The UNIVERSAL machine

ALAN TURING (CHARLES BABBAGE)

ADA LOVELACE

SOFTWARE ⟶ M/C CODE

CODE GENERATION

1 - REGISTER MACHINE

1 LOCUS OF COMPUTATION

\+ MEMORY

X: variable name → memory loc.

1,2,3,... : memory locations

LOAD X put the value in memory location X into the accumulator

STORE X → puts the value in the acc. into location named by X

STORE n

ADD X,## ADD n

MUL X,## MUL n

n: an integer naming a memory location

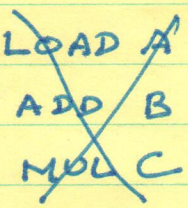
A*B + C	ACC ↓
LOAD A	A
MUL B	A*B
ADD C	A*B+C

A*B + C*D

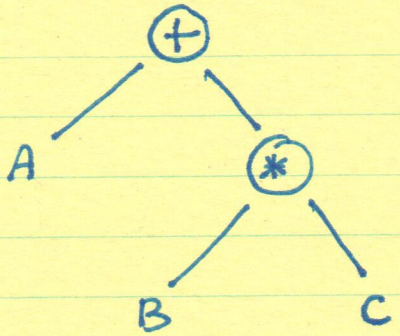
LOAD	A	A
MUL	B	A*B
STORE	1	A*B
LOAD	C	C
MUL	D	C*D
ADD	1	C*D + A*B

A + B*C

LOAD	A
STORE	1
LOAD	B
MUL	C
ADD	1



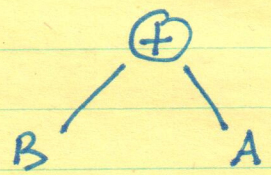
The code can only be generated from the expression tree.



CODEGEN is done by traversing this tree RECURSIVELY!

But the CODEGEN needs to keep some <u>contextual</u> information.

At the node A I should do
LOAD A
or should I?

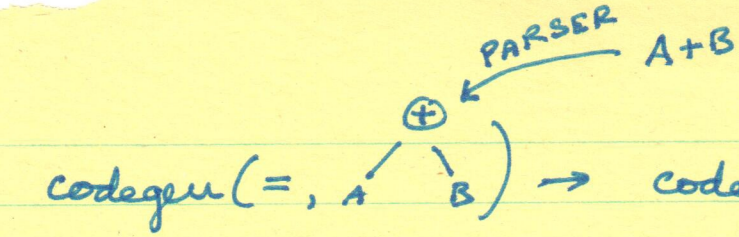


At node A you should do
ADD A

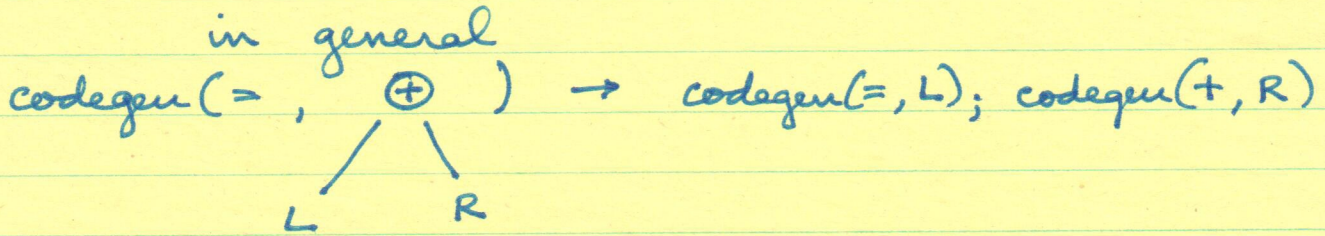
Our ~~codegen~~ code generator will have 2 arguments : (context tag, tree)

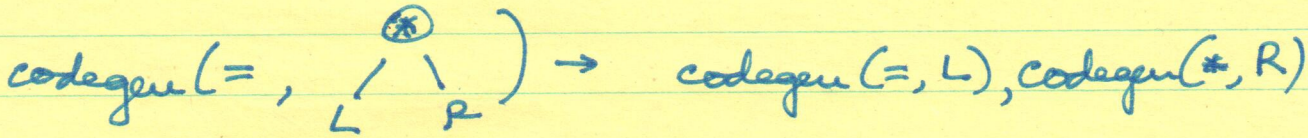
=	: starting fresh
*	: doing a MUL
+	: doing a Plus

codegen(=, (A)) → LOAD A
codegen(+, (A)) → ADD A
codegen(*, (A)) → MUL A

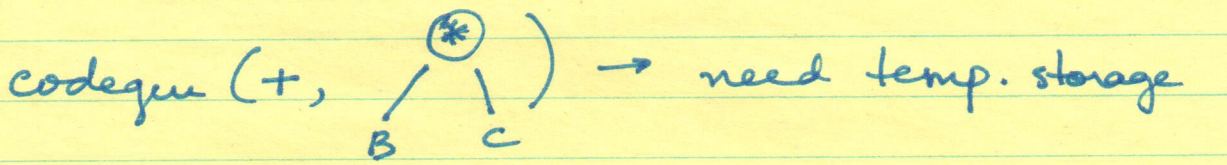


in general



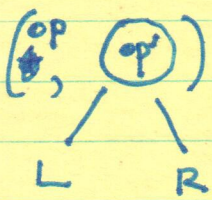


What if we have



keep a <u>global variable</u> tempstore
its value is the memory location <u>just</u> used.

