SER 502 - Emerging programming languages and paradigms

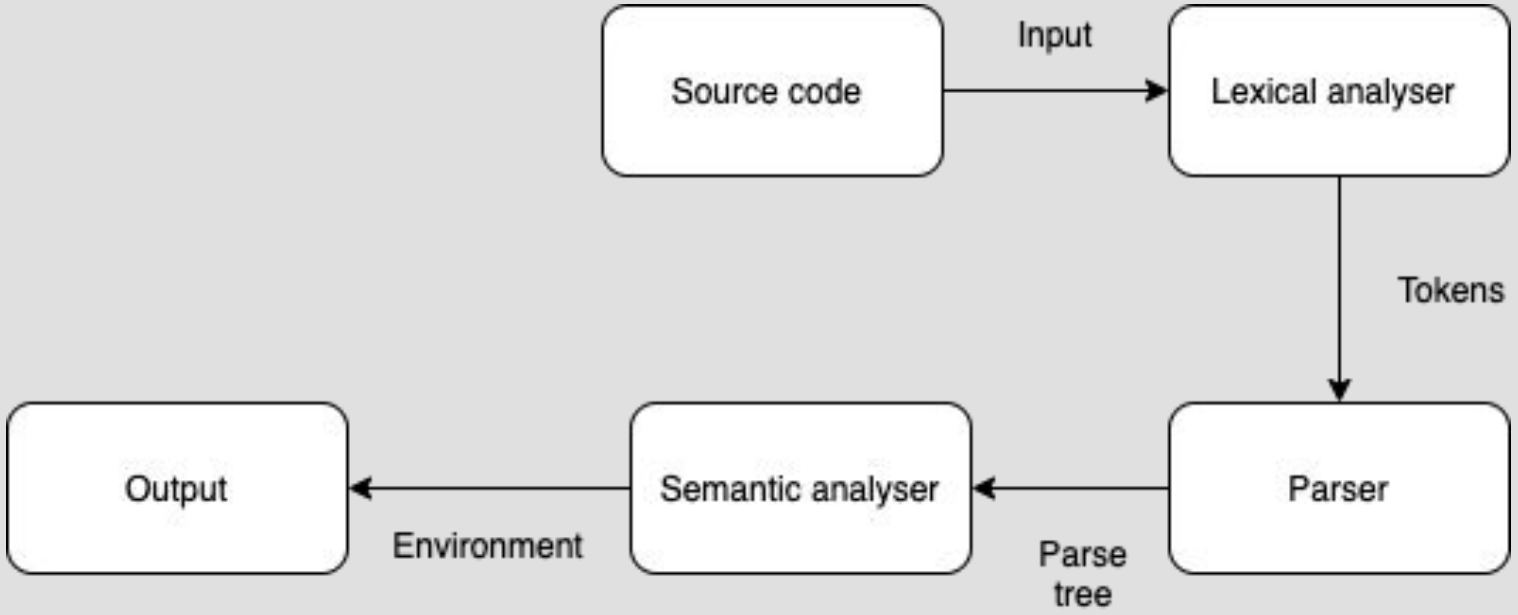
...

Trace by Team 22 ,
Akhilesh Krishnan-1217092786
Rahul Suresh - 1217112143
Idhant Haldankar - 1217178677
Swarnalatha Srenigarajan - 1217035534

Topics covered in this ppt

- Grammar used for the language
- Lexical Analyzer
- Parser/ Syntax Tree
- Semantics
- Sample input and output

Language design flow:



Data types used in Trace

- Number : Represents integer or floating point number, 0 is assigned as a default value if the variable is not initialised.
- Boolean : Represents the type boolean. It takes values true or false and has the default value of false.
- String : It is a sequence of characters enclosed within double quotes. It is assigned “ “ (empty) by default.

Operators Supported by trace

Arithmetic operators

- Modulus : %
- Multiplication : *
- Division : /
- Addition : +
- Subtraction : -
- Increment : ++
- Decrement : --

Relational Operators

- Lesser : <
- Greater : >
- Less than or equal to : <=
- Greater than or equal to : >=
- Not equal to : !=
- Equality Check : ==

Logical Operator and Assignment operator

- And : and
- Or : or
- Not : not
- Equals : =

Decision construct used in Trace

The “if (condition) then” statement executes a block if the condition evaluates to true. If the condition evaluates to false, the block following the “else then” gets executed. This is used to execute different blocks based on different conditions. The keyword “else if (condition) then” is used to check for another condition when the “if (condition) then” condition evaluates to false.

