Compiler Construction
RM - Programming Language

Mashal ZAINAB and Ribiea RAMZAN

Presentation Outline

RM Description

- RM is a basic procedural programming language.
- Typed language.
- Single line comments begin with $ and end with end of line.
- Multi line comments begin with $$ and end with $$.
- Types: Integer, Float, String, Boolean, Null, List and Arrays.
- Reserved Keywords: If , else , else if, for, while, function, foreach, continue, break, pass, return, switch, start, end, read, out, in.
- Variables:

Tokens - Data Types

- ➔ **Integer** -> [‘1’ - ‘9’] ([‘0’ - ‘9’])* | ‘0’
- ➔ **SpecialCharacter** ->
- ➔ “ , @ . - _ ‘ | ! “ % & \ / ( ) [ ] = + * # $ < > : ; ^ ? “
- ➔ **Letter** -> [‘a’ - ‘z’ , ‘A’ - ‘Z’]
- ➔ **String** -> Character String
- ➔ **Character** -> Letter | SpecialCharacter | Integer | ε
- ➔ **Variable** -> Letter (Letter | Integer |’_’)* | ‘_’ (Letter | Integer |’_’)*
- ➔ **Boolean** -> ‘True’ | ‘False’
- ➔ **Float** -> (Integer)$^+$ ‘.’ (Integer)$^+$
- ➔ **Null** -> ‘NULL’
- ➔ **List** -> ‘[‘ (ListExpression) ‘]’
- ➔ **ListExpression** -> Object (‘,’ Object)* | ε
- ➔ **Object** -> Integer | Float | String | Boolean | Null | List | Array
- ➔ **ObjectType** -> ‘integer’ | ‘float’ | ‘string’ | ‘boolean’
- ➔ **Array** -> ‘[‘ (ArrayExpression) ‘]
- ➔ **ArrayExpression** -> Integer (‘,’ Integer)* | Float (‘,’ Float)* | String (‘,’ String)* | Boolean (‘,’ Boolean)* | Array | ε

Arithmetic Expressions

ArithmeticExpression -> ArithmeticTerm_1 ((‘>>’ | ‘<<’) ArithmeticTerm_1)*

$\textbf{ArithmeticTerm}_1$ -> ArithmeticTerm_2 ((‘+’ | ‘-’) ArithmeticTerm_2)*

$\textbf{ArithmeticTerm}_2$ -> ArithmeticTerm_3 ((‘*’ | ‘/’ | ‘%’) ArithmeticTerm_3)*

$\textbf{ArithmeticTerm}_3$ -> ArithmeticTerm_4 | ‘-’ ArithmeticTerm_4

$\textbf{ArithmeticTerm}_4$ -> ArithmeticTerm_5 (‘**’ ArithmeticTerm_5)*

$\textbf{ArithmeticTerm}_5$ -> ArithmeticTerm_6 | ArithmeticTerm_6 ‘++’ | ArithmeticTerm_6 ‘--’

$\textbf{ArithmeticTerm}_6$ -> ArithmeticTerm_7 | ‘++’ ArithmeticTerm_7 | ‘--’ ArithmeticTerm_7

$\textbf{ArithemticTerm}_8$ -> ‘(‘ ArithmeticExpression ‘)’ | Integer | Float | Variable | FunctionCall

Relational Expressions

RelationalExpression -> RelationalTerm$_1$ ((‘!=’ | ‘==’ | ‘>=’ | ‘<=’ | ‘>’ | ‘<’) ConditionalTerm$_1$)*

RelationalTerm$_1$ -> ‘(‘ RelationalExpression ‘)’ | ArithmeticExpression | Variable | Integer | FunctionCall

Logical Expressions

LogicalExpression -> LogicalTerm_1 (‘||’ LogicalTerm_1)*

$\textbf{LogicalTerm}_1$ -> LogicalTerm_2 (‘&&’ LogicalTerm_2)*

$\textbf{LogicalTerm}_2$ -> LogicalTerm_3 (‘|’ LogicalTerm_3)*

$\textbf{LogicalTerm}_3$ -> LogicalTerm_4 (‘^’ LogicalTerm_4)*

$\textbf{LogicalTerm}_4$ -> LogicalTerm_5 (‘&’ LogicalTerm_5)*

$\textbf{LogicalTerm}_5$ -> ‘(‘ LogicalExpression ‘)’ | RelationalExpression | Boolean

Conditional Expressions

IfExpression -> ‘if’ ‘(‘ RelationalExpression ‘)’ ‘start’ (Statement)$^+$ ‘end’

