Abstract Syntax Tree (AST) and Control Stack

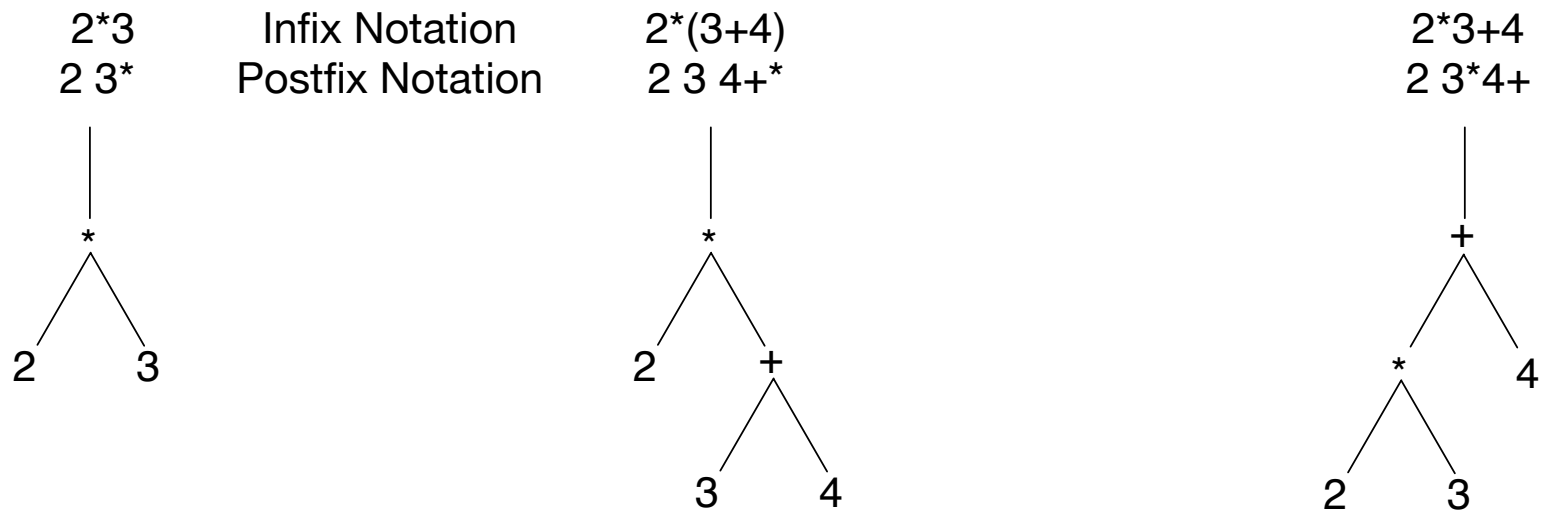
An **abstract syntax tree** (**AST**), or just **syntax tree**, is a *tree representation of the abstract syntactic structure of source code* written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is "abstract" in not representing every detail appearing in the real syntax.

© http://en.wikipedia.org/wiki/Abstract_syntax_tree , 11 Oct. 2013

A **Control Stack** is a *stack data structure* that stores information about the active subroutines of a computer program. This kind of stack is also known as an **execution stack**, **call stack**, **run-time stack**, or **machine stack**, and is often shortened to just "the stack".

© http://en.wikipedia.org/wiki/Call_stack , 11 January 2015

AST for Arithmetic Expressions + Traversal



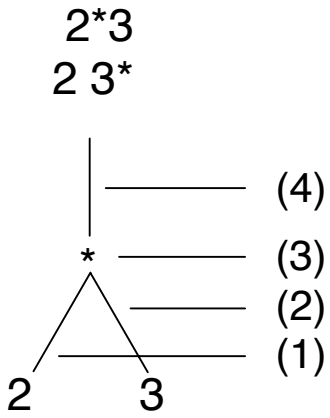
Each leaf represents an operand and non leaf an operator

