Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Lab Professor / Lab Session:

[Paulo ]

Team:

[Meet Kotadiya] - Id: [041070018] / [Dhanush Bajaj ] - Id: [041067058]

Language Name [GOLANG]

***This template is suggested (not mandatory) to answer A11 Specification.***

|  |  |
| --- | --- |
| **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**

1. ***Language Name:***
   * *GoLang*
2. ***Filename Extension:***
   * ***.go*** *is the extension for Go*
3. ***Language Inspiration:***
   * *"GoLite is primarily patterned after the Go programming language. Our choice was inspired by Go's simplicity, efficiency, and readability. While GoLite takes many cues from Go, it's designed to be a more lightweight version, streamlined for beginners or for specific application domains. This makes GoLite a perfect choice for those wanting to grasp the essence of Go without diving deep into its intricacies."*

**Element 2 – Comments**

***Single-line Comments:***

* *For single-line comments, use the double forward slashes* ***//****. Anything after these slashes on the same line will be treated as a comment and will not be executed.*
* *Example:*

*// This is a single-line comment in GoLite.*

*let x = 10; // This is an inline comment.*

***Multi-line Comments:***

* *For comments that span multiple lines, wrap the comment content between* ***/\**** *and* ***\*/****.*
* *Example:*

*/\**

*This is a multi-line comment in GoLite.*

*It spans several lines and is ignored by the compiler.*

*\*/*

*let y = 20;*

**Element 3 – Keywords**

*Certainly! Here's a concise list of reserved keywords for "GoLite":*

* *`int`*
* *`float`*
* *`string`*
* *`var`*
* *`const`*
* *`if`*
* *`else`*
* *`for`*
* *`switch`*
* *`case`*
* *`default`*
* *`func`*
* *`return`*
* *`package`*
* *`import`*
* *`true`*
* *`false`*
* *`and`*
* *`or`*
* *`not`*
* *`break`*
* *`continue`*
* *`defer`*
* *`struct`*
* *`interface`*
* *`nil`*

**Element 4 – Datatypes**

* ***int****: Integer data type.*
  + *Bytes: 4 (32-bit)*
  + *Range: -2,147,483,648 to 2,147,483,647*
* ***float****: Floating-point numbers.*
  + *Bytes: 4 (32-bit)*
  + *Range: Approximately ±3.4E−38 to ±3.4E+38 with 7 decimal places of precision.*
* ***string****: Sequence of characters.*
  + *Bytes: Variable (depends on the length of the string)*
  + *Range: Can store any Unicode character.*

**Element 5 – Variables**

* *Integer variables:*
  + *Syntax:* ***var variableName int = value;***
  + *Example:* ***var count int = 10;***
* *Floating point variables:*
  + *Syntax:* ***var variableName float = value;***
  + *Example:* ***var area float = 45.6;***
* *String variables:*
  + *Syntax:* ***var variableName string = "value";***
  + *Example:* ***var greeting string = "Hello, GoLite!";***

**Element 6 – Methods / Functions**

* *Function definition:*
  + *Syntax:* ***func functionName(parameterName1 dataType1, parameterName2 dataType2) returnType {***
  + *Example:* ***func addNumbers(a int, b int) int {***
* *If the function doesn't return any value:*
  + *Syntax:* ***func functionName(parameterName1 dataType1, parameterName2 dataType2) {***
  + *Example:* ***func displayMessage(message string) {***

*To call or use the function:*

* *Syntax:* ***functionName(value1, value2);***
* *Example:* ***addNumbers(5, 3);*** *or* ***displayMessage("Hello!");***

