**1 . Let language L1 denotes 3 strings &L2 denotes 4**

**strings. L1 L2=**ϕ**(no common strings) Then no of**

**strings for language L1 L2**

( a) 3

( b ) 7

( c ) 4

**( d ) 12**

**2 . Regular expression for a language which contains**

**number of a’ s are more than number of b’s**

( a) a+b\*

( b ) (a+b)\* a

( c ) a(a+b\*)

**( d ) (aba+a+aab+baa)+**

**3 . In the figure 3 represent NFA.The total number of**

**states when converted into DFA Figure3**

( a) 4

( b ) 2

( c ) 5

**( d ) 3**

**4 . In figure 4, find its , equvalent DFA**

( a) Figure 4 a Figure 4 a

( b ) Figure 4 b Figure 4 b

**( c ) Figure 4 c Figure 4c**

( d ) Figure 4 d Figure 4 d

**5 . Let N1 is the finite state M/c for Regular expression**

**R1 and N2 is the finite state machine for Regular**

**expression R2 . With respect to Union operation .**

**Statement1: Introduce new initial state and provide**

**NULL transitions for initial states of Machine**

**N1,Machine N2.**

**Statement2:The final states of Machine N2 becomes**

**final states for N1 U N2 .**

( a) Only statement 2 is TRUE

**( b ) Only statement 1 is TRUE**

( c ) Both statements are TRUE

( d ) Neither statement is correct

**6 . Which of the following statement is TRUE with**

**respect to Input buffering**

( a) The single buffer input scheme gives move

performance

( b ) The single buffer input scheme with sentinels gives

more performance

**( c ) The double buffer input scheme with sentinels**

**gives more performance**

( d ) The double buffer input scheme gives more

performance

**7 . Consider the grammer Be xpr b exprorbterm | b term**

**b term b term and b factor | bfactor b factor not b factor**

**| (b expr) | true | fabe The Non-terminals of the grammer**

( a) { o r , a n d , n ot }

( b ) { b e x p r , o r , b t e r m , ( , ) }

**( c ) { b e x p r , b t e r m , b f a c t or }**

( d ) { t r u e , f a b e , o r , a n d , n o t }

**8 . S→OSI | ISO | Λ denote a language L. Identify most**

**correct answer**

**(a) equal number of o’s and equal number of 1’s with all**

**combinations**

( b ) on 1n,n≥0

( c ) 0 n , 1 m 0 n , m , n ≥ 0

(

**9 . FIRST&FOLLOW functions are used to construct**

**parsing Table. The production A→α is to be placed in**

**all First(α) entries. If α is NULL then A→ is to be**

**placed in entries of passing Table** .

( a) FIRST (A) ∩ FOLLOW (A)

**( b ) FOLLOW(A)**

( c ) FIRST(A)

( d ) FIRST (A), FOLLOW (A)

**10. A→Bα if α is NULL the FOLLOW (B) is**

**( a) FOLLOW (A)**

( b ) FIRST (B)

( c ) FIRST (A)

( d ) FOLLOW (B)

**11. Identify the false statement**

( a) LL(K) grammer must be un-ambiguous

( b ) LL(K) grammer must be free from left recussion

( c ) All LL(K) grammers are context free Grammers

**( d ) Some LL(K) grammers does not be longs to CFG**

**12. S→ BabcDe D →e/f Trailing symbols of S,D**

( a) {e},{e}

( b ) {c,e},{e,f}

( c ) {e},{c,f}

**( d ) {e},{e,f}**

**13. Predictive Parser must free from**

( a) No restriction on the production

( b ) common prefixes & left factoring

( c ) Left recursion & left factoring

**( d ) common prefixes and left recursion**

**14. Consider the following LL(1) passing Table x y $ S**

**S→ xEy E E→ E→ YE FOLLOW(E), FIRST(S) are**

( a) {x}, {y}

( b ) {in},{x}

( c ) {x}, {in}

**( d ) {x},{x}**

**15. Which of the following passer is most power full**

( a) SLR

( b ) Operator precedence

( c ) CALR

**( d ) LALR**

**16. A→Bac**

**B→cd**

**C→e**

**Which of the following items does not belongs LR(o) I o**

**group**

( a) B → .Cd

**( b ) C→ e**

( c ) C → .e

( d ) A → .Bac

**17. Which of the following production does not belongs**

**to operator precedence grammer**

**( a) E → EAE**

( b ) E→ E\*E

( c ) E→ E+E

( d ) E→ (E)

