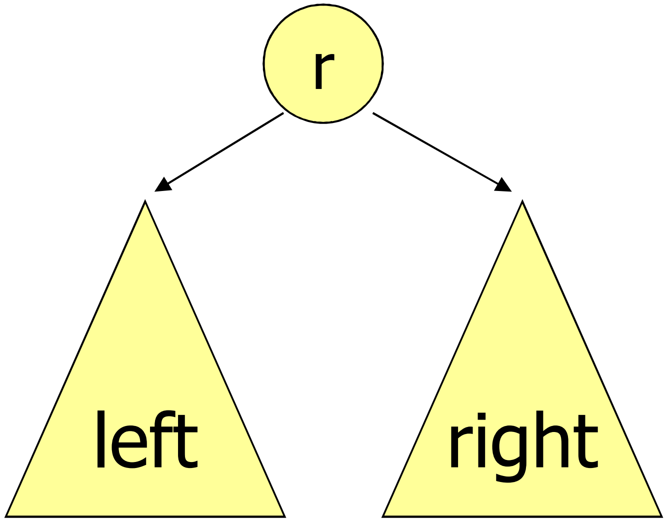
Trees

Paolo Camurati

Dip. Automatica e Informatica
Politecnico di Torino

Binary Trees

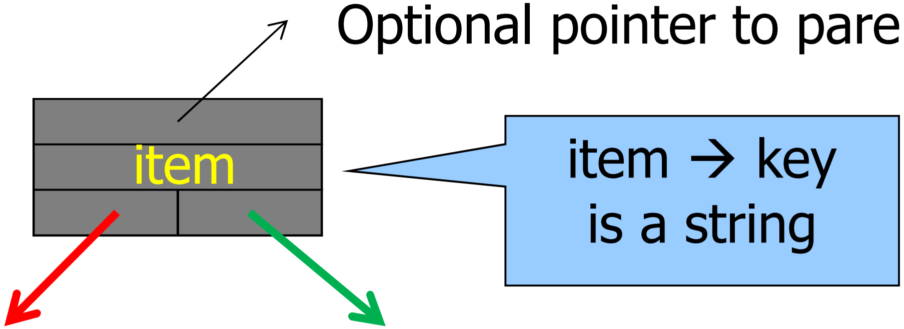
- Recursive definition
  - Empty set of nodes
  - Root, left subtree, right subtree



Binary Trees

- Node

Optional pointer to parent



item → key
is a string

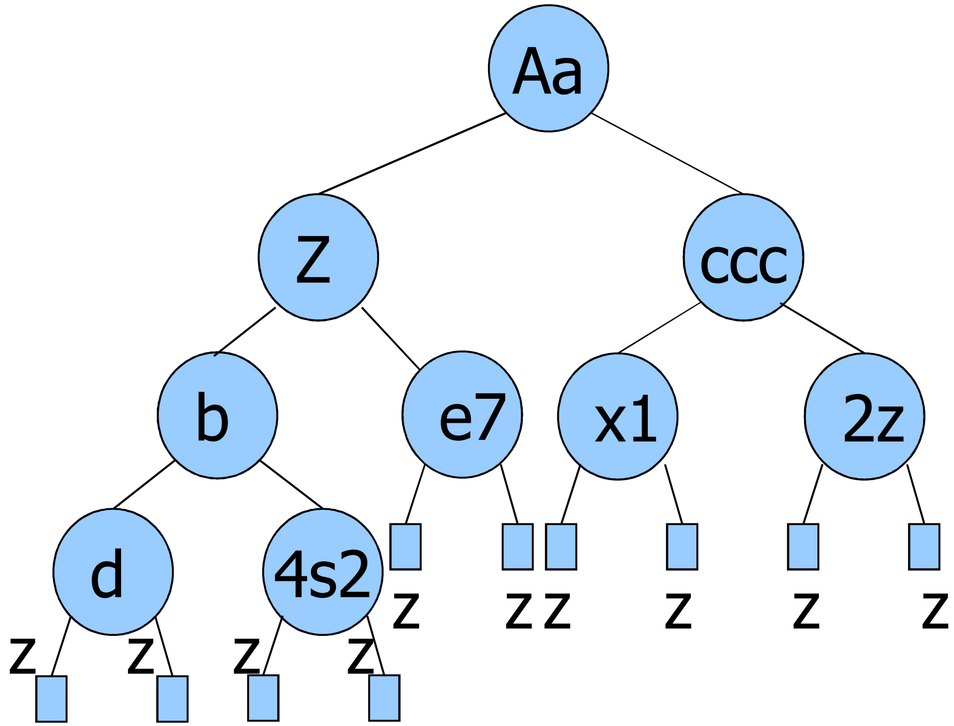
pointer to left child

ointer to right child

```
typedef struct node *link;
struct node {
   Item item;
   link l;
   link r;
};
```

