American Journal of Engineering Research (AJER)
E-ISSN: 2320-0847 p-ISSN: 2320-0936
Volume-8, Issue-2, pp-45-52
www.ajer.org
Research Paper Open Access

An efficient approach for error handling and recovery strategies in the context of compiler design

Md. Alomgir Hossain
[1]*Senior Lecturer, Department of Computer Science and Engineering,*
[2]*Md. Mohosin Miah, Student, Department of Computer Science and Engineering*
[3]*Mahmudul Islam Sajib, Student, Department of Computer Science and Engineering*
[4]*Shoeib Mahmud Shargo, Student, Department of Computer Science and Engineering*
IUBAT–International University of Business Agriculture and Technology, Uttara, Dhaka, Bangladesh
Corresponding Author: Md. Alomgir Hossain

ABSTRACT: *The task of a compiler is to compile a program or instruction which is written in a particular source language and convert it into a targeted language via various phases of compiler. Error Handling in compiler design is associated with failure, mainly because of errors in the compiler or its environment, incomplete understanding of source language, transcription errors, incorrect data, etc. In this paper, we have shown the types of error compiler faces and error detection techniques of compiler. We have also introduced an algorithm for performing error detection for parsing computer programs. The purpose of this paper is to provide entire knowledge about Error handling in compiler design and its implementation.*

KEYWORDS: *Compiler, Error handling, compiler design, error detection, lexical error.*

Date of Submission: 24-01-2019 Date of acceptance: 08-02-2019

I. INTRODUCTION

Compiler is basically a program designed to convert human readable higher-level programming language into machine language, or source code. While converting these codes or programs, compiler faces some errors. Compiler is unable to determine the main reason for the error, but it is certainly able to detect and analyze the visible symptoms. However, the main reason for error handling is to help the programmer by highlighting inconsistency in their code. Error Handling has a low frequency in comparison with other compiler task and that is why the time it takes to complete is irrelevant. Like doctors, error handles only symptoms. From these symptoms, compilers detect, determine and diagnosis the underlying error. The diagnosis of these symptoms are complex and uncertain, hence, we just report the symptoms with no more attempt to diagnosis. [1]

Here, I point out the rest of the paper is planned as follows. Covers the literature review section II. and III, Methodology, IV Error detection processes and Error recovery methods, V Algorithm, VI Result analysis and calculations, VII. Comparison, VIII Findings and last all about Conclusion IX and References X.

II. LITERATURE REVIEW

Error handling means anticipation, detection and resolution of programming, application, and communications errors [2]. Error handlers are available for some applications, those are specialized program. The best programs of this sort forestall errors if doable, recover it from them when they happen without terminating the application, or terminate an affected application and save the error data to a log file. In programming, an error is one that can be prevented. Such an error can happen in syntax phase or logical phase. Syntax errors which are typing mistakes or improper use of special characters like semicolon are handled by through proof reading. Logical errors which are also called bugs happen when executed code does not produce the expected or required result. Logical errors are best handled by rigorous program debugging. A run time error takes place throughout the execution of a program and usually occurs because of adverse system parameters or invalid input data. An example is that the lack of enough memory to run an application or a memory conflict with another program. Run-time errors can be resolved, or their impact can be minimized by the use of error handler programs. [2]

III. METHODOLOGY

According to the performance analysis we signify a graph and that will represent the performance of our proposed algorithm into different criteria. So, we think our analysis goes into quantitative method.For making this paper, we collect data from secondary source and we also make out some research papers which are already published as well as some articles. There are few steps that we followed to prepare this paper:
Step 1: Collect data from existing research paper
Step 2: Collect data from internet
Step 3: Examining existing algorithms
Step 4: Implementing our proposed algorithm
Step 5: Comparing the existing and new Algorithm on different benchmark
Step 6: Final we obtain the result

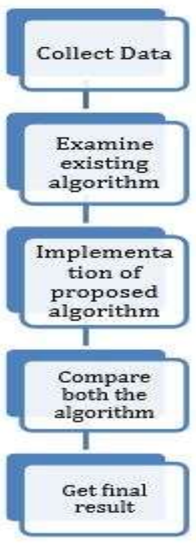

Collect Data
Examine existing algorithm
Implementa tion of proposed algorithm
Compare both the algorithm
Get final result