Examples of conditionals:

1. if (x == 2) then { x = x + 2 }
   else then { x = x + 1}
2. if (x == 2) then {x = x + 2}
   else if (x == 3) then { x = x + 1 }
   else then { x = x + 3}
3. if (x == 2) then { x = x + 2 }

Iterative constructs supported by trace

Simple for loop

A loop statement starts with “for” followed by parenthesis (). Inside the parentheses we have three parts which are namely “declaration”, “conditional expression” and “increment statement”. In declaration we would declare the iterable variable, later the bool_expr will evaluate the iteration limit for the loop and lastly the update statement will update the iterable variable.

Examples of Simple for pattern :

```
for(i=0; i<10; i=i+1;) {
    i = i+ 1;
    }
```

Range pattern for loop

A loop statement starts with “for” followed by iterable variable initialization.In declaration we would declare the iterable variable, and range of iterable values the iterable variable can take. The range function generates a list of iterable values which the iterable variable can take in each iteration.

Examples of Simple for pattern :

```
for i in range(1,10) {
    i = i+ 1;
    }
```

Simple While loop

The while loop will begin with the “while” keyword followed by parenthesis (). Inside the parentheses the bool_expr, which gets executed each time the while loop is called. The loop continues until bool_expr return true and ends when it returns false.

Examples of Simple for pattern :

```
while (i < 10){
    i = i+ 1;
    }
```

Grammar for trace - Program

```
program -> [execute], block.
block -> ['{'], declarationList, statementList, ['}'].
```

Grammar for trace - Declaration

```
declarationList -> declaration, [';'], declarationList. declarationList -> [].
%------
declaration -> numberDeclaration.
declaration -> booleanDeclaration.
declaration -> stringDeclaration.
%------
numberDeclaration -> [number], var_name, ['='], number_expr.
numberDeclaration -> [number], var_name.
%------
booleanDeclaration -> [bool], var_name, ['='], bool_expr.
booleanDeclaration -> [bool], var_name.
%------
stringDeclaration -> [string], var_name, ['='], string_expr.
stringDeclaration -> [string], var_name.
```

Grammar for trace - Statements

```
/* Statements part */
statementList -> statement, [';'], statementList.
statementList -> loopStatement, statementList.
statementList -> conditionalStatement, statementList.
statementList -> [].
%-----
statement -> print_statement.
statement -> var_name, ['='], number_expr.
statement -> var_name, ['='], bool_expr.
statement -> var_name, ['='], string_expr.
statement -> var_name, ['++'].
statement -> var_name, ['--'].
%---
print_statement -> [print],['('], string_expr, [')'].
print_statement -> [println],['('], string_expr, [')'].
%---
```

Grammar for trace - Loop and conditional statements

```
/* Looping Statements -> (for loop, for in range() loop and while loop ) */
loopStatement -> [for], ['('], declaration, [';'], bool_expr , [';'], statement, [')'], block.
loopStatement -> [for], var_name, [in], [range], ['('], number, [','], number ,[')'], block.
loopStatement -> [while],['('], bool_expr ,[')'], block.


/* Conditional Statements -> (empty, if then, if then else, if then else if... ) */
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block.
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block , [else], [then], block.
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block , [else], conditionalStatement.
conditionalStatement -> bool_expr, ['?'], statement, [':'], statement.
number_expr-> [len],['('],string_expr,[')'].
number_expr -> number_expr, ['+'], level_1.
number_expr -> number_expr, ['-'], level_1.
number_expr -> level_1.
level_1 -> level_1, ['*'], level_2.
level_1 -> level_1, ['/'], level_2.
level_1 -> level_1, ['%'], level_2.
level_1 -> level_2.
level_2 -> ['('], number_expr, [')'].
level_2 -> number.
level_2 -> var_name.
```

