1 Basic Language Features to be Implemented

1.1 Lexical Structure

Code is composed of ascii characters only.

- **Comments**: line comments ( // comment) and general comments ( /* comment */)
- **Identifiers**: String of alphanumeric characters (with underscore), with first character not a digit.
- **Keywords**:
  break case const continue default else fallthrough for func if import package range return struct switch type var
  optional keywords: go goto interface map chan select defer (may not be implemented)
- **Operators and Delimiters**:
  
  + & += &= && == != ( )
  - | -= &= && == ! = ( )
  * ^ *= ^= || < <= [ ]
  / << /= <<= ++ = := , ;
  % >> %= >>= -- ! ... . :
  &^ &^=

- **Literals**: Integer, character and floating point literals as in C.
  We also have length terminated immutable string literals like "abc" or 'abc' (raw string literal).

1.2 Types

- **Boolean Type**: bool takes values true and false.
- **Numeric Types**:
  int int8 int16 int32 int64
uint uint8 (= byte) uint16 uint32 uint64
float float32 float64
uintptr (an integer big enough to store a pointer)

- **Array Type**: [32]byte arr1
  [3][5]int arr2

- **Slice Type**: slice is a reference to a part of array
  a = arr[5:9] // a is a slice of array arr

- **String Type**: store immutable string literals

- **Struct Type**: basic struct type similar to C, however structs may have methods associated with them
  struct {
    x, y int
    u float32
  }
  We will implement automatic pointer indirection with accessing struct members.
  var ptr *point
  ptr.x = 2 // shortcut for (*ptr).x = 2

- **Pointer Types**: A pointer type can be associated with a type (like *int with int)

- **Function Types**: can store function literals
  
  func(a, b int) bool
  // is a function type which takes two ints and returns a bool

1.3 Constants

We will support only typed constants ie; a constant is simply a typed variable whose value cannot be changed.

const val int = 5

1.4 Declarations

- constant decl. like: const u, v float32 = 0, 3
- type decl. like: type IntArray [16]int
- variable decl. like: var x, y float32 = -1, -2 and short versions like: var i = 0
- function decl. like:
  func min(x int, y int) int {
    if x < y {
      return x
    }
    return y
  }
Functions may return multiple values.

- method decl. like:
  ```go
  func (p *Point) Length() float64 {
    return math.Sqrt(p.x * p.x + p.y * p.y)
  }
  ```

1.5 Expressions

- **Function Literals**: We will implement C function pointer analogue of function literals. However, they will not be closures i.e., they cannot refer to variables defined in a surrounding function.
- Arithmetic expressions.
- Assignment expressions. These include multiple assignments using comma operators.
  ```go
  i, j = 2, 3
  ```
- Function and method calls.
- Type conversions like: `str = string(-1)`

1.6 Statements

- Expressions statements comprising of expressions (described in previous section).
- IncDec statements like: `i++;
- If statement with optional initialization
  ```go
  if x := f(); x < y { return x }
  ```
- Switch statements: expressions not only constants are permitted in switch in GO.
  ```go
  switch tag { case tag < 0 : .... }
  ```
- for statement: similar to as in C.
  ```go
  for i := 0; i < 10; i++ { ... }
  for a { ... } // GO analogue of while in C
  ```
- return, break, continue, fallthrough statements

1.7 Built-in Functions

- `len(entity)`: gives length of entity
- `cap(entity)`: gives capacity of entity
- `new(type)`: allocates memory of size equal to that of type and returns a pointer to it.
- `make(type,length,capacity)`: returns a slice of given length, capacity and type.
1.8 Packages

We will implement importing and declaring package.

```go
default
package abc
import "xyz"
```

2 Extensions

- Interface Type implementation.
- Maps
- Automatic garbage collection
- Goroutines (basic implementation only)

3 Features not to be implemented

- UTF-8 characters support
- Imaginary literals
- Untyped constants
- Closure for function literals
- defer statements
- Advanced concurrency related features