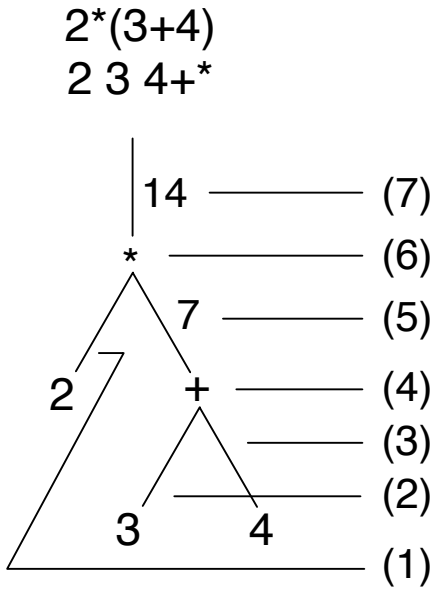
Postorder traversal:
1. Visit left subtree, in postorder
2. Visit right subtree, in postorder
3. Visit the root

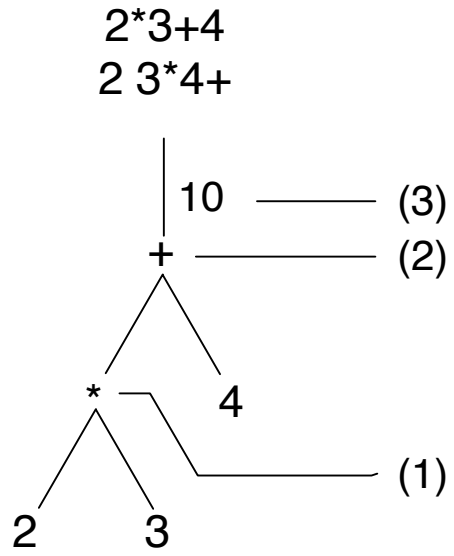
=> Corresponds to the postfix expression!

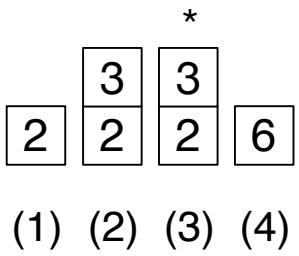
Postorder results easy to process:
- Process elements left to right
- Number? Push it on a stack
- Binary operator? Remove two top elements, apply operator to it, push result on stack

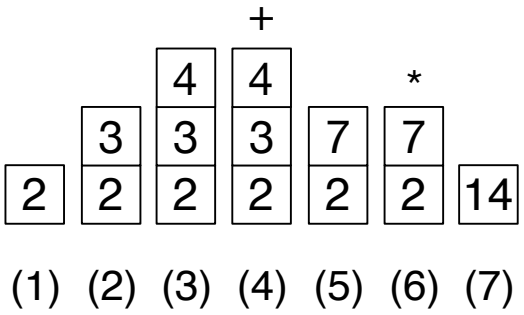
AST for Arithmetic Expressions (revisited) and the Associated Control Stacks for some significant steps)

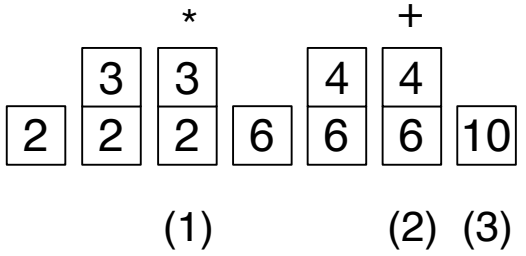








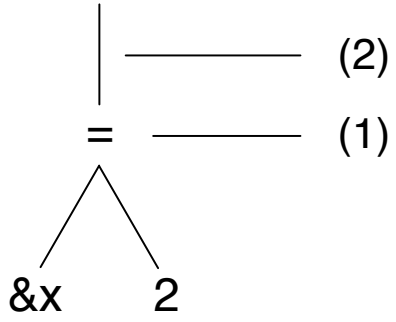




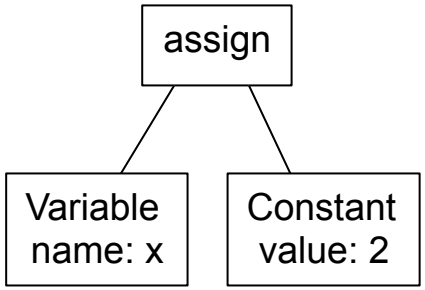
AST and CS for an assignment = and a statement ;

