str.strip
    if done(str) then
      puts "Bye."
    else
      if str.index('.wml') then
        data = File.new(str).read()
        # Check for WINAML "include"-code before parse.
        data = include_file(data)
        write_file = File.new(str.gsub(".wml",".html"), 'w')
        parsed_data = @winaml_parser.parse(data)
        parsed_data_eval = parsed_data.eval().to_s
        html = init_html()
        write_file << html['html_begin']
        write_file << parsed_data_eval
        write_file << html['html_end']
        puts html['html_begin'] + parsed_data.eval().to_s + html['html_end']
        write_file.close
      else
        puts "Wrong file format"
      end
      parse
    end
  end

  def log(state = true)
    if state
      @winaml_parser.logger.level = Logger::DEBUG
    else
      @winaml_parser.logger.level = Logger::WARN
    end
  end
end

# Start WINAML parser.
Winaml.new.parse
```

nodes.rb

```
# Set default value for html document
$vars = {
  '@title' => 'New WINAML document',
  '@language' =>'sv'
}

# Stack used for arrays.
$stack = 0
$stack_vars = {}

# Any text or number
class Constant_node
  attr_accessor :value
  def initialize(const)
    @value = const
  end
  def eval()
    return @value
  end
end

# For Expression_node at @operator
def evalnot(op)
  not op
end
def evaland(op)
  self and op
end
def evalor(op)
  self or op
end

class Expression_node
  attr_accessor :operand1, :operand2, :operator
  def initialize(op, op1, op2)
    @operand1 = op1
    @operand2 = op2
    @operator = op
  end
  def eval()
    return @operand1.eval().send(@operator, @operand2.eval())
  end
end
```

```
class Variable_node
  attr_accessor :id
  def initialize(id)
    @id = id
  end
  def eval()
    return $vars[@id.eval()] # Variable "@id" is evaled to get the correct value
  end
end

# Special variable
class Special_node
  attr_accessor :id
  def initialize(id)
    @id = id
  end
  def eval()

    if (@id == 'item') then
      return $stack_vars[$stack-1].eval()
    end
  end
end

class Array_node
  attr_accessor :node_var, :node_table
  # Gets a Variable_node and a Array_box_node or Array_join_node
  def initialize(node_var, node_table)
    @node_var   = node_var
    @node_table = node_table
  end
  def eval()
    value = @node_var.eval()

    stack = $stack
    $stack = $stack + 1
    html = "\t<table>\r\n"
    value.each do |row|
      value = row.eval()
      html += "\t<tr>\r\n"
      value.each do |col|
        html += "\t\t<td>\r\n"
        # For @item variable
        $stack_vars[stack] = col
        html += @node_table.eval()
```

```
      html += "\t\t</td>\r\n"
    end
    html += "\t</tr>\r\n"
  end
  html += "</table>\r\n"

  $stack = $stack - 1

  return html
 end
end

class Array_box_node
 attr_accessor :node
 def initialize(node)
  @node = node
 end
 def eval()
  [@node]
 end
end

class Array_join_node
 attr_accessor :node1, :node2
 def initialize(node1, node2)
  @node1 = node1
  @node2 = node2
 end
 def eval()
  @node1.eval() + @node2.eval()
 end
end

# Assign to variable
class Assign_node
 attr_accessor :var, :expr
 def initialize(var, expr)
  @var = var
  @expr = expr
 end
 def eval()
  value = @expr.eval()
  $vars[@var.eval()] = value
  nil
 end
end
```

```
class Condition_node
  attr_accessor :boolean, :node_true, :node_false
  def initialize(boolean, node_true, node_false=false)
    @boolean = boolean
    @node_true = node_true
    @node_false = node_false
  end
  def eval()
    if (@boolean.eval()) then
      @node_true.eval()
    else
      if (node_false) then
        @node_false.eval()
      end
    end
  end
end

# Merge other nodes
class Join_node
  attr_accessor :node1, :node2
  def initialize(node1, node2)
    @node1 = node1
    @node2 = node2
  end
  def eval()
    @node1.eval().to_s + @node2.eval().to_s
  end
end
```