COMPILER FOR INDONESIAN PROGRAMMING LANGUAGE

Gun Gun Febrianza Amd.Kom[1], Ken Kinanti Purnamasari S.Kom., M.T[2]

[1]Informatic Engineering – Indonesia Computer University
Jl. Dipatiukur 112-114 Bandung
E-mail : gungunfebrianza@gmail.com

ABSTRAK

Programming language used by humans in order to control the computer, but basicly computer only understand machine language. So that a high-level programming language created to simplify computer programming. The simpler programming language the higher abstraction and the details that hidden inside. High-level programming language can still be classified, there are english based programming language and non-english based programming language.

In this research Indonesian Programming language were made to simplify the programming language that is easily to understood, remembered and studied by those who understand the native language in Indonesia. So we create compiler for Indonesian language programming with object oriented paradigm. Object oriented programming paradigm used to solves some problem from previous programming paradigm.

The compiler itself using two stage translation, using an execution engine so the produced program from compiler can work properly across all the operating system like linux or windows 32 or 64 bit.

Keywords : *High Level Language, Compiler, Indonesian Programming Language, Object Oriented Programming*

1. INTRODUCTION

Programming language is a notation to describe a computation for humans and machine computer [1]. There are a wide variety of programming languages which can be classified, some of them can be classified based on the language of abstraction ranging from low to high level, there is a programming language based on interpreted programming language and compiled programming language, in terms of language there is english based programming language and non-english based programming language, based on programming paradigm there is exist object-oriented programming paradigm, imperative programming, declarative programming and so on. To create a compiler that can translate Indonesian programming language with object-oriented programming style, the necessary techniques for lexical analysis, syntactic analysis, semantic analysis and code generation in particular can be used to deal with these specifications. So in this study we developed compilers for indonesian programming language called PBI (Pemrograman berbahasa Indonesia) for general purpose programming language.

Other compelling reason to make Indonesian language as programming language because Indonesia is the fourth largest population in the world. So it has a huge potential to get development, contribute and utilization of Indonesian-programming language for various studies.

This paper is divided into several sections, in section 2 explained the background of this paper, some research on native-language programming in various places in the world and the development of native-language programming in Indonesia itself, becomes a main driver of this research are made. In Section 3 described the compilation process of PBI, in section 4, the method for doing lexical analysis, in section 5 described a method for analyzing the syntax and optimization, in section 6 described methods to perform semantic analysis and code generation, in section 7 described execution engine that is used and in section 8 are the conclusions for future research.

2. CONTENT OF RESEARCH

2.1 Backgrounds

The entire programming language that globalized today is dominated using english language due to historical and linguistic factors. In terms of historical factors in the computer world, english is the lingua franca (common language) because the character encoding was first designed for the English language is ASCII in 1960 [2]. In terms of the linguistic, english character are short and does not have diacritical marks (accent characters). Even so, non-English-based programming language exist (Lexico in Spanish, Aheui in Korean, Sako in Polish and Rapira in Russian [3]). In Indonesia research the use

of Indonesian as the programming language has been done. Among of them there are KILANG language in the book about the creation of an Indonesian-language interpreter entitled KILANG 002: BASIC dalam Bahasa Indonesia, published in 1984 [5], then language ALIN for the expression of Linear Algebra in the book Tuntunan praktis merekayasa interpreter, published in 1995[6] and BAIK (Bahasa Anak Indonesia Untuk Komputer), first release in 2008[7]. To translate from high level language, yet KILANG, ALIN and BAIK using program called interpreter. So far in Indonesia there is not yet available compiler that are specifically used to translate Indonesian language programming. Compilers to translate Indonesian Programming Language to be able produce programs and libraries that can be used simultaneously. An executable used so the programs that created from computer language can be directly executed and the dynamic link library is the library that created with computer language. As the contents of library, class and method inside can be used by other programs for interoperability. Furthermore, the output executable or dynamic link library can be produced and used in the multi-operating system without having to use another translator.

2.2 Compilation Process

When compilation started there are stages to be performed by several programs in the compiler, the first program is Lexical Analyzer for producing token, the second program is Syntax Analyzer to perform parsing on sequence tokens, then the process of Tree Traversal using program Semantic Analyzer to explore data tree representation or a list of data results from parsing to build a symbol table. Symbol table used for Semantic Analysis with Semantic Analyzer program in order to perform Code Generation. The target language that will be produced is IL Assembly or CIL (Common Intermediate Language).

