ERMIAS TEFERA DESIGN OF AMHARIC PROGRAMMING LANGUAGE WITH A PROTOTYPE OF TOKENIZER AND PARSER FOR SELECTED CONSTRUCTS NEU ALAMREW 2019

DESIGN OF AMHARIC PROGRAMMING LANGUAGE WITH A PROTOTYPE OF TOKENIZER AND PARSER FOR SELECTED CONSTRUCTS

A THESIS SUBMITED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
ERMIAS TEFERA

In Partial Fulfilment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2019

DESIGN OF AMHARIC PROGRAMMING LANGUAGE WITH A PROTOTYPE OF TOKENIZER AND PARSER FOR SELECTED CONSTRUCTS

A THESIS SUBMITED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
ERMIAS TEFERA ALAMREW

In Partial Fulfilment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2019

ERMIAS TEFERA ALAMREW: DESIGN OF AMHARIC PROGRAMMING LANGUAGE WITH A PROTOTYPE OF TOKENIZER AND PARSER FOR SELECTED CONSTRUCTS

Approval of Director of Graduate School of Applied Science

Prof. Dr. Nadire ÇAVUŞ

We certify this thesis is satisfactory for the award of the degree of Master of Science in Software Engineering

Examining Committee in Charge:

Assoc. Prof. Dr. Yöney Kırsal EVER	Department of Software Engineering, NEU
Asst. Prof. Dr. Boran Şekeroğlu	Department of Information Systems Engineering, NEU
Assoc. Prof. Dr. Kamil Dimililer	Supervisor, Department of Automotive Engineering, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conducts. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Ermias T. Alamrew

Signature:

Date:

ACKNOWLEDGEMENTS

First off, I would like to extend my deepest gratitude to my thesis supervisor Assoc. Prof. Dr. Kamil Dimililer, for his continuous and unwavering support, for his patience, motivation and immense knowledge. I would also like to thank him for devoting his valuable time to read and understand the core idea of the thesis at each and every stage of the process and provided me with constructive and insightful comments which helped realize the thesis. I couldn't have Imagined having a better supervisor.

I would also like to thank my Academic advisor Assist. Prof. Dr. Boran Şekeroğlu who has always been there whenever I need him, who prepared me for this time.

Last but not least I would like to thank my parents who has been in every step of the way on my academic journey. I would also like to give enormous gratitude to my big sister Askale Mariam Tefera without her support and motivation, I wouldn't have gotten to this stage of my life. This thesis and my life's work wouldn't have been possible without you, Thank you.

To My Parents and My sister…

ABSTRACT

Programming language is an important course in computer science and other computer-related fields. A programming language can be employed in different sectors to program a system that helps or replaces humans. Computer programming is very important that different countries are teaching it from an early age. But this is difficult for non-English speaking countries because most of the Programming language available is based on the English language. That is even if languages allow programmers to write identifiers using their native language all other important parts of the language are based on the English language this include the keywords, the error generated by the compiler and all libraries and their documentation. This made teaching Programming from a young age in non-English speaking countries very difficult. This is also the case in Ethiopia. In Ethiopia, there is no a single programming language that uses the Amharic language. The fact that there is no Amharic programming language to programmer prevented schools to teach students programming from a young age and to develop the native language. There is very little done on the Amharic language on technology related areas. This thesis focuses on designing a programming language that is based on the Amharic language that will help solve the aforementioned problems. This thesis presents the design of an Amharic language's grammar by using EBNF (Extended Backus-Naur form) which is used to write context-free grammars. The EBNF is used to write both the parser and lexer part of the grammar.The thesis also includes the generation of the parser and the lexer by using an automatic language translation tool called ANTLR. The grammars legality, the presence of unwanted production rules and the efficiency of the parser has been tested using ANTLR. The thesis also presents a sample debugger which is written by using Java and is used to evaluate the grammar.

Keywords: Amharic Programming language; Context-free grammar; parser; lexer; Debugger; Extended Backus-Naur Forms

ÖZET

Programlama dili, bilgisayar bilimleri ve diğer bilgisayarla ilgili alanlarda önemli bir derstir. Bir programlama dili, farklı sektörlerde insanlara yardım eden veya onun yerine geçen bir sistemi programlamak için kullanılabilir. Bilgisayar programlama, farklı ülkelerin erken yaşlardan itibaren öğretmeleri için çok önemlidir. Ancak bu İngilizce konuşamayan ülkeler için zordur, çünkü mevcut Programlama dilinin çoğu İngilizce diline dayanmaktadır. Bu, diller programcıların ana dillerini kullanarak tanımlayıcılar yazmasına izin verse bile dilin diğer tüm önemli bölümleri İngilizce diline dayanmaktadır, bunlar arasında anahtar kelimeler, derleyici tarafından oluşturulan hata ve tüm kütüphaneler ve bunların belgeleri bulunur. Bu, İngilizce konuşamayan ülkelerde genç yaştan itibaren Programlamayı çok zorlaştırdı. Etiyopya'da da durum böyle. Etiyopya'da, Amharca dilini kullanan tek bir programlama dili yoktur. Programcı için Amharca programlama dili olmaması, okulların genç yaşta programlama öğrencilere ders vermesini ve anadili geliştirmesini engelledi. Amharca dilinde teknoloji ile ilgili alanlarda çok az şey var. Bu tez, yukarıda belirtilen problemleri çözmeye yardımcı olacak Amharca diline dayanan bir programlama dili tasarlamaya odaklanmaktadır. Bu tez, bağlamsız gramer yazmak için kullanılan EBNF (Genişletilmiş Backus-Naur formu) kullanılarak bir Amharca dilinin gramerinin tasarımını sunmaktadır. Tez aynı zamanda ANTLR adlı bir otomatik dil çeviri aracını kullanarak çözümleyici ve sözcü oluşturulmasını da içermektedir. Tez ayrıca, Java kullanılarak yazılmış ve dilbilgisini değerlendirmek için kullanılan örnek bir hata ayıklayıcıyı sunar.

***Anahtar Kelimeler*:** Amharca Programlama dili; Bağlamsız gramer; ayrıştırıcı; lexer; Debugger; Genişletilmiş Backus-Naur Formları

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABRIVATIONS

ANTLR:	Another Tool for Language Recognition
ICT:	Information Communication Technology
EBNF:	Extended Backus-Naur Form
BNF:	Backus-Naur Form
GUI:	Graphical User Interface
IDE:	Integrated Development Environment
LHS:	Left Hand Side
RHS:	Right Hand Side
BCPL:	Basic combined programming language
ANSI:	American National Standards Institute

CHAPTER 1
INTRODUCTION

Information and communication technologies(ICT) are a core element of the knowledge based-society (Papaioannou & Dimelis, 2007). The use of computers and other computer-aided devices are growing rapidly (Kirsal Ever & Dimililer, 2018). Engaging in information technology in an extensive manner undoubtedly in Ethiopia was long overdue. A countries security and development depend highly on the government's willingness to invest vastly in the technology sector. Countries all around the world have been raising their investment in ICT to be competent and stay at the top of the technological advancement of the 21st century. This has been done in countries like America and some members of the European Union like England (Papaioannou & Dimelis, 2007).

Information technology has helped developed countries to improve their production efficiency that has already been improved thanks to the industrial revolution. Even though the technology is not the answer to every problem society faces but it can be used to solve a considerable amount of the problem they are facing (Bekele, 2001).

Unfortunately, Ethiopia has done very little to advance ICT and its use (Bekele, 2001). Even if information technology has been in Ethiopian curriculum for preparatory schools for a long time the ICT sector has not shown a tangible change. This is mostly because the topics included in the curriculum are rudimentary even for lower graders. This age group in most countries is the time students engage in developing and implementing systems that will solve the problems society is facing. This hasn't been the case in Ethiopia for quite a long period of time

It has been observed that the reason for the governments' decision to not start teaching students basics of computer and programming is the lack of resources that have been localized in the language's students understand from an early age. Ethiopia has as a country been facing a big problem to localize technologically related material so that it can be understood and used by many in the country. Many of the technologically related material in the country are in a language that is not the countries official language which is Amharic

(አማርኛ) either in English or Chinese, this has made the life of the people unbearable when it comes to technology and made it hard for students to be competent and contribute to their country in computer-related areas and has been pulling the country backward.

Amharic(አማርኛ) is the working language of the government of Ethiopia and some of the other Regional states. The Language is widely used in the Amhara regional state and the capital of Ethiopia, Addis Ababa (Asfawwesen, 2016). Next to Arabic Amharic is the second most spoken Semitic Language in the world (Gezmu et al., 2018). Amharic has 33 basic characters, each of which has seven forms depending on which vowel to be pronounced. Which makes the number of characters in the language 231 without the numbers and the punctuation marks (Cowell & Hussain, 2003)

In the recent curriculum reform in Ethiopia, the government brought the information technology course from preparatory school to high school. That has helped students to be introduced to technology at least before students start their university preparation school. But from other countries point of view, this is far from enough.

Many countries all around the world have been introducing or finishing up preparation to introduction computer programing to lower grades. Many countries in Europe like England and Italia has finished their preparation to introduce computer programing to lower grade students of ranging from the age of 5-16. This student will be taught algorithm, coding and debugging depending on their age.

Introducing students to computer programing at an early age will help students to be familiar with advanced staffs when it comes to software development and artificial intelligence. Bringing computer programming in lower grades in Ethiopia is very difficult, the reason behind it is Ethiopia's official language is different from the language used in technological-related materials and software's. Almost all computer programming related materials and resources are written either in English or in a language other than Amharic. So, to introduce computer programming the government and researchers in the country has to do more to localize and make them suitable to students of early age who only speak a language that they understand and is their first language, which is Amharic.

This thesis will be focusing on addressing this issue which has been looming in the country for quite a long period of time. The thesis will be focusing on designing a computer

programming language, that uses the official language spoken in Ethiopia as a basis for the keywords. The syntax and grammar of the language will be designed based on the Amharic(አማርኛ) Language.

1.1 Background of the Study

The focus of this thesis is to design a programming language that uses keywords based on Amharic which is the working language of the Ethiopian government. This thesis will try to close the gap in technological advancement and the Amharic language. The design of the language will also help students to programmer by using the language they understand well and are using in their day to day activity.

Studies have indicated that students who learn any subject area by using their mother tongue show a far better performance than students who learn by using non-native language. Walter, (2012) in his study indicated that the use of mother tongue as a medium of instruction increases the level of outcome. The other study conducted learning to code in a localized programming language by Dasgupta & Hill, (2017) also showed that students who program in localized language shows better-understanding programming features and develop new programming concepts. So, this study will focus on designing programming that will help students in Ethiopia learn to program in Amharic.

A programming language specific to a particular language has been an area of study long ago. Countries like Russia, China, and France have a programming language developed based on the respective countries working language. But in Ethiopia, there are not any programming language developed using Amharic. There is a language named AxumLight which is developed by Ethiocloud according to their official website but it is not accessible for download either for free or premium.

When it comes to the design of non-English programming language there are two suggested ways to go about it. The first one is a direct translation of an already existing programming language. An example of translate language is Chinese python which is a Python 2.1.3 programming language in the Chinese language. The second and obvious way of design is to design a new programming language. This thesis follows the second way of programming language design

1.2 Statement of the Problem

There is an obvious lack of resource that is localized in Amharic, Ethiopians working language. And this has been a great impair for a long time in the technology industry. Ethiopia has been at the back of the technological advancement that has been happening all around the world. We haven't contributed to the innovation and we didn't also engage in the localization of the new technologies.

As stated in the previous section programming has been employed extensively in almost every sector imaginable to make it better, easy and productive than it was before. Programming has helped reduce the effort we dispense to do a task by letting the machine do it for us.

In Ethiopia there, a lack of technological tools customized into the local language. Almost everything that is out there has to come as it is and be used by professionals and those who attended school.

Mostly the problem lays on the lack of programming language in the local language. One-third of the programming language in use is designed by using English as the base for the keywords used in the language's syntax. This has prevented countries whose working language is other than English and their children don't speak English at an early age from teaching programming language in lower grades.

According to a study on how well students respond to learning programming in their mother tongue, students who have been taught to program using a programming language where the keywords are constructed by using their mother tongue has shown far better achievement and has developed new programming concepts in quite a short time than those students who start programming with a programming language that have English as a base for the keywords used in the syntax (Dasgupta & Hill, 2017). So not teaching students computer coding in their own language is not effective but not teaching them at all is problematic.

The lack programming language has also contributed to the lack of software's developed by programmers who are from Ethiopia who understand the society and the culture better than anyone and also contributed to the lack of software that takes input and shows output in the working language.

The problem with programing language written in a foreign language is not just the fact that the keywords used in the language are not understandable it is that the language used to write the documentation of the language, the errors displayed when compile time or run time errors happen, and almost any other related materials are in a language that students don't understand.

There is a way to solve this problem at least halfway, that is to translate the materials in a language student that they understand and change a programming language word by word in to the local language, but that will hinder the student's performance because the mere translation of the words in programming language will result in the generation of words that are unrelated to the things they do.

Many countries have either modified already existing programming languages developed elsewhere into the language that is more suited to students learning in the country or developed a new programming language taking the goods from different programming languages. These are some of the countries that have a programming language in their working language, Russia, China, France, Arab countries and others.

1.3 Motivation of the Study

It is observed that many of the children in school don't have access to computers even when they do there is no way that they can relate to it as easily as they do if it was in a language they understand. Solving this problem to some extent is the first reason for this thesis. The lack of material and resource in the local language to teach students the basics of computer and computer programming is the other reason to start designing a programming language in Ethiopia's official working language, Amharic. Because there is great benefit teaching students about the basics of computer and coding using the language they are already used to and easily understand I chose to undertake research about the design of programming language with Amharic. This research when implemented in the future will help students in an early age learn computer programming without the need to learn the English language to understand what the keywords mean or to understand the documentation and figure out the errors that occur during programming and fix it.

1.4 Thesis Objective

1.4.1 General objective

The General objective of the thesis is to design a programming language that uses Amharic keywords.

1.4.2 Specific objective

To achieve this thesis there are lots of specific objectives. The specific objectives are listed here

- The first part of the design of this programming language is to identify the keywords of the language in Amharic(አማርኛ) and also identify the data types, characters, numbers and so on.
- Next using the keywords identified to the design of the Grammar of the language using Extended Backus-Naur Form. The grammar of the language will guide the overall structure of the language and will help us identify all necessary components.
- The next is by using the grammar developed to show the syntax of the language. How a program written in this language will look like
- Write the prototype of the Tokenizer by using Java's Regular expression for selected constructs of the language to show what tokens are identified in the language
- Write a parser to show how a program construct is identified in the language
- Handle characters that are different in form but have the same pronunciation and usage as ኦ and ዓ

1.5 Scope and Limitation of the Study

1.5.1 Scope

This study will focus on designing a programming language in the official working languages of Ethiopia (አማርኛ). The research will focus on identifying easily understandable Amharic (አማርኛ) keywords that will be used in designing the language. As keywords are the basic and unchangeable part of the language due focus will be given to make the keywords understandability high. The other thing this thesis will focus on is developing the grammar

of the language by using EBNF (Extended Backus-Naur Form) which is a family of meta-syntax notation which can be used to express a context-free grammar.

The other focus is on the syntax of the grammar, as programming languages are a structured way of giving instruction to computers the syntax of the language that is being designed will be designed keeping in mind the easiness to be understood, simplicity clearness and a higher degree of resemblance to existing high-level programming language's in English. The reason for that is to help make easy the transition from this language which is introductory by its nature to other languages that are complex and have been used to design existing software in the world.