x = 2

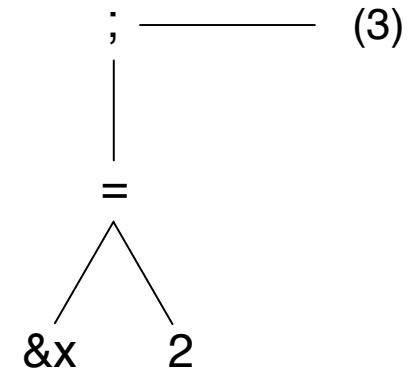
AST:



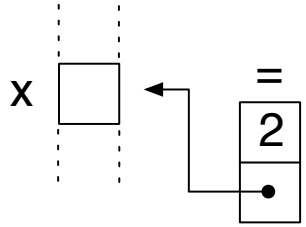
Simplified notation:



x = 2 ;

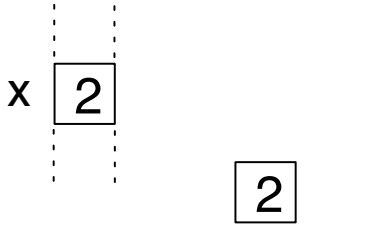


MM & CS:



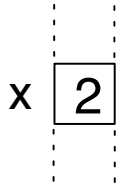
Main Memory Control Stack

(1)



Main Memory Control Stack

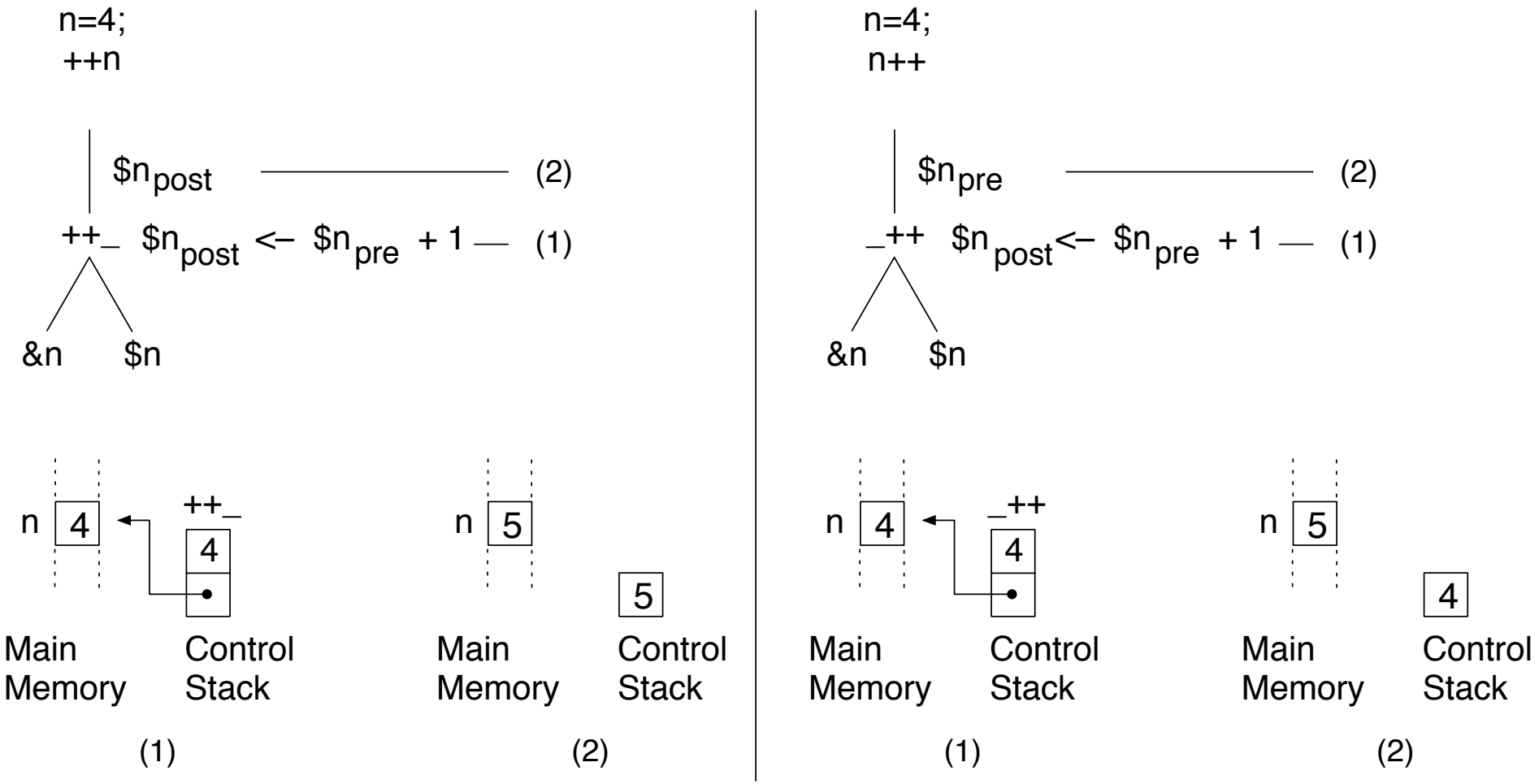
(2)