Using temp. storage:

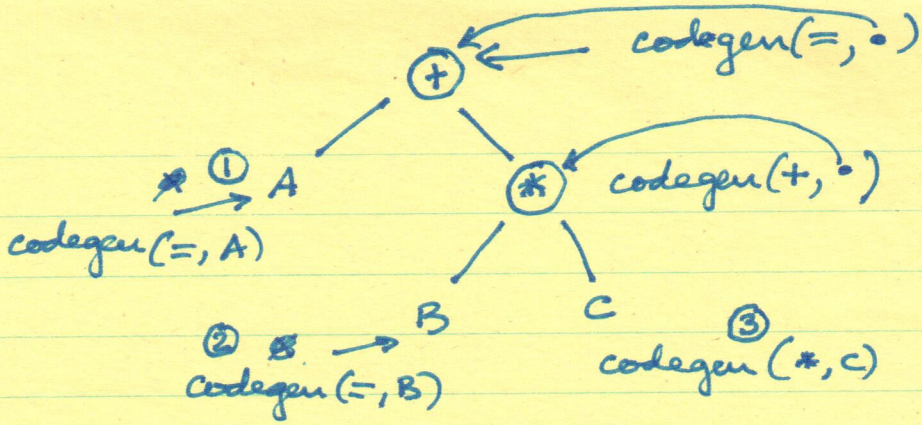



```
tempstore := tempstore+1;
print ("STORE", tempstore);
codegen(=, L);   } These are recursive calls
codegen(op', R); } they work by magic
if op = '+' then print ("ADD", tempstore)
 else if op = '*' then print ("MUL", tempstore)
  else "error"
tempstore := tempstore-1
```

TOP LEVEL CALL codegen(=, TREE)

A + B * C

LOAD A ✓
STORE 1 ✓
LOAD B ✓
MUL C ✓
ADD 1 ✓

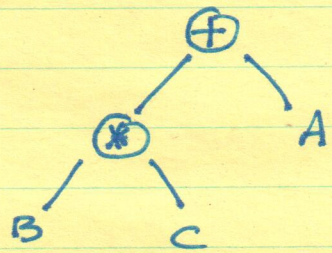


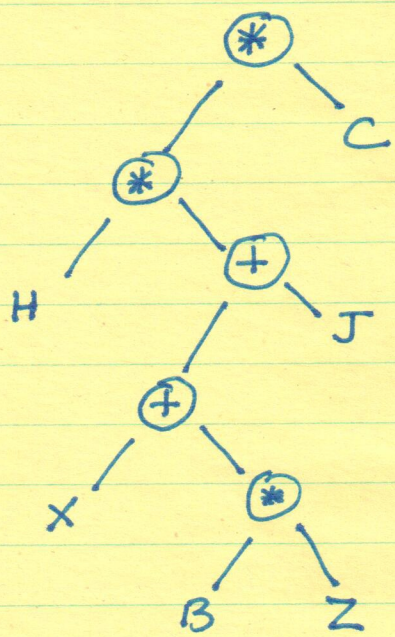
A + (B + (C + · · · ·)) · ·)

——— X ———

(B * C) + A

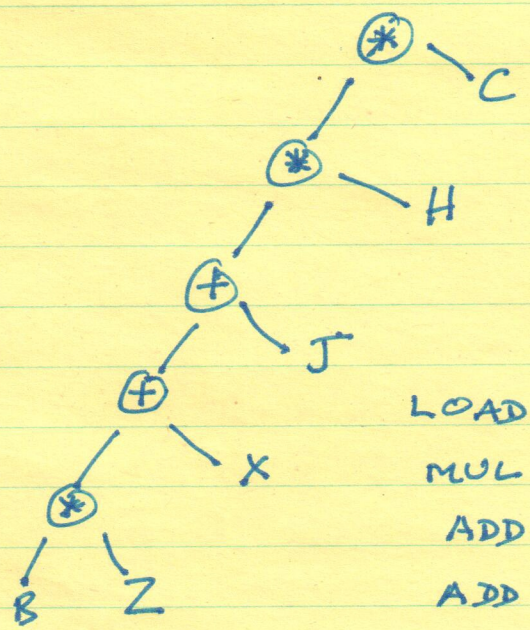
LOAD B
MUL C
ADD A





LOAD H
STORE 1
LOAD X
STORE 2
LOAD B
MUL Z
ADD 2
ADD J
MUL 1
MUL C

10 steps
2 extra memory cells



LOAD B
MUL Z
ADD X
ADD J
MUL H
MUL C

6 steps
no extra memory