# Linker Loader and Assembler

# UGC NET CS 2017 Jan-II

Which of the following statement(s) regarding a linker software is/are true?  
I A function of a linker is to combine several object modules into a single load module.  
II A function of a linker is to replace absolute references in an object module by symbolic references to locations in other modules.  
**(A)** Only I

**(B)** Only II

**(C)** Both I and II

**(D)** Neither I nor II

# GATE CS 1998

A linker reads four modules whose lengths are 200, 800, 600 and 500 words respectively. If they are loaded in that order, what are the relocation constants?  
**(A)** 0, 200, 500, 600

**(B)** 0, 200, 1000, 1600

**(C)** 200, 500, 600, 800

**(D)** 200, 700, 1300, 2100

# GATE CS 2001

The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned addresses is called  
(A) Assembly

(B) Parsing

(C) Relocation

(D) Symbol resolution

# GATE CS 1995

A linker is given object modules for a set of programs that were compiled separately.

What information need not be included in an object module?

(A) Object code

(B) Relocation bits

(C) Names and locations of all external symbols defined in the object module

(D) Absolute addresses of internal symbols.

# GATE CS 1998 | ISRO CS 2008

In a resident- OS computer, which of the following system software must reside in the main memory under all situations?  
**(A)** Assembler

**(B)** Linker

**(C)** Loader

**(D)** Compiler

# GATE CS 1993

A part of the system software which under all circumstances must reside in the main memory is:

(A) Text editor

(B) Assembler

(C) Linker

(D) Loader

# GATE CS 2010

Which data structure in a compiler is used for managing information about variables and their attributes?  
**(A)** Abstract syntax tree

**(B)** Symbol table

**(C)** Semantic stack

**(D)** Parse Table

# Phases of Compiler

# GATE-CS-2009

Match all items in Group 1 with correct options from those given in Group 2.

**Group 1 Group 2**

**P. Regular expression 1. Syntax analysis**

**Q. Pushdown automata 2. Code generation**

**R. Dataflow analysis 3. Lexical analysis**

**S. Register allocation 4. Code optimization**

**(A)** P-4. Q-1, R-2, S-3

**(B)** P-3, Q-1, R-4, S-2

**(C)** P-3, Q-4, R-1, S-2

**(D)** P-2, Q-1, R-4, S-3

**Answer:** **(B)**

# GATE-CS-2015 (Set 2)

Match the following:

**List-I List-II**

**A. Lexical analysis 1. Graph colouring**

**B. Parsing 2. DFA minimization**

**C. Register allocation 3. Post-order traversal**

**D. Expression evaluation 4. Production tree**

(A) A-2 B-3 C-1 D-4

(B) A-2 B-1 C-4 D-3

(C) A-2 B-4 C-1 D-3

(D) A-2 B-3 C-4 D-1

**Answer:** **(C)**

# GATE-CS-2016 (Set 2)

Match the following:

(P) Lexical analysis (i) Leftmost derivation

(Q) Top down parsing (ii) Type checking

(R) Semantic analysis (iii) Regular expressions

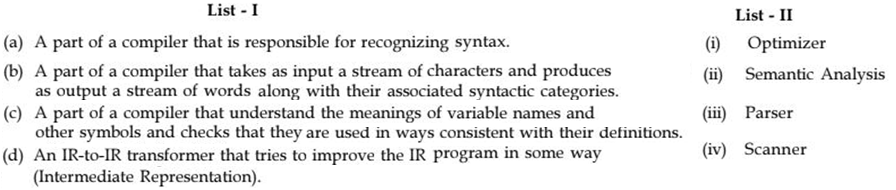
(S) Runtime environments (iv) Activation records



**Answer:** **(B)**

# UGC-NET CS 2017 Dec 2

Match the description of several parts of a classic optimizing compiler



(A) A-iii B-iv C-ii D-i

(B) A-iv B-iii C-ii D-i

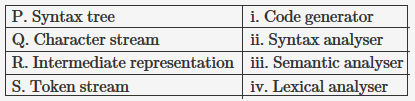
(C) A-ii B-iv C-i D-iii

(D) A-ii B-iv C-iii D-i

**Answer:** **(A)**

# GATE-CS-2017 (Set 2)

Match the following according to input(from the left column) to the compiler phase(in the right column) that process it:



