Sucharita Das

Principle of Programming Language

Assignment 2

Date 7/12/2019

**Program Description**

This program is written in the Go, which is a open source modern language. In this programming assignment you will create a Go program that reads expressions from standard input (i.e., the console), evaluates them, and prints the result. These expressions support addition, subtraction, multiplication, division, and exponentiation as well as parentheses.

------------------------------------------------------------------

**Sources**

Since the 1st part of the Go code was given to us. Tutorials given in class were my biggest source here. I have learnt how to execute on the paper and learnt from Google how to apply Go program. Also Learn from professor that how to download and create src, bin and pkg folder. Then need to set the environment and path of the code. Also learnt how to install golint. Stack.go and calculator.go file were given. I have saved stack.go in the stack folder and calculator-hint.go inside the calculator-hint folder. Both folders are inside src folder. Then I can run go file from command prompt and getting output. This output I got from calculator-hint.go file. But I have modify this file and also stack.go file to execute my program. I got help from Google a lot to execute this program. But overall it was good learning a new language.

This is the example output of the given code.

10 + (20 \* 30)^2

360010

2 ^ 3 ^ 2

512

Here is the given code by professor:

package main

import "bufio"

import "fmt"

import "os"

import "stack"

// Global operator and operand stacks

var operandStack stack.Stack

var operatorStack stack.Stack

// Returns x ^ y. This is a brute force integer power routine using successive

// multiplication. (There are more efficient ways to do this.)

func intPower(x int, y int) (pow int) {

pow = 1

for i := 0 ; i < y ; i++ {

pow \*= x

}

return

}

// Returns true if the character is a digit.

func isDigit(c byte) bool {

return '0' <= c && c <= '9'

}

// Returns the precedence of the operator.

func precedence(op byte) (prec int) {

switch op {

case '+', '-': prec = 0

case '\*', '/': prec = 1

case '^': prec = 2

default: panic("unknown operator")

}

return

}

// Apply the top operator on the operator stack to the top two operands on the

// operand stand and push the result onto the operand stack.

func apply() {

// Pop the operator off the operator stack

op, err := operatorStack.Pop()

if err != nil {

panic("operator stack underflow")

}

// Pop the right operand off the operand stack

right, err := operandStack.Pop()

if err != nil {

panic("operand stack underflow")

}

// Pop the left operand off the operand stack

left, err := operandStack.Pop()

if err != nil {

panic("operand stack underflow")

}

// Apply the operator to the left and right operands and push the result

// onto the operand stack

switch op.(byte) {

case '+': operandStack.Push(left.(int) + right.(int))

case '-': operandStack.Push(left.(int) - right.(int))

case '\*': operandStack.Push(left.(int) \* right.(int))

case '/': operandStack.Push(left.(int) / right.(int))

case '^': operandStack.Push(intPower(left.(int), right.(int)))

default: panic("unknown operator")

}

return

}

// Evaluate an expression and print the result.

func evaluate(expr string) {

defer func() {

if r := recover(); r != nil {

fmt.Println("illegal expression:", r)

}

}()

// Process the expression character by character left to right

operandExpected := true

i := 0

for i < len(expr) {

switch expr[i] {

// Digit: Extract the operand and push it on the operand stack

case '0', '1', '2', '3', '4', '5', '6', '7', '8', '9':

if !operandExpected {

panic("operator expected but operand found")

}

v := 0

for i < len(expr) && isDigit(expr[i]) {

v = 10\*v+int(expr[i]-'0')

i += 1

}

operandStack.Push(v)

operandExpected = false

// Operator: Apply pending operators of greater or equal precedence

// then push the operator on the operator stack

case '+', '-', '\*', '/', '^':

if operandExpected {

panic("operand expected but operator found")

}

for !operatorStack.IsEmpty() {

op, \_ := operatorStack.Top()

if precedence(op.(byte)) >= precedence(expr[i]) {

apply()

} else {

break

}

}

operatorStack.Push(expr[i])

i += 1

operandExpected = true

case '(':

fmt.Printf("%q is an open parenthesis\n", expr[i])

i += 1

case ')':

fmt.Printf("%q is a close parenthesis\n", expr[i])

i += 1

case ' ':

i += 1

default:

panic(fmt.Sprintf("%q is an illegal character", expr[i]))