Main Memory Control Stack

(3)

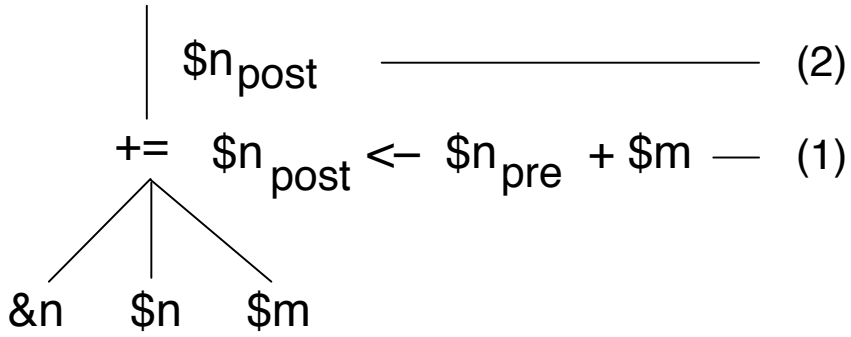
AST and CS for ++n and n++

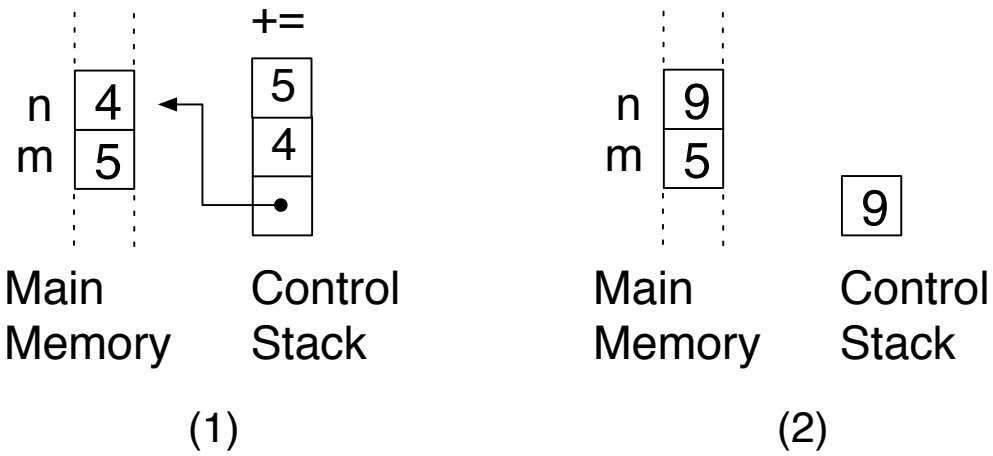


