Simplifying Scientific Programming with *Streak©*

Francisca O. Oladipo, PhD
Department of Computer Science, Faculty of Science
Federal University Lokoja
Lokoja, Nigeria
Email: francisca.oladipo [AT] fulokoja.edu.ng

Victor U. Anawo
Department of Computer Science, Faculty of Science
Federal University Lokoja
Lokoja, Nigeria

Abstract— **It is still often perceived that programming is only for experts. This in combination with the low expressiveness features of many programming languages discourage a lot more people from going fully into the field of programming. The result of this research is a quasi-programming environment named *Streak*©. *Streak*© parades a command line interface where all programming activities including variable declaration can be carried out. *Streak*©, supports primitive data types, comments using the "#" symbol and an array of reserved words like streak, streakvar, feedin, feedout, end, etc. Being a quasi-environment, *Streak*© does not have its own Graphical User Interface, but the output is displayed on the Command Line Interface. The language building blocks is made up of a translation program, grammatical structures based on the Backus-Naur form and regular expressions. It runs on a command line interface which can be integrated into popular IDE and has its own set of easy to remember and use keywords. *Streak*© provides extensive support for breakpoints which in turn allows the debugger to debug programs faster.**

Keywords- programming language, quasi-environments, CLI, mac/windows interface

I. INTRODUCTION

Languages in general is a means for communication and they are made up of the structure which refers to the syntax and the rules which define the meaning of a language –the semantics. A computer system only understands and communicates in the binary code (0s and 1s), this is known as the machine language. While it is possible to straightforwardly write programs in the machine language, it is tedious and mistake inclined to oversee individual bit and calculate numerical addresses and constants manually. Therefore, machine language is nearly never used to write programs in present day settings; this prompted the introduction of the assembly language [1].

Assembly language was birthed to ease programming; it wiped out a great part of the error prone, and tedious binary code system in programming with the earliest computer; liberating software engineers from tedium, for example, recollecting numeric codes and calculating addresses. An assembly language, frequently represented as "asm", is a low level programming language, in which there is a solid correspondence between the language design and the computer's machine code instructions; every assembly language is particular to a specific computer engineering [2].

Despite the fact that Assembly languages are quicker and superior to the binary coding system, it is as yet ailing in many regards: there are no typical names for memory locations, the codes are difficult to read, the code is still machine dependent, it is difficult to maintain and debug, and the code must be vigorously documented. This prompted the introduction of high-level languages, which encourages programming in a human-like language [3].

According to [4], the term "High-level language" refers to a larger amount of reflection from machine language. High level languages are programming languages that are machine independent, they are nearer to human languages and require to be translated to machine language before the instructions they hold can be executed. Thus, the advancement of programming languages from machine language to high-level language has accomplished its aim, which is to simplify communication between humans and computers. Some programming languages were created to be used in a wide variety of fields while others were created to be used within a certain domain; they are the general purpose programming languages and the domain-specific programming language [5].

A general purpose programming language is mostly intended for writing programs in an assortment of application domains, it has this property because it does not include language constructs intended to be utilized inside a particular application domain. A domain-specific programming language is one intended to be utilized inside a particular application domain [6].

Programming, as a branch of computer science deals with writing sets of instructions to be executed by a computer in a particular programming language. However, programming is usually perceived to be difficult, and as such, novices in the field of computing and students alike often shy away from programming. This can be due to a variety of reasons which may be complex rules governing a programming language, confusing keywords, or difficult process of debugging [7].

According to [8], bugs hinder the learning process of novices in two ways: the students get easily discouraged from following their curriculum material due to the number of bugs they encounter which are unrelated to the concepts being learned; secondly, novices often possess some misconceptions about the syntax and semantics of a programming language, which lead to confusion when their programs behave differently from what they expect.

This research hopes to simplify programming by reducing the amount of time spent on the process of debugging, thus, reducing the stress encountered by newbies in the field of computing. It aims at reducing the complexity of programming experienced by novices in the field by simplifying the process of program writing through the development of *Streak©*, a quasi-programming language with support for breakpoints thereby enabling efficient debugging. *Streak©* will provide support for error handling by suggesting possible solutions to errors, and reduce the length of program codes with the concept of economy of expression.

This section on introduction will be followed by a review of related research, approaches and products. A brief description of the materials and methods deployed towards the construction of *Streak©* will be presented before the design artefacts. After an extensive description of *Streak©*'s design results, the paper shall present the implementation before the concluding remarks and recommendations.

II. Review of Related Research

A. Evolution Of Programming Languages

Programming languages are languages that determine an arrangement of guidelines which may be used to yield different kinds of outputs. Programming languages for the most part comprises of instruction sets for a computer. Programming languages may be used for writing programs which implement particular algorithms. Programming languages generally describe computation on a few, perhaps abstract machines. Programming languages vary natural languages and that regular languages are mainly used for communication between individuals. Programming languages enables people to communicate instructions to computer machines [9].

Amid a nine-month time frame in 1842 and 1843, Ada Lovelace deciphered the diary of Italian statistician Luigi Menabrea on Charles Babbage's most current planned engine, the analytical engine. With the editorial, she attached an arrangement of records which indicated in total detail a strategy for computing Bernoulli numbers through the engine, perceived by a few history specialists as per the world's original computer program [10].

Konrad Zuse created **Plankalkul,** a high-level programming language in 1945, in which the main constructs are: variable assignments, guarded commands and while loops, arithemetical and logical operations. Plankalkul was created for Z1, a programmable computer built by Zuse in-between 1936 to 1945. However, the language, Plankalkul, remained not effected up until 1998 and 2000 [11].

Some years later, John Mauchly's **Small Code** which was planned in 1949, was among the first high-level languages at any point created to be used on electronic computers. Notwithstanding, the program must be converted to machine code each time it was run, making the procedure much slower than executing the equal machine code [12].

In the mid-1950s, Alick Glennie created **Autocode** at University of Manchester. A programming language that used a compiler for automatic conversion of the language to machine code. The primary compiler and code was created in 1952 for the Mark 1 PC at University of Manchester, and is thought to be the earliest compiled high-level programming language, it was also faster than John Mauchly's Small code [13].

Around that period, John Backus invented the **FORTRAN** language in the year 1954 at IBM. It was the initial high level and also general-purpose programming language to be widely used. It has a functional implementation. Though the language is quickly going into extinction, it is still a widely held language for high-performance computing [14]. The name FORTRAN is short for FORmula TRANslating. Today, the FORTRAN language is considered restraining as it just includes the DO, IF and GOTO statements, yet at that time, these statements were a major stride forward in the field of computing. The basic data types being used today was started in FORTRAN; they include logical variables which are TRUE or FALSE statements, real, integer, and double [15].

Grace Hopper discovered that data processing business customers were not comfortable using mathematical notation (as was the case with FORTRAN). Therefore, she conceived the idea that led to the production of another programming language, **FLOW-MATIC.** It was produced to be used on the UNIVAC I computer within the period of 1955 and 1959. Hopper and her group designed a description for the programming language which is in English and executed a prototype in 1955 [16]. FLOW-MATIC compiler turned out to be openly accessible in 1958 and was whole in 1959. FLOW-MATIC was the main factor which led to the development of COBOL [17].

Business computing took off in 1959, **COBOL** programming language was developed because of business computing. COBOL was developed by CODASYL in 1959 and was partially based on FLOW-MATIC. COBOL is an acronym for Common Business-Oriented Language. It was mainly designed for use in business. It is procedural, imperative and object-oriented [18]. Its data types are strings and numbers. It allows for the grouping of data types into records and arrays so that data can be managed properly. The statements in COBOL are very much like English grammar, which makes it easy to learn. All these features were implemented in order for it to be easy to learn and adopt it by the average businessman [19].

A new type of research began during mid-1950s; it is known as the Artificial Intelligence (AI). John McCarthy of MIT in 1958 designed the **LISP** programming language. LISP stands for LISt Processing language. It was developed for Artificial Intelligence (AI) research. Since it was intended for a particular field of study, the first release of LISP had unique language structure. A conspicuous contrast with this language (in its original form) from other languages is: the essential and only datatype is list; in 1960's, LISP procured other data types. LISP list can be denoted by series of items encircled by parenthesis. LISP programs are written as an arrangement of lists, with the goal that LISP will be capable of altering itself, and consequently develop on its own. LISP is still in use today because it has an abstract nature and it is highly specialized.

The first complete compiler which was written in LISP was implemented in 1962 by Mike Levin and Tim Hart [20].

