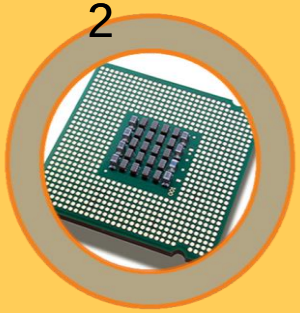
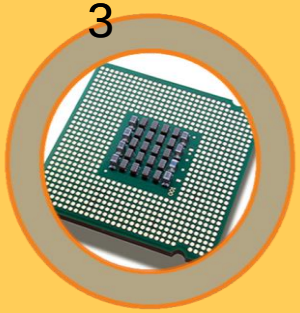
# 7.2: Methods for defining syntax



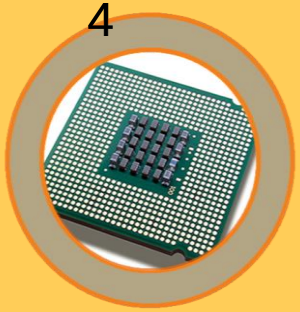
# What this module is about

- In this module we discuss:
- explain how functions, procedures and their related variables may be used to develop a program in a structured way, using stepwise refinement;
- describe the use of parameters, local and global variables as standard programming techniques;
- explain how a stack is used to handle procedure calling and parameter passing;
- explain the need for, and be able to create and apply, BNF (Backus-Naur form) and syntax diagrams;
- **explain the need for reverse Polish notation;**
- **convert between reverse Polish notation and infix form of algebraic expressions using trees and stacks.**



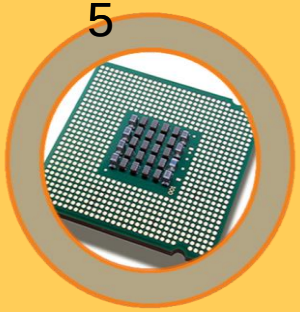
# Introduction

- One of the common requirements of the CPU is to be able to handle algebraic expressions such as: 4(3-2)*(5+6)
- We know that mathematical operators have precedence - they have an order. The brackets have to be evaluated first, then mulitply or divide and so on. Working with the example above we get : 4(1)*(11)
- then deal with the left term we get: 4*11
- Answer: 44
- This form of notation is called algebraic '**infix**' notation.
- CPUs do not evaluate expressions in this way. They use the '***stack***' data structure to evaluate expressions, and infix notation is not compatible with a stack and so another method must be found.



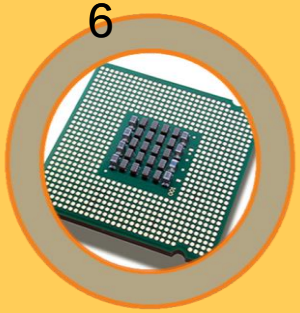
# Reverse Polish Notation

- For example a standard infix expression looks like: **4+5**
- where the + operator is placed between the numbers being added. And the answer is 9 of course.
- In reverse polish notation, the same expression is set out as follows: **4 5 +**
- The rule is that the operator acts upon the preceding two numbers. Therefore adding 4 and 5 together is still 9.



# Reverse Polish Notation

- Consider this expression: 6*(4+5) - 25/(2+3)
  - The answer is 49
- The same expression in reverse polish: 6 4 5 + * 25 2 3 + / -
  - To work it out, start from the left and go to the first operator then carry out that operation on the preceding two numbers.
  - 6 4 5 + * 25 2 3 + / - becomes 6 9 * 25 2 3 + / -
    - the next operator is a multiply * so the expression is now
      - 54 25 2 3 + / -
    - add the 2 and 3 together to make 5, the expression is now
      - 54 25 5 / -
    - the next operator is a divide, so divide 25 by 5 to get the expression
      - 54 5 -
    - finally subtract 5 from 54 and the answer is 49. The same as the infix expression.



