IRA A. FULTON SCHOOLS OF engineering
ARIZONA STATE UNIVERSITY

BITSY Documentation

Project 2

SER-502

By

Sweta Singhal (ASU ID: 1209339946)
Tamalika Mukherjee (ASU ID: 1207688959)
Yogesh Pandey (ASU ID: 1209374189)

Acknowledgements

We cannot express enough thanks to our professor Dr. John Femiani and his teaching assistant Mandar Patwardhan for their continued support and encouragement. We offer our sincere appreciation for the learning opportunities provided by them.

Table of Contents

2. Overview

2.1 Design Goals

BITSY is a programming language that gets compiled to a stack based low-level intermediate code from the high level source code. This intermediate code is executed in the run-time environment.

2.2 Tools

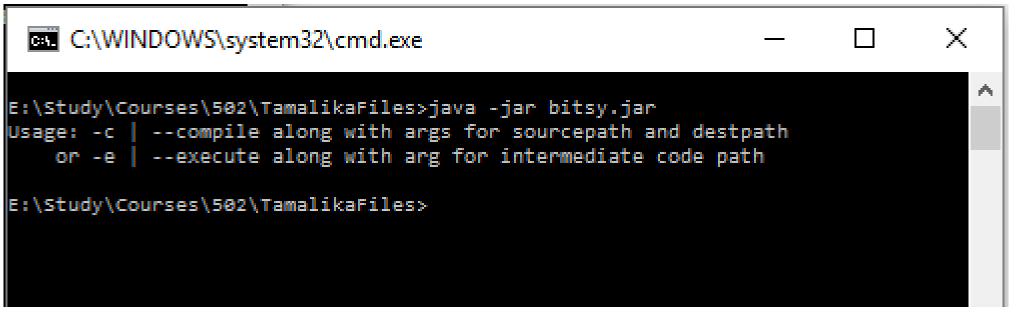
- *ANTLR 4*
    - Parser generation
    - Tokenization
    - Translation
- Virtual Machine
    - Stack based model
    - Runtime environment uses *JRE(Java Runtime Environment)*
- Compiler written in Java.

3. Installation

Download *bitsy.jar* to get started with BITSY. It is that simple.

To ensure proper installation, before compiling and running programs you can do the following:

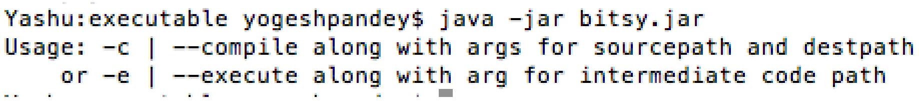
- *Windows Users*:
    - Open Command Prompt from the file location of bitsy.jar
    - Type in the following command: *java -jar bitsy.jar*
    - You are good to go if you get the Usage instructions on screen [Refer to *Figure 3.1*].



```
E:\Study\Courses\502\TamalikaFiles>java -jar bitsy.jar
Usage: -c | --compile along with args for sourcepath and destpath
    or -e | --execute along with arg for intermediate code path

E:\Study\Courses\502\TamalikaFiles>
```

Figure 3.1

- *Mac Users*
    - Open the Terminal from the file location of bitsy.jar
    - Type in the following command: *java -jar bitsy.jar*
    - You are good to go if you get the Usage instructions on screen [Refer to Figure 3.2]



```
Yashu:executable yogeshpandey$ java -jar bitsy.jar
Usage: -c | --compile along with args for sourcepath and destpath
    or -e | --execute along with arg for intermediate code path
```

Figure 3.2

4. Getting Started

4.1 Windows Users

4.1.1 Compilation

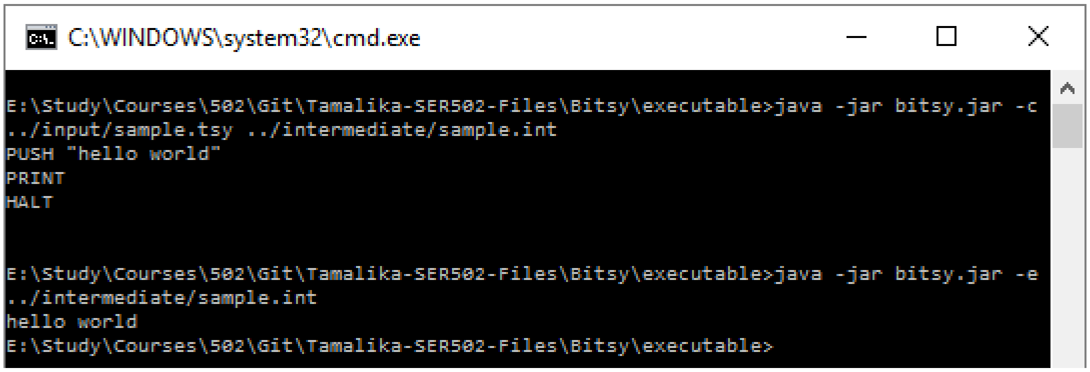
After following the installation steps, type in the following command for compiling a program:

java -jar bitsy.jar -c sourcePath destinationPath

Refer to *Figure 4.1* for easy understanding.

Few important points:

- Intermediate code gets generated and displayed after compilation.
- Intermediate code get generated in the destination path.
- Mentioning the destination path is optional.
- In the absence of the destination path, intermediate code is generated in the source file path mentioned.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c
../input/sample.tsy ../intermediate/sample.int
PUSH "hello world"
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e
../intermediate/sample.int
hello world
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 4.1

4.1.2 Execution

Type in the following command for execution:

```
java -jar bitsy.jar -e intermediateCodePath
```

Refer to *Figure 4.1* for easy understanding. The destination path or the path for intermediate code generation specified in the compilation step needs to be passed in as argument.

4.1.2 Mac Users

4.2.1 Compilation

After following the installation steps, type in the following command for compiling a program:

```
java -jar bitsy.jar -c sourcePath destinationPath
```

Refer to *Figure 4.2* for easy understanding.

Few important points:

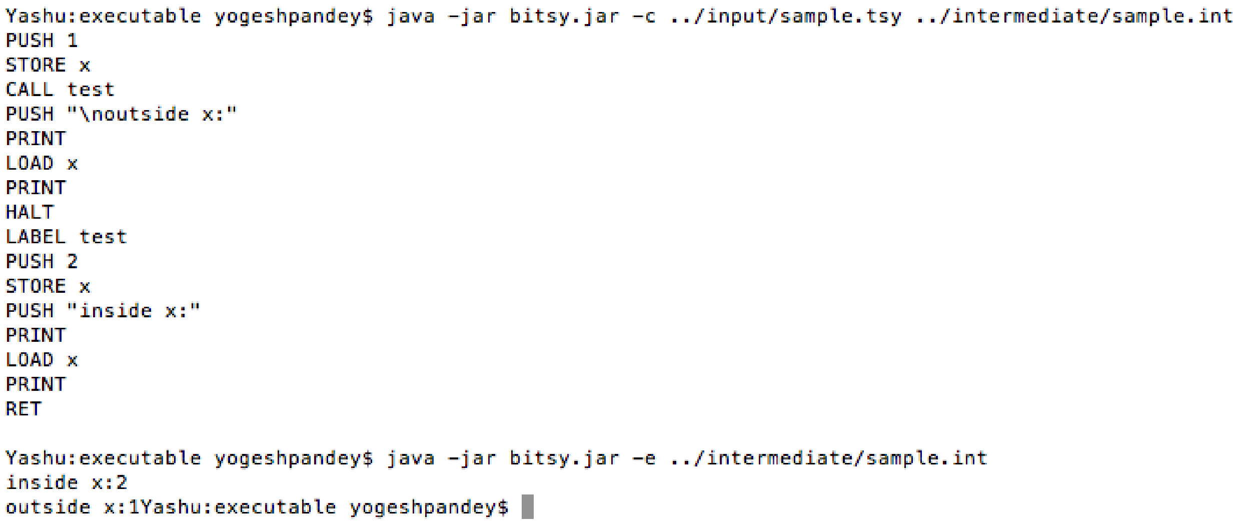
- Intermediate code gets generated and displayed after compilation.
- Intermediate code get generated in the destination path.
- Mentioning the destination path is optional.
- In the absence of the destination path, intermediate code is generated in the source file path mentioned.