Figure 1: Diagram of workflow

Sources of Error: Error Handling process are used to determine each error, report it to the user and then the process make some recover strategy and implement them to handle error [3] While checking or looking for these errors the processing time of program should not be slow. However, an error is the blank entries in the symbol table. Two types of error: run-time and compile time error:

1. An error which takes place during the execution of a program is run time error and it usually occurs because of adverse system parameters or invalid input data. The lack of sufficient memory to run an application or a memory conflict with another program and logical error are example of this. When executed code does not produce the desired result. By meticulous program debugging logical errors are removed.
2. An error that occurs during compile time is called compile time error and it happens before the execution of the program. Examples of compile time error are syntax error or missing file reference that prevents the program from successfully compiling [3]

Compile-time errors:
1. Lexical: Misspellings of identifiers, keywords or operators are included in this category
2. Syntactical: Unbalanced parenthesis or missing semicolon.
3. Semantical: Incompatible value assignment or type mismatches between operator and operand.
4. Logical: infinite loop, code not reachable.

Finding or reporting an error:
For early detection of syntax errors viable prefix of a parser is used.
1. Goal: without further consuming unnecessary input detecting of an error as soon as possible

2. How: as soon as the prefix of the input does not match a prefix of any string in the language detection of an error occurs.
3. Example: for (;), this will report an error as for have two semicolons inside braces.

Error Recovery:
The main requirement for the compiler is to stop and issue a message, and cease compilation. There are some common recovery methods:

1. **Panic mode recovery:** Basically, it prevents the parser from developing infinite loops while recovering error and this is the easiest way of error recovery. The parser discards the input symbol one at a time until one of the designated (like end, semicolon) set of synchronizing tokens is found. This is enough when the presence of multiple errors in same statement is rare. For example: Consider the erroneous expression- (1 + + 2) + 3. Panic mode recovery skips ahead to next integer and then continues. Bison: use the special terminal error to describe how much input to skip.
2. E-> int|E+E|(E)|error int|(error)

3. **Phase level recovery:** error correction is difficult in this strategy. But, perform local correction on the input to repair the error.
4. **Error productions:** There are some common errors known to the compiler designers that may occur in the code. Augmented grammars can also be used, as productions that generate erroneous constructs when these errors are encountered. Example: write 5x instead of 5*x .
5. Global correction: Goal is to make as few changes as possible while converting an incorrect input string to a valid string. This strategy is very costly to implement [3].

IV. ERROR DETECTION PROCESSES AND ERROR RECOVERY METHODS IN COMPILER:

In this step of source program compilation process, all possible user generated errors are diagnosed and reported to the user in form of error messages. The method in which a compiler can identify possible errors and outline the errors to end users of is known as Error Handling process [4].
Tasks of Error handler:
1. Detection 2. Reporting 3. Recovery

I. Lexical phase errors:
These errors are diagnosed during the lexical analysis phase. Regular lexical (scanner) errors are:
1. Identifier or numeric constants lengths that is extraneous.
2. Presence of illegal characters
3. Strings that aren’t matched
Example 1: printf ("Hello World"); ^ Since ^ is an illegal character, a lexical occurs at the end of the statement.
Example 2: A comment */ since the beginning of the comment is not present, a lexical error occurs here.

Lexical Error recovery:
1. If until a classified set of synchronizing tokens is found, consecutive characters from the input are deleted one at a time in this method. Tokens that are synchronized (e.g. } or ;) are called delimiters.
2. Ease of implement is an advantage of this mode and it makes sure not to go to infinite loop.
3. A remarkable amount of input is skipped without checking it for additional errors is a disadvantage of this mode.