Binary Trees

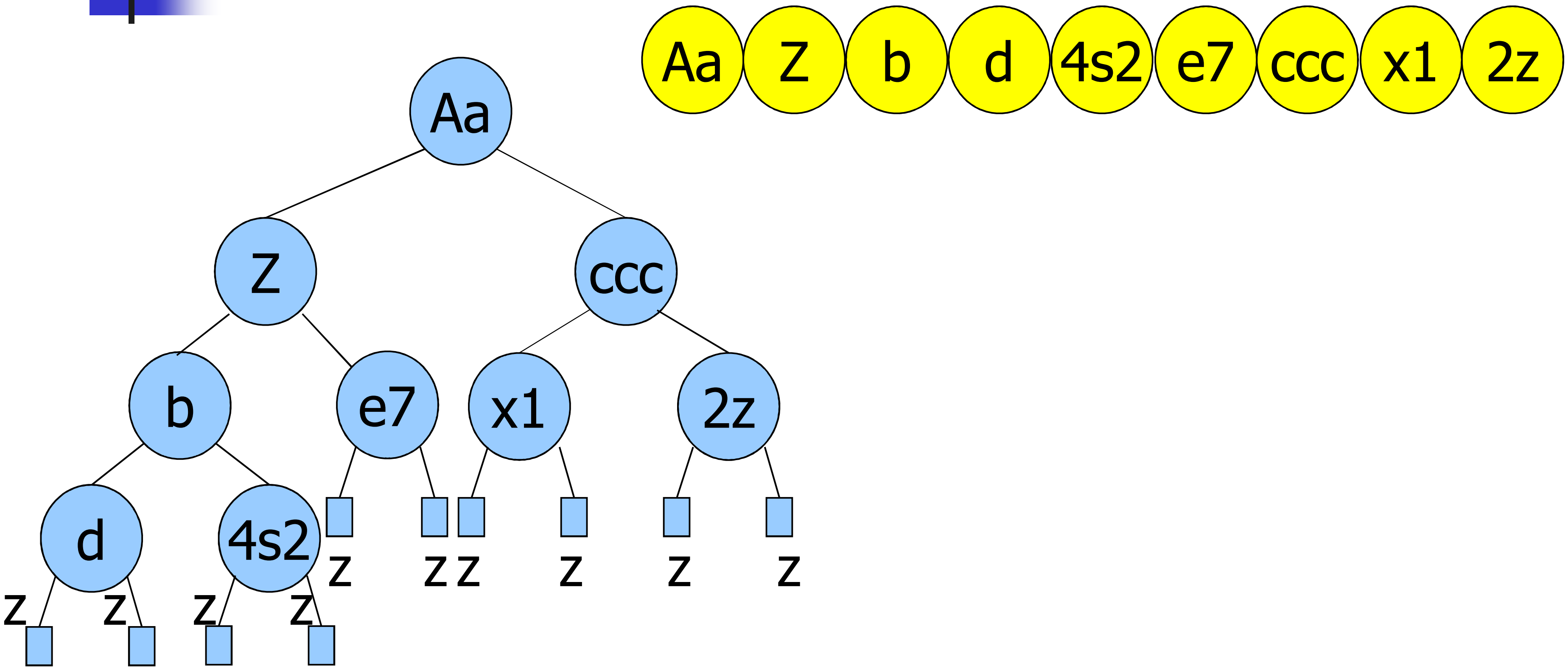
- Tree
  - Access through pointer to root
  - Dummy sentinel node z or NULL pointer



Visits

- A tree traversal or a tree visit lists the nodes according to a strategy
  - Pre-order
    - Root, Left child (l), Right child (r)
  - In-order
    - Left child (l), Root, Right child (r)
  - Post-order
    - Left child (l), Right child (r), Root

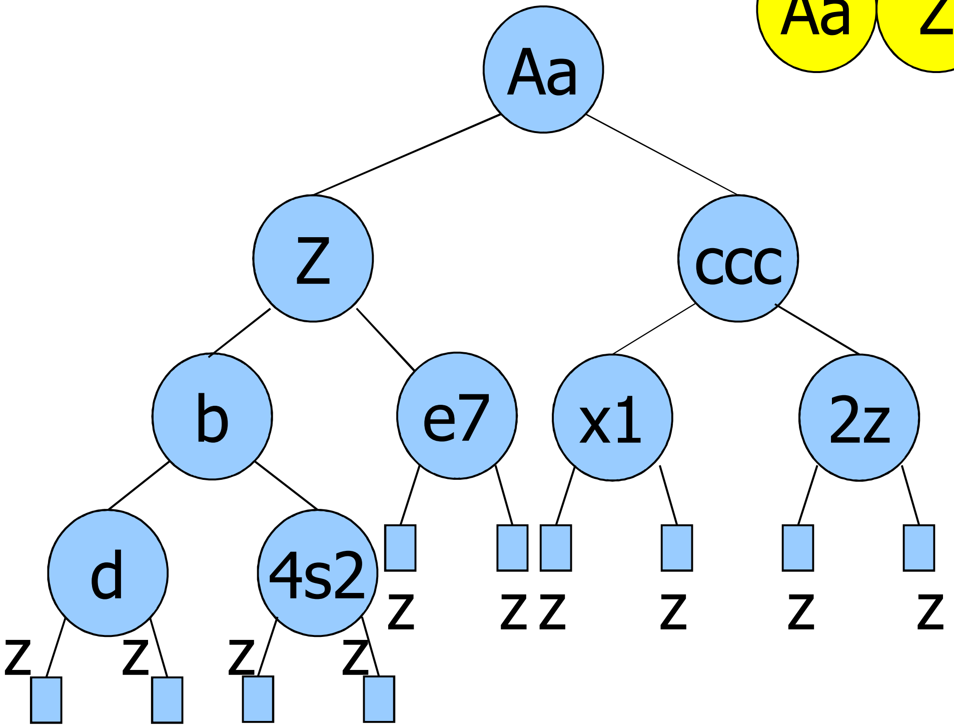
Pre-order



Aa Z b d 4s2 e7 ccc x1 2z

Pre-order

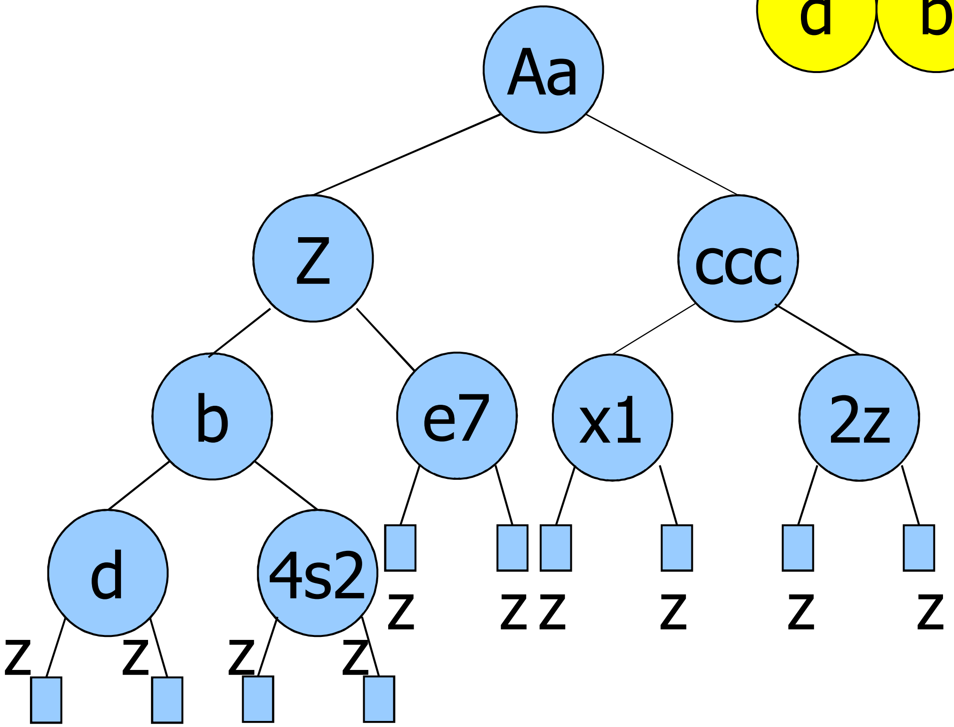
Aa Z b d 4s2 e7 ccc x1 2z



```
void preorder_r (
  link root,
  link z
) {
  if (root == z)
    return;
  printf("%s ", root->item);
  preorder_r (root->l, z);
  preorder_r (root->r, z);
  return;
}
```