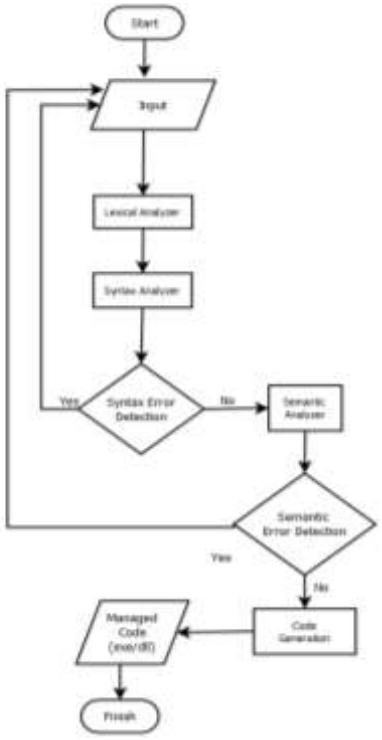


Figure 1. Compilation Process

CIL or Common Intermediate Language is the term used European Computer Manufacturers Association (ECMA) to create an international standard in the CLI (Common Language Infrastructure). IL Assembly has the same instruction which is also owned by the native assembly language, for example if you want to perform addition operation then use the add instruction, to perform subtraction using sub instruction, and IL Assembly also has the same instruction set as the native assembly language in general.

Table 1. Sample Mnemonic and Opcode IL

Mnemonic	***Opcode***	***Purpose***
Add	0x58	Adding two items on evaluation stack
Add.ovf	0xd6	
Add.ovf.un	0xd7	

Instructions Add.ovf has the ability to detect overflow and Add.ovf.un is also equal but only detect overflow for unsigned data types. Compilers at PBI using a two-stage translations, generated IL Assembly in code generation will be translated by IL Assembler (ILASM) to generate managed code in either form as an executable or dynamic link library. Below is a list of two-stage translation that exist today and compilers for PBI using CIL Generator.

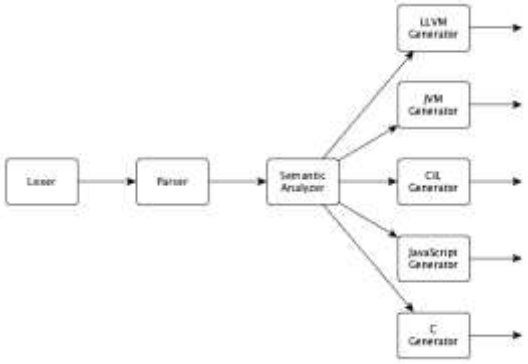


Figure 2. Two Stage Translation

Furthermore, managed code will be executed on the execution machine Common Language Runtime (CLR).

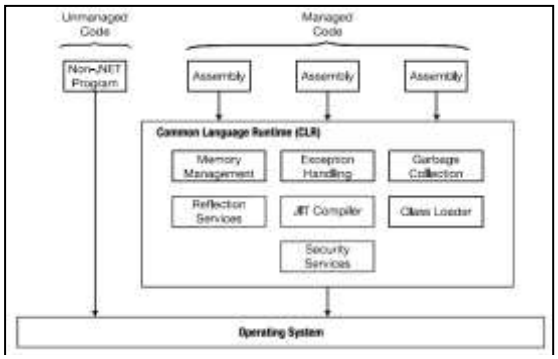


Figure 3. Managed Code Execution Inside CLR

Inside Common Language Runtime exist Just-In Time compiler that will perform JIT Compilation, managed code will be translated into machine language that can be executed by the CPU.

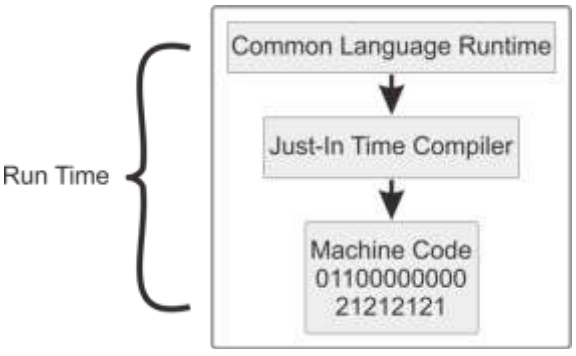


Figure 4. Runtime Execution under CLR

Modern-day compiler use Two-stage Translation to face the challenges of difference Assembly language on each processor, each Architectures have different assembly language for example at Intelx86 Architectures and ARM Assembly Language both is different.