**(A)** P -> (ii), Q -> (iii), R -> (iv), S -> (i)

**(B)** P -> (ii), Q -> (i), R -> (iii), S -> (iv)  
**(C)** P -> (iii), Q -> (iv), R -> (i), S -> (ii)

**(D)** P -> (i), Q -> (iv), R -> (ii), S -> (iii)

**Answer:** **(C)**

# UGC NET CS 2017 Jan – II

Consider the following statements related to compiler construction :  
I. Lexical Analysis is specified by context-free grammars and implemented by pushdown automata.  
II. Syntax Analysis is specified by regular expressions and implemented by finite-state machine.  
Which of the above statement(s) is/are correct ?  
**(A)** Only I

**(B)** Only II

**(C)** Both I and II

**(D)** Neither I nor II

**Answer:** **(D)**

# GATE CS 2018

Which one of the following statements is FALSE?

**(A)** Context-free grammar can be used to specify both lexical and syntax rules.  
**(B)** Type checking is done before parsing.  
**(C)** High-level language programs can be translated to different Intermediate Representations.  
**(D)** Arguments to a function can be passed using the program stack.  
  
**Answer:** **(B)**

# GATE CS 1998

Type checking is normally done during  
**(A)** Lexical analysis **(B)** Syntax analysis

**(C)** Syntax directed translation **(D)** Code optimization

**Answer:** **(C)**

# Analysis Phase- Lexical Analysis

# GATE CS 2011

In a compiler, keywords of a language are recognized during  
**(A)** parsing of the program **(B)** the code generation  
**(C)** the lexical analysis of the program **(D)** dataflow analysis  
  
**Answer:** **(C)**

# GATE CS 2011

The lexical analysis for a modern computer language such as Java needs the power of which one of the following machine models in a necessary and sufficient sense?  
**(A)** Finite state automata **(B)** Deterministic pushdown automata  
**(C)** Non-Deterministic pushdown automata **(D)** Turing Machine  
  
**Answer:** **(A)**

# GATE CS Mock 2018 | Set 2

Consider the following statements:  
(I) The output of a lexical analyzer is groups of characters.  
(II) Total number of tokens in printf("i=%d, &i=%x", i, &i); are 11.  
(III) Symbol table can be implementation by using array and hash table but not tree.

Which of the following statement(s) is/are correct?  
**(A)** Only (I) **(B)** Only (II) and (III)  
**(C)** All (I), (II), and (III) **(D)** None of these  
  
**Answer:** **(D)**

# GATE CS 2018

A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.

T1: a?(b∣c)\*a  
T2: b?(a∣c)\*b  
T3: c?(b∣a)\*c

Note that ‘x?’ means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix.

If the string *bbaacabc* is processes by the analyzer, which one of the following is the sequence of tokens it outputs?

**(A)** T1T2T3 **(B)** T1T1T3 **(C)** T2T1T3 **(D)** T3T3  
  
**Answer:** **(D)**

# UGC NET CS 2017 Jan – III

From the given data : a b b a a b b a a b  
which one of the following is not a word in the dictionary created by LZ-coding (the initial words are a, b)?  
**(A)** a b **(B)** b b **(C)** b a **(D)** b a a b  
  
**Answer:** **(B)** **(D)**

# GATE 2000

The number of tokens in the following C statement is

|  |
| --- |
| printf("i = %d, &i = %x", i, &i); |

**(A)** 3 **(B)** 26 **(C)** 10 **(D)** 21  
  
**Answer:** **(C)**

# UGC NET CS 2014 Dec – III

How many tokens will be generated by the scanner for the following statement ?  
x = x ∗ (a + b) – 5;  
**(A)** 12 **(B)** 11 **(C)** 10 **(D)** 07  
  
**Answer:** **(A)**

# Analysis Phase- Parser

# GATE-CS-2000

Which of the following derivations does a top-down parser use while parsing an input string? The input is assumed to be scanned in left to right order.  
**(A)** Leftmost derivation

**(B)** Leftmost derivation traced out in reverse

**(C)** Rightmost derivation

**(D)** Rightmost derivation traced out in reverse

**Answer:** **(A)**

# GATE-CS-2007

Which one of the following is a top-down parser?  
**(A)** Recursive descent parser.

**(B)** Operator precedence parser.  
**(C)** An LR(k) parser.

**(D)** An LALR(k) parser

**Answer:** **(A)**

# GATE CS 1998

Which of the following statement is true?  
**(A)** SLR parser is more powerful than LALR.  
**(B)** LALR parser is more powerful than Canonical LR parser.  
**(C)** Canonical LR parser is more powerful than LALR parser.  
**(D)** The parsers SLR, Canonical LR, and LALR have the same power.  
  
