SEQUENCE CONTROL

Control Structure in a PL provides the basic framework within which operations and data are combined into a program and sets of programs.

Sequence Control: Control of the order of execution of the operations
Data Control: Control of transmission of data among subprograms of program

Sequence Control may be categorized into four groups:

1. **Expressions:** They form the building blocks for statements. An expression is a combination of variable constants and operators according to syntax of language. Properties as precedence rules and parentheses determine how expressions are evaluated
2. **Statements:** The statements (conditional & iterative) determine how control flows from one part of program to another.
3. **Declarative Programming:** This is an execution model of program which is independent of the program statements (Logic programming model of PROLOG).
4. **Subprograms :** In structured programming, program is divided into small sections and each section is called subprogram. Subprogram calls and co-routines, can be invoked repeatedly and transfer control from one part of program to another.

Implicit and Explicit Sequence Control
Implicit or default sequence-control structures are those defined by the programming language itself. These structures can be modified explicitly by the programmer (Example: Most languages define physical sequence as the sequence in which statements are executed).
Explicit sequence-control structures are those that programmer may optionally use to modify the implicit sequence of operations defined by the language (Example: Use parentheses within expressions, or GOTO statements and labels).

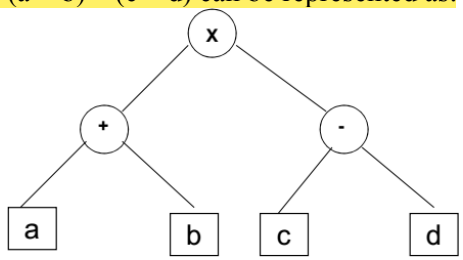
Sequence Control within Expressions
Expression is a formula which uses operators and operands to give the output value.
i) Arithmetic Expression: An expression consisting of numerical values (any number, variable or function call) together with some arithmetic operator is called "Arithmetic Expression".
Evaluation of Arithmetic Expression
Arithmetic Expressions are evaluated from left to right and using the rules of precedence of operators. If expression involves parentheses, the expression inside parentheses is evaluated first

ii) Relational Expressions:
An expression involving a relational operator is known as "Relational Expression". A relational expression can be defined as a meaningful combination of operands and relational operators. (a + b) > c c < b
Evaluation of Relational Expression
The relational operators <, >, <=, >= are given the first priority and other operators (== and !=) are given the second priority The arithmetic operators have higher priority over relational operators. The resulting expression will be of integer type, true = 1, false = 0.

iii) Logical Expression: An expression involving logical operators is called 'Logical expression". The expression formed with two or more relational expression is called logical expression. Ex. a > b && b < c
Evaluation of Logical Expression
The result of a logical expression is either true or false. For expression involving AND (&&), OR (||) and NOT(!) operations, expression involving NOT is evaluated first, then the expression with AND and finally the expression having OR is evaluated.

Controlling the evaluation of expressions
a) Precedence (Priority): If expression involving more than one operator is evaluated, the operator at higher level of precedence is evaluated first
b) Associativity: The operators of the same precedence are evaluated either from left to right or from right to left depending on the level Most operators are evaluated from left to right except + (unary plus), - (unary minus) ++, --, !, & Assignment operators = , +=, *=, /=, %=

Expression Tree
An expression (Arithmetic, relational or logical) can be represented in the form of an "expression tree". The last or main operator comes on the top (root).
Example: (a + b) * (c – d) can be represented as:



Syntax for Expressions
a) Prefix or Polish notation: Named after polish mathematician Jan Lukasiewicz, refers to notation in which operator symbol is placed before its operands.
*XY, -AB, /*ab-cd
Cambridge Polish: variant of notation used in LISP, parentheses surround an operator and its arguments.
(/(*ab)(-cd))

b) Postfix or reverse polish
Postfix refers to notation in which the operator symbol is placed after its two operands.
AB*, XY

c) Infix notation: It is most suitable for binary (dyadic) operation. The operator symbol is placed between the two operands

Semantics for Expressions
Semantics determine the order of expression in which they are evaluated.

a) Evaluation of Prefix Expression
If P is an expression evaluate using stack
1. If the next item in P is an operator, push it on the stack. set the arguments count to be number of operands needed by operator. (if number is n, operator is n-ary operator).
2. If the next item in P is an operand, push it on the stack
3. If the top n entries in the stack are operand entries needed for the last n-ary operator on the stack, apply the operator on those operands. Replace the operator and its n operands by the result of applying that operation on the n operands.

b) Evaluation of Postfix Expression
If P is an expression evaluate using stack
1. If the next item in P is an operand, push it on the stack.
2. If the next item in P is an n-ary operator, its n arguments must be top n items on the stack. Replace these n items by the result of applying this operation using the n items as arguments.

c) Evaluation of Infix Expression
Infix notation is common but its use in expression causes the problems:
1. Infix notation is suitable only for binary operations. A language cannot use only infix notation but must combine infix and postfix (or prefix) notations. The mixture makes translation complex.
2. If more than one infix operator is in an expression, the notation is ambiguous unless parentheses are used.

Execution-Time Representation
If we want to perform the expression than first transform each expression into its tree representation, we permit the translator to make alternative choices for efficient evaluation of the expression.

1. Machine code sequences: The expression is translated into actual machine code performing the above two stages of translation in one step. The ordering of the instructions reflects the sequence-control structure of the original expression.

2. Tree structures: The expressions may be executed directly in their natural tree-structure representation, using a software interpreter.

3. Prefix or Postfix form: Expression is prefix or postfix form may be executed by the simple interpretation algorithms.

Evaluation of tree representation

Eager evaluation - evaluate all operands before applying operators.
Lazy evaluation - first evaluates all operands and then apply operations

Problems:

1. **Side effects -** some operations may change operands of other operations.
2. **Error conditions -** may depend on the evaluation strategy (eager or lazy evaluation)
3. **Boolean expressions -** results may differ depending on the evaluation strategy.

Sequential Control within Statement

1. Basic Statements
i) Assignment Statement (Assignment operator (=), compound assignment operator (+=), MOVE A TO B. – COBOL)

ii) Input and Output Statement (printf, scanf)

iii) Declaration Statement (int age;)
iv) GoTo statement: (Explicit sequence control statement. Used to branch conditionally from one point to another in the program)
Example

```
int a, b;
Read:
scanf ("%d", &a);
if (a == 0) goto Read;
y = sqrt(x);
prinf("%d", y);
goto Read;
```

v) Break Statement: (An early exit from a loop can be accomplished by using break statement.)

2. Statement Level Sequence Control

A. **Implicit Sequence Control:** The natural or default programming sequence of a PL is called implicit sequence. They are of 3 types.
   i. **Composition Type:** Standard form of implicit sequence. Statements placed in order of execution.
   ii. **Alternation Type:** There are two alternate statement sequences in the program; the program chooses any of the sequence but not both at same type.
   iii. **Iteration Type:** Here normal sequence is given to statements but the sequence repeats itself for more than one time.

B. Explicit Sequence Control: The default sequence is altered by some special statements
   i. **Use of Goto statement**
   ii. Use of Break Statement

3. Structured Sequence Control

a) Compound Statement: Collection of two or more statements may be treated as single statement.
In C we have
{
……
……
……
}

b) Conditional Statements

if (conditional exp) then … statements

if (conditional exp) then … statements else … statements

if (conditional exp) then … statements else if (conditional exp) then … statements else … statements

switch (exp)
{
case val1: …statements break;
case val2: …statetments break;
default: statements break;
}

c) Iteration Statements

do {…….} while (conditional exp)

while (conditional exp) { ……}

for (initialization; test condition; increment) { …….}