}

}

// Apply any remaining operators

for !operatorStack.IsEmpty() {

apply()

}

// The result is the one operator remaining on the stack.

result, \_ := operandStack.Pop()

if !operandStack.IsEmpty() {

panic("too many operands")

}

fmt.Printf("%v\n", result)

}

// Main routine to read expressions from standard input, calculate their values,

// and print the result. (Use an end of file, control-Z, to exit.)

func main() {

// Make a scanner to read lines from standard input

scanner := bufio.NewScanner(os.Stdin)

// Process each of the lines from standard input

for scanner.Scan() {

// Initialize the operator and operand stacks

operandStack = stack.New()

operatorStack = stack.New()

// Get the current line of text.

line := scanner.Text()

// fmt.Println(line)

// Evaluate the expression and print the result

evaluate(line)

}

}

------------------------------------------------------------------

**Specifics**

In this assignment I have modified the program to get the result of float value. I have also little changed on stack.go file. So I am including my stack.go file with this assignment submission. This program is running properly and giving correct answer with decimal value.

**This is the output of the modified version of the file**

**Following are sample result:**

C:\Users\KKD>go run calculator

(2.456\*23)+2

Final result value is of type: float64.

58.487999999999985

C:\Users\KKD>go run calculator

2.56 + 3.456

Both types are of type float64

Final result value is of type: float64.

6.016

C:\Users\KKD>go run calculator

( 2 \_2.456)

Invalid Character '\_'. Please try again.

panic: illegal character

goroutine 1 [running]:

main.main()

C:/gocode/src/calculator/calculator.go:240 +0xca3

exit status 2

C:\Users\KKD>go run calculator

(2.45 \* 2)

Final result value is of type: float64.

4.8999999999999995

------------------------------------------------------------------------------------

**Here is the modified code :**

var operatorStack = stack.New()

var operandStack = stack.New()

//have added exponential and float value function in my code

func precedence(op byte) uint8 {

switch op {

case '(', ')': return 0

case '+', '-': return 1

case '\*', '/': return 2

case '^': return 3

default: panic("illegal operator")

}

}

// Returns x ^ y. This is a brute force integer power routine using successive

// multiplication. (There are more efficient ways to do this.)

func intPower(x int, y int) (pow int) {

pow = 1

for i := 0 ; i < y ; i++ {

pow \*= x

}

return

}

func floatPower(x float64, y int) (pow float64) {

pow = 1

for i := 0 ; i < y ; i++ {

pow \*= x

}

return

}

func apply() {

op := operatorStack.Pop().(byte)

// Create two interfaces

var right interface{}

var left interface{}

// Create two values to hold the Type of a Float64 and Int

floatT := reflect.TypeOf(0.0)

intT := reflect.TypeOf(0)

if !operandStack.IsEmpty() {

right = operandStack.Pop()

}else {

panic("Error: Operand stack is empty")

}

if !operandStack.IsEmpty() {

left = operandStack.Pop()

}else {

panic("Error: Operand stack is empty")

}

// If types are the same

if(reflect.TypeOf(right) == reflect.TypeOf(left)){

// Print which type they are

fmt.Println("Both types are of type " + reflect.TypeOf(right).String())

// Compare to Float64 type

if(reflect.TypeOf(right) == floatT){

// Push result onto stack as floating point number

switch op {

case '+': operandStack.Push(left.(float64) + right.(float64))

case '-': operandStack.Push(left.(float64) - right.(float64))

case '\*': operandStack.Push(left.(float64) \* right.(float64))

case '/': operandStack.Push(left.(float64) / right.(float64))

case '^': operandStack.Push(floatPower(left.(float64), right.(int)))

default: panic("illegal operator")

}

}else if (reflect.TypeOf(right) == intT) {

// Push result onto stack as integer number

switch op {

case '+': operandStack.Push(left.(int) + right.(int))

case '-': operandStack.Push(left.(int) - right.(int))

case '\*':operandStack.Push(left.(int) \* right.(int))

case '^': operandStack.Push(intPower(left.(int), right.(int)))

case '/':

// Check if division will result in a remainder. If so,

// We need to use floating point.

if((left.(int) % right.(int)) == 0){

operandStack.Push(left.(int) / right.(int))

}else {

fmt.Println("Integer division will result in non-zero remainder,\nSwitching to floating point.")

