Report on

"Building a Java Mini Compiler"

Submitted in partial fulfillment of the requirements for ***Sem VI***

Compiler Design Laboratory

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Shreya Sri	**PES1201800132**
Vibha Kurpad	**PES1201800158**
Mahah Sadique	**PES1201801529**

Under the guidance of

Prof. Preet Kanwal
Assistant Professor
PES University, Bengaluru

January – May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

1. INTRODUCTION

The language we have chosen is the Java programming language. We have implemented the front end of the compiler for Java using Flex and Bison and have handled the following constructs.

- If construct
- If-Else construct
- For Loop

The input to the project are .java class files, and the output is the final optimised intermediate code. The frontend of the compiler including Symbol table generation, Abstract Syntax tree construction, Intermediate Code generation and Code Optimization was implemented using flex and bison.

SAMPLE INPUT AND EXPECTED OUTPUT:

Input Program:

```
/* comment line */
//another comment

public class test {
      public static void main (String[] args) {
      int a = 12/3;
      char cr = 'a';
      float fl = 6.7;
      if (true) {
            int ff = 555;
            int y =99;
            int c=5+10;
            ff = 5+y;
            int oo =444;
            if(a>=2) {
                  a=10;
            }
            else {
                  a=8;
            }
      }
      else {
            int ff = 10;
      }
      int fact = 1;
      for( int i =1;i<=5;i++)
      {
```

```
            fact = fact * i;
            int dummy = 10;
      }
      int R = 10;
      int z = a + 10;
      String str = "string";
      System.out.print(a);
      System.out.print(cr);
      System.out.print("hello world");
      System.out.print("1234");
      System.out.print(fact);
      System.out.print(z);
      }
}
```

Symbol Table Output:

Type	Name	Value	Scope	line	Size
int	a	8	1	6	4
char	cr	a	1	7	2
float	fl	6.700000	1	8	4
int	ff	104	2	10	4
int	y	99	2	11	4
int	c	15	2	12	4
int	oo	444	2	14	4
int	ff	10	2	29	4
int	fact	2	1	31	4
int	i	2	2	32	4
int	dummy	10	3	35	4
int	R	10	1	37	4
int	z	18	1	38	4
string	str	string	1	39	12