Still in the year 1958, the **ALGOL** programming language was made by a group for scientific use. ALGOL is short for Algorithmic Language. ALGOL was the first language to implement nested function definition with lexical scope. It also introduced block of codes and the "begin" and "end" pairs for delimiting them. It was additionally the language which introduced a formal grammar, which was branded as Backus-Naar Form (BNF). Its main impact is being the foundation of the hierarchy that has prompted languages such as Java, Pascal, C++ and C [21].

Using the development of ALGOL programming language, Niklaus Wirth designed the **Pascal language** in 1968 to 1969 and was published in 1970. It was written out of a need for a decent teaching tool. Pascal was planned in a systematic approach, it consolidated a hefty portion of the finest elements of the languages being used at that time; FORTRAN, COBOL and ALGOL. At that parallel time, a considerable lot of the abnormalities and odd statements of the languages in existence then, were tidied up, which enabled it pick up users [22]. It is mainly intended to encourage programming styles using structured programming and data structuring Pascal additionally helped the growth of dynamic variables; the variables could be made with the NEW and DISPOSE orders while a program ran. Notwithstanding, Pascal did not execute dynamic arrays, or group of variables, which turned out to be required and prompted its fall. Wirth later made a successor to Pascal, Modula-2, yet when it showed up, C was picking up prominence and users at a fast pace [23].

C language, also developed between 1969 and 1973 during the same time as Pascal programming language by Dennis Ritchie while working at Bells Labs was used to re-write the Unix OS. The evolution in use from the first main programming languages to the main programming languages in our present day happened with the move from Pascal to C. Its immediate predecessors are BCPL and B, however its likenesses with Pascal are very self-evident. The greater part of the components of Pascal, together with the fresh ones, for example, the CASE construct are obtainable in C. C utilizes pointers broadly and was made to be quick and effective to the detriment of being hard to use. But since it settled the greater part of the errors Pascal had, it prevailed upon previous Pascal users quickly. C is a general-purpose and imperative programming language; it supports structured programming, recursion, and lexical variable scope. C makes available constructs that map efficiently to typical machine instructions, and in this way, it has found enduring use in applications that had once been coded in assembly language, including operating systems, and in addition, different application software for computers extending from supercomputers to embedded systems [24] [25].

Within 1970's and 1980's, another programming technique was created. It was known as Object Oriented Programming (OOP). Objects are bits of data which can be bundled and controlled by the developer. Bjarne Stroustroup enjoyed this strategy and created expansions to C known as "**C with classes**". This arrangement of augmentations formed into the fully developed language known as **C++,** and was published in 1983. C++ was intended to organize the power of C utilizing OOP, yet keep up the speediness of C and have the capacity to keep running on a wide range of sorts of computers. C++ is regularly used in simulations, for example, games. It was developed with an inclination concerning system programming and resource-constrained, embedded and big systems, with flexibility, efficiency and performance as its design highpoints [26]. The International Organization for Standardization (ISO) is responsible for the standardization of C++, it was firstly standardized in 1998 and was adjusted by the C++03 standard in 2003. The current standard is the C++14 standard which supersedes the previous C++11 standard with latest features and a bigger standard library. In July 2017, the C++17 standard will be due and the next scheduled standard is the C++20 standard [27].

In 1991, the idea of **Java programming language** was initiated by James Gosling, Patrick Naughton and Mike Sheridan. It was originally intended for interactive television; it later became too advanced for the digital television industry at that time. The Java language was firstly called Oak which was named after an oak tree standing outside Gosling's office at the time. The name of the project was later changed to Green, then was finally named Java, after Java coffee [28]. Java programming language was intended to allow software developers to "compose once, run anywhere" which means that any compiled Java code will be able to run on platforms which support Java without any need to recompile. Java codes are compiled to bytecode which can run on all Java virtual machine (JVM) while not regarding the architecture of the computer. Java was developed at Sun Microsystems. Oracle Corporation later acquired Sun Microsystems. Java was released in 1995 [29].

The programming language **Visual Basic** was developed based on the language Basic which was created by Thomas Kurtz and John Kemeny in 1964. The BASIC language is very limited and was mainly designed for people of other fields of study which is not computer science. Statements can be run in sequence but program modification can be made through the GOSUB and IF…THEN statements which runs a block of code, then returns back to its initial point in the codes. BASIC is short for Beginner's All-purpose Symbolic Instruction Code [30].

In 1991, Microsoft expanded BASIC in its new product, Visual Basic. In 1998, Visual Basic 6.0 was released. Microsoft stopped supports for the Visual Basic 6.0 IDE [31]. Visual Basic .NET was launched in 2002 by Microsoft. It relied on .NET framework 1.0 [32].

In 1999, Andrew Hejlsberg of Microsoft gathered together a team to develop a new language which would be called "cool". It was later renamed to C#. C# encompasses imperative, functional, declarative, strong typing generic, component-oriented and object-oriented programming disciplines [33].

Programming Languages have and will continue to evolve as new ideas are implemented. All programming languages in existence possess primitive building blocks for describing data [34].

B. Paradigm Shifts in Programming Languages

In the 1970s, the stepwise refinement and top-down methodologies were developed. Prior to the time, deficiencies discovered in primary programming languages were inadequacy of control statements which required the widespread use of GOTOs and incompleteness of type checking.

A shift began to occur in the late 1970s from procedure-oriented design methodologies to data-oriented design methodologies. The latter design methodology focused more in the use of abstract data types to find solutions to problems. In the evolution of data-oriented development of software, the latest step which began in 1980 is Object-oriented design. Object-oriented design methodology starts with data abstraction. This captures data processing objects and controls access to data. It also adds dynamic method binding and inheritance. Inheritance in OOP empowers new objects to adopt the properties of objects which are already in existence. Procedure-oriented programming methodology on the other hand is the opposite of data-oriented programming methodology. Despite the fact that data-oriented methodology is mainly adopted in most programming languages, procedure-oriented programming has not been abandoned as there have recently been an increased research on it especially in the field of concurrency [35].

C. Programming languages Implementation Principles

There are generally two methods of implementing a programming language. They are the interpretation and compilation. Interpreters directly executes programs written in high-level programming language without compiling them into machine language. An interpreter interprets the source code line in sequence. Interpreters are generally flexible. Interpretation are the step of conversion, transmission and implementation. The various characteristics of an interpreter are:

i. A short amount of time is spent analyzing and running the program.
ii. The subsequent code is some kind of intermediate code
iii. The resultant code is interpreted by a different program.
iv. Execution of programs is relatively slow [36].

Compilers are programs that converts source codes in a programming language into the target language. It translates programs written in a high-level programming language to that of a low-level machine language. Compilers were written first before an interpreter.

The following are characteristics of a compiler:

i. Spends a considerable amount of time analyzing and running the program.
ii. The resultant executable code is in a way machine-specific binary code.
iii. The hardware of the computer executes the resultant code.
iv. Execution of programs is fast [37].

Both an Interpreter and a compiler translates codes from high-level language to low-level machine language. A compiler translates codes all at once while an interpreter translates codes line by line.

D. Creating a Programming Language

Creating a programming language can be structured in stages. Each stage possess data which have been formatted in a certain way and also has functions which converts data from one stage to the next. The following are two of the stages; Lexical analysis and Syntactic analysis (parsing).

Lexical analysis is the conversion of sequences of characters into a sequence of tokens. These characters may either be a computer program, an expression or a web page. Lexemes or tokens are strings of characters that form a syntactic unit. Tokenization is the procedure of demarcation and classification of sections of strings of input characters [35].

Syntactic analysis, also known as parsing is the procedure of analyzing a set of symbols and checking if they conform to the rules of a formal grammar. Parsing can be done in two ways:

1. Top-down Parsing: Top-down parsing constructs parse trees from the start symbol (top) to the through, to the down.
2. Bottom-up Parsing: Bottom-up parsing constructs parse trees from the terminal symbols (leaves) to the up [38].

E. Relevant Approaches to Creating a Programming Language

Computer programs are written in a programming language; mostly in high-level languages. The computer, however, cannot understand these programs in high-level language as it only understands the machine language. Hence, it needs a translation program to translate the program (source codes) into machine language. The translation program is known as a compiler.

The process of translation is steered by a structure of the examined code. The process of translation is structured into parts: these parts also are the steps followed when creating a translation program (compiler). They are:

1. The arrangement of characters of the source code is converted into a relating succession of symbols of the terminology of the language. This is also known as Lexical Analysis.
2. The arrangement of symbols is translated into a depiction which reflects the structure of the syntax of the source code and allows the structure to be easily recognized. This stage is known as the Syntax Analysis.
3. In high-level languages, objects such as functions and variables are categorized accordingly to their type.

There are rules which exist among the types of operators and operands. These rules define the language. Therefore, confirming if these rules are properly followed by a program is an added feature of a compiler. This verification process is known as type checking.