II. Syntactic phase errors:
These errors are detected during syntax analysis phase. General syntax errors are
1. Structural Errors
2. Incorrectly spelled keywords
3. Missing operators
4. Parenthesis that aren’t balanced

Example: while (condition){ ....}
Here, the keyword while is written as while which is an error. Hence, “Unidentified keyword” error occurs in this case.
Syntactical Error recovers: Panic Mode Recovery
1. If until a classified set of synchronizing tokens is found, consecutive characters from the input are deleted one at a time in this method. Tokens that are synchronized (e.g. } or ;) are called delimiters.
2. Ease of implement is an advantage of this mode and it makes sure not to go to infinite loop

3. A remarkable amount of input is skipped without checking it for additional errors is a disadvantage of this mode [3]

Statement Mode recovery:
1. In this method, when a parser runs into an error, it accomplishes necessary correction on remaining input and the rest of the input statement allows the parser to parse ahead.
2. Removing an extra semicolon or putting into a missing semicolon can be the way of correction.
3. First and foremost concern should be on not going in untended infinite loop, while accomplishing correction
4. The drawback is that it finds hard to tackle situations where actual error occurred before the point of diagnosis.

Error production:
1. If end users have common errors that can be tackled beforehand, these errors can be consolidated by augmenting the grammar with error productions that produce invalid constructs.
2. If this method is used then, during parsing proper error messages can be created and parsing won't halt.
3. Drawback is - maintenance is an overhead.

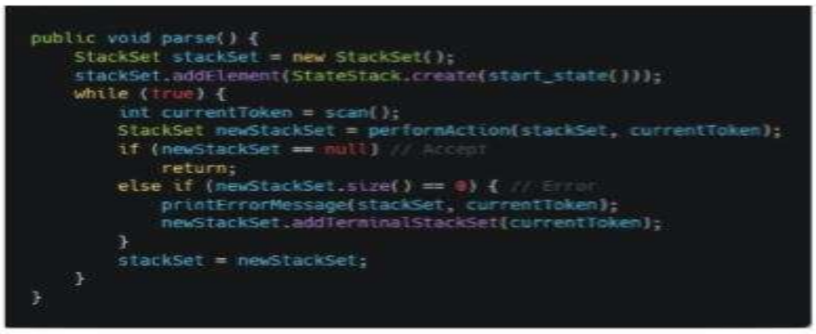
Global Correction:
1. The Entire source program is investigated by the parser and it seeks to get the nearest error free match.
2. The nearest match program has ignorable number of insertions, deletions and changes of tokens to retrieve from in accurate input.
3. This method is not applied in real-life, due to unreasonable time and space complexity.

III. Semantic errors:
These errors are detected during semantic analysis phase. Typical semantic errors are
1. Operands with non-compatible type
2. Variables those are undeclared
3. Formal and actual arguments miss-match
Example: int x [5], y;   ....   x = y;
Since an incompatible type of x and y is occurred, a semantic error is generated.

Semantic Error recovery:
1. A symbol table entry for correlating identifier is created to recover in case of occurrence of "Undeclared Identifier" error.
2. If data types of two operands are not compatible with each other then, the compiler performs automatic type conversion. [5]

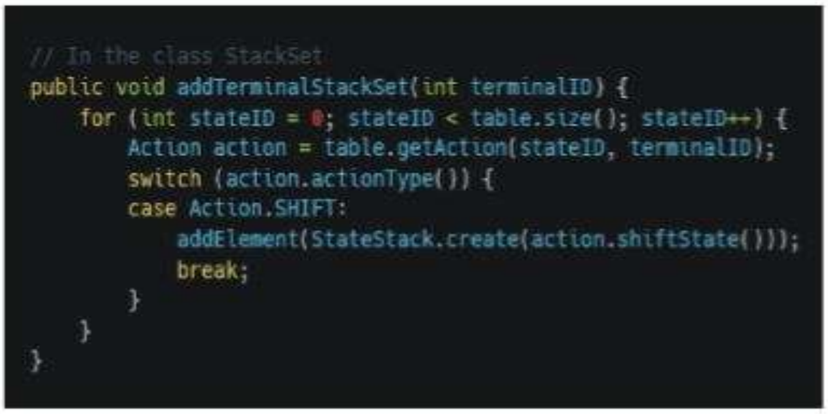
V. AN ALGORITHM FOR ACCOMPLISHING ERROR DETECTION WHILE PARSING SOURCE PROGRAMS:

We used bottom up parsing technique, for example LALR (1) parsing, which are used by yacc, Java CUP [6]. We have a finite set of partial stacks at each stage. This set contains a single stack initially, along with the start state pushed on. The set of partial stacks grasp a single element that correlates to the typical stack as long as no error encounters. The set of partial stacks often does contain several stacks after a syntax error is encountered. We call them "partial stacks", since we do not have a prior context. We know which states are presented on the top portion of the stack, but we've no idea about the bottom portion.


```
public void parse() {
    StackSet stackSet = new StackSet();
    stackSet.addElement(StateStack.create(start_state()));
    while (true) {
        int currentToken = scan();
        StackSet newStackSet = performAction(stackSet, currentToken);
        if (newStackSet == null) // Accept
            return;
        else if (newStackSet.size() == 0) { // Error
            printErrorMessage(stackSet, currentToken);
            newStackSet.addTerminalStackSet(currentToken);
        }
        stackSet = newStackSet;
    }
}
```

Code Snippet 1: Error detection algorithm

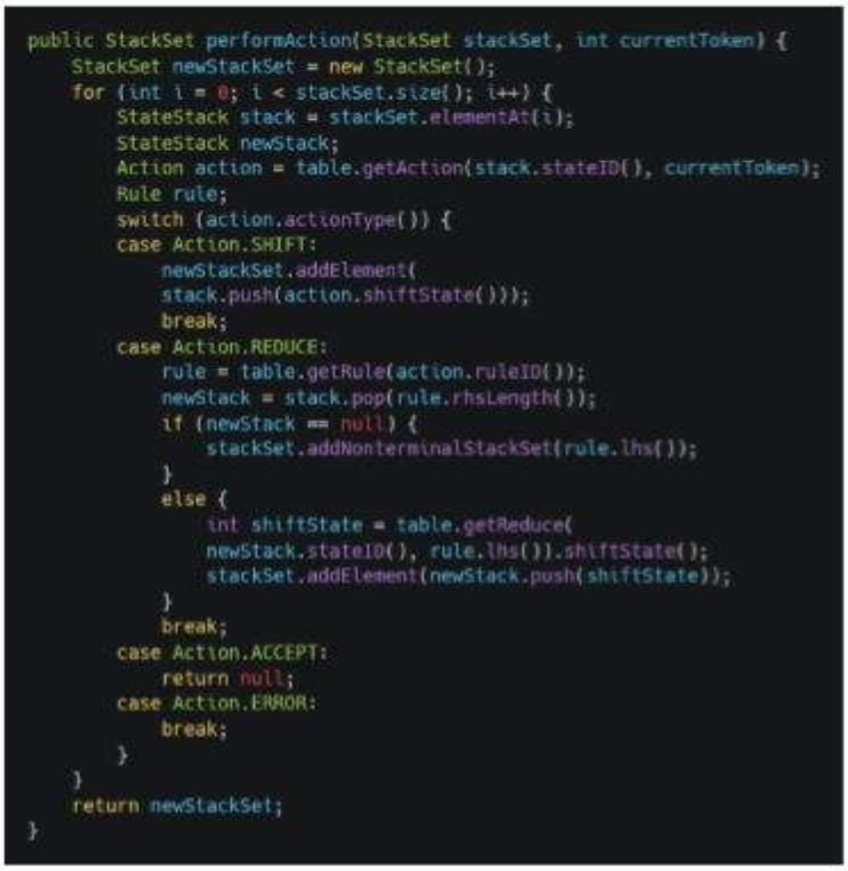
If we get empty set of partial stacks, and no second parse is there, then we encounter syntax error. We have output an error message and all possible partial stacks are generated with a single state that could generate after switching the current token on the stack.



```
// In the class StackSet
public void addTerminalStackSet(int terminalID) {
    for (int stateID = 0; stateID < table.size(); stateID++) {
        Action action = table.getAction(stateID, terminalID);
        switch (action.actionType()) {
        case Action.SHIFT:
            addElement(StateStack.create(action.shiftState()));
            break;
        }
    }
}
```