(‘else if’ ‘(‘ RelationalExpression ‘)’ ‘start’ (Statement)$^+$ ‘end’)* |

‘if’ ‘(‘ RelationalExpression ‘)’ ‘start’ (Statement)$^+$ ‘end’

(‘else if’ ‘(‘ RelationalExpression ‘)’ ‘start’ (Statement)$^+$ ‘end’)*

‘else’ ‘start’ (Statement)$^+$ ‘end’

TernaryExpression -> RelationalExpression ‘?’ Expression ‘:’ Expression

Loops

ForExpression -> ‘for’ ‘(‘ Expression ‘;’ RelationalExpression ‘;’ ArithmeticExpression ‘)’

‘start’ (Statement)$^+$ ‘end’

WhileExpression -> ‘while’ ‘(‘ RelationalExpression ‘)’

‘start’ (Statement)$^+$ ’end’

ForEachExpression -> ‘foreach’ ‘(‘ Variable ‘in’ (Array | List) ‘)’

‘start’ (Statement)$^+$ ‘end’

Functions

FunctionExpression -> ‘function’ FunctionName ‘(‘ Arguments ‘)’
‘start’ (Statement)$^+$ ‘end’

Arguments -> (ObjectType Argument (‘,’ ObjectType Argument)*) | ε

Argument -> Integer | Float | String | Boolean | List | Array

FunctionName -> Letter (Letter | Integer |’_’)* | ‘_’ (Letter | Integer |’_’)*

FunctionCall -> FunctionName ‘(’ ‘)’ | FunctionName ‘(‘ (Argument (‘,’ Argument)*) ‘)’

Jump and Switch Expressions

JumpExpression -> ‘continue’ | ‘break’ | ‘pass’ | ‘return’ (ArithmeticExpression | TernaryExpression | LogicalExpression | FunctionCall)*

SwitchExpression -> ‘switch’ ‘(‘ SwitchTerm ‘)’

‘start’

(‘case’ ‘(‘ Object | SwitchTerm ‘)’ ‘start’ (Statement)* ‘end’)*

‘default’ ‘start’ (Statement)* ‘end’

‘end’

SwitchTerm -> Variable | FunctionCall | RelationalExpression | LogicalExpression

Assignments & Declaration

AssignmentOperator -> = | += | -= | *= | /= | >>= | <<=

Expression -> ObjectType Variable AssignmentOperator ExpressionType|

Variable AssignmentOperator ExpressionType

ExpressionType -> ArithmeticExpression | LogicalExpression | RelationalExpression

Declaration -> ObjectType Variable

UserInput -> ObjectType Variable AssignmentOperator ‘read()‘ |

Variable AssignmentOperator ‘read()’

Output -> ‘out’ ‘(‘ (Variable | Integer | Float | String) (‘,’ (Variable | Integer | Float | String))* ‘)’

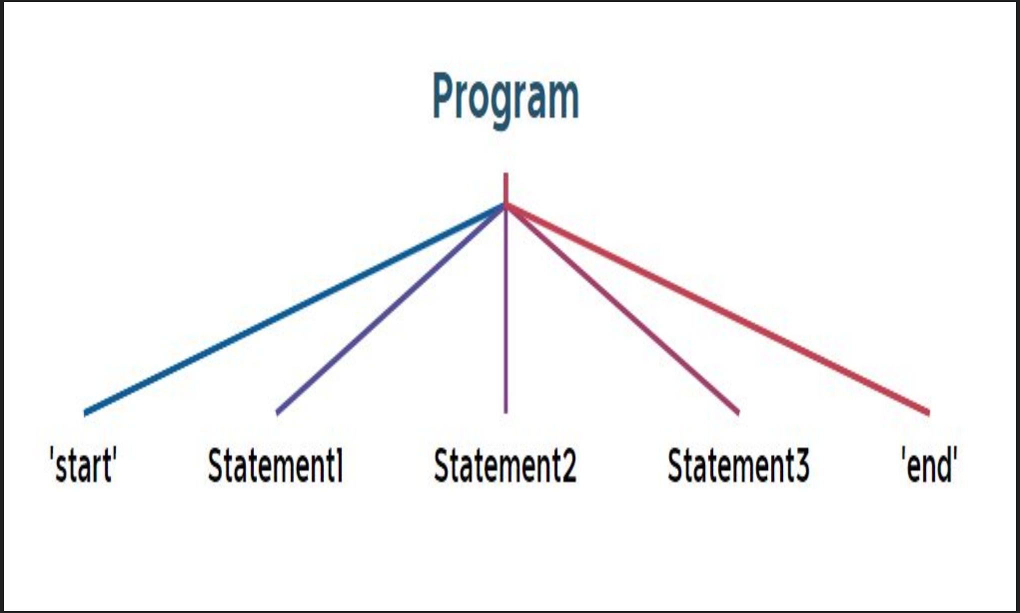
Statement -> (Expression | ForExpression | ForEachExpression | WhileExpression | IfExpression | TernaryExpression | Declaration | UserInput | Output | JumpExpression | FunctionCall) ‘;’