4.1.2 Execution

Type in the following command for execution:

```
java -jar bitsy.jar -e intermediateCodePath
```

Refer to *Figure 4.2* for easy understanding. The destination path or the path for intermediate code generation specified in the compilation step needs to be passed in as argument.



```
Yashu:executable yogeshpandey$ java -jar bitsy.jar -c ../input/sample.tsy ../intermediate/sample.int
PUSH 1
STORE x
CALL test
PUSH "\noutside x:"
PRINT
LOAD x
PRINT
HALT
LABEL test
PUSH 2
STORE x
PUSH "inside x:"
PRINT
LOAD x
PRINT
RET

Yashu:executable yogeshpandey$ java -jar bitsy.jar -e ../intermediate/sample.int
inside x:2
outside x:1Yashu:executable yogeshpandey$ 
```

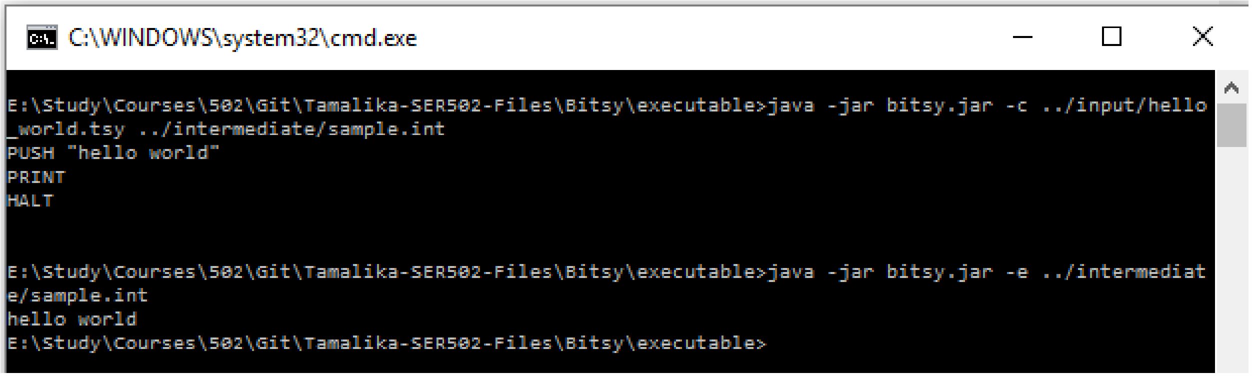
Figure 4.2

4.3 The Hello World Program

This is how the *Hello World* program is written in BITSY.

```
//This is hello world program

print("hello world");
```

Figure 4.3 shows the output.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/hello
_world.tsy ../intermediate/sample.int
PUSH "hello world"
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediat
e/sample.int
hello world
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 4.3

5. Operators

Data types are usually associated with a set of *operators* [1] implicitly. BITSY supports various binary and unary arithmetic and relational operators.

In this section, we look at BITSY operators in a tabular format.

5.1 Binary operators BITSY supports

- *Assignment Operators:*

Table 5.1.1 lists the binary assignment operators BITSY supports in the order of precedence.

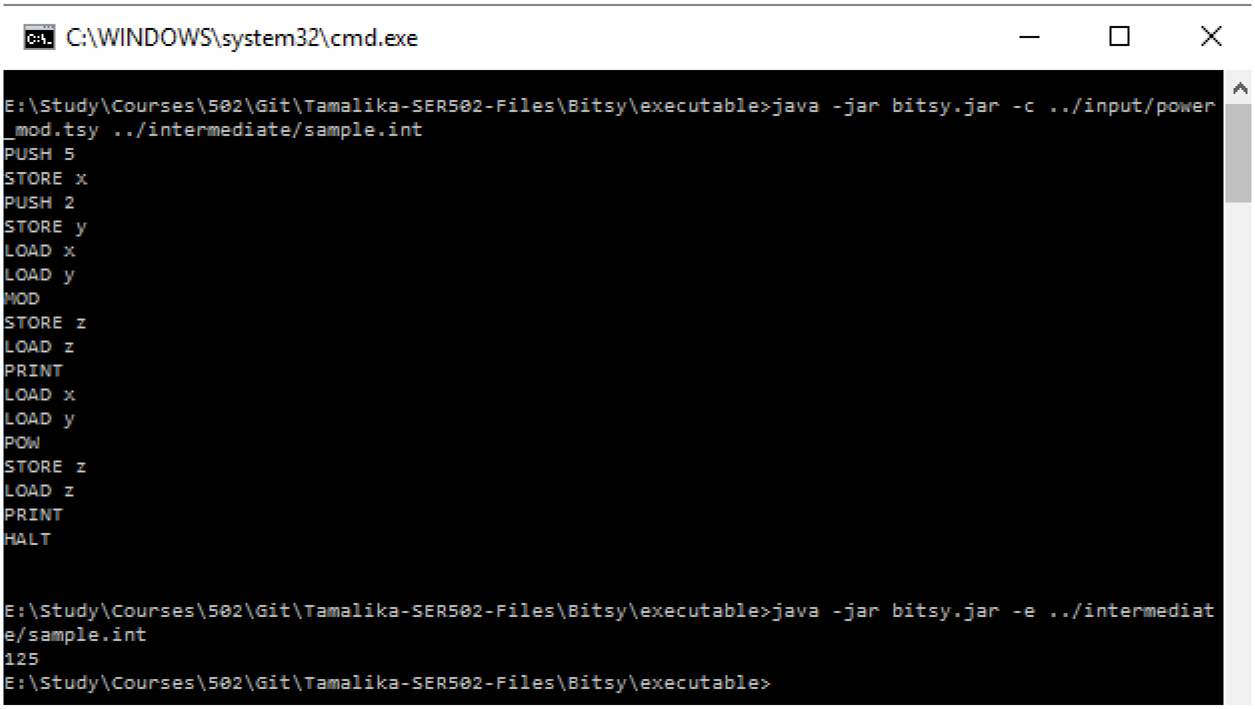
Table 5.1.1: Lists assignment operators BITSY supports

Operator	Name	Operands	Calculates
^	Power	*x^y*	Value of *x* when raised to the power of *y*
%	Modulus	*x%y*	Remainder when *x* is divided by *y*
/	Divide	*x/y*	Value of *x* when divided by *y*
*	Multiply	*x*y*	Value of *x* when multiplied with *y*
-	Subtract	*x-y*	Value of *y* when subtracted from *x*
+	Add	*x+y*	Value of *x* and *y* added together

The following code shows how power and modulus operators are used in BITSY:

```
int x=5;
int y=2;
int z= x%y;
print(z);
z = x^y;
print(z);
```

Figure 5.1.1 shows the output along with the generated intermediate code.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/power
_mod.tsy ../intermediate/sample.int
PUSH 5
STORE x
PUSH 2
STORE y
LOAD x
LOAD y
MOD
STORE z
LOAD z
PRINT
LOAD x
LOAD y
POW
STORE z
LOAD z
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediat
e/sample.int
125
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 5.1.1

- *Relational Operators:*

Table 5.1.2 lists the binary relational operators BITSY supports in the order of precedence.

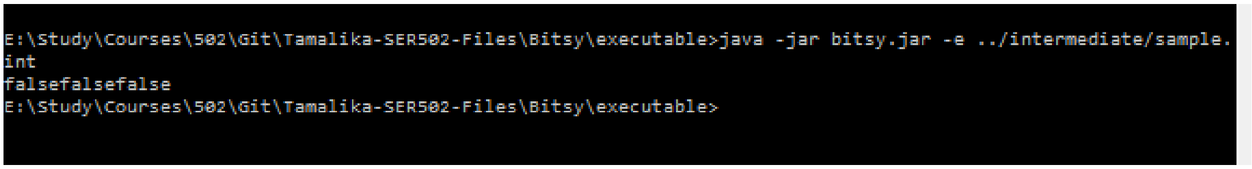
Table 5.1.2: Lists the relational operators BITSY supports

Operator	Name	Operands	Evaluates to true if
<	Less than	*a<b*	*a* less than *b*
>	Greater than	*a>b*	*a* greater than *b*
<=	Less than equals	*a<= b*	*a* less than or equal to *b*
>=	Greater than equals	*a>=b*	*a* greater than or equal to *b*
==	Is equal	*a==b*	*a* equals to *b*
!=	Not equal	*a!=b*	*a* is not equal *b*
&&	Logical AND	*a && b*	Expression *a* and *b* are both true
||	Logical OR	*a || b*	Either of expression *a* or *b* are true

The following code shows how relational operators are used in BITSY

```
int x;
x = false && true;
print(x);

int y;
y = false || false;
print(y);

int z;
z = 6 <= 8 && 5 > 7;
print(z);
```

Figure 5.1.2 shows the output.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/sample.
int
falsefalsefalse
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 5.1.2

5.2 Unary Operators BITSY supports

Table 5.2.1 lists the unary operators BITSY supports in the order of precedence.

