variable✔

function declarations✔

blocks✔

conditionals statements and expressions✔

while loops->only for loop in Go✔

|  |  |  |
| --- | --- | --- |
|  | Go | JSONAST |
| Variable  DONE | package main  func main() {  var x = 3  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"x"}],"expr":[{"tag":"lit","val":3}]}]}}  DONE |
|  | package main  func main() {  var e,f int  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"e"},{"tag":"nam","sym":"f"}],"expr":[{"tag":"lit","val":0},{"tag":"lit","val":0}]}]}}  DONE  Only return last f |
|  | package main  func main() {  var a, b, c, d = "hi", 1, 1.12, true  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"a"},{"tag":"nam","sym":"b"},{"tag":"nam","sym":"c"},{"tag":"nam","sym":"d"}],"expr":[{"tag":"lit","val":"hi"},{"tag":"lit","val":1},{"tag":"lit","val":1.12},{"tag":"lit","val":true}]}]}}  DONE  Only return last d |
|  | package main  func main() {  f, g := "apple", "pine"  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"f"},{"tag":"nam","sym":"g"}],"expr":[{"tag":"lit","val":"apple"},{"tag":"lit","val":"pine"}]}]}}  DONE  Only return last g |
|  | package main  func main() {  var x = 3+4  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"x"}],"expr":[{"tag":"binop","sym":"+","frst":{"tag":"lit","val":3},"scnd":{"tag":"lit","val":4}}]}]}}  DONE 7 |
| conditional | package main  func main() {  n := 0  if n > 0 {  n += 1  } else if n == 0 {  n += 1  } else {  n -= 1  }  } | {"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"n"}],"expr":[{"tag":"lit","val":0}]},{"tag":"cond","pred":{"tag":"binop","sym":">","frst":{"tag":"nam","sym":"n"},"scnd":{"tag":"lit","val":0}},"cons":{"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"n"}],"expr":[{"tag":"lit","val":1}]}]}},"alt":{"tag":"cond","pred":{"tag":"binop","sym":"==","frst":{"tag":"nam","sym":"n"},"scnd":{"tag":"lit","val":0}},"cons":{"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"n"}],"expr":[{"tag":"lit","val":1}]}]}},"alt":{"tag":"blk","body":{"tag":"seq","stmts":[{"tag":"assmt","sym":[{"tag":"nam","sym":"n"}],"expr":[{"tag":"lit","val":1}]}]}}}}]}} |
| for loop | package main  import "fmt"  func main() {  x := 0      i := 0      for i < 100 {          j := 0          for j < 100 {              x = x + i + j              j = j + 1          }          i = i + 1      }  } |  |
| function declare& call | package main  import "fmt"  func fact(n int) int {      return factIter(n, 1, 1)  }  func factIter(n, i, acc int) int {      if i > n {          return acc      } else {          return factIter(n, i+1, acc\*i)      }  }  func main() {      result := fact(4)  } |  |

Standard Project 2: Concurrent virtual machine for Go

The Concurrent virtual machine project applies the notion of a virtual machine (Module 4) to a concurrent programming language. Baseline expectations:

* Web-based implementation based on Source Academy frontend and js-slang (see [GitHubLinks to an external site.](https://github.com/source-academy))
* Implementation of a sublanguage of Go, consistent with a recent language specification
* Sequential language constructs: variable and function declarations, blocks, conditionals statements and expressions, while loops
* Concurrent constructs: Go routines, concurrency control (wait, wait group etc)
* Implementation should use a virtual machine
* Implementation must use a low-level memory model, and all runtime data structures need to be allocated from a single ArrayBuffer.
* Optional components:
  + visualization of heap and runtime stack
  + type checking
  + memory management