COMPILER PROJECT: BINGOLY

Faculty: BSSE - Morning - 5th semester
Professor: Miss Farheen Faisal
Course: Compiler Construction (CSSE- 501) (2+1)

CONTRIBUTORS:

❖ Amaim Shaikh	B19103008
❖ Hareem Saad	B19103019
❖ Neha Haroon	B19103047
❖ Syeda Sughra Raza	B19103065
❖ Waiza Waqar	B19103067

INTRODUCTION:-

Language Specification

Line terminators:

- ➔ ;

  Usage

  Int a = 5 **;**

OOP:

class operator:

- ➔ .

  Usage:

  class_name**.**class_attribute;

  Shape.sides;

  class_name**.**class_method();

object creation:

- ➔ new

  Usage

  Class_name object_name =

  new Class_constructor();

???

- ➔ this

  Usage

```
Class Shape
{
Int num_of_sides;
str color;
void Shape(int Sides , str
Color)
{
this.num_of_sides = Sides;
this.color = Color;
}
}
```

Operations:

+ Add

- sub

* Multiply

/ Divide

^ power

% mod

Identifiers

Declaration

- ➔ Data_type identifier_name ;

  Usage

  int a **;**

  str myname **;**

Initalization

➔Identifier = value;

Usage

```
a  = 5;
myname = "Bingoly";
```

Conditional statements:

➔If ( ) {...} else if ( ) {....} else {...}

Usage

```
If ( a + b == 4)
{
        print("Answer = 4");
} else if (a + b == 5)
{
        print("Answer = 5");
}else
{
        print(" Answer = neither 5 nor 4 ");
}
```

Loops:

➔while( ){...}

Usage

```
int i = 0;
while(i < 5)
{
print(i);
i++ ;
}
```

➔for (initiation ; stop_criteria ; increment / decrement){...};

Usage

```
for( int i = 0 ; i < 5 ; i ++ )
{
print(i);
};
```

Array:

Declaration

➔array[size] data_type name={...};

Usage

```
array[5] int arr = { 0 , 1 , 2 , 3 , 4 };
```

Call

➔name[index];

Usage

```
int Starter = arr[4];
```

Comments

➔//

➔**Usage**

```
// this is a comment
```

Functions:

- **Declare function:**

  ➔Return-type func-name(parameter_datatype parameters…){…};

  Usage

  ```
  Int Sumation(int a , int b )
  {
  returns a+b;
  }
  ```

- **Function call:**

  ➔call functionname();

  Usage

  ```
  call Sumation(2 + 2);
  ```

  - Will have a main function

Main Function:

➔main (){}

Usage

```
main()
{
        print("hello world!");
}
```

Return:

➔returns

Usage

```
Return-type
func-name(parameters)
{
…
returns Return-type-value
};
```

Outputs :

➔print ();

Usage:

```
Print("hello world!");
```

Inputs :

➔input();

Usage:

```
Int varName;
input(varName);
```

Concatenation

- Done by + operator

Usage:

```
Print("hello " + "world!");
```

Float

- Done by 1 operator

Usage:

```
Float x = -8`2;
```

Regex - Regular Expressions

CharacterConstant

```
public static boolean isCharacterConstant(String IC) {
   // tells matcher class to match identifier against this RE
   Pattern p = Pattern.compile("^[ \b\t\n\f\r\'\"]$");
   // matches against RE
   Matcher m = p.matcher(IC);
   // m.matches return true if matched
   return (m.matches());

}
```

Identifier

```
public static boolean isIdentifier(String Identifier) {

   // tells matcher class to match identifier against this RE
   Pattern p =
Pattern.compile("^[$][A-Za-z][A-Za-z0-9_]|[A-Za-z_][A-Za-z0-9_]*$")
   // matches against RE
```

```
    Matcher m = p.matcher(Identifier);
    // m.matches return true if matched
    return (m.matches());
}
```

Integer Constant

```
    public static boolean isIntegerConstant(String intC) {
        // tells matcher class to match identifier against this RE
        // Pattern p = Pattern.compile("^[A-Z][A-Za-z0-9]*+$");
        Pattern p = Pattern.compile("^[0-9]+|[+-][0-9]+$");
        // matches against RE
        Matcher m = p.matcher(intC);
        // m.matches return true if matched
        return (m.matches());
    }
```

Float

```
    public static boolean isFloat(String fl) {
        // tells matcher class to match identifier against this RE
        Pattern p =
Pattern.compile("^[0-9]*[`][0-9]+|[+-][0-9]+[`][0-9]+$");
        // matches against RE
        Matcher m = p.matcher(fl);
        // m.matches return true if matched
        return (m.matches());
    }
```