Even if this thesis does not include implementation of the programing language, there will be a sample tokenizer included to show how the tokens, the simplest chunk of data in the language, are identified during implementation. So, a tokenizer of the languages selected basic construct will be included in this thesis. The other thing is, a continuation of the tokenizer which is parsing that will take the tokenized data and process to check if the sequence of words that matches any of the languages constructs that is included in the grammar. Tokenizing and paring are part of the compiler design. They are the front end of the compiler design. The parsing part, which also is a prototype like a tokenizer to show how the basic selected constructs of the language are parsed and identified as a meaningful part of the language's syntax.

1.5.2 Limitation

The focus of this study is on the design of an Amharic programming language as such it will not include the implementation of the language. The sample tokenizer and parser are not written for the programming construct. The thesis will not talk in detail about the compiler design of the language. It will only give a highlight on what to consider when designing the compiler in the future.

1.6 Literature Review

In the next chapter, an extensive review of topics that are relevant to the success of the thesis will be conducted. Some of the topics that will be reviewed are Programming Languages from its history to what they are intended to do, Non-English Programming Languages

which will include a review of literature that relates to a programming language with non-English keywords.

1.7 Software Tools

To design the Amharic(አማርኛ) programming language different software tools will be employed based on their relevance to the design. The first thing we use is a met-syntax Language used to write the Grammar of the Language. Meta-syntax Languages are Languages which have their own grammar, syntax, and keywords, like programming languages, to write the grammars of a programming language.

The thesis will include also a prototype of the tokenizer and the parser, which will be implemented by using Java SE. To write the sample tokenizer and parser a java regular expression library will be used. There is also a sample code editor which will be used to show how the syntax of the language looks like. This part of the thesis will be done by using JavaFX. According to sun JavaFX is a new platform to write rich client application(Topley, 2011).

1.8 Significance of the Study

This thesis upon completion will lay the groundwork for a new programming language that will fully accept input in Amharic (አማርኛ), display both output and error, and also have documentation that is written using Ethiopia's working Language Amharic. The thesis will help programmers to dive street into the implementation of the programming language without the need to worry about the design of the language. It will also help other researchers to have the confidence to research on programming language in advance.

CHAPTER 2
LITERATURE REVIEW

2.1 Programming Languages

When we want to control and instruct computers to perform a task or solve a problem, we need a way of communicating with it. That way of communication happens by using programs, which is a piece of text, that has its own structure and dictionary (Allain, 2013). Programming is the process of writing an algorithm and converting it into a form computer understand. The language a machine understands and execute is called Machine language. Programming is a human activity that is a great challenge, involving the design of machine behavior that at times assist humans in their work and at times replace humans in intellectual tasks (Kitchenham & Carn, 1990).

Programs quite often are written as a humanly understandable language which is a series of words to instruct computers to do what and how we want to solve our problems. Computers don't understand human languages; internal applications don't communicate as humans do. Computers do have their own way of understanding each other. The software's in a computer communicate by sending messages to one another. Since we humans have a hard time writing a computer program using a language computer understand and also machines don't understand the language we use, we develop a programming language to help us with that problem. So, the program that is written in a language humans understand must first be translated into a form that computers are able to understand and execute (Liang, 2013).

To write a humanly understandable program we use programming languages. Programming language is a set of instruction comprised of human understandable text used to write computer programs and then converts them in a form that the machine can understand easily. In other terms, a programming language is used by programmers to write and provide specific sets of instruction to the computer which the computer read the instruction, process it and then execute to produce what the programmer wanted to happen (Allain, 2013).

Programming languages have been in use for quite a long period of time and they have shown us how effective and helpful they can be when handling difficult works. Programming languages while they are human understandable and unorganized, they rather are structured and with specific vocabulary. The vocabulary even if it is small it is used to write programs that are more sophisticated and thousands of lines of code.

There are thousands of programming languages designed and implemented. Each of the programming languages designed after the other shown great improvement that was better in one way or the other than the languages that preceded it. But that high number of programming languages doesn't mean that a programmer has to study all or most of them. In practice, most programmers do not tend to use more than a few languages (Terrence W. Pratt, 2000).

Programming languages are versatile and of great importance. Different Programming Languages perform different task depending on the functionality they provide. Some programming languages are more suited to accomplish mathematically related tasks, others are good for building software's and some others are used to do simulation for architectures. Finding a programming Language that does all the task is impractical and impossible. There are no ideally suited programming languages (MacLennan, 1986).

2.2 Programming Paradigms

As stated, above programming language can be used to do different tasks that can be used to reduce human workloads. A programming paradigm is a way of grouping programming language based on their features. A paradigm is an approach that is preferred for programming that programming language support.

Different problems are better suited to different paradigms. Some programming language support different paradigms (Fernández-Villaverde et al., 2018). For example, Python programming language supports both functional and object-oriented type of programming. There are different programming paradigms that are applied in programming. The division between the different programming paradigms sometimes is not clear. Sometime one programming paradigm may have an aspect of another paradigm. But the general aspect of

the paradigms is quite different and determine how we design programs(Pfenning, 2006). An overview of the four basic and widely used programming paradigms are explained here.

2.2.1 Imperative programming

Imperative programming is the oldest programming paradigm that is mostly used for small programs (Fernández-Villaverde et al., 2018). Imperative programming is a programming paradigm which uses statement's that change the program's state. It focuses on describing how programs operate (Gurbani et al., 2008).

Nørmarks (2011) define imperative as asking for something to be done. It is a paradigm that closely models computers, it works based on moving bits and changing states. Imperative uses natural language to pass instruction to the computer. Its basic unit of abstraction is a procedure, there are a group of statements inside the procedure and are executed sequentially. The sequential flow can be modified using conditional and looping statements.

Procedures are a named sequence commands and the name can be used to invoke the procedure. When this happens, it is called procedural programming. Some languages that support imperative programming is Pascal, Cobol, Fortran (Vujošević-Janičić & Tošić, 2008). One characteristic of imperative programming is the incremental change of the program state with time. It is very similar to the day-to-day routine description, like food recipes.

2.2.2 Logic paradigm

Logic programming is one way of approaching programming(Pfenning, 2006). Logic programming is a type of programming paradigm which is based on formal logic and declarative programming. It is a set of sentences in logic form, which focuses on expressing rules of a specific problem rather than the decomposition of the problem into an algorithmic description. Logic paradigm is designed for theorem proof and artificial intelligence but allows the general computation (Şehitoğlu, 2008). A logical program is a collection of logical declarations describing the problem to be solved. It consists of

- Axioms ---define facts about objects
- Rules --- define a way for inferencing facts
- A goal statements---define a theorem provable by axioms or rules

The rule of inference is applied on axioms and a goal statement is produced. Examples of programming languages that follow logical programming are Prolog and Gödel (Vujošević-Janičić & Tošić, 2008)

2.2.3 Functional paradigm

An Introduction to Functional programming book defines Functional programming as programming consists of building definitions and functions and using computers to evaluate expressions(Bird & Wadler, 1988). In functional programming, computation proceeds by rewriting functions and not by changing states like imperative programming. The fundamental characteristics of programs written using this programming paradigm is that of not possessing the concepts of memory (Maurizio & Simone, 2012).

Programs written in Functional paradigm are a collection named functions invoked inside other function using the name. It allows programmers to think in a higher level of abstraction it encourages thinking about the nature of the problem rather than the sequence of actions. It uses two fundamental mechanisms, namely binding which is associations of values with names and applications which computes new values. One example of functional programming is Lisp (Vujošević-Janičić & Tošić, 2008).

2.2.4 Object-Oriented paradigm

The conceptual model of this paradigm is developed from the simulation of the real world we live in. Object-oriented programming is based on objects that exist in the real world and encapsulate property and operations. As real-world objects interact, objects in object-oriented programming use message passing to capture interactions between objects (Vujošević-Janičić & Tošić, 2008).

In his paper in 1987 Wagner defined “Object-oriented” as a culmination of objects, classes, and inheritance(Wegner, 1987). This is a good definition of object-oriented programming. Object-oriented programming is all about constructing the building blocks of objects, which is classes and instantiating them. Objects in object-oriented programming share property by using inheritance. Inheritance is the backbone of object-oriented programming, which lets codes to be reused by organizing similar operation and data’s in the same class called parent class. Wagner also stated that to call a programming object-oriented programming its class

must be able to instantiate objects and the classes are structured in hierarchical in an inheritance manner (Wegner, 1987).

According to Wenger Objects are an autonomous entity that responds to messages or operations and share properties. Classes classify objects based on the operation they perform and properties. Data abstraction is used to hide data and operation implementation in the object.

Another definition is given by Kendal in this object-oriented programming in Java book, which states that object-oriented paradigm is based on the ideas of Encapsulation, Inheritance, Generalization, and polymorphism and help the development of software and system that models both the operation in the software and the data associated with it. Proponents of this paradigm argue that this leads to the re-use of codes thus saving significant development time and cost (Kendal, 2009)

Encapsulation is the foundation of the object-oriented approach. It is the process of hiding the property and methods in a single unit. It helps protect the code or the detail in a unite from being changed. Encapsulation only allows other class to use not modify the details.

Daniel Liang in his book on Java programming put Inheritance as an important and powerful feature of reusing software (Liang, 2013). Inheritance is another characteristic of object-oriented programming. It is a very important part of any object-oriented programming system. It is characterized by sharing resources. In inheritance, similar properties and operation are grouped in a single class called parent classes and other classes, usually called child classes will inherit properties and operation from the parent class.

In this thesis, Object-oriented programing is the selected paradigm. The reason for that is most of the popular programming languages in use are in one way or the other object based. The Amharic programming Language will support Class, object creation and inheritance. Doing this will make the transition as easy as possible for students who learn this programming to other widely used programming languages like Java, C++, and Python

2.3 English based Programming Language

Programming has a long history, but a notable resemblance of the programming we know now started in the years of Charles Babbage, who designed two mechanical computational machines during the years(1820-50), The Difference machine which was based on finite difference theory, and the Analytical machine which has a lot of similarity with a modern computer (Georgatos, 2002).

After Charles Babbage's introduction those mechanical machines the programming world has shown very promising advancement. But the most interesting discovery happened in the early '50s of the 20th century with the introduction of a programming language called Fortran. According to (MacLennan, 1986) Fortran was introduced by Backus and his colleagues in the office of Naval Research Symposium as a paper focused on speed coding. Speed coding was aimed at designing a programming language using mathematical notation. By 1954 a preliminary the external specification of FORTRAN (which is short for Formula Translation) was produced.

Until the introduction of Object-oriented programming language, quite a lot of programming languages has been introduced to the world. From those languages, Algol60, Pascal, Ada, and Lisp were some of them. These programming languages were designed keeping in mind the professionals and also were designed by professionals. Allan Kay developed the first object-oriented programing Language, Smalltalk. It is a language that can be used by anyone interested and can be incorporated into personal computers (MacLennan, 1986).

Smalltalk design and existence were realized because there was a notion that there is a way to describe by composing a single kind from different building blocks and hide state and process inside itself and can interact with other by exchanging messages. Allan Kay called this Object-Oriented. Smalltalk by doing so improved the efficiency of modeling and compositions in designing programming (Kay, 1993).

The other widely used programming Language after Smalltalk was C. C followed the programming Language B and BCPL. It was developed in the year 1970 by Daniel M. Ritchie. C has often been referred to as false high-level language or middle-level language.

It was designed to replace Assembler, Cobol, and Fortran as the language of choice in the mainframe world (Lindstorm, 2005).

C Programming Language has the capability of accessing the low-level functionality of the computer. Which makes it powerful and fast. (Ritchie, 2005). In 1983 the C programming Language becomes standardized by the American National Standards Institute. ANSI compiled a committee to work on a version of C that is the machine-independent definition of the language (Brian W. Kernighan, 1978). C popularity exceeded the expectation of the founders.

Here is a Hello world code written using the C Language

```
#include <stdio.h>

main() {

        printf("Hello, world!\n")

}
```

This is a very simple example which we see in most programming languages. This code snippet includes the #include statement to include the necessary library, in this case, the standard Input Output. Then followed with the main function where any C program starts executing. This function is a must for all C programs. The printf function inside the curly brace is used to display/write the string inside the bracket.

The C programming language was then followed by the widely accepted C++, Python, and Java. These Languages are high-level Languages. Python is a powerful and easy-to-learn programing language that is based on previous programming languages and it has been developed to be more suited to the current operating system, networks and hardware (Lindstorm, 2005). It was developed in Netherland which is a Non-English-speaking country by Guido Van Rossum. The reason python was developed using English as a base for its keyword was for internationalizing purposes.

Python was designed to realize readability and easy typing. It is powerful that it has been in use in a wide area of application. Python provides programmers with very important

capability like simple text processing, file processing, network operations, and most importantly it provides a very rich library to GUI programming (Lindstorm, 2005). Python runs every where and can be used by anyone since it is open source. It is the first choice of programming for novice programmers.

Unlike most programming languages in use, Python is an interpreted language i.e. a language that are stored the same as how the programmer wrote it and converted to machine code at runtime so that the computer understands it. Which lets a program be run without the need to compile the whole program. If there is some erroneous code in python, the interpreter will interpret the one that works fine until it finds the error. But if the error is at the beginning of the program it will return an Exception. Python also takes advantage of the underlying C libraries found on most computers. This makes python more powerful and rich in libraries. Python also combined its simple syntax with Java libraries and created JPython (Lindstorm, 2005).

Java programming language is a well-known, object-oriented, general purpose programming Language. It is a lot similar to C and C++ with its syntax but it avoided parts that are confusing, complex and unsafe(Lindholm et al., 2015). The Java programming language is an indispensable resource for everyone, from novice programmers to advanced programmers (Arnold et al., 2013).

Java was developed by Sun Microsystems in 1995 which then is incorporated by Oracle Corporation, which manages the famous Oracle Database. Java has a wide range of use and application. The major ones are Desktop application, Graphical User interface, Server-side, and Client-side web applications. Java like python and other object-oriented programming languages is a write once, run everywhere language. Which means a program written using Java can be run in any operating system. There are lots of websites and applications that are dependents on Java and don't work unless Java is installed. Java applications are compiled to bytecode that can be run using virtual machine.

As an object-oriented programming language, Java programs are built using classes. Using the classes in the program, it is possible to create a great number of objects where each of them can have a different property and perform a different operation. Objects are called an instance of a class (Arnold et al., 2013).

2.4 Non-English Programming Languages

The fact that many of the programming language in use today are developed basing the English language for their keyword had a significant effect on the growth of technology-related areas in Non-English-speaking countries. To solve those problems researchers in different country developed a programming language using their own native language as the base for the languages keyword. In this part of the document different non-programming languages are studied in detail. Unfortunately, at the time of writing this document, there are no programming languages that are developed in Ethiopia. But a programming language in Amharic which is in development has been reviewed lightly. The language is named AxumLight even if the programming language is not available for download it has been tried to assess and review by using online document and other resources.