4. Based on the results from the syntactic analysis phase of the translation program, a set of instructions is generated from the instruction set of the target machine. This stage is known as the Code Generation phase. This part of translation program is often the most complex part because the instruction sets of a lot of computers lack regularity. Hence, in most cases, the code generation part is further divided into subparts.

When building a high-level programming language, these steps are generally followed [39].

F. Review of Existing Relevant Literature

Computers play a significant role in today's world as programs are constantly being developed to perform many functions in different aspects of life. These programs are written in different programming languages. Different kinds of programming language for different applications has been developed in a large amount and they are broadly categorized into imperative, logical, markup, object-oriented and functional languages. The languages are structured differently from each other and also in how they operate. A few of the existing programming languages are reviewed below:

Ada is a multi-paradigm, object-oriented high level language obtained from extending the structure and semantics of Pascal. It is a strongly typed, static, safe and nominative language with its file extension as *.adb* and *.ads*. designed with mainly development of large systems in mind [40]. Ada eases debugging easier by allowing the programmer to specify the conditions attached to codes, or the restrictions on a code. Its major disadvantage is the relatively obscure nature, making it difficult for system developers developing in other programming language to be able to offer assistance if needed [41].

Algol, short form for Algorithmic Language, appeared first in 1958. It was a great influence to many of the programming languages currently in existence now such as Simula, B, BCPL, C, Pascal [42]. Algol pioneered the code blocks with the "begin" and "end" pairs which were used for delimiting them, nested functions and the Backus-Naur form which is a formal grammar for language design were firstly implemented in Algol 60 [21].

BC is short for Basic or Bench Calculator. It is a calculator with arbitrary precision whose syntax is closely related to C programming language. BC is mainly used as a scripting language or interactive shell for mathematics; with support for single-lettered arrays, functions and variables. It appeared firstly in 1975 and was invented by Lorinda Cherry and Robert Morris of Bell Labs [43].

BCPL is short for Basic Combined Programming Language. It is a structured, procedural and imperative language designed by Martin Richards in 1966. BCPL was influenced by CPL and it in turn influenced B, C and Go programming languages. The language was intended to be a "compiler's compiler" i.e. for creating compilers to use in other programming languages. Though no longer in frequent use again, its existence is still acknowledged because of the language B: which is almost like a newer and cleaner version of BCPL. C programming language was based on B. BCPL actually was the first to use brace while programming, and these braces survived the test of time and various changes in syntaxes [44].

The **C** programming language is an imperative structured language. It appeared firstly in 1972 and was designed by Dennis Ritchie. It was influenced Algol 68, Fortran, B, PL/l and it in turn influenced numerous languages such as AWK, AMPL, C--, C++, C#, csh, Objective-C, BitC, Go, Java, D, LPC, Perl, Limbo, Julia, PHP, Pike, Rust, Processing, Seed7, Python, Verilog, Vala [45]. C provides support for constructs which maps effectively to machine instructions. C was developed for compilation process to be done in a straight fashion to make provision for low-level access to memory. The C language encourages cross-platform programming. as a program written in C can be run in a variety of system platforms from simple micro-controllers to super-computers. C is weakly typed but static, is very powerful; has a large traditional base which makes it easy to find libraries; the codes are compiled and are stand-alone (which means it has no need for interpreters), it is one of the fastest languages running. It has a few disadvantages which are: platform dependency (programs must be compiled in each platforms), codes can easily get chaotic, it is hard to port non-trivial programs. Nevertheless, the C language remains the father of many languages [3] [46].

C# is a static, safe, dynamic, strong, nominative, and partially inferred language developed by Microsoft in the year 2000. Its extension is .cs. The C# language was influenced by Eiffel, Modula-3, Pascal, C++, Java, Haskell, F#, J#, Rust, ML [47] [48]. The "C sharp" name was inspired by music notes in which a "sharp" shows that note should be higher by a semitone in pitch. The language was built with the intention of making it simple, a general-purpose, modern and object-oriented programming language [34].

C++ is a multi-paradigm programming language (i.e. procedural, object-oriented, functional and generic) and its typing discipline is partially inferred, nominative and static. The file extensions are .cc, .cxx, .cpp, .C, .hh, .hxx, .hpp, .h++. C++ was developed with efficiency, performance and flexibility as the highlights of its design. C++ has influenced the development of many programming languages such as Ada 95, C99, D, Chapel, C#, Java (later versions), Lua, Python, PHP, Rust, Perl [49] [26].

COBOL is short for Common Business-Oriented Language developed in 1959 for use primarily in the field of business, and administrative systems for various companies.

COBOL is still in use today but is being quickly replaced by other programming languages as many programs written originally in COBOL are being ported to other modern programming languages. The COBOL language was based on a previous language designed by Grace Hopper, hence, she was known as the grandmother of COBOL [18] [50].

D (extension '.d') is a strongly typed, compiled and multi-paradigm language designed by Walter Bright in 2001. It was influenced by C, C#, C++, Java, Python, Eiffel and in return influenced a few programming languages such as DScript, Genie, MiniD, Qore, Swift, Vala. D's main design aim was to join both the safety and performance of languages which are compiled with the express power of dynamic languages [51] [52].

Euphoria is a high-level general-purpose imperative-procedural interpreted language created by Robert Craig in Toronto, Ontario, Canada. The first marketable release in 1993 was intended for the 16-bit DOS platform. Programs written in Euphoria programming language can be bound with the Euphoria interpreter to create executable files which are stand alone as it provides supports for a number of GUI libraries which includes Win32lib and wrappers for GTK+, wxWidgets and IUP. Euphoria became an open-source program in the year 2006 with the release of its version 3. The current version is the version 4 release [53].

F# is a multi-paradigm and a strongly typed language that can be used to generate JavaScript and also graphics processing unit. It is mainly used as a cross-platform Common Language Infrastructure (CLI) and it found its root from Microsoft Research, Cambridge [54] [55]. One major feature of F# is; there is no need for type declaration as the compiler deduces the data types during compilation [56]. New values in F# are defined through the use of the "type" keyword. F# makes provision for record, list, tuple, option, discriminated union and result types [57].

Fortran, is an acronym for Fornula Translation [58]. Fortran is a general-purpose programming language mainly designed for computing numbers and also for scientific computation. Fortran has developed through the years with the latest version being Fortran 2008, which supports concurrent programming [59]. Fortran influenced many other programming languages such as ALGOL 58, C, Julia, PACT I, Ratfor, BASIC. Fortran 2008 contains block constructs (which holds the declaration of objects), Submodules (more improved structuring methods for modules), Contiguous attribute (which allows for specification of storage for layout restrictions) [60][61][62].

Go! programming language is developed Google Inc and designed by Robert Griesemer, Ken Thompson and Rob Pike). It is a compiled, imperative, concurrent and structured language originally released in 2009 [63] [64]. It was a project undertaken by Google engineers as they desired to write a new programming language which would maintain most of the positive aspects of already existing popular languages and still resolve the negative aspects of these existing popular programming languages. Go achieved most of its objectives and was a successful project [65][66].

Google Apps Script is a scripting programming language which is used for development of applications which are light-weighted in Google Apps platform [67]. It is adequately easy to learn because it is based on JavaScript, has a debugger which is cloud based and is used in web browsers for debugging Apps Scripts. The language is useful in the creation of tools which are relatively simple for the internal consumption of an organization but has some limitations which include producing results which may be incorrect when dealing with related functions to date and time because of data crossing time zones [68].

Groovy is another object oriented programming language for Java platform released in January 2007 and upgraded in July, 2012 [69]. It is dynamic in nature and has features which are similar to Ruby, Smalltalk, Perl and Python [70]. The extension is '.groovy'. Groovy is compiled dynamically to Java virtual machine bytecode and also interoperates with extra Java libraries and code.

Haskell is a functional high-level language which gives an interesting view on numerous programming issues. Like other functional languages of the present day, higher-order functions give Haskell its energy [71]. The security offered by Haskell is of solid, adaptability of polymorphism and static writing; a blend which prevents programming blunders without an overwhelming syntactic overhead [72][73].

Java programming language is similar to C++, however streamlined to dispose of language characteristics that cause basic programming blunders [30]. Java source code records are ordered into a configuration called bytecode (documents with a .class augmentation), which would then be able to be executed by a Java mediator. Java codes which have been compiled can keep running on most computer platforms since Java translators and runtime conditions, known as Java Virtual Machines (JVMs), exist for most working frameworks, which includes Windows, the Macintosh OS, and UNIX OS [74].

Julia, a high-performance dynamic language meant mainly for numerical computing first appeared in the year 2012. It was created mainly by Jeff Bezanson, Stefan Karpinski, Viral Shas [75]. Julia makes provision for a compiler, numerical accuracy, distributed parallel execution and a broad mathematical function library [76]. It is also very much suited for general purpose programming.

