How to Create a Programming Language

LECTURE 2: SAMPLING OF PROGRAMMING LANGUAGES / PLAYING WITH A BASIC LANGUAGE

1

LECTURE OUTLINE

- REVIEW OF LANGUAGE TYPES
- EXAMPLES OF IMPERATIVE LANGUAGES
- EXAMPLES OF FUNCTIONAL LANGUAGES
- EXAMPLES OF LOGIC LANGUAGES
- A SIMPLE EXPRESSION LANGUAGE

2

PROGRAMMING LANGUAGE CLASSIFICATION

- **IMPERATIVE** – FOCUS: **HOW** THE COMPUTER SHOULD DO A TASK
- **DECLARATIVE** – FOCUS: **WHAT** THE COMPUTER SHOULD DO

3

IMPERATIVE PROGRAMMING LANGUAGES

- **VON NEUMANN** – BASED ON MODIFICATION OF VARIABLES/STATE VIA SIDE-EFFECTS
  - C
  - FORTRAN
  - ADA
  - PASCAL
  - ETC.
- **OBJECT-ORIENTED** – BASED ON SEPARATION OF DATA AND CODE INTO SEMI-INDEPENDENT 'OBJECTS'
  - SMALLTALK
  - C++
  - JAVA
  - ETC.

4

DECLARATIVE PROGRAMMING LANGUAGES

- **FUNCTIONAL** – BASED ON (POSSIBLY RECURSIVE) FUNCTIONS
  - LISP
  - ML
  - HASKELL
- **DATAFLOW** – BASED ON A 'FLOW' OF 'TOKENS' TO PROCESSING 'NODES'
  - ID
  - VAL
- **LOGIC/CONSTRAINT-BASED** – BASED ON FINDING VALUES THAT FIT A CRITERIA (GOAL-DIRECTED SEARCH) PRINCIPLES INCLUDE PREDICATE LOGIC.
  - PROLOG

5

IMPERITIVE – VON NEUMAN

C

```
#include <stdio.h>
int main(int argc, char **argv) {
   int n1, n2, i, gcd;
   printf("Enter two integers: ");
   scanf("%d %d", &n1, &n2);
   for(i=1; i <= n1 && i <= n2; ++i) {
      // Checks if i is factor of both integers
      if ( n1%i==0 && n2%i==0)
         gcd = i;
    }
   printf("gcd of %d and %d is %d", n1, n2, gcd);
   return 0;
}
```


6

IMPERITIVE – VON NEUMAN

Python

```
n1 = int(input("Integer 1: "))
n2 = int(input("Integer 2: "))
if n1 > n2:
   (w1, w2) = (n1, n2)
else:
   (w1, w2) = (n2, n1)
while True:
   if w1%w2 != 0:
      w1 = w1 - w2
      if w1 < w2:
         (w1, w2) = (w2, w1)
   else:
      gcd = w2
      break
print("gcd of " + str(n1) + " and " + str(n2) + " is " + str(gcd))
```


7

IMPERITIVE – OBJECT ORIENTED

Java

```
import java.util.Scanner;
public class gcd {
   public static void  main(String[] args) {
      int n1, n2, i, gcd = 0;
      Scanner cons = new Scanner(System.in);
      System.out.print("Enter two integers: ");
      n1 = cons.nextInt();
      n2 = cons.nextInt();
      for(i=1; i <= n1 && i <= n2; ++i) {
        if (n1%i==0 && n2%i==0)
           gcd = i;
      }
      System.out.println("gcd of " + n1 + " and " + n2 + " is " + gcd);
   }
}
```


8

FUNCTIONAL

Lisp

```
(defun mygcd (a b)
  (if (= b 0) a (mygcd b (mod a b)))
  )

(write "Enter 2 integers: ")
(setq n1 (read))
(setq n2 (read))
(write (mygcd n1 n2))
```


9

LOGIC

Prolog

```
gcd(X,Y):-X=Y,write('GCD of two numbers is '),write(X);
X=0,write('GCD of two numbers is '),write(Y);
Y=0,write('GCD of two numbers is '),write(X);
Y>X,Y1 is Y-X,gcd(X,Y1);
X>Y,Y1 is X-Y,gcd(Y1,Y).
```


10

OH, AND THEN THERE IS…

APL – 'A Programming Language'

```
      ∇DET[□]∇
     ∇ Z←DET A;B;P;I
[1]   I←□IO
[2]   Z←1
[3]  L:P←(|A[;I])⍳⌈/|A[;I]
[4]   →(P=I)/LL
[5]   A[I,P;]←A[P,I;]
[6]   Z←-Z
[7]  LL:Z←Z×B←A[I;I]
[8]   →(0 1 ∨.=Z,1↑⍴A)/0
[9]   A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]  →L
[11] ⍝EVALUATES A DETERMINANT
     ∇
```

ANOTHER VON NEUMAN LANGUAGE BUT WITH A BIZARRE CHARACTER SET AND NON-STANDARD PRECEDENCE/ASSOCIATIVITY

<= THIS IS NOT A GCD FUNCTION.

