Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Lab Professor / Lab Session:

Paolo Sausa / 012

Team:

Tasfia Khan Aohana - ID: 041076039 / Philogene Villanueva - ID: 041064813

Language Name: GoPhia

|  |  |
| --- | --- |
| **Part**  **1** | **Language User Reference** |

**EXPLANATION**

The purpose of this assignment is to invent a new computer language.

* This language can have the syntax and structure of your choosing.
* Option 1: Adapt the ‘Sofia language to be Go compatible (see <https://go.dev/>).
* Option 2: Define a **DSL** – Proper to solve specific problems (ex: science, economy, music, etc.)..

This is going to be a basic language. There's a lot of functionality that we'll be skipping over, while we implement the basics. You will need to tell me those basics, of course. In this document, I'm going to explain the steps of what to do with a bit of detail.

* 1. **User Manual**

**Element 1: Name / Extension**

Name of Language: **GoPhia**

Extension: **.gop**

The Language is inspired by: **C, Java, Go**

**Element 2 – Comments**

**Single line comment syntax: //**

example : //comment

**Multi-line comment syntax: /\* \*/**

example : /\*comment\*/

**Element 3 – Keywords**

break default func interface select  
case defer go map struct  
chan else goto package switch  
const fallthrough if range type  
continue for import return var

**Element 4 – Datatypes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Keyword** | **Number of Bytes** | **Range/ Description** |
| Integer | int32 | 4 | Signed 32-bit integer, range from -2,147,483,648 to 2,147,483,647 |
| Boolean | bool | 1 | Represents true or false values |
| Float | float32 | 32 | Single-precision floating-point number, roughly 7 decimal digits of precision |
| String | string | variable | A sequence of bytes representing a text string of variable length |
| Char | char | 1 | A single byte representing a character or small integer value, typically ASCII characters or extended character sets. |

**Element 5 – Variables**

*[Variables: How would a programmer define variables that can hold integer numbers (numbers with no decimal point), floating point numbers (numbers with a decimal point) or text (ie: strings in Java). This is element 1. Consider if you want to flag the variables in a special way, like SOFIA or BASIC, or not, like C or Java.]*

*var variable\_list optional\_data\_type*

*Ex.*

*var i, j, k int*

*var c, ch byte*

*var f, salary float32*

*d = 42*

**Element 6 – Methods / Functions**

*[Variables: How would a programmer define methods]*

**Functions are defined as:**

func function\_name( [parameter list] ) [return\_types]

{

body of the function

}

**Element 7 - Commands**

* ***Attribution / assignment****: How does your language let a programmer assign a value to a variable? (Will you allow casting? If so, how will it work?) How will your language handle math, and will it allow strings to be concatenated (merged)?*
  + var type variablename = value
* ***Selection****: How does your language do if-style logic? (Optional: Do you want to do some kind of switch/case as well?). You will need to explain how "conditionals" work in your language. How do you write Boolean operations, such as "or", "and", "not", and other conditions, such as less than, greater than, etc?*

*Syntax:*

***If statement:***

*if(boolean\_expression)*

*{*

*/\* statement(s) will execute if the boolean expression is true \*/*

*}*

***If, else statement:***

*if(boolean\_expression) {*

*/\* statement(s) will execute if the boolean expression is true \*/*

*} else {*

*/\* statement(s) will execute if the boolean expression is false \*/*

*}*

***Nested if statements:***

*if(boolean\_expression) {*

*/\* statement(s) will execute if the boolean expression is true \*/*

*} else {*

*/\* statement(s) will execute if the boolean expression is false \*/*

*}*

*Expression Switch − In expression switch, a case contains expressions, which is compared against the value of the switch expression.*

*Type Switch − In type switch, a case contain type which is compared against the type of a specially annotated switch expression.*

***Switch statement:***

*switch(boolean-expression or integral type){*

*case boolean-expression or integral type :*

*statement(s);*

*case boolean-expression or integral type :*

*statement(s);*

*/\* you can have any number of case statements \*/*

*default : /\* Optional \*/*

*statement(s);*

*}*

***Select statement:***

*select {*

*case communication clause :*

*statement(s);*

*case communication clause :*

*statement(s);*

*/\* you can have any number of case statements \*/*

*default : /\* Optional \*/*

*statement(s);*

*}*

* ***Interaction****: How will your code handle looping? (You can do one or more of a for-style loop, a while/do loop, etc.)*