Table 5.2.1: Lists unary operators supported by BITSY

Operator	**Name**	**Operands**	**Returns**
++	Postfix Increment	*a*++	Incremented value of *a*
--	Postfix Decrement	*a*--	Decremented value of *a*
+	Positive	+*a*	Value of *a*
-	Negative	-*a*	Negative of *a*
++	Prefix Increment	++*a*	Incremented value of *a*
--	Prefix Decrement	--*a*	Decremented value of *a*

Notable features of Unary Operators in BITSY:

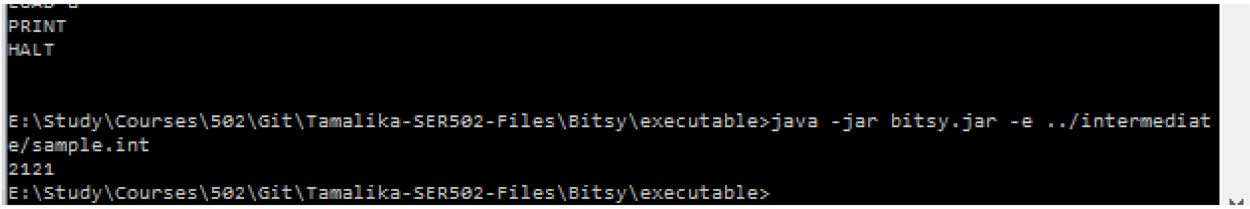
- Prefix and Postfix increment and decrement operators on being applied to a variable update the value of the variable upon increment or decrement.

- However, prefix and postfix operators on being applied to numbers just increment or decrement the values.
- Unary operators have greater preference over binary operators.

The following shows an example code to evaluate prefix and postfix expressions:

```
int a;
a = 1;
a++;
print(a);
a--;
print(a);
++a;
print(a);
--a;
print(a);
```

The following output is displayed. Refer to *Figure 5.2*.



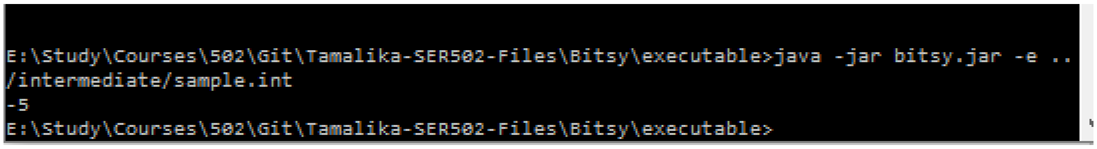
```
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediat
e/sample.int
2121
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 5.2

The following example shows how BITSY handles negative unary operations.

```
int a;
a = -2 -3;
print(a);
```

The output is shown in *Figure 5.3.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ..
/intermediate/sample.int
-5
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

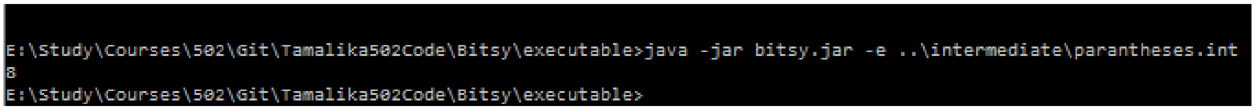
Figure 5.3

5.3 Use of parenthesis

Parenthesis or ‘(‘ and ‘)’ have highest precedence in expressions. An example program to illustrate their use is as shown below.

```
int x;
x = 2 * 4 / (2 - 1);
print(x);
```

Figure 5.3.1 shows the output.



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\parantheses.int
8
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 5.3.1

6. Data Types

A *data type*[1] can be defined as a collection of values, bundled with a collection of operations on those values having certain properties.

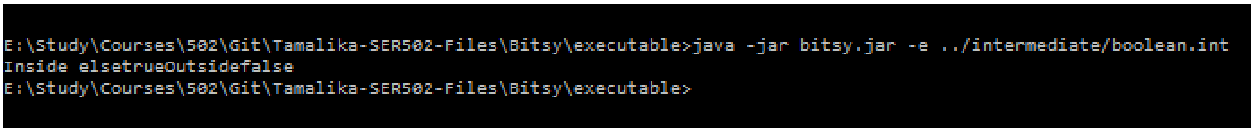
6.1 Types BITSY supports

- *int:*
  - ○ Supports signed numeric integer types such as { *-1, 1, -3, 0, 6* } etc.
- *bool:*
  - ○ Supports boolean values *true* and *false*.

Let us consider an example to illustrate the use of boolean statements.

```
bool x = false;
int y;
y = 5;
if ( x ){
        print("Inside if");
        print(y);
}else{
        print("Inside else");
        x = true || x;
        print(x);
        x = x && false;
}

print("Outside");
print(x);
```

The output is shown in *Figure 6.1.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/boolean.int
Inside elsetrueOutsidefalse
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 6.1

6.2 Features of BITSY's Type System

- Statically typed
- Strongly typed Language

7. Identifiers

An ***identifier*** is a sequence of one or more characters. In BITSY, the first character has to be an upper or lower case letter. Consecutive characters in the identifier are optional and could comprise of alphanumeric characters.

8. Variables and Constants

8.1 Variables

Variables *[1]* are used to name locations which store data values of a certain type. The type is dictated by the kind of data value stored. The stored value of variables can change during program execution.

BITSY supports variable **declarations and assignments operations**. We would see these in more details in *Section 10.1.*

8.2 Constants

Constants are entities whose value remain unchanged throughout program run. Examples include numbers such as 1, 2, 0 etc.

BITSY supports use of constants to assign to variables. We would look at examples etc. in more details in *Section 10.1*.

Consider the following code statement.

```
int a = 9;
```

In the above example, *a* is the name of the variable, int is the data type and 9 is a constant.

9. Expressions

An ***expression*** is a construct comprising of variables and/or operands, constants and operators which evaluate to a single value.

BITSY supports expression evaluation in the order of precedence of operators as defined in *Section 5*.

10. Statements

Valid ***statements*** in BITSY end in a semicolon ';'. In other words, ';' indicates the end of a statement in BITSY.

10.1. Variable Declaration and Assignment Statements

BITSY supports variable declaration and assignment statements.

In BITSY, variable declaration has the following signature

dataType   variableName   ;

The following code should demonstrate variable declaration.

```
int a; /*variable declaration statement*/
a = 5; /*assignment statement */
```

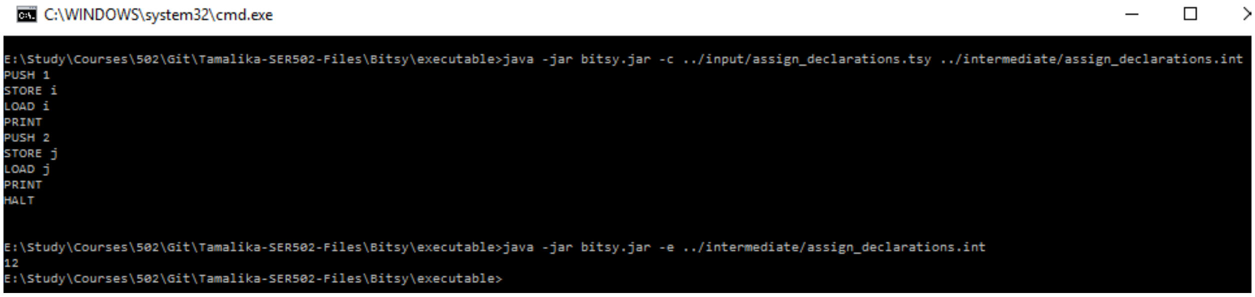
The assignment and the declaration statements could also be clubbed together as one statement. Thus BITSY allows **assignment and declaration at the same step**. The syntax would be :

dataType   variableName   =   Constant ;

The following code demonstrates that.