2.3 Lexical Analysis

Lexical Analyzer is one of the programs in the compiler used to perform lexical analysis in the first phase of compilation. Lexical Analyzer will accept input source code in the form of character stream. Next, character stream will be processed to produce a token, each token has a pattern which will produce various lexemes to generate a keyword, identifier, number, float, string, and so on.

Table 2. Token and Lexeme

Token	***Lexeme***
Kata Kunci	bulat32, bulat64, pecahan, kalimat, logika, kosong, utama
Pengena;	*Variable1, var, var_1*
Tititk Koma	;
Kurung Kiri	(
Kurung Kanan	)

Lexical analysis methods using Deterministic Finite Automata. A DFA consists of 5 tuple M = (Q, Σ,δ, q , F)
Q = Finite set from"internal state"
Σ = Finite set of symbol called as "input alphabet"
δ = lowercase delta as notation for transition fuction
q0 ∈ Q = q0 element from Q which become start state
F ⊆ Q = F single element or subset from set Q for final state

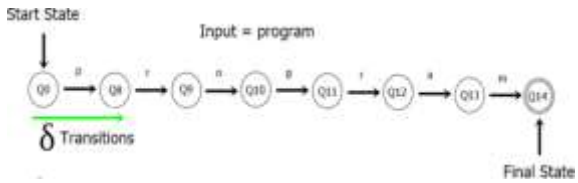


Figure 5. DFA Components

If the input is string program, the path will come to the state Q14 which becomes the final state., because every transition that occurs is a subset transtitions known by DFA engine.

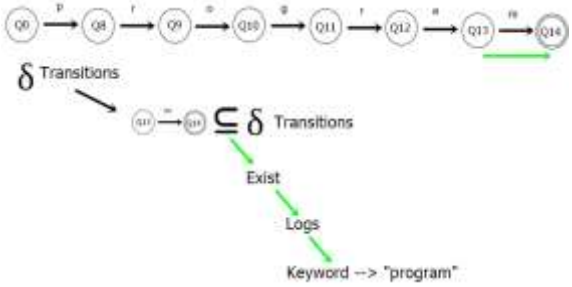


Figure 6. Input reach the final state

Every final state will provide input that will be a token some type attribute, the available attributes for token is comments, keywords, identifier, string, integer, float, or symbol.

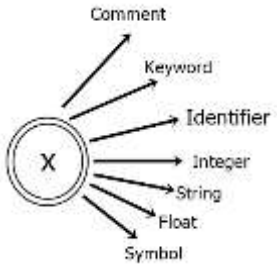


Figure 7. Final State give an attribute

Each token will have some properties which are:
1. Token ID
2. Token Name
3. Token Types
4. Token Index
5. Token Line

If input string fail to reach the final state, the input fails identified and given attribute as an identifier.

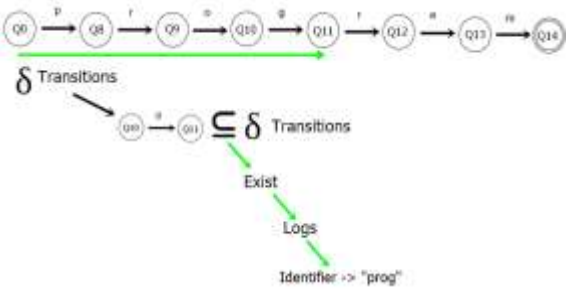


Figure 8. Input fail to reach the final state

2.4 Syntax Analysis

Syntax Analyzer is a program in the compiler used to perform syntax analysis. Syntax Analyzer receiving series of token as the output from the lexical analyzer. Syntax Analysis is often called parsing [7]. Furthermore, a series of tokens will syntactically tested, with matching grammar that has been established to build a structured hierarchy called the syntax tree or list data. Syntax analysis method uses a mathematical model of set theory and borrow some terms in regular expression.