1. ***for loop***
2. ***nested loops***

***Syntax for loop***

*for [condition | ( init; condition; increment ) | Range] {*

*statement(s)*

*}*

***Syntax for nested loop***

*for [condition | ( init; condition; increment ) | Range] {*

*for [condition | ( init; condition; increment ) | Range] {*

*statement(s);*

*}*

*statement(s);*

*}*

***Input****: How does your program get input from the keyboard? (Strings are easiest.)*

* ***Output****: What would a programmer type to put output on the screen? What sort of variables or data will your code take?*
* ***Functions****: [Function definition: parameters and returning types]*
  + *What will be the syntax for making a function or subroutine?*
  + *How will it take parameters?*
  + *How will it return results?*
* **Input:** All inputs are taken in as Strings.
  + To get input from the keyboard (stdin), the **fmt** package's **Scan, Scanf** or **Scanln** functions are used to read data as strings from the standard input.
    - Scan() - receives the user input in raw format as space-separated values and stores them in the variables. Newlines count as spaces.
    - Scanln() - similar to Scan(), but it stops scanning for inputs at a newline (at the press of the Enter key.).
    - Scanf() - receives the inputs and stores them based on the determined formats for its arguments.
* **Output:** 
  + The fmt package's Print, Printf, or Println functions are used to display values stored in variables or a string of characters.
    - Print() - prints its arguments with their default format.
    - Println() - similar to Print() with the difference that a whitespace is added between the arguments, and a newline is added at the end
    - Printf() - formats its argument based on the given formatting verb and then prints them.
      * %v is used to print the value of the arguments
      * %T is used to print the type of the arguments
* **Functions:** defined using the func keyword. Functions can take parameters, and they can also return values.
  + **Syntax:**

// Function definition with parameters and a return type

func functionName(parameter1 Type1, parameter2 Type2, ...) ReturnType {

// Function body

return value

}

**Element 7 – Proper elements**

*[Include specific features / elements to be included in your language]*

* *What you could include / modify? Think about new datatypes / structures / commands, etc.*
* *Note: Do not share this info (it is supposed to be your proper elements in the language.*
* Using Namespaces, so that method calls can be simplified

fmt.Println -> println

* Documentation comments, defined as:

/\*\*

\* This is a function called functionName, which …

/

|  |  |
| --- | --- |
| **Part**  **2** | **Language Comparison** |

**Comparing with C language**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | [Explanation]   * Every programming statement in C must be followed by a semicolon. This informs the compiler of the boundary between one statement and the subsequent one. Semicolons are acceptable in Go, however they are typically implied. * Although the majority of contemporary C compilers automatically initialize variables to zero, the C specification states that variables should instead get the value currently stored in memory. Values in Go are always initialized to zero. Go becomes more memory safe as a result of this. With pointers, this distinction gets even more intriguing. * Keep in mind that imported identifiers employ the Go package specifier. Like the C utilities printf and scanf from stdio.h, fmt for functions implements formatted input and output. * Go's main function consistently returns with the exit code 0. Call os if you want to return a different value.Where n is generally 1 to signal an error, use exit(n). To end the program, call this from any location—not just main. The exit(n) function, which is defined in stdlib.h, can be used in C to accomplish the same task. |  |

**Advantages / Disadvantages (in comparison with C)**

|  |  |  |
| --- | --- | --- |
|  | Disadvantage:   * Lack of libraries * Limited support for generics   Advantage:   * Elimination of semicolon at the end of every statements * Easy to learn with simple syntax * Built-in Garbage Collection |  |