Puntuations

```
public static boolean isPunctuation(char toCheck) {
        // defined punctuations
        ArrayList<String> punc = new ArrayList<>();
        punc.add(","); // comma
        punc.add(".");
        punc.add(";");
        punc.add("+"); // concatenation
        // punc.add("'"); // single quotation
        // punc.add("\""); // double quotation
        // Round brackets
        punc.add("(");
        punc.add(")");
        // Square brackets
        punc.add("[");
        punc.add("]");
        // Curly brackets
        punc.add("{");
        punc.add("}");

        for (int i = 0; i < punc.size(); i++) {
            if (punc.get(i).equals(Character.toString(toCheck))) {
                return true;
            }
        }
        return false;
    }
```

Single Operators

```
public static boolean isSingleOperator(char toCheck) {
      // defined operators
      ArrayList<String> singleOperator = new ArrayList<>();
      // Arithmetic operators
      singleOperator.add("+");
      singleOperator.add("-");
      singleOperator.add("*");
      singleOperator.add("/");
      singleOperator.add("^");
      singleOperator.add("%");
      // Assignment operators
      singleOperator.add("=");
      singleOperator.add(">"); // greater than
      singleOperator.add("<"); // less than
      // Bitwise operators
      singleOperator.add("&"); // AND-b op
      singleOperator.add("|"); // OR-b-op
      singleOperator.add("#"); // XOR-b
      singleOperator.add("~"); // NOT-b
```

```
    for (int i = 0; i < singleOperator.size(); i++) {
        if (singleOperator.get(i).equals(Character.toString(toCheck))) {
            return true;
        }
    }
    return false;
}

// Double operators regex
public static boolean isDoubleOperator(String toCheck) {
    // defined operators
    ArrayList<String> doubleOperator = new ArrayList<>();
    // increment and decrement
    doubleOperator.add("++");
    doubleOperator.add("—");
    // Assignment operators
    doubleOperator.add("+=");
    doubleOperator.add("-=");
    doubleOperator.add("*=");
    doubleOperator.add("/=");
    doubleOperator.add("%=");
    doubleOperator.add("!=");
    doubleOperator.add("==");
    doubleOperator.add("<=");
    doubleOperator.add(">=");
    // Logical operators
    doubleOperator.add("||"); // OR-l-op
    doubleOperator.add("&&"); // AND-l-op
    doubleOperator.add("!!"); // NOT-l-op
    // Shift operators
    doubleOperator.add("<<"); // double shift left
    doubleOperator.add(">>"); // double shift right

    for (int i = 0; i < doubleOperator.size(); i++) {
        if (doubleOperator.get(i).equals(toCheck)) {
            return true;
        }
    }
    return false;
}
```

Keyword

```
static boolean isKeyword(String toCheck) {
    // defined keywords
    ArrayList<String> keywords = new ArrayList<>();

    keywords.add("int");
    keywords.add("float");
    keywords.add("bool");
    keywords.add("array");
    keywords.add("str"); // changed
    keywords.add("const"); // a final var ; not changing var
    keywords.add("char");
    keywords.add("void");
    keywords.add("break");
    keywords.add("case");
    keywords.add("continue");
    keywords.add("default");
```

```
        keywords.add("else");
        keywords.add("for");
        keywords.add("if");
        keywords.add("instanceof");
        keywords.add("returns");
        keywords.add("switch");
        keywords.add("while");
        keywords.add("super");
        keywords.add("this");
        keywords.add("true");
        keywords.add("false");
        keywords.add("print"); // changed
        keywords.add("input"); // changed
        keywords.add("Class");
        keywords.add("func"); // for declaration of function changed
        keywords.add("call"); // for calling a function changed
        keywords.add("abstract");
        keywords.add("inherits"); // extends changed
        keywords.add("implements");
        keywords.add("interface");
        keywords.add("new");
        keywords.add("static");
        keywords.add("main");
        keywords.add("public");
        keywords.add("private");
        keywords.add("protected");
        keywords.add("import");

        for (int i = 0; i < keywords.size(); i++) {
            if (keywords.get(i).equals(toCheck)) {
                // System.out.println("valid keyword");
            return true;
          }
        }
        // System.out.println("invalid keyword");
        return false;
    }
```

Keywords

KEYWORDS	
primitive types-	int,
	float,
	bool,
	str,
	char,
	void
Flow control-	switch,
	case,
	continue,
	break,

	default,
	if,
	else,
	instanceof,
	returns,
	for,
	while
Reference Variables-	super,
	this
//For identification-	ID
Output-	print
Input-	input
Boolean	true
	false
<u>OOP CONCEPTS:</u>	
Class,method,variable modifiers-	abstract,
	Class,
	inherits,
	implements,
	interface,
	new,
	static,
	main,
	func,
	call.
Access modifiers-	public,
	private,

	protected.
Packaging and API-	import

PUNCTUATIONS	
{	Open Curly Brackets
}	Closed Curly Brackets
[	Open Square Brackets
]	Closed Square Brackets
(	Open Round Brackets
)	Closed Round Brackets
'	Quotes
;	semicolon
,	coma
.	dot