Kotlin is a statically-typed programming language that is executed on the Java virtual machine. It can be compiled also to the source code of JavaScript. It was developed by a team of "JetBrains" programmers which are based in St. Petersburg in Russia. Though its syntax is not harmonious with Java, it is however designed to be interoperable with Java codes [77]. As of 2017, Kotlin programming language was declared officially to be one of the programming languages used for developing Android applications alongside with C++ and Java [78]. Declaring variables and parameter list in Kotlin is similar to that of Pascal in which the data type comes after the variable name has been written. There is no need for a statement

terminator in Kotlin, but if needed, semicolons can act as the statement terminator [79].

Perl is a High-level, and a general purpose, dynamic programming language. It was intended to be simple for people, as opposed to, simple for computers to get it. The sentence structure of the language is part more like human language than strict structures. It is easy to port from one platform to the other as Perl is available for many popular platforms [80][81]. Perl ended up plainly prevalent for two noteworthy reasons: First, the vast majority of what is being done on the Web occurs with text, and is best finished with a language that is intended for processing texts. All the more critically, Perl was considerably superior to whatever choice was there at that time [82].

Ruby is an intelligent, dynamic, general purpose and object-oriented programming language that joins language structure motivated by Perl`s realism with Smalltalk`s calculated style, Python's simplicity of learning like elements. Ruby holds support for multiple programming paradigms which are object-oriented, reflective, functional and imperative. It has a type system which is dynamic and an automatic memory management [83][84].

Visual basic.NET is a general purpose object-oriented programming language. Any software engineer can create applications rapidly with Visual Basic. It is an exceptionally easy to use language [85]. One only needs to simply organize components by utilizing visual tools and then coding the arranged components. Pushing ahead, Microsoft's .NET framework is made out of prearranged code that users can get to at any time. This prearranged code is known as the class library. The programs existing in the class library can be joined or adjusted so as to suit the necessities of developers [33].

III. Materials And Methods

Because *Streak©* was expected to be developed quickly, the Agile SDLC methodology (Figure 1) was adopted in its development. This model combines both the incremental and iterative processes as it focuses more on user satisfaction and adaptability of processes by rapid delivery of software products which are working. Through the Agile method, the product is broken into small incremental builds, and the builds are iterated as each iteration averagely lasts from one to three weeks.

The following development tools were deployed in the construction of *Streak©*

- Cygwin version 2.881 (64bit) as Linux/Window integrating tool
- Bison (GNU Bison) 3.0.4
- Flex 2.6.4
- UML diagrams for the logical descriptions of *Streak©*

From the project plan (Figure 2), *Streak©'s* development spanned a period of ten (10) weeks including planning and integration testing. Other pre-design activities conducted during this research are:

1. *Studies to determine the technical, economic and environmental feasibilities of developing Streak©*
2. *Requirement elicitation collection through oral interviews of selected novice programmers and brainstorming with experts and other researchers.*
3. Specification of the functional and non-requirements for *Streak©*

After the pre-design phase, the specification of the design artefacts for *Streak©* was carried out and the design documents generated. Further post-design activities carried out in the course of developing *Streak©* are:

1. Specification of the modulus, component areas and algorithms necessary for the implementation of the Streak programming language.
2. Application of the programming language design theories and specification earlier identified into the implementation of the programming environment.
3. Transformation of the defined architecture and artefacts into the syntax of Java programming language.
4. Release of the first version to users.

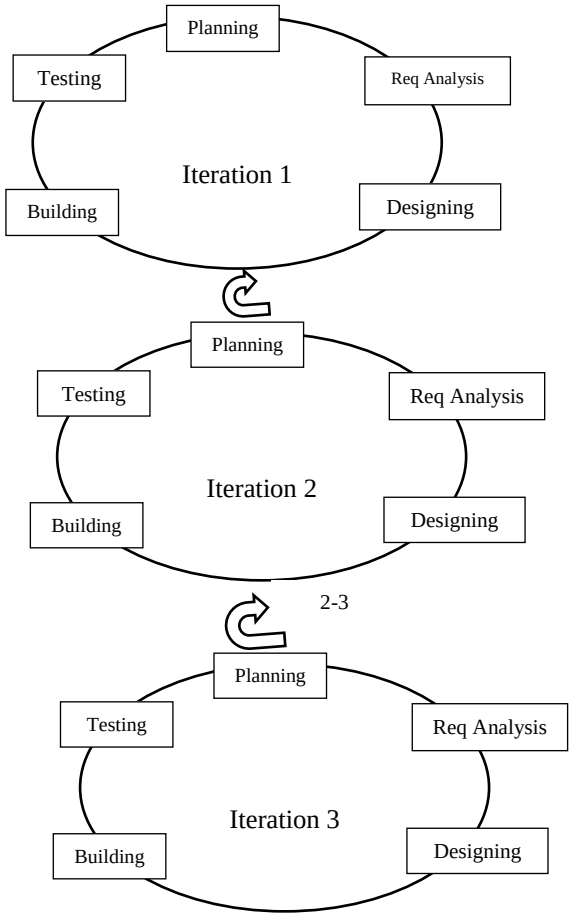

Figure 1. Agile Methodology for *Streak©*

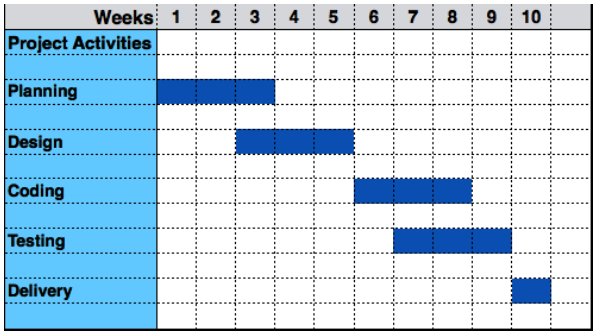

Figure 2: Project Plan for the Development of *Streak©*

IV. Evolution of Streak©

A. Functional Requirements for Streak©

1. The Streak programming language must have the Syntax which are the grammar rules that defines legal statements.
2. The programming language must have the semantics which defines the meaning of things.
3. The programming language must allow statements which are instructions that directs on what to do.
4. The programming language must have variables. Variables are places which holds data in memory during program execution.
5. The programming language must possess primitive data types and procedures.
6. The programming language must provide a means of combining expressions (values), composition (objects or procedures) and containment (placing objects in other objects).

B. Non-Functional Requirements for Streak©

1. The programming language will allow support for breakpoints which will enable efficient debugging.
2. The programming language will provide support for error handling by suggesting possible solutions to errors.

C. Design artefacts for Streak©

In the ourse of medeling *Streak©*, a number of design artefats were developed. This section present the high-level specifications for the *Streak©* environment.

The High-level Model of *Streak©* Programming Compiler (Figure 3) showed the process of compilation of *Streak©* codes in phases with each of the phases accepting inputs from the previous phase. In the first phase which is the lexical analysis, the source code is collected from the source program and converted into lexeme (tokens). The next phase which is the Syntax analysis stage, the tokens from the lexical analysis phase is used as input and a parse/syntax tree is generated: the check for syntactically correct expression which is made by the lexemes is done here. The parse tree generated is now checked to see if it follows the rules of the language. This is the semantic analysis phase. An annotated syntax tree is produced as an output by the semantic analyzer. In the next phase, code generation, an intermediate code is generated by the compiler for the target machine. Code optimization is done in the next phase. The codes are well arranged in a way that it can be executed without resource wastage. The optimized code as mapped into the machine language. The Symbol table is a data structure which is maintained all through the stages of the compiler. The symbol table is used for management of scope. The identifier record can be easily searched quickly by the compiler through the use of the symbol table.

The data and control flow specification for *Streak©* (Figure 4) shows the text editor as the input on which the source code is written. The source code after it is written can still be edited before it is passed on to the Compiler/Interpreter to be translated to object code which the machine understands and hence, the machine can now execute the instructions contained in the source code.