**Comparing with another language**

**Language Name: Java**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | Java: Java relies on threads and locks for concurrency. It has a more traditional threading model.  Go: Go uses goroutines and channels for concurrency. Goroutines are lightweight threads, and channels facilitate communication between goroutines. Go's concurrency model is designed to be simpler and more efficient than traditional threading.  Typing:  Java: Java is both statically and strongly typed. It requires explicit type declarations and type checking at compile time.  Go: Go is also statically typed and strongly typed, but it allows for type inference in some cases, reducing the need for explicit type declarations.  Syntax:  Java: Java has a more verbose syntax compared to Go, with more boilerplate code required for common tasks.  Go: Go is known for its concise and minimalistic syntax, which promotes readability and simplicity.  Object-Oriented vs. Simplicity:  Java: Java is a class-based, object-oriented language that encourages the use of inheritance, encapsulation, and polymorphism.  Go: Go promotes a more composition-based approach to code reuse instead of relying heavily on inheritance. It prioritizes simplicity and ease of use.  Error Handling:  Java: Java uses exceptions for error handling, which can sometimes lead to complex code.  Go: Go uses explicit error values (usually returned as a tuple with the result), which can lead to more straightforward error handling.  Garbage Collection:  Java: Java has a garbage collector that manages memory automatically.  Go: Go also has a garbage collector, but it is designed to minimize latency spikes, making it more suitable for real-time applications.  Ecosystem and Libraries:  Java: Java has a vast and mature ecosystem with a wide range of libraries, frameworks (e.g., Spring, Hibernate), and tools.  Go: Go's ecosystem is growing but is still not as extensive as Java's. It is known for its simplicity and a "batteries-included" standard library.  Cross-Platform Support:  Java: Java applications are known for their "write once, run anywhere" capability, thanks to the Java Virtual Machine (JVM).  Go: Go also supports cross-platform development and can compile code for various operating systems and architectures.  Generics:  Java: Java has had generics since Java 5, allowing developers to write more type-safe and reusable code.  Go: Go introduced generics in version 1.18, addressing a long-standing limitation of the language.  In summary, Go and Java have different design philosophies and strengths, which make them suitable for different use cases. Go is often favored for its simplicity and efficiency in concurrent programming, while Java's maturity and extensive ecosystem make it a strong choice for various application domains, particularly in enterprise software development. The choice between them should be based on project requirements and developer preferences. |  |

**Advantages / Disadvantages (in comparison with this second language)**

|  |  |  |
| --- | --- | --- |
|  | Advantage   1. Concurrency Support: Go was designed with built-in support for concurrency through goroutines and channels. This makes it easier to write concurrent and parallel code compared to Java, which relies on threads and locks. 2. Strongly Typed and Statically Typed: Go is statically typed, which means that many errors can be caught at compile time, reducing the likelihood of runtime errors compared to Java, which is also statically typed but has a more complex type system. 3. Simplicity and Readability: Go is known for its simplicity and readability. It has a clean and minimalistic syntax, which can be easier for developers to understand and maintain compared to Java's more verbose syntax. 4. Fast Compilation: Go's compilation process is typically faster than Java's, which can lead to quicker development cycles. 5. Garbage Collection: Go has a garbage collector that helps manage memory efficiently, making it easier for developers to write memory-safe code. While Java also has garbage collection, Go's garbage collector is designed to minimize latency spikes, making it more suitable for real-time applications. 6. Small Binary Size: Go compiles to a single binary file with minimal dependencies, making it suitable for creating lightweight and portable applications. 7. Cross-Platform: Go supports cross-platform development and can compile code for various operating systems and architectures without much effort.   Disadvantage   1. Less Mature Ecosystem: Java has been around for a long time, and as a result, it has a vast and mature ecosystem with a wide range of libraries, frameworks, and tools. Go's ecosystem is growing but is still not as extensive as Java's. 2. Lack of Generics (until Go 1.18): One of the long-standing criticisms of Go was the lack of generics, which made it less expressive in certain situations. However, Go introduced generics in version 1.18, addressing this issue. 3. No Inheritance: Go uses composition instead of inheritance for code reuse, which can be a departure from the traditional object-oriented approach used in Java. Some developers may find this design paradigm less intuitive. 4. Less Enterprise Adoption: While Go is gaining popularity, Java is still the dominant language in many large enterprises, making it easier to find Java developers and support for Java-based systems. 5. Less Tooling: Java has a vast array of IDEs, build tools, and debugging tools, while Go's tooling is more minimalistic in comparison. However, Go's tooling is improving over time. |  |

|  |  |
| --- | --- |
| **Part**  **3** | **Architectural Questions** |

**Advantages**

*[What's the goal of your language? Are you trying to make something simple, fun, complicated? My personal language, Chambly, is based around being useful to scientists. (You can just make something up here, honestly. Think about it a little bit, have a little fun.)]*