Program

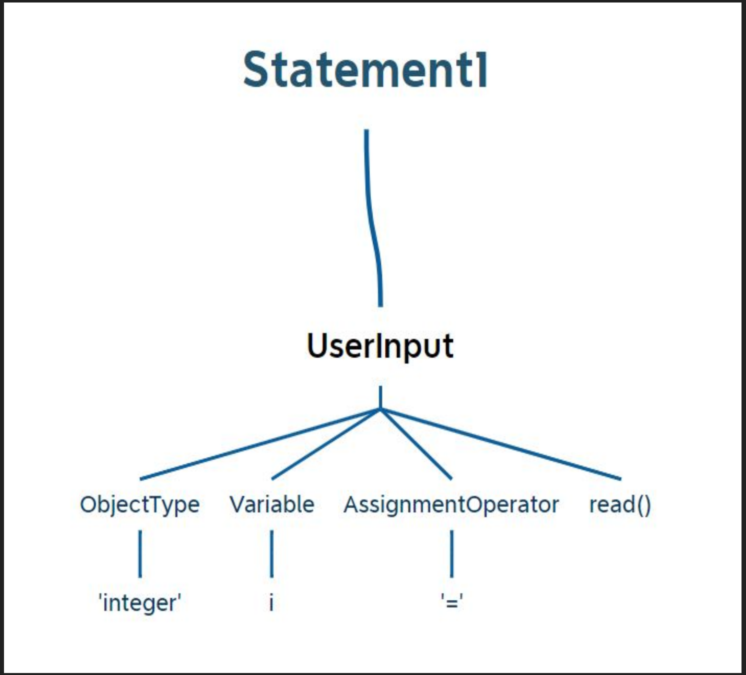
```
Program -> ‘start’
           ( Statement | FunctionExpression )*
           ‘end’
```

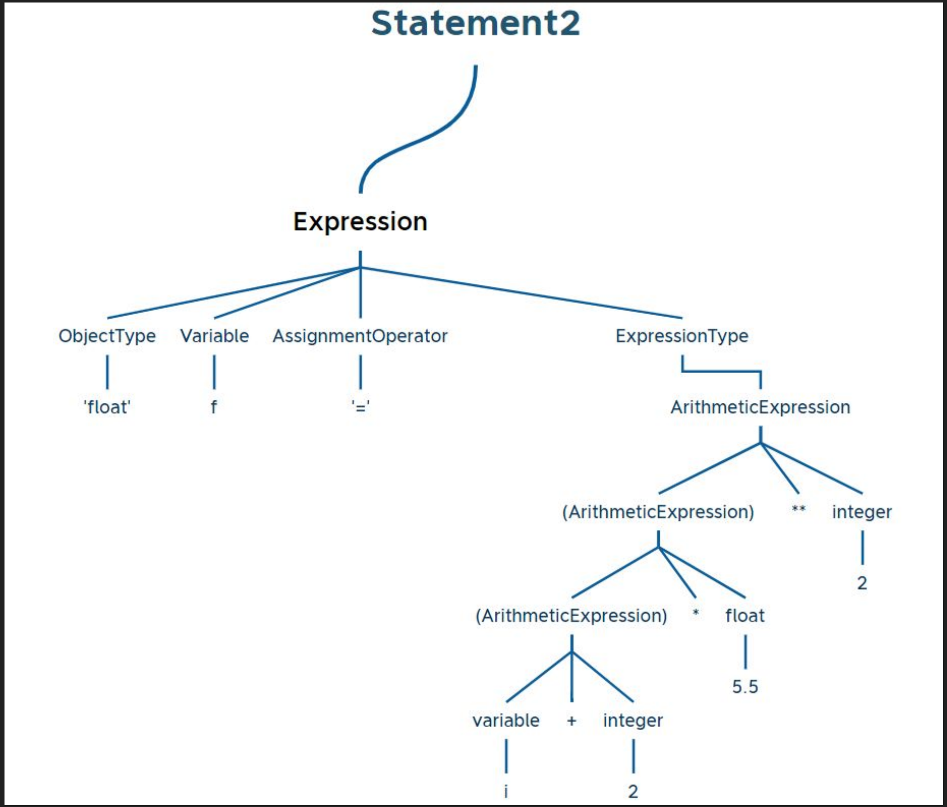
Example Program

```
start
integer i = read();
float f = ((i + 2)*5.5)**2;
if (f > 10)
start
out("The result is greater than 10:", f);
end
else
start
out("The result is:", i);
end
end
```

