# The Go Programming Language Specification

***Group Members:***
***Shubh Agrawal 190828***
***Shubhankar Gambhir 190835***
***Tarun Kanodia 190902***
***V Pramodh Gopalan 190933***

***Group - 24***

**Source language: Go**
**Implementation language: C++**
**Target language: x86 (64 bit) assembly**

# Identifiers

Identifiers name program entities such as variables and types. An identifier is a sequence of one or more letters and digits. The first character in an identifier must be a letter in Golang.

```
identifier = letter { letter | unicode_digit } .
```

**Native Data types (integer, boolean, character)**

- byte (used as char)
- bool
- Int

# Keywords

The following keywords are reserved and may not be used as identifiers.

```
break       func       struct       else       package       if
continue    for        import       return     var
true        false      type
```

# Handling I/O in Go

Input is taken from `STDIN` using `"scanf"` while output is printed to `STDOUT` using `"printf"`.

```
//following is code to print something onto the output stream
package main
import "fmt" // need to import this library
func main() {
    printf("!... Hello World ...!")
}
```

```
// Printing Variables
var first int = 5
printf(“%d”,first)
// Taking Inputs (Follow this format only)
var first int
second := &first
scanf(“%d”, second)

// %d is for int , %c is for byte , %t is for bool
```

# Variable declaration -

**Variables can be declared in Go using the following syntax:**

```
// declaring and initializing the variable
var a int = 30
// declaring and initializing the variable  of char type
var my_char byte = 'a'
var b bool = true
```

**Note:** *Multiple assignments can be done as follows:*

```
var x,y int  = 2,3
```

**Shorthand declarations are also supported in Go:**

```
// declaring and initializing the variable and is of type int
    a:= 30
```

```
// declaring and initializing the variable  of boolean type
    b:= true
```

**Note:** *Multiple assignments can be done as follows:*

```
x, y := 2, true
```

# Expressions

**Primary expressions:**

Primary expressions are the operands for unary and binary expressions.
Example: `x`, 2, `f(3, true) etc.`

**Operators:**
Go supports all standard arithmetic operators like addition, subtraction, multiplication, division
The expressions are of the form:

```
Expression = UnaryExpr | Expression binary_op Expression
```

where `binary_op` denotes the binary operators

**List of unary operators: "+" , "-" , "!" , "*" , "&"**
**List of binary operators: "||" , "&&" , "==" , "!=" , "<" , "<=" , ">" , ">=" , "+" , "-" , "|" , "^", "*" , "/" , "%" , "<<" , ">>" , "&", “++” , “--” , “+=”, “-=” , “*=” , “/=” , “&=” , “|=” “^=” , “%=” , “<<=” , “>>=”.**

**Operator precedence:**
Unary operators have the highest precedence. As the `++` and `--` operators form statements, not expressions, they fall outside the operator hierarchy**.**
There are five precedence levels for binary operators. Multiplication operators bind strongest, followed by addition operators, comparison operators, **`&&` (logical AND),** and finally **`||` (logical OR).**

# Conditionals

## Single `if` condition:

```
var v int = 700
     if(v < 1000) {
         // print the following if condition evaluates to true
        printf("v is less than 1000\n")
    }
```

## `if else` condition

```
var v int = 700
     if(v < 500) {
         // print the following if condition evaluates to true
        printf("v is less than 500\n")
    } else if (v<=1000){
   // print the following if we have the if condition evaluates
   //to false and else if condition evaluates to true
        printf("v is less than 1000\n")
    }
```

## `if-else if-else` Condition

```
 var v int = 700
  if(v < 500) {
        printf("v is less than 500\n")
    } else if (v<=600){
        printf("v is less than 600\n")
    } else {
         // print the following if condition and the else if
          //condition evaluates to true
         printf(" yoyo")
    }
```

# Loops:

## General:

```
for [condition |  ( init; condition; increment )] {
    // statements
}
```

## `for` loops:

```
for i := 0; i < 4; i++{
    // statements
 }
```

The above code runs all the statements inside the for loop 4 times.