ACCEPTED

AST Generation:

```
                                        class_stmnt
                                            / \
                                           /   \
                                          /      \
                                         /        \
                                        /          \
                                      NULL   method_stmnt
                                                  / \
                                                 /    \
                                                /      \
                                               /        \
                                             NULL  main_method
                                                          \
                                                        stmnt
                                                         / \
                                                        /   \
                                                       /     \
                                                    stmnt   NULL
                                                     / \
                                                    /   \
                                                   /     \
                                                stmnt   NULL
                                                 / \
                                                /   \
                                               /     \
                                            stmnt   NULL
                                             / \
                                            /   \
                                           /     \
                                        stmnt   NULL
                                         / \
                                        /   \
                                       /     \
                                    stmnt   NULL
                                     / \
                                    /   \
                                   /     \
                                stmnt   NULL
                                 / \
                                /   \
                               /     \
                            stmnt   NULL
                             / \
                            /   \
                           /     \
                        stmnt   NULL
                         / \
                        /   \
                       /     \
                    stmnt   NULL
                     / \
                    /   \
                   /     \
                stmnt   NULL
                 / \
                /   \
               /     \
            stmnt   NULL
             / \
            /   \
           /     \
        stmnt   NULL
         / \
        /   \
       /     \
      /       \
     /         \
    /           \
 stmnt           =
  / \           / \
 /   \         /   \
```

```
                                                      / \          /  \
                                                     /   \       /    \
                                                    /     \     /      \
                                                stmnt      =   str  "string"
                                                  / \     / \
                                                 /   \   z   +
                                                /     \     / \
                                            stmnt      =  a  10
                                              / \     / \
                                             /   \   R  10
                                            /     \
                                           /       \
                                          /         \
                                         /           \
                                        /             \
                                       /               \
                                      /                 \
                                     /                   \
                                    /                     \
                                   /                       \
                                  /                         \
                                 /                           \
                              stmnt                         FOR
                               / \                          /  \
                              /   \                        /    \
                             /     \                      /      \
                            /       \                    /        \
                           /         \                  /          \
                        stmnt         =                /            \
                         / \         / \              /              \
                        /    \      /   \            /                \
                       /      \  fact    1          /                  \
                      /        \                   /                    \
                     /          \                 /                      \
                    /            \            FOR_ARGS                 stmnt
                   /              \              / \                     / \
                  /                \            /   \                   /   \
                 /                  \          /     \                 /     \
                /                    \        /       \               /       \
            stmnt                 IF_ELSE    =   loop_conds          /         \
              / \                   / \     / \       / \        stmnt          =
             /   \                 /   \   i   1     /   \         / \         / \
            /     \               /     \           /     \       /   \       /   \
           /       \             /       \       <=     ++    NULL   =   dummy  10
        stmnt       =           /         \       / \     /           / \
         / \       / \        IF       stmnt    i   5   i           /   \
        /   \     /   \      / \         / \                     fact    *
       /     \  fl   6.7    /   \       /   \                          / \
      /       \            /     \   NULL    =                        /   \
   stmnt       =         true   stmnt                                fact    i
    / \       / \                 / \        / \
   /   \     /   \               /   \      /   \
NULL    =   cr   'a'            /     \   ff   10
       / \                     /       \
      a   /                   /         \
         / \                 /           \
        12  3               /             \
                           /               \
                          /                 \
                         /                   \
                     stmnt                IF_ELSE
                       / \                   / \
                      /   \                 /   \
```

```
                     / \                   / \
                    /   \                 /   \
                   /     \               /     \
                  /       \             /       \
               stmnt       =           /         \
                / \       / \         IF        stmnt
               /   \     /   \       / \         / \
              /     \   oo   444    /   \       /   \
           stmnt     =             /     \    NULL   =
            / \     / \           >=    stmnt       / \
           /   \   ff  +         / \     / \       a   8
          /     \     / \       a   2   /   \
       stmnt     =   5   y            NULL   =
        / \     / \                         / \
       /   \   c   +                       a  10
      /     \     / \
   stmnt     =   5  10
    / \     / \
   /   \   y  99
 NULL   =
       / \
      /   \
     ff   555
```

ACCEPTED

Intermediate Code:

```
t0 = 12 / 3
a = t0
cr = 'a'
fl = 6.7
if(not true) goto L1
ff = 555
y = 99
t1 = 5 + 10
c = t1
t2 = 5 + y
ff = t2
oo = 444
t3 = a >= 2
if(not t3) goto L2
a = 10
goto L3

L2:
a = 8

L3:
goto L4

L1:
ff = 10

L4:
fact = 1
i = 1

L5:
t4 = i <= 5
if t4 goto L6
goto L8

L7:
t5 = i + 1
i = t5
goto L5

L6:
t6 = fact * i
fact = t6
dummy = 10
goto L7

L8:
R = 10
t7 = a + 10
```

```
z = t7
str = "string"

Line No. is 47
 ACCEPTED
```

Intermediate Code in quadruples format:

#	op	arg1	arg2	result
0	/	12	3	t0
1	=	t0		a
2	=	'a'		cr
3	=	6.7		fl
4	iffalse	true		L1
5	=	555		ff
6	=	99		y
7	+	5	10	t1
8	=	t1		c
9	+	5	y	t2
10	=	t2		ff
11	=	444		oo
12	>=	a	2	t3
13	iffalse	t3		L2
14	=	10		a
15	goto			L3
16	label			L2
17	=	8		a
18	label			L3
19	goto			L4
20	label			L1
21	=	10		ff
22	label			L4
23	=	1		fact
24	=	1		i
25	label			L5
26	<=	i	5	t4
27	if	t4		L6
28	goto			L8
29	label			L7
30	+	i	1	t5
31	=	t5		i
32	goto			L5
33	label			L6
34	*	fact	i	t6
35	=	t6		fact

36	=	10		dummy
37	goto			L7
38	label			L8
39	=	10		R
40	+	a	10	t7
41	=	t7		z
42	=	"string"		str

----

2. ARCHITECTURE OF LANGUAGE

Compiler for the following constructs is created:

- `for` loop
- `if, if-else` construct
- `int` data type
- `float` data type
- `char` data type
- `String` data type
- import statements
- `return, break, continue` statements
- modifiers and function calls
- Arrays
- Arithmetic and logical operators
- Comments (both single line and multiline comments)
- implicit conversion from int/char to float