Grammar for trace - Boolean Expressions

```
/* Boolean Expression -> (true, false, not, and ,or, ==, !=, >, <, >=, <=) */
bool_expr -> [not], bool_expr.
bool_expr -> bool_expr, [and] , bool_expr.
bool_expr-> bool_expr, [or] , bool_expr.
bool_expr -> number_expr, ['='], ['='], number_expr.
bool_expr -> number_expr, ['!'], ['='], number_expr.
bool_expr -> number_expr, ['>'], number_expr.
bool_expr -> number_expr, ['<'], number_expr.
bool_expr -> number_expr, ['>'], ['='], number_expr.
bool_expr -> number_expr, ['<'], ['='], number_expr.
bool_expr -> string_expr, ['='], ['='], string_expr.
bool_expr -> string_expr, ['!'], ['='], string_expr.
bool_expr -> bool_expr, ['='], ['='], bool_expr.
bool_expr -> bool_expr, ['!'], ['='], bool_expr.
bool_expr -> var_name.
bool_expr -> ['true'].
bool_expr -> ['false'].
```

Grammar for trace - Boolean Expressions

```
/* Boolean Expression -> (true, false, not, and ,or, ==, !=, >, <, >=, <=) */
bool_expr -> [not], bool_expr.
bool_expr -> bool_expr, [and] , bool_expr.
bool_expr-> bool_expr, [or] , bool_expr.
bool_expr -> number_expr, ['='], ['='], number_expr.
bool_expr -> number_expr, ['!'], ['='], number_expr.
bool_expr -> number_expr, ['>'], number_expr.
bool_expr -> number_expr, ['<'], number_expr.
bool_expr -> number_expr, ['>'], ['='], number_expr.
bool_expr -> number_expr, ['<'], ['='], number_expr.
bool_expr -> string_expr, ['='], ['='], string_expr.
bool_expr -> string_expr, ['!'], ['='], string_expr.
bool_expr -> bool_expr, ['='], ['='], bool_expr.
bool_expr -> bool_expr, ['!'], ['='], bool_expr.
bool_expr -> var_name.
bool_expr -> ['true'].
bool_expr -> ['false'].
```

Grammar for trace - String Expressions

```
/* String Expression -> checks for string type.*/
string_expr -> string.
string_expr -> var_name.
string_expr -> [str],['('], number_expr, [')'].
string_expr -> [str],['('], bool_expr, [')'].
string_expr -> string_expr, ['+'], string_expr.
```

Sample runs of codes in Trace

Sample run 1 - testing boolean expressions

Program

```
execute{
    bool a = true;
    bool b = false;
    bool c = a and b;
    bool d = a or b;
    bool e = not a;
    println(a);
    println(b);
    println(c);
    println(d);
    println(e);

}
```

Output

```
[true
 false
 false
 true
 false
```

Sample run 2 - testing numeric expressions

Program

```
execute{
    number a = 1;
    number b = 2;
    number c = a + b;
    number d = a - b;
    number e = a * b;
    number f = a / b;
    println(a);
    println(b);
    println(c);
    println(d);
    println(e);
    println(f);
}
```

Output

```
1
2
3
-1
2
0.5
```

Sample run 3 - testing string expressions

Program

```
execute{
    string a = "swarna";
    print(a);

}
```

Output

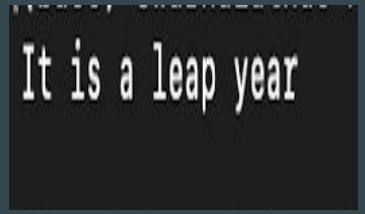
```
swarna
```

Sample run 4 - testing if-then, else-then statements

Program

```
execute{
number year = 2000;
if(year%4 == 0) then {
    if(year % 100 == 0) then{
        if(year % 400 == 0) then{
            print("It is a leap year");
        } else then {
           print("it is not a leap year");
        }
    } else then{
        print("It is a leap year");
    }
}
else then{
   print("it is not a leap year");
}
}
```

Output

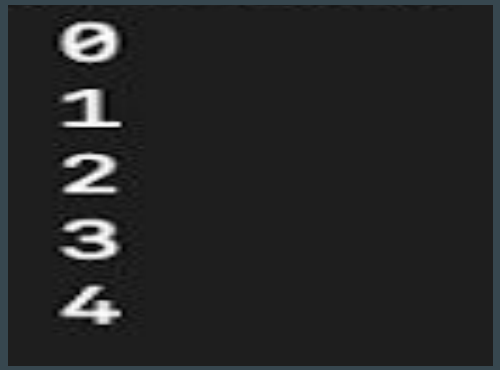


Sample run 5 - testing simple for loop statements

Program

```
execute{
for (number i = 0; i < 5; i++) {
  println(i);
}
}
```

Output



Future works

- Adding support for functions
- Including switch conditional statement