From the official page of AxumLight it is a programming language written using the Amharic lexicon (EthioCloud, n.d.). Which lets Ethiopian developers develop programs by using their native language. It is built and runs using the .NET framework . It is based on the Amharic alphabet and uses Semitic based Geez characters native to Ethiopian languages. The purpose of AxumLight is to enable users to learn how to program and develop Amharic software components and applications. Users can use both English and Amharic in the editor. The editor also supports code highlighting. Like stated above the programming language that is described in the Ethiocloud is not available for download so there is no way to check its performance. The next picture is taken from the official website and it somehow enable us to see how the language looks like

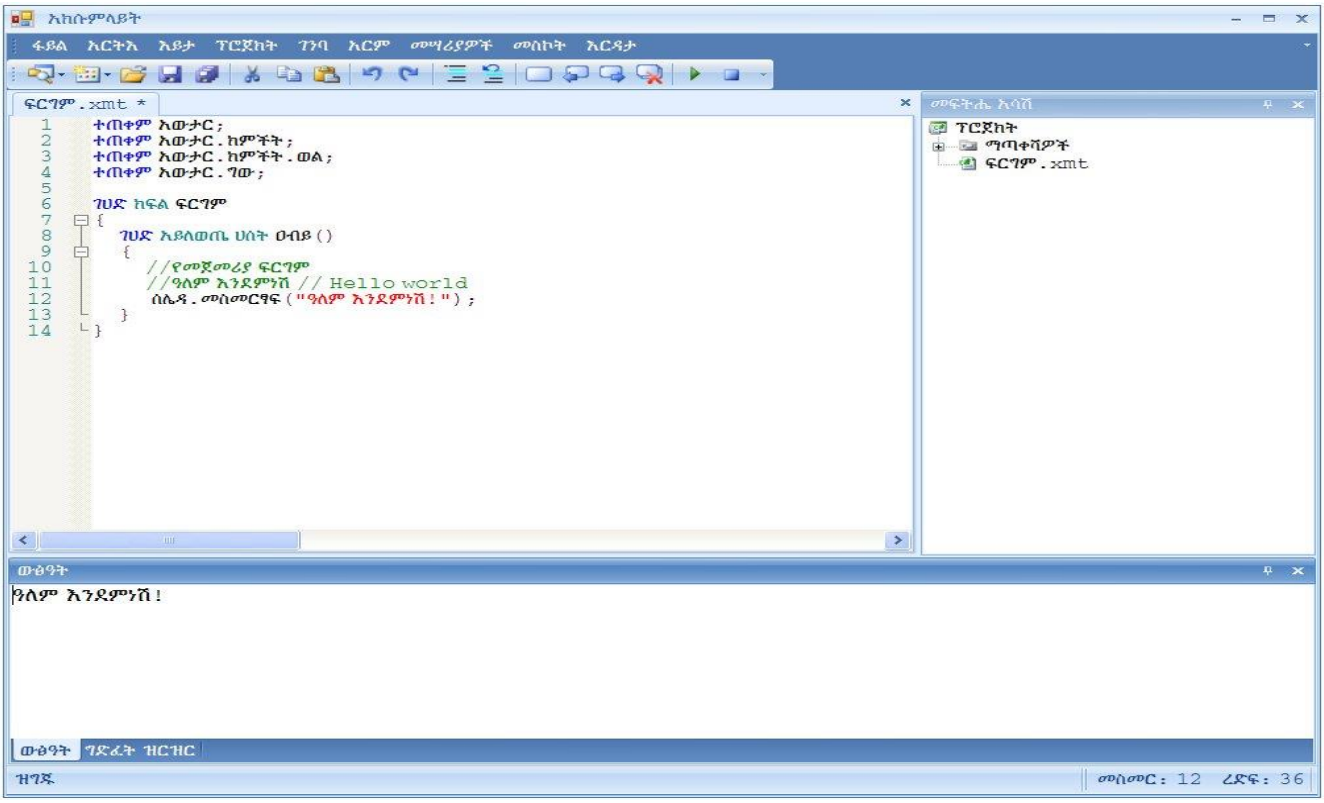

Figure 2.1: Hello world using AxumLight taken from EthioCloud.com

In the figure above is a HelloWorld code written using AxumLight, the language first uses an import which is designated using the Amharic word ተጠቀም to include libraries that are used in showing the hello world text to the console. Next, the program wrote the class declaration like

ገሀድ ክፍል ፍርገም

its translation in English is

public class <identifier>

this declaration is followed by the delimiter '{'curly brace which shows the start of the class scope then it is followed by the main method declaration

ገሀድ አይለወጤ ሀሰት ዐብይ () {

its equivalent English translation

public static void main () {

this is where code execution commences. Then inside the main block, there is a system method to display Strings into the console

ሰሌዳ.መስመርፃፍ(“ዓለም አንደምነሽ”)

the English equivalent is

Console. WriteLine (“hello world”)

When the system is run the text inside a quotation will be displayed in the console. From the way the syntax is constructed, we can most certainly say that the language is a direct translation of the high-level language c#. For languages that are a direct translation of another high-level language, it is good to use a source-to-source compiler or trans compiler also known as a transpiler.

Until now it has been tried to show how a programming language that is in development in Amharic is structured. We have used the information provided in the official page of AxumLight and a snapshot by the developers. In the next part, other non-English programming languages will be looked at.

According to an article written by (Bingöl et al., 2018) chameleon is a Turkish programming language written using the Turkish language. The goal of this programming language is to solve the problem that is being faced in teaching Turkish students at early age computer programming. As a study conducted on the usage of local or first language to teach students has indicated students who have been taught computer coding using their local language can learn to program faster and develop new programming concepts that those students who learn computer coding using English, when English is their second language(Dasgupta & Hill, 2017).

Chameleon is a language which was developed to let Turkish speaking students and developers create an application with Turkish coding structure on windows operating system and mobiles. Chameleon is used to write a portable program. Once a program is written using Chameleon and compiled it will run in any platform mobile or computer. The program has four transitions from source code, which is a text file with a dot but extension to the final portable form which is a dot munf file. According to study conduct on students from high school on the chameleon programming language, students found it easy to program.

Here is a sample source code as a figure taken from (Bingöl et al., 2018). For readability purpose the sample code is written in its English form.

```
Class Hello
{
    Function start ()
    {
        Print ("Hello world with chameleon")
    }
}
```

In the above code snippet, the first line is a class declaration

```
class Hello
```

the third line is the main function which is where the programs start execution

```
Function start
```

Then print will display the text inside quotation on the command line.

Like Chameleon, there are a couple of programming languages designed in Arabic. The most notable ones are قلب roughly pronounced as QALB and ARABIAN. QALB which means heart in English is an Arabic programming language developed by Ramsey Nasser that fully uses Arabic keywords(Nasser, 2012). QALB is a programming language that explores the human role in coding. It is implemented using JavaScript.

ARABIAN is a programming Language that is designed using the Arabic language as the base for the keywords in the language. It is a simple and imperative language, designed for educational purposes. The syntax in the language is emphasized on efficiency and simplicity. There is no dynamic storage allocation and parameter passing happens using value and references. The Language shows Compilation and runs time errors in a simple and understandable Arabic language. The evaluation on the design showed that simplicity and

efficiency were achieved by removing the parts which will be inefficient in the implementation of the language as well (Al-A'Ali & Hamid, 1995).

There are some programming languages that are developed using the Spanish language. Some of them are GarGar, Latino, RoboMind, and Tango. GarGar is a procedural programming language based on the Pascal for learning purpose. Latino is another programing language with a syntax based fully on the Spanish language (Zegiestowsky, 2017). These programming languages are not the purview of this review.

Tango is another programming language that is similar to that of the above-mentioned programming languages. This programming language like any other non-English programming languages is aimed at solving the language difficulty in writing programs using languages with English based keyword. The language is designed to make programming easy and effective for the Spanish speaking society. The language includes Spanish language accents and also uses cross-compilation to java to generate codes. Tango requires java to be installed on the computer in order to use it (Zegiestowsky, 2017).

Tango differs from the Latino and RoboMind in three basic ways. First Tango supports usage of special characters, it supports accent markers and tildes in both the keywords and other language parts. The second difference is that it uses Transcompiler. That is the languages source code will be translated into Java source code which facilitates simple installation and usage. The language generates a java executable file. For tango to function properly there must be a stable version of Java to be installed in the system. The final difference is that the languages keyword has a java like flavour.

Similarity between Java and Tango

Java version

public void func_name (int param1, int param2) {…}

Tango version

func func_name públic@ vaci@ (ent param1, ent param2) {…}

English has also been a problem for students in China who wants to learn to program. Because students are new to English and the words are not familiar to non-native speaker

made learning programming harder and took them too much time. This problem is prevalent mostly to young students and programmers who don't have exposure to English. There are millions of experienced programmers in China who program using English.

There are different programming languages designed using Chinese language as the base for the keyword. Some of the major programming languages that are common among novice Chinese programmers. Basic Chinese programming language is a name given to a different version of the basic programming language. It was developed in the 1980s. The other programming language is ChinesePython which is the Chinese version of the well-known programming language. The other programming language used to develop 3D animation and game is Mama. Mama was designed to help young students engage in the animation and game development world using their native language.

The other programming language that uses chinse language is RoboMind. RoboMind is a programming language that is available in many different programming including Chinese. RoboMind is an educational programming environment that has its own scripting language that teaches the student to know the basics of computer by developing a simulated robot.

Eyuyan literally translated as "Easy Language" is one of the rare programming languages in China which uses Chinese characters and punctuation fully. According to its creators "Easy Language" is a functional development tool. This language was created by a person name Wu Tao.

Eyuyan start as a free programming language that can be used by everyone but after the internet revolution in China, i.e. after the coming of Baidu, Alibaba and Tencent, Wu started charging for the IDE (Integrated Development Environment). The language even if it charges its users it helped the test integrity on the Chinese language compatibility with modern technology.

But Eyuyan (Easy) Programming language has also a drawback that affected the usage of the language. This programming language wasn't just developed to write programs that are legal, it was also used to write hacking tools and game cheating scripts. The reason for that is the language has a powerful library. This has opened the gates to write malicious programs. Most online tutorials written for this language teaches "account stealing tools", like the Chinese version of w3schools (Zhang, 2017).

Chinese python is a programming language that is a bit different from the above mentions programming languages. The reason is that Chinese python is a Chinese translated form python 2.1.3 programming language. This language contains Chinese translated python keywords. In addition to that, it is possible to use Chinese characters for variable names and pythons' built-in functions can also be operated in Chinese.

All the above mentioned Non-English programming Languages were aimed at solving the problem that has been faced by students of different level when it comes to computer programming. All the aforementioned languages achieved simplicity, easy understandability and development of their respective countries' language in technology.

2.5 The Amharic(አማርኛ) Language

Amharic is the official language of Ethiopia, which belongs to the Semitic language groups. Next to Arabic Amharic has the largest speaker in the Semitic language group (Gezmu et al, 2018) Even if the Amharic language was in use in Ethiopia for a long period of time its availability when it comes to the technology related area it is surprisingly not good. The use of the language in computers and mobile phone is rare. There are not a lot of study on the Amharic languages but it is stepping up in recent days. Researchers in a university tried to study the application of Amharic language in different areas like Amharic character recognition, Amharic speech recognition and so on. There were different studies on natural language processing on Amharic and Amharic letter recognition. This thesis focuses on the application of Amharic language in the design of programming language.

2.5.1 Amharic characters

The Amharic language has 33 characters and each character has 7 making the basic letters to 231. Which is quite a lot number of letters. The Amharic orthography contains inconsistency, where some letters represent the same phoneme or have the same pronunciation. The overrepresentation phonemes or the user of different letters to represent the same phoneme is because the Amharic language is descended or evolved from Geez and in Geez the letters that are redundant in Amharic represent different phonemes (Negesse & Ado, 2016). This representation of redundant letter for the same phonemes makes a word in Amharic to be represented in a different collection of letters but with the same pronunciation

and meaning. For example, the Amharic equivalent for the English word ‘work can be represented in two ways one as ‘ሥራ’ or ‘ስር’. The two Amharic words have the same pronunciation and also have the same meaning. In Amharic, unlike English there are no cases, there is only a single case. Amharic language in addition to the 231 characters Amharic also has characters that are the production of the fourth form of the base characters. These characters make up part of the Amharic words and are also part of the Amharic Unicode in the Ethiopic block.

2.5.2 Geez numbers and punctuation

Even if the number used in Mathematical calculation and another day to day activity of the people is the Arabic numbering system which represents numbers that range from 0 to 9, Amharic also uses Geez numbers to represent numbering, years and other. The Geez numbers, unlike Arabic numbers, don’t start from 0, it starts from 1, which is represented by ፩.

There are also punctuation marks used in the Amharic language. There is word separator ‘:’ punctuation mark now a time they are replaced by space. There is also punctuation used to separate lists like comma which is designated by ‘፣’. Punctuation marks are not used in this language design because punctuation tends to affect the readability. The only Amharic punctuation mark that is used in the grammar is a statement terminating symbol in Amharic which is represented by ‘።’. This symbol is used to write a multi-line comment

2.6 Context Free Grammar

Context Free Grammar is a more powerful method to describe languages. It is used to describe features like optional substitutions, features that are recursive in nature and others.

The basic definition of context-free grammar is given below

A grammar consists of a set of production which is collection of substitution rules. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow. The symbol is called a variable. The string consists of terminal symbol which comprises variables and other symbols. The variable symbols often are represented by capital letters. The terminals are analogous to the input alphabet and often are represented by lowercase

letters, numbers, or special symbols. One variable is designated as the start variable. It usually occurs on the left-hand side of the topmost rule (Sipser, 2012).

Context-Free Grammar is a set of production rules to generate a word. It consists of the following components

- A set of terminal symbols, which are the characters of the string of the generated word. This symbols as the name indicates are terminal. They will not be part of the substitution.
- A set of Non-terminal symbols, these are placeholders for the terminal symbols. Non-terminal symbols will be replaced by a pattern of terminal symbols at some point of the production.
- A set of productions, which are rules that guides how the non-terminal symbols will be replaced by a terminal symbol or other non-terminal symbols
- A start symbol, which is an unreplaceable non-terminal symbol that shows the start of the grammar.

The purpose of context-free grammars is providing rules from which a syntactically valid string is generated

Each rule in production takes the form $X \rightarrow \gamma$

Where X is a non-terminal symbol and γ represents a set of the terminal or non-terminal symbols (possibly empty)

A representation of an arithmetic expression using context-free grammar definition is given below

expression → number

expression → expression

expression → expression + expression

expression → expression - expression

expression → expression / expression

expression → expression * expression

Here the terminal symbols can be identified as (+, -, / and *) whereas the non-terminal symbols are expression and numbers. The first production rule used in the above snippet states that expression can be replaced by numbers, we can also define a number as a list that contains the number 0 up to 9. In the above production rule, an expression can also be replaced with another expression.

2.7 Extended Backus–Naur Form

Extended Backus-Naur form is similar to that of Backus-Naur form with a little modification. A context-free grammar was used in a programming language for the first time in the design of ALGOL60 programming language by Backus. It was named Backus-Naur form, after the members of the ALGOL60 committee John Backus, who previously was involved with Fortran and Peter Naur. BNF, unlike context free grammar, doesn't have a large set. Some of the changes in BNF

Arrows '->' to '::='

Non-terminals are written surrounded by angle brackets <NT>. This is different in Extended BNF which writes both terminals and non-terminals as they appear without angle brackets or quotations.

Terminals or non-terminals with the same head are grouped by a vertical bar '|', like 1|2|3|4

Extended Backus-Naur Form is a notation to describe the syntax of a programming language. Syntax shows the way to write features in a given programming language (Feyman, n.d.). Different programming languages do have related but different syntax. A program written using a syntax of one programming language cannot be understood by a compiler of another language. The control forms in EBNF are sequence, decision, repetition, and recursion.

The extensions in EBNF are

'*' Kleene star: means zero or more occurrence

'+' Kleene cross: means one or more occurrence

'?': means zero or more occurrence

Use of parenthesis for grouping

2.7.1 EBNF rules and descriptions

An EBNF is an order list of EBNF description. Each EBNF rule like Backus-Naur form has three parts: left-hand side, right-hand side, and separator. The separator can be::=, = or <= and it can be read as ‘is defined as’

LHS:: = RHS or

LHS = RHS

LHS <= RHS

In this paper the equal separator is used.