**18. Synthesized attribute can be easily simulated by** .

**( a) LR grammer**

( b ) Ambiguous grammer

( c ) LL grammers

( d ) Regular grammer

**19. Let the folowing Anotated passe Tree 3+5 Shown in**

**Figure 19 F i g u re 19**

(a) 3

**(b) 8**

(c) 15

(d) 5

**20. The translation rule part of YACC specification is**

**enclosed between**

( a) % — — - %

( b ) % % — — - %

**( c ) % % — — - % %**

( d ) % — — - % %

**21.Let the language**

**L1={hope, fear} L2**

**={less, more} Then**

**L1 L2 denotes**

**( a) { hope less, hope more, fear less, fear more}**

( b ) {less hope , less fear, more hope, more fear}

( c ) {hope more, fear less }

( d ) {hope less, fear more}

**22 . Which of the following language is not regular over**

**an alphabet = {a,b}**

( a) Number of a’s are two

( b ) string contain sub string aba

( c ) Equal number of a’s and equal number of b’s

**( d ) strings start with a end with b**

**23 . Which of the following regular expression**

**represent identifiers if D represent regular**

**expression for digits and L denote regular expression**

**for alpha numeric characters**

( a) D(L+D)\*

( b ) L\*D\*

**( c ) DL\***

( d ) L(L+D)\*

**24 . In the figure 4 m/c represent NFA. The total number**

**of states in its equivalent DFA denote**

**NFA, what will the number of states when converted to**

**DFA F i g u re 4**

( a) 4

( b ) 3

**( c ) 5**

( d ) 2

**25 . Let N1 denotes finite state Machine for Regular**

**expression R1 while N2 denote finite state**

**machine for regular expression R2 then to get machine**

**for R1 R2 (concatenation of R1 with R2)**

**Statement1 : The final states of N1 are converted into N**

**on final states and the Null transition**

**Is introduced between final state of N1 and initial state**

**of N2**

**Statement2: The final states of N2 becomes final state**

**for R1 R2**

**(a) Both statements are TRUE**

( b ) o n l y s t at e m e nt 1 i s T RU E

( c ) N e i t h e r t h e s t at e m e nt s ar e T RU E

( d ) o n l y s t at e m e nt 2 i s T RU E

**26 . Consider a regular expression \*.0 used in Unix It**

**extracts**

( a) Al l fi l e n a m e s s ta r ti n g w i th \*

**( b ) All file names which has extension 0**

( c ) a fi l e w h os e n a m e i s \*. 0

( d ) A l l fi l e s w i t h e xt e n s i on 0

**27 . With respect to the following grammer E → E+E | E\***

**E | id identify correct statement**

( a) I t g e n e r a t e s s tr i n g s w h o s e l e n g th i s a t m

o s t t h r e e

( b ) I t g e n e r a t e s s tr i n g s w h o s e l e n g th i s a t l

e a s t t h r e e

( c ) I t g e n e r a t e s s tr i n g s o f l e n gt h t h r e e

**( d ) It is ambiguous grammer**

**28 . Which of the following grammer production is left**

**recursive**

( a) A → aA

**( b ) A → Ab**

( c ) A → aB

( d ) A → Ba

**29 . Non-recersive descent passing is free from**

( a) s t a ck ove r fl ow

**( b ) Back Tracking**

( c ) e r r or s

( d ) p a s s i n g Tab l e m e m or y

**30. The rule used in Non-recursive Predictive Passing if**

**input symbol and Top symbol of stack**

**are matched**

( a) Po p t h e t o p s y mb o l o f s t a ck w i t h ou t ad van c

i n g i n p u t s ymb o l p oi nt e r

( b ) A d va n c e i n p u t s y mb ol p o i nt e r

( c ) Po p t h e t o p s y mb o l o f s t a ck

**( d ) Pop the top symbol of stack and advance input**

**symbol pointer**

**31. S → iEtSS1 /a**

**S1 → eS/**

**E → b**

(a) l e f t re c u rs i ve gr a m m e r

( b ) L L (1 )

**( c ) Ambigous**

( d ) U n - a mb i gu o u s gr am m e r

**32. A → aBcdefG**

**B → b**

**Leading symbols of A,B are**

( a) { a , c } , { b }

( b ) { a , f } , { b }

( c ) { f } , { b }

**( d ) { a }, { b }**

**33. Which of the following Top down parsers are Table**

**driven**

( a) R e c u r s i ve d e s c e nt, Re c u r s i ve P r e d i c t i

ve p a r s e r

( b ) R e c u r s i ve d e s c e nt, B r u t e - Fou r c e p ar s er

**( c ) Non-recursive Predictive parser**

( d ) R e c u r s i ve d e s c e nt, N on - r e c u r s i ve p r e d

i c t i ve p a r s e r

**34. If the production A → α, α # NULL and a is a**

**terminal. The entry of M[A, a] is A → α iff**

( a) a F I R S T ( A )

**( b ) a FIRST (α)**

( c ) a FOL L OW ( α )

( d ) a FOL L OW ( A )

**35. LALR passing Table is constructed from**

( a) L L (0 )

( b ) L L (1 )

( c ) L R( 0 )

 **( d ) LR(1)**

**36. The dis-advantage of Panic-Mode of error recovery**

**( a) It takes more implementation complexity**

( b ) I t n e ve r ge t s i n t o i n fi n i t e l o o p s

( c ) S u i t ab l e f o r al l s i t u at i o n s

( d ) I t i s m o re e ffi c i e nt

**37. The function of following TRANSLATIONS CHEME**

**for input a+b+c**

**E → TR**

**R → + T {Print (”+”)} R, R → T → id {print (id.name)}**

( a) a+b c +

( b ) + + ab c

**( c ) ab+c+**

( d ) a b c + +

**38. The structure of YACC program contains**

( a) D e c l a r at i o n Par t , s u p p or t i n g C - Ro u ti n g

Pa r t

( b ) D e c l a r at i o n Par t , s u p p or t i n g C - Ro u ti n g

Pa r t

**( c ) Declaration Part, Translation rules Part, supporting**

**C- Routing Part**

( d ) Tra n s l a t i on Par t , s u p p o r t i n g C - R ou t i n g

Pa rt

**39. The number of states in DFA for strings over an**

**alphabet = {a, b} such that the last but one symbol of**

**the string is a**

( a) 2

( b ) 5

( c ) 4

**( d ) 3**

**40. A grammer is said be in minimal form if it is free**

**from use less productions and use less symbols. The**

**minimized grammer for the frammer S → 0 | A**

**S → AB**

**B → 1**

( a) S → 0 /A , A → A B

( b ) S → 0 , B → 1

**( c ) S → 0**

( d ) S → 0 | A

**41. The resulting grammer when common pre tixes are**

**removed using left factoring from the following**

**grammer S → ab | ad | cs**

( a) S → c s | a B | a d B → d

( b ) S → c s | B B → a b | a d

**( c ) S → aB1 | cs ,B1→ b | d**

( d ) S → B | c s | D B → a b D → a d

**42. The rule used in non-recursive predictive parsers if**

**input symbol is terminal and top of the**

**stack is Non-terminal**

**( a) Push the associated production on to stack without**

**advancing the input pointer**

( b ) N o - o p e r at i o n o n e i t h e r s ta ck a n d n o a d

va n c e m e nt o f i n p u t p oi nt e r

( c ) P u s h t h e a s s o c i a t e d p r o d u c t i on on to s t

a ck a n d a d va nc e i n p u t p oi nt e r

( d ) Po p t h e p r o d u c t i on f r om s t ack a n d a d va n c

e i n p u t p o i nt e r

**43. A → aBcdefG**

**B → b Leading symbols of A,B are**

**( a) { a }, { b }**

( b ) { a , c } , { b }

( c ) { a , f } , { b }

( d ) { f } , { b }

**44. If any entry of LL(1) passing Table contains more**

**than one Production then**

( a) G r a m m e r i s LL ( 1 )

**( b ) Grammer is ambiguous**

( c ) G r a m m e r i s LL ( 2 )

( d ) G r a m m e r i s u n amb i g uo u s

**45. In SLR passing Table if A → α. (Reduce) belongs to**

**I, then A → α is written in all places of M[I,a] where a**

( a) a FIRST (α)

( b ) a F I R S T ( A )

**( c ) a FOLLOW (A)**

( d ) a FOL L OW ( α )