ARITHMETIC OPERATORS	arithmetic operators
+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Modulus
^	power

Relational Operators	relational operators
=	Assignment
==	Equivalence
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
!=	not equal

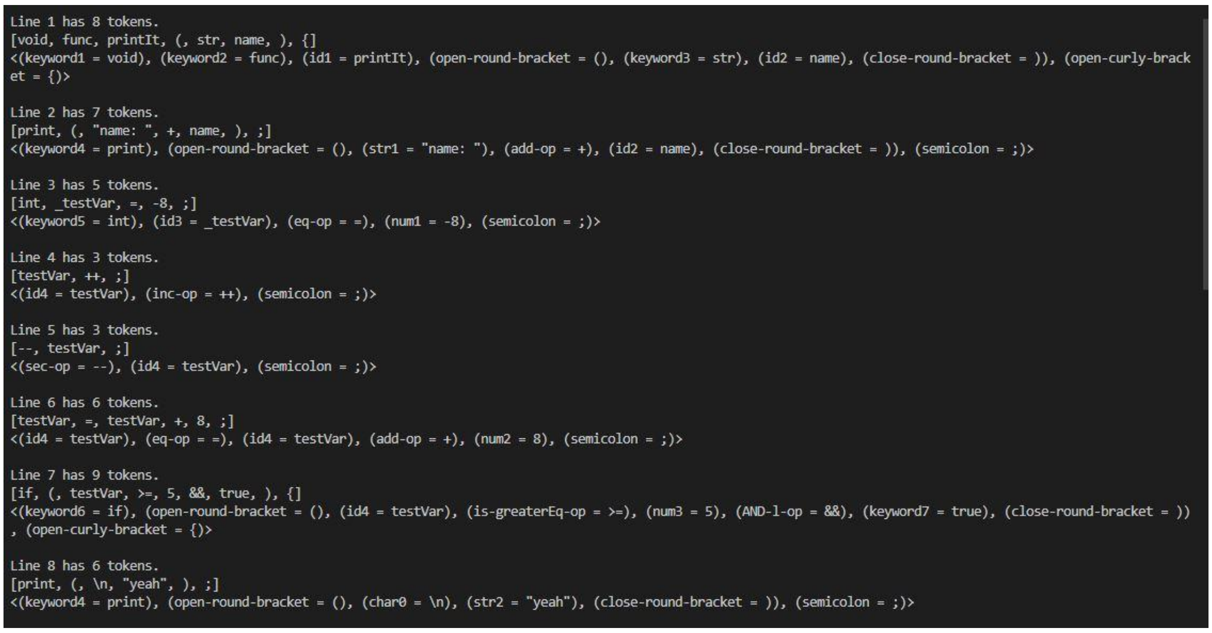
Logical Operators	
&&	Logical AND
||	Logical OR
!!	Logical NOT

Bitwise Operators	
&	Bitwise AND
|	Bitwise OR
#	XOR

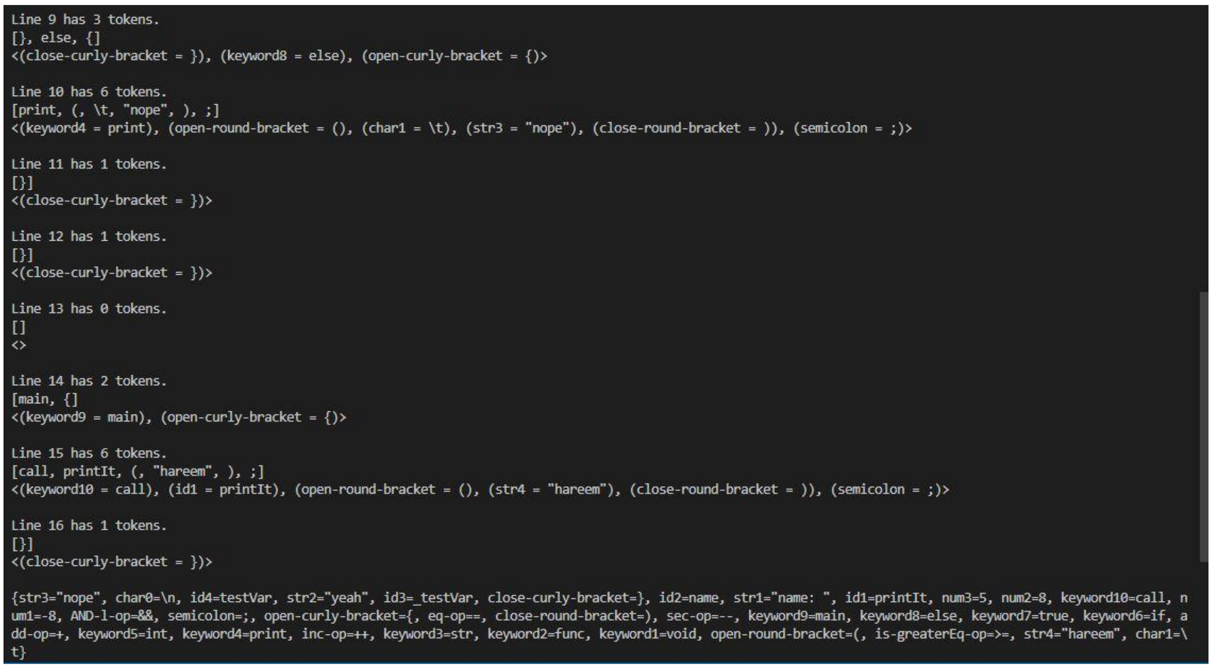
~	Bitwise NOT
<<	Bitwise left shift
>>	Bitwise right shift