Notation: $n is the value of n

AST and CS for n += m

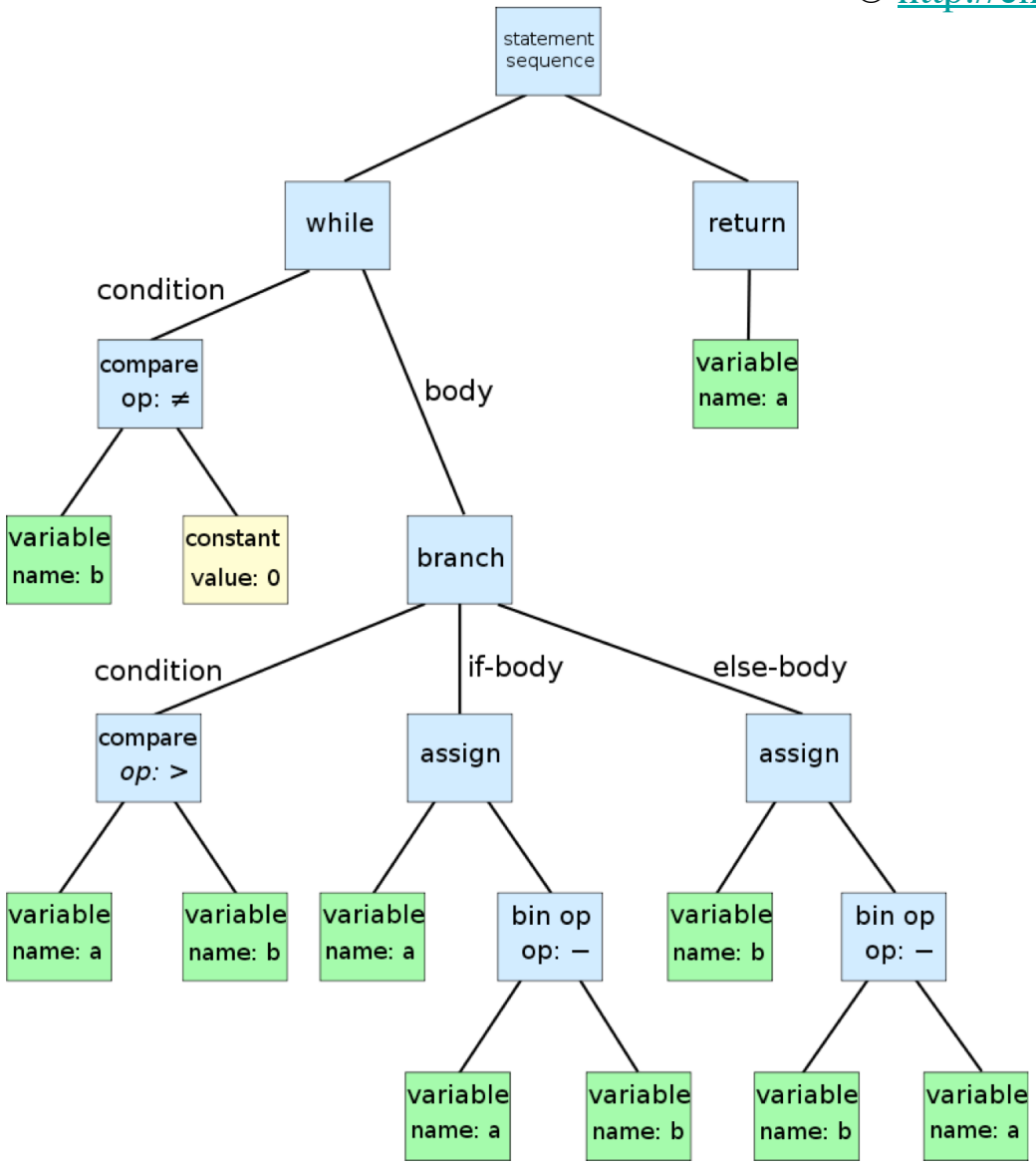
n=4; m=5;
n += m





AST for 'statement sequence' and 'control statements'