**46. LR(K) denote**

( a) R i ght t o l e f t s c an n i n g t h e i n p u t, Re ve r e Ri

g ht m os t d e ri vat i o n , K - s y mbol s l

o ok ahead

( b ) R i ght t o l e f t s c an n i n g t h e i n p u t, Re ve r s e

L e f t m o s t d e ri vat i o n , K - s ymb ols L o o ka h e a d

**( c ) Left to right scanning of the input, Right most**

**derivation in reverse, K symbols Look ahead**

( d ) L e f t t o r i g ht s c an n i n g o f th e i n p u t , L e f t m

o s t derivation i n r e ve r s e , K – s ymbols Look ahea d

**47. In a syntax Directed Translation scheme if the value**

**of an attribute of a node is a function of the values of**

**attributes of its parent and children then it is called as**

( a) S y nth e s i z e d at t ri b u t e

(b ) P r o s p e c t i ve a t tr i b u t e

**( c ) Inheritted attribute**

( d ) C a n n on i c al a t t ri b u t e

**48. The TRANSLATION scheme given below**

**E → TR**

**R → + T { Print (”+”)} R1 R → T → id {print (id .name)}**

**functions**

**( a) infix to post fix**

( b ) p o s t fi x t o p o s t fi x

( c ) i n fi x t o i n fi x

( d ) p o s t fi x t o p r e fi x

49**. A finite automation M denoted by. 5 terple {Q,**

**,δ,qo,qF} the δ maps (Let p ≤Q)**

( a) Q → X 2 Q

( b ) Q X → 2 Q

( c ) Q → X p

**( d ) Q X → p**

**50. Which of the following string formed by alphabet = {**

**a,b} does not belongs to NFA shown in figure 4 F i g u**

**re 4**

( a) ab b

( b ) b b

( c ) a aa b b

**( d ) abab**

**51. The transition function of NFA δ: Q × → ?**

( a) Q Q

**( b ) ZQ**

( c ) Q

( d ) Q 2

**52. Which of the following is not the compiler analysis**

**phase**

( a) S e m a nti c A n al y s i s

( b ) L e x i c a l A n a l ys i s

 ( c ) S y nta x A n a l ys i s

**( d ) Object code Analysis**

**53. Which of the following string is not derivable from**

**the Grammer**

**S → (L) | a**

**L → L, S | S**

( a) ( a, ( a , a) ), ( a , (a , a ) ) )

( b ) ( a, a )

( c ) ( a, ( a , a) )

**( d ) ( a, )**

**54. A grammer is said to be in Greibach normal form if**

**it is free from NULL (unless S → ) and every**

**production starts with terminal. Which of the following**

**context free Grammer is in GNF form**

**( a) A → bABb**

( b ) A → B b B b

( c ) A → B a b

( d ) A → A B C

**55. The rule used in Non-recursive Predictive Passing if**

**input symbol and Top symbol of stack are matched**

( a) Po p t h e t o p s y mb o l o f s t a ck

**( b ) Pop the top symbol of stack and advance input**

**symbol pointer**

( c ) A d va n c e i n p u t s y mb ol p o i nt e r

( d ) Po p t h e t o p s y mb o l o f s t a ck w i t h ou t ad van

c i n g i n p u t s ymb o l p oi nt e r

**56. LL(1) stands**

( a) S c a n n i n g t h e i n p u t s t r i n g f r o m l e f t t o r i

g ht a n d a p p l y i n g r e ve r s e l e f t m

os

t d e r i va t i on w i t h on e i n p u t s y mb ol l o ok a h e ad

( b ) s c a n n i n g t h e i n p u t s tr i n g f r o m r i g ht to l e

f t a n d ap p l y i n g l e f t m os t d e r i

va ti

o n w i th on e i n p u t s y mb o l l o ok a h e a d

**( c ) scanning the input string from left to right and**

**applying left most derivation with one input symbol**

**look ahead**

( d ) s c a n n i n g t h e i n p u t s tr i n g f ro m l e f t to ri g

ht a n d ap p l y i n g r i g ht m os t d e r i

va t i

on w i t h on e i n p u t s y mb ol l o ok h e a d

**57. Action of LL(1) passer if Top of the stack and**

**current input symbols are same term in a l**

**which is end- marker($)**

( a) p a rs e r d e c l a r e s e rr o r

( b ) p a rs e r h a l ts d u e t o t h e e r ro r

**( c ) Successful completion parser Halts**

( d ) Po p t h e s ymb o l f r o m s t a ck

**58. The production A → is to be placed in M[A, a]**

**entries of parsing Table i ff**

( a) a FOL L OW ( )

**( b ) a FOLLOW(A)**

( c ) a F I R S T ( )

( d ) a F I R S T ( A )

**59. Relation between SLR,LALR, LR Parsers**

( a) S L R ≤ L R ≤ L A L R

( b ) L A L R ≤ L R ≤ S L R

( c ) L R ≤ L A L R ≤ S L R

**( d ) SLR ≤ LALR≤ LR**

**60. Let the grammer**

**S1 → S #**

**S → AaAb**

**S → BbBa**

**A →**

**B →**

**Identify the correct st atement as FOLLOW(A)=**

**FOLLOW(B)**

**( a) It result reduce, reduce conflict**

( b ) I t d e n ot e S LR ( 1 )

( c ) I t r e s u l t s h i f t , s h i f t c o n fl i c t

( d ) I t r e s u l t s h i f t , r e d u c e c o n fl i c t

**61. S → aABe**

**A → Abc | b**

**B → d**

**The sentence ’abbcde’ is to be reduced to s the starting**

**reduction**

( a) B → d

( b ) A → b & B → d

( c ) A → A b c

**( d ) A → b**

**62. Synthesized attribute can be easily simulated by .**

( a) R e gu l a r g ra m m e r

( b ) L L g ra m m e r s

( c ) A mb i gu o u s gr a mm e r

**( d ) LR grammer**

**63. Which of the semantic rule voilates L- attributed**

**syntax Directed Definition Production**

**Semantic Rule A → QRR. i= r(A. i)Q. i= q(R. S)A.**

**s=f(Q.S)**

**( a) Q.i=q(R.S)**

( b ) R . i = r ( A . i )

( c ) R . i = f ( Q. S )

( d ) A . S = f (Q . S )

**64. The expression a+b\* when represented in 3-**

**address code becomes (\* has higher**

**Precedence than + )**

**( a) T1= b\*c T2=T1+a**

( b ) T 1 = a T 2 = T 1 + b \*c

( c ) T 1 = a +b T 2 = T 1 \* c

( d ) T 1 = a T 1 = b T 3 = T 1 + T 2 \* c

**65. Pumping lemma is to be prove certaing languages**

**are not regular. Let string W= xyz | w | =**

**p, which belongs to language L. If string xyi z, i≥0 must**

**be long to the language if it is regular.**

**The conditions needed for dividing string W into 3 parts**

**x,y z**

( a) | y | > 1 , | x y | ≥ p

( b ) | y | < 1 , | x y | ≤ p

( c ) | y | > 0 , | x y | ≥ p

**( d ) |y | > 0, |xy|≤ p**

**66. The language L={W (0+1)\*/W ends with 00 } The**

**number of states in an associated N FA**

( a) 2

( b ) 4

( c ) 5

**( d ) 3**

**67. The number of Tokens in the following linguistic**

**statement (Ignore blanks) int a=b\*c+d ;**

**( a) 9**

( b ) 8

( c ) 1 2

 ( d ) 7

**68. Which of the following production belongs to CFG**

**(Context Free Grammer)**

( a) a b → A B

**( b ) A → aBb**

( c ) a A → b B

( d ) A a → B b

**69. The recussive descent passer is implemented with**

**stack, input buffer and parsing Table. If there are Two**

**entries for any place of parsing table then the grammer**

**is**

**( a) ambiguous not LL(1)**

( b ) U n - A mb i g u ou s & n ot LL ( 1 )