We show errors for the following

- Redefinition of identifiers within the same scope
- Use of undeclared identifiers
- Unterminated comments
- If length of identifier exceeds maximum limit
- Syntax errors

All the errors and warnings are displayed along with line number

3. LITERATURE SURVEY

Lex Yacc and its internal working

https://www.tldp.org/HOWTO/Lex-YACC-HOWTO.html#toc1

Compiler Design fundamentals

https://web.stanford.edu/class/cs143/lectures/lecture01.pdf

Java Grammar

https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html

Building a mini-compiler - tutorial

https://www.tutorialspoint.com/compiler_design/index.htm

Expression evaluation using Abstract Syntax Tree

https://mariusbancila.ro/blog/2009/02/03/evaluating-expressions-part-1

4. CONTEXT FREE GRAMMAR

```
S:
      compilation_unit
      ;

compilation_unit:
      package_statement import_statement class_stmt
      ;

package_statement:
      T_PACKAGE T_IDENTIFIER T_SEMC
      |
      ;

import_statement:
      T_IMPORT package_name T_DOT T_MUL T_SEMC
      | T_IMPORT package_name T_DOT T_IDENTIFIER  T_SEMC
      | T_IMPORT T_MUL T_SEMC
      |
      ;

package_name:
      T_IDENTIFIER T_DOT T_IDENTIFIER
      ;

class_stmt:
      modifier T_CLASS T_IDENTIFIER T_OF stmnt method_stmnt T_CF
      ;

modifier:
      T_PUBLIC
      | T_PRIVATE
      | T_PROTECTED
      ;

method_stmnt:
      other_method main_method
      ;

other_method:
```

```
        other_method T_PUBLIC T_VOID T_IDENTIFIER T_OC T_CC T_OF
        stmnt T_CF
        |other_method T_STATIC T_VOID T_IDENTIFIER T_OC T_CC T_OF
        stmnt T_CF
        |
        ;


main_method:
        T_PUBLIC T_STATIC T_VOID T_MAIN T_OC T_STRING T_OS T_CS
        T_ARGS T_CC T_OF stmnt T_CF
        ;

stmnt:
        stmnt s1
        |
        ;

s1:
        variable_declaration T_SEMC
        | expression T_SEMC
        | if_else
        |for_stmt
        |print_stmnt T_SEMC
        | T_SEMC
        ;

variable_declaration:
        dtypes
         ;

dtypes:
        T_INT ids1
        | T_FLOAT ids2
        | T_CHAR ids3
        | T_STRING  ids4
        | array
        ;

array:
        T_INT T_OS T_CS T_IDENTIFIER T_ASSIGN T_NEW T_INT T_OS T_NUM
        T_CS   |T_FLOAT T_OS T_CS T_IDENTIFIER T_ASSIGN T_NEW
        T_FLOAT T_OS T_NUM T_CS
        ;

ids1:
        T_IDENTIFIER T_ASSIGN arithm_e
        | ids1 T_COMMA T_IDENTIFIER
        | T_IDENTIFIER
```

```
    ;

ids2:
    T_IDENTIFIER T_ASSIGN float_e
    | ids2 T_COMMA T_IDENTIFIER
    | T_IDENTIFIER
    ;

ids3:
    T_IDENTIFIER T_ASSIGN char_e
    | ids3 T_COMMA T_IDENTIFIER
    | T_IDENTIFIER
    ;

ids4:
    T_IDENTIFIER T_ASSIGN str_e
    | ids4 T_COMMA T_IDENTIFIER
    | T_IDENTIFIER
    ;

expression:
    arithm_e
    | rel_e
    ;

rel_e:
    arithm_e T_LT arithm_e
    | arithm_e T_GT arithm_e
     | arithm_e T_LE arithm_e
     | arithm_e T_GE arithm_e
     | arithm_e T_EQ arithm_e
     | arithm_e T_NE_OP arithm_e
     | T_TRUE
     | T_FALSE
     ;

arithm_e:
    arithm_e T_MUL arithm_e
    | arithm_e T_DIV arithm_e
    | arithm_e T_ADD arithm_e
    | arithm_e T_SUB arithm_e
    | T_IDENTIFIER
    | T_NUM
    | T_IDENTIFIER T_INC_OP
    | T_IDENTIFIER T_DEC_OP
    | T_INC_OP T_IDENTIFIER
    | T_DEC_OP T_IDENTIFIER
    | T_IDENTIFIER T_ASSIGN arithm_e
    ;
```

```
float_e:
	T_IDENTIFIER
	| T_IDENTIFIER T_ASSIGN float_e
	| T_DECIMAL
	;


char_e:
	T_IDENTIFIER
	| T_IDENTIFIER T_ASSIGN char_e
	| T_CONSTANT
	;

str_e:
	T_IDENTIFIER
	| T_IDENTIFIER T_ASSIGN str_e
	| T_STRING_LITERAL
	;

if_stmt:
	T_IF T_OC rel_e T_CC T_OF stmnt T_CF else_stmnt

	;

else_stmnt:
	T_ELSE if_stmt
	| T_ELSE T_OF {block = 1;} stmnt T_CF {block = 0;}
	|
	;

for_stmt:
	T_FOR T_OC for_args T_CC T_OF stmnt T_CF
	;

for_args:
	loop_init T_SEMC loop_cond
	;

loop_cond:
	arg2 T_SEMC arg3
	;

loop_init:
	variable_declaration
	| expression
	|
	;

arg2:
	rel_e
```

```
      |
      ;

arg3:
      arithm_e
      |
      ;

print_stmnt:
      T_SOP T_OC T_STRING_LITERAL T_CC
      | T_SOP T_OC T_IDENTIFIER T_CC
      ;
```