Inside PBI exist term set of orders, a series of syntactic structures that represented in the form of data numeric inside a set. example:

<Declaration Program> = {1,2,3} (4.1)

Data numerics 1,2,3 is a member of the set <Declaration Program>, each numeric data represents a token. For example, number 1 is a representation of the terminal 'program', number 2 is a representation of a terminal identifier (ID) and the number 3 is a left curly terminal (left brace). If the enumeration is done then:

<Declaration Program> = {1,2,3}
<Declaration Program> =
{'program', '([_]?[a-zA-Z] [0-9])', '{'}

Identifier is a kind of token that can generate different lexeme, to simplified this complexity we need Regular Expression notation for identifier.

Inside PBI there also term Sequence of Order, an arrangement token in predetermined position. Each set of orders have sequence of orders that will be used to syntax analysis. On the previous Set of Order in advance if transformed into a sequence of order, then:

1 → 2 → 3 (*Correct Order*) (4.2)

In above notation parser will start syntactic analysis in number 1 and parsing flow is from left to the right on sequence of order. Furthermore, the parser will read the sequence of orders one by one to test sequences are correct.

Read Input Buffer → Parse Current Token (Target Order) → Check Correct Order

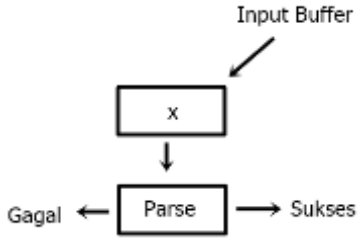


Figure 9. *Parsing Sequence of Order*

For error recovery in PBI there are two types of errors that will occur if we use Set of Order:

1. Error because the token in the Input Buffer still members of the sequence of order.
If Input Buffer = {3,2,3} then the parsing process that occurs is as follows:
Parse token 3 failed! Solution needed token 1.

2. Error due to the token in the Input Buffer is not a member sequence of order.
If Input Buffer = {1,4,3} then the parsing process that occurs is as follows:
Parse token 1! Success!
Parse token 4 failed! Solution 2 tokens required!

PBI also have Alternative Parsing there is optional correct order and multi correct order and the correct order. In sequence of order below umbers 1 optional correct order:

1? → 2 → 3 (4.3)

Optional correct order marked with the notation '?' So that the token is present or not in the sequence of order, the parser will not give syntax error. Multi Correct Order characterized by a set of data numeric inside [n l .. n], where n is a token and 'l' symbol of separation between other tokens. A token is in multi correct order one of them could be accepted by the

parser. So if there is a sequence of orders like this then:

1? → 2 → 3 → [7 l 8 l 9] → 4 → 5 → 6 (4.4)

If the token 1 is not exist then Error Recovery will not happen in the optional correct order. If the token 7, 8 or 9 appears after 3 token then Error Recovery will not occur in multi correct order, but if one of multi correct order is not exist error recovery will occur with multi solution. Solutions are needed token 7, 8 or 9.

Every parser passes a set of orders for example, from set of order program declaration to the class declaration then in a hierarchical set of terminal and non-terminal will establish data representation of a tree or a list of data that will be used for Semantic Analysis.

2.5 Semantic Analysis

Semantic Analyzer is a program in the compiler used to perform semantic analysis. Semantic Analyzer will perform error check where the error is the semantic type of error that can not be detected by the syntax analyzer. For example if there is a variable with the same identifier it can not be detected by the syntax analyzer but is able to be detected by a semantic analyzer with the help of the symbol table. At PBI, semantic analyzer is the phase to analyze error code that can not be done by syntax analyzer before producing IL Assembly. Semantic analysis performed first so the source code to be translated must free of errors. Below is the entire parse tree consisting of a set of terminal and non-terminal nodes that produced from syntax analyzer to perform tree traversal.

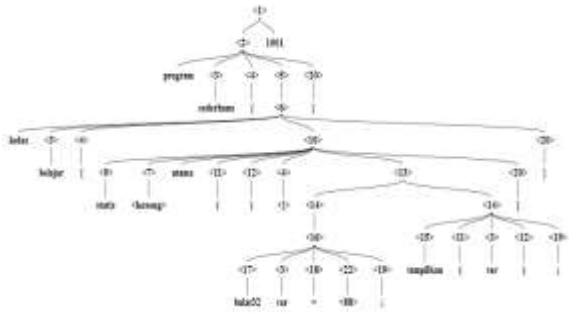


Figure 10. Tree Representation

An abstract representation of the non-terminal node is marked with node id inisde left and the right bracket and terminal representations can be seen directly.