In the System architecture (Figure 5), the source code passes through the preprocessor and the output (preprocessed code) now moves into the compiler. After the stages of compilation has fully taken place and the machine code (object code) is generated, the object code will fetch whatever it needs from the library and the linker takes the object code along with the imports from the library and create an executable file

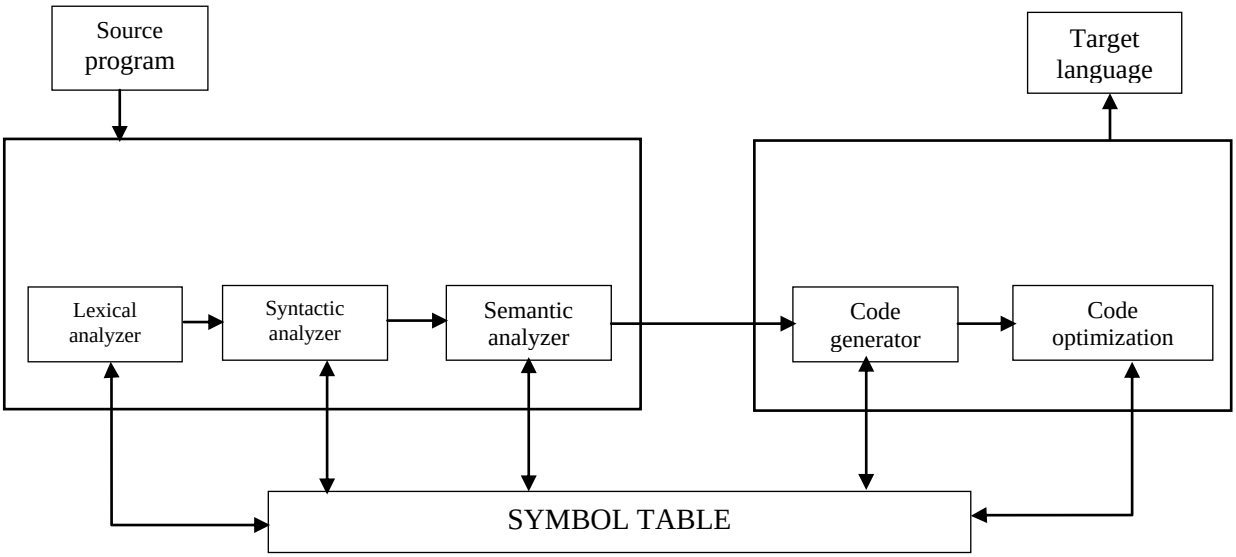

Figure 3. High-level Model of the *Streak*© Programming Language

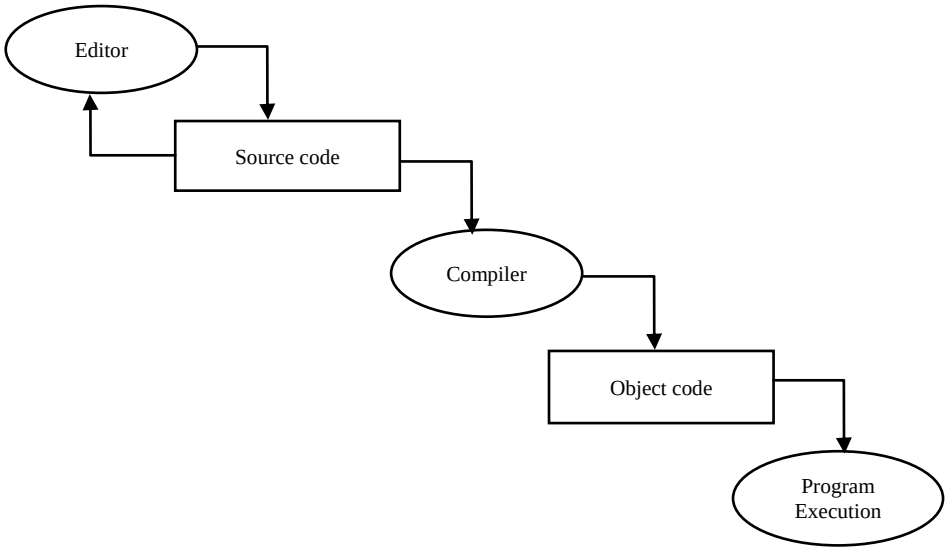

Figure 4. High level View of Data and Control Flow for *Streak*©

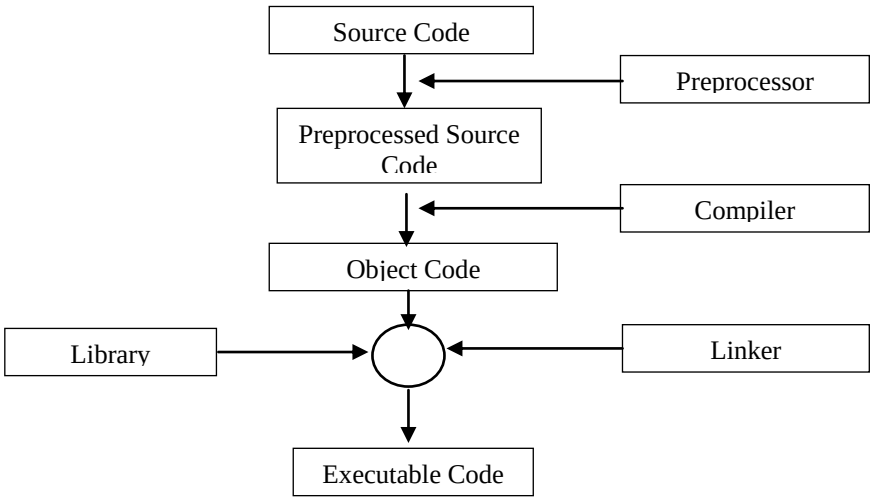

Figure 5. System Architecture of *Streak*© Programming Language

Streak© does not use a GUI, rather the codes are written directly in the command line interface (CLI) and the ouput will also be displayed on the CLI following a set of compilation commands. The Streak programming language can also be integrated into the Eclipse IDE. The basic language elements for input are data types, variables, constants, operators and delimiters (Figure 6).

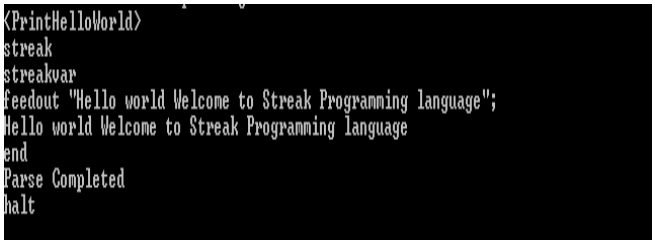

Figure 6: *Streak*© Command Line Interface

One major design artefat for *Streak*© is a use case description that identified and partitioned the system to actors and use cases depicting the roles that could be played by the actors. The use-case diagram below shows the actor (Developer) with roles of managing the writing of new programs, debugging an already written program and adding new features to a program (Figure 7).

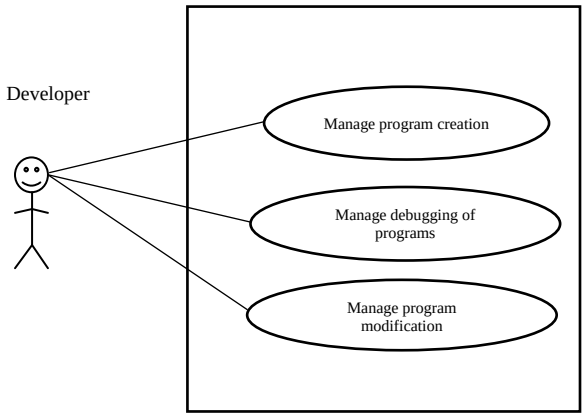

Figure 7 *Streak*© Use-case Description

The workflows for the use cases of *Streak*© were modeled in *an* activity diagram (Figure 8) The source code moves through the parsing phase and the abstract syntax tree is generated. The rough machine code is created and is optimized and the executable code understandable by the machine is generated. The written instructions in the source code is then carried out by the computer.

The sequence diagram (Figure 9) is another design artefact which shows how the user interacts with the *Streak*© compiler and how the executable code is created. The user writes a program and compiles it, the compiler can either load the compiled files into the file system directly or save the generated object files into the file system. The compiler however, passes

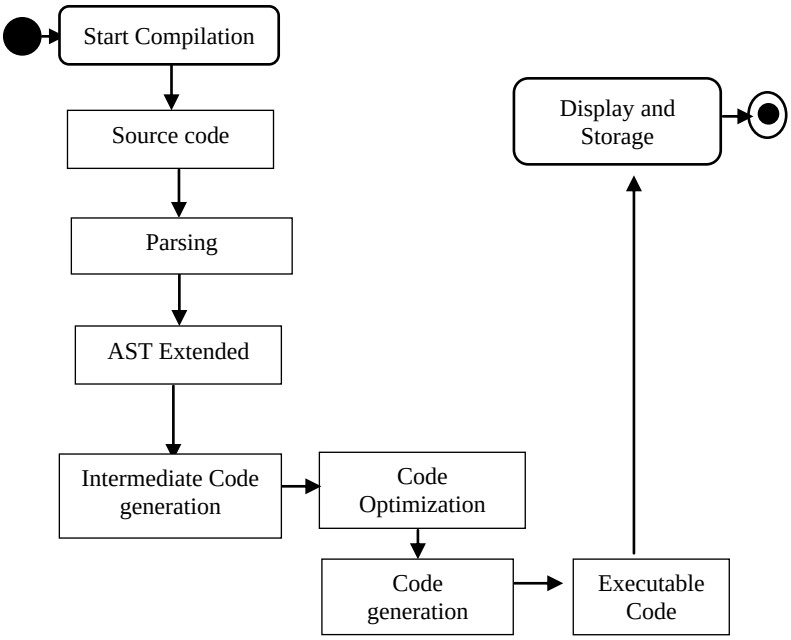

Figure 8. Activity diagram

the object files to the linker and the linker loads them and creates an executable file for the system.