**Element 7 - Commands**

* + ***1. \*\*Attribution / Assignment:\*\****
  + ***- Syntax: `variableName = value;`***
  + ***- Casting: `variableName = (dataType)value;`***
  + ***- Math Operations: Standard operators like `+`, `-`, `\*`, `/`, and `%`.***
  + ***- String Concatenation: Using the `+` operator.***
  + ***- Example: `string1 + string2;`***
  + ***2. \*\*Selection:\*\****
  + ***- `if` statement:***
  + ***- Syntax: `if (condition) { ... } else { ... }`***
  + ***- `switch` statement (optional):***
  + ***- Syntax:***
  + ***```***
  + ***switch (variable) {***
  + ***case value1:***
  + ***...***
  + ***break;***
  + ***case value2:***
  + ***...***
  + ***break;***
  + ***default:***
  + ***...***
  + ***}***
  + ***```***
  + ***- Boolean Operations:***
  + ***- `and` for logical AND (`&&` in many languages)***
  + ***- `or` for logical OR (`||` in many languages)***
  + ***- `not` for logical NOT (`!` in many languages)***
  + ***- Comparison Operations:***
  + ***- `<`, `>`, `<=`, `>=`, `==`, `!=`***
  + ***3. \*\*Interaction:\*\****
  + ***- `for` loop:***
  + ***- Syntax: `for (initialization; condition; update) { ... }`***
  + ***- `while` loop:***
  + ***- Syntax: `while (condition) { ... }`***
  + ***4. \*\*Input:\*\****
  + ***- Syntax: `input(variableName);`***
  + ***- Example: To get a string input, `input(userName);`***
  + ***5. \*\*Output:\*\****
  + ***- Syntax: `print(variableOrLiteral);`***
  + ***- Example: `print("Hello, " + userName + "!");`***
  + ***6. \*\*Functions:\*\****
  + ***- Definition:***
  + ***- Syntax: `func functionName(parameter1 dataType1, parameter2 dataType2) returnType { ... }`***
  + ***- Example: `func add(a int, b int) int { ... }`***
  + ***- Calling a function:***
  + ***- Syntax: `functionName(argument1, argument2);`***
  + ***- Example: `sum = add(5, 3);`***
  + ***These commands and syntaxes provide a foundation for "GoLite" while ensuring that the language is intuitive and aligns with many modern programming languages.***

**Element 7 – Proper elements**

*### Element 7 – Proper Elements*

*The goal of introducing proper elements is to add unique features to "GoLite" that set it apart from other languages while enhancing its capabilities. Here's a set of potential elements:*

*1. \*\*Enhanced String Manipulation:\*\**

*- `substr(stringVar, startIndex, length)`: Extracts a substring from a given string.*

*- `contains(stringVar, subString)`: Checks if the string contains a specific substring.*

*2. \*\*Tuple Data Type:\*\**

*- Allows storing multiple values of varied data types together.*

*- Syntax: `var tupleVar = (value1, value2, value3);`*

*3. \*\*List Comprehensions:\*\**

*- Provides a concise way to create lists.*

*- Syntax: `[expression for item in iterable if condition]`*

*4. \*\*Pattern Matching:\*\**

*- Allows checking a value against a pattern.*

*- Syntax:*

*```*

*match (variable) {*

*pattern1 => action1,*

*pattern2 => action2,*

*\_ => defaultAction // "\_" serves as a wildcard*

*}*

*```*

*5. \*\*Lambda Functions (Anonymous Functions):\*\**

*- Define small functions without naming them.*

*- Syntax: `var lambdaVar = (parameter) => expression;`*

*6. \*\*Extension Methods:\*\**

*- Allows adding new methods to existing data types without modifying them.*

*- Syntax: `func dataType.extensionMethodName(parameters) returnType { ... }`*

*7. \*\*Integrated Testing Framework:\*\**

*- Introduce a `test` keyword to define unit tests within the code, making testing an integral part of "GoLite".*

*- Syntax:*

*```*

*test functionName {*

*assert(expression, expectedValue);*

*}*

*```*

*8. \*\*Chainable Methods:\*\**

*- Allows methods to be chained together for more concise code.*

*- Example: `list.filter(condition).map(transformation).reduce(operation);`*

*9. \*\*Optional and Named Parameters in Functions:\*\**

*- Provides more flexibility in function calls.*

*- Syntax: `func exampleFunc(requiredParam, optionalParam = defaultValue, namedParam: value) { ... }`*

*These unique elements can make "GoLite" more versatile and distinctive. They cater to modern programming needs while introducing features that can simplify and optimize code writing.*