**Answer:** **(C)**

# GATE-CS-2015 (Set 3)

Among simple LR (SLR), canonical LR, and look-ahead LR (LALR), which of the following pairs identify the method that is very easy to implement and the method that is the most powerful, in that order?

**(A)** SLR, LALR

**(B)** Canonical LR, LALR

**(C)** SLR, canonical LR

**(D)** LALR, canonical LR

**Answer:** **(C)**

# GATE-CS-2001

Which of the following statements is false?  
**(A)** An unambiguous grammar has same leftmost and rightmost derivation  
**(B)** An LL(1) parser is a top-down parser  
**(C)** LALR is more powerful than SLR  
**(D)** An ambiguous grammar can never be LR(k) for any k  
  
**Answer:** **(A)**

# GATE-CS-2006

Consider the following grammar:

S → FR

R → S | ε

F → id

In the predictive parser table, M, of the grammar the entries M[S, id] and M[R, $] respectively.  
**(A)** {S → FR} and {R → ε } **(B)** {S → FR} and { }  
**(C)** {S → FR} and {R → \*S} **(D)** {F → id} and {R → ε}  
  
**Answer:** **(A)**

# GATE-CS-2003

Consider the grammar shown below  
S → i E t S S’ | a  
S’ → e S | ε  
E → b  
In the predictive parse table. M, of this grammar, the entries M[S’, e] and M[S’, $] respectively are  
**(A)** {S’ → eS} and {S’ → e} **(B)** {S’ → eS} and {}  
**(C)** {S’ → ε} and {S’ → ε} **(D)** {S’ → eS, S’→ ε} and {S’ → ε}

**Answer:** **(D)**

# GATE-CS-2003

Which of the following suffices to convert an arbitrary CFG to an LL(1) grammar?  
**(A)** Removing left recursion alone

**(B)** Factoring the grammar alone  
**(C)** Removing left recursion and factoring the grammar

**(D)** None of these

**Answer:** **(D)**

# GATE-CS-2004

Which of the following grammar rules violate the requirements of an operator grammar ?

Note: P, Q, R are non-terminals, and r, s, t are terminals symbols.

1. P → Q R 2. P → QsR 3. P → ε 4. P → QtRr

**(A)** 1 only

**(B)** 1 and 3 only

**(C)** 2 and 3 only

**(D)** 3 and 4 only

**Answer:** **(B)**

# GATE CS 2013

Consider the following two sets of LR(1) items of an LR(1) grammar.

|  |  |
| --- | --- |
| X -> c.X, c/d  X -> .cX, c/d  X -> .d, c/d | X -> c.X, $  X -> .cX, $  X -> .d, $ |

Which of the following statements related to merging of the two sets in the corresponding LALR parser is/are FALSE?

1. Cannot be merged since look aheads are different.

2. Can be merged but will result in S-R conflict.

3. Can be merged but will result in R-R conflict.

4. Cannot be merged since goto on c will lead to two different sets.

**(A)** 1 only **(B)** 2 only **(C)** 1 and 4 only **(D)** 1, 2, 3, and 4

**Answer:** **(D)**

# GATE-CS-2003

Consider the grammar shown below.

S → CC

C → cC | d

The grammar is  
**(A)** LL(1) **(B)** SLR(1) but not LL(1)  
**(C)** LALR(1) but not SLR(1) **(D)** LR(1) but not LALR(1)  
  
**Answer:** **(A)**

# GATE CS 2010

The grammar S → aSa | bS | c is  
**(A)** LL(1) but not LR(1) **(B)** LR(1)but not LR(1)  
**(C)** Both LL(1)and LR(1) **(D)** Neither LL(1)nor LR(1)

**Answer:** **(C)**

# GATE-CS-2003

Assume that the SLR parser for a grammar G has n1 states and the LALR parser for G has n2 states. The relationship between n1 and n2 is:  
**(A)** n1 is necessarily less than n2 **(B)** n1 is necessarily equal to n2  
**(C)** n1 is necessarily greater than n2 **(D)** none of these  
  
**Answer:** **(B)**

# GATE-CS-2007

Consider the following two statements:

P: Every regular grammar is LL(1)

Q: Every regular set has a LR(1) grammar

Which of the following is TRUE?  
**(A)** Both P and Q are true **(B)** P is true and Q is false

**(C)** P is false and Q is true **(D)** Both P and Q are false

**Answer:** **(C)**

# UGC NET CS 2015 Jun – II

Which one from the following is false?  
**(A)** LALR parser is Bottom – Up parser  
**(B)** A parsing algorithm which performs a left to right scanning and a right most deviation is RL (1)  
**(C)** LR parser is Bottom – Up parser.  
**(D)** In LL(1), the 1 indicates that there is a one – symbol look – ahead.  
  