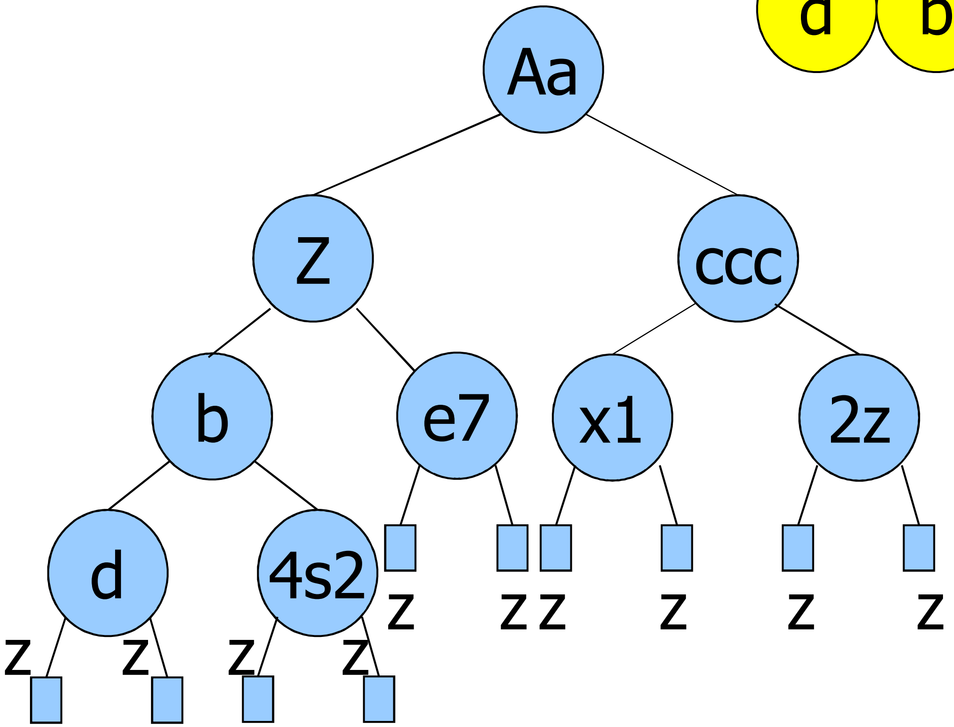
In-order

d b 4s2 Z e7 Aa x1 ccc 2z



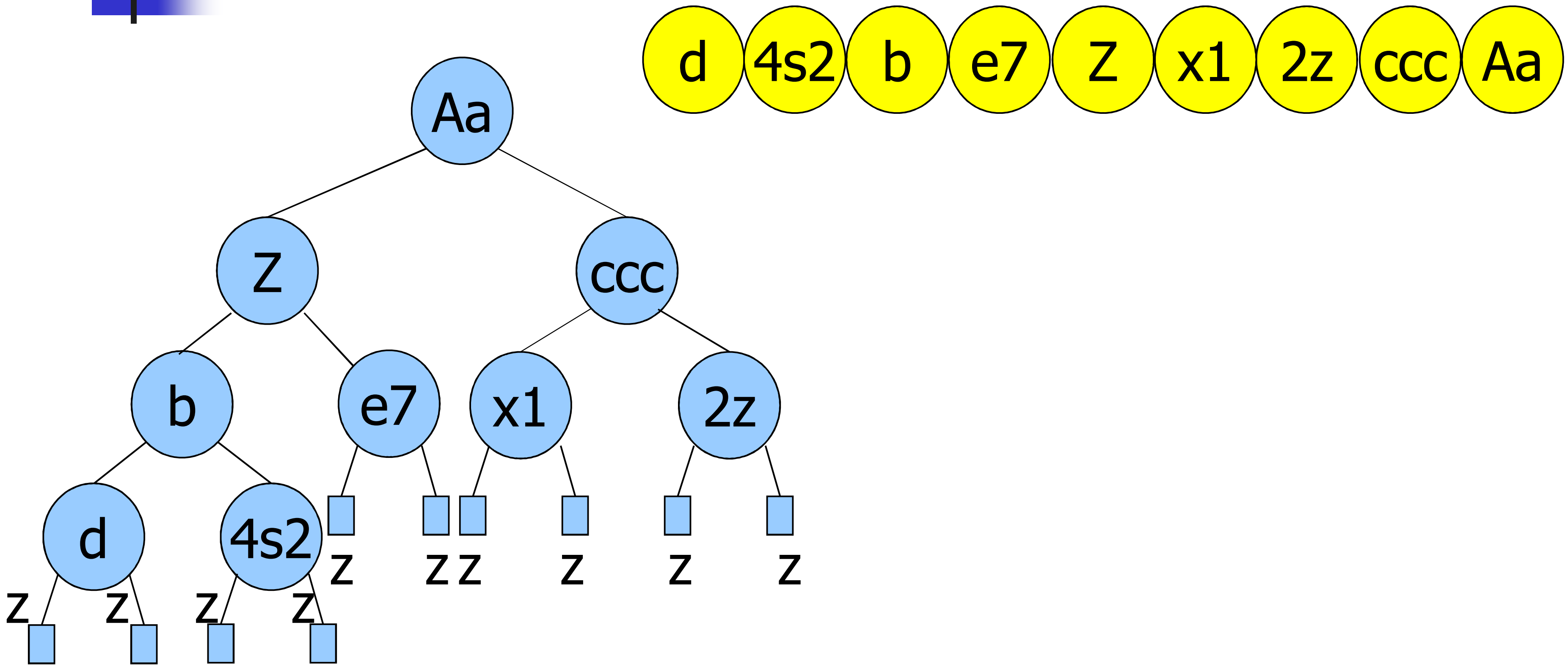
In-order

d b 4s2 Z e7 Aa x1 ccc 2z



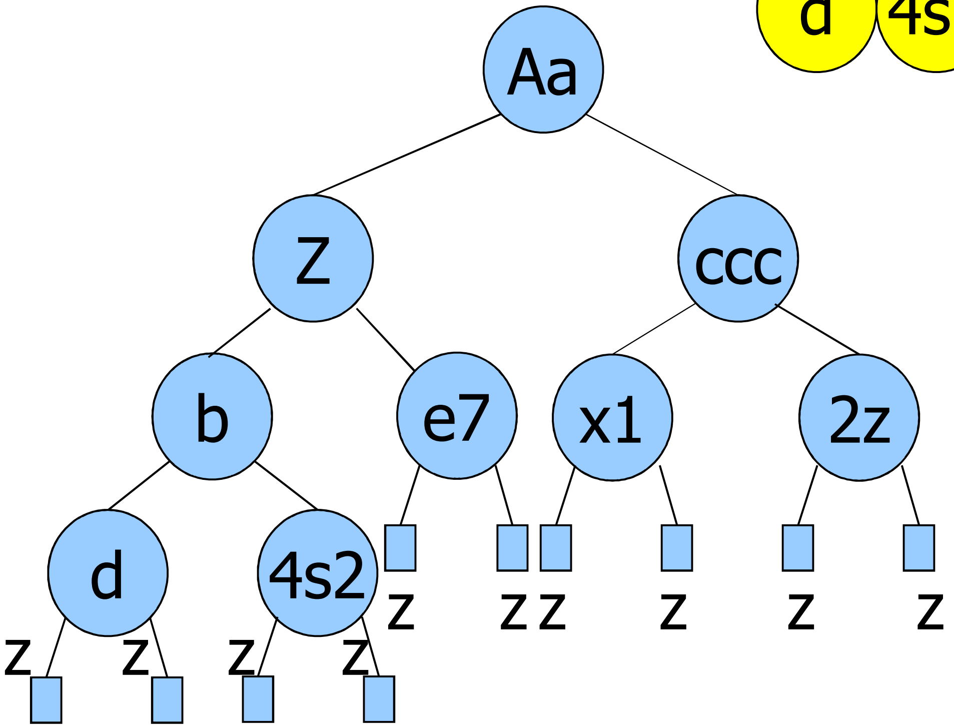
```
void inorder_r (
  link root,
  link z
){
  if (root == z)
    return;
  inorder_r (root->l, z);
  printf("%s ", root->item);
  inorder_r (root->r, z);
  return;
}
```

Post-order



Post-order

d 4s2 b e7 Z x1 2z ccc Aa



```
void postorder_r(
  link root,
  link z
){
  if (root == z)
    return;
  postorder_r (root->l, z);
  postorder_r (root->r, z);
  printf("%s ",root->item);
  return;
}
```

Complexity analysis: Case 1

Divide and conquer
a = 2 b = 2

- Complete tree
  - $D(n) = \Theta(1)$, $C(n) = \Theta(1)$
  - a = 2, b = 2 (two sub-problems of overall size n-1, conservatively approximated to n, i.e., n/2 and n/2)
- Recurrence equation
  - $T(n) = 1 + 2T(n/2)$ $\quad n > 1$
  - $T(1) = 1$
- $T(n) = O(n)$

Complexity analysis: Case 2

Divide and conquer
$a = 2$ $b = 2$

- Totally unbalanced tree (degenerated into a list)
  - $D(n) = \Theta(1)$, $C(n) = \Theta(1)$
  - $a = 1$, $k_i = 1$
- Recurrence equation
  - $T(n) = 1 + T(n-1)$ $n > 1$
  - $T(1) = 1$
- $T(n) = O(n)$

Parameter computation

Number of nodes

```
int count(link root, link z) {
  int u, v;

  if (root == z)
    return 0;

  u = count(root->l, z);
  v = count(root->r, z);

  return (u+v+1);
}
```

Parameter computation

Height

```
int height(link root, link z) {
 int u, v;

 if (root == z)
   return -1;

 u = height(root->l, z);
 v = height(root->r, z);

 if (u>v)
   return u+1;
 else
   return v+1;
}
```

A Binary Tree Application: Expressions

- Given an algebraic expression (brackets to change operator priority), it is possibile to build the corresponding tree according to the simplified grammar
  - <exp> = <operand> | <exp> <op> <exp>
  - <operand> = A .. Z
  - <op> = + | * | - | /