11

LAB

- 30 MINS, IN TEAMS OF 2 STUDENTS
- RESEARCH (ONLINE) TWO LANGUAGES FROM **DIFFERENT** CLASSIFICATIONS
- NOTE THE DIFFERENCES
- JOT SOME IDEAS DOWN ABOUT HOW THE CLASS OF LANGUAGE HELPS ITS EFFECTIVENESS FOR SPECIFIC PROBLEM DOMAINS
  - FINANCIAL APPLICATIONS
  - EMBEDED AVIONICS
  - FORMAL METHODS [MATHEMATICAL PROOFS OF CORRECTNESS]
  - COMPILERS FOR TRANSLATING LANGUAGES TO MACHINE CODE
  - ETC.

12

A SIMPLE EXPRESSION LANGUAGE

Standard Infix Notation – Operator comes between 'operands'
5 + 10 => 15

- More complex expressions raise questions about 'precedence' and 'associativity'

Polish notation – Operator comes before 'operands'
+ 5 10
=> 15

Reverse Polish Notation – Operator comes after 'operands'
5 10 +
=> 15

- MORE COMPLEX EXPRESSION:
5 10 + 11 * ➔ ((5+10) * 11)
5 10 11 * + ➔ 5 + (10 * 11)

13

LETS CALL THE LANGUAGE SRP

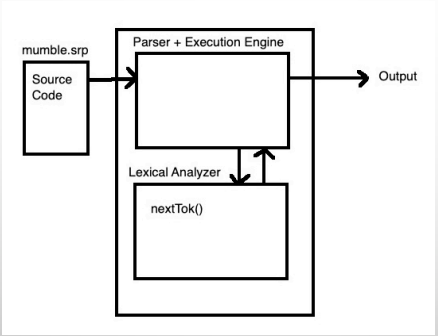
Let's look at writing a quick interpreter…
We will NOT generate machine code
We will read the source and perform the requested operations directly.

1st pass: No variables. All constants. Numbers are a single digit only

Example:
5 7 + 9 *

14

BASIC STRUCTURE OF INTERPRETER



15

SRP SOURCE [LEXICAL ANALYZER]

```
global loc
loc = 0

# Lexical analyzer - Tokenizes input
def nextTok(theline):
   # Check for number
   global loc
   if loc >= len(theline):
      return ('e', 0)

   if theline[loc] >= '0' and theline[loc] <= '9':
      # Handle integer
      retval = ('v', int(theline[loc]))
      loc = loc + 1
   elif theline[loc] in ['+', '-', '*', '/', '=']:
        retval = ('o', theline[loc])
   else:
      retval = ('!', '!')
      loc = loc + 1
      print('Invalid character')
   loc = loc + 1
   while loc < len(theline) and theline[loc] in [' ', '\n']:
      loc = loc + 1

   return retval
```


16

SRP SOURCE [UTIL ROUTINES]

```
global stack
global sp

stack = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
sp = -1
def push(val):
    global stack
    global sp
    if sp > len(stack):
        print("Error stack full!")
    else:
        sp = sp + 1
        stack[sp] = val
def pop():
    global stack
    global sp
    if sp >= 0:
        retval = stack[sp]
        sp = sp - 1
    else:
        retval = ''
    return retval
```

```
def do_op(o1, o2, op):
    if op == '+':
        res = int(o1) + int(o2)
    elif op == '-':
        res = int(o1) - int(o2)
    elif op == '*':
        res = int(o1) * int(o2)
    else:
        res = int(o1) / int (o2)
    return res
```


17

SRP SOURCE [PARSER]

```
progfile = open('test.srp', 'r')

nextline = progfile.readline()

print ("Next line: " + nextline)

tok = nextTok(nextline)
while tok[0] != '!' and tok[0] != 'e':
    if tok[0] == 'v':
        push(tok[1])
    else:
        o1 = pop()
        o2 = pop()
        res = do_op(o1, o2, tok[1])
        push(res)

    tok = nextTok(nextline)

print("Res: " + str(pop()))
```


18

QUESTIONS

19