Table 3. *Node ID* dan *Abstraction*

***Node* ID**	***Abstraction***
<1>	<Program>
<2>	<Deklarasi Program>
<3>	<Pengenal>
<4>	<Kurawal Kiri>
<5>	<Kelas>
<20>	<Kurawal Kanan>
<6>	<Deklarasi Kelas>
<10>	<Metode Utama>
<9>	<Nilai Balik>
<7>	<Ciri>
<11>	<Kurung Kiri>
<12>	<Kurung Kanan>
<13>	<Spernyataan>
<14>	<Pernyataan>
<16>	<Deklarasi Variabel>
<17>	<Tipe Primitif>
<18>	<Sama Dengan>
<22>	<Bilangan Bulat>
<19>	<Titik Koma>
<15>	<Metode Masukan Keluaran>
<11>	<Kurung Kiri>

Semantic analysis process starts from the root to get the information that will be used to transmit information to the symbol table, so that the process of deeper semantic analysis can be done before the compiler process code generation.

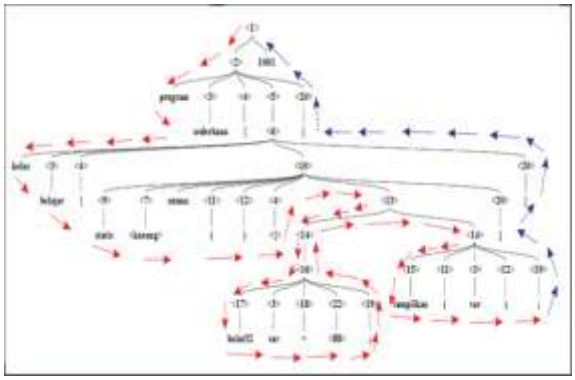


Figure 11. Semantic Analysis Process

In the image above the tree traversal performed depth-first from left to right starting from the root at the top, marked with a red arrow direction. The first Rule Node is <program> flow into the rule node of <Declaration Program> and so on until every terminal or leaves (set of tokens in the parse tree) reached and return again to the root. The process from bottom to top is marked with a blue arrow direction. When the tree traversal access a particular node in the data tree, there is a process by which information is given to the symbol table while the process of semantic analysis and code generation.

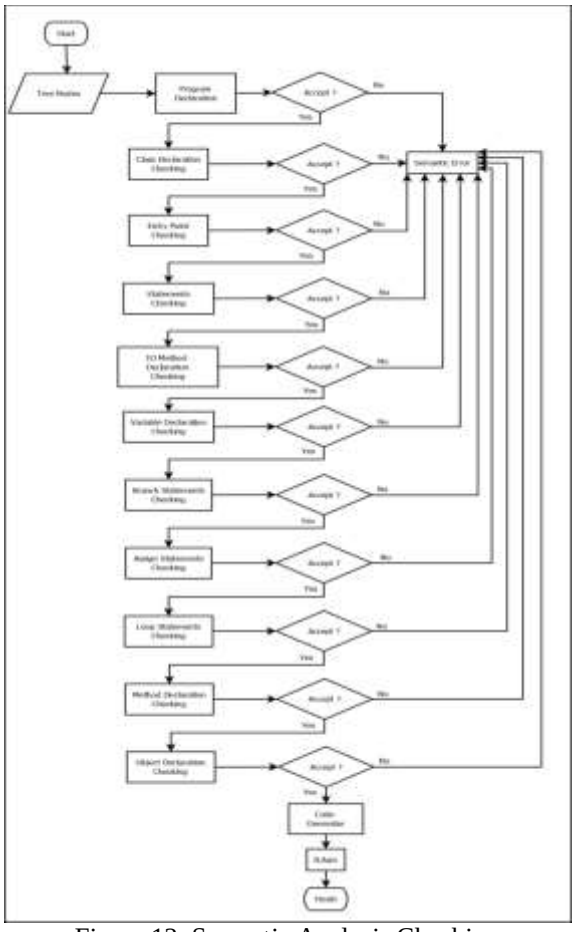


Figure 12. Semantic Analysis Checking

Semantic Analysis performed so each rule node is free of semantic rerrors when the tree traversal process to produce the Common Intermediate Language (CIL).