Recursion

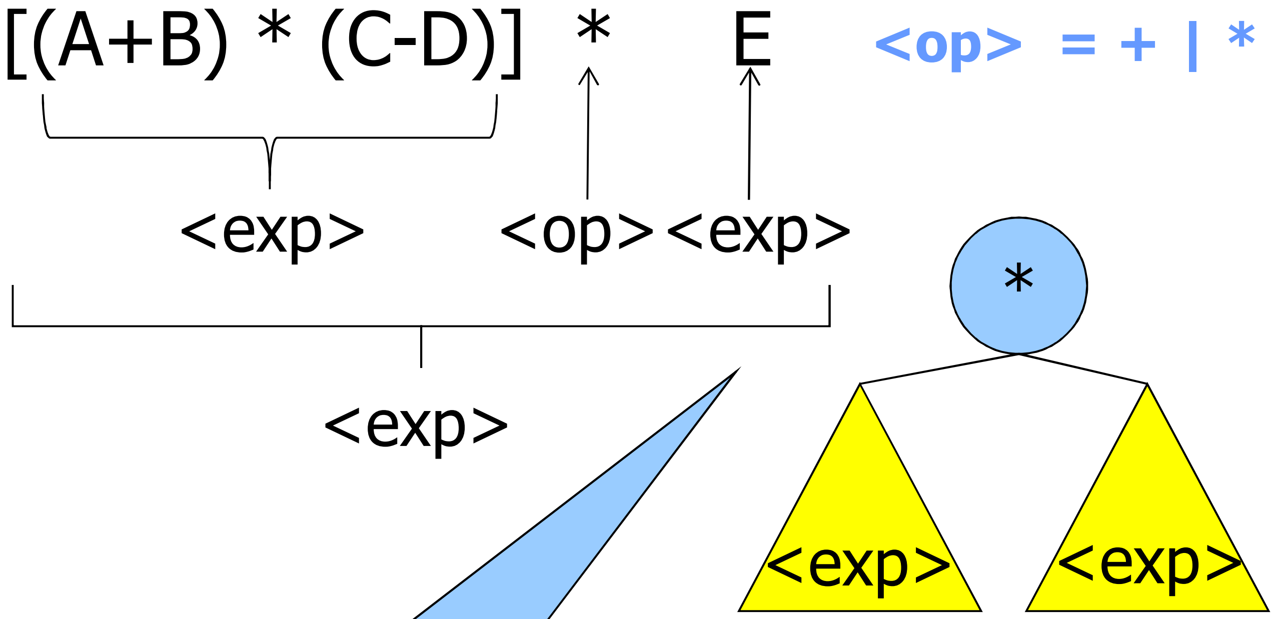
Termination condition

An example

<exp> = <operand> |
<exp> <op> <exp>

<operand> = A .. Z

<op> = + | * | - | /

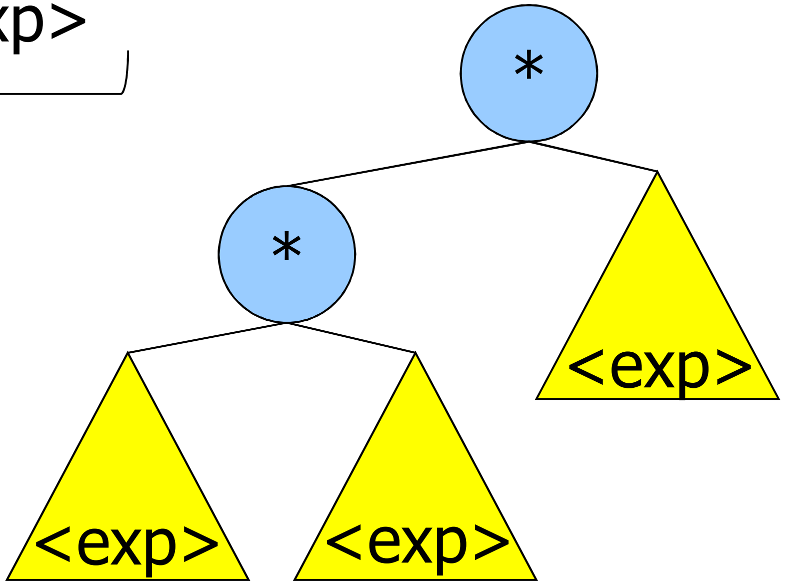


Parsing the expression
we obtain

An example

<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

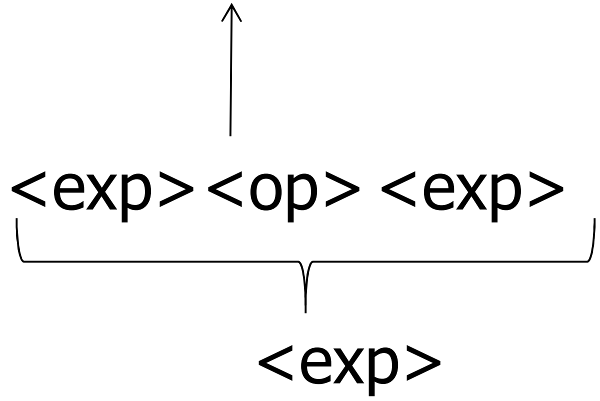
$$\underbrace{[\underbrace{(A+B)}_{<exp>}\ \underbrace{*}_{<op>}\ \underbrace{(C-D)}_{<exp>}]}_{<exp>} * \quad E$$



An example

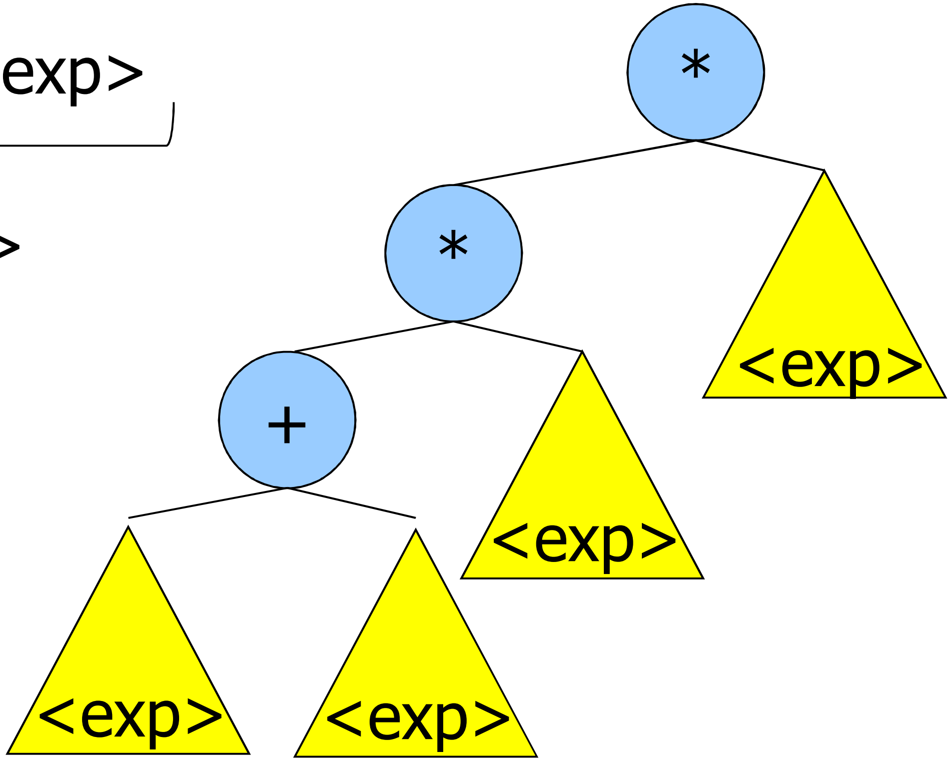
<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] *   E