Following is an example of integers in EBNF

EBNF description: integer

digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

integer = [+ | -] digit {digit}

2.8 Design Principles

In early times due to the scarcity of memory and speed of the machine are slow the primary worry of programmers was speed and memory usage. This principle is called the efficiency of execution. This criterion still matters because as the programmer gets bigger the execution tends to slow. So, designing a programming language with better efficiency for execution is appealing for programmers. In this section, principles of programming language design are included

In most programming language abstraction is a principle of avoiding details. Abstraction is a way that helps avoid something’s to be described or written more than once. It is a way to avoid recurring patterns. Most languages support different unit of abstraction (MacLennan, 1986). In object-oriented language class and method is the common unit of abstraction. On the other hand, for functional programming language, the unit of abstraction are functions or procedures. Abstraction will also help with simplicity, readability, and writability of a programming language by avoid repetition of concepts.

A programming language should be efficient. The efficiency of a programming language can be described as the ability to allow translators to generate efficient executable code. There are different design decision that helps in creating an efficient programming language. To have an efficient executable code for instance programming language designers choose static data type typing. Having a statically typed data will help speed up the run time because there is no need for checks during runtime since the types of data are checked by the compiler. On the contrary using dynamic typing like python does will force the runtime to check the type. This cause execution to be slow.

Efficiency in addition to creating efficiently executable code can also be defined as programmer efficiency. The ability of a person to easily read and write programs in the language also determine the efficiency of the language. How easily can a programmer express a complex structure using small number of constructs. How concise is the syntax of the language? These traits also contribute to the efficiency of a programming language (Louden & Lambert, 2011).

Integration of programming features is also another issue when designing a programming language. How well programming language features are integrated is known us regularity. When a programming language allow greater integrity among constructs means there are less restrictions. Regularity has three aspects generality, orthogonality and uniformity. Orthogonal design allows programming constructs to be combined with no or minimum restrictions. Uniformity of a programming language design means similar constructs must look and function similarly and different constructs must look different and function differently.

The last principle included in this document is extensibility. Extensibility is a programming languages ability to allow users to add features. An example of extensibility in programming language is to define new data types and new operations(functions). There are few programming languages that allows users to add new syntax and semantics. Lisp is a programming language that allows programmer to add syntax and semantics in addition to adding functions and data types.

CHAPTER 3
DESIGN METHODS AND APPROACH

3.1 Tools and Methods

3.1.1 ANTLR (Another Tool for Language Recognition)

Parsing is an important part of compiler design which is an area of study in universities. But writing parser by hand is of a tedious and error-prone process (T Parr & Fisher, 2011). To solve that problem there are different and effective parser generators. In this document, a language recognition tool named ANTLR (Another Tool for Language Recognition) is used to generate the parsers, laxer and other necessary data. Here this ANTLR is used to check the validity of the grammar, the generate code is not used to implement the prototype of the parse or laxer. The sample tokenizer and parser has been written by hand so that to only include the parts that we need.

ANTLR v4 is a computer-based language recognition tool. It generates a parser by taking a grammar as an input. The tool will read the grammar which is a structed text file written by using a meta-syntax notation called Extended Backus-Naur Form (EBNF), process it, execute and translate it (Terence Parr, 2013). It is written in java and generates a parser and Abstract Syntax Tree (AST) of the grammar. ANTLR generates an LL (*) parser for a given grammar.

It also provides syntax highlighting and syntax error checking mechanism to make sure the grammar written complies with standards and doesn't cause problem during translation. The other nice feature utilized from ANTLR is its Live grammar interpreter for grammar preview. This feature provides a preview of how the grammar reacts to a given data.

LL parser is a top down parser for context-free grammars. The parser parse form left to right, performing leftmost derivations.

ANTLR has been used to parse twitters search queries which host more than 2 billion queries a day. The other major use of ANTLR is in NetBeans IDE. NetBeans IDE uses ANTLR to

parse C++. It has a feature that can be used to test the grammar. ANTLR notifies errors that could cause problems when the grammar is translated to parser.

ANTLR v4 deals with left recursion correctly unless it is indirect left recursion. Examples of a grammar with indirect recursion is given below

leftRecurssion: indirectRecursion;

indirectRecursion: leftRecurssion

This type of grammar is using indirect recursion which is confusing during translation. To avoid any confusion that happens because of indirect recursion it has been decided to not use indirect recursion.

The other issue that has been handled by ANTLR v4 is arithmetic precedence. Dealing with arithmetic precedence is confusing and error prone. For instance, in the int x = 4 + 5 * 6 the value of x can be different depending on how the compiler gives precedence. The value of x can be either 54 if the addition operation precedes the multiplication operation or it can be 34 if it is done in the reverse. The grammar has been written in a way it follows usual mathematical arithmetic operation precedence's. the grammar also deals with operation of equal precedence. The precedence and the syntax tree of arithmetic operation are shown in in the next chapter.

3.1.2 Grammar

ANTLR uses grammar written using Extended Backus-Naur Form as input. The Grammar of the language used latest standard so that developers who intend to use the grammar can use it without major modifications (Tio & Niekerk, 2001). Extended Backus-Naur Form has been used to write the grammar language. The Grammar defines all necessary rules of the language. The complete production rules of the language is given in Appendix 1.

The grammar of the language has been tested by using ANTLR. Data's has been given to the grammar and the generated parser and laxer has checked the data's conformity to the syntax of the language and a syntax tree has been generated to who the hierarchy of the tokens in the given sample source code.

The grammar for the Amharic language is a two-part file. The first part of the grammar is a parser grammar which contains the necessary symbols and the production rules. All the

grammar rules are included in this file. The second part of the file is the lexer grammar which contains the keywords and all the literals. The file is separated for clarity purposes.

3.2 Design of the Language

There has been a design decision that has been taken when designing this Amharic programming language. The design decisions are made to make the language as simple as possible as it is meant for beginners. In addition to simplicity, the language decisions are made to make the language resemble widely used high-level languages. Due to the fact that most of the widely used programming languages are high level and to make the transition to other languages smooth, the language designed here is also a high-level language. The language is an object-oriented language, by which students are able to create objects of different properties and operations. Because it is an object-oriented language it will introduce students to widely known programming language concepts and help understand other object-oriented programming languages easily.

The language gives due considerations to students who are in lower grade who don't have the chance to learn English, those who don't attend school but want to learn programming without the need to wait to learn English first and for computer science students who want to research in using Amharic language in technological-related areas and also develop advanced Amharic programming languages with great capabilities. The language is supposed to be implemented to only windows operating system. The reason behind it is that almost all computers that are available in schools at this time run windows. Since this is a programming language most lower grade students can be easily familiar with. After the first version of the language is implemented, the next version will include portability.

It has been tried to make the language as simple as possible. When we talk about simplicity it is to mean the language is easy to learn and easy to use. Even if we will talk about this part in detail later on, to reduce the complexity of the language access modifiers, modifiers, in general, are not included. The other thing is to reduce the hierarchy of class and minimize confusion among beginners' students' class and method recursion has been left out of the design. This will help avoid the confusion of stairs of class and methods to novice students. In order to make the language easy to learn, it has been tried to remove most the delimiters

available in most programming languages like Java and C++ and give it a python like structure without the colon delimiter used after block structures like method and if blocks. This will make the grammar clean and only include necessary tokens.

To make learning the program easy, it has been tried to reduce the number of keywords in the language. Making the number of keywords in the language smaller will reduce the number of concepts they have to remember when programming. The other design decision taken when designing the language was to avoid using the direct translation of keywords that are prevalent in a most programming language. The direct translation of keywords that are common in most languages doesn't make sense when it comes to the Amharic language. For example, if we take one of the keywords that exist in object-oriented programing languages 'class', is translated as one of this two Amharic words 'ክፍል' and 'መደብ'.

An attempt has been made to make the words as much as possible words that are common to most of the people rather than choosing words which are not common on day to day speech. This makes the keywords easy to remember and will guide programmers to make use of names that are familiar to others and easily understandable when writing class and methods. It has also been tried to make the keywords appropriate and relate to the meaning of the things they do. In addition to the familiarity of the keywords, there is also as object-oriented support of abstraction. That with a small number of keywords will help to write clear code and that will improve readability.

The regularity of a programming language is the other principle that is considered to be incorporated in this language. Regularity is a measure of language integrity. For a regularity to be achieved in a programming language features integrity should have a minimum of special cases or exception. Regularity can also be defined in layman's term as to how human-friendly a language is. There are three aspects of regularity which are generality, orthogonality, uniformity. Generality is having closely related constructs in one general construct. Whereas orthogonality is the ability to combine constructs without unexpected restrictions. Orthogonality is the ability to use programming features with a different structure. The last aspect regularity is uniformity. Uniformity is a measure of how similarly looking are similar features. Similar items must have similar meanings and look similar and

on the other hand, different things must have different meaning and look (Louden & Lambert, 2011).

The other design decision taken was to choose between the two type of variable typing, static and dynamic typing. Dynamic typing is a process of assigning the data type of a variable at run time i.e. to create a variable a programmer only needs to include the name of the variable and the value of that variable. This to some extent reduces the programmers coding time. The other type of typing is static typing, this is when we give the type of a variable during declaration. Which means a programmer has to tell the compiler what type of data he wants and the compiler will handle the rest. To create a variable in this method a programmer will need to provide the compiler with the name of the data type, the name of the variable and the value of that variable. Even if using dynamic typing reduce the effort made by the programmer in this programming language static typing has been used. The reason behind it is it will help early detection of programming mistakes, help in creating more opportunities for compiler optimization. It also helps increase runtime efficiency by performing type check during compilation time.

The language also supports for increment and decrement operations. There are two different types of this kind of expression which are pre and post expression. The pre-increment and decrement operations are designated by a preceding ++ or – operator and an expression. This type of expression is evaluated before other expressions. The post-increment and decrement operations on the other hand have expression followed by ++ or --. This will shorten expressions which will otherwise make use of redundant operands to do a simple increment and decrement. This feature is added to make the language as simple as possible.

In addition to the afore mentioned reasons, as the language is intended to novice programmers or students it will teach them about data types and it will also help handle data type related errors before doing operation on the variable.

3.3 Construct Description

3.3.1 Types

Data types are a data which tells the compiler how to interpret and use a particular data. In programming there are different operation performed on a variable of a particular data type,

because of that it is wise to tell the compiler what type of data it is dealing with. Telling the compiler what type of data it is dealing with will decrease the number of errors handling a compiler should deal with. There are three basic data types in this language.

Basic types (መሰረታዊ መደብ)	
Integers	ቁጥር
Float	ነጥብ
Boolean	እዉነታ

These are the three data types. Each data type represents different data collections. Integers represent numbers. But there are two type of number system in Amharic language, one is the Hindu-Arabic number which ranges from 0 to 9 and Geez numerals which contains a range of numbers but it doesn't start with zero. Geez numbering system is not used as a numeral for instruction in any schools except religious schools. The geez number is mostly used to represent years and numberings. Because of that Geez numbers are not used for calculation purpose in this language. The limit to the size of integer is unlimited it is decided by the memory size of the operating system.

Floating number represents numbers that contains decimal numbers like 1.1, .4 . Floating numbers are represented in Amharic keyword 'ነጥብ'. This data type will be used to represent numbers with fractional parts (it is expressed with a decimal point). The other basic keyword is Boolean which represents a logic which can be represented as true or false. These keywords are represented in the Amharic programming language using 'እዉነታ'. These basic keywords represent Amharic statements that can be either 'እዉነት' or 'ሐሰት'. This basic type can be used to represent variables that can be used to hold expression which can be transferred to true or false.

There are other data types that can be create by the user using the basic data type like Integer. These other data types which can be created by the programmers are Strings and Lists. In this language we have avoided using char as a data type of its own because it will help remove one more keyword from the keywords list. Strings are data types that are used to store character sequence starting from one. Strings represent any and all characters of any type or language that are enclosed with in quotation mark. The other data type that can be

created by the programmers is Lists which is a data type that holds a list of things which is enclosed by this '{}' two delimiters. This will allow users to create their own list of things and use them the way they want.

3.3.2 Declaration

In this part of the document decision regarding different declaration parts of the language. It details with reason in deciding how to structure different declaration. Since the language is an intermediary language for other high-level languages the declarations are constructed to make them have a very close resemblance and at the same time become simple enough to be understood by a beginner.

Declarations introduce names in to a program, such as variable names, method names and class names. There are different attributes associated to declaration in addition to the name of the construct. The most common attribute related to declaration is type. Type will tell how the compiler reads the data associated with that specific name. Amharic programming language contain three basic parts: the first and smallest part of it are statements which ranges from variable declaration up to constructor declaration, the second one is methods, which represents operation of a class. Method declaration is different from variable declaration. The last and final part is class which encloses everything. Both properties and methods are enclosing inside of a method and can be accessed by using the statement class name dot property or operation names.

3.3.2.1 Class declarations

The class declaration component of this programming language declares the name of the class one more attribute which is optional, the class's superclass. The minimum and simple declaration of class must contain the 'ስብስብ' keywords and the name of the 'ስብስብ' that is being defined. The simplest declaration of a class looks like this

Class <class name> ስብስብ <የስብስቡ ስም>
<block> <ሌሎች>

For example, the following code snippet creates a class named calculations

Class calculations ስብስብ ስሌቶች
<block> <ሌሎች>

One thing to note here is there is not visible delimiter that separates the class from the rest

of the code. To make the looks of the code smooth to readers it has been decided that new line and INDENT's are used to separate one construct from the other. The end of a construct will be recognized with DEDENT's. It has been tried to handle whitespace differently in this language.

There is also one more attribute that can be include with a class declaration, it is the import attribute. The extend attribute will be used to include functionalities inside of another class. The extend declaration will take the form keyword + the parent class name.

class child extend parent. ስብስብ ወራሽ ውርስ አውራሽ

3.3.2.2 Constructor declaration

A class contains constructors that later will be used to create objects of different property and operations. Constructor declaration vary from languages to languages. Each language has their own constructor declaration signature. In this language it has been decided that the constructor only contains the name of the class and parameters. It is possible to declare a constructor more than once but the constructor must have a different parameter or else an error will be thrown by the compiler for declaring the same constructors. Here is an example of constructor declaration with one parameter

Class example ስብስብ ምስሌ
example (int x) ምሳሌ(ቁጥር ሀ)

In the following example there is a class name example and the class have a constructor called with one parameter. This constructor will be used to create objects, object creation will take the form

class object = new(parameters) ስብስብ_ስም ስም = አዲስ(parameter)

Here object creation only uses the "አዲስ" keyword. In this language polymorphism is not supported. The new keyword will take the parameters of the constructor.

3.3.2.3 Method declarations

The other construct that is represent in this language is Methods which designates the operation performed by an object. The method declaration component declares the return type of the method, the name which has to be different from other methods declared inside

the class and are not keywords, it also declares a bracketed list of parameters. A simple method declaration contains the return type, the name of the method and an empty bracket. A simple method declaration is shown below

<type> <name> () <አይነት> <ስም>()
<block <ሌሎች>

Methods also take parameters(ግብዓት) inside of the bracket and calculation can be performed. An example of a method declaration is given below with parameters

int add (int x, int y) ቁጥር ድምር (ቁጥር ሀ, ቁጥር ለ)
<block> <ሌሎች>

The above example contains a declaration of class which will be seen in the next topic. Like state above there are different return types that can be used when declaring including list. This will help code reusability and abstraction.