( c ) A mb i gu o u s & LL ( 1 )

( d ) U n - A mb i g u ou s & L L (1 )

**70. S1 → S$**

**S → aABe**

**A → Abc | b**

**B → d |**

**FOLLOW(S), FOLLOW(A), FOLLOW(B)**

( a) { $ } , { b } , { e }

( b ) { $ } , { d , b , } , { e }

( c ) { $ } , { d , b } , { e }

**( d ) { $ } , {d, b, e}, { e}**

**71. F1 → F$**

**F → T1E1**

**E1 → +T1E1 |**

**T1 → FT1**

**T1 →\* FT1|**

**F → (E1)|id**

**T1 → is be placed in LL(1) parsing Table entries**

( a) M [ T 1 , + ] , M [ T 1 , i d ] , M [ T 1 , $ ]

( b ) M [ T 1 , + ] , M [ T 1 , \* ] , M [ T 1 , 0 ]

**( c ) M[ T1, +], M[T1, $ ]**

( d ) M [ T 1 , + ] , M [ T 1 , \* ] , M [ T 1 , ) ]

**72. The possible actions of shift reduce - parser**

**i ) Shift (shifts the next input symbol on stack)**

**ii) Reduce(handle is reduced by associated Nonterminal**

**iii) Accept(It the end marker is reached)**

**iv) Error(Activate error recovery Procedure on error)**

( a) i , i i , i v

**( b ) i , ii , iii , iv**

( c ) i i , i i i , i v

( d ) i , i i i , i v

**73. E → (E)| E+T/T**

**T → T\*F/F**

**F → (E)/ id Which of the following belongs to LR(o) of I**

**o state**

**( a) F → .id**

( b ) F → ( . E )

( c ) E → E . + T

( d ) E → T .

**74. For which of the following situation inheritted**

**attribute is a natural choice**

( a) I m p l a c e m e nt i f c o n s t ru c t

( b ) e val u a t i on o f ar t h e m a ti c e x p r e s s i o n s

( c ) C h e ck i n g f o r c o r re c t u s e of L - val u e s

**( d ) Keeping Track of variable declaration**

**75. Which of the following language is infinite over an**

**alphabet = {0,1}**

( a) s t r i n gs w i t h l e n g t h 2 o r 3

( b ) s t r i n gs w i t h l e n g t h e q u a l t o 2

**( c ) strings with length atleast2**

( d ) s t r i n gs w i t h l e n g t h a t m o s t 2

**76. Regular expression for a language which contain**

**single 1 over an alphabet = { 0 , 1 }**

**( a) 0\*10 \***

( b ) ( 1 0 \*)

( c ) 1

( d ) ( 0 \*1 )

**77. Minimum number of states in DFA to represent the**

**regular expression ab(a+b)\***

( a) i n fi n i t e

**( b ) 3**

( c ) 4

( d ) 2

**78. The number of states for NFA which represent**

**regular expression 0\*1+0\*0**

( a) 1

( b ) 2

( c ) 4

**( d ) 3**

**79. The seperation of analysis phase of the compiling**

**into lexical analysis and parsing provides**

**( a) all the above**

( b ) I m p r ov i n g t h e e ffi c i e n c y o f c o m p i l e r

( c ) s u p p o r t s p o r t ab i l i ty

( d ) S i m p l e r D e s i gn

**80. With respect to the following grammer E → E+E | E \***

**E | id identify correct statement**

( a) I t g e n e r a t e s s tr i n g s w h o s e l e n g th i s a t m

o s t t h r e e

**( b ) It is ambiguous grammer**

( c ) I t g e n e r a t e s s tr i n g s o f l e n gt h t h r e e

( d ) I t g e n e r a t e s s tr i n g s w h o s e l e n g th i s a t l

e a s t t h r e e

**81. Which of the following grammer generates**

**Palindromes of even length over = {a, b}**

( a) S → a s b S → a b

( b ) S → b s a s → a b

( c ) S → a s a | a

**( d ) S → asa S → bsb S → Λ**

**82. A → aBb\BC**

**B → dd**

**C → e|f FOLLOW(B)**

( a) { d }

( b ) { e , f }

**( c ) { b, e, f }**

( d ) { b , d , e , f }

**83. S1 → S$**

**S → ABa | b**

**A → b| d**

**B → d The production S → ABa is to be placed in the**

**following entries of parsing Table**

**( a) M[s, b], M[s, d]**

( b ) M [ s , a ] , M [ s , d ]

( c ) M [ S , $ ] , M [ S , a ]

( d ) M [ s , # ] , M [ s , d ]

**84. Which of the following statement is not correct with**

**respect to A → w productions A → k**

**( a) FIRST (W) can be NULL**

 ( b ) F I R S T ( A ) c an b e N U L L

( c ) F I R S T ( A ) c ont ai n s F I R S T ( W )

( d ) F I R S T ( W ) c ont ai n s F I R S T ( A )

**85. Which of the following passer requires back-**

**Tracking**

**( a) Recursive Descent Passer**

( b ) R e c u r s i ve p r e d i c t i ve Pa s s e r

( c ) L L (1 ) Pa s s e r

( d ) N o n - r e c u r s i ve P re di c ti ve p as s e r

**Downloadedfromwww.strikingsoon.com**

**© jntuk.strikingsoon.com**

**86. Action of LL(1) passer if Top of the stack is a nonterminal**

**X and current input symbol is terminal, a**

( a) G e t th e c o nte nt of p as s i n g Ta b l e e nt r y M [ x,

a ] p u s h i t o n t o t h e s t a ck a n d a d

va

nc e i n p u t s ymb o l p o i nt e r

( b ) a d va n c e i n p u t s ymb o l p oi nt e r

( c ) Po p - t h e N o n - t e r m i n a l f r o m s t a ck

**( d ) Get the content of passing Table ex try M [x, a] and**

**push it on to the stack without advancing input symbol**

**pointer**

**87. The ’Handle’ of wrtto shift- reduce passers**

( a) I t i s t h e e n d m a rke r

**( b ) It is the input string matches with right hand side**

**of the production**

( c ) I t i s t h e i n p u t s t ri n g m a t ch i n g w i t h s t r i n

g N on - te rm i n a l

( d ) I t i s t h e i n p u t s t ri n g m a t ch e s w i th l e f t h a

n d s i d e o f t h e p r o d u c t i on

**88. The graph dependencies of the attributes of**

**different nodes of a pars e Tree is called as**

( a) K a r n au g h G r a p h

( b ) S t e fi G ra p h

( c ) F l ow G r a p h

**( d ) Dependency Graph**

**89. Consider the following syntax Directed Definition**

**Production Semantic Rule**

**D→ TLL. in = T. type**

**T→ int/real T. type=integer, T. type= real**

**L→ L, id L1. in= Lin, add type (id.entry, L.in)**

**L → id add type (id. entry, L. in) The nature of attributes**

**of T&L**

( a) I n h e r i t t e d , I n h e r i tt e d

**( b ) synthesized, Inheritted**

( c ) s y nt h e s i z e d , s y nth e r i z e d

( d ) I n h e r i tt e d , s y nth e r i z e d

**90. Let the Translation Rule for YACC program**

**statement is given below**

**% % E1: E1 + T{$$=$1+$3;} The $1,$3, denote**

( a) va l u e s of E 1 , T

**( b ) values of E, T**

( c ) va l u e s +, T

( d ) va l u e s of + , E 1

-

1. **Uniform symbol table**
	1. Contains all constants in program
	2. Is a permanent table of decision rules in the form of patterns for matching with the uniform symbol table

to discover syntactic structure

**c. Consists of full or partial list of tokens as they appear in program created by lexical analysis and used for syntax analysis and interpretation**

