316 10-06-09

<statement> → <assignment> | "{" <slist> "}"

<assignment> → <id> = <E> ;

<slist> → { <statement> }+

①
```
void statement()
{
  if (t is "{")
  {
    getToken();
    slist();
    if (t is "}") getToken();
    else print("Error: } expected ");
  } //end of block case
  else assignment();
}
```

②
```
void assignment()
{
  if (t is <id>)
  {
    getToken();
    if (t is "=")
    {
      getToken();
      E();
      if (t is ";") getToken();
      else print("Error: ; expected ");
    }
    else print("Error: = expected ");
  }
  else print("Error: <id> expected ");
}
```

③
```
void slist()
{
  statement();
  while (t is <id> || t is "{")
  {
    statement();
  }
}
```

Example of intermediate-code instruction emission from top-down parsers:

Arithmetic expressions <E> with +, −, *, /, unary−

Statements consisting of assignments and brackets

```
{ x = b * c / d;
  { y = A - B * C;
    z = B * y + A;
  }
}
```

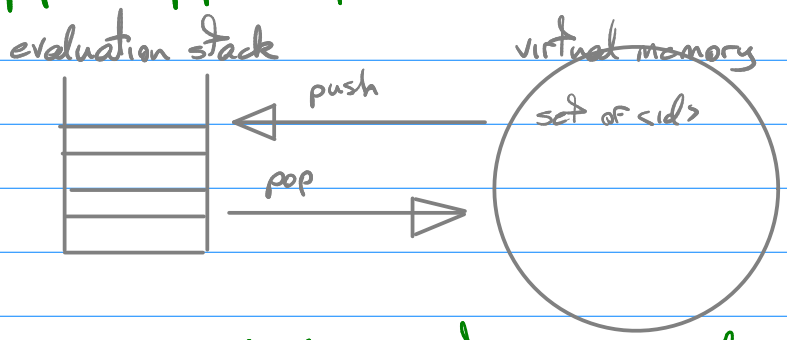
→ (top-down parser) → implicit parse trees in the form of tree of function calls → instruction stream

Use the following for a stack-based virtual machine:

push <id>: push the value of <id> onto the evaluation stack

push <const>: push the <const> value onto the evaluation stack

pop <id>: pop the top element of the evaluation stack and store it in <id>



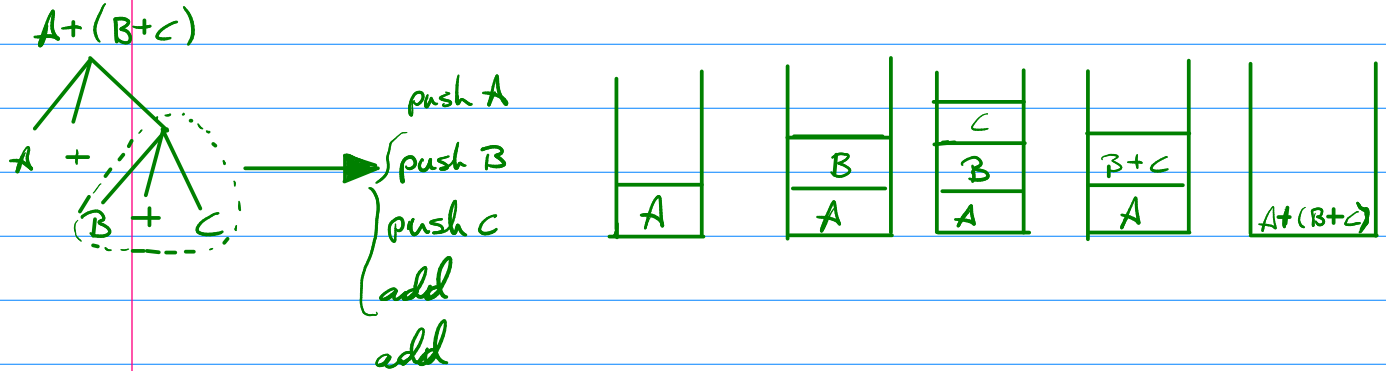
add: pop the top two elements of the evaluation stack, compute stack[top-1] + stack[top], push the value back onto the stack