3.3.2.4 Variable declaration

In programming we include many variables with which we store a value that can change. In this language every object created has access to class variables. Variable declaration component contains the name declaration. The name declaration is preceded by the type of the variable which is followed by assign sign '=' then a value. A simple variable declaration is shown here

String name = "Ermias" ሐረግ ስም = "ኤርምያስ"

There is also another way of declaring variable, i.e. to only declare the name of the variable and the type of the variable. A declared variable can be instantiated later on the program. This will give the programmer flexibility of assigning variables depending on a specified condition. Here is how we declare a variable

int x ቁጥር x

3.3.3 Statements

There are different statements inside of a program: assignment statement, selection statement, repetition statement, control statement and input/output statement

3.3.3.1 Import statement

The first but optional statement written in programming using this language is the import statement. Importing is a very import part of programing the reason for that is importing

enables class to take advantage of functionalities that are in other class. When a program calls the import statement the compiler will look on a specified location to look for that particular module. Here is how import statement is written

import <file path> ተጠቀም <የፋይል አቅጣጫ>

3.3.3.2 Assignment statement

Assignment is a way of storing a value in a variable. Assignment statement uses the assignment operator to store a value to the specified variable. Assignment statement is part of the variable declaration which has two parts. Variable declaration and assignment can be written separately first a variable can be declared with the type specified with it and then the assignment will follow. If the two parts are in reverse order the compiler will issue assigning undeclared variable error.

variableName = expression ስም = "አበበ"

3.3.3.3 Selection statement

Selection statement is a statement that directs the flow a program in a specified direction based on whether a certain condition is true or false. There are different selection statements that are part of different programming language. The well-known selection statements are the if-else statement and switch statement. There is one way of handling selection statement in this language and it is done by using the keywords 'ከሆነ', 'ካልሆነግን' and 'ካልሆነ'. The selection statement always starts with the 'ከሆነ' keyword. A selection statement without the '' will be terminated during compilation. This selection statement looks a lot like this

(expression) if
    statement
(expression) elseif
    statement
else
    statement

A simple example which check age is given below

```
(እድሜ > 18) ከሆነ
        <አረፍተነገር>
ጨርስ
(እድሜ == 18) ከሆነግን
        <አረፍተነገር>
ጨርስ
ካልሆነ
        <አረፍተነገር>
ጨርስ
```

This means if age is greater than 18 execute the statement inside of it but if age equals 18 execute the statement inside the elseif but if the expression is not mate then else will be executed. In this case the reason the Amharic keyword for 'if' is at the end is because this is how selection statement are spoken and written in Amharic grammar. It could have come in the beginning of the sentence but because it is easy for the language to be understood by novice students if it mimics the spoken Amharic language.

3.3.3.4 Repetition statement

Repetition statements are statements that performs an action as long as a particular expression is true. It is used to repeat an operation for a certain period of time. In Amharic language we can use the "እስከሆነ" keyword to declare a repetition statement like if-else while statement has the same structure.

```
(ድምር > 0) እስከሆነ
        <ዓረፍተነገር>
```

This is to mean while sum is greater than zero perform a statement inside of it. The while loop must have a way of terminating itself or else it will cause infinity loop that will result in system malfunction.

3.4 Program Execution

In every programming language there is a place where program execution commences. Every statement and method to be executed in a program it has to be called inside of the main method or be inside of a method that is called in the main method. Unless that is the

case a method will not be executed. The overall program will not also start execution unless there is an entry point to the program execution.

In this language program execution starts when the compiler finds the method '*መጀምሪያ*' which signals the compiler to start executing the program. If there is nothing inside of the main program, the program will not return anything. The method that is used as an entry point to program execution is like any other method it can have everything method have. It can declare variables that can be only be used inside of the method, it can also declare selection statements, it can declare control statements. Statements inside of the main method will start executing automatically without the need to be called by other method.

```
መጀምሪያ()
  <ዓረፍተነገር>
```

In the above code snippet when the program finds '*መጀምሪያ*' statement, statements and method calls inside of it will be executed.

3.5 Compilation and Run-time Errors

Compile and Runtime error are programming term that shows different level of programming. The process of compiling a source code written in a particular programming language. An error that happen during compile time of a program is called compiler time error. The design of the language focuses on giving students who learn this programming a clear and helpful error message. Because the clearness of an error message will help programmers understand the error very easily and reduce the time spent in understanding the error and spent more time on solving the error.

3.5.1 Compilation errors

The proposal for handling compilation errors is done by developing a compiler which handles errors that occur during compilation. Compilation errors happen when the source code doesn't follow the proper syntax and semantics the languages grammar. A compile time error can happen because of missing brackets, missing comma or over use of delimiters, mistyping keywords and the likes.

```
ስብስብ ስህተት
  መጀመርያ()
    (ሀ == ለ ከሆነ
      ፃፍ(ሀ)
    ጨርስ
    ካልሆነ
      ፃፍ(ለ)
    ጨርስ
```

For example, in this piece of code there is a syntax error at line three, to display such an error it. It has been tried to make the information provided about the error as clear as possible and make it easily understandable. The compilation error information contains the number where the error occurred, what type of error happened and suggestion on how to fix it. At compile time the following message will be generated.

Error

In class Error at line
3: There is a missing bracket when declaring if
Here is how if statement is declared
(expression) if
    <statement>

ችግር

በ ስህተት ስብስብ ዉስጥ መስመር 3 ላይ ስህተት ተፈጥሯል፡
ከ ከሆነ ዓረፍተነገር ሲፃፍ የመዝጊያ ቅንፍ ይቀረዋል
ትክክለኛ ከሆነ ዓረፍተነገር ለመፃፍ ይህን መገድ ይከተሉ
    (እዉነት/ሐሰት) ከሆነ
        <ዓረፍት ነገር>

The error message will include the name of the class the error occurred because the error may be from the parent class in case of inheritance. The message also includes the line where the error occurred and description of the error that occurred. It also includes suggestion on how to fix the error.

3.5.2 Run-time errors

Runtime errors are programming errors that happen while the programming is running. Runtime errors encompasses a variety of more specific error's types such as IOErrors, logic errors, division by zero errors and other errors that are not compilation errors. The best and informative runtime error which is division by zero is used to show how this programming language handles runtime errors.

Here is a division code snippet

```
ቁጥር ክፋይ(ቁጥር ሀ, ቁጥር ለ)
    ቁጥር መ = ሀ / ለ

    መልስ(መ)
```

For instance, in the above code there is no problem in the syntax, if this code is compiled by the compiler there will not be an error because all the requirements for writing a valid syntax are met. But an error will occur when executing the code if we pass zero to the second parameter. This type of error is called division by zero.

Error	ችግር
In class name of class at line X:	በስብስብ የስብስብ ስም ዉስጥ መስመር መ ላይ፡
Divisionbyzero: denominator must be different from zero	በዜሮማካፈል፡ ታእታይ ከ ዜሮ የተለየ መሆን አለበት

Here the error message shows the class name and the line number where the error happened and the type of error happened and what to do about it.

3.6 Sample Parser and Tokenizer

The sample parser and tokenizer are written by hand using java. This sample is used to show how different constructs in the language works. Java's regular expression has been extensively used in to parse the source code provided.

Regular expression is a sequence of characters that is used to match other strings or sets of strings. It is a way of describing strings in a general term (Bruce Eckel, 2003) In the sample it has been tried to show how the language will be read by a compiler, how the source code is broken in to small but understandable pieces of text, how the tokenized data are used by the parser to identify the different constructs of the language.

The sample also have an interactive code editor written using JavaFx. The code editor is used to stream the necessary codes to the debugger. The code editor extensively used JavaFx library but there are also other libraries used in addition to JavaFx. One of the nice parts of

using JavaFx is its capability let programmer use cascading style sheet (CSS) to style text and background. The editor has also used cascading style sheet to highlight the source code.

CHAPTER 4
EVALUATION AND RESULTS

4.1 Grammar

The final result of the design of the Amharic programming language is the grammar of the language. A lot of design decisions have been made to make sure the languages grammar is all encompassing and easy to implement. After all the decisions made on the design of the language that is the structure of the languages constructs the grammar of the language has been written by using Extended Backus-Naur Form. The full Grammar Specification has been included in this document in Appendix 1 which contains both the Amharic and English version of the grammar.

The result part of this document has a couple of grammar snippets to show how the grammar of the language has been structured. The grammar of the language has been tested for unwanted and unused production rules by using a language translation tool called ANTLR (Another tool for Language Recognition). Using ANTLR the validity of the grammar has been tested and a parser and lexer has been generated.

The grammar of the language consists two parts. The first part which is parser grammar contains the production rules of the language and the second part which contains the datatype of the language is the lexer grammar. Unused production rules have been found on the grammar of the language. Unused production rules were identified by ANTLR. All identified unused production rules have been dealt with and the grammar are made to contain only necessary production rules.

The grammar is written using EBNF, a meta syntax used to write context free grammars. The grammar is then passed to an automated grammar translator called ANTLR which takes grammar and translate it to parser and lexer. The grammar doesn't support backtracking because backtracking makes debugging a grammar hard by parsing the same input multiple

times. The other reason for it is it doesn't give a good and instructive error message when given invalid input.

The top level of the grammar is the class declaration of the language. As class is a parent block that holds all other constructs of the language the class declaration grammar specification a good way to show the result of the design. The following grammar snippet shows how the class declaration of the language has been handled.

```
classDecl : CLASS Identifier   superClass?  classBody;

superClass : EXTENDS Identifier;

classBody :  classBodyDecl*;

classBodyDecl : mememberDecl | staticInitializer
              | constructorDecl;

mememberDecl : fieldDecl
             | methodDecl
             | arrayDecl;
```

The above grammar snippet shows how a class has been structed in the grammar by using Extended Backus-Naur Form. The class declaration production contains the name of the class the identifier and to operational non-terminal symbol.

Operational precedence handling is part of the grammar. ANTLR v4 handles the operation precedence in the Grammar with greater accuracy. By using left recursion operation precedence was handled in a neat way. The grammar snippet of addition and multiplication statements is provided here to show how operator precedence work.

```
addExpr : multExpr
      | addExpr ADD multExpr
      | addExpr SUB multExpr;

multExpr : unaryExpr
       | multExpr MUL unaryExpr
       | multExpr DIV unaryExpr
       | multExpr MOD unaryExpr;
```

as shown in this grammar the multiplication expression takes

Below is a screenshot of a sample operation and the operation precedence displayed by using parse tree. For the operation 1 + 2 * 3, the parser handles the precedence easily like shown in the following figure

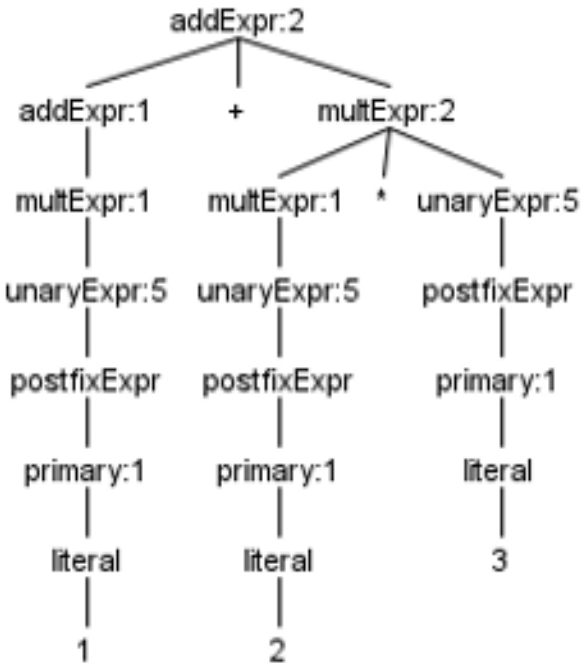

Figure 4.1: Parse tree for the operation 1+2*3

The above figure shows the syntax tree of the operation 1 + 2 * 3. As it's shown in the figure the parsers first calculated the product of 2 and 3 because as stated in the grammar snippet the multiplication operation takes precedence. The parser then takes the result of the product and add it with 1 and return the correct result.

4.2 Parse Time

In this part of the document the time it took to parse different part of the different constructs of the language. The parse time of the constructs has been tested by using ANTLR's preview tool. A sample code for each of the constructs are given both in English and Amharic and the parse time with the syntax tree of the code are given below

In the following the parse time of a class with two variable and a single addition method is given. The parse time is calculated by using the number of tokens that are the result of the lexer

```
class test
   int x = 4
   int y = 3

   int sum ()
      int w = x + y
      return w
```

```
ስብስብ ሙከር
   ቁጥር ሀ = 3
  ቁጥር ለ = 4

  ቁጥር ድምር ()
     ቁጥር መ = ሀ + ለ
     መልስ
```

The above example is a class declaration that contains to variable declaration and one method that returns int and a declaration and assignment of declaration. In the above example the code has been tokenized and the parser has parsed the code. The time it took to parse the code given by the generated parser is 11ms. Prediction time of a parser is the time a take to predict the next token. The average prediction time in the following code snippet is 12ms.

The other language construct that is include is If else language construct which is a bit different form existing if else statements because the if keyword appear at the end of the expression. The parse time of a sample if else code is given here. The prediction time of the code has also been calculated.

```
(x > 3) if
   ++x
end
else
   --x
end
```

```
(ሀ > 3) ከሆነ
   ++ሀ
ጨርስ
ካልሆነ
   --ሀ
ጨርስ
```

The above if else code has been given to ANTLR grammar tester. The previewer then tokenized the code and calculate the time it took to parse the language construct. The average prediction time has also been calculated in addition to the parse time. The parse time according to ANTLR previewer is 10ms and the prediction time is 6ms.

Even if the parse time is not the measure of the overall compile time but knowing the parse time help us to optimize the compile time of the language.

4.3 Criteria

Most of the design principles mentioned in this document are not quantitatively measurable. Because of that it was hard to measure the design of a programming language quantitatively. This part of the document contains the evaluation of the design of the programming language

by comparing the grammar of the language with other existing programming languages. There are different aspects of the designed programming language that has been the center of evaluation. The evaluation focused on the design criteria's that can affect the language in the long run, principles that make the language easy to use and implement.

To evaluate the language, the languages prototype of basic constructs has been implemented and used. The evaluation has been done during the implementation of a prototype of the language. The aim of the evaluation was to figure out how much and how well the design criteria are met?

The design criteria test can be broken down into the following questions: how readability is the in comparison with other existing programming language? What functionalities make the language hard to write? How simple are the language features? How delimiters affected parsing? As there are no preliminary version of the language that can be used to evaluate the language, the language is evaluated by trying to implement the tokenizer and parser of the language. This and other questions were the focus of the evaluation and the design problems and the fix that has been done are included in this chapter.

4.4 Simplicity

The design has been evaluated for simplicity and modification have been made to solve some of the design problems and complexity. The design problems and complexities are found when implementing a sample for the programming language. Simplicity of the language can be achieved by a compromise between the factors that enhance readability and those features that enhance the writability of the language. The language grammar also has tried to achieve high expressiveness to make it easy to use. It also helps reading and writing of the language easier. The support of greater abstraction helps the simplicity of the language by not redundantly writing codes.

4.5 Readability

Most of a programmer's time is spent by reading a program written themselves and by other programs when codes are being written and are being debugged. So, the time a programmer spent by reading a program written by other programmers shouldn't be high because the

time spent for reading a should be used by the programmer to do what they want. If a program written by other programmers a difficult to read and understand it will make incorporating it to one's own code becomes difficult.

So, during evaluation factors that affect the readability of the language has been identified and modifications has been made on the grammar of the language to make sure programs written in the language has a better readability.

The first Readability problem identified was the first design decision made on if else clauses. In the first design of if else clause the bracket that encloses the expression was left out of the grammar to reduce the error that occur due to unmatched brackets. But the absence of the brackets greatly affects the readability of the language. The brackets were first of all left on purpose to help writability and avoid syntax error because of mismatched brackets.

