Compiler Design – 19CSE401

Language Chosen – Scala
Group - 7

S.No	Name	Roll No
1	Penugonda Tandava Sai Naga Koushik	CB.EN.U4CSE19449
2	Ravella Abhinav	CB.EN.U4CSE19453
3	Singadi Shanthan Reddy	CB.EN.U4CSE19459

Introduction:

Scala is a Java-like programming language which unifies object-oriented and functional programming. It is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes. Scala also supports a general notion of pattern matching which can model the algebraic types used in many functional languages.

Important Features of Scala:

- It's a high-level programming language
- It has a concise, readable syntax
- It's statically-typed (but feels dynamic)
- It has an expressive type system.
- It's a functional programming (FP) language
- It's an object-oriented programming (OOP) language
- It supports the fusion of FP and OOP
- Contextual abstractions provide a clear way to implement term inference
- It runs on the JVM (and in the browser)
- It interacts seamlessly with Java code
- It's used for server-side applications (including micro services), big dataapplications, and can also be used in the browser with Scala.js

Lexars:

Reserved Words:

Name	Description
_	The wildcard operator, representing an expected value.
:	Delimits a value, variable, or function from its type.
@	Defines an annotation for a class or its member. Annotations are a JVM feature but are seldomly used in Scala, with @annotation.tailrecbeing a popular exception.
#	A type projection, which delimits a type from its subtype.
<-	Delimits a generator from its identifier in a for-loop.
←	A single-character (\u2190) alternative to <-.
<:	The upper-bound operator, restricting types to those that are equal to or extend the given type.
<%	The view-bound operator, allowing any type that may be treated as the given type.
=	The assignment operator.

=>	Used in match expressions and partial functions to indicate a conditional expression, in function types to indicate a return type, and in function literals to define the function body.
⇒	A single-character (\u21D2) alternative to =>.
>:	The lower-bound operator, restricting types to those that are equal to or are extended by the given type.
abstract	Marks a class or trait as being abstract and uninstantiable.
case	Defines a matching pattern in match expressions and partial functions.
catch	Catches an exception. An alternate syntax that predates the util.Try monadic collection.
class	Defines a new class.
def	Defines a new method.
do	Part of the do..while loop definition.
else	The second part of an if..else conditional expression.
final	Marks a class or trait as being nonextendable.
finally	Executes an expression following a try block.
for	Begins a for-loop.
if	The first part of an if..else conditional expression, or the main part of an if conditional statement.

implicit	Defines an implicit conversion or parameter.
import	Imports a package, class, or members of a class to the current namespace.
lazy	Defines a value as being lazy, only defined the first time it is accessed.
match	Begins a match expression.
new	Creates a new instance of a class.
null	A value that indicates the lack of an instance. Has the type Null.
object	Defines a new object.
override	Marks a value or method as replacing the member of the same name in a base type.
package	Defines the current package, an incremental package name, or a package object.
private	Marks a class member as being inaccessible outside the class definition.

protected	Marks a class member as being inaccessible outside the class definitionor its subclasses.
return	Explicitly states the return value for a method. By default, the last expression in a method is used as the return value.
sealed	Marks a class as only allowing subclasses within the current file.
super	Marks a class member reference as one in the base type, versus one overridden in the current class.
this	Marks a class member reference as one in the current class, versus aparameter with the same name.
throw	Raises an error condition that breaks the current flow of operation and only resumes if the error is *caught* elsewhere.
trait	Defines a new trait.
true	One of the two Boolean values.
try	Marks a range of code for catching an exception. An alternate syntaxthat predates the util.Try monadic collection.
type	Defines a new type alias.

val	Defines a new, immutable value.
var	Defines a new, mutable variable.
while	Part of the do..while loop definition.
with	Defines a base trait for a class.
yield	Yields the return value from a for-loop.

Tokens:

To construct tokens, characters are distinguished according to the following classes (Unicode general category given in parentheses):

1. Whitespace characters. \u0020 | \u0009 | \u000D | \u000A.
2. Letters, which include lower case letters (Ll), upper case letters (Lu),title case letters (Lt), other letters (Lo), letter numerals (Nl) and the two characters \u0024 '$' and \u005F '_'.
3. Digits '0' | … | '9'.
4. Parentheses '(' | ')' | '[' | ']' | '{' | '}'.
5. Delimiter characters '`' | ''' | '"' | '.' | ';' | ','.
6. Operator characters. These consist of all printable ASCII characters(\u0020 - \u007E) that are in none of the sets above, mathematical symbols (Sm) and other symbols (So).

Operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Arithmetic Operators:

Operator	Description
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus operator finds the remainder after division of one number by another

Relational Operators:

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Logical Operators:

Operator	Description
&&	It is called Logical AND operator. If both the operands are non zero then condition becomes true.
||	It is called Logical OR Operator. If any of the two operands is non zero then condition becomes true.
!	It is called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Bitwise Operators:

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The bit positions of the left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The Bit positions of the left operand value is moved right by the number of bits specified by the right operand.
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

Assignment Operators:

Operator	Description
=	Simple assignment operator, Assigns valuesfrom right side operands to left side operand
+=	Add AND assignment operator, It adds rightoperand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operandand assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operandand assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assignthe result to left operand
%=	Modulus AND assignment operator, It takesmodulus using two operands and assign the result to left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator

Operators Precedence in Scala:

Category	Operator	Associativity
Postfix	() []	Left to right
Unary	! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Semantics:

Semantic analysis is the task of ensuring that the declarations and statements of a program are semantically correct, i.e, that their meaning is clear and consistent with the way in which control structures and data types are supposed to be used.

Scala Type Inference:

Scala Type Inference makes it optional to specify the type of variable provided that type mismatch is handled. With type inference capabilities, we can spend less time having to write out things compiler already knows. The Scala compiler can often infer the type of an expression so we don't have to declare it explicitly.

Let us first have a look at the syntax of how immutable variables are declared in scala.

- val variable_name : Scala_data_type = value

Implicit Conversions in Scala:

Implicit conversions in Scala are the set of methods that are apply when an object of wrong type is used. It allows the compiler to automatically convert of one type to another.

Implicit conversions are applied in two conditions:

- First, if an expression of type A and S does not match to the expected expression type B.
- Second, in a selection e.m of expression e of type A, if the selector m does not represent a member of A.

In the first condition, a conversion is searched which is appropriate to the expression and whose result type matches to B. In the second condition, a conversion is searched which appropriate to the expression and whose result carries a member named m.

Type Casting in Scala:

A Type casting is basically a conversion from one type to another. In Dynamic Programming Languages like Scala, it often becomes necessary to cast from type to another.Type Casting in Scala is done using the **asInstanceOf[]** method.

Applications of asInstanceof method

- This perspective is required in manifesting beans from an application context file.
- It is also used to cast numeric types.
- It can even be applied in complex codes like communicating with Java and sending it an array of Object instances.

Lexer Grammer:

lexer grammar Scala_lexer;

KEYWORDS:

***