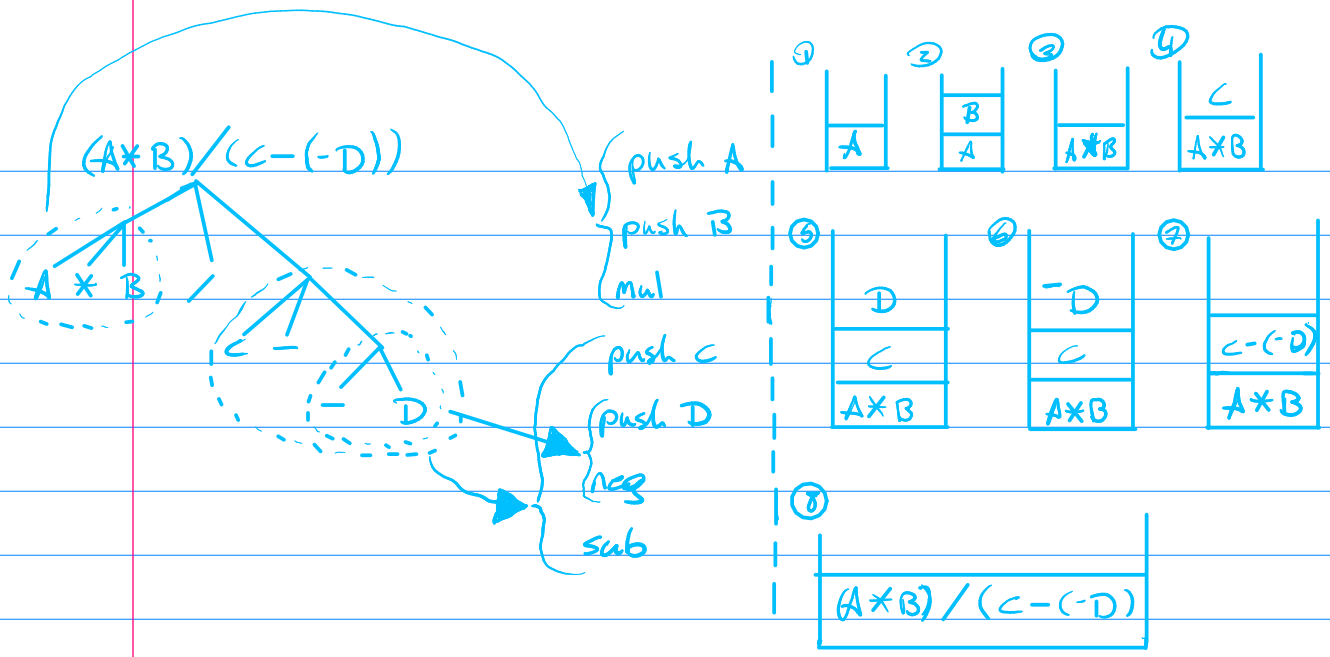
anonymously for sub(for -), mul(for *), div(for /). In fact, this action is done for any binary interactions

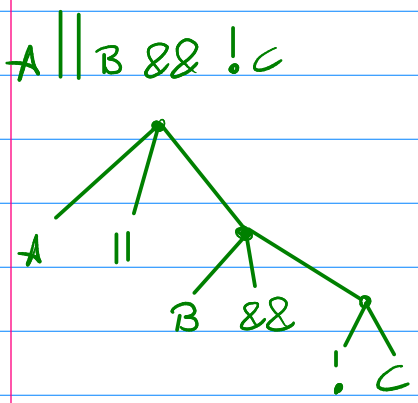
neg: for unary -, Pop the top element of the evaluation stack, compute -stack[top], push the value onto the stack. In fact, this action is done for any unary instruction

How to generate these instructions for <E>

- Do post-order traversal of the parse tree for <E>
  - visit argument subtrees, each in post order
  - visit the operator
- IF leaf node is labeled by t=<id>, <int>, <float> → emit "push t"
- " " " " " " binary +, -, *, / → emit add, sub, mul, div (respectively)
- " " " " " " unary - → emit neg




(A*B)/(C-(-D))
A * B
C -
- D
push A
push B
mul
push C
push D
neg
sub
1
A
2
B
A
3
A*B
4
C
A*B
5
D
C
A*B
6
-D
C
A*B
7
C-(-D)
A*B
8
(A*B)/(C-(-D))


A || B && !C
A
||
B
&&
!
C