Figure 10 shows the class hierarchy of the *Streak*© programming environment. The Command line interface (CLI) is a superclass, the `Lexer` a subclass of the CLI class and inherits the attributes of the class. While the parser is a subclass of the `Lexer`, its class has its own subclasses which are the `Classparser`, `Import parser` and the `Interface parser`. These parsers perform different functions when called by the *Streak*© compiler.

D. Streak© Application Manual

Streak© is originally developed for Linux OS environment but Cygwin 1.7 tool was deployed for Linux/Windows integration by installing the Command Line Interface (CLI) on Windows OS based systems in order to allow them run Streak©.

To use the Streak© programming language, users are required to follow the following steps:

1. Place the Streak© files in a named folder
2. Open the CLI
3. In the CLI, change the default directory to the folder which houses the Streak© files.
4. Ensure that all the necessary environment files and variables such as the lex and parser files as well as the executable file of the Streak© programming language is complete in the Streak© folder. The

command "ls" may be used to list out the names of the files present in the folder. If the executable file with the filename and extension "streak.exe" exists, the user can then proceed to the next step.

5. Type command "./Streak" in the CLI and the Streak© programming environment will start to run.

All programs written in Streak must begin with the keyword "Streak". The streak programming language has statement terminator which is ";".

The `End` keyword must be used to end a program in *Streak*©.

The comments in *Streak*© should be enclosed in <comment> without any space.

V. Conclusion

Programming deals with writing sets of instructions to be executed by a computer in a particular programming language. However, students and novices in computer science often perceive programming to be a task meant for only experts in the field of computing and as such, are discouraged from exploring the potentials of programming. This can be due to a variety of reasons which may include the complex rules governing a programming language; confusing keywords, or difficult process of debugging. This research had simplified programming by developing Streak©, a quasi-programming environment that accepts simple text-based codes without the cumbersome IDEs thereby reducing the amount of time spent on the process of debugging and reducing the stress encountered by newbies in the field of computing.

Streak© was constructed to use breakpoints in order to speeds up the debugging process making it easy to handle large-component codes. With debugging made easy in a programming language, much less time and effort would be spent in building and running programs therefore enabling the programmer to record much success, hence, encouraging more people to delve into programming.

Streak was an effective tool for teaching and learning programming. The easy-to-use debugging system was of great importance to novice programmers as it enabled them to achieve success with less effort, the Streak environment greatly ignited their passion by enticing them to go further in software development and learn other major languages such as Java and C++.

References

[1] Hennessy, J. L., & Patterson, D. A. (2014). Computer Orgarnization and Design: The Hardware/Software Interface (5th ed.). Waltham, USA: Elsevier.

[2] Hyde, R. (2010). The Art of Assembly Language (2nd ed.). Canada: William Pollock.

[3] Aaby, A. (2004). Introduction to Programming Languages.

[4] HThreads. (2007, August 26). RD - Glossary. Retrieved from HThreads: http://www.ittc.ku.edu/hybridthreads/glossary/index.php

[5] Watt, D. A. (2004). Progrmming Language Design Concept. England: John Wiley & Sons.

[6] Kosar, T., Oliveira, N., Mernik, M., Pereira, M. J., Crepinsek, M., Cruz, D. d., & Henriques, P. R. (2000). Comparing General-Purpose and Domain-Specific Langauges: An Empirical Study. Portugal. doi:10.2298/CSIS1002247K

[7] Casperson, M., Utting, I., Tew, A. E., McCraken, M., Thomas, L., Bouvier, D., . . . Wilusz, T. (2001). A Fresh Look at Novice Programmers' Performance and their Teachers' Expectations. doi:10.1145/2543882.2543884

[8] Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137-172.

[9] Mostrom, J. E. (2011). A Study of Student Problems in Learning to Program. Sweden

[10] Fuegi, J., & Francis, J. (2003). Lovelace & Babbage and the creation of the 1843 'notes'. Annals of the History of Computing, 25(4), 16, 19, 25. doi:10.1109/MAHC .2003.1253887

[11] Rojas, R., Goktekin, C., Friedland, G., & Kruger, M. (2000). Plankalkul: The First High-Level Programming Language and its Implementation. Berlin: Feinarbeit.

[12] Sebesta, R. W. (2006). Concepts of Programming Languages (Seventh Edition). Boston: Pearson/Addison-Wesley.

[13] Bentley, P. J. (2012). The Science of Computers and how it Shapes Our World. London: Oxford University Press.

[14] Loh, E. (2010, June 18). The Ideal HPC Programming Language. Association of Computing Machines.

[15] Chivers, I. D., & Sleightholme, J. (2013). Compiler support for the Fortran 2003 & 2008 standards. ACM SIGPLAN, pp. 26-28. doi:10.1145/152.752.1520755

[16] Bergin, T. J. (2007, May). A History of the History of Programming Languages. Communications of the ACM.

[17] O'Regan, G. (2015). Pillars of Computing: A Compendium of Select, Pivotal Technology Firms. Springer.

[18] Mitchell, R. (2012, March 14). Brain drain: Where Cobol systems go from here. Retrieved from Computerworld: https://www.computerworld.com/article/2502420/data-center/brain-drain-where-cobol-systems-go-from-here.html

[19] Ferguson, A. (2000). A History of Computer Programming Languages.

[20] Maynard, C., Jones, R., & Stewart, I. (2012, December 6). The Art of Lisp Programming. Springer Science & Business Media, p. 2.

[21] Nerlove, M. (2003). Programming Languages: A Short History for Economics. Maryland.

[22] Pausch, C. K. (2003). Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. Pittsburgh.

[23] Wirth, N. (2002). Pascal and is Successors. In e. a. M. Broy, Software are Engineering.Pioneers: Contributions to Soft. Springer-Verlag.

[24] Prinz, P., & Crawford, T. (2006, December 16). C in a Nutshell. Sebastopol: O'Reilly Media, Inc.

[25] Kernighan, B. W., & Ritchie, D. M. (1998). The C Programming Language (2nd ed.). New Jersey: Prentice Hall.

[26] Stroustrup, B. (2000). The C++ Programming Language. Addison-Wesley.

[27] International Organization for Standardization. (2014, December). Information Technology -- Programming Langages -- C++. (4).

[28] Kabutz, H. (2003, July 15). Once Upon an Oak. Retrieved from Artima: http://www.artima.com/weblogs/viewpost.jsp?thread=7555

[29] Gosling, J., Joy, B., Steele, G., Bracha, G., & Bukley, A. (2005). The Java Language Specification. Addison-wesley.

[30] Root, R., & Sweeney, M. R. (2006). A Tester's Guide to .NET Programming. USA: Apress Berkerly.

[31] Microsoft. (2015, October 15). Support Statement for Visual Basic 6.0 on Windows Vista, Windows Server 2001, Windows

7, Windows 8.1. Windows server 2012 and Windows 10. Retrieved from https://msdn.microsoft.com/en-us/vstudio/ms788708.aspx

[32] Mackenzie, D. (2006). Navigate The .NET Framework and Your Projects with the My Namespace. Retrieved from MSDN Magazine Visual Studio 2005 Guided Tour 2006: Microsoft

[33] Ecma International. (2006). C# Language Specification (4th ed.).

[34] Sethi, P. (2014, Oct-Dec). Programming Language. KAAV International Journal of Science, Engineering & Technology, 1(4).

[35] Sebesta, R. W. (2012). Concepts of Programming Languages Tenth Edition. Pearson Education, Inc.

[36] Theodore H. Romer, D. L.-L. (n.d.). The Struture and Performance of Interpreters. Seattle.

[37] Knuth, D. E., & Pardo, L. T. (2012). The Early Development of Programming Languages. In N. Metropolis, J. Howlett, & G. Rota, A History of Computing in the Twentieth Century: A Collection of Essays with Introductory Essay and Indexes (pp. 197-274). New York: NY: Academic Press/HBJ.

[38] Frost, R., & Callaghan, R. H. (2007). Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars. 10th International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE, 109-120.