* 1. A permanent table which lists all key words and special symbols of language in symbolic form
1. **When inserted the characters of the string: KRPCSNYTJM into hash table of size 10 by using hash function H (X) = (ORD (X)-ORD ('A') + 1) mod 10 and linear probing to resolve collisions. Insertions that cause collisions are**
	1. C, S
	2. Y, M
	3. **J, M**
	4. S, T
2. **A hash function is defined as F(key) = key mod 7 with linear probing is used to insert the keys 37,38,72,48,98,11,56 into a hash table indexed from 0 to 6 .Then in which location the key 11 is stored.**
	1. 6
	2. **5**
	3. 1
	4. 4
3. **A hash function is defined as F (key) = key mod 7 with linear probing is used to insert the keys 37,38,72,48,98,11,56 into a hash table indexed from 0 to 6 .Then following key is stored as the last element**
	1. 98
	2. 56
	3. 11
	4. **48**
4. **The storage strategy in which activation record is maintained even after the execution of a procedure is completed is**
	1. Stack allocation
	2. **Heap allocation**
	3. Static allocation
	4. Dynamic allocation
5. **The average search length of the table in the linear list of 'n' records for the look up operation Requires**

**b** c d

1. **In the search trees implementation the expected time required to enter 'n' names and make 'm' enquiries is proportional to**
	1. **(n +m) log n**
	2. (m +n) log m
	3. m n log n
	4. m n log m
2. **Time is independent of number of entries in a symbol table among**
	1. Linear list
	2. Search tree
	3. **Hash table**
	4. Self-organizing lists
3. **Based on the property of locality of reference the symbol table is implemented in**
	1. Linear list
	2. search tree
	3. **hash table**
	4. self-organizing list
4. **Access time of symbol table will be logarithmic if it is implemented by**
	1. linear list
	2. **search tree**
	3. hash table
	4. self-organizing list
5. **The following symbol table implementation has the minimum access time.**
	1. **hash table**
	2. search tree
	3. linear list
	4. self-organizing list
6. **A data structure which is used to store information about various source languages constructs is**
	1. parse tree
	2. synthesized grammar
	3. **symbol table**
	4. derivation tree
7. **The names referenced frequently will be at the front among the following implementation of the symbol table**
	1. search trees
	2. hash tables
	3. **self-organizing lists**
	4. linear lists
8. **Requirement of storage for an organization using the variable length entries to the fixed length entries is**
	1. **less**
	2. more
	3. equal
	4. can't compare
9. **In linked list, to insert 'n' names and 'm' enquiries in the symbol table, the total work is**
	1. cnm
	2. m+ n
	3. **cn(n+m )**
	4. c m (n + m)
10. **On an average, the time required per operation to insert 'n' names and 'm' enquiries in the symbol table by hashing requires**
	1. c n ( n + m )
	2. c m (n + m)
	3. **constant**
	4. no
11. **The following allocation can become stack allocation by using relative address for storage in activation records**
	1. **Static**
	2. Heap
	3. Dynamic
	4. Hash
12. **The allocation strategy in which only one occurrence of each object is allowable at a given moment during program execution is**
	1. Stack
	2. **Static**
	3. Register
	4. Heap
13. **During compile time it is impossible to determine the number of times recursive procedure is going to be involved in the following allocation strategy**
	1. Stack
	2. **Static**
	3. Register
	4. Heap
14. **The allocation in which the activation record is fixed at compile time in memory is**
	1. **Static**
	2. Dynamic
	3. Heap
	4. Stack
15. **The following is not possible in static storage schema**
	1. Fixed length strings
	2. **Nested procedures**
	3. Switch statements
	4. Structures
16. **The allocation strategy in which if names are bounded, then there is no need for a runtime support package is**
	1. Stack
	2. **Static**
	3. Register
	4. Heap
17. **The allocation which manages storage for all data objects at compile time is**
	1. Dynamic
	2. **Static**
	3. Heap
	4. Stack
18. **The allocation strategy in which the size of each object must be known at compile time is**
	1. **Static**
	2. Stack
	3. Register
	4. Heap
19. **FORTAN has the following storage allocation strategy**
	1. Stack
	2. **Static**
	3. Heap
	4. Register
20. **In the following memory allocation program variables remain permanently allocated irrespective of their accessibility at any stage of programs execution.**
	1. Stack
	2. **Static**
	3. Heap
	4. Register
21. **Data structures cannot be created dynamically in the following allocation**
	1. **Static**
	2. Stack
	3. Heap
	4. Register
22. **Language that does not support dynamic allocation is**
	1. ALGOL
	2. **FORTAN**
	3. PASCAL
	4. PL/I
23. **Dynamic allocation of storage areas with VSAM files is accomplished by**
	1. Hashing
	2. **Control splits**
	3. Overflow areas
	4. Relative recording
24. **The following allocation can become stack allocation by using relative address for storage in activation records.**
	1. **Static**
	2. Heap
	3. Dynamic
	4. Hash
25. **The allocation in which the array base address is not known at compile time.**
	1. Static
	2. **Dynamic**
	3. Register
	4. Paged
26. **The storage allocation followed for Strings in SNOBOL is.**
	1. Stack
	2. **Heap**
	3. Register
	4. Static
27. **In stack allocation a new activation record is pushed into the stack for each exception of**
	1. Data base
	2. Assignment statement
	3. **Procedure**
	4. MACRO
28. **When the procedure ends in stack allocation the record is**
	1. Pushed
	2. **Popped**
	3. Peeped
	4. Displayed
29. **When there is reference to storage that has been allocated, then this may occur**
	1. **Dangling**
	2. Indexed
	3. Offset
	4. Virtual
30. **The allocation which is useful for implementing data whose size varies as the program is running is**
	1. Stacks
	2. Static
	3. **Heap**
	4. Register
31. **The following manages runtime storage as stack**
	1. Static
	2. Dynamic
	3. Heap
	4. **Stack**
32. **The allocation strategy which allocates and deallocates storage as needed at runtime from heap data area is**
	1. Static
	2. Stack
	3. **Heap**
	4. Register
33. **Heap allocation is required for languages that**
	1. support recursion
	2. **support dynamic data structure**
	3. use dynamic scope rules
	4. use dynamic programming
34. **In the following allocation words in the activation record can be accessed as offsets from the value in the register.**
	1. **Static**
	2. Dynamic
	3. Heap
	4. Stack
35. **Storage for names local to the procedure appears in**
	1. **Activation record**
	2. Symbol table
	3. Hash table
	4. Process table
36. **The call statement in the intermediate code is implemented by**
	1. MOV
	2. GOTO
	3. **MOV and GOTO**
	4. HALT
37. **The allocation which is useful for handling recursive procedure is**
	1. **Stack**
	2. Heap
	3. Static
	4. Register
38. **A record in PASCAL is defined by type rec = record**