**Answer:** **(B)**

# UGC NET CS 2014 Dec – II

Which of the following is true?  
**(A)** Canonical LR parser is LR (1) parser with single look ahead terminal  
**(B)** All LR(K) parsers with K > 1 can be transformed into LR(1) parsers.  
**(C)** Both (A) and (B)  
**(D)** None of the above  
  
**Answer:** **(C)**

# GATE-CS-2017 (Set 1)

Consider the following grammar

p --> xQRS

Q --> yz|z

R --> w|∈

S -> y

Which is FOLLOW(Q)?

**(A)** {R} **(B)** {w} **(C)** {w, y} **(D)** {w, ∉}

**Answer:** **(C)**

# GATE CS 2019

Consider the following given grammar:

S → Aa

A → BD

B → b|ε

D → d|ε

Let a, b, d and $ be indexed as follows:

Compute the FOLLOW set of the non-terminal B and write the index values for the symbols in the FOLLOW set in the descending order. (For example, if the FOLLOW set is {a, b, d, $}, then the answer should be 3210). ………………

**Answer:** **31**

# GATE-2019 | UGC NET CS 2018 July – II

A bottom-up parser generates:  
**(A)** Left-most derivation in reverse **(B)** Right-most derivation in reverse  
**(C)** Left-most derivation **(D)** Right-most derivation

**Answer:** **(B)**

# ISRO CS 2013

Shift reduce parsing belongs to a class of  
**(A)** bottom up parsing **(B)** top down parsing **(C)** recursive parsing **(D)** predictive parsing

**Answer:** **(A)**

# GATE CS 1997

Which of the following is essential for converting an infix expression to the postfix from efficiently?

**(A)** An operator stack **(C)** An operand stack and an operator stack   
**(B)** An operand stack **(D)** A parse tree

**Answer:** **(A)**

# Analysis Phase- Syntax directed definitions

# GATE-CS-2003

Consider the translation scheme shown below

S → T R

R → + T {print ('+');} R | ε

T → num {print (num.val);}

Here num is a token that represents an integer and num.val represents the corresponding integer value. For an input string ‘9 + 5 + 2’, this translation scheme will print  
**(A)** 9 + 5 + 2

**(B)** 9 5 + 2 +

**(C)** 9 5 2 + +

**(D)** + + 9 5 2  
  
**Answer:** **(B)**

# GATE-CS-2006

Consider the following translation scheme.  
S → ER  
R → \*E{print(“\*”);}R | ε  
E → F + E {print(“+”);} | F  
F → (S) | id {print(id.value);}  
Here id is a token that represents an integer and id.value represents the corresponding integer value. For an input ‘2 \* 3 + 4’, this translation scheme prints  
**(A)** 2 \* 3 + 4

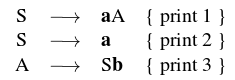
**(B)** 2 \* +3 4

**(C)** 2 3 \* 4 +

**(D)** 2 3 4+\*  
  
**Answer:** **(D)**

# GATE-CS-2016 (Set 1)

Consider the following Syntax Directed Translation Scheme (SDTS), with non-terminals {S, A} and terminals {a, b}}.

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2016/02/gt46.png)

Using the above SDTS, the output printed by a bottom-up parser, for the input **aab** is  
**(A)** 1 3 2

**(B)** 2 2 3

**(C)** 2 3 1

**(D)** Syntax Error  
  
**Answer:** **(C)**

# GATE-CS-1995

A shift reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar

S  xxW {print “1”} S  y {print “2” } W  Sz {print “3” }

What is the translation of xxxxyzz using the syntax directed translation scheme described by the above rules ?

(A) 23131 (B) 11233 (C) 11231 (D) 33211

**Answer:** **(A)**

# GATE-CS-2004

Consider the grammar with the following translation rules and E as the start symbol.

E → E1 # T { E.value = E1.value \* T.value }

| T { E.value = T.value }

T → T1 & F { T.value = T1.value + F.value }

| F { T.value = F.value }

F → num { F.value = num.value }

Compute E.value for the root of the parse tree for the expression: 2 # 3 & 5 # 6 & 4.  
**(A)** 200 **(B)** 180 **(C)** 160 **(D)** 40  
  
**Answer:** **(C)**

# GATE-CS-2005

Consider the following expression grammar. The seman­tic rules for expression calculation are stated next to each grammar production.