The other issue with respect to if else clause was the position of the "ከሆነ" keyword. First it was conceived for reason to smooth transition to other high-level languages positioning the "ከሆነ" keyword before the expression. But comparing this design decision with other programming languages it is identified that this decision will affect the readability and easy understandability of the language because this structure is in direction opposition of how the Amharic language uses conditional expression. All most all high-level programming languages up to date tried to mimic the language used to base their keyword.

The other factor that affects readability of the language the unrestricted length of identifiers. The reason the length of identifiers is unrestricted is because programmers should be free when choosing identifiers. Comments also affect the readability of the syntax specially comments that are inside methods and other blocks which eclipse the important structures.

Declaration and instantiation also contribute to problem in readability. Writing declaration and instantiation separately will affect readability and also make it hard for beginners to understand. The extensive use of indents and dedents are included to help readability if the source code. Almost all constructs in this Amharic programming language use indents to separate themselves from other constructs and to enclose their members. The indent will help structure the code and help improve the readability by avoiding unnecessary delimiters.

There is an exception when it comes too if else statements. If else statements use the end keyword to terminate themselves. The reason if else statements are made to use a terminating keyword is because it helps for making the readability better by giving the block a separation so that the source code avoid confusion between parts of the constructs. The other reason for making it this way is to make implementation easier.

4.6 Writability

Writability is a measure of easy is a source code to write. During evaluation it's found that the writability of the language was not only affected by the languages of the grammar but also the Amharic language input tools. Currently available input tools make a bit hard to write. Some of the available language input tools are PowerGeez, Google input tools and others. Google input tools uses transliteration which uses English letters to write Amharic words.

Writability of the language can be affected by the use of concise and regular syntactic structure. Concise and regular syntactic structures help the writability of the language. The grammar also makes some things optional so that structure of the syntax can be flexible and programmers can leave some parts to make writing and reading the source code easier.

Having simple statements also affects the languages writability. The programmer contains statements that are simple which makes the writability and readability if the language better. Removing delimiters that are not necessary. Delimiters are replaced by white space this makes writing in this language will be easier and readability better.

Evaluation also identify the presence of redundant structure affects the writability of the language. Due to that the language also tried to avoid redundancy in the languages grammar as such making the language concise.

Having limited length of identifiers greatly helps the readability if the language. But this part is left for the programmer to handle. The reason for this is user should not be affected by the restrictions imposed on identifiers. They should be free to choose any length of identifier as long as it is what is best for the programmer they write. When conflicts happen the one item that matters the most must be given priority over the other item. There is only guidance for

choosing identifiers, where users are able to choose appropriate identifiers to help readability and writability.

The keywords chosen and included in the grammar of the language are words that do not include a letter(ፊደል) that are difficult to be written by using the currently available Amharic language input tools. The reason this affects writability of the language is that there are letters in Amharic which is difficult to write and as such time consuming.

In addition to readability of the language, Amharic punctuation marks are left unused to avoid writability issues. One punctuation mark that is used in this language is the Amharic equivalent for full stop '።'. Data types of a method can also be left unspecified to help both the readability and writability of the language.

4.7 Expressiveness

The expressiveness of the language will also affect the simplicity, readability and writability of the language. The evaluation of the expressiveness of the language was also done by allowing the team to write things they want to do. But without the availability of built-in libraries it has been difficult to figure out the expressiveness of the language. The language's grammar has the necessary constructs to express things.

The grammar supports arrays which allow users to express a list of objects as easily as possible. Arrays are expressed using the keyword "ድርድር" which allow the creation of a list of objects. when creating arrays, the type of the array and the size of the array must be first specified. When creating objects 'Arrays are a curly brace enclosed list of objects separated by comma. Arrays can be accessed by using the index of the elements inside the array

Expressiveness also evaluated with respect to already existing programming languages that are widely used in the programming world. The grammar of the language contains most of the features used in existing programming languages. There are some features that are left to avoid complication when using the languages to teach in primary school. The grammar only included one of the looping ways which is while loop. Since it is possible to express all necessary loops by using while loop.

The language also enables to create objects by using class. It also enables to express the operations that can be performed on the objects. It also enables to express inheritance of previous declared class. This enables class to share properties and operation instead of declaring them over and over again. This will help the reuse of object which will increase the abstraction of the language. There also an import functionality which will enable to reuse codes written and compiled by other programmers.

4.8 Efficiency of Implementation

To assist the efficiency of the implementation of the language and to not make compiling slow features that are known to be inefficient are identified during evaluation. Left recursion in syntax description of a programming language which is the grammar sometimes leads to an endless loop. Since including left recursion in the grammar of the language results in an endless loop which greatly affects the efficiency of the implementation efforts has been made to avoid left recursion as much as possible but when it is must to include them ANTR v4 will handle it and make sure that the parser doesn't stumble into a loop that it can get out off.

Production rules are rules which handles how we replace non-terminal symbol with a combination of terminal and non-terminal symbols. Even if production rules in general are as useful as it comes there are times where a useless production included in the grammar of a programming language. The presence of useless production rules increases the parsing time of the compiler as such making the language in general slow compared to when there is no useless production included in the grammar. The evaluation has indicated the presence of useless production rules which affect the efficiency of the parser. All useless production rules found during evaluation has been removed from the grammar.

To help the efficiency of the language the declaration of variables includes the type of value the variable hold. This is called static typing. static typing helps type checking be handled during compilation of the source code which in turn makes the runtime faster. Static typing is chosen in expense of readability and writability.

There are a huge set of data structures in money programing languages. Handling this huge sets of data structure takes time to process. As a programming language intended to be given as a first programming language it is necessary to include a must have data structure sees in the language. This makes handling the data structure faster as such making the program faster.

4.9 Abstraction

Abstraction is an important part of programming especially object-oriented paradigm. Abstraction help the programmer access functionalities without the need to know the nitty-gritty of how its implemented internally. Abstraction hides all data inside the structure except the data that are relevant in order to reduce complexity of the program and increase the efficiency.

There are different levels of abstraction included in Amharic programming language. The highest level of abstraction is classes. Abstraction helps the readability and writability of the language by hiding implementation details of programing constructs.

4.10 Keyword

Keywords also known as reserved words are identifiers that are frequently used words when programming. So, the choice of keywords was part of the readability and writability test of the language. Choosing keywords that are easy to write and to read as part of the Amharic programming language.

4.10.1 Sample

In this part of the document, a sample program written by Amharic programming language and debugged by using a prototype debugger. The sample program is a hello world program written in the simplest combination of constructs. This part in addition to the source code program contains the tokens and parse tree. The parse tree is for the English version of the grammar. This part also contains the screenshot of a prototype of the programming language debugger. The debugger contains three parts the side bar which shows the list of files in the particular directory, the code editor part which also has a code highlighting feature is used

to write the source code of the language to the debugger so that it is examined for conformity two the grammar of the language, the last and final part of the debugger is a console to show the output of the code after.

The debugger doesn't contain a code generation feature. The debugger is used to show how a piece of code is tokenized and parsed. It has not been used to test user interaction because it only contains a couple of constructers implemented and it is deemed ineffective to test with just only a couple of constructs. The source code written in the code editor is passed to the debugger's tokenizer so that the tokens in the program are identified and passed to the parser. The parser then checks if the tokens identified by the tokenizer makes up a meaningful construct. Then it is interpreted by using Java to show how the result are displayed. Following is a screen shoot of the sample hello world program an English equivalent of the source code is given below.

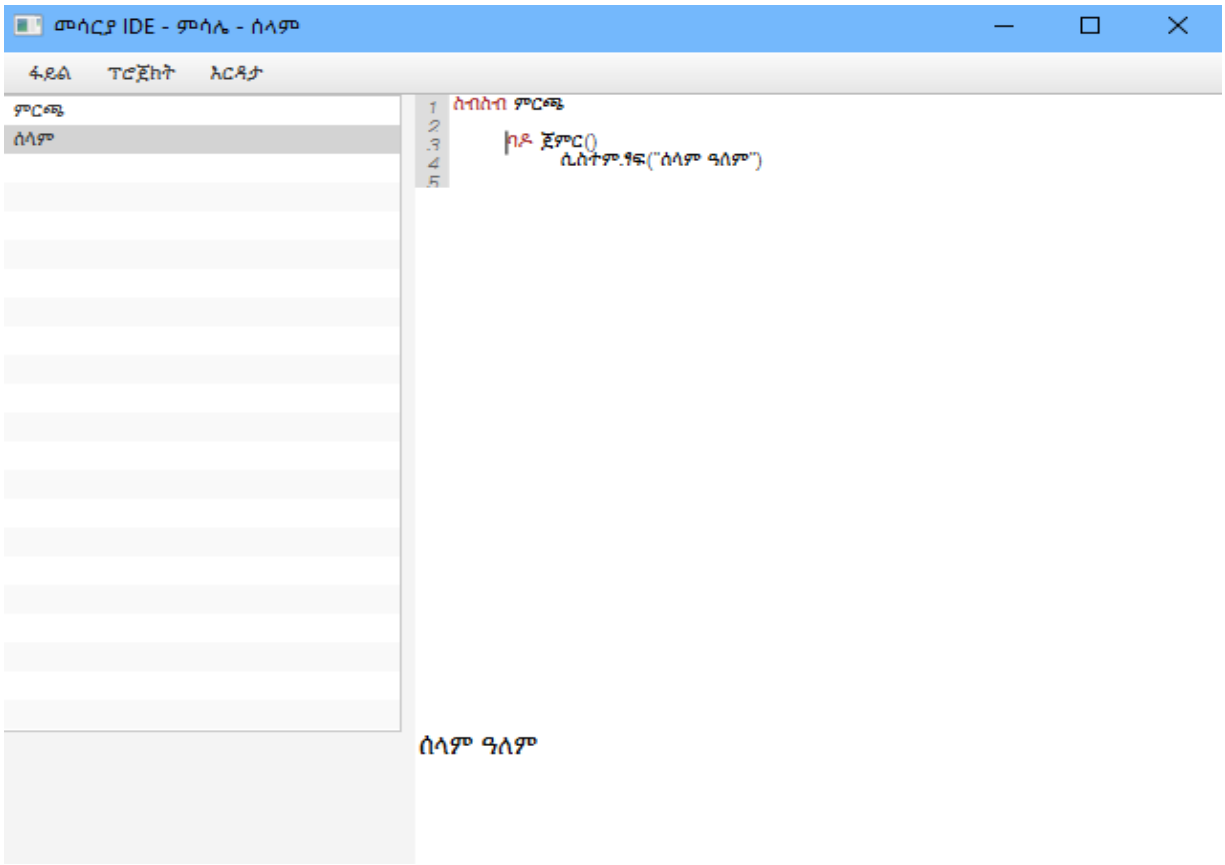

Figure 4.2: Hello world sample program written in Amharic programming Language

The figure shows a hello world sample code written using the prototype. In this sample code there are different levels of parsing class declaration, method declaration and method invocation parsing. The prototype takes a couple of steps to execute this sample code. The English equivalent of for the code in the figure is given here

```
class variable
    void start ()
        write (“Hello world”)
```

4.11 Parse Tree

The parse tree of a code shows the hierarchical structure of the tokens. It’s a tree that represents the syntactic structure of a text based on a context free grammar. The root of the syntax tree is the always the start production rule which shows the start of the parsing. After the start root node there are two child nodes which encloses the rest of the child nodes. The first child nodes of the start production rule are the import production rules which contains the paths of the import class as a child. The second child node of the start production rule is the class Declaration production rule which contains the class declaration and everything in it.

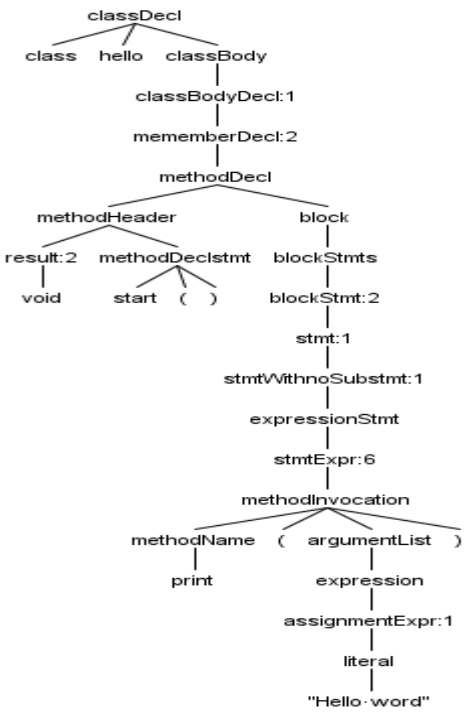

Figure 4.3: Parse tree for English version of hello world program

This figure shows the hierarchy of the hello world program. The parser takes the code and checks if it matches the grammar rules of the language. This figure for example shows the code starts with the class keyword followed by the class names. As the class follows the correct grammar rules the syntax tree doesn't show any error.

The other sample included here is a program written to show how if else is implemented in Amharic programming language. If lese statements are constructed so that it has a resemblance to the Amharic language. In Amharic language the expression to be evaluated comes before the if keyword. Overall the language is made to look like the formal Amharic language.

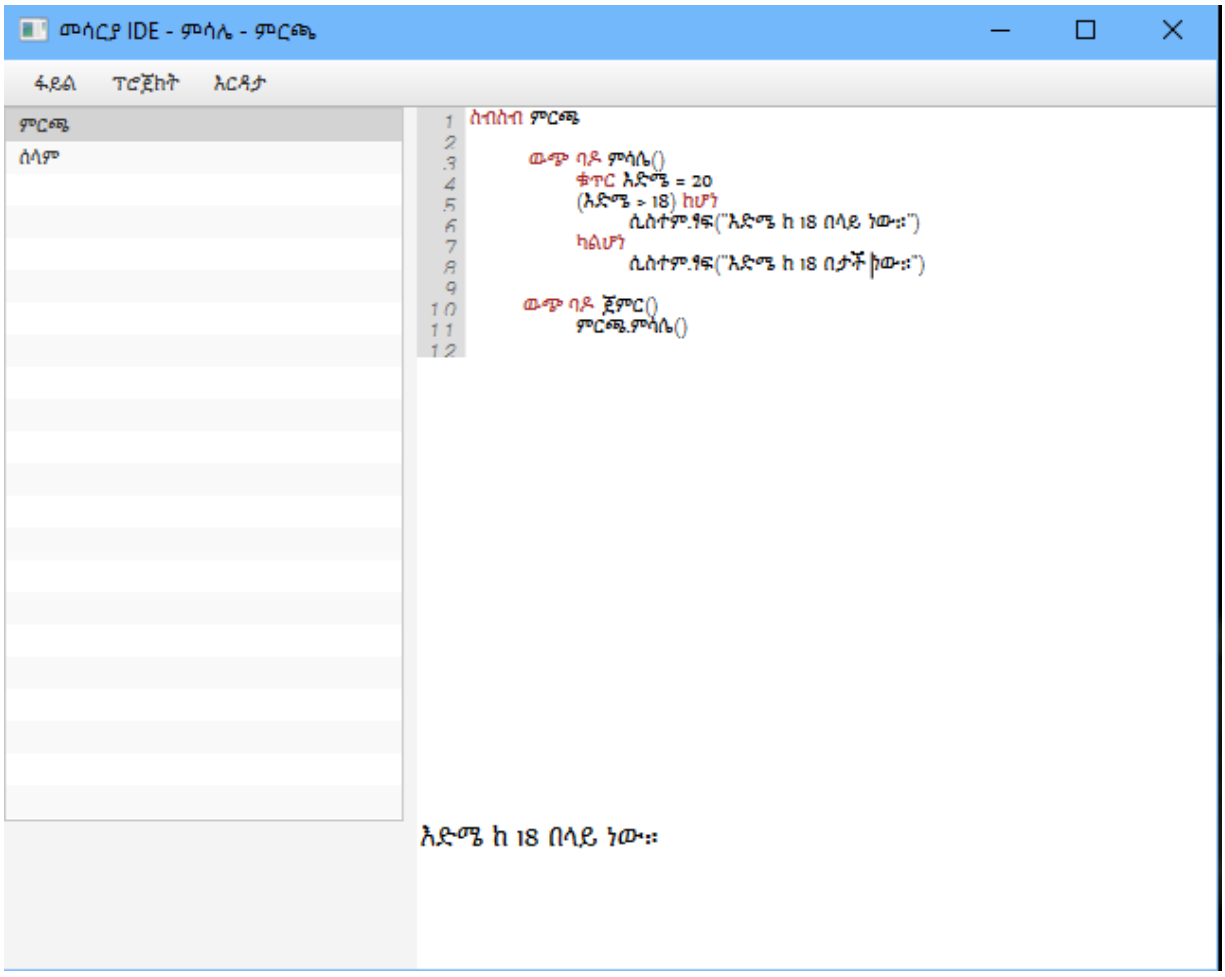

Figure 4.4: If Else expression written using Amharic language and debugged using the sample debugger

The if else expression example above shows how an if else statement is structured in Amharic programming language. The above code snippet contains the declaration and instantiation of a variable እድሜ. If else declaration as shown shows an expression a head of the if keyword.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Computers have become a very essential part in our day to day activities. They help accomplish tasks in a fast, better and effective manner. We humans use programming languages to instruct computers to do specific tasks. Programming is a way of instructing a computer to do tasks. Most Programming languages are human understandable to make them easy to program.