Assignment Operators	
+=	Addition , Assignment
- =	Subtraction ,Assignment
/=	Multiplication ,Assignment
***=**	Division , Assignment
%=	Modulus, Assignment
++	Increment
- -	Decrement

Screenshots



```
Line 1 has 8 tokens.
[void, func, printIt, (, str, name, ), {]
<(keyword1 = void), (keyword2 = func), (id1 = printIt), (open-round-bracket = (), (keyword3 = str), (id2 = name), (close-round-bracket = )), (open-curly-brack
et = {)>

Line 2 has 7 tokens.
[print, (, "name: ", +, name, ), ;]
<(keyword4 = print), (open-round-bracket = (), (str1 = "name: "), (add-op = +), (id2 = name), (close-round-bracket = )), (semicolon = ;)>

Line 3 has 5 tokens.
[int, _testVar, =, -8, ;]
<(keyword5 = int), (id3 = _testVar), (eq-op = =), (num1 = -8), (semicolon = ;)>

Line 4 has 3 tokens.
[testVar, ++, ;]
<(id4 = testVar), (inc-op = ++), (semicolon = ;)>

Line 5 has 3 tokens.
[--, testVar, ;]
<(sec-op = --), (id4 = testVar), (semicolon = ;)>

Line 6 has 6 tokens.
[testVar, =, testVar, +, 8, ;]
<(id4 = testVar), (eq-op = =), (id4 = testVar), (add-op = +), (num2 = 8), (semicolon = ;)>

Line 7 has 9 tokens.
[if, (, testVar, >=, 5, &&, true, ), {]
<(keyword6 = if), (open-round-bracket = (), (id4 = testVar), (is-greaterEq-op = >=), (num3 = 5), (AND-l-op = &&), (keyword7 = true), (close-round-bracket = ))
, (open-curly-bracket = {)>

Line 8 has 6 tokens.
[print, (, \n, "yeah", ), ;]
<(keyword4 = print), (open-round-bracket = (), (char0 = \n), (str2 = "yeah"), (close-round-bracket = )), (semicolon = ;)>
```



```
Line 9 has 3 tokens.
[}, else, {]
<(close-curly-bracket = }), (keyword8 = else), (open-curly-bracket = {)>

Line 10 has 6 tokens.
[print, (, \t, "nope", ), ;]
<(keyword4 = print), (open-round-bracket = (), (char1 = \t), (str3 = "nope"), (close-round-bracket = )), (semicolon = ;)>

Line 11 has 1 tokens.
[}]
<(close-curly-bracket = })>

Line 12 has 1 tokens.
[}]
<(close-curly-bracket = })>

Line 13 has 0 tokens.
[]
<>

Line 14 has 2 tokens.
[main, {]
<(keyword9 = main), (open-curly-bracket = {)>

Line 15 has 6 tokens.
[call, printIt, (, "hareem", ), ;]
<(keyword10 = call), (id1 = printIt), (open-round-bracket = (), (str4 = "hareem"), (close-round-bracket = )), (semicolon = ;)>

Line 16 has 1 tokens.
[}]
<(close-curly-bracket = })>

{str3="nope", char0=\n, id4=testVar, str2="yeah", id3=_testVar, close-curly-bracket=}, id2=name, str1="name: ", id1=printIt, num3=5, num2=8, keyword10=call, n
um1=-8, AND-l-op=&&, semicolon=;, open-curly-bracket={, eq-op==, close-round-bracket=), sec-op=--, keyword9=main, keyword8=else, keyword7=true, keyword6=if, a
dd-op=+, keyword5=int, keyword4=print, inc-op=++, keyword3=str, keyword2=func, keyword1=void, open-round-bracket=(, is-greaterEq-op=>=, str4="hareem", char1=\
t}
```