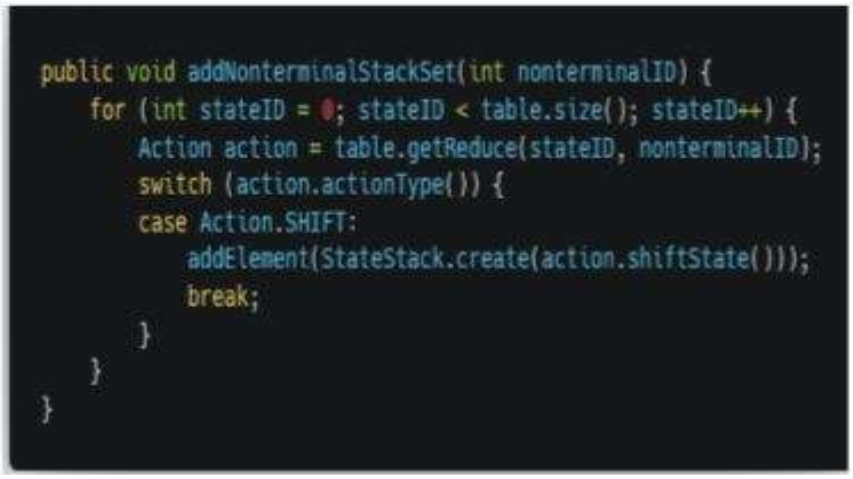
Code Snippet 2: Error detection algorithm

Accomplishing a shift-reduce parse which includes a set of partial stacks is similar to accomplishing it for a single stack. We create a brand-new stack by performing the action indicated by the top of stack state, and current token for each stack in the previous old set [7]. If the action is accepting, the action is clear otherwise action is error and we discard the stack. In case, the action is to reduce by a rule, there are two situations we can take into account. We have to pop off as many states from the stack as there are symbols on the right-hand side of the rule, and push on a state corresponding to the left-hand side of the rule. If we notice that the stack contains more states on it than we need to pop off. The normal task can be accomplished. If there are inadequate states to pop off, we delete the stack, and simulate all possible partial stacks with a single state that could occur after switching the left-hand side onto the stack.



```
public StackSet performAction(StackSet stackSet, int currentToken) {
    StackSet newStackSet = new StackSet();
    for (int i = 0; i < stackSet.size(); i++) {
        StateStack stack = stackSet.elementAt(i);
        StateStack newStack;
        Action action = table.getAction(stack.stateID(), currentToken);
        Rule rule;
        switch (action.actionType()) {
        case Action.SHIFT:
            newStackSet.addElement(
            stack.push(action.shiftState()));
            break;
        case Action.REDUCE:
            rule = table.getRule(action.ruleID());
            newStack = stack.pop(rule.rhsLength());
            if (newStack == null) {
                stackSet.addNonterminalStackSet(rule.lhs());
            }
            else {
                int shiftState = table.getReduce(
                newStack.stateID(), rule.lhs()).shiftState();
                stackSet.addElement(newStack.push(shiftState));
            }
            break;
        case Action.ACCEPT:
            return null;
        case Action.ERROR:
            break;
        }
    }
    return newStackSet;
}
```

Code Snippet 3: Error detection algorithm

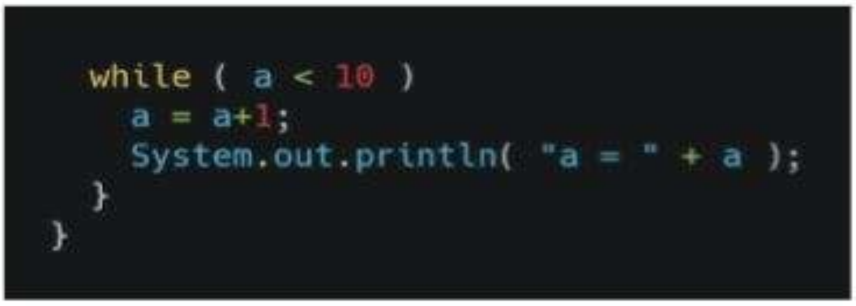


```java
public void addNonterminalStackSet(int nonterminalID) {
    for (int stateID = 0; stateID < table.size(); stateID++) {
        Action action = table.getReduce(stateID, nonterminalID);
        switch (action.actionType()) {
        case Action.SHIFT:
            addElement(StateStack.create(action.shiftState()));
            break;
        }
    }
}
```