# Using stack with reverse polish

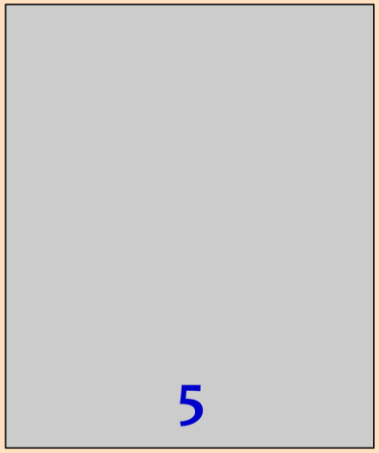
The Stack and Reverse Polish Notation

| Infix expression | Reverse Polish Notation |
|---|---|
| 5+((1+2)*4)-3 | 5 1 2 + 4 * 3 - + |

Steps

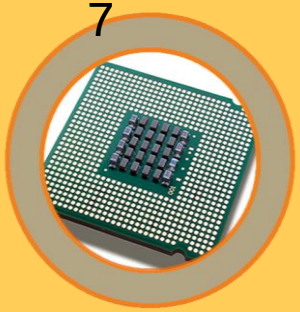
1. Push value 5
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 

The Stack



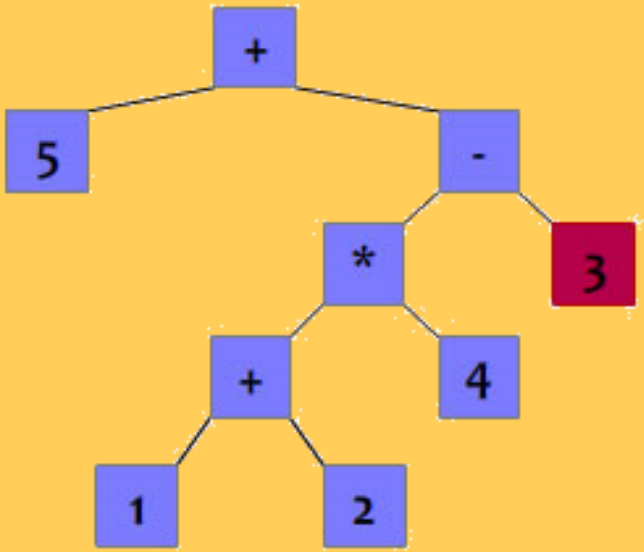

Press for next step

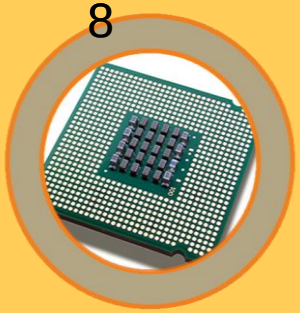
*By R Sorhaindo for Hartismere*



# Binary Tree

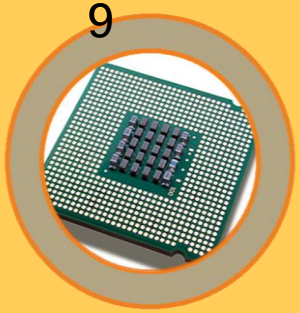
- You can use a binary tree to convert between infix and reverse Polish notation.
- The image below is for the infix for:
  - 5+((1+2)*4)-3





# Binary tree and reverse Polish

- The algorithm to use to derive the reverse Polish expression is listed below
  1. Traverse the left sub tree until there isn't one
  2. Traverse the right sub tree (if there is one)
  3. Re-visit the root of the current branch
  4. Repeat 1,2,3 until every node has been visited.
- This is called the '**postorder**' algorithm. The algorithm is an example of recursion because it repeats the same set of actions at every node.
- It is possible to also derive the expression in reverse polish by using another algorithm called the '**preorder**' algorithm. This is
  1. Visit the root node
  2. Visit the left branch
  3. Visit the right branch