```
int i = 1;
print(i);
int j;
j = 2;
print(j);
```

Figure 10.1.1 shows the intermediate code and the output.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/assign_declarations.tsy ../intermediate/assign_declarations.int
PUSH 1
STORE i
LOAD i
PRINT
PUSH 2
STORE j
LOAD j
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/assign_declarations.int
12
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 10.1.1

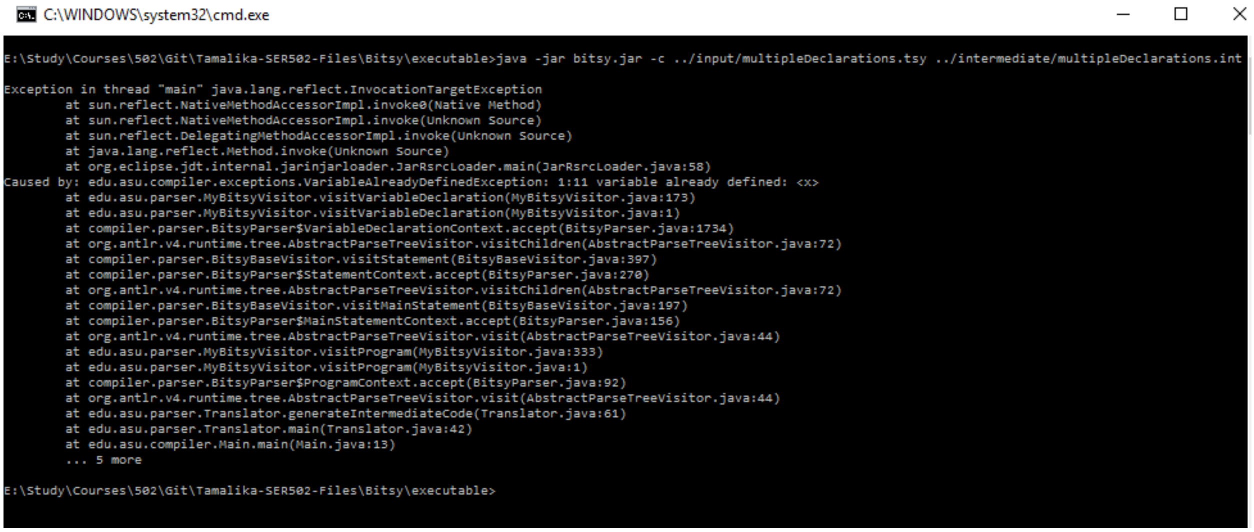
BITSY supports the following:

- Declare a variable
- Assign values to a variable
- Modify values of variables
- Declare a variable and assign value at the same step.

BITSY however, does not allow multiple declarations of the same variable. Such a code would throw an error! Consider the following example.

```
int x; int x;
```

Compiling the above program throws an Exception. Refer to *Figure 10.1.2.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/multipleDeclarations.tsy ../intermediate/multipleDeclarations.int

Exception in thread "main" java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoader.java:58)
Caused by: edu.asu.compiler.exceptions.VariableAlreadyDefinedException: 1:11 variable already defined: <x>
        at edu.asu.parser.MyBitsyVisitor.visitVariableDeclaration(MyBitsyVisitor.java:173)
        at edu.asu.parser.MyBitsyVisitor.visitVariableDeclaration(MyBitsyVisitor.java:1)
        at compiler.parser.BitsyParser$VariableDeclarationContext.accept(BitsyParser.java:1734)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visitChildren(AbstractParseTreeVisitor.java:72)
        at compiler.parser.BitsyBaseVisitor.visitStatement(BitsyBaseVisitor.java:397)
        at compiler.parser.BitsyParser$StatementContext.accept(BitsyParser.java:270)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visitChildren(AbstractParseTreeVisitor.java:72)
        at compiler.parser.BitsyBaseVisitor.visitMainStatement(BitsyBaseVisitor.java:197)
        at compiler.parser.BitsyParser$MainStatementContext.accept(BitsyParser.java:156)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:44)
        at edu.asu.parser.MyBitsyVisitor.visitProgram(MyBitsyVisitor.java:333)
        at edu.asu.parser.MyBitsyVisitor.visitProgram(MyBitsyVisitor.java:1)
        at compiler.parser.BitsyParser$ProgramContext.accept(BitsyParser.java:92)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:44)
        at edu.asu.parser.Translator.generateIntermediateCode(Translator.java:61)
        at edu.asu.parser.Translator.main(Translator.java:42)
        at edu.asu.compiler.Main.main(Main.java:13)
        ... 5 more

E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

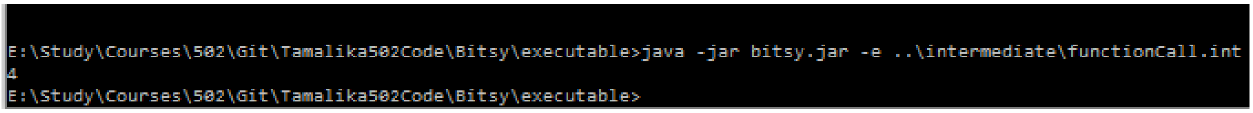
Figure 10.1.2

10.2 Function Declaration and Function Call Statements

Let us look at example codes which illustrate how to work with functions in BITSY.

```
int x = 3;
int y = 5;
print(avg(x, y));
func avg(int a, int b) {
        return (a + b)/2;
}
```

The output is displayed in *Figure 10.2.1.* Refer to *Section 14* for further details.



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\functionCall.int
4
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 10.2.1

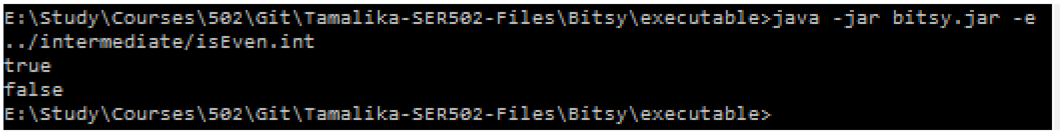
10.3 Comments

Comments [1] are used to document programs to make them readable,understandable and maintainable. Comments are used for self documentation thus. Hence, during program execution comments are skipped from getting executed.

BITSY allows comments. Let us consider the following example.

```
/** Check if number is Even or Odd
 * @param x: input number
 * @return: false if Odd
                       true if Even
 */
func isEven(int x) {
        return x%2==0;
}
print(isEven(4));// should print true
print("\n");
print(isEven(7));// should print false
```

Figure 10.3 shows the output.



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e
../intermediate/isEven.int
true
false
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 10.3

As illustrated in the above code example, the syntax of using comments is as follows:

