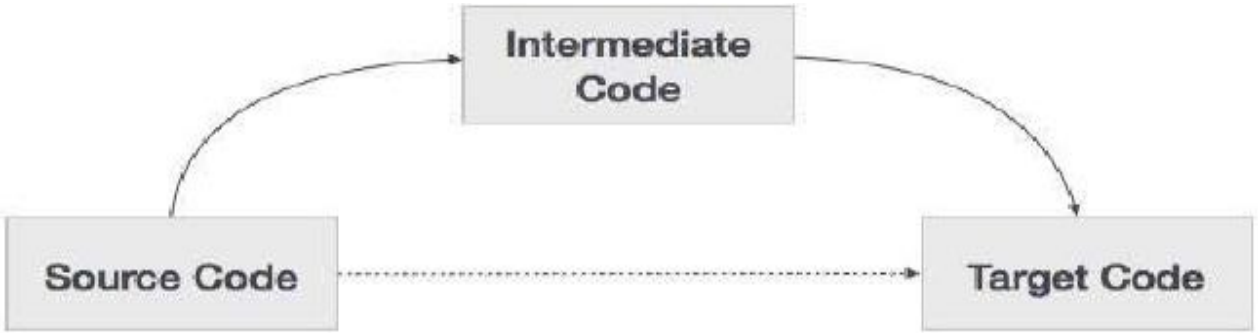
# Topics

## Intermediate Code generation

# What is intermediate code?

During the translation of a source program into the object code for a target machine, a compiler may generate a middle-level language code, which is known as **intermediate code**



# intermediate code

The following are commonly used intermediate code representation：

- Syntax tree
- Postfix Notation
- Three-Address Code

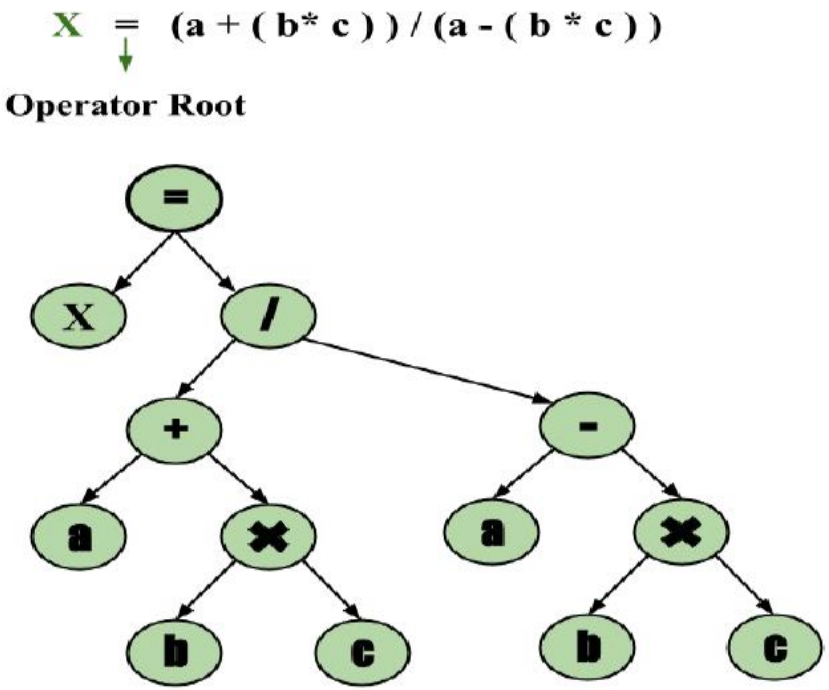
# Syntax tree

Syntax tree is nothing more than condensed form of a parse tree. The operator and keyword nodes of the parse tree are moved to their parents and a chain of single productions is replaced by single link in syntax tree the internal nodes are operators and child nodes are operands. To form syntax tree put parentheses in the expression, this way it's easy to recognize which operand should come first.

# Syntax tree

**Example –**

x = (a + b * c) / (a – b * c)



# Postfix Notation

- The ordinary (infix) way of writing the sum of a and b is with operator in the middle : a + b
- The postfix notation for the same expression places the operator at the right end as ab +. In general, if e1 and e2 are any postfix expressions, and + is any binary operator, the result of applying + to the values denoted by e1 and e2 is postfix notation by e1e2 +. No parentheses are needed in postfix notation because the position and arity (number of arguments) of the operators permit only one way to decode a postfix expression. In postfix notation the operator follows the operand.

# Postfix Notation

**Example –** The postfix representation of the expression (a – b) * (c + d) + (a – b) is

ab – cd + ab -+*.

# Three-Address Code

A statement involving no more than three references(two for operands and one for result) is known as three address statement. A sequence of three address statements is known as three address code. Three address statement is of the form x = y op z , here x, y, z will have address (memory location). Sometimes a statement might contain less than three references but it is still called three address statement.

For Example :a = b + c * d;

The intermediate code generator will try to divide this expression into sub-expressions and then generate the corresponding code.

```
r1 = c * d;
r2 = b + r1;
a = r2
```

# Three-Address Code

A three-address code has at most three address locations to calculate the expression. A three-address code can be represented in two forms :

- Quadruples
- Triples
- Indirect Triples

# Quadruples

Each instruction in quadruples presentation is divided into four fields: operator, arg1, arg2, and result. The example is represented below in quadruples format:

```
r1 = c * d;
r2 = b + r1;
a = r2
```

| Op | $arg_1$ | $arg_2$ | result |
|---|---|---|---|
| * | c | d | r1 |
| + | b | r1 | r2 |
| + | r2 | r1 | r3 |
| = | r3 | | a |

# Triples

Each instruction in triples presentation has three fields : op, arg1, and arg2.The results of respective sub-expressions are denoted by the position of expression.

```
r1 = c * d;
r2 = b + r1;
a = r2
```

| Op | $arg_1$ | $arg_2$ |
|---|---|---|
| * | c | d |
| + | b | (0) |
| + | (1) | (0) |
| = | (2) | |

# Indirect Triples

This representation is an enhancement over triples representation. It uses pointers instead of position to store results. This enables the optimizers to freely re-position the sub-expression to produce an optimized code.

# Thank You