## `while` loops:

```
 j:=0
 for j<10{
          printf(“%d\n”, j)
          j+=1
  }
```

The above code prints all numbers from 0 to 9, each number on new line.

## `break` statements:

```
j := 0
 b := true
 for b == true {
       printf("%d\n", j)
       j += 1
       if j == 5 {
             // when j==5, this code segment is executed,
             // thus exiting the loop
             break
       }
 }
```

The above code prints all numbers from 0 to 4, each number on new line.

## `continue` statements:

```
j:=0
for j<=5 {
        j+=1
        if(j==2){
         // when j==2, this code segment is executed, thus
         // skipping the later instruction(print instruction here)
         // for this iteration of the loop
                continue
        }
        printf("%d\n", j)
}
```

This code prints numbers 1 3 4 5 6 with each number on a new line.

# Array:

## General:

```
var variable_name [SIZE]variable_type
```

## Example:

```
var myarr [3]int
// Elements are assigned using index
myarr[0] = 561
myarr[1] = 872
myarr[2] = 1289
// Accessing the elements of the array using index value
printf("Elements of Array:\n")
printf("Element 1: %d\n", myarr[0])
printf("Element 2: %d\n", myarr[1])
printf("Element 3: %d\n", myarr[2])
--------Output-----------
Elements of Array:
Element 1:  561
Element 2:  872
Element 3:  1289
```

# Functions:

## General:

```
func <name of function> (params, ... ) <return type>{
     // statements
}
```

params is of the form = <variable name> <variable type>

## Example:

```
func area(length int, width int)int{
    Ar := length* width
    return Ar
}
```

This function takes 2 inputs which are length and width both of type integer and it just multiplies and returns the result.
**Note:** We have also supported multiple returns from a function. Example:

```
package main

import "fmt"

func char_ascii() (char, int) {
     return 'a', 97
}
func main() {
     x, y := char_ascii()
     printf("Character: %c, ASCII value: %d\n", x, y)
}

Output:
Character: a, ASCII value: 97
```

# Structures:

## Declaration:

### General:

```
type <struct name> struct{
     Member1 datatype
     Member2 datatype
     Member3 datatype
      ...
}
```

### Example:

```
package main

import "fmt"

type Point struct {
     X     int
     Y     int
     Z     int
     label byte
}

func main() {
     var p Point
     p.X = 1
     p.Y = 2
     p.Z = 4
     p.label = 'c'
     printf("%d %d %d %c\n", p.X, p.Y, p.Z, p.label)
}
```

## Assigning values:

### General:

```
var <var name> <struct name>
<var name>.<Member1> = val1
<var name>.<Member2> = val2
<var name>.<Member3> = val3
```

### Example:

```
var a Point;
a.X = 0
a.Y = 1
a.Z = 2
a.label = ‘a’
```

## Accessing values:

### General:

```
<var name>.<Member>
```

### Example:

```
printf("X: %d", a.X)
printf("Y: %d", a.Y)
printf("Z: %d", a.Z)
printf("Label: %d", a.label)
```

```
--------Output-----------
X:  0
Y:  1
Z:  2
Label:  a
```

# Pointers:

## Declaration:

### General:

```
var <var name> *type
```

### Example:

```
var a *int
```

## Usage:

### General:

```
*<var name> = val
```

### Example:

```
package main

import "fmt"

func main() {
	var b int = 3
	a := &b
	*a += 2
	printf(“Address of b variable: %x\n", &b)
	printf("Address stored in a variable: %x\n", a)
	printf("Value of *a and b: %d %d\n", *a, b)

}
```

```
--------Output-----------
Address of b variable: ef802868
Address stored in a variable: ef802868
Value of *a and b: 5 5
```

## Some pointers about implementation:

1.) Comments need to be inserted on a new line rather than on the same line.
2.) The test cases expect a new line at the end of the file (for lexer to know the end)
3.) Function parameters must be of basic type only.
4.) Struct Field Members must be of basic type only.
5.) Scanf format needs to be precisely as shown in the above examples: when taking input for a variable, a second pointer variable needs to be created and assigned address of the first variable, rather than passing the address of first variable directly to scanf.