**- - - - - - - - - - - - - - n:integer;**

**- - - - - - - - - - - - case(var1,var2) of**

**- - - - - - - - - - - - - - - - - - - - - var1:(x,y:integer);**

**- - - - - - - - - - - - - - - - - - - - - var2:(p,q:real);**

**- - - - - - - - - - - - - - end; end;**

**Suppose an array of 100 such records was declared as a machine which uses 4 bytes for an integer and 8 bytes for a real. How much space would the compiler has to reserve for an array?**

* 1. 1600
	2. 1800
	3. **2000**
	4. 2020
1. **The program fragment that follows is written in a block structured language .Assuming that it is syntactically correct and determine its output**

**begin**

**- - - - - integer x,y;**

**- - - - - - - - - - x:=3;**

**- - - - - - - - - - y:=7; begin**

* **- - - - integer x; begin**
* **- - - - integer y;**

**- - - - - - - y:=9;**

**- - - - - - - x:=2\*y; end;**

**- - - - - x:=x+y;**

**- - - - - print(x); end;**

**- - - - - print(x); end.**

a. 3 - - - 25

**b. 25 - - - 3**

c. 3 - - - 3

d. 25 - - - 25

**46. Assuming 32 bit 2's complement representation for integers and 8-bit ASCII representation for characters, how many bytes of memory are occupied by the following Pascal declaration**

**var rec: RECORD**

**- - - - - - - - - - - - F1:RECORD**

**- - - - - - - - - - - - - - - part1:RECORD**

**- - - - - - - - - - - - - - - - - - - - p1:char;**

**- - - - - - - - - - - - - - - - p2:integer;**

**- - - - - - - - - - - - - - - - END;**

**- - - - - - - - - - - - - - - - - - - - part2 :char; F2: array [1..2] of RECORD**

**- - - - - - - - - - - - - - - - - - - - x1: integer;**

**- - - - - - - - - - - - - - - - - - - - x2: array [1..3] of char;**

**- - - - - - - - - - - - - END;**

**- - - - - END;**

**Total allocated memory is** a. 10

**b. 20**

* 1. 30
	2. 40
1. **struct**

**{**

**short s[5]; union{ float y; long z;**

**} u; }]t;**

**Assume short, float, long occupy 2 bytes,4 bytes,8 bytes respectively .The storage for variable 't' ignoring alignment is**

* 1. 22 bytes
	2. 14 bytes
	3. **18 bytes**
	4. 10 bytes
1. **The output of following program in a language that has dynamic scoping is: var x:real;**

**procedure s1; begin**

**, , , , , , , , , , , , , , print(x); end;**

**procedure s2; var x:real; begin x:=0.125; s1;**

**end; begin x:=0.25; s1;**

**s2; end;**

* 1. 0.125 - - 0.125
	2. **0.25 - - 0.125**
	3. 0.25 - - 0.25
	4. 0.125 - - 0.25
1. **The program fragment that follows is written in a block structured language Assuming that it is syntactically correct and determine its output**

**union{ struct**

**{**

**char a,b,c,d; }s1;**

**struct**

**{**

**int i ,j; short s;**

**} s2; long z; float f; double m; } u;**

**The size of u is**

* 1. 8 bytes
	2. **4 bytes**
	3. 2 bytes
	4. 16 bytes
1. **If the language has dynamic scoping and parameters are passed by reference, then what will be printed by the following?**

**p( )**

**var n:int;**

**procedure w (var x:int)**

**begin x=x+1; print(x); end;**

**procedure D( ) begin**

**var x:int; x=3; w(n); end;**

**begin /\* main program\*/ n=10;**

**D; end.**

* 1. 10
	2. 11
	3. 3
	4. **4**
1. **Consider the following variant record declaration in Pascal Type abc =RECORD**
	* + **- - - - - - - - - - - - - - - - - x:integer**
		+ **- - - - - - - - - - - - - - - - - - - - - - - case(y:integer) of**
		+ **- - - - - - - - - - - - - - - - - - - - - - - - - - 1:(m:integer,n:real);**
		+ **- - - - - - - - - - - - - - - - - - - - - - - - - - 2:(e,f:integer); END;**

**suppose a program uses an array of p such records .Integer needs 2 bytes of storage and real**

**needs 4 bytes of storage. If an array occupies 480 bytes, the value of p is** a.10

b.64

* 1. **60**
	2. 80
1. **A region of validity every name possesses in the source program is**
	1. **Scope**
	2. Life time
	3. Binding
	4. Domain
2. **A following for the procedure is a number that is obtained by standing with a value of one for the main and adding one to it every time we go from an enclosing to enclosed procedure.**
	1. Nesting loop
	2. **Nesting depth**
	3. line numbering
	4. Recursion
3. **Consider the following block of code p( )**