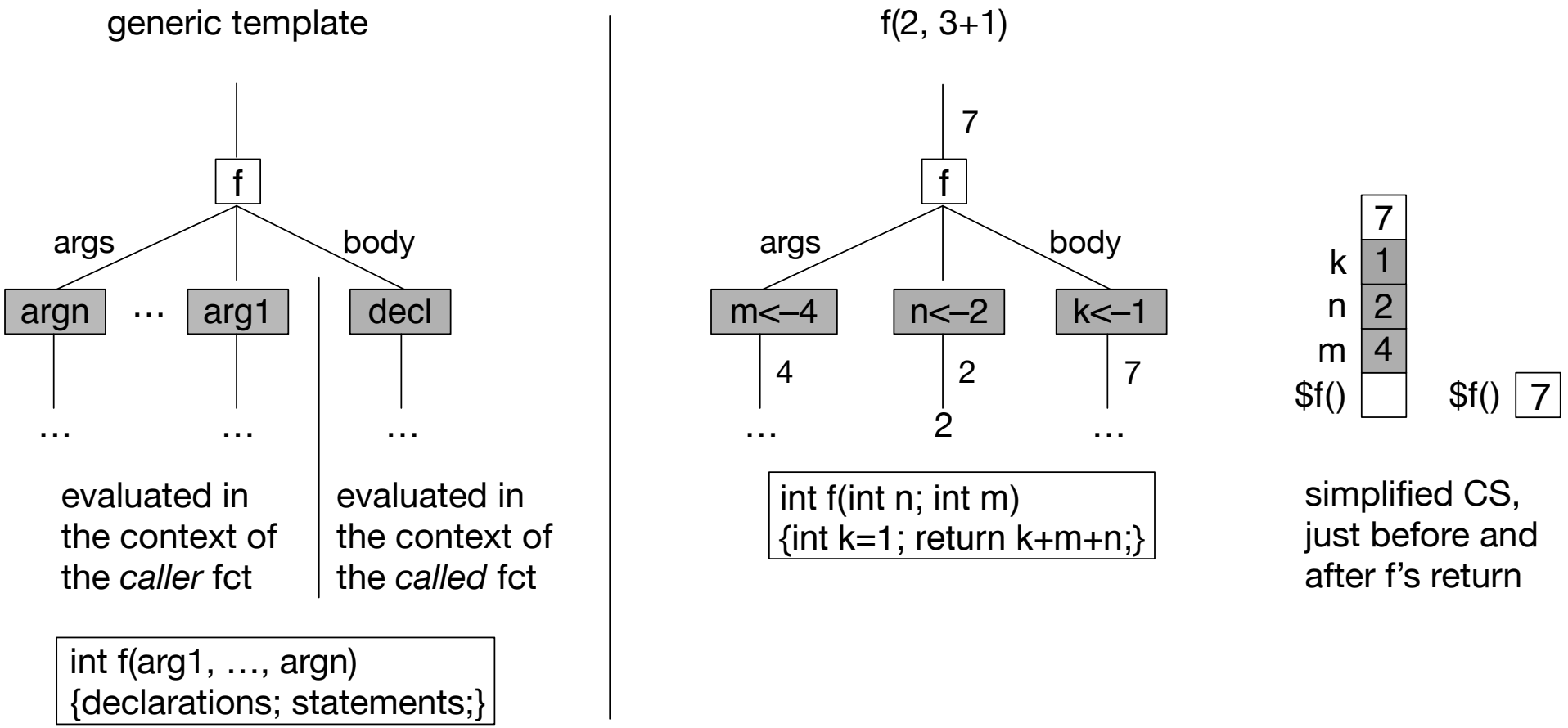
© http://en.wikipedia.org/wiki/Abstract_syntax_tree




```
while (b != 0)
{
    if (a > b) a = a -
b;
    else b = b - a;
}
return a;
```

a syntactic construct like an *if-condition-then-else* may be denoted by means of a single node with three branches