Code Snippet 4: Error detection algorithm

VI. RESULT ANALYSIS (APPLYING THE ALGORITHM):

Suppose we have a syntax error in a Java program, followed by the input



```java
   while ( a < 10 )
     a = a+1;
     System.out.println( "a = " + a );
   }
}
```

Code Snippet 5: Error detection algorithm

The parser prints an error message and creates the three alternative partial stacks corresponding to the three states that can result by shifting “while”.

WHILE 330
WHILE 433
WHILE 471

These three states correspond to an ordinary “while” statement not nested inside the “then” part of an “if” statement, a “while” statement nested inside the “then” part of an “if” statement, and a “do... while” statement.

{WhileStatement→WHILE. LEFT Expression RIGHT Statement}

{While Statement No ShortIf→WHILE. LEFT Expression RIGHT Statement No ShortIf, While Statement→ WHILE. LEFT Expression RIGHT Statement}

{Do Statement→ DO Statement WHILE, LEFT Expression RIGHT SEMICOLON}

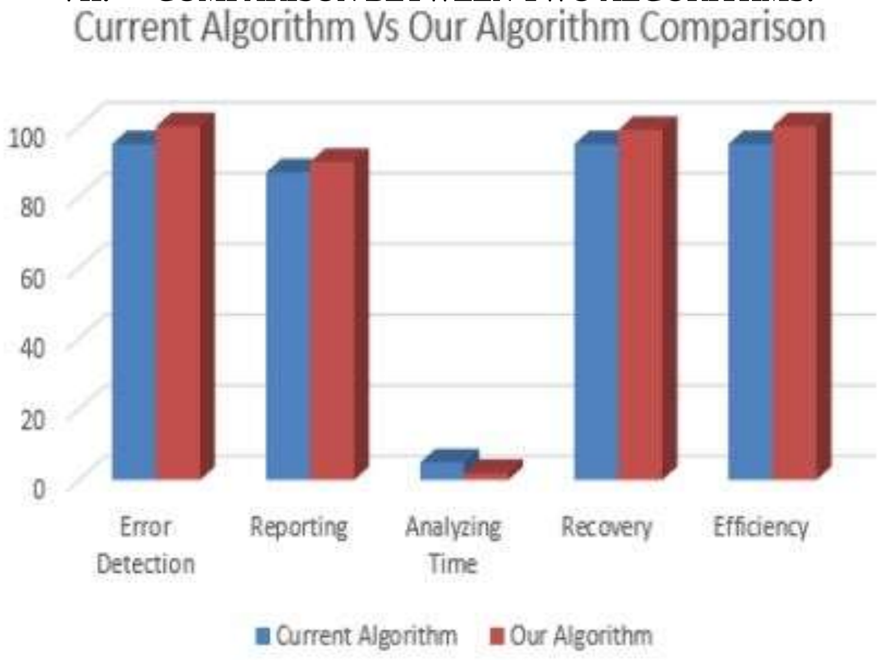
After parsing up to the beginning of the assignment statement, the third alternative drops out. WHILE 330 LEFT 365 Expressions 366 RIGHT 367 IDENT 333
WHILE 433 LEFT 461 Expression 462 RIGHT 463 IDENT 439 Eventually the sub-statement is shifted onto the stack.
WHILE 330 LEFT 365 Expressions 366 RIGHT 367 Statements Expression 286 SEMICOLONS 484
WHILE 433 LEFT 461 Expressions 462 RIGHT 463 Statements Expression 286 SEMICOLONS 484
The whole “while” statement is reduced to “Block Statement”, then “Block Statements”, and the identifier is shifted onto the stack.