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

**Comparing with C language**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | 1. **Syntax:**    * "GoLite" has a more modern syntax, taking inspiration from Go. For instance, "GoLite" uses **var** for variable declarations, whereas C uses data type identifiers like **int**, **float**, etc.    * "GoLite" supports lambda functions and pattern matching, features not native to C. 2. **Data Types:**    * While both languages support fundamental data types like integers, floats, and strings, "GoLite" introduces tuples, allowing for grouped data of varied types. 3. **Memory Management:**    * C relies heavily on manual memory management with **malloc()** and **free()**. "GoLite", being a higher-level language, abstracts away such complexities. 4. **Functions:**    * "GoLite" supports optional and named parameters in functions, offering more flexibility compared to C's strict function signatures. 5. **Error Handling:**    * C traditionally uses error codes and global variables like **errno**. "GoLite" might employ a more modern approach, using mechanisms similar to Go's multiple return values for errors. 6. **Integrated Testing:**    * "GoLite" introduces an integrated testing framework using the **test** keyword, a feature not present in C. 7. **Modularity:**    * "GoLite" might have a more structured package and module system inspired by Go, whereas C relies on header files and source files for modularity. |  |

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

|  |  |  |
| --- | --- | --- |
|  | **Advantages:**   1. **Simplified Syntax:** "GoLite" offers a more intuitive and modern syntax, making it easier for beginners to pick up. 2. **Enhanced Features:** With features like pattern matching, lambda functions, and integrated testing, "GoLite" provides advanced capabilities out of the box. 3. **Memory Safety:** Abstracting away direct memory management can reduce the risk of issues like buffer overflows, common in C.   **Disadvantages:**   1. Performance: C, being a lower-level language, often delivers better performance, especially for system-level tasks. 2. Maturity and Libraries: C has been around for decades and boasts a vast array of libraries and tools. "GoLite", being new, would not have such a rich ecosystem. 3. Direct Hardware Access: C offers more direct access to computer memory and hardware, which might be preferable for certain applications like embedded systems programming. |  |

**Comparing with another language**

**Language Name: JAVA**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | 1. **Platform Dependency:**    * Java is platform-independent due to its "Write Once, Run Anywhere" philosophy, which relies on the Java Virtual Machine (JVM). "GoLite" doesn't specify such a virtualization layer and may be platform-dependent. 2. **Syntax:**    * Java requires classes and objects for most operations, adhering to its object-oriented paradigm. "GoLite", inspired by Go, has a more procedural approach with optional object-oriented features. 3. **Primitive Data Types:**    * Java has specific primitive data types like **int**, **double**, **char**, etc., whereas "GoLite" simplifies this with generalized types like **int**, **float**, and **string**. 4. **Memory Management:**    * Java has automatic garbage collection. While "GoLite" abstracts direct memory management like C, it doesn't specify a garbage collector. 5. **Error Handling:**    * Java uses exceptions for error handling, while "GoLite" might employ a return value-based approach, inspired by Go. 6. **Concurrency:**    * While Java supports concurrency through threads, "GoLite" might introduce a more simplified concurrency model inspired by Go's goroutines. 7. **Modularity:**    * Java uses packages and has a class-based structure. "GoLite" might employ a simpler package system without the need for class-based organization. |  |

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

|  |  |  |
| --- | --- | --- |
|  | **Advantages:**   1. **Simplified Syntax:** "GoLite" offers a more streamlined syntax without the need for boilerplate code, which is often required in Java (like public static void main). 2. **Lightweight:** Without the need for a virtual machine (JVM), "GoLite" programs might be lighter and have faster startup times. 3. **Procedural Approach:** For developers who prefer procedural programming over object-oriented, "GoLite" provides a more familiar environment.   **Disadvantages:**   1. **Platform Independence:** Java's ability to run on any platform with a JVM is a significant advantage that "GoLite" lacks. 2. **Maturity and Libraries:** Java, with its vast ecosystem, offers libraries and frameworks for almost every imaginable task. "GoLite" wouldn't have such a breadth of external support. 3. **Garbage Collection:** Java's automatic garbage collection aids in memory management, reducing the risk of memory leaks, which might not be as optimized in "GoLite". |  |

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

**Advantages of GoLite:**

**Goal and Vision:**  
The primary goal of "GoLite" is to bridge the gap between high-level, feature-rich languages and low-level, performance-oriented languages. In the vast spectrum of programming languages, there's a niche for a language that offers modern conveniences while retaining a close-to-the-metal efficiency. "GoLite" seeks to fill that niche.