```
// Statements
```

OR

```
/*
       Statements
       Statements

*/
```

11. Scope

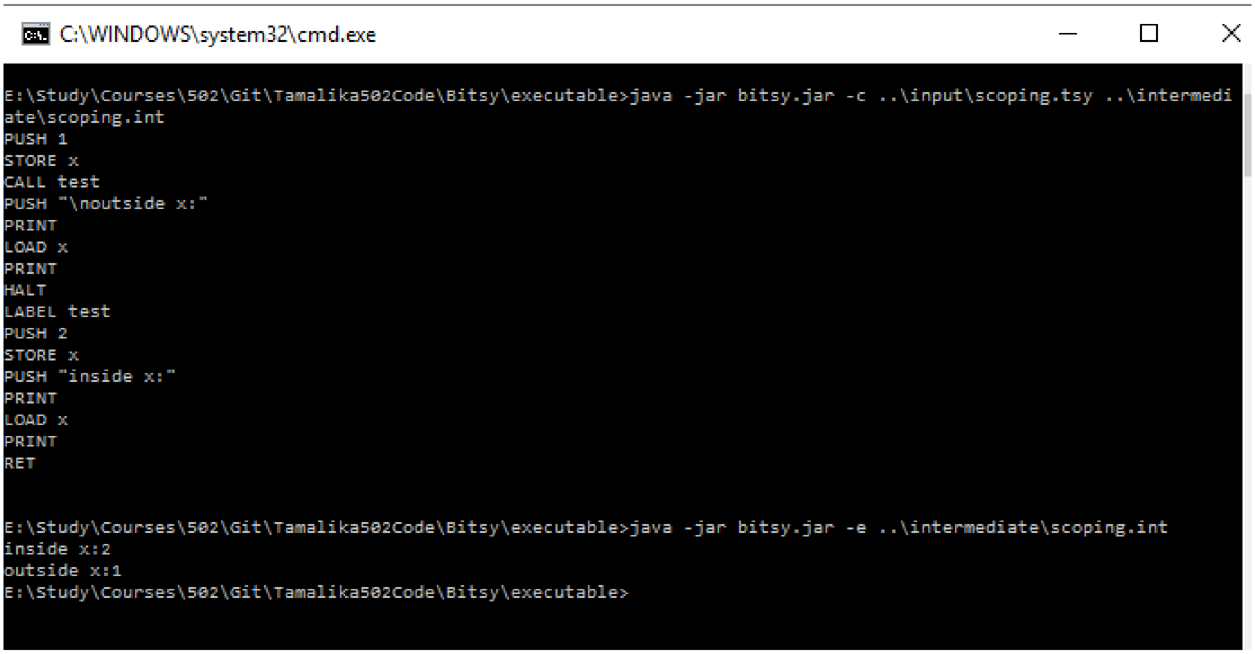
BITSY allows function level static ***scoping***. Blocks are indicated using curly braces as shown in the following example.

```
/*start of block*/
{
       int a = 1;
       print (a);
}
/*end of block*/

1 /*output*/
```

Local and ***non-local or global*** variables are defined in BITSY. To elaborate this, let us consider the following example.

```
int x = 1;
func test() {
  x = 2;
  print("inside x:");
  print(x);
  return;
}
test();
print("\noutside x:");
print(x);
```

The concept of visibility is well established in this example. Refer to *Figure 11.1* for the intermediate code and the output.

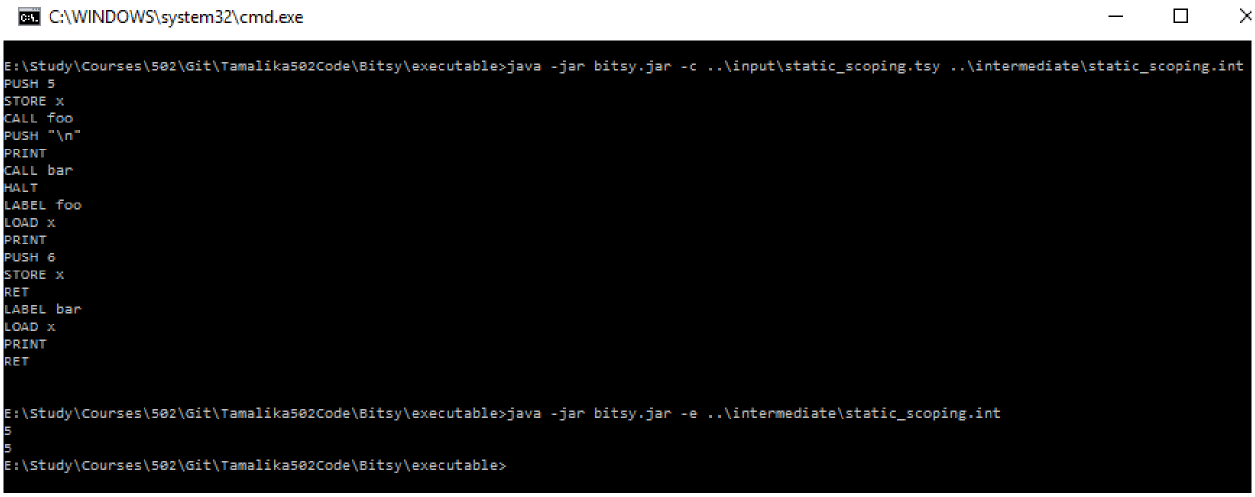


```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -c ..\input\scoping.tsy ..\intermedi
ate\scoping.int
PUSH 1
STORE x
CALL test
PUSH "\noutside x:"
PRINT
LOAD x
PRINT
HALT
LABEL test
PUSH 2
STORE x
PUSH "inside x:"
PRINT
LOAD x
PRINT
RET


E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\scoping.int
inside x:2
outside x:1
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 11.1

The next example illustrates the concept of static scoping. Refer to *Figure 11.2* for the intermediate code and the output.

```
int x = 5;
foo();
print("\n");
bar();
func foo() {
        print(x);
        x = 6;
        return;
}
func bar() {
        print(x);
        return;
}
```



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -c ..\input\static_scoping.tsy ..\intermediate\static_scoping.int
PUSH 5
STORE x
CALL foo
PUSH "\n"
PRINT
CALL bar
HALT
LABEL foo
LOAD x
PRINT
PUSH 6
STORE x
RET
LABEL bar
LOAD x
PRINT
RET


E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\static_scoping.int
5
5
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 11.2

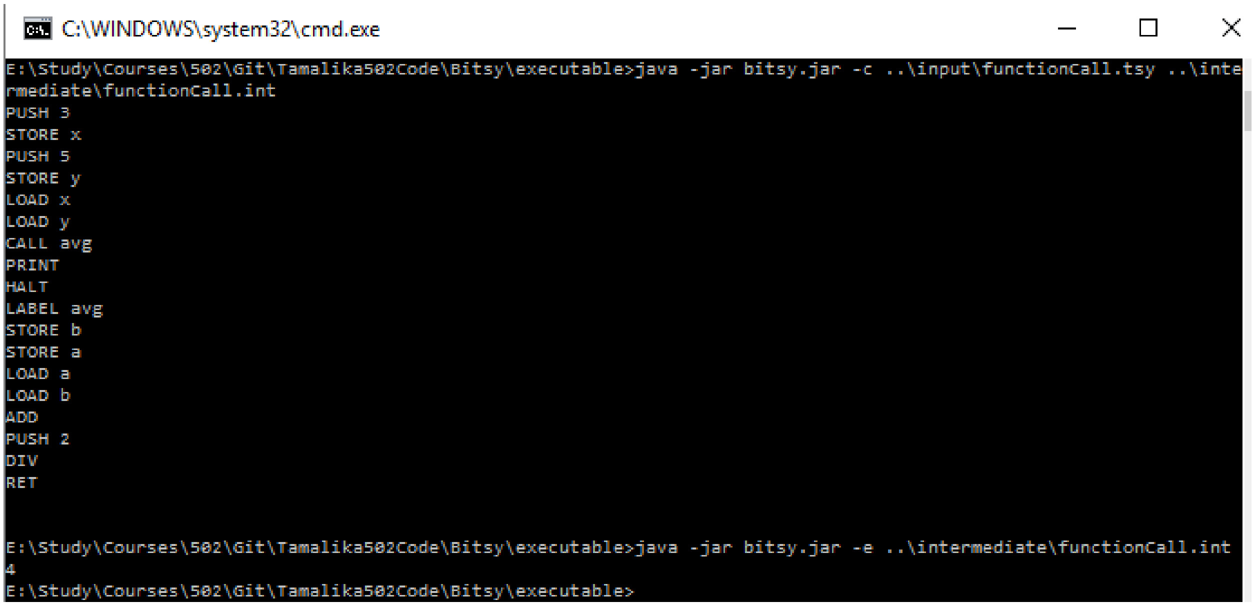
12. Functions

Functions are used to perform an operation. Functions in a program are modules which collectively solve the computational problem at hand. BITSY supports the use of functions.

Let us look at a simple example to demonstrate the use of functions in BITSY.

```
int x = 3;
int y = 5;
print(avg(x, y));
func avg(int a, int b) {
        return (a + b)/2;
}
```

The intermediate code generated along with the output is shown in *Figure 12.1*.