# Binary tree and reverse Polish

## Binary Tree and Reverse Polish Notation

**Binary Tree for**

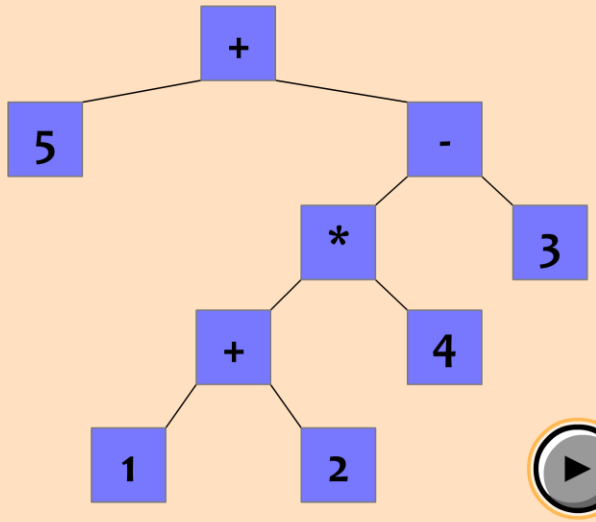
5+((1+2)*4)-3

**Reverse Polish Notation**

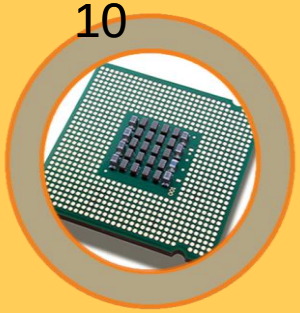
5 1 2 + 4 * 3 - +

Algorithm used to derive the RPN from the tree

1. Traverse the left sub tree until there is none
2. Traverse the right sub tree(if there is one)
3. Re-visit the root of branch
4. Repeat until all nodes seeen

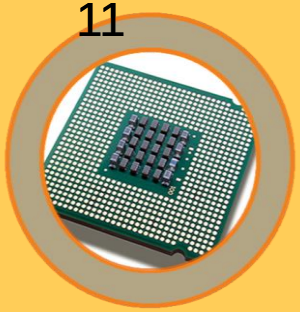



*By R Sorhaindo for Hartismere*



# Binary tree and infix notation

- The algorithm used to traverse the binary tree in order to derive the infix expression is:
  1. Traverse the left branch until there is no left branch
  2. Visit the root of the current branch
  3. Traverse the right sub tree if there is one
  4. Repeat steps 1, 2 and 3 until every node has been visited.
- This is called the 'inorder' algorithm. The algorithm is an example of recursion because it repeats the same set of actions at every node.



# Binary tree and infix notation

## Binary Tree and Infix Notation

**Inix Expression**

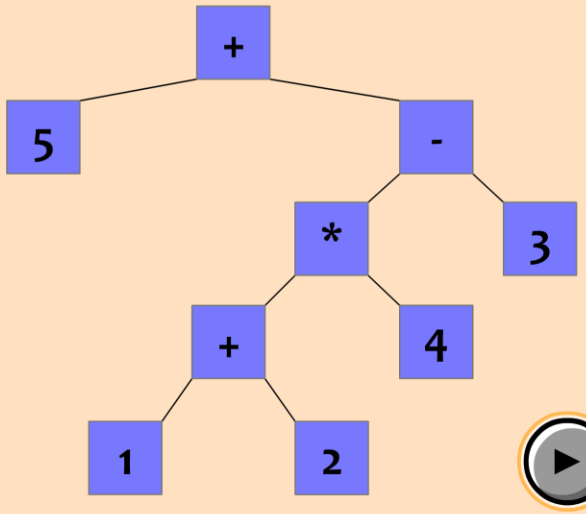
5+((1+2)*4)-3

**Reverse Polish Notation**

5 1 2 + 4 * 3 - +

Algorithm used to derive the Infix Expression from the tree

1. Traverse the left sub tree until there is none
2. Re-visit the root of the branch
3. Traverse the right sub tree if there is one
4. Repeat until all nodes seeen




*By R Sorhaindo for Hartismere*