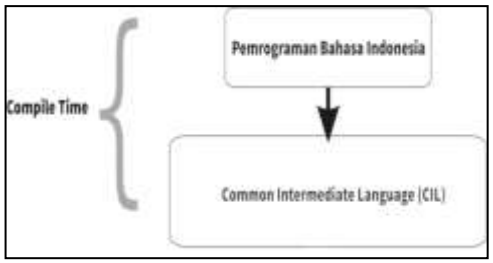


Figure 13. Code Generation Process

2.6 Execution Engine

Common Language Runtime is the heart of the .Net Framework. CLR is an execution environment or execution engine that is used to load and execute code. A code that runs on the CLR is called managed code and a code created specifically for win32 executables and dynamic link libraries (DLLs) referred to unmanaged code.Compared to unmanaged code, the managed code has the advantage of automatic memory management, type safety, security, and the opportunity to run on other platforms are supported by the CLR.

Additionally Common Language Runtime also provides a variety of services including:

1. Loading and executing a program or application.
2. Memory isolation on a program or application.
3. Verification of type safety.
4. Translating IL Assembly into native code.
5. Provide Metadata.
6. Management of exception and error.

In the CLI (Common Language Infrastructure) the results of assembled IL assembly named Assembly. Inside every Assembly exist Metadata and IL Code. A CLR-compliant compiler will accept more than one input to produce an assembly.

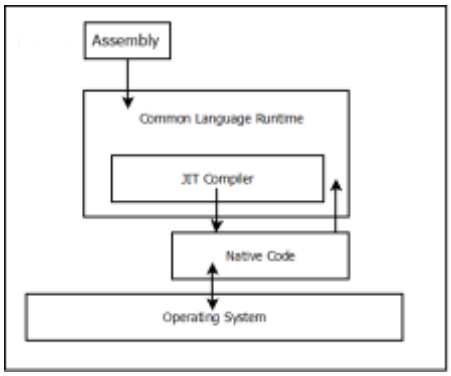


Figure 14. CLR Execution Model

Terminology Assembly itself in the Common Language Infrastructure is includes managed code, icon and the resources that used. Furthermore, when an assembly is executed JIT Compilation is performed by the JIT Compiler. High-level language from Indonesian programming language after the two-stage translation the output called Assembly either in the form of executable or dynamic link library

3. CLOSING

Designed Compiler for PBI from previous methods :



Figure 15. GUI *Code Editor*

Tampilan *token* hasil Analisis Leksikal :

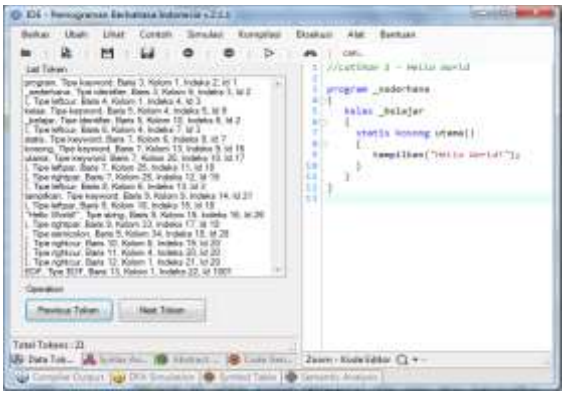


Figure 16. List Token

Below user interface is Data Tree from Parsing :

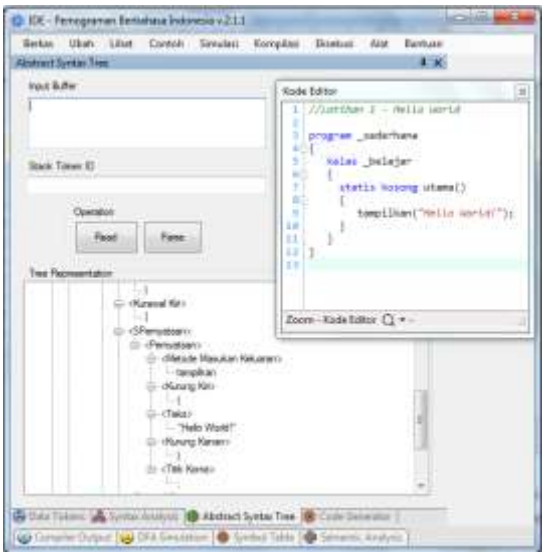


Figure 17. Data Tree from syntax analysis

Below user interface is history of tree traversal from semantic analysis.