<exp><op><exp>

<exp>

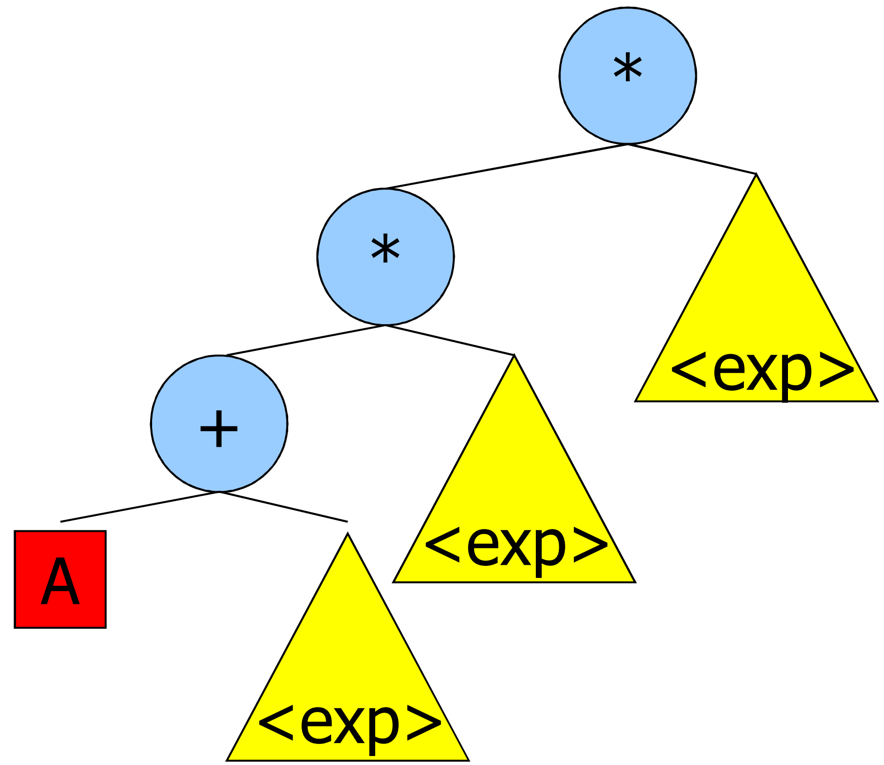


An example

<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] * E

<operand>



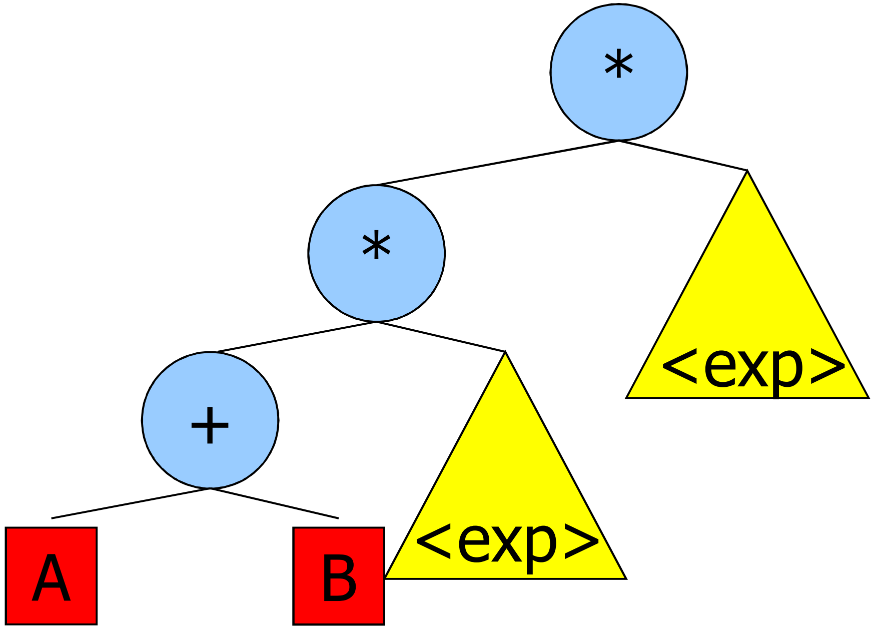
An example

<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] * E

↑

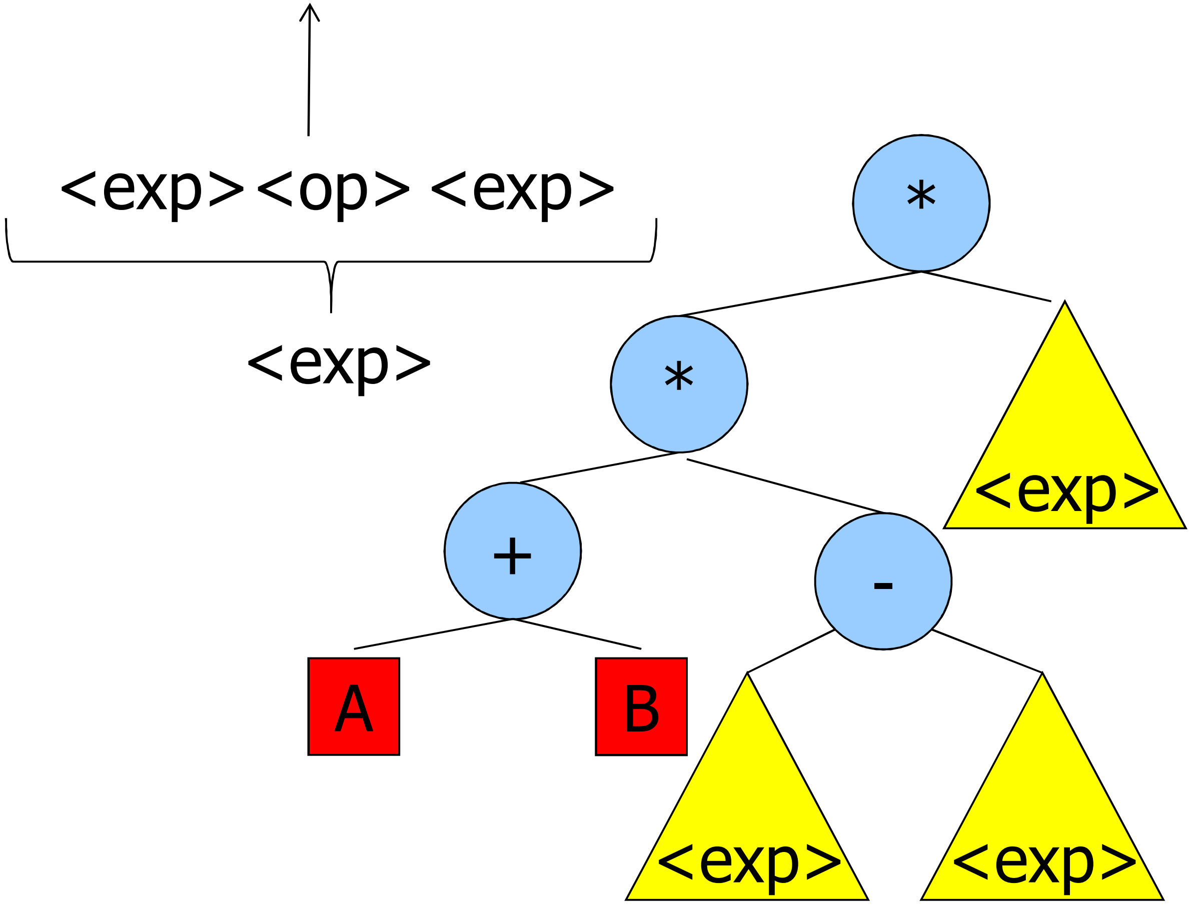
<operand>



An example

<exp> = <operand> |
                <exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] *   E