1. **Simplicity and Readability:**
   * "GoLite" is designed with clarity in mind. By reducing boilerplate and introducing an intuitive syntax, new learners can quickly grasp its concepts, while seasoned developers can appreciate its succinctness.
2. **Efficiency and Performance:**
   * While being lightweight, "GoLite" doesn't compromise on performance. Its design principles ensure that programs written in "GoLite" can execute swiftly, making it suitable for real-time applications.
3. **Portability and Compatibility:**
   * "GoLite" aims to be platform-agnostic. While it doesn't possess the JVM-level universality of Java, its design ensures easy compilations across major platforms.
4. **Domain Adaptability:**
   * Whether it's web development, system programming, or data science, "GoLite" provides the tools and structures to adapt. Its core features, combined with extensibility, allow it to mold according to the domain's needs.
5. **Community and Evolution:**
   * The vision for "GoLite" is not just a static language but an evolving ecosystem. With open-source principles at its heart, the language will evolve with the community's contributions, ensuring it stays relevant with changing technological trends.

**Niche Use-Case: Educators and Students:**  
Taking a leaf from the "Chambly" playbook, which is tailored for scientists, "GoLite" finds its unique space among educators and students. Its simplicity makes it an excellent tool for teaching programming fundamentals without overwhelming beginners. At the same time, its efficiency ensures that students can undertake complex projects as they advance, all within the "GoLite" ecosystem. This dual nature makes "GoLite" a preferred choice for academic institutions, coding bootcamps, and self-learners.

In essence, "GoLite" isn't just another programming language; it's a philosophy—a commitment to simplifying code without sacrificing capability. It's the embodiment of the belief that programming should be accessible, fun, and efficient, all at once.

**Strategy: C Implementation**

*### Strategy: C Implementation*

*\*\*1. Implementing Data Types in C:\*\**

*Given that we are basing the implementation in C, the data types of "GoLite" can be mapped to C data types:*

*- `int` in "GoLite" can be represented using `int` in C.*

*- `float` can be translated to `float` or `double` in C, based on precision requirements.*

*- `string` can be translated to character arrays (`char[]`) in C.*

*\*\*2. Parsing the Language:\*\**

*A lexical analyzer (often called a lexer) will be used to break the source code into tokens. These tokens represent the smallest units in the language, such as variables, operators, and literals.*

*Pseudocode for parsing:*

*```*

*function parse(source\_code):*

*tokens = tokenize(source\_code)*

*abstract\_syntax\_tree = generate\_AST(tokens)*

*return abstract\_syntax\_tree*

*```*

*\*\*3. Identifying Language Elements:\*\**

*To recognize commands or functions in the language, a combination of regular expressions and grammar rules can be employed.*

*For the "write to the console" command, let's assume it's represented as `print` in "GoLite". When the lexer encounters the word `print`, it categorizes it as a function call. The subsequent tokens, until a semicolon or line break, will be considered as parameters to this function.*

*Pseudocode for handling `print`:*

*```*

*if token == "print":*

*next\_token = get\_next\_token()*

*if next\_token == "(":*

*content\_to\_print = get\_content\_until(")")*

*output\_to\_console(content\_to\_print)*

*else:*

*error("Expected '(' after print")*

*```*

*For literal text, it can be detected if the token starts and ends with double quotes. If so, the content within is treated as a string literal.*

*\*\*4. Identifying Scope and Blocks:\*\**

*To handle scope in "GoLite", we can use curly braces `{` and `}` similar to C. When the parser encounters an opening brace, it marks the beginning of a new block or scope. All subsequent tokens are considered part of this block until a closing brace is encountered.*

*Pseudocode for scope:*

*```*

*if token == "{":*

*current\_scope = new\_scope(previous\_scope)*

*while next\_token != "}":*

*process\_token\_within\_scope(next\_token, current\_scope)*

*```*

*In essence, while "GoLite" has its unique syntax and features, its underlying representation leans on the foundational constructs provided by C. This approach ensures that the language's implementation is both efficient and feasible.*

**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**

1. *The Go Programming Language Documentation.*
2. *The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie.*
3. *ChatGPT*

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

Algonquin College

Fall, 2023