**begin x=10, y=3;**

**func (y,x,x); print (x,y); end;**

**func (x,y,z) begin y=y+4; z=x+y+z; end;**

**If parameters are passed by reference ,the output of above procedure is**

* 1. **31 - - 3**
	2. 3 - - 31
	3. 3 - - 3
	4. 31 - - 31
1. **In analyzing the compilation of a program 'machine independent optimization' is associated with**
	1. Recognition of basic syntactic construction through reduction
	2. Recognition of basic elements and creation of uniform symbols
	3. **Creation of more optimal matrix**
	4. Use of macro processor to produce more optimal assembly code
2. **The following is used as the key to accessing the scope information from the symbol table**
	1. Procedure name
	2. **Procedure name, nesting depth**
	3. Nesting depth
	4. Usage count
3. **The following of a procedure is a number that is obtained by standing with a value of one for the main and adding one to it every time we go from an enclosing to enclosed procedure**
	1. Nesting loop
	2. **Nesting depth**
	3. Nesting breadth
	4. Recursion
4. **Generation of intermediate code based on a 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 front end of compiler
	4. It is not possible to generate code for real machines directly from high level Language programs
5. **An optimized compiler**
	1. Is optimized to occupy less space
	2. Is optimized to take less time for execution
	3. **Optimized the code**
	4. Optimized to small typing font
6. **The following refers to the techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program is**
	1. Code generation
	2. **Code optimization**
	3. Code execution
	4. Code debugging
7. **The "90-10" rule states that**
	1. 90 % of code is executed in 10 % of time
	2. **90 % of time is spent in 10 % of code**
	3. 90 % of time is spent in correcting 10 % of errors
	4. 10 % of time is spent in correcting 90 % of errors
8. **The most heavily travelled parts of programs are**
	1. **Inner loops**
	2. Constants
	3. Static variables
	4. Global variables
9. **Machine independent code optimization can be applied to**
	1. Source code
	2. **Intermediate representation**
	3. Object code
	4. Run-time output
10. **At a point in a program if the value of variable can be used subsequently, then that variable is**
	1. **Live**
	2. Dead
	3. Duplicate
	4. Aliasing
11. **The region of validity every name possesses in the source program is**
	1. **Scope**
	2. Life time
	3. Binding
	4. Domain
12. **Which of the following is not a peephole optimization?**
	1. Removal of unreachable code
	2. Elimination of multiple groups
	3. **Elimination of loop invariant compilation**
	4. Loop unrolling
13. **At point p if no matter what path is taken from p, the expression e will be evaluated before any of its operands are defined, then expression e is said to be**

a.Lazy b.Late

* 1. **Busy**
	2. Dummy
1. **The evaluation which avoids the action at all which is clearly advantageous**
	1. Lat
	2. **Lazy**
	3. Critical
	4. Smart
2. **Which of the following comments about peep hole optimization are false.**
	1. It is applied to small part of the code
	2. It can be used to optimize intermediate code
	3. To get the best out of this, it has to be applied repeatedly
	4. **It can be applied to the code that is contiguous**
3. **The following examines short sequences of code and determines a sequence can be replaced by a shorter equivalent sequence.**
	1. **Peep hole optimizer**
	2. Assembler
	3. Linker
	4. Loader
4. **The evaluation ordering in which we know beforehand that will have to perform the action anyway, but we find it advantageous to perform it as late as possible**
	1. **Late**
	2. Lazy
	3. Critical
	4. Smart
5. **The evaluation ordering in which, code for node is issued as soon as the code for all of its Operands has been issued**
	1. **Early**
	2. Late
	3. Lazy
	4. Critical
6. **Peep-hole optimization is form of**
	1. **Local optimization**
	2. Constant folding
	3. Copy propagation
	4. Data flow analysis
7. **If the transformation of a program can be performed by looking only at statements in basic block then it is said to be**
	1. **Local**
	2. Global
	3. Algebraic
	4. Matrix
8. **If E was previously computed and the values of variable in E have not changed since previous computation ,then an occurrence of an expression E is**
	1. Copy propagation
	2. Dead code
	3. **Common sub expressions**
	4. Constant folding
9. **In block B if s occurs in B and there is no subsequent assignment to y within B, then the copy statement s:x :=y is**
	1. **Generated**
	2. Killed
	3. Blocked
		1. Dead
10. **In block B if x or y is assigned there and s is not in B then s: x :=y is**
	1. Generated
	2. **Killed**
	3. Blocked
	4. Dead
11. **If two or more expressions denote same memory address, then expressions are**
	1. **Aliases**
	2. Definitions
	3. Superior
	4. Inferior
12. **Operations that can be removed completely are called**
	1. Strength reduction
	2. **Null sequences**
	3. Arithmetic simplification
	4. Constant folding
13. **Given the following code X=A+B;**

**Y=A+B;**

**And the corresponding optimized code as**

**- - - - - - - - - - Z=A+B;**

**- - - - - - - - - - X=Z;**

**- - - - - - - - - - 100 Y=Z;**

**When will be optimized code pose a problem?**

* 1. **Z may not remain same after the control reaches here first time since it is labeled**
	2. When Z is undefined
	3. When memory is consideration
	4. Doesn't pose a problem
1. **Can the loop invariant X = A - B from the following code be moved out For I = 1 to 10**

**A = B \* C; X = A - B;**

* 1. **Yes**
	2. No
	3. X=A-B is not invariant
	4. Wrong code
1. **The following transformation takes an expression that yields the same result independent of the number of times.**
	1. Reduction in strength
	2. Induction variable
	3. Constant folding
	4. **Code motion**
2. **The optimization that avoids a test at each iteration is**
	1. **Loop unrolling**
	2. Loop ramming
	3. Loop nesting
	4. Loop scrolling
3. **The method that merges the bodies of two loops is**
	1. Loop nesting
	2. Constant folding

**c.Loop ramming**

* 1. Loop unrolling
1. **The code for repetition can be optimized by**
	1. **Loop unrolling**
	2. Loop jamming
	3. Nested loop
	4. Code motion
2. **The following can not be used to identify loops**
	1. Depth first ordering
	2. Reducible graphs
	3. Dominators
	4. **Flow chart**
3. **Replacement of a string concatenation operator "||" by an addition is example for**
	1. **Strength reduction**
	2. Operator overriding
	3. Operator over loading
	4. Operator elimination
4. **The replacement of an expensive operation by a cheaper one**
	1. Operator overriding
	2. Operator overloading
	3. **Reduction in strength**
	4. Operator elimination
5. **The loop which does not contain any other loops are**
	1. Natural loop
	2. **Inner loop**
	3. Main loop
	4. Nested loop
6. **Identify the basic blocks in the following code**

**10 goto 20**

**20 goto 10**

* 1. Go to 20
	2. Go to 10
	3. **Two independent basic blocks**
	4. Both statements in one block
1. **Loop is collection of nodes that**
	1. Is loosely connected
	2. Is strongly connected with several entries
	3. Is having several entries
	4. **Is strongly connected with unique entry**
2. **In loop optimization technique a loop whose body is rarely executed is**
	1. Dead code
	2. Blank code
	3. Redundant code
	4. **Blank stripper**
3. **If the following optimization is attempted, count of number of jumps to each label can be found from symbol table of that label**
	1. Back tracking
	2. **Peephole optimization**
	3. Dynamic programming
	4. Global optimization
4. **The use of the following greatly improves the code when pushing or popping a stack, as in parameter passing**
	1. Flow of control
	2. Auto increment addressing modes
	3. Auto decrement addressing modes
	4. **Both auto increment and auto decrement addressing modes**
5. **The following can be done through peephole optimization**
	1. Dynamic programming
	2. Greedy method
	3. **Unreachable code clinination**
	4. Code generation
6. **The following algebraic identities can be eliminated through peephole optimization**
	1. x:=x+0
	2. x:=x\*1
	3. **x:=x+0 and x:=x\*1**
		1. Jumps over jumps
	4. **Replacing the exponent operator by the shift of multiplication is**
		1. Constant folding
		2. **Reduction in strength**
		3. Copy propagation
		4. Elimination of dead variables
	5. **Method to improve the target program by examining a short sequence of target instructions**
		1. Back tracking
		2. **Peephole optimization**
		3. Dynamic programming
		4. Global optimization
	6. **Program transformation that is not characteristic of peephole optimization**
		1. Redundant instruction elimination
		2. Flow of control optimization
		3. **Back patching**
		4. Use of machine idioms
7. **The following is the small moving window in target program**
	* 1. Screen saver
		2. Graphics
		3. **Peephole**
		4. Greedy window
8. **Use of machine idioms is one of the characteristic of**
	* 1. Back tracking
		2. **Peephole optimization**
		3. Dynamic programming
		4. Global optimization
9. **Peephole optimization is to eliminate**
	* 1. Constant folding
		2. **Jumps over jumps**
		3. Copy propagation
		4. Elimination of dead variables
10. **Machine independent optimization is**
	* 1. Register allocation
		2. Frequency reduction
		3. **Data intermixed with instructions**
		4. Machine features instructions
11. **Computing two or more identical computations to a place in the program where the computation can be done once and the result to used in the entire original places is**
	* 1. Replacing
		2. **Profiling**
		3. Hoisting
		4. Rearranging
12. **Deducing at compile time that the value of an expression is a constant and using that constant**