<exp><op> <exp>

<exp>

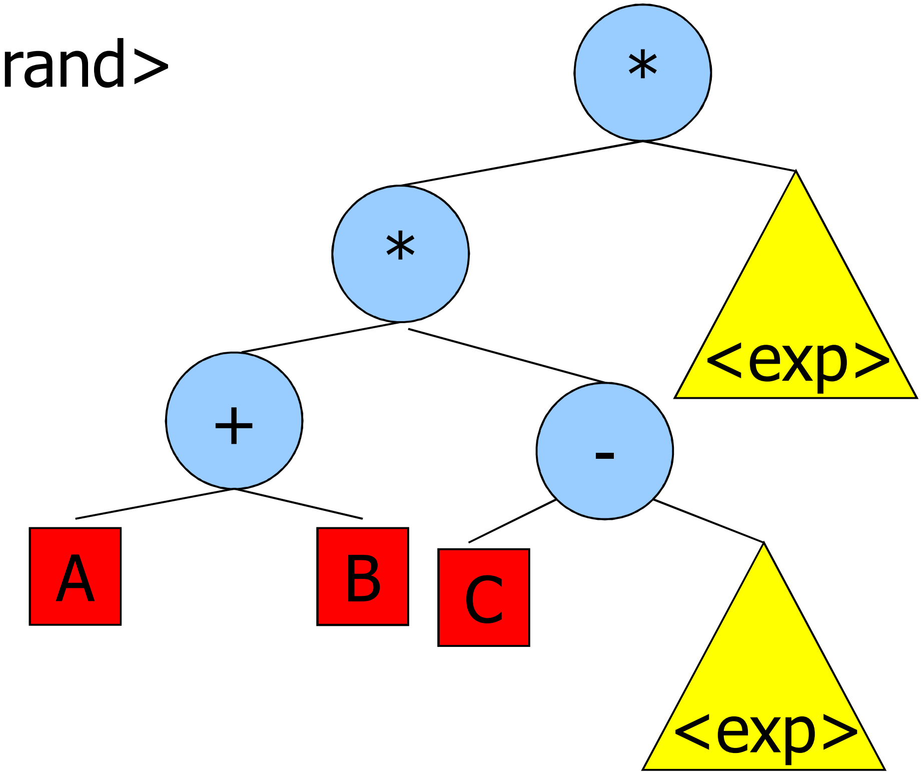
An example

<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] * E

↑

<operand>



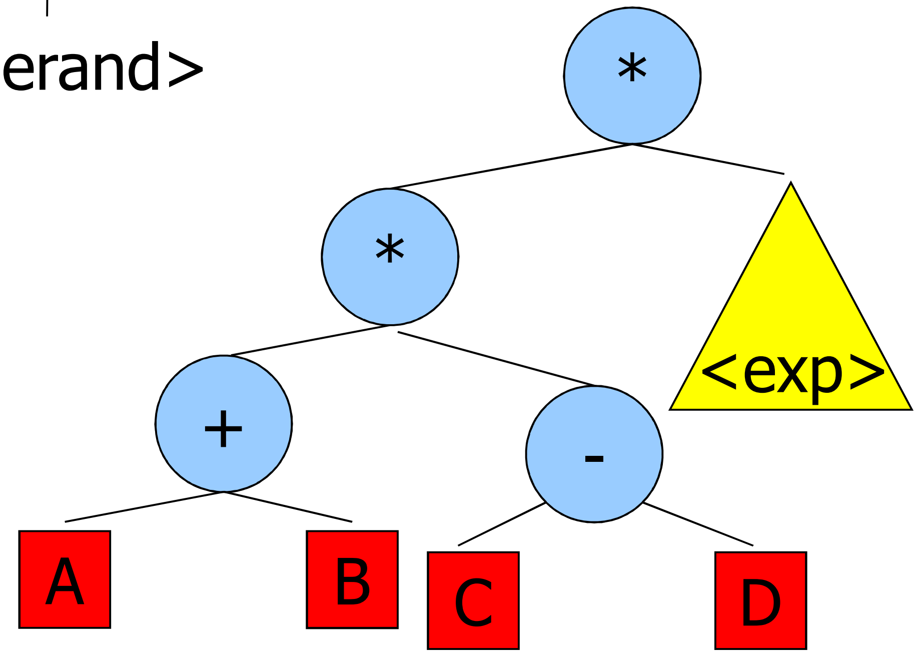
An example

<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] * E

↑

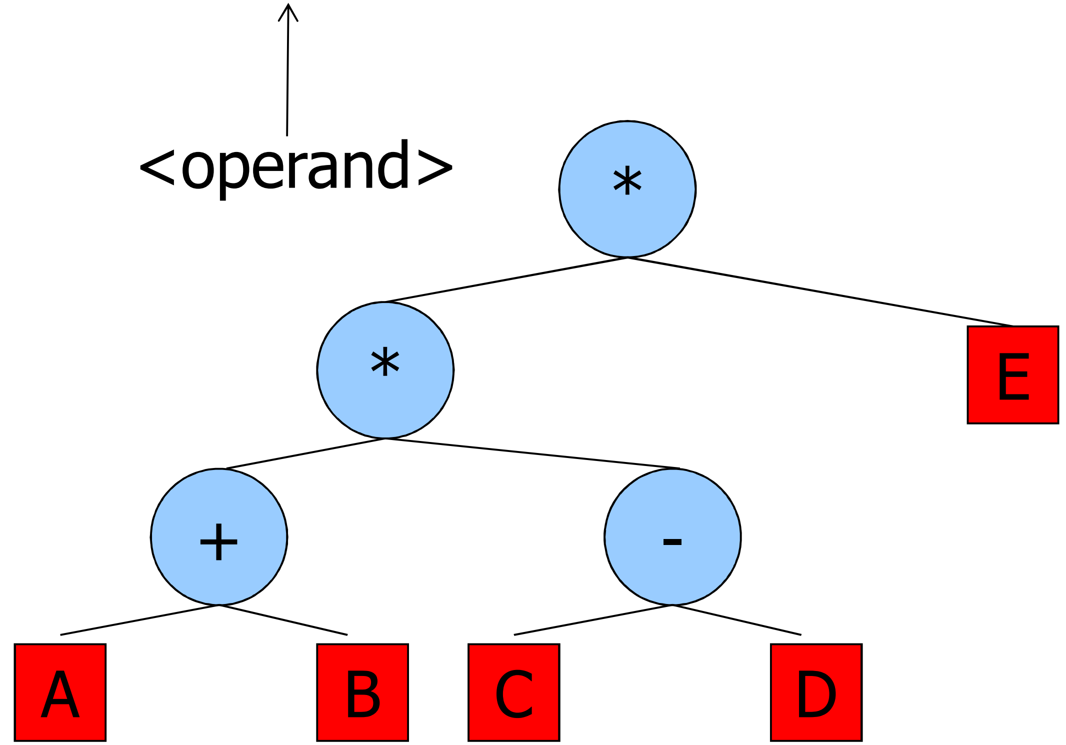
<operand>



An example

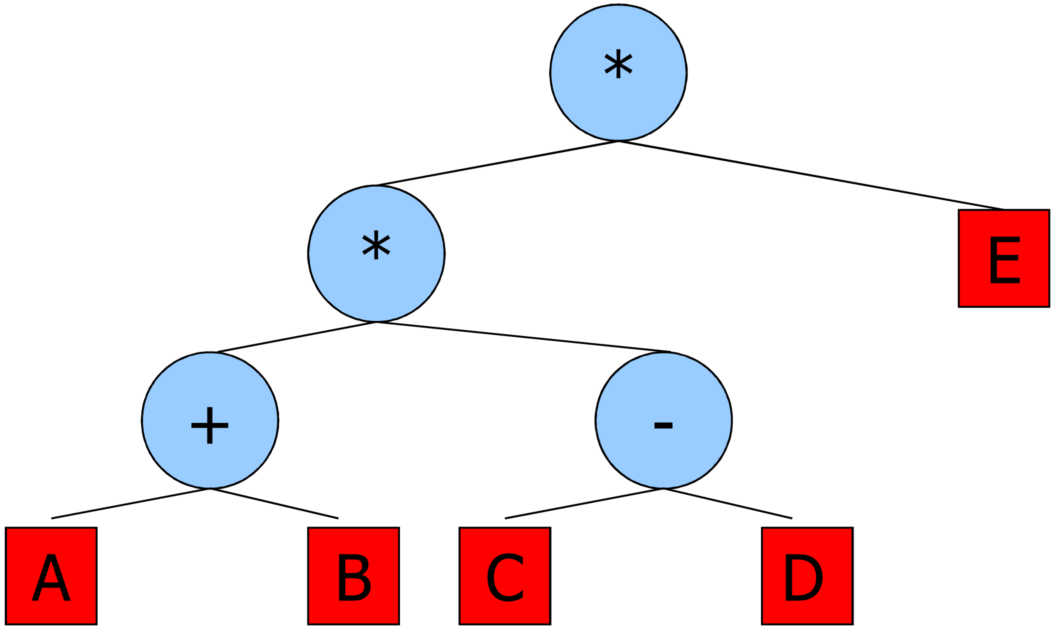
<exp> = <operand> |