[39] Terrence, P., & Zelkowitz, M. (2000). Programming Languages: Design and Implementation (4th ed.). Prentice Hall.

[40] Ben-Ari, M. (2005). Ada for Software Engineers. Chichester: John Wiley & Sons.

[41] Ryder, B. G., & Soffa, M. L. (2003). Influences on the Design of Exception Handling. ACM SIGSOFT Project on the Impact of Software Engineering Research on Programming Language Design.

[42] Aretz, F. K. (2003, June 30). The Djikstra-Zonneveld ALGOL 60 compiler for the Electrological X1. Software Engineering.

[43] Nelson, P. A. (2001, March 20). bc Command Manual. Retrieved from Free Software Foundation: https://www.gnu.org/software/bc/manual/html_mono/bc.html#SEC18

[44] Richards, M. (1969). BCPL - A Tool for Compiler Writing and Systems Programming. Proceedings of the Spring Joint Computer Conference, 34, pp. 557-566.

[45] Harbison, S. P., & Steele, G. L. (2002). C: A refernce Manual (5th ed.). Englewood Cliffs, NJ: Prentice Hall.

[46] Stroustrup, B. (2002). Sibling Rivalry: C and C++. New Jersey: AT&T Labs.

[47] Naugler, D. (May 2007). C# 2.0 for C++ and Java Programmer: Conference Workshop. Journal of Computing Sciences in Colleges.

[48] Hamilton, N. (2008, October 1). The A-Z of Programming Languages. Retrieved from Computerworld: http://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/?pp=7

[49] Chip, W. (2014). Programming and Problem Solving with C++ (6th ed.). Jones & Bartlet Learning.

[50] Lohr, S. (2008). Goto: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists and Iconoclasts--The Programmers who Created the Software Revolution. Basic Books, 52.

[51] Alexandrescu, A. (2010). The D Programming Language (1st ed.). Upper Saddle River, NJ: Addison-Wesley.

[52] Bright, W. (2011). D Programming Language Specification (e-book ed.). Digital Mars.

[53] OpenEuphoria. (n.d.). Welcome to OpenEuphoria. Retrieved from OpenEuphoria: http://openeuphoria.org

[54] Edwards, K. (2008, December 23). The A-Z of Programming Languages: F#. Retrieved from networkworld.com: http://www.networkworld.com/article/2271225/software/the-a-z-of-programming-languages--f-.html

[55] Syme, D., & Adam Granicz, A. C. (2007). Expert F#. Apress.

[56] Petricek, T. (2009). Real World Functional Programming With Examples in F# and C#. Manning Publications.

[57] Astborg, J. (2013). F# for Quantitative Finance. Packt Publishing.

[58] American Heritage Dictionary of English Language. (2011). FORTRAN. Retrieved from The Free Dictionary: http://www.thefreedictionary.com/FORTRAN

[59] Goerz, M. (2014). Modern Fortran Reference Card. Retrieved from http://michaelgoerz.net/refcards/fortran_refcard_a4.pdf

[60] Adams, V. P. (1981, October 5). Captain Grace M. Hopper: the Mother of COBOL. InfoWorld, p. 33.

[61] Chapman, S. J. (2007). Fortran 95/2003 for Scientists and Engineers (3rd ed.). McGraw-Hill.

[62] Pigott, D. (2006). FORTRAN - Backus et al high-level compiler (Computer Language). Encyclopedia of Computer Languages.

[63] Schmager, F., Cameron, N., & Noble, J. (2010). GoHotDraw: Evaluating the Go Programming Language with Design Patterns. Evaluation and Usability of Programming Languages and Tools. ACM.

[64] Balbaet, I. (2012). The Way to Go: A Thorough Introduction to the Go Programming Language. iUniverse.

[65] Chisnalle, D. (2012). The Go Programming Language Phrasebook. Addison-Wesley.

[66] Tang, P. (2010). Multi-core Parallel Programming in Go. Proc. First International Conference on Advanced Computing and Communications.

[67] Kienle, H. (2010, May-June). It's About Time to Take JavaScript (More) Seriously. IEEE Software, pp. 60-62.

[68] Meyer, D. (2009, August 20). Google Apps Script gets green light. Retrieved from CNet: http://news.cnet.com/8301-1001_3-10314002-92.html

[69] Konig, D., King, P., Laforge, G., D'Archy, H., Champeau, C., Pragt, E., & Skeet, J. (2015). Groovy in action, Second Edition. Manning.

[70] Stratchan, J. (2003, August 29). Groovy - the birth of a new dynamic language for the Java Platform. Retrieved from http://radio.weblogs.com/0112098/2003/08/29.html

[71] Peyton Jones, S. (2003). Haskell 98 Languages and Libraries: The Revised Report. Cambridge University Press.

[72] Marlow, S. (2010). Haskell 2010 Language Report.

[73] Bird, R. (2014). Thinking Functionally with Haskell. Cambridge University Press.

[74] Chaudhary, H. H. (2014, July 28). Cracking The Java Programming Interview :: 2000+ Java Interview Que/Ans. Retrieved from https://books.google.fr/books?id=0rUtBAAAQBAJ&lpg=PA133&pg=PA133#v=onepage&q&f=true

[75] Bezanson, J., Karpinski, S., Shah, V., & Edelman, A. (n.d.). Why we Created Julia. Retrieved June 16, 2017, from JuliaLang.org.

[76] Krill, P. (2012, April 18). New Julia language seeks to be the C for scientists. Retrieved from InfoWorld: http://www.infoworld.com/d/application-development/new-julia-language-seeks-be-the-c-scientists-190818

[77] Heiss, J. (2013, April). The Advent of Kotlin: A conversation with Jetbrain's Andrey Breslay. Retrieved from Oracle Technology Network: http://www.oracle.com/technetwork/articles/java/breslav-1932170.html

[78] Shafirov, M. (2012, May 17). Kotlin on Android. Now Official. Retrieved from Oracle:

http://www.oracle.com/technetwork/articles/java/breslav-1932170.html

[79] Waters, J. (2012, February 22). Kotlin Goes Open Source. Retrieved from Enterprise Computing Group: http://adtmag.com/articles/2012/02/22/kotlin-goes-open-source.aspx

[80] Wrox. (2012). Beginning Perl 1st Edition. A beginner's tutorial for those new to programming or just perl.

[81] Dominus, M. J. (2005). Higher Order Perl. Morgan Kaufmann.

[82] O'Reilly. (2012). Programming Perl 4th Edition. The Definitive Perl refrence.

[83] Cooper, P. (2009). Beginning Ruby: From Novice to Professional. Berkeley: APress.

[84] Metz, S. (2012, September 5). Practical Object-Oriented Design in Ruby. p. 272.

[85] Microsoft. (2013, June 7). Visual Basic Language Specification 11.0. Retrieved from Microsoft Corporation: http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=15039