AST and Simplified CS for Function Calls



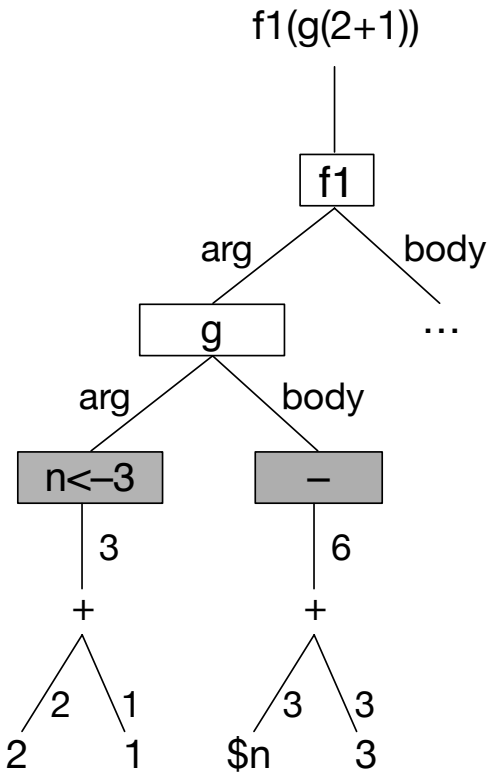
Def. The greyed areas are called f's **function frame**

Remark. f's function frame belongs to f's context

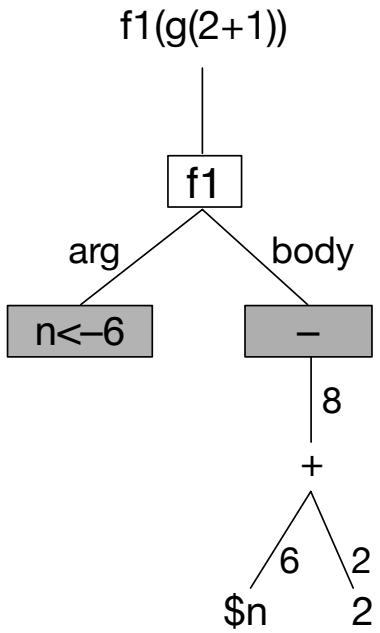
Remark 2. In the AST representation 2 important features of the context switch are encoded implicitly: 1) where to return and 2) where is the function frame of the caller function. In the Control Stack representation, these 2 informations will have to be encoded explicitly in each function frame.

AST for Function Calls with a Function as Argument

int f1(int n) {return n+2;}
int g(int n) {return n+3;}

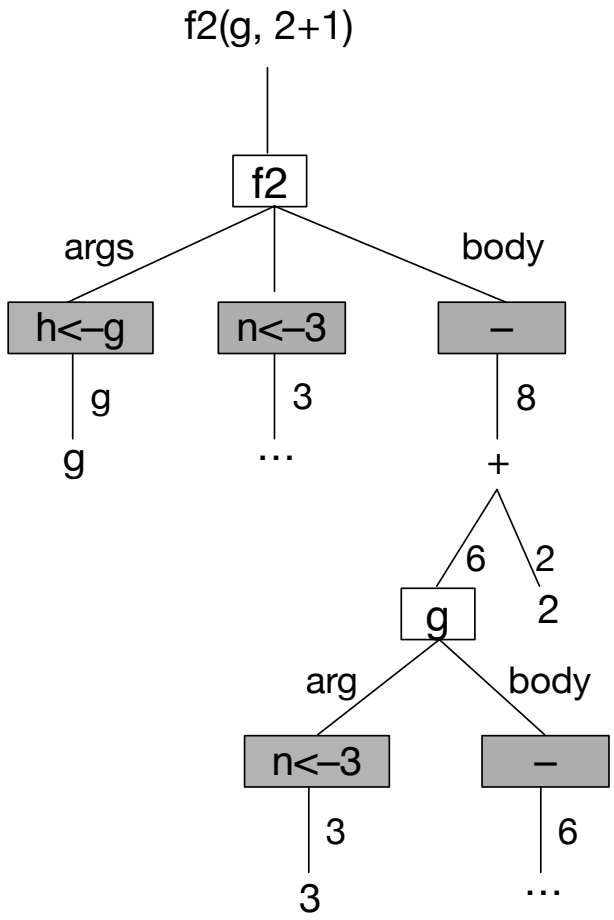


just before g's return



just before f's return

int f2(int h(int), int n)
{return h(n) + 2;}



just before f's return