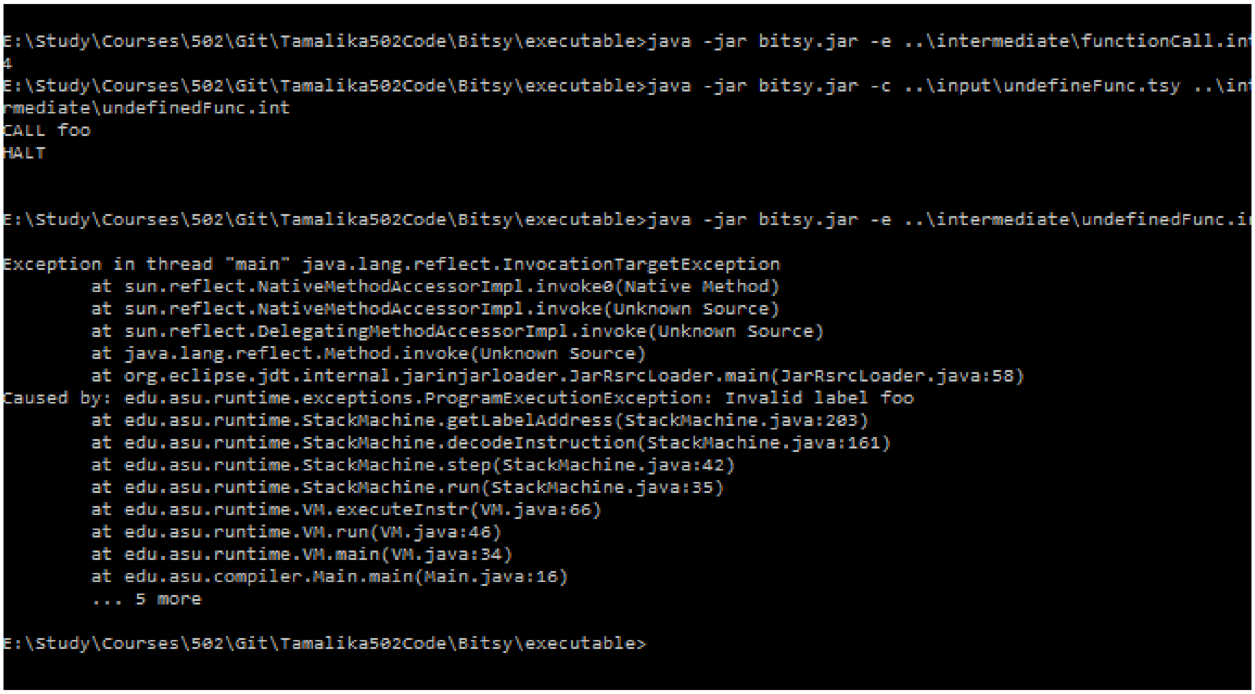


```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -c ..\input\functionCall.tsy ..\inte
rmediate\functionCall.int
PUSH 3
STORE x
PUSH 5
STORE y
LOAD x
LOAD y
CALL avg
PRINT
HALT
LABEL avg
STORE b
STORE a
LOAD a
LOAD b
ADD
PUSH 2
DIV
RET


E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\functionCall.int
4
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 12.1

Making a call to an undefined function is again illegal in Bitsy. Following example illustrates this fact. Refer to *Figure 12.2*.

```
//Calling undefined function
foo();
```



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\functionCall.in
4
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -c ..\input\undefineFunc.tsy ..\in
rmediate\undefinedFunc.int
CALL foo
HALT


E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\undefinedFunc.i

Exception in thread "main" java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoader.java:58)
Caused by: edu.asu.runtime.exceptions.ProgramExecutionException: Invalid label foo
        at edu.asu.runtime.StackMachine.getLabelAddress(StackMachine.java:203)
        at edu.asu.runtime.StackMachine.decodeInstruction(StackMachine.java:161)
        at edu.asu.runtime.StackMachine.step(StackMachine.java:42)
        at edu.asu.runtime.StackMachine.run(StackMachine.java:35)
        at edu.asu.runtime.VM.executeInstr(VM.java:66)
        at edu.asu.runtime.VM.run(VM.java:46)
        at edu.asu.runtime.VM.main(VM.java:34)
        at edu.asu.compiler.Main.main(Main.java:16)
        ... 5 more

E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

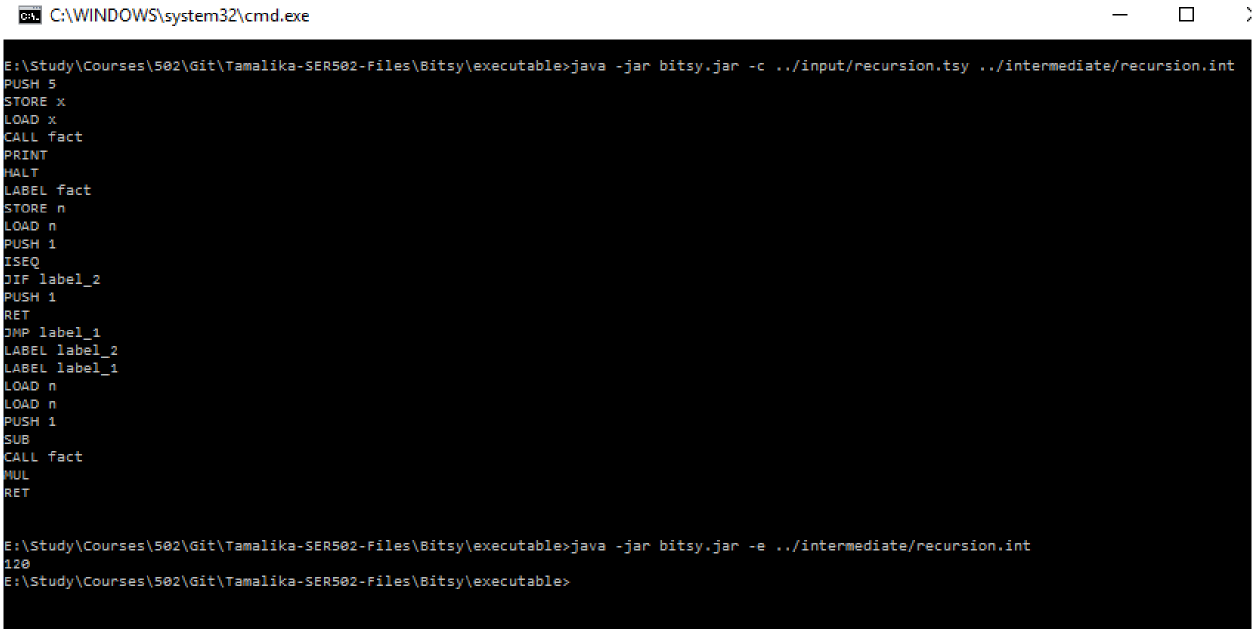
Figure 12.2

13. Recursion

BITSY is able to handle ***recursive functions*** well. Let us look at the factorial example, which is programmed using recursion.

```
int x;
x = 5;
print(fact(x));
func fact(int n) {
        if(n==1) {
                return 1;
        }
        return n * fact(n-1);
}
```

The intermediate code followed by the output is illustrated in *Figure 13.1.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/recursion.tsy ../intermediate/recursion.int
PUSH 5
STORE x
LOAD x
CALL fact
PRINT
HALT
LABEL fact
STORE n
LOAD n
PUSH 1
ISEQ
JIF label_2
PUSH 1
RET
JMP label_1
LABEL label_2
LABEL label_1
LOAD n
LOAD n
PUSH 1
SUB
CALL fact
MUL
RET


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/recursion.int
120
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

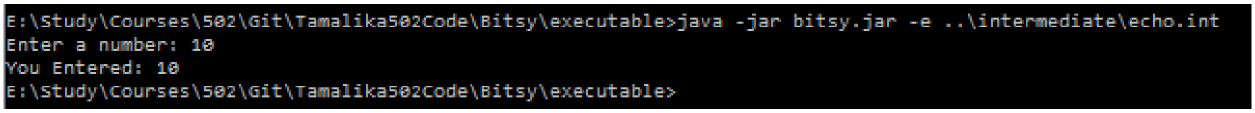
Figure 13.1

14. I/O Operations

BITSY supports Input/Output(or, I/O) operations. In other words, users are able to enter input and view results as output on screen. The following example illustrates BITSY's way of handling I/O operations.