5. DESIGN STRATEGY

- **Symbol table creation-** The symbol table was implemented using a linear array of a structure that contains the identifier, scope, type and its value.

- **Abstract Syntax Tree-** This tree is constructed as the input is parsed. Each node of this tree contains a pointer to left, a pointer to right and a member for a string.

- **Intermediate Code Generation-** Intermediate code was generated that makes use of temporary variables and labels. Also all if-else statements were optimized to ifFalse statements to reduce the number of goto statements (an additional optimization provided).

- **Code Optimization-** Constant folding, Constant propagation, Copy Propagation and Dead code elimination were implemented as part of machine independent code optimization.

Constant Folding

When an arithmetic expression is encountered, we check to see if all the operands contain digits and are not identifiers. If all the operands are numbers we evaluate the expression.

Constant Propagation

When an identifier is encountered, we check the symbol table to see if an entry exists. If the entry exists we perform constant propagation.

Copy Propagation

Copy propagation is the process of replacing the occurrences of targets of direct assignments with their values. A direct assignment is an instruction of the form `x = y`, which simply assigns the value of `y` to `x`.

Removal of dead code

Dead code elimination (also known as DCE, dead code removal, dead code stripping, or dead code strip) is a compiler optimization to remove code which does not affect the program results.

Error handling- Type error and semicolon missing error have been handled. Redefinition of identifiers in the same scope, use of undeclared identifiers, unterminated comments, syntax errors have also been covered. If identifier length exceeds maximum limit error is shown for that as well.

6. IMPLEMENTATION DETAILS

Flex and Bison have been used to implement the following:

- **Symbol table creation:**

Implemented in parsefile.y. The symbol table is a linear array of the following structure

```
struct SymTable {

    char idName[50];

    int value;

    float f_val;

    char c_val;

    char s_val[100];

    int type;  //0-int , 1-float, 2-char, 3-String, 4-int[], 
5-float[]

    int line_no;

    int scope;

    int size;

}; struct SymTable st[50];
```

```
bison -d parsefile.y
flex lexfile.l
gcc lex.yy.c parsefile.tab.c
./parsefile.out < input2.java
```

- **Abstract Syntax Tree:**

Implemented in AST.y and AST.c

To implement this in flex and bison, we first redefine the YYSTYPE in the yacc file that defaults to int. We create a node structure as follows:

```
typedef struct node

  {

    struct node *left;

    struct node *right;

    char *value;

  } node;

  node *createnode(node *left, node *right, char *value) {

      node *newnode = (node *)malloc(sizeof(node));

      char *newstr = (char *)malloc(strlen(value)+1);

      strcpy(newstr, value);

      newnode->left = left;

      newnode->right = right;

      newnode->value = newstr;

      return(newnode);

}
```

bison -d ast.y
flex lexfile.l
gcc lex.yy.c ast.tab.c ast.c
./ast.out < input2.java

- **Intermediate Code Generation:**

Implemented in intercode.y

The given code was converted into 3 address code

bison -d intercode.y

flex lexfile.l
gcc lex.yy.c intercode.tab.c intercode.c
./intercode.out < input2.java

- **Code Optimization:**

Implemented in optimize.py

Constant Folding

Constant folding is the process of recognizing and evaluating constant expressions at compile time rather than computing them at runtime. Terms in constant expressions are typically simple literals, such as the integer literal 2, but they may also be variables whose values are known at compile time.

