Compilation Technique Learning Design Using Automatic Lessimic Analysis Method

Compilation Technique Learning Design Using Automatic Lessimic Analysis Method

S Nita*, E R N Sari, O Prismatura

Teknik Informatika Fakultas Teknik Universitas PGRI Madiun

*nita@unipma.ac.id

Abstract. The compilation process is the process of translating programs written in high-level programming languages (source programs) into machine language (object programs). Source programs such as, pascal, basic, c++ while the program objects such as compilation files have an extension file such as*.exeor*.com.The file can run on its own without he help ofthe source program (the file has property executable). Translators are divided in to two types of interpreter sand compilers.Both functions are the same, the difference lies in the translation process. In this research, a compilation machine called Automatic Lessimic Analyzer which can be used as an analysis machine. There are two job descriptions of the analysis machine, in the front as an analysis and behind as a synthesis function. For example, a simple program is included (c++) can be analyzed successfully and it can detect errors as a source program. While at the end, it will generate code generation and program object optimization.

1. Introduction

In learning compilation techniques a lot of references are needed can support to understand the work process of compilation machines. The compilation engine starts working when a program was run by the user. The program was made by a programmer who has mastered programming techniques. Many programs were made with different programming languages (source languages), one of them with c ++ programming languages. Programming is required to be able to analyze program algorithms are in accordance with the rules of the specified algorithm. According to Donald Ervin Knuth (1968) in his book states that Algorithms are a sequence of steps in problem solving in the form of sentences with a limited number of words and arranged logicallya and systematically [1].

When the program starts (running) it will be seen how the quality of the algorithm has been translated into programming languages. The faster the results of a program the better than this program and it is clearly free from errors in supported programs. An important factor which the programmer must know is a successful compilation technique. FirarrUtdirartatmo (2005) said that, compilation technique is a technique in reading a program written in the source language, then translated into another language called the target language. The translation process carried out by the compiler is called the compilation process (compiling)[2].

For this reason, the author asks the program to complete and manage the process to make the program inside, which contains errors and must be able to resolve the error.

The problem of the question writer is quickly and precisely the results of the compilation process with the application help. The author found a reference of the help application to test the support of compilation programs, namely by Automatic Lessimic Analyzer [3].

2. Theoretical Review

2.1. Compilation Process.

The compilation process was called the translation process. So,compilation programs were a translation process from programs written in high-level languages to low-level languages (program objects)[4]. A good compilation (Compiler) can work well on computer machines and it does not require a long process. There were 3 ways to make compilation, and the language used : [5]

a. Machine language, its characteristics are very difficult to understand by humans, because it is very dependent on the engine
b. Assembly language, the features of the facility are limited, the order is brief.
c. High Level Language, character is easily understood by humans, more facility

The main task of the compiler is divided into 2, namely [6]:

a. The front end function, the task of decomposing the source program into its basic parts.
b. Synthesis (back end) function, the task is to generate and optimize the object program.

The compiler task is illustrated in the form of a compiler model as below:

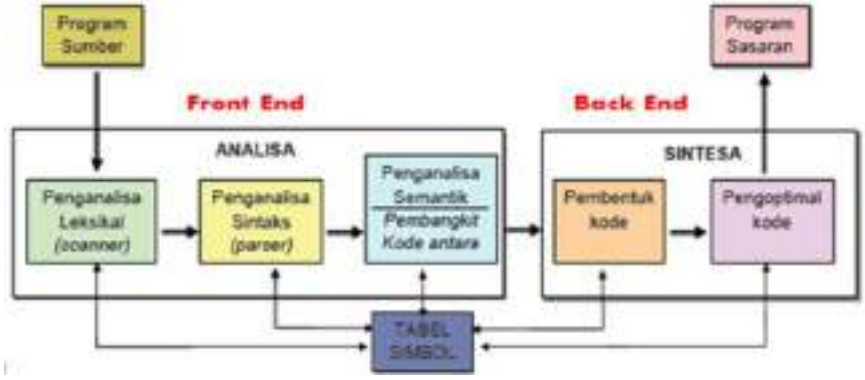

Figure 1. Compiler Model

First Function Analysis was divided into 3 stages:

a. Lexical Analyzer (Scanner), decomposes the source program into small parts.
b. Syntactic Analyzer (Parser). Syntax is the arrangement of sentences and the rules in forming sentences are called grammar. The syntax analyzer in the compilation field was often called a Parser. To analyze sentences, parse-tree help was usually used.
c. Semantic Analyzer Intermediate Code Generator, usually in realization it will be combined with an intermediate code generator (the part that functions to generate the intermediate code) [7]

The Second Function of Synthesis was divided into 2 stages:

a. Code Generator, generates object code.
b. Code Optimizer, minimizes results and speeds up the process.