E → number E.val = number. val

| E '+' E E(1).val = E(2).val + E(3).val

| E '×' E E(1).val = E(2).val × E(3).val

The above grammar and the semantic rules are fed to a yacc tool (which is an LALR (1) parser generator) for parsing and evaluating arithmetic expressions. Which one of the following is true about the action of yacc for the given grammar?  
**(A)** It detects recursion and eliminates recursion  
**(B)** It detects reduce-reduce conflict, and resolves  
**(C)** It detects shift-reduce conflict, and resolves the conflict in favor of a shift over a reduce action  
**(D)** It detects shift-reduce conflict, and resolves the conflict in favor of a reduce over a shift action

**Answer:** **(C)**

# GATE-CS-2005

Consider the following expression grammar. The seman­tic rules for expression calculation are stated next to each grammar production.

E → number E.val = number. val

| E '+' E E(1).val = E(2).val + E(3).val

| E '×' E E(1).val = E(2).val × E(3).val

Assume the conflicts in Part (a) of this question are resolved and an LALR(1) parser is generated for parsing arithmetic expressions as per the given grammar. Consider an expression 3 × 2 + 1. What precedence and associativity properties does the generated parser realize?  
**(A)** Equal precedence and left associativity; ex­pression is evaluated to 7  
**(B)** Equal precedence and right associativity; ex­pression is evaluated to 9  
**(C)** Precedence of ‘×’ is higher than that of ‘+’, and both operators are left associative; expression is evaluated to 7  
**(D)** Precedence of ‘+’ is higher than that of ‘×’, and both operators are left associative; expression is evaluated to 9

**Answer:** **(B)**

# Synthesis phase- [Code Generation and Optimization](https://www.geeksforgeeks.org/code-generation-and-optimization-gq/)

# GATE-CS-2014-(Set-3)

One of the purposes of using intermediate code in compilers is to  
**(A)** make parsing and semantic analysis simpler.  
**(B)** improve error recovery and error reporting.  
**(C)** increase the chances of reusing the machine-independent code optimizer in other compilers.  
**(D)** improve the register allocation.  
  
**Answer:** **(C)**

# GATE CS 2008

Some code optimizations are carried out on the intermediate code because  
**(A)** they enhance the portability of the compiler to other target processors  
**(B)** program analysis is more accurate on intermediate code than on machine code  
**(C)** the information from dataflow analysis cannot otherwise be used for optimization  
**(D)** the information from the front end cannot otherwise be used for optimization  
  
**Answer:** **(A)**

# GATE CS 1994

Generation of intermediate code based on an abstract machine model is useful in compilers because

1. it makes implementation of lexical analysis and syntax analysis easier
2. syntax-directed translations can be written for intermediate code generation
3. it enhances the portability of the front end of the compiler
4. it is not possible to generate code for real machines directly from high level language programs

**Answer:** **(C)**

# GATE-CS-2014-(Set-3)

Consider the basic block given below.

a = b + c

c = a + d

d = b + c

e = d - b

a = e + b

The minimum number of nodes and edges present in the DAG representation of the above basic block respectively are  
**(A)** 6 and 6

**(B)** 8 and 10

**(C)** 9 and 12

**(D)** 4 and 4

**Answer:** **(A)**

# GATE-CS-2015 (Set 2)

Consider the intermediate code given below:

1. i = 1

2. j = 1

3. t1 = 5 \* i

4. t2 = t1 + j

5. t3 = 4 \* t2

6. t4 = t3

7. a[t4] = –1

8. j = j + 1

9. if j <= 5 goto(3)

10. i = i + 1

11. if i < 5 goto(2)

The number of nodes and edges in the control-flow-graph constructed for the above code, respectively, are

**(A)** 5 and 7

**(B)** 6 and 7

**(C)** 5 and 5

**(D)** 7 and 8

**Answer:** **(B)**

# GATE CS Mock 2018

Consider the following source code :

c = a + b

d = c

c = c – e

a = d – e

b = b \* e

b = d/b

Which of the following is correct optimization of given code?

|  |  |  |  |
| --- | --- | --- | --- |
| **(A)**  c = a + b  t = b \* e  a = d – e  b = d/t  c = a | **(B)**  c = a + b  d = c  c = c – e  a = d – e  b = d/b | **(C)**  d = c  c = c – e  a = d – e  b = b \* e  b = d/b | **(D)** None of the above |