<exp> <op> <exp>
<operand> = A .. Z
<op> = + | * | - | /

[(A+B) * (C-D)] * E



An example

A post-order visits returns the expression in postfix form (Reverse Polish Notation)

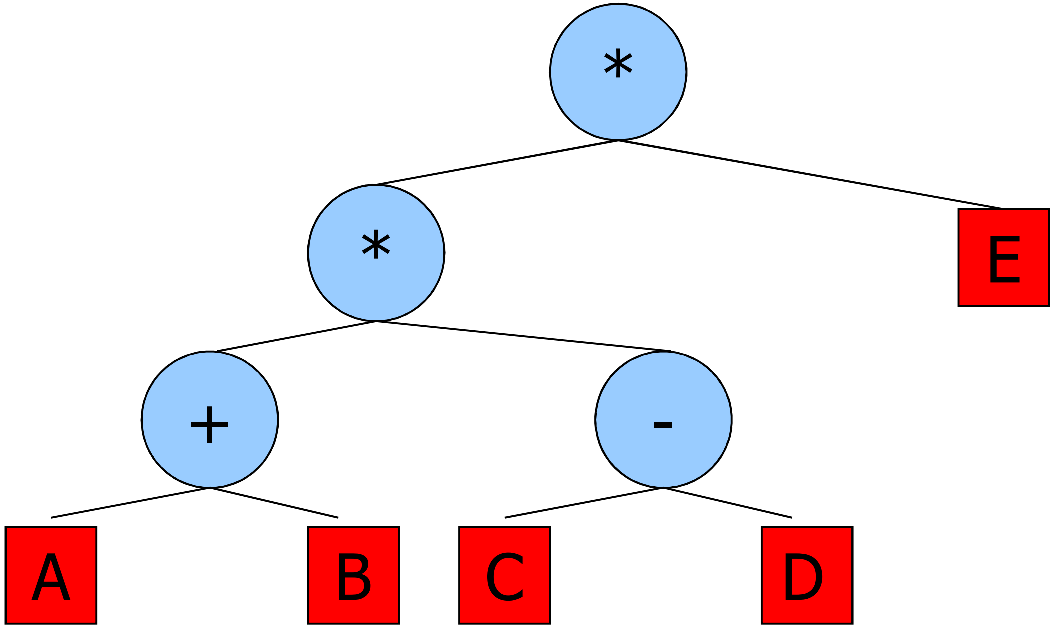


A B + C D - * E *

Brackets no more needed

An example

A pre-order visits returns the expression in the seldom used prefix form (Polish Notation)



* * + A B – C D E

Brackets no more needed