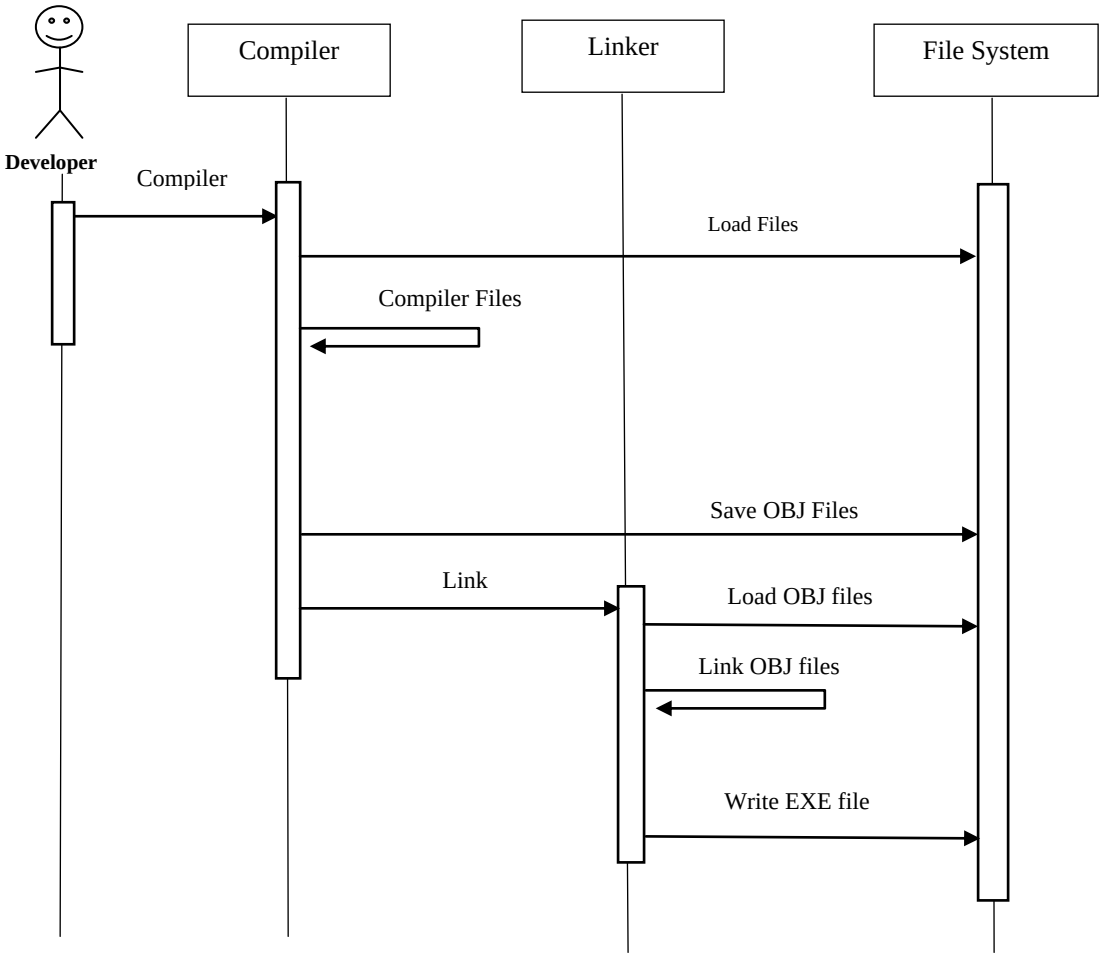

Figure 9. Sequence Diagram for *Streak*©

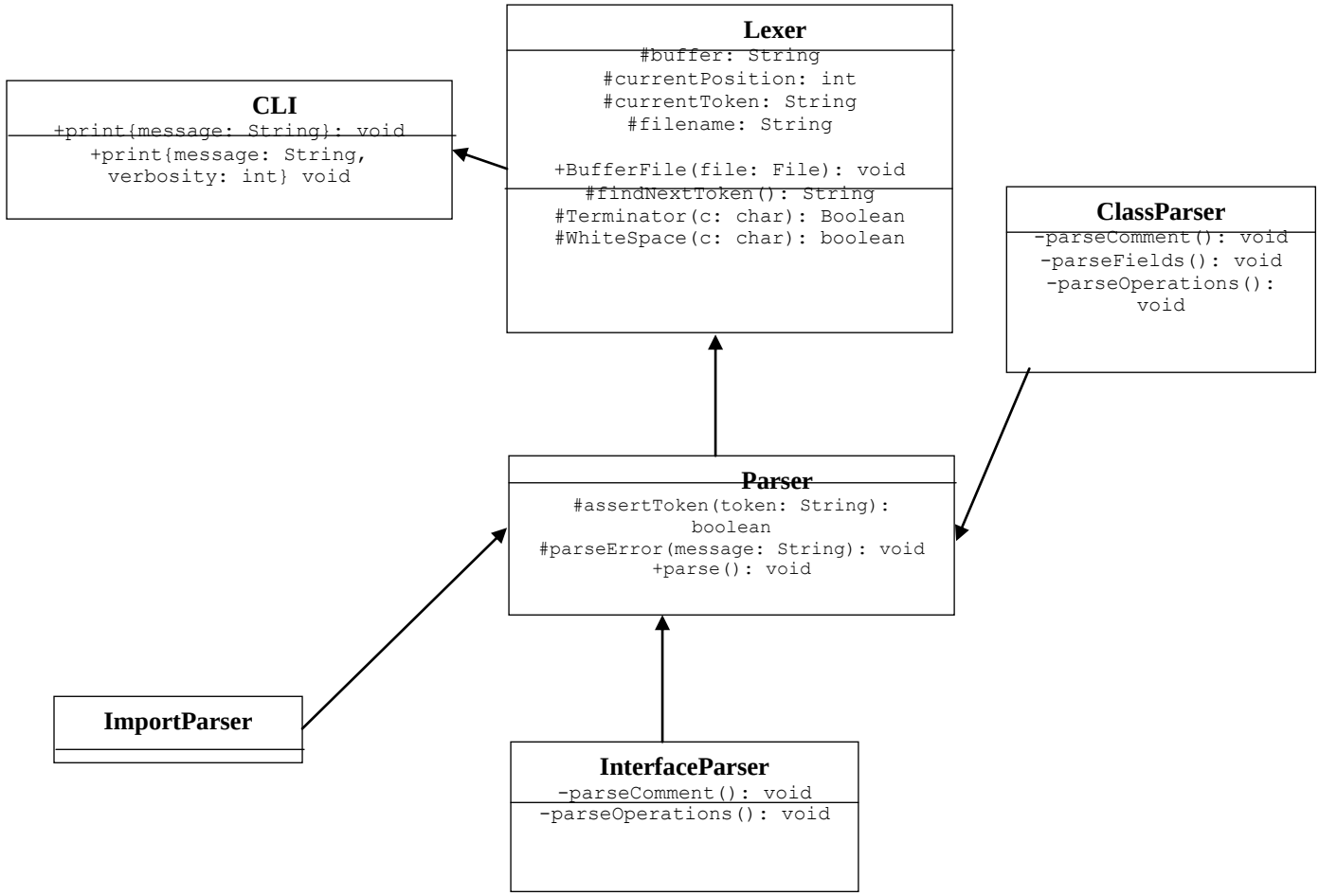

Figure 10. Streak©Class Hierarchy

APPENDIX: SAMPLE CODES IN STREAK©

A. Program to Post Increment any number

```
<PostIncrement>
streak
x;
streakvar
feedin x;
feedout x++;
end
Parse Completed
feedin: 5
feedout: 6
halt
```

B. Program to Sum Numbers from 1 to 1000

```
<Sum_of_numbers_from_1_to_1000>
streak
sm,i;
streakvar
sm = 0;
i = 1;
while i =< 1000 do sm = sm + i; i = i + 1; endwhile;
feedout sm;
end
Parse Completed
feedout: 500500
halt
```

C. Fibonacci Series

```
<FibonacciSeries>
streak
p,ne,s,n,c;
streakvar
feedin c;
n = 1;
p = 1;
ne = 1;
while n =< c do feedout p; s = p + ne; p = ne; ne = s; n = n + 1; endwhile;
end
Parse Completed
feedin: 10
feedout: 1
feedout: 1
feedout: 2
feedout: 3
feedout: 5
feedout: 8
feedout: 13
feedout: 21
feedout: 34
feedout: 55
halt
```

D. Post Decrement

```
<PostDecrement>
streak
x;
streakvar
feedin x;
feedout x--;
end
Parse Completed
feedin: 5
feedout: 4
halt
```

E. Factorial Program using for loop statement

```
<Factorial_program_using_for_loop>
streak
fac,i,num;
streakvar
feedin num;
fac = 1;
i = 1;
for i =< num do fac = fac * i; i = i + 1; endfor;
feedout fac;
end
Parse Completed
feedin: 10
feedout: 3628800
halt
```

F. Program to print the length of a String

```
<Strings>
streak
namelen;
streakvar
namelen = strlen("Victor");
feedout namelen;
end
Parse Completed
feedout: 6
halt
```

G. Testing Bitwise Operator

```
<Bitwise_Operator_program>
streak
streakvar
feedout 3|4;
feedout 3&4;
feedout 3 << 4;
feedout 3 >> 4;
end
Parse Completed
feedout: 7
feedout: 0
feedout: 48
feedout: 0
halt
```

H. Program to print even numbers from 0 to 100 inclusive

```
<EvenNumber>
streak
first,last;
streakvar
feedin first;
feedin last;
while first =< last do feedout first; first = first + 2; endwhile;
end
Parse Completed
feedin: 0
feedin: 100
feedout: 0
feedout: 2
feedout: 4
feedout: 6
feedout: 8
feedout: 10
feedout: 12
feedout: 14
feedout: 16
feedout: 18
feedout: 20
feedout: 22
feedout: 24
feedout: 26
feedout: 28
feedout: 30
feedout: 32
feedout: 34
feedout: 36
feedout: 38
feedout: 40
feedout: 42
feedout: 44
feedout: 46
feedout: 48
feedout: 50
feedout: 52
feedout: 54
feedout: 56
feedout: 58
feedout: 60
feedout: 62
feedout: 64
feedout: 66
feedout: 68
feedout: 70
feedout: 72
feedout: 74
feedout: 76
feedout: 78
feedout: 80
feedout: 82
feedout: 84
feedout: 86
feedout: 88
feedout: 90
feedout: 92
feedout: 94
feedout: 96
feedout: 98
feedout: 100
halt
```