The following figure was the phases of a compilation process[8]:

Figure2.Compilation Process Phase

Quality Compiler depends on 3 factors a.l [9]:

a. Speed & compilation time (quality depends on writing compiler algorithm)
b. Program Quality Objects, determined by the size (produced) and the speed of execution of the object program.
c. Integrated Environment has facilities integrated in the compiler.

The compiler as a compilation machine was divided into 2 types [10]:

a. Compiler
b. Interpreter

Both have the same function, it is as translators, the difference is in the compilation process. If the compiler works when the data and source code were processed at different times, while the source code interpreters and data was processed at the same time.

The following is an illustration of the compilation process and the working interpreter.

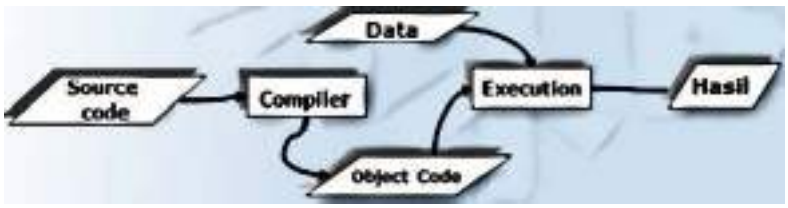

Figure3. Compiler process

Another difference from the interpreter does not produce a form of object code, but the result of the translation is only in an internal form, where the parent program must always exist, different from the compiler.

Figure 4. Interpreter process

2.2. Black Box Testing.

Black box testing was a test which done only observing the results of execution through test data and functional checking of software. It can be analogous to seeing the shape of a black box only; can only see its outer appearance, without knowing what is behind the black box. So, it can be interpreted that black box testing only evaluates on its external appearance (interface), without knowing what actually happens in the detailed process (in short, only knowing inputs and outputs).

So, this Black Box testing method performs software testing on application functionality, does not involve internal structures. Special knowledge of the application code/ internal structure and programming knowledge are generally not needed. Test cases are built based on specifications and requirements, namely the application what should be done. Use an external description of the software, including specifications, requirements, and designs to reduce test cases. This test can be functional or non-functional, although usually functional. The designer tests by selecting valid and invalid inputs and determining the correct output. The picture below shows the scheme of black box testing:

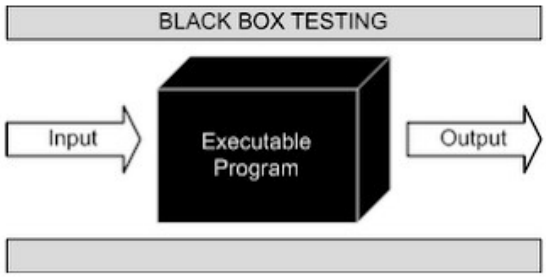

Figure 5. Black Box Testing

Testing on the Black Box will find errors such as:
a. Incorrect or missing functions
b. Interface error
c. Errors in data structures or external database access
d. Performance error
e. Initialization and termination errors

Score limits for testing, including several values, namely
a. Minimum value of input variable
b. Score above minimum score
c. Normal value
d. Score below the maximum score
e. Maximum score

The advantages of black box testing:
a. Program specifications can be identified at the beginning
b. It can be used to assess program consistency
c. Testing is carried out based on specifications
d. There is no need to look at the detailed program code

The disadvantages of black box testing: if the program specifications are made less clear and concise, it will be difficult to make documentation properly [11].

3. Methodology

There are 2 methods used in this study, namely:

a. Descriptive method, where in descriptive writing the results of the study will be written clearly and describe the essence of writing
b. Software Development Method, the method used is SDLC quoted from Sukamto and Shalahuddin's book, in the year of 2013 inYokiFirmansyah[12]. The figure below shows the parts of the SDLC method.