// Get value of operands

rt := reflect.ValueOf(right)

lt := reflect.ValueOf(left)

// Get data from Value

rt = reflect.Indirect(rt)

lt = reflect.Indirect(lt)

// Convert data to Float64 type

rt = rt.Convert(floatT)

lt = lt.Convert(floatT)

// Push result onto stack as Float64

operandStack.Push(lt.Float() / rt.Float())

}

default: panic("illegal operator")

}

}else{

panic("Error: Invalid operand type.")

}

}else{

// Get value of each interface

rt := reflect.ValueOf(right)

lt := reflect.ValueOf(left)

// Get the data held by the Value

rt = reflect.Indirect(rt)

lt = reflect.Indirect(lt)

// Convert the actual data to floating point type

rt = rt.Convert(floatT)

lt = lt.Convert(floatT)

// Print out the types of each operand

//fmt.Println("Left operand is of type: " + reflect.TypeOf(left).String() + ".")

//fmt.Println("Right operand is of type: " + reflect.TypeOf(right).String() + ".")

switch op {

case '+': operandStack.Push(lt.Float() + rt.Float())

case '-': operandStack.Push(lt.Float() - rt.Float())

case '\*': operandStack.Push(lt.Float() \* rt.Float())

case '/': operandStack.Push(lt.Float() / rt.Float())

//case '^': operandStack.Push(intPower(left.(int), right.(int)))

case '^': operandStack.Push(floatPower(left.(float64), right.(int)))

default: panic("illegal operator")

}

}

}

-----------------------------------------------------------------------

**This is my modified stack.go code :**

package stack

//import "errors"

type Stack []interface{}

func New() Stack {

return make(Stack, 0)

}

func (this \*Stack) IsEmpty() bool {

return len(\*this) == 0

}

func (this \*Stack) Push(v interface{}) error {

\*this = append(\*this, v)

return nil

}

func (this \*Stack) Pop() (interface{}) {

if len(\*this) == 0 {

return nil

}

v := (\*this)[len(\*this)-1]

\*this = (\*this)[:len(\*this)-1]

return v

}

func (this \*Stack) Top() (interface{}) {

if len(\*this) == 0 {

return nil

}

return (\*this)[len(\*this)-1]

}

------------------------------------------------------------------

**Golint**

First I have installed git, then install git in my computer.I have installed Golint in my system through command prompt and this is another new thing I have learnt from this assignment. I came to know from Google how to run golint in the program and how to get the error messages. After getting the errors, I have fixed few errors. But I couldn’t fix all of them. I got following messages.

C:\Users\KKD>golint calculator

I did not get any error message here because I have modified all error. So after that I got empty message.

This is my modified stack.go, which one I have run in with golint and got following messages.

C:\Users\KKD>golint stack

C:\gocode\src\stack\stack.go:5:6: exported type Stack should have comment or be unexported

C:\gocode\src\stack\stack.go:7:1: exported function New should have comment or be unexported

C:\gocode\src\stack\stack.go:11:1: exported method Stack.IsEmpty should have comment or be unexported

C:\gocode\src\stack\stack.go:11:1: receiver name should be a reflection of its identity; don't use generic names such as "this" or "self"

C:\gocode\src\stack\stack.go:15:1: exported method Stack.Push should have comment or be unexported

C:\gocode\src\stack\stack.go:15:1: receiver name should be a reflection of its identity; don't use generic names such as "this" or "self"

C:\gocode\src\stack\stack.go:20:1: exported method Stack.Pop should have comment or be unexported

C:\gocode\src\stack\stack.go:20:1: receiver name should be a reflection of its identity; don't use generic names such as "this" or "self"

C:\gocode\src\stack\stack.go:29:1: exported method Stack.Top should have comment or be unexported

C:\gocode\src\stack\stack.go:29:1: receiver name should be a reflection of its identity; don't use generic names such as "this" or "self"

------------------------------------------------------------------

**Discussion and Conclusion**

I wrote this code in Go and this is a new programming language for me. As we had very short time to learn a new language. I tried my best to execute the program. Professor explained in the class a lot and the step by step algorithm. I understand the algorithm part. I like to learn more this modern language. I can understand the go code after reading somebody else’s code. I am excited to see what I can do in the future with this innovative, promising language. Google made this language to use extensively.