```
print("Enter a number: ");
int x = input();
print("You Entered: ");
print(x);
```

Refer to *Figure 14.1* for the output.



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\echo.int
Enter a number: 10
You Entered: 10
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

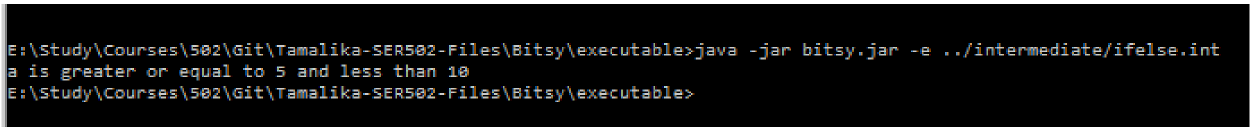
Figure 14.1

15. Conditionals

BITSY supports conditional statements and nested conditionals. The following example would illustrate the use of ***if-else*** statements.

```
int a = 6;
if(a < 5) {
        print("a is less than 5");
} elif(a < 10) {
        print("a is greater or equal to 5 and less than 10");
} else {
        print("a is greater than 10");
```

The output is shown in *Figure 15.1.*



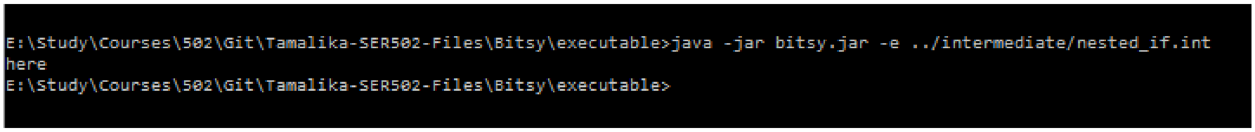
```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/ifelse.int
a is greater or equal to 5 and less than 10
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 15.1

Bitsy also supports ***nested if-else*** statements. Let us see an example.

```
if(2>1) {
        if(3>2) {
                print("here");
        }
} else {
        print("hello");
}
```

The output is displayed in *Figure 15.2.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/nested_if.int
here
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

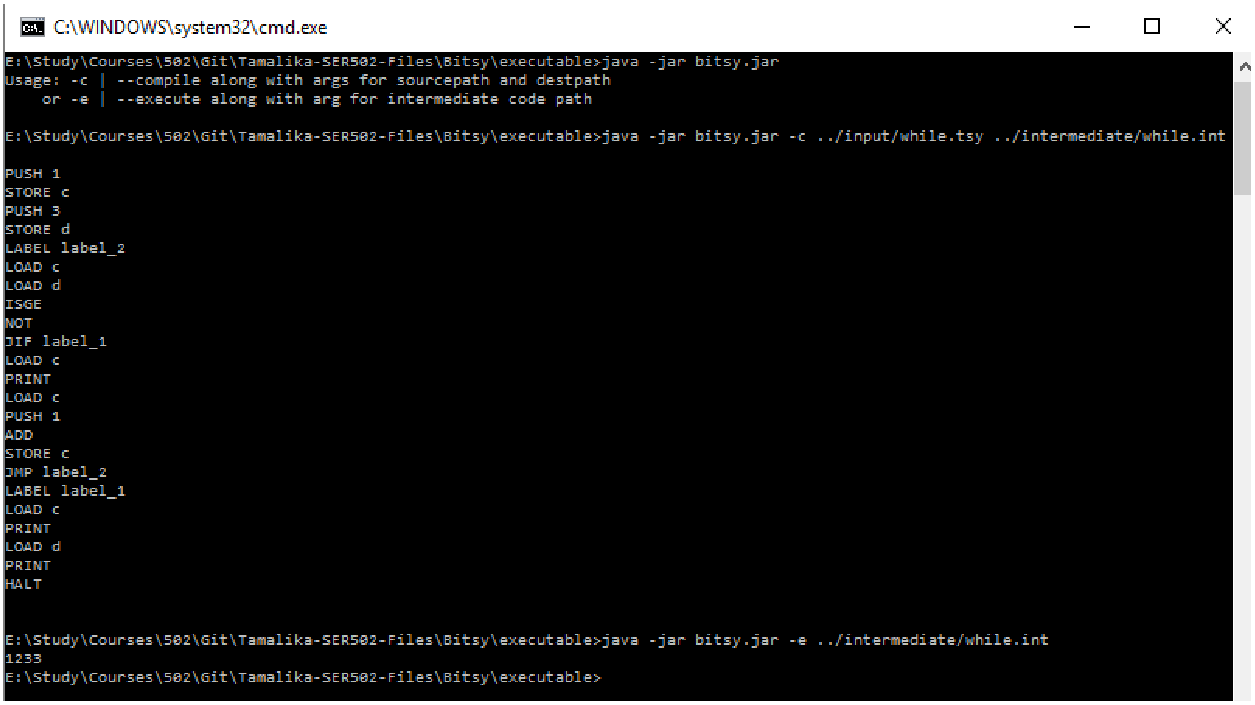
Figure 15.2

16. Loops

An entry controlled loop or iterating construct ***while*** is provided by BITSY. The following code demonstrates the use of *while* loop.

```
int c= 1;
int d = 3;
while(c<d) {
        print(c);
        c++;
}
print(c);
print(d);
```

The generated intermediate code and the output is shown in *Figure 16.1*



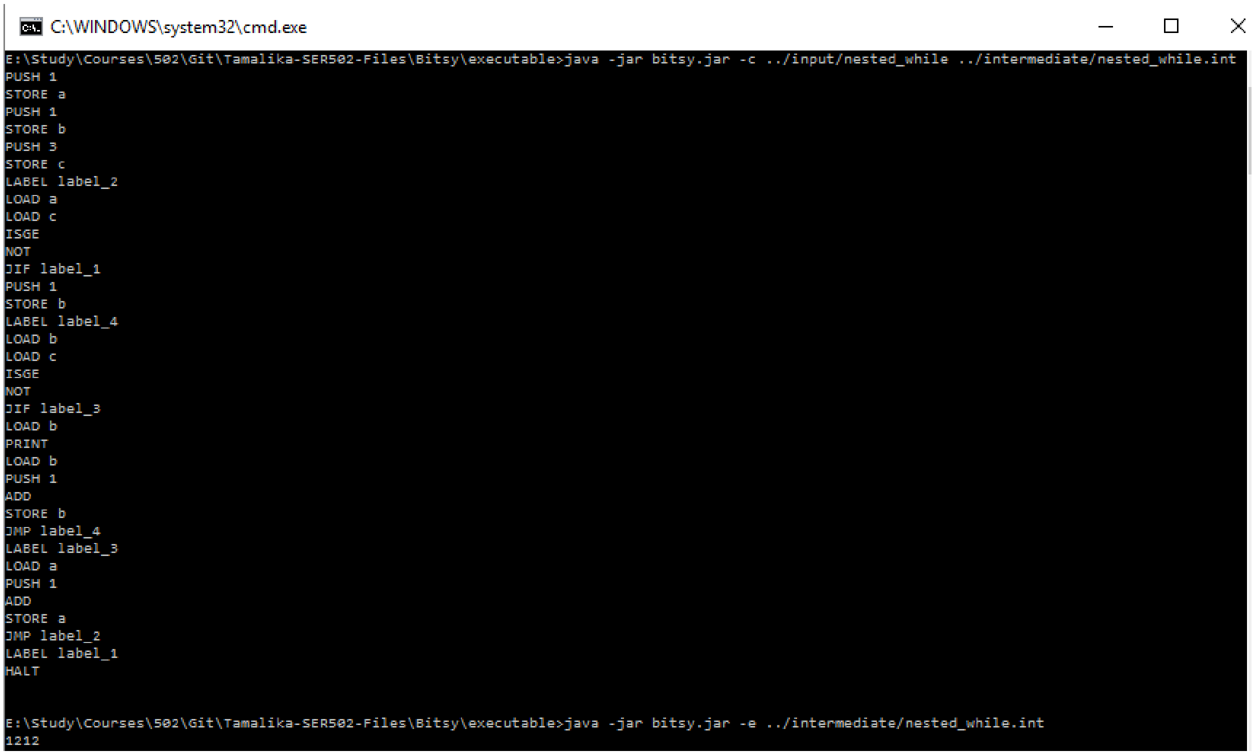
```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar
Usage: -c | --compile along with args for sourcepath and destpath
    or -e | --execute along with arg for intermediate code path

E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/while.tsy ../intermediate/while.int

PUSH 1
STORE c
PUSH 3
STORE d
LABEL label_2
LOAD c
LOAD d
ISGE
NOT
JIF label_1
LOAD c
PRINT
LOAD c
PUSH 1
ADD
STORE c
JMP label_2
LABEL label_1
LOAD c
PRINT
LOAD d
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/while.int
1233
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 16.1

Nested while loops are also supported by BITSY. We show an example program below.