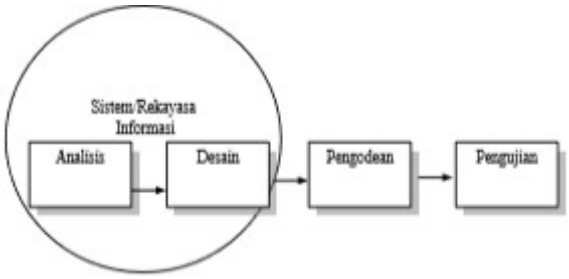

Figure 6. SDLC Method
Source:SukamtodanShalahuddin (2013:29)

While the explanation of the SDLC method in figure 6 above is:

a. Analysis, the analysis phase of system requirements, both hardware and software needed in making applications include:
   1) Hardware: Minimum laptop 4 GB RAM, 100MB ROM space and quad-core 2.2 processorGHz.
   2) Software: Visual Studio 2010 for making the Automatic Lessimic Analyzer interface
b. Design, stages include the interface design for Automatic Lessimic Analyzer using
c. Visual Studio 2010 and some tools consist of labels, textbox, group box and button.
d. Encoding, in the Automatic Lessimic Analyzer application using a programming languageC ++.
e. Testing, the final stage in the SDLC method uses black box testing and evaluates if there are errors in the application that has been made to be repaired.

4. Results and Discussion

4.1.Lexical Analysis (scanner).

Lexical Analysis is identifying all the magnitudes that build a language. In this case the character flow that forms the source program is read from left to right and grouped called tokens, which are character sequences in a single unit that has its own meaning. This analysis of translating inputs into forms is more useful for the next compilation stages. Lexical analysis is the interface between source code and parser analysis. The scanner checks characters per character in the input text, breaking the source of the program into parts called tokens. Lexical Analysis works on grouping character sequences into the main components: identifier, delimiter, operator symbols, numbers, keyword, word noise, blanks, and comments, and so on to produce a Lexical Token that will be used in Syntactic Analysis. Scanners must also be able to identify tokens in full and distinguish keywords and identifiers. For this reason the scanner requires a symbol table. The scanner enters the identifier into the symbol table, enters literal and numeric constants into the symbol table itself after conversion into an internal form.

The main tasks of this lexical analysis include:

a. Read the source code by tracing character by character.
b. Identify lexic quantities (identifiers, keywords, and constants).
c. Transform into a token and determine the type of token.
d. Send tokens.
e. Remove or ignore white-space and comments in the program.
f. Deal with errors.
g. Handle the symbol table.

Example : case studies of source programs from c ++ programming languages
Known: statements in a source program that support arithmetic expressions

Fahrenheit: = 35 + celcius * 1.8 ;

Translation results into tokens in lexical analysis as in table-1.

Table 1: Lexic Magnitude

No	Lexic Magnitude	Source Program	Sum
1	Identifier :		
	- Keyword	-	0
	- Name	Fahrenheit, celcius	2
2	Konstanta	35 , 1.8	2
3	Operator	:= , + , *	3
4	Delimeter	;	1
5	White-space	-	0
		Total	**8**

4.2. Syntax Analyst (parser)

Tree syntax (Tree) is a non-circular connected graph that has one node / node / vertex called Root (root), from which it will have a path to each node. Tree syntax / tree decline / parser tree is useful for describing how to obtain strings by lowering terminal symbols, where each symbol variable will be lowered into a terminal, until nothing has been replaced [13].

A context-free grammar with production rules (symbol initial symbolized S):

$S \rightarrow AB$, read S produces AB orSreduce AB
Requirement :$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$
*The Final Result :**a a b bb***

The parser formation is in the form of a tree as shown in Figure 7 below.

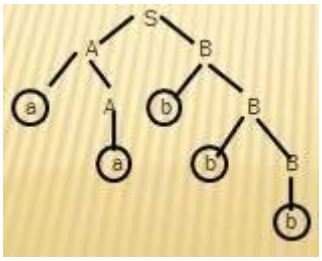

Figure7.Syntax Tree

4.3. Semantic analysis produces the syntax tree generated in the parsing process. The general analysis of semantics is to determine the meaning of the agreement provided in the source program. So in essence, this semantic analysis uses the syntax tree produced by the Parsing process. The syntax of syntactic analysis and semantic analysis is two processes that are very easy to understand and difficult to approve. Semantic analysis, often also combined in thegeneration of intermediate code that produces Intermediate Code output, which will be used in the next compilation process.