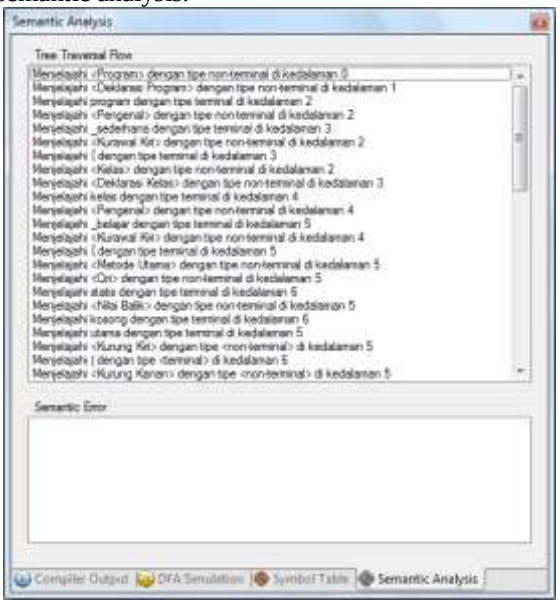


Figure 18. Tree Traversal History

Below user interface is produced code generation:



Figure 19. Code Generation Result

Below user interface is output from executable :

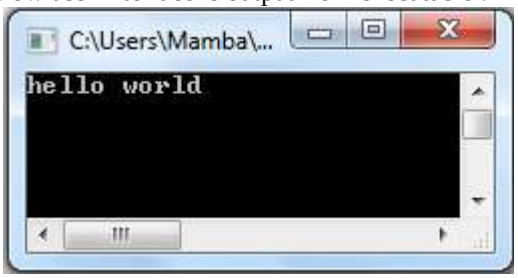


Figure 20. Executable Executed

3.1 Conclusion

Based on the test results in 50 variations of the source code for PBI (Pemrograman Berbahasa Indonesia) the process of translating high-level programming language based on indonesian language success achieved. PBI also success to produce output executable and dynamic link library using high-level programming language based on Indonesian language.

3.2 Advice

There is some suggestion that this research could be better in the future, which are:

1. Develop IDE Compiler for Indonesia Programming Language that is able to run on Linux and Macintosh operating systems.
2. Develop IDE Compiler for Indonesia Programming language that supports applications based GUI (Graphic Unit Interface).

BIBLIOGRAPHY

[1] Alfred, Aho "Introduction", *Compiler Principles, Techniques, & Tools*, : Pearson, 2006. pp 1

[2] Fraser, John "Assesment of The English Language", "*English : The Prototypical World Language for The Twenty First Century*", : Lulu, 2009. pp 107

[3] Shampa Banik, Md Monjurul Islam, Md Azher Uddin, "New Approach to Develop Bilingual Compiler", *Journal of Engineering Research and Applications*,. pp 257-261, February, 2014.

[4] Dali S. Naga, "Kilang 002 : Basic Dalam Bahasa Indonesia," Perpusnas.go.id, 1984 [Online]. Tersedia : http://onesearch.perpusnas.go.id/Record/IOS1-INLIS000000000029667 [Diakses : 31 Januari 2017]

[5] Dali S. Naga, "Kilang 002 : Basic Dalam Bahasa Indonesia," Perpusnas.go.id, 1984 [Online]. Tersedia : http://onesearch.perpusnas.go.id/Record/IOS1-INLIS000000000029667 [Diakses : 31 Januari 2017]

[6] Hassanudin, Haris, "BAIK (Bahasa Anak Indonesia Untuk Komputer) - Programming Language Based on Indonesian Lexical Parsing For Multitier Web Development", Jurnal Ilmu Komputer dan Informasi, Volume 3, pp 47

[7] Puntambekar. "Overview of Compilation," *Compiler Design*. India : Technical Publication Pune. 2009. pp 4