Programming language is an essential area in computer science. To advance the prevalence of Amharic language in Technology related subjects, creating an Amharic programming language is necessary because it is the fundamentals of computer science and almost everything technology related areas are dependent on computers. Hence, leaning the essential part of computer science using one's mother tongue help understand the concept of programming better. In addition to that students who learn programming using their native language tend to learn programming concepts faster and better than those who learn using foreign language.

In this paper an investigation has been conducted on the necessity of programming using one's native language. An extensive research has been done on non-English programming languages. Programming languages that are created for Arabic, Chinese and Spanish speaking countries are included in the research. The document includes a review of Extended Backus-Naur Form which is used to write the grammar of the language. The design principle which are the criteria's of designing a programming language are included.

The design of Amharic programming language contains a keyword chosen by translating keywords in other popular programming languages and choosing words based what they do not direct translation to make them. The grammar of the language contains production rules that signifies the different programming constructs.

In this thesis a grammar translator has been used to translate the grammar written by using EBNF to their respective tokens and parser which recognize the different constructs in the language. talks about the design of a programing language using keywords based on the official working language of Ethiopian, Amharic. The paper focused on the design of the programming language and the lexer and the parser part of the compiler.

This paper also has a simple prototype of the debugger of the language used for evaluating the designed programming language. The prototype contains the basic constructs of the programming language like declaring a class, method and variables

5.2 Recommendation

Now a time, countries are engaged in advancing computer related fields of study which are the corner stone of a country's development and security. Ethiopia is a lot behind in programming and computer science that has a toll in the development of the country's economy. To help improve that it should engage extensively in advancing the science and technology specially in computer programming.

This paper contains a design of an Amharic programing language other researchers who wants to implement this design can use the following recommendations. There are two implementation of the compiler one is trans compiler which translates the program written in one language to another language which has its own compiler. The program will be translated into equivalent of other programs and then it can be transformed into machine without the need to write an independent compiler for the program. The drawback of this approach is that the machine in addition to the performance of the language itself will also inherit the performance of the other program as such making the language slower.

The other approach to implement the compiler of the language is to develop the compiler itself from scratch. This paper included the first part of the compiler development which tokenizing and parsing of the grammar of the language.

5.3 Future Work

The focus of the thesis is to design a programming language that uses keywords based on the Amharic language and the implementation of the tokenizer and the parser. Future works

will include the full implementation of the compiler of the language with a rich integrated development environment which helps the development of programs.

The compiler of the language will be written in a program that will help the speed of the language and to make it portable so that it can be used by users who work on different operating systems. The language will also be improved and grorw to support functionalities that are not part of the original design of the program.

To make it also easy to use and help the advancement of the language future works will include a future in the language that will help students to program by using only dragging and dropping future.

REFERENCES

Al-A'Ali, M., & Hamid, M. (1995). Design of an arabic programming language (ARABLAN). *Computer Languages*, *21*(3–4), 191–201. https://doi.org/10.1016/0096-0551(95)00006-2

Allain, A. (2013). *Jumping into c++*.

Arnold, K., Gosling, J., & Holmes, D. (2013). *The Java programing language*. *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Asfawwesen, D. (2016). The Inceptive Construction And Associated Topics In Amharic And Related Languages.

Bekele, D. (2001). Impact of Government Policies on the Development of ICT in Ethiopia. *International Conference on African Development Archives*. Retrieved from http://scholarworks.wmich.edu/africancenter_icad_archive/17

Bingöl, O., Küçüksille, E. U., & Kuru, İ. (2018). Chameleon Turkish Programming Language. *European Journal of Science and Technology*, (14), 77–82. https://doi.org/10.31590/ejosat.442334

Bird, R., & Wadler, P. (1988). *Introduction to Functional Programming*. *System*. Prentice Hal International(UK). Retrieved from http://www.di.uminho.pt/~jas/Teaching/Courses/mp3/06-07/tp.pdf

Brian W. Kernighan, D. M. R. (1978). *The C Programming Language* (2nd ed.). Prentice Hall.

Bruce Eckel. (2003). *Thinking in Java , 3 rd Edition*.

Cowell, J., & Hussain, F. (2003). Amharic character recognition using a fast signature based algorithm. *Proceedings of the International Conference on Information Visualisation*, *2003-Janua*, 384–389. https://doi.org/10.1109/IV.2003.1218014

Dasgupta, S., & Hill, B. M. (2017). Learning to Code in Localized Programming Languages. *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale - L@S '17*, 33–39. https://doi.org/10.1145/3051457.3051464

EthioCloud. (n.d.). AxumLight. Retrieved March 4, 2019, from https://www.ethiocloud.com/axumlight.aspx

Fernández-Villaverde, J., Guerrón, P., & Valencia, D. . . (2018). Programming Paradigm. Retrieved March 3, 2019, from https://www.sas.upenn.edu/ ~jesusfv/ Lecture_HPC_7_Programming_Paradigms.pdf%0A

Feyman, R. (n.d.). EBNF: A Notation to Describe Syntax. Retrieved March 2, 2019, from https://www.ics.uci.edu/~pattis/ICS-33/lectures/ebnf.pdf

Georgatos. (2002). " How applicable is Python as first computer language for teaching programming Fotis Georgatos, (June).

Gezmu, A. M., Seyoum, B. E., Gasser, M., & Nürnberger, A. (2018). Contemporary Amharic Corpus : Automatically Morpho-Syntactically Tagged Amharic Corpus. *Proceedings of the First Workshop on Linguistic Resources for Natural Language Processing*, 65–70.

Gurbani, K., Shelar, S. R., & Jitesh, P. (2008). *Imperative Programming*. Himalaya Publishing House.

Kay, A. C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices*, *28*(3), 69–95. https://doi.org/10.1145/155360.155364

Kendal, S. (2009). *Object Oriented Programming using Java*. Simon Kendal & Ventus Publishing.

Kirsal Ever, Y., & Dimililer, K. (2018). The effectiveness of a new classification system in higher education as a new e-learning tool. *Quality and Quantity*, *52*(s1), 573–582. https://doi.org/10.1007/s11135-017-0636-y

Kitchenham, B., & Carn, R. (1990). *Psychology of Programming*. *Psychology of Programming*. https://doi.org/10.1016/B978-0-12-350772-3.50022-7

Liang, Y. D. (2013). *Java Programming Comprehensive Version* (9th ed.). PEARSON.

Lindholm, T., & et al. (2015). *The Java Virtual Machine Specification Java SE 8 Edition*. *Managing*. Retrieved from http://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

Lindstorm, G. (2005). Programming with Python. *IT Professional*, *7*(5), 10–16. https://doi.org/10.1109/MITP.2005.120

Louden, K. C., & Lambert, K. A. (2011). *Programming Languages: Principles and Practices* (3rded.). Retrieved from http://books.google.com/books?id=6MOiYFg1DoIC&pgis=1

MacLennan, B. J. (1986). *Principles of Programming Language*. (R. and W. Holt, Ed.) (2nd ed.).

Maurizio, G., & Simone, M. (2012). *Programming Languages : Principles and Paradigms*. *Saudi Med J* (Vol. 33). https://doi.org/10.1073/pnas.0703993104

Nasser, R. (2012). The قلب Programming Language. Retrieved March 4, 2019, from https://github.com/nasser/---

Negesse, F., & Ado, D. (2016). visual recognition of graphic variants of amharic letters : psycholinguistic experiments, *8*(1), 173–200.

Nørmarks, K. (2011). Programming Paradigm. Retrieved March 3, 2019, from http://people.cs.aau.dk/~normark/prog3-03/pdf/paradigms.pdf

Papaioannou, S. K., & Dimelis, S. P. (2007). Information Technology as a Factor of Economic Development: Evidence from Developed and Developing Countries. *Economics of Innovation and New Technology*, *16*(3), 179–194. https://doi.org/10.1080/10438590600661889

Parr, T, & Fisher, K. (2011). LL(*): The foundation of the ANTLR parser generator. *SIGPLAN Not. (USA)*, *46*(6), 425–436.

Parr, Terence. (2013). *The Definitive ANTLR 4 Reference*. *Climate Change 2013 - The Physical Science Basis*. https://doi.org/10.1088/1751-8113/44/8/085201

Pfenning, F. (2006). Logic Programming. Retrieved March 4, 2019, from https://www.cs.cmu.edu/~fp/courses/lp/lectures/lp-all.pdf

Ritchie, D. M. (2005). The development of the C language. *ACM SIGPLAN Notices*, *28*(3), 201–208. https://doi.org/10.1145/155360.155580

Şehitoğlu, O. . (2008). Programming Languages: Logic paradigm. Retrieved March 3, 2019, from http://ocw.metu.edu.tr/pluginfile.php/2986/mod_resource/content/0/lectures/14-lp-paradigm.pdf

Sipser, M. (2012). *Introduction to the Theory of Computation*. (M. Lee, Ed.) (3rd ed.). Cengage Learning.

Terrence W. Pratt, M. V. Z. (2000). *Programming Languages: Design and Implementation* (4th ed.). Prentice Hall.

Tio, H., & Niekerk, V. (2001). introductioin, (1974), 1999–2001.

Topley, K. (2011). *JavaFX ™ Developer's Guide*. Addison-Wesley.

Vujošević-Janičić, M., & Tošić, D. (2008). The Role of Programming Paradigms in the First Programming Courses. *The Teaching of Mathematics*, *11*(2), 63–83.

Walter, S. L. (2012). Mother Tongue-based Education in Developing Countries : Some emerging insights, (February), 1–25.

Wegner, P. (1987). Dimensions of Object-Based Language Design, *10*(3), 168–182. Retrieved from http://dl.acm.org/citation.cfm?id=38823

Zegiestowsky, A. M. (2017). Tango : A Spanish-Based Programming Language Tango, *3*.

Zhang, P. (2017). Chinese Programming Language : Practice and Context, 1–27.

APPENDIX 1
GRAMMAR

The following section contains the full grammar specification of the programming language that has been the focus of study in this thesis.

```
primitiveType : numericType |
BOOLEAN;

numericType : integerType | floatingType;

integerType : INT;

floatingType:  DOUBLE;

expressionName : Identifier
                | ambigousName
                DOT Identifier;

methodName : Identifier;

ambigousName : Identifier |
ambigousName DOT Identifier;

/* production unit*/
start : importDeclaration*
classDeclaration* EOF;

importDeclaration : IMPORT Identifier ('.'
Identifier)* NEWLINE;
```

```
መሠረታዊዓይነት ፡ ቁጥራዊዓይነት | እወነት;

ቁጥራዊዓይነት ፡ ሙሉቁጥርዓይነት | ክፍልፋይዓይነት;

ሙሉቁጥርዓይነት ፡ ቁጥር;

ክፍልፋይዓይነት ፡ ነጥብ;

የትዕዛዝስም ፡ መለያ
            | አወዛጋቢስም ‘.’ መለያ;

የትዕዛዝስም ፡ መለያ;

አውዛጋቢያስም ፡ መለያ | አውዛጋቢያስም ‘.’ መለያ;

መጀመርያ ፡ ተጠቀምትዕዛዝ* | ስብስብትዕዛዝ*
EOF;

ተጠቀምትዕዛዝ : ተጠቀም መለያ(‘.’ መለያ)*
ዓመስመር;
```

APPENDIX 1 Continued

```
/* production from class */

classDeclaration : CLASS Identifier
superClass?  classBody;

superClass : EXTENDS Identifier
NEWLINE;
classBody :  INDENT
classBodyDeclaration* DEDENT
NEWLINE;
classBodyDeclaration :
classMememberDeclaration
                | staticInitializer
                | constructorDeclaration;

classMememberDeclaration :
fieldDeclaration
                | methodDeclaration
                | arrayDecl;
fieldDeclaration : varType
variableDeclarationList;

variableDeclarationList : variableDecl
(COMMA variableDecl)*;
variableDecl : variableId (ASSIGN
variableInitializer)?;
variableId :  Identifier dims?;

variableInitializer : expression;
```

```
ስብስብትዕዛዝ ፡ ስብስብ መለያ አዉራሽስብስብ?
ስብስቦች;

አዉራሽስብስብ፡ ዉርስ መለያ ዓመስመር;

ስብስቦች፡ ገባ ስብስብዝርዝር* ወጣ ዓመስመር;;


ስብስብዝርዝር ፡ ስብስብዓባላት
                | ፍፀምአፀፃፍ
                | መስራችአፀፃፍ;

ስብስብዓባላት:
                ዓባላትአፀፃፍ
                | ትዕዛዝፅተፃፍ
                | ድርድርአፀጻፍ;


ዓባላትአፀፃፍ ፡ ተለዋዋጭዓይነት
ተለዋዋጭአፀፃፍዝርዝር;

ተለዋዋጭአፀፃፍዝርዝር፡ ተለዋዋጭአፀፃፍ(‘,’
ተለዋዋጭአፀፃፍ)*;
ተለዋዋጭአፀፃፍ : ተለዋዋጭመለያ  (‘=’
ተለዋዋጭመሰየምያ);
ተለዋዋጭመለያ  ፡ መለያ (‘[‘’]’)?;
ተለዋዋጭመሰየምያ : ትዕዛዞች;
```