There are two kinds of Intermediate Codes, namely: Postfix Notation and N-Tuple :

a, Notasi Postfix.

Syntaxin the form of Arithmetic expressions:

<operan><operan><operator>

Example :

(a+b)*(c+d) →*Postfix :*ab + cd + *

F:=(A+B)*(C+D) →*Postfix :* *AB*+ CD+* F:=

In the form of logic:

Logic Expression (syntax) : IF <exp> THEN <stmtT1> ELSE <stmt2>

PostfixForm :<esp><label1> BZ <stmtT1><label2> BR <stmt2>

b. Notasi N-Tuple

If Postfix each instruction line consists of only one Tuple, while in N-Tuple line consists of several Tuples:

Syntax Notasi N-Tuple is: operator ..... N-1 operand

c. Triple Notation (prefex)

Syntax : <operator ><operand><operand >

Example :

A:=D*C+B/E	Hirarkie Operator :
1. * , D , C	*
2. / , B , E	/
3. + , (1) , (2)	+
4. := , A , (3)	-

d. Qudrapules Notation

Syntax : <operator><operand><operand><result>

The result is a temporary variable that can be placed in the memory or register. Existing problems how to manage temporary variables (results) to a minimum.

Example Instructions: A := D * C + B / E

When the between Code is made:

1. * , D , C , T1 → T1 := D*C

2. / , B , E , T2 → T2:= B/E
3. + , T1 , T2 , A → A := T1+T2

e. Code Generation

The results of the analysis phase will be received by the code generation section. Here the Between Code program is usually translated into assembly language or machine language.

Example: (A + B) * (C + D)

Where is the intermediate code in the form of quadruples:

1. + , A , B , T1 → (A+B) = T1
2. + , C , D , T2 →(C+D) = T2
3. * , T1 , T2 , T3 → (T1*T2) = T3

Can be translated into assembly language with a single accumulator:

LDA A LDA T1
ADD BA+B=T1 MUL T2 T1*T2=T3
STO T1STO T3
LDA C →The LDA command loads the contents of the register/memory.
ADD D C+D=T2 →To the accumulator *(load to accumulator)*
STO T2 →The STO command stores the contents of the accumulator to memory/register (store from accumulator)

f. Error Handling

The form of an error that occurs in the source code in the form of an error message will be displayed in the Automatic Lessimic Analyzer application. The compiler will handle errors from the lexical, syntactic and semantic analysis processes.

g. The results of Design

The user interface of the Automatic Lessimic Analyzer application as in Picture 8 below consist of 8 columns, they are program code, error messages, intermediate code, table identifier, keyword table, data type table, operator table, delimeter table.

The following is the initial display image of the Automatic Lessimic Analyzer application :

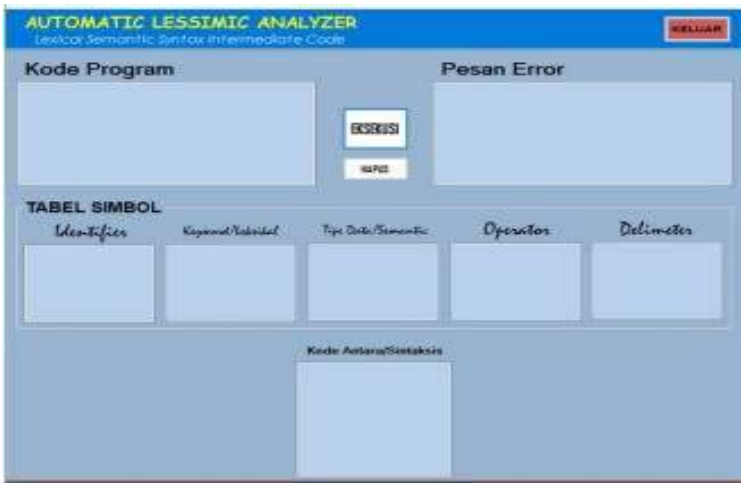

Figure 8. Initial appearance of the AtomaticLessimic Analyzer machine

If the source code is included and performs compiling the program, if there are no errors in writing the program then look like in picture 9 below:

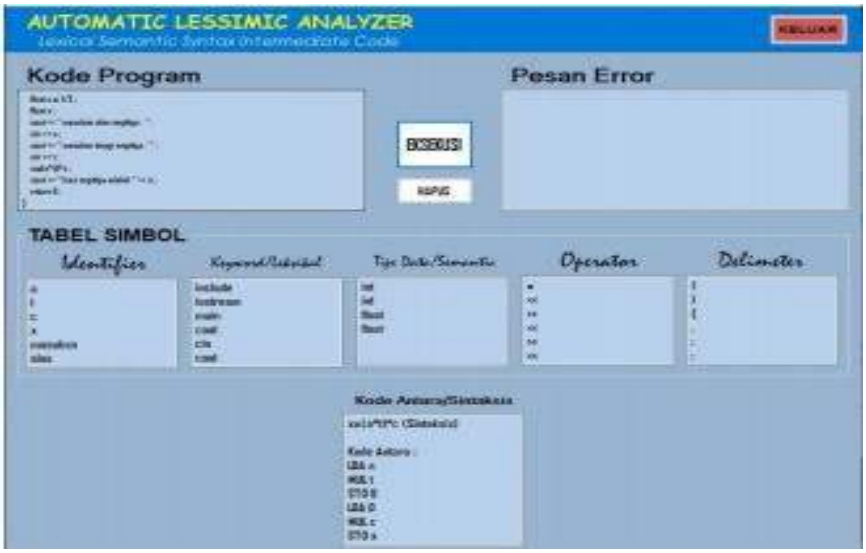

Figure 9.Automatic Lessimic Analyzer Machine Without Error Messages

In Picture 9 it can be seen that the Error Message textbox does not have any contents (blank), meaning that the program code entered by the user has no error. But if the program code entered is not correct, then the textbox in the Error Message will be filled in according to the errors that occur, for example can be seen in Picture 10 below.

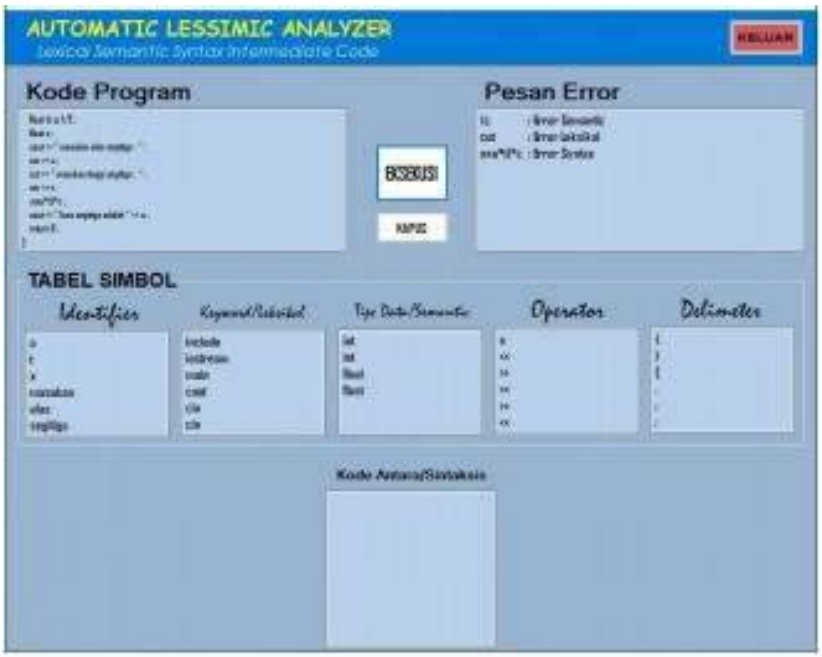

Figure 10. Automatic LESSIMIC Analyzer machine with Error Message

While the results obtained from the application research are explained in Table 2 below.

Table 2. Results of Discussion of Applications with Black Box Testing.