```
int a = 1;
int b = 1;
int c = 3;
while(a < c){
        b=1;
        while(b < c){
                print(b);
                b++;
        }
        a++;
}
```

The generated intermediate code and the output is shown in *Figure 16.2.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/nested_while ../intermediate/nested_while.int
PUSH 1
STORE a
PUSH 1
STORE b
PUSH 3
STORE c
LABEL label_2
LOAD a
LOAD c
ISGE
NOT
JIF label_1
PUSH 1
STORE b
LABEL label_4
LOAD b
LOAD c
ISGE
NOT
JIF label_3
LOAD b
PRINT
LOAD b
PUSH 1
ADD
STORE b
JMP label_4
LABEL label_3
LOAD a
PUSH 1
ADD
STORE a
JMP label_2
LABEL label_1
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/nested_while.int
1212
```

Figure 16.2

17. Data Structures

In addition to the predefined types *int* and *bool,* BITSY supports the ***stack*** *data structure.*

The following program would illustrate the functionalities the *stack* data structure provide in BITSY:

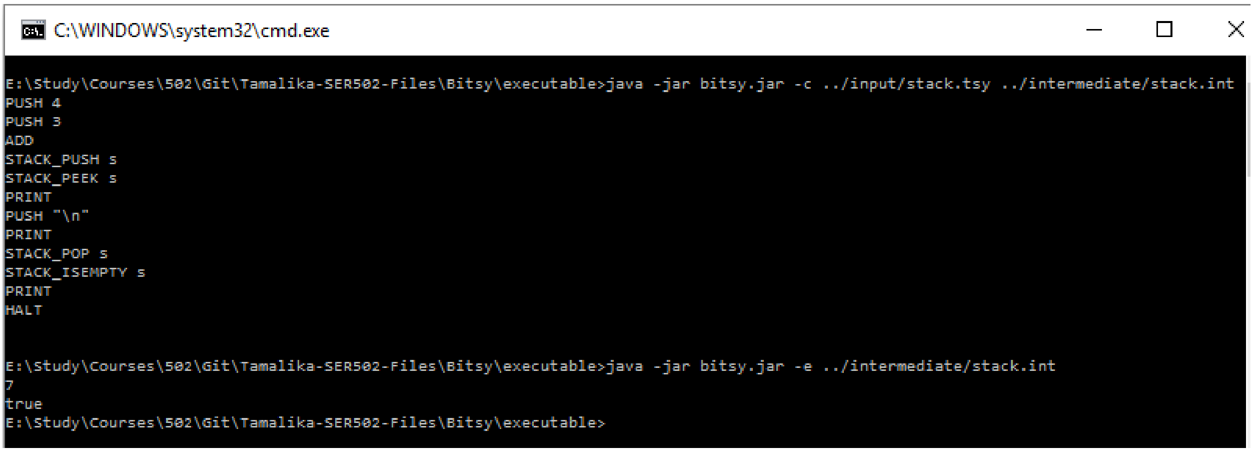
```
stack s;
s.push(4+3);
print(s.peek());
print("\n");
s.pop();
print(s.isEmpty());
```

As you can see from the code above, the following functionalities are provided with the stack data structure:

Table 17.1: Stack Operations

Stack Function	Purpose
push(*a*)	Pushes argument *a* on top of stack
pop()	Pops out top element. Updates top to next element in stack.
peek()	Returns top element
isEmpty()	Checks if stack is empty. Returns true if empty, false otherwise.

The intermediate code and the output is shown in *Figure 17.1.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -c ../input/stack.tsy ../intermediate/stack.int
PUSH 4
PUSH 3
ADD
STACK_PUSH s
STACK_PEEK s
PRINT
PUSH "\n"
PRINT
STACK_POP s
STACK_ISEMPTY s
PRINT
HALT


E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e ../intermediate/stack.int
7
true
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 17.1

18. Sample Programs and Examples

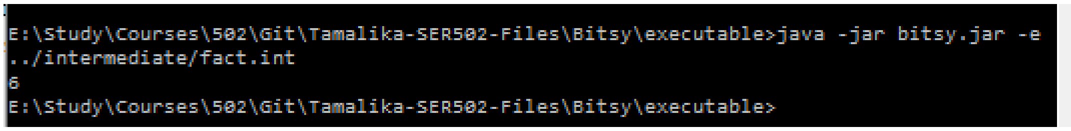
In this section we look at few sample programs that were written in BITSY.

18.1 Factorial Program

The factorial program calculates the factorial of a number entered. Following is the code for the factorial program written in BITSY:

```
func fact(int x) {
        if(x == 1) {
                return x;
        }
        return x * fact(x-1);
}
print(fact(3));
```

The output is shown in *Figure 18.1.*



```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e
../intermediate/fact.int
6
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

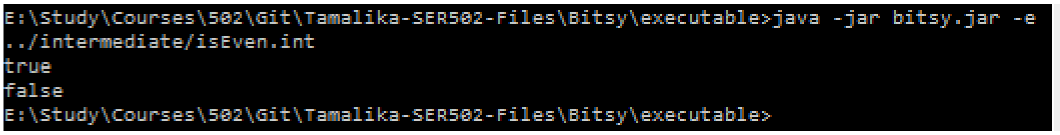
Figure 18.1

18.2 Even or Odd Checker

The following program checks if the number entered is even or odd.

```
/** Check if number is Even or Odd
 * @param x: input number
 * @return: false if Odd
                   true if Even
 */
func isEven(int x) {
        return x%2==0;
}
print(isEven(4));// should print true
print("\n");
print(isEven(7));// should print false
```

Figure 18.2 shows the output.



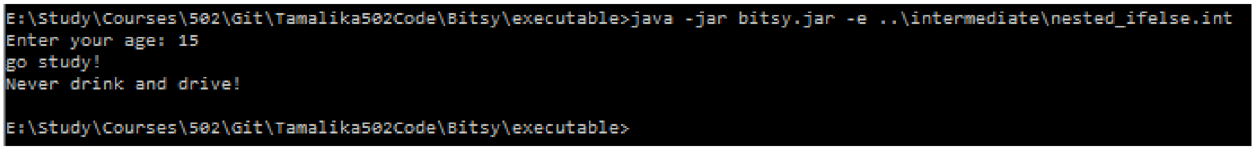
```
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>java -jar bitsy.jar -e
../intermediate/isEven.int
true
false
E:\Study\Courses\502\Git\Tamalika-SER502-Files\Bitsy\executable>
```

Figure 18.2

18.3 Assess how close are you to driving!

```
print("Enter your age: ");
int age = input();
if(age < 18) {
        print("go study!\n");
} elif(age >= 18) {
        if(age > 21) {
                print("can drink and can drive\n");
        } else {
                print("can't drive\n");
        }
}
print("Never drink and drive!\n");
```

The output is shown in *Figure 18.3*.



```
E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>java -jar bitsy.jar -e ..\intermediate\nested_ifelse.int
Enter your age: 15
go study!
Never drink and drive!

E:\Study\Courses\502\Git\Tamalika502Code\Bitsy\executable>
```

Figure 18.3

References

[1] Kenneth C. Louden and Kenneth A. Lambert, “Data Types” in *Programming Languages Principles and Practice,* Third ed. Boston: Cengage Learning, 2011, pp. 1-375.