The purpose of GoPhia is developed with the goal of being a simple language that can be a powerful tool for new programmers with its readability and efficiency. The language is designed to combine the minimalistic syntax with powerful performance and concurrency-support required to build modern, robust software systems.

**Strategy: C Implementation**

*[How your language can be implemented in C – ex: datatypes]*

* *In plain English, or maybe even some high-level pseudocode, how are you going to parse your language? You will be writing a compiler for your language, so these are some things you need to think about.*
* *Since the compiler will be implemented using ANSI C language, I need to respect definitions such as: char datatypes using one single byte (i.e. Unicode cannot be accepted) and using proper (new) structs to simulate datatypes that do not exist originally.*

***Note 1: C Datatypes***

*Remember that you are implementing your language in ANSI C. For this reason, you cannot create arbitrarily your language (from scratch). You need to use what is already provided by C Compiler. For this reason, think about using and defining the language obeying the datatypes.*

*[Your ideas about how to identify elements from language]*

* *Consider your "write to the console" command as an example. How will your compiler detect it? How will it sort out what to write to the console? What if there's some literal text (ie: "this is going to get printed") instead of variables?*

The code will follow a multi-step process of lexical analysis/tokenization, which involves breaking down the source code into smaller units called tokens.

1. **Lexical Analysis**: Breaks down the source code into tokens, identifying keywords, identifiers, literals, operators, and punctuation.
2. **Token Classification**: Categorizes tokens into types such as keywords, identifiers, literals, operators, and punctuation.
3. **Symbol Table**: Maintains a data structure to track identifiers and their attributes, facilitating scope resolution and type checking.
4. **Syntax Analysis**: Analyzes the arrangement of tokens according to the language's grammar rules, creating a parse tree or abstract syntax tree (AST).
5. **Semantic Analysis**: Ensures the code's meaning is correct by performing checks like type checking, scope analysis, and variable declaration verification.
6. **Intermediate Code Generation**: Produces an intermediate representation of the code for optimization and translation to lower-level code.
7. **Optimization**: Applies transformations to improve code efficiency, including reducing redundant operations and optimizing memory usage.
8. **Code Generation**: Translates the optimized code or intermediate representation into executable code, such as machine code or bytecode, for execution.

**The tokenization will allow the compiler to understand and correctly translate the code into the desired output. Literal texts will be output to the screen as texts, provided the correct syntax is followed for the output to console command. All outputs must be enclosed by () and double quotes, for literal texts.**

*[Your ideas about how to identify scope (ex: blocks between conditionals or functions)]*

* *How do you mark a block of code? If I use your loop logic, how do I control what portion of code gets looped through? In C, you might use { and }. In Python, the indentation is what matters. How does it work in your language?*

To mark blocks of code, curly braces ‘{ }’ are used to explicitly define the beginning and end of a block of code.

**FINAL SUGGESTIONS**

*Here some ideas to think about your language....*

* *Don't make this assignment harder than it needs to be on yourself. Focus on making the syntax for your language that meets our requirements. Worry about extra features later.*
* *Don’t worry if your new language winds up having really difficult parts. You'll be allowed to change your language as you go along, as long as you make "patch notes" to explain those changes. We'll tell you about this later.*
* *There's a marking key at the end of* ***CST8152\_Compilers\_F23-A11\_AnswerTemplate*** *that should steer you along for grades. Focus your efforts on where you'll get the best results.*
* *Finally, think about creating an “master-piece”: until now, you have used several languages. And if you have conditions to define yours, how it could be?*

**References**

*[Include eventual references used here]*

|  |  |
| --- | --- |
|  | * ***NOTE****: Even if you use any AI tool (ex: ChatGPT), report here, including the references used.* |

Algonquin College

Fall, 2023