Fase	Source Code	Aplikasi *Automatic LESSIMC Analyser*	Keberhasilan Fungsi Aplikasi

Leksikal (menganalisa bahasa pemrograman)	if (temp2 != "") kumpulankata.Add(temp2);
	for (int i = 0; i < kumpulankata.Count; i++)
	{
	if (_keyword(kumpulankata[i]))
	{ cekkumpulankata.Add(kumpulankata[i] + "" + System.Environment.NewLine);
	}
	}
	if (error2) { }
	else
	{
	for (int i = 0; i < cekkumpulankata.Count; i++)
	{
	result2 += cekkumpulankata[i] + "";
	}
	} tabelkey.Text = result2;
			Berhasil

Sintaksis (menganalisa aritmatika) & Kode Antara (menterjemahkan bahasa sumber, menjadi bahasa target)	if (temp7 != "") kumpulankata.Add(temp7);
	for (int i = 0; i < kumpulankata.Count; i++)
	{
	if (_kodeantara(kumpulankata[i]))
	{ cekkumpulankata.Add(kumpulankata[i] + " " + Environment.NewLine);
	}
	}
	if (error7) { }
	else
	{
	for (int i = 0; i < cekkumpulankata.Count; i++)
	{
	result7 += cekkumpulankata[i] + "";
	}
	} label_antara.Text = result7;
			Berhasil

Semantik (menganalisa variabel)	if (temp6 != "") kumpulankata.Add(temp6);
	for (int i = 0; i < kumpulankata.Count; i++)
	{
	if (_int(kumpulankata[i]))
	{ cekkumpulankata.Add(kumpulankata[i] + "" + System.Environment.NewLine);
	}
	}
	if (error6) { }
	else
	{
	for (int i = 0; i < cekkumpulankata.Count; i++)
	{
	result6 += cekkumpulankata[i] + "";
	}
	} tabelsipe.Text = result6;
			Berhasil

Table 2 explains the discussion of testing (trials) using Automatic Lessimic Analyzer with the Black Box testing process. Black Box testing in the application of Automatic Lessimic Analyzer is a test that is carried out to determine the functions that exist in the application has been running accordingly or not.

5. Conclusion

Based on the final results of the test stated that:

a. With the application of Automatic Lessimic Analyzer it has succeeded in carrying out three analyzes they are lexical, syntax, semantic and intermediate code and error handling with source code using the C++ programming language.
b. It has been successful in using Black Box testing for lexical, syntactic, semantic analysis and intermediate codes and error handling according to the input given.

So, by using the Automatic Lessimic Analyzer application we can know and understand the performance of compiling programs effectively and efficiently.

REFERENCES

[1] Knuth D, 2003,*The Art of Computer Programming*, Fundamental Algoritms Third Edition, Volume 1, Addison-Wesley, United States, 0-201-03801-3
[2] FirrarUtdirartatmo, 2005, *TeknikKompilasi*, GrahaIlmu, Yogyakarta,Isbn 979-756-058-1, hal xiv+188
[3] AndezApriansyah. A, dll., 2019, *DesainMesin Compiler untukPenganalisaLeksikal, Sintaksis, Semantik, KodeAntaradan Error Handling PadaBahasaPemrogramanSederhana*, Journal of Applied Informatics and Computing (JAIC) Vol.3, No.1, pp. 01~07 e-ISSN: 2548-6861.
[4] Utdirartatmo, F., 2011, *TeknikKompilasi*, J & J Learning Jogjakarta
[5] RinaldiMunir, 2015, *TeoriKomputasi IF5110*,Bandung : ITB,.
[6]M. SidiMustaqbal, dll, *PengujianAplikasiMenggunakan Black Box Testing*,Boundary Value Analysis, ISSN : 2407 – 3911
[7] YokiFirmansyah&Udi,2018, *PenerapanMetode SDLC Waterfall DalamPembuatanSistemInformasiAkademikBerbasis Web StudiKasusPondokPesantren Al-HabiSholehKabupatenKubu Raya, Kalimantan Barat*, JurnalTeknologi&ManajemenInformatika, Vol. 4 No.1
[8] J. Suciadi, 2001, *StudiAnalisisMetode-Metode Parsing danInterpretasiSemantikPada Natural Language Processing*, J. Inform., vol. 2, pp. 13–22,.