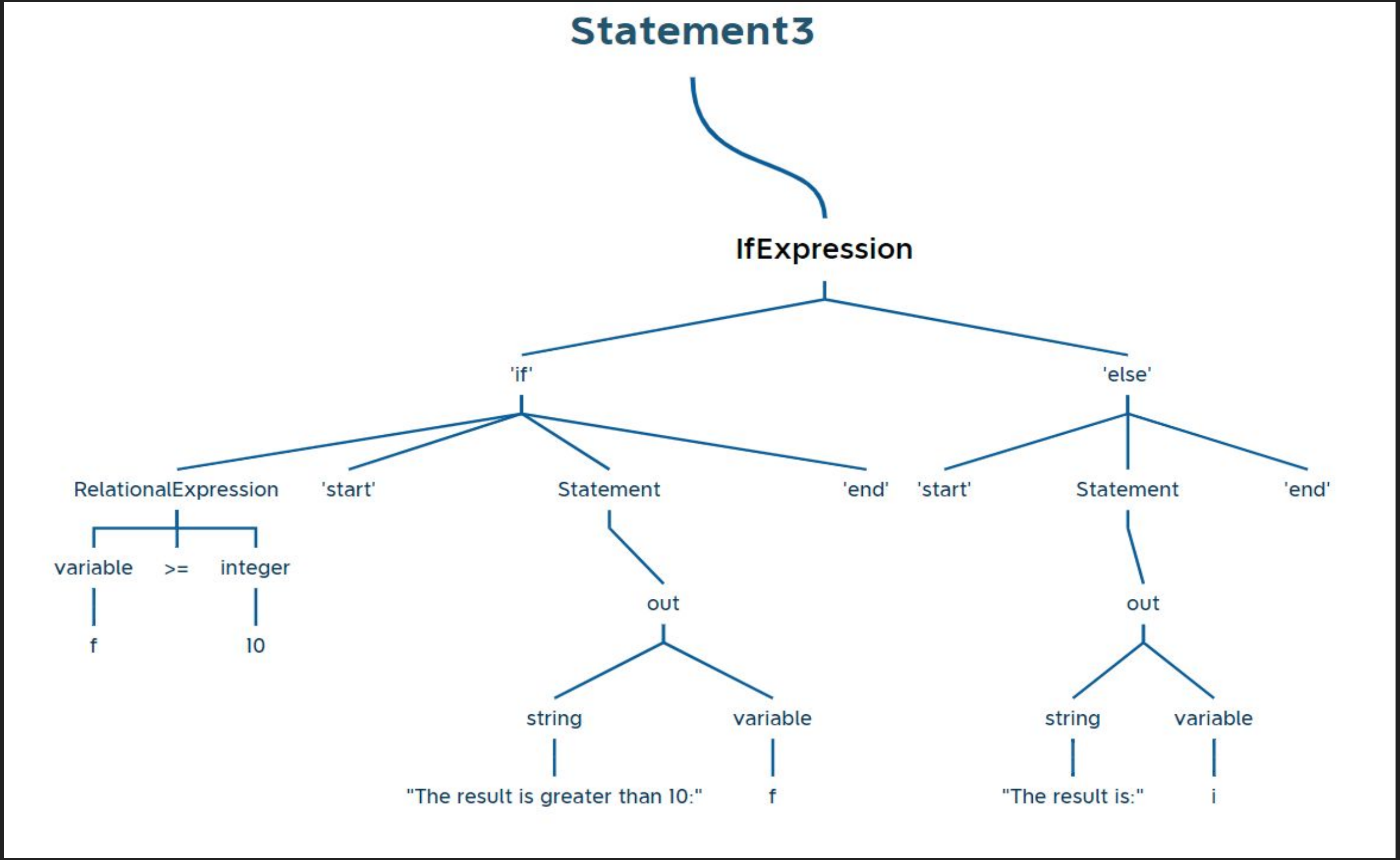


A parse tree of the above program





A parse tree of the above program contd.



Thank you!