Block Statements 486 IDENT 333 Block Statements 280 IDENT 333 Block Statements 383 IDENT 333 Block Statements 356 IDENT 333
It is worth noting that the number of alternatives increases at this point. [6] The reason for this is that there are four different contexts in which “Block Statements” can occur, namely
{Block→LEFTCURLY Block Statements. RIGHTCURLY}
{ConstructorBody→LEFTCURLY Block Statements. RIGHTCURLY}
{ConstructorBody→LEFTCURLY ExplicitConstructorInvocation Block Statements. RIGHTCURLY}
{SwitchBlockStatementGroup→ Switch Labels Block Statements}

The parsing continues, shifting the method invocation Onto the stack, reducing it to “Statement Expression”, shifting the semicolon onto the stack, reducing “Statement Expression;” to “Block Statement”, then “Block Statements Block Statement” to “Block Statements”, etc., shifting on the right brace, and obtaining

Block Statements 486 RIGHTCURLY 488 Block Statements 280 RIGHTCURLY 485
Block Statements 383 RIGHTCURLY 385 \ SwitchBlockStatementGroups 352 RIGHTCURLY 354
After shifting on the final right brace, we get
ClassBodyDeclarationsOpt 260 RIGHTCURLY 265
Block Statements 486 RIGHTCURLY 488 Block Statements 280 RIGHTCURLY 485 Block Statements 383 RIGHTCURLY 385
SwitchBlockStatementGroups 352 RIGHTCURLY 354
The current token is now end of file, and the first alternative manages to reduce down an accept state, so there are no more error messages. [6]

VII. COMPARISON BETWEEN TWO ALGORITHMS:



Graph 1: Total performance of each algorithm.

From graph 1, we have analyzed that existing algorithm and our proposed algorithm result is not same. Also it is exposed that our proposed algorithm can handle an error more efficiently and effortlessly comparing with current algorithm.

VIII. FINDINGS

In our paper, calculation made by java based IDE’s software. In this work, I found some limitation regarding programming skills and knowledge. Java is an object oriented programming language.
1. Must have knowledge at least basic understanding of how to write program in java, compile, run and need to Object Oriented Programming concept [8].
2. This is a great problem in the Java language and developers of the language haven’t been able to beat this difficulty. Java takes extra memory space than the other native programming languages like C and C++ [8].

IX. CONCLUSION

In this stage, Compiler is a translator tools that translates one language to another language and performs as like as compiler, interpreter and assembler. It converts the code written in one language to some other language without changing the meaning of the program. It is also estimated that a compiler should make the target code resourceful and optimized in terms of time and space. It covers basic translation mechanism and error detection & recovery. It includes lexical, syntax, and semantic analysis as front end, and code generation and optimization as back-end. Mainly the paper focused the result of existence and proposed algorithm which is measured efficiency and details of error encountered by compiler and a variety of error detection techniques. We proposed an algorithm that can be used for error detection effortlessly and the compiler can perform required action effortlessly if it encounters any error. Though there is still room for improvement on optimizing the algorithm so that the compiler faces less obstacle but it will work on most cases effectively. Our future work will investigate new model and techniques for error recovery in compiler design.

REFERENCES

[1]. H. C. A. V. Jatin Chhabra, "Research paper on Compiler Design," International Journal of Innovative Research in Technology, vol. 5, no. January 2019, pp. 1-2, 2014.
[2]. M. Rouse, "SearchSoftwareQuality," 2007. [Online]. Available: https://searchsoftwarequality.techtarget.co m/definition/error-handling. [Accessed 08 January 2019].
[3]. P. Agrawal, "geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/errorhandling-compiler-design/. [Accessed 08 January 2019].
[4]. K. C. Louden, Compiler Construction: Principles and Practice, 1997.
[5]. M. S. L. R. S. J. D. U. Alfred V. Aho, Compilers: Principles, Techniques, and Tools, vol. 2, Pearson, 1986.
[6]. The University of Auckland," [Online]. Available: https://www.cs.auckland.ac.nz/courses/co mpsci330s1c/ lectures/ 330ChaptersPDF/ A pp1.pdf. [Accessed 08 January 2019].
[7]. H. M/Senbrck, "A Generator for Production Quality Compilers," Springer, p. 01, 2018.
[8]. https://www.quora.com/What-are-disadvantages-of-Java