APPENDIX 1 Continued

varType : unPrimitiveType | unReferenceType;	ተለዋዋጭመለያ ፡ የመሰረታዊዓይነት | የተወካይአይነት;
unPrimitiveType : numericType | BOOLEAN;	የመሰረታዊዓይነት ፡ ቁጥራዊዓይነት | እዉነታ;
unReferenceType : referenceType;	የተወካይአይነት ፡ ዉክልዓይነት;
referenceType : classType | typeVariable | arrayType;	ዉክልዓይነት ፡ ስብስባይነት | ተለዋዋጭዓይነት |ድርድርዓይነት;
classType : Identifier typeArguments;	ስብስብዓይነት ፡ መለያ መተክያዓይነት;
typeVariable : Identifier;	ተለዋዋጭዓይነት ፡ መለያ;
arrayType : primitiveType dims | classType dims | typeVariable dims;	ድርድርዓይነት ፡ መሠረታዊዓይነት ’[‘’]’ | ስብስብዓይነት ’[‘’]’ | ተለዋውጭዓይነት ’[‘’]’;
dims : **'[']'** (**'[']'**)*;	
typeArguments : referenceType (COMMA referenceType)*;	መተክያዓይነት ፡ ውክልዓይነት (‘,’ ውክልዓይነት);
methodDeclaration : methodHeader block;	ትዕዛዝአፀፃፍ ፡የትዕዛዝአራስጌ | የትእዛዝጥርቅሞ;
methodHeader : result methodDecl;	የትእዛዝአራስጌ : ውጤት ትእዛዝአፀፃፍ;
result : varType | VOID;	ውጤት : ተለዋዋጭአይነት | ባዶ;
methodDecl : Identifier LPAREN formalParameterList? RPAREN;	ትእዛዝአፃፃፍ ፡ መለያ ግቅንፍ ቦታመያጣዎችዝርዝር? ቀቅንፉ;

APPENDIX 1 Continued

formalParameterList : formalParameters COMMA formalParameters |formalParameters;	ቦታመያጣዎችዝርዝር ፡ ቦታመያጣዎች ‘,’ ቦታመያጣዎች | ቦታመያጣዎች;
formalParameters : formalParameter (**','** formalParameter)*;	ቦታመያጣዎች : ቦታመያጣ (',‘ ቦታመያጣ)*;
formalParameter : varType variableId;	ቦታመያጣ ፡ ተለዋዋጭዓይነት ተለዋዋጭመለያ;
staticInitializer : '**static**' block;	ፍፁምአፀፃፍ ፡ ‘ፍፁም’ የትእዛዝጥርቅሞ;
constructorDeclaration : constructorDecl constructorBody;	መስራችአፀፃፍ ፡ መስራችእራስጌ መስራችአካል;
constructorDecl : Identifier LPAREN formalParameterList? RPAREN;	መስራችእራስጌ ፡ መለያ ግቅንፍ ቦታመያጣዎችዝርዝር? ቀቅንፍ;
constructorBody : INDENT blockStmts? DEDENT;	መስራችአካል ፡ ገብ ጥርዝዓረፍተነገሮች? ወጣ;
*/*production from Arrays */* arrayDecl : ARRAY Identifier ('=' arrayType arrayInitializer)?; arrayInitializer : '**{**' variableInitializerList'**}**';	ድርድርአፀፃፍ ፡ ‘ድርድር’ መለያ (‘=’ ድርድርዓይነት ድርድርመሰያሚያ)?; ድርድርመሰየሚያ ፡ ‘ {‘ ተለዋዋጭመሰየሚያዝርዝር’} ’;
variableInitializerList : variableInitializer (**','** variableInitializer)*;	ተለዋዋጭመሰየሚያዝርዝር ፡ ተለዋዋጭመሰየሚያ (‘,’ ተለዋዋጭመሰየሚያ)*;

APPENDIX 1 Continued

/ production from blocks and stmts */*

```
block :  INDENT blockStmts? DEDENT;

blockStmts : blockStmt+;

blockStmt : localVariableDeclaration
NEWLINE
        | stmt;

localVariableDeclaration : varType
variableDeclarationList;

stmt : stmtWithoutTrailingSubstmt
    | ifStmt
    | ifElseStmt
    | whileStmt;

stmtFullIf : stmtWithoutTrailingSubstmt
           | ifElseStmtFullIf
           | whileStmtFullIf;
stmtWithoutTrailingSubstmt :
expressionStmt
                    | breakStmt
                    | continueStmt
                    | returnStmt
                    | tryStmt;
expressionStmt : stmtExpr;
stmtExpr : assignment
```

```
የትዕዛዝጥርቅም ፡ ገባ ጥርዝዓረፍተነገሮች? ወጣ;

ጥርዝዓረፍተነገሮች ፡ ጥርዝዓረፍተነገር+;

ጥርዝዓረፍተነገር ፡ ጊዜያዊተለዋዋጭአፀፃፍ
አዲስመስመር | ዓረፍተነገር;

ጊዜያዊተለዋዋጭአፀፃፍ ፡ ተለዋዋጭዓይነት
ተለዋዋጭአፀፃፍዝርዝር;

ዓረፍተነገር ፡ ተከታይየሌለውዓረፍተነገር
          | ከሆነዓረፍተነገር
          | ከሆነካልሆነዓረፍተነገር
          | እስከሆነዓረፍተነገር;

ሙሉከሆነዓረፍተነገር ፡ ተከታይየሌለውዓረፍተነገር
                | ከሆነካልሆነዓረፍተነገርሙሉከሆነ
;
                | እስከሆነዓረፍተነገርሙሉከሆነ;

ተከታይየሌለውዓረፍተነገር ፡ ትእዛዛዊዓረፍተነገር
                    | አቋርጥዓረፍተነገር
                    | ቀጥልአረፍተነገር
                    | መልስዓረፍተነገር
                    | ሞክርዓረፍተነገር
ትእዛዛዊዓረፍተነገር ፡ ትእዛዛዊዓ_ነገር
ትእዛዛዊዓ_ነገር ፡ ሲያሜ
```

APPENDIX 1 Continued

| preIncrExpr	| ቅድመጭመራትዕዛዝ
| preDecrExpr	| ቅድመቅነሳትዕዛዝ
| postIncrExpr	| ድህረጭመራትዕዛዝ
| postDecrExpr	| ድህረቅነሳትዕዛዝ
| methodInvocation	| ስብስብምስረታትዕዛዝ
| classCreationExpr;	| ትዕዛዝምስረታ;

ifStmt : LPAREN expression RPAREN '**if**' stmt '**\n**' END;	ከሆነዓረፍተነገር ፡ ‘(‘ ትዕዛዝ ‘)’ ከሆነ ዓረፍተነገር አዲስመስመር ጨርስ
ifElseStmt : LPAREN expression RPAREN IF stmtFullIf END ELSE stmt END; ifElseStmtFullIf : LPAREN expression RPAREN IF stmtFullIf END ELSE stmtFullIf END;	ከሆነካልሆነአረፍተነገር ‘(‘ ትዕዛዝ ‘)’ ዓ_ነገርሙሉከሆነ ጨርስ ካልሆነ ዓረፍተነገር ጨርስ ከሆነካልሆነዓረፍተነገርሙሉከሆነ ፡ ‘(‘ ትዕዛዝ ‘)’ ከሆነ ዓ_ነገርሙሉከሆነ ጨርስ ካልሆነ ዓ_ነገርሙሉከሆነ ጨርስ
whileStmt : LPAREN expression RPAREN WHILE INDENT stmt DEDENT; whileStmtFullIf : LPAREN expression RPAREN WHILE INDENT stmtFullIf DEDENT;	እስከሆነዓረፍተነገር ፡ ‘(‘ ትዕዛዝ ‘)’ እስከሆነ ገባ ዓረፍተነገር ወጣ; እስከሆነዓረፍተነገርሙሉከሆነ : '('ትዕዛዝ')' እስከሆነ ገባ ዓ_ነገርሙሉከሆነ ወጣ;
breakStmt : BREAK;	አቋርጥዓረፍተነገር ፡ አቋርጥ;
returnStmt : RETURN expression?;	መልስዓረፍተነገር ፡ መልስ ትዕዛዝ?;
continueStmt : CONTINUE;	ቀጥልአረፍተነገር ፡ ቀጥል;
tryStmt : TRY block catches;	ሞክርዓረፍተነገር ፡ ሞክር የትዕዛዝጥርቅም ቁጥጥር;
catches : catchClause catchClause*;	ቁጥጥር ፡ ስህተትቁጥጥር ስህተትቁጥጥር*;

APPENDIX 1 Continued

catchClause : CATCH LPAREN catchFormalParameter RPAREN block;	ስህተትቁጥጥር ፡ ቁጥጥር ‘(‘ የስህተትዓይነት’)’ የትዕዛዝጥርቅም;
catchFormalParameter : catchType variableId;	የስህተትዓይነት ፡ ስህተትዓይነት ስህተትመለያ;
catchType : classType*;	ስህተትዓይነት ፡ ስብስብዓይነት*
/* *Expr* */ primary : literal | **'('** expression **')'**;	/*ትዕዛዝ*/ ግብዓት ፡ ግብዓትጥርቅም | ‘(‘ ትዕዛዝ’)’
classCreationExpr : variableDecl ASSIGN NEW LPAREN Identifier RPAREN;	ስብስብምስረታትዕዛዝ ፡ ተለዋዋጭአፀፃፍ ‘=’ አዲስ’(‘መለያ’)’
methodInvocation : methodName LPAREN argumentList?RPAREN;	ትዕዛዝምስረታ ፡ የትዕዛዝስም ‘(‘ቦታመያዣተለዋዋጭዝርዝር’)’
argumentList : expression (COMMA expression)*;	ቦታመያዣተለዋዋጭዝርዝር ፡ ትዕዛዝ (‘,’ ትዕዛዝ)*
expression : assignmentExpr;	ትዕዛዝ ፡ ስያሜትዕዛዝ
assignmentExpr : conditionalExpr | assignment;	ስያሜትዕዛዝ ፡ ሁኔታዊትዕዛዝ | ስያሜ
assignment : leftHandSide assignmentOperator expression	ስያሜ ፡ የግራክፍል ስያሜምልክት ትዕዛዝ
leftHandSide : expressionName;	የግራክፍል ፡ የትዕዛዝስም
assignmentOperator : assign | mul_assign	ስያሜምልክት ፡ ስያሜ | ብዜት_ስያሜ

APPENDIX 1 Continued

```
                | div_assign
                | mod_assign | add_assign
                | sub_assign;

conditionalExpr : conditionalOrExpr ;

conditionalOrExpr : conditionalAndExpr
                | conditionalOrExpr OR
conditionalAndExpr;

conditionalAndExpr : andExpr
                | conditionalAndExpr AND
andExpr;

andExpr : equalityExpr
      | andExpr BITAND equalityExpr;

equalityExpr : relationExpr
           | equalityExpr EQUAL
relationExpr;

relationExpr  : addExpr
            | relationExpr LT expression
            | relationExpr GT expression
            | relationExpr LE expression
| relationExpr GE expression;
addExpr : multExpr        | addExpr ADD
multExpr
      | addExpr SUB multExpr;
```

```
                | ማካፈል_ስያሜ
                | ቀሪ_ስያሜ   | ድምር_ስያሜ
                | መቀነስ_ስያሜ

ሁኔታዊትዕዛዝ ፡ ሁኔታዊወይትዕዛዝ

ሁኔታዊወይትዕዛዝ ፡ ሁኔታዊእናትዕዛዝ |
ሁኔታዊወይትዕዛዝ ወይ ሁኔታዊእናትዕዛዝ

ሁኔታዊእናትዕዛዝ ፡ እናትዕዛዝ
                | ሁኔታዊእናትዕዛዝ
                እና   እናትዕዛዝ

እናትዕዛዝ ፡ እኩሌታትዕዛዝ
          | እናትዕዛዝ
          ነጥለእና እኩሌታትዕዛዝ

እኩሌታትዕዛዝ ፡ ማወዳደሪያትዕዛዝ
               | እኩሌታትዕዛዝ እኩል
ማወዳደርያትዕዛዝ

ማወዳደርያትዕዛዝ ፡ ድምርትዕዛዝ
          | ማወዳደርያትዕዛዝ ያንሳል ትዕዛዝ
          | ማወዳደርያትዕዛዝ ይበልጣል ትዕዛዝ
          | ማወዳደርያትዕዛዝ ያንሳልእኩል ትዕዛዝ
          | ማወዳደርያትዕዛዝ ይበልጥእኩል ትዕዛዝ
ድምርትዕዛዝ ፡ ብዜትትዕዛዝ
           | ድምርትዕዛዝ ድምር ብዜትትዕዛዝ
           | ድምርትዕዛዝ ቀንስ ብዜትትዕዛዝ
```

APPENDIX 1 Continued

```
addExpr : multExpr
        | addExpr ADD multExpr
        | addExpr SUB multExpr;

multExpr : unaryExpr
         | multExpr MUL unaryExpr
         | multExpr DIV unaryExpr
         | multExpr MOD unaryExpr;

unaryExpr : preIncrExpr
              | preDecrExpr
              | ADD unaryExpr
              | SUB unaryExpr
              | postfixExpr;
preIncrExpr: INC unaryExpr;

preDecrExpr: DEC unaryExpr;
postfixExpr:( primary | expressionName)
    (  INC     |  DEC     )*;
postIncrExpr: postfixExpr INC;

postDecrExpr: postfixExpr DEC;
/* keywords */
Array: 'array'
BOOLEAN: 'boolean'
BREAK: 'break'
CATCH: 'catch'
CLASS: 'class'
```

```
ድምርትዕዛዝ ፡ ብዜትትዕዛዝ
          | ድምርትዕዛዝ ድምር ብዜትትዕዛዝ
          | ድምርትዕዛዝ ቀንስ ብዜትትዕዛዝ

ብዜትትዕዛዝ ፡ ነጠላትዕዛዝ
           | ብዜትትዕዛዝ ብዜት ነጥላትዕዛዝ
           | ብዜትትዕዛዝ ሲካፈል ነጥላትዕዛዝ
           | ብዜትትዕዛዝ ቀሪ ነጥላትዕዛዝ

ነጥላትዕዛዝ ፡ ቅድመጭመራትዕዛዝ
            | ቅድመቅነሳትዕዛዝ
            | ድምር ነጠላትዕዛዝ
            | ቀንስ ነጠላትዕዛዝ
            | ድህረቅፅልትዕዛዝ;
ቅድመጭመራትዕዛዝ ፡ ጭመሪ ነጥላትዕዛዝ;

ቅድመቅነሳትዕዛዝ ፡ ቅናሽ ነጥላትዕዛዝ;
ድህረቅፅልትዕዛዝ ፡ (ግብዓት | የትዕዛዝስም ) (ጭማሪ |
ቅናሽ)*;
ድህረጭመራትዕዛዝ፡ ድህረቅፅልትዕዛዝ ጭማሪ
ድህረቅነሳትዕዛዝ፡ ድህረቅፅልትዕዛዝ ቅናሽ

/* አብይቃላት*/
ድርድር = 'ድርድር'
እዉነታ = 'እወነታ'
አቁርጥ = 'አቋርጥ'
ቁጥጥር = 'ቁጥጥር'
ስብስብ = 'ስብስብ'
አይለወጤ = 'አይለወጤ'
```

APPENDIX 1 Continued

CONST: 'const'

CONTINUE: 'continue'

DOUBLE: 'double'

ELSE: 'else'

END: 'end'

EXTENDS: 'extends'

If: 'if'

IMPORT: 'import'

INT: 'int'

NEW: 'new'

RETURN: 'return'

STATIC: 'static'

TRY: 'try'

VOID: 'void'

TRUE: 'true'

FALSE:’false’

ቀጥል = 'ቀጥል'

ነጥብ = 'ነጥብ'

ካልሆነ = 'ካልሆነ'

ጨርስ = 'ጨርስ'

ዉርስ ='ዉርስ'

ከሆነ = 'ከሆነ'

ተጠቀም = 'ተጠቀም'

ቁጥር = 'ቁጥር'

አዲስ = 'አዲስ'

መልስ = 'መልስ'

ፍፁም = 'ፍፁም'

ሞክር = 'ሞክር'

ባዶ = 'ባዶ'

እወነት = 'እወነት'

ሐሰት = ‘ሐሰት’