**Answer:** **(D)**

|  |  |  |
| --- | --- | --- |
|  |  |  |

# GATE-CS-2015 (Set 2)

In the context of abstract-syntax-tree (AST) and control-flow-graph (CFG), which one of the following is True?  
**(A)** In both AST and CFG, let node N2 be the successor of node N1. In the input program, the code corresponding to N2 is present after the code corresponding to N1  
**(B)** For any input program, neither AST nor CFG will contain a cycle  
**(C)** The maximum number of successors of a node in an AST and a CFG depends on the input program  
**(D)** Each node in AST and CFG corresponds to at most one statement in the input program  
  
**Answer:** **(C)**

# GATE-CS-2014-(Set-1)

Which one of the following is FALSE?  
**(A)** A basic block is a sequence of instructions where control enters the sequence at the beginning and exits at the end.  
**(B)** Available expression analysis can be used for common subexpression elimination.  
**(C)** Live variable analysis can be used for dead code elimination.  
**(D)** x = 4 ∗ 5 => x = 20 is an example of common subexpression elimination.

**Answer:** **(D)**

# GATE-CS-2006

Consider the following C code segment.

for (i = 0, i<n; i++)

{

    for (j=0; j<n; j++)

    {

        if (i%2)

        {

            x += (4\*j + 5\*i);

            y += (7 + 4\*j);

        }

    }

}

Which one of the following is false?  
**(A)** The code contains loop invariant computation   
**(B)** There is scope of common sub-expression elimination in this code  
**(C)** There is scope of strength reduction in this code  
**(D)** There is scope of dead code elimination in this code  
  
**Answer:** **(D)**

# UGC NET CS 2016 Aug – II

In compiler optimization, operator strength reduction uses mathematical identities to replace slow math operations with faster operations. Which of the following code replacements is an illustration of operator strength reduction ?  
**(A)** Replace P + P by 2 \* P or Replace 3 + 4 by 7.

**(B)** Replace P \* 32 by P < < 5  
**(C)** Replace P \* 0 by 0

**(D)** Replace (P < <4) – P by P \* 15   
  
**Answer:** **(B)**

# GATE CS 1997

The expression (a\*b)\* c op……..

where ‘op’ is one of ‘**+**‘, ‘**\***‘ and ‘**↑**‘ (exponentiation) can be evaluated on a CPU with a single register without storing the value of (a \* b) if  
**(A)** ‘op’ is ‘+’ or ‘\*’

**(B)** ‘op’ is ‘↑’ or ‘\*’  
**(C)** ‘op’ is ‘↑’ or ‘+’

**(D)** not possible to evaluate without storing  
  
**Answer:** **(A)**

# ISRO CS 2011

Which of the following statements about peephole optimization is False?  
**(A)** It is applied to a small part of the code  
**(B)** It can be used to optimize intermediate code  
**(C)** To get the best out of this, it has to be applied repeatedly  
**(D)** It can be applied to the portion of the code that is not contiguous  
  
**Answer:** **(D)**

# ISRO CS 2016

Peephole optimization is form of  
**(A)** Loop optimization

**(B)** Local optimization

**(C)** Constant folding

**(D)** Data flow analysis  
  
**Answer:** **(B)**

# GATE-CS-2017 (Set 1)

Consider the following grammar:

stmt -> **if** expr **then** **else** expr; stmt | ε

expr -> term **relop** term | term

term -> id | number

id -> **a** | **b** | **c**

number -> [**0**-**9**]

where **relop** is a relational operate (e.g < >, ….), ε refers to the empty statement, and **if**, **then**, **else** are terminals.

Consider a program P following the above grammar containing ten if terminals. The number  
of control flows paths in P is \_\_\_\_\_\_\_\_\_\_\_\_.

For example, the program **if** e1 **then** e2 **else** e3 has 2 control flow paths, e1 -> e2 and e1 -> e3

**Answer:** **1024**

# ISRO CS 2011

In compiler terminology reduction in strength means  
**(A)** Replacing run time computation by compile time computation  
**(B)** Removing loop invariant computation  
**(C)** Removing common subexpressions  
**(D)** replacing a costly operation by a relatively cheaper one  
  
**Answer:** **(D)**

# ISRO CS 2009

Substitution of values for names (whose values are constants) is done in  
**(A)** Local optimization **(B)** Loop optimization **(C)** Constant folding **(D)** Strength reduction  
  
**Answer:** **(C)**