This is done using the below function in our code:

```
def constant_folding(list_of_lines)
```

Constant Propagation

Constant propagation is the process of substituting the values of known constants in expressions at compile time. Such constants include those defined above, as well as intrinsic functions applied to constant values.

This is done using the below function in our code:

```
def constant_propagation():
```

Copy propagation

Copy propagation is the process of replacing the occurrences of targets of direct assignments with their values. A direct assignment is an instruction of the form `x = y`, which simply assigns the value of `y` to `x`.

```
def copy_propagation():
```

Removal of dead code

Dead code elimination (also known as DCE, dead code removal, dead code stripping, or dead code strip) is a compiler optimization to remove code which does not affect the program results.

```
def remove_dead_code(list_of_lines) :
```

● **Error Handling-** Implemented in parsefile.y using conditional statements for type error and semicolon missing is checked using the grammar and unused variables. Redefinition of identifiers in the same scope, use of undeclared identifiers, unterminated comments, syntax errors have also been covered. If identifier length exceeds maximum limit error is shown for that as well.

python3 optimize.py #change the path to the file name to icg2.txt

7. RESULTS

A mini java compiler that can compile the chosen constructs has been created.

8. SNAPSHOTS

<u>SYMBOL TABLE GENERATION</u>

bison -d parsefile.y
flex lexfile.l
gcc lex.yy.c parsefile.tab.c
./parsefile.out

<u>Input file1:</u>

```
vibha@PES1201800158client:~/Desktop/CD$cat input2.java
/* comment line */
//another comment

public class test {
	public static void main (String[] args) {
	int a = 12/3;
	char cr = 'a';
	float fl = 6.7;
	if (true) {
		int ff = 555;
		int y =99;
		int c=5+10;
		ff = 5+y;
		int oo =444;
		if(a>=2) {
			a=10;
		}
		else {
			a=8;
		}
	}
	else {
		int ff = 10;
	}
	int fact = 1;
	for( int i =1;i<=5;i++)
	{
		fact = fact * i;
		int dummy = 10;
	}
	int R = 10;
	int z = a + 10;
	String str = "string";
	System.out.print(a);
	System.out.print(cr);
	System.out.print("hello world");
	System.out.print("1234");
	System.out.print(fact);
	System.out.print(z);
	}
}vibha@PES1201800158client:~/Desktop/CD$
```

Output1:

```
vibha@PES1201800158client:~/Desktop/CD$./parsefile.out < input2.java
8
a
hello world
1234
2
18

| Type   | Name  | Value | Scope | line  | Size
| int    | a     | 8     | 1     | 6     | 4
| char   | cr    | a     | 1     | 7     | 2
| float  | fl    | 6.700000      | 1     | 8     | 4
| int    | ff    | 104   | 2     | 10    | 4
| int    | y     | 99    | 2     | 11    | 4
| int    | c     | 15    | 2     | 12    | 4
| int    | oo    | 444   | 2     | 14    | 4
| int    | ff    | 10    | 2     | 23    | 4
| int    | fact  | 2     | 1     | 25    | 4
| int    | i     | 2     | 2     | 26    | 4
| int    | dummy | 10    | 3     | 29    | 4
| int    | R     | 10    | 1     | 31    | 4
| int    | z     | 18    | 1     | 32    | 4
| string        | str   | string        | 1     | 33    | 12

 ACCEPTED
vibha@PES1201800158client:~/Desktop/CD$
```

Input2:

```
vibha@PES1201800158client:~/Desktop/CD$cat input.java
class Test
{
//Comment 1
/* multi line comment*/
                int b;
                public static int main()
                {
                int a=5;
                b=1;
                int c=3;
                b = a+c;
                System.out.print(c);

        }
        int c=9;
        System.out.print(c);
}

vibha@PES1201800158client:~/Desktop/CD$
```