**instead is**

* 1. **Constant folding**
	2. Constant deduction
	3. Macro substitution
	4. Copy propagation
1. **The following computation is done and placed the expression before the loop in the transformation of code motion**
	1. **Loop-invariant**
	2. Overloading
	3. Overriding
	4. Constant folding
2. **Changing the form to preserve the program semantics is**
	1. Coding
	2. **Frequency reduction**
	3. Constant folding
	4. Constant substitution
3. **Replacing expressions by their value if the value can be computed at compile time is called**
	1. Constant substitution
	2. **Constant folding**
	3. Macro
	4. Infix expression
4. **Replacing the expression 2\*3.14 by 6.28 is**
	1. **Constant folding**
	2. Induction variable
	3. Strength reduction
	4. Code reduction
5. **An estimate of how frequently a variable used is**
	1. **Usage count**
	2. Reference count
	3. Program count
	4. Process count
6. **The modification that decreases the amount of code in a loop is**
	1. Constant folding
	2. Constant reduction
	3. **Code motion**
	4. Reduction strength
7. **Movement of the code from inside to outside is**
	1. Coding
	2. **Frequency reduction**
	3. Constant folding
	4. Constant substitution
8. **In DAG the interior nodes are labeled with**
	1. Numbers in DFS
	2. Numbers in BFS
	3. **Identifiers**
	4. Special colors
9. **Sorting techniques used for evaluation of interior nodes of DAG in any order is**
	1. Quick sort
	2. Selection sort
	3. Merge sort
	4. **Topological sort**
10. **If every path from the initial node goes through that particular node, then the node is said**
	1. Header
	2. **Dominator**
	3. Parent
	4. Descendant
11. **DAG has**
	1. Only one root
	2. **Any number of roots**
	3. No root
	4. No nodes at all
12. **A basic block can be analyzed by**
	1. Flow graph
	2. Flow chart
	3. **DAG**
	4. RAG
13. **In DAG, interior nodes are labeled by**
	1. **Operator symbol**
	2. Unique identifiers
	3. Operands
	4. Node numbers
14. **Common sub expressions can be detected automatically by using**
	1. Flow graph
	2. Flow chart
	3. **DAG**
	4. RAG
15. **Date structures useful for implementing transformations on basic blocks are**
	1. RAG
	2. **DAG**
	3. Gaunt chart
	4. B+ tree
16. **DAG representation of a basic block allows**
	* 1. **Automatic detection of local common sub expression**
		2. Automatic detection of induction variables
		3. Automatic detection of loop invariables
		4. Automatic motion to code
	1. **In DAG, labeling of nodes are**
		1. Mandatory
		2. **Optional**
		3. Compulsory
		4. Automatically generated
17. **The statement that may be safely removed without changing the value of basic block is**
	* 1. Live code
		2. **Dead code**
		3. Temporary variables
		4. Common sub-expression
18. **The block which can be transformed into an equivalent block in which each statement that defines a temporary defines a new temporary is**
	* 1. **Normal form**
		2. Temporary block

c.Procedural block

* 1. Null block
1. **The following basic block permits all statement interchanges that are possible**
	1. **Normal form**
	2. Dead block
	3. Temporary block
	4. Null block
2. **Determine the number of definition and reference points of variable A in the following code A = Z; /\* statement S0\*/**

**A = B; /\* statement S1\*/**

**C = D; /\* statement S2\*/ C = C + A;**

**a. 1,3** b.2,3

* 1. 0,4
	2. Can't be determined
1. **Determine the definition points d; affecting the expression C = C+A In the following code**

**A=Z; /\* statement S0\*/ A=B; /\* statement S1\*/ C=D; /\* statement S2\*/ C=C+A;**

* 1. Statement S0 and S1
	2. **Statement S1 and S2**
	3. Statement S0 and S2
	4. Undetermined
1. **Structure preserving transformations on basic blocks are**
	1. **Dead code elimination**
	2. Strength reduction
	3. Copy propagation
	4. Constant folding
2. **In reducible flow graph, the following edges consists only of edges whose heads dominate their tails.**
	1. Forward edges
	2. **Back edges**
	3. Pre-header
	4. Header
3. **Determine the pre-dominant block of block B2 in the program flow graph from the following code**

**../\* Block B0\*/**

**10 go to 100 /\* Block B1\*/**

**100 go to 10 /\* Block B2\*/**

* 1. B0
	2. B1
	3. B2
	4. **Code insufficient**
1. **A sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end is**
	1. Flow graph
	2. DAG
	3. **Basic block**
	4. Loop
2. **The first statement in basic block is**
	1. **Leader**
	2. Header
	3. Main statement
	4. Follow
3. **Any statement that immediately follows the following statement is a leader**
	1. First statement
	2. **GOTO or conditional GOTO**
	3. While loop
	4. Array declaration
4. **The following among these is not structure preserving transformations on basic blocks**
	1. Common sub-expression elimination
	2. **Invariant variables elimination**
	3. Dead code elimination
	4. Interchanging of two independent adjacent statements
5. **Determine the path(s) in the program flow graph for the code**

**10 goto 100 /\* Block B1\*/**

**100 goto 10 /\* Block B2\*/**

* 1. (B1, B2)
	2. (B2,B1)
	3. **(B1,B2,B1)**
	4. No path
1. **The graph that shows the basic blocks and their success of relationship is called**
	1. DAG
	2. **Flow graph**
	3. Control graph
	4. Hamilton graph
2. **Which of the following class of statement usually produces no executable code when compiled.**
	1. Declaration
	2. Assignment
	3. Input and output
	4. **Structural statements**
3. **Edges in the flow graph whose heads dominate their tails are called**
	1. **Back edges**
	2. Front edges
	3. Flow edges
	4. Headedges