Output:

```
vibha@PES1201800158client:~/Desktop/CD$./parsefile.out < input.java
Error: syntax error, unexpected T_CLASS, expecting T_PUBLIC or T_PRIVATE or T_PROTECTED
Error occured at  Line No.  1
Error before : class
vibha@PES1201800158client:~/Desktop/CD$
```

AST Generation:

bison -d ast.y
flex lexfile.l
gcc lex.yy.c ast.tab.c ast.c
./ast.out

Input:

```
public class test {
        public static void main (String[] args)
        {

                int a=0;
                int b = a;
                System.out.print("HELLO WORLD");

        }

}
```

Output:

```
vibha@PES1201800158client:~/Desktop/CD$./AST.out < input.java
  class_stmnt
      / \
     /   \
    /     \
   /       \
  /         \
 /           \
NULL   method_stmnt
            / \
           /   \
          /     \
         /       \
       NULL  main_method
                   \
                  stmnt
                   / \
                  /   \
                 /     \
              stmnt   NULL
               / \
              /   \
             /     \
          stmnt   NULL
           / \
          /   \
         /     \
      stmnt     =
       / \     / \
      /   \   b   a
    NULL   =
          / \
         a   0


 ACCEPTED
vibha@PES1201800158client:~/Desktop/CD$
```

INTERMEDIATE CODE GENERATION:

bison -d intercode.y
flex lexfile.l
gcc lex.yy.c intercode.tab.c intercode.c
./intercode.out

Input:

```java
public class test {
        public static void main (String[] args)
        {

                int a=0;
                int b = a;
                System.out.print("HELLO WORLD");

        }

}
```

Output:

```
vibha@PES1201800158client:~/Desktop/CD$./intercode.out < input.java
a = 0
b = a

Line No. is 11
 ACCEPTED



the value of inx 2

--------------------------------------------------------

INTERMEDIATE CODE

--------------------------------------------------------
--------------------------------------------------------

              #        op      arg1      arg2    result
--------------------------------------------------------

              0         =         0                   a
              1         =         a                   b
                 ----------------------
vibha@PES1201800158client:~/Desktop/CD$
```

CODE OPTIMIZATION:

python3 optimize.py

Output:

```
vibha@PES1201800158client:~/Desktop/CD$python3 optimize.py
ICG:
a = 0
b = a
dummy = 10
t0 = 5 + 3
c = t0
t1 = c + 10
d = t1



After Copy Propagation:
a = 0
b = a
dummy = 10
t0 = 5 + 3
c = t0
t1 = t0 + 10
d = t1



After Constant Propagation:
a = 0
b = a
dummy = 10
t0 = 5 + 3
c = t0
t1 = t0 + 10
d = t1



After Constant Folding:
a = 0
b = a
dummy = 10
t0 = 8
c = t0
t1 = t0 + 10
d = t1
```

```
After Constant Propagation:
a = 0
b = a
dummy = 10
t0 = 8
c = t0
t1 = 8 + 10
d = t1



After Constant Folding:
a = 0
b = a
dummy = 10
t0 = 8
c = t0
t1 = 18
d = t1



After Dead Code Elimination:
a = 0
b = a
dummy = 10
t0 = 8
c = t0
t1 = 18
d = t1
vibha@PES1201800158client:~/Desktop/CD$
```

9. CONCLUSIONS

A compiler for JAVA was thus created using flex and bison. In addition to the constructs specified, basic building blocks of the language (declaration statements, assignment statements, function creation etc) were handled.

This compiler was built keeping the various stages of Compiler Design, ie, Lexical Analysis, Syntax Analysis, Semantic Analysis and Code Optimisation in mind.

As a part of each stage, an auxiliary part of the compiler was built (Symbol Table, Abstract Syntax Tree and Intermediate Code). Each of these components are required to compile code successfully.

In addition to this, basic error handling has also been implemented.

Through this process, all kinds of syntax errors and certain semantic errors in a JAVA program can be caught by the compiler.

10. FURTHER ENHANCEMENTS

- Functionality for `switch` and other flavours of `while` can be implemented.
- The compiler can be constructed to recover from more kinds of errors.
- More code optimization strategies can be implemented.