QUESTIONS FROM UNIVERSITY PAPER

DEPARTMENT OF COMPUTER SCIENCE

COMPILER DESIGN

MODULE I

1. State the role of lexical analyzer. Identify the lexemes and their corresponding tokens in the following statement:
   printf (“Simple Interest=%f\n”, si);
2. Distinguish between front end and back end of a compiler.
3. Specify the analysis and synthesis parts of compilation.
4. Define the terms token, lexemes and patterns with examples
5. Explain the different phases of a compiler. Illustrate with a source language program statement
6. List and explain any three compiler construction tools.
7. Explain in detail the various phases of a compiler with a neat diagram.
8. Illustrate the output of each phase for the input x = 2 * a + b, where a and b are float variables.
9. Demonstrate bootstrapping
10. Explain any three tools that help a programmer in building a compiler efficiently.
11. Explain the different phases of a compiler with a running example.
12. List and explain any three compiler construction tools.
13. What is a regular definition? Give the regular definition of an unsigned integer.
14. Express the role of transition diagrams in recognition of tokens.
15. Trace the output after each phase of the compiler for the assignment statement:
    a = b + c * 10, if variables given are of float type.

16.Scanning of source code in compilers can be speeded up using input buffering. Explain.

17.Explain the working of different phases of a compiler. Illustrate with a source language statement.

18. Explain how the regular expressions and finite state automata are used for the specification and recognition of tokens?

19.Describe input buffering scheme in lexical analyzer

APRIL 2018

1. Draw the transition diagram for the regular definition
   relop →< | <= | = | <>|>
2. With an example source language statement, explain tokens, lexemes and patterns

3. A. Apply bootstrapping to develop a compiler for a new high level language P

on machine N

B. Now I have a compiler for P on machine N. Apply bootstrapping to obtain acompiler for P on machine M

4. Define cross compilers
5. For a source language statement a=b*c-2, where a,b and c are float variables, * and – represents multiplication and subtraction on the same data types,show the input and output at each of the compiler phases

MAY 2019

6. Describe input buffering scheme in lexical analyzer
7. Develop a lexical analyzer for the token identifier
8. Explain any four compiler writing tools (Repeated in December 2019)

DECEMBER 2019

9. Scanning of source code in compilers can be speeded up using input buffering. Explain
10. Explain how the regular expressions and finite state automata can be usedfor the specification and recognition of tokens
11. Explain the different phases of a compiler. Illustrate with a source language program statement (Repeated in May 2019)

MODULE II

1.What is left recursive grammar? Give an example. What are the steps in removing left recursion?

2. Eliminate the ambiguity from the given grammar

E → E*E | E-E | E^E | E/E | E+E | (E) | id.

The associativity of the operators is as given below. The operators are listed in the decreasing order of precedence.

(i) ( )

(ii) / and + are right associative

(iii) ^ is left associative.

(iv) * and – are left associative

3. Is the grammar S --> S | (S) S / Ɛ ambiguous? Illustrate your answer.

4. Recall backtracking with an example.

5.Is the grammar S --> S | (S) S / Ɛ ambiguous? Justify your answer.

6. What is left recursive grammar? Give an example. What are the steps in removing left recursion?

7. Identify if following grammar is LL(1) by constructing a parse table:

S → (L) | a

L→ S L’

L’ → , S L’ | ε

Note that ‘,’ is a terminal and ε is the empty string.

8. Find the FIRST and FOLLOW of the non-terminals S, A and B in the grammar :

S→aABe

A→Abc | b

B→d

9.Consider the following grammar

E-→E or T | T

T→ T and F |F

F→not F | (E) | true | false

(i) Remove left recursion from the grammar.

(ii) Construct a predictive parsing table.

(iii) Justify the statement “The grammar is LL (1)”

10. What is Recursive Descent parsing? List the challenges in designing such a parser?

APRIL 2018

1. Define LL(1) grammars.
2. Is the grammar S->S(S)S/ Ɛ ambiguous? Justify your answer
3. Consider the following

   grammarE-> E or T | T

   T-> T and F |F

   F-> not F | (E) | true | false

   (i) Remove left recursion from the grammar.

   (ii) Construct a predictive parsing table.

   (iii) Justify the statement “ The grammar is LL (1)”.
4. Design a recursive descent parser for the grammar

   S->cAd, A->ab/ b
5. Compute the FIRST and FOLLOW for the following Grammar

   S->Bb/Cd

   B->aB/Ɛ

   C->cC/Ɛ

MAY 2019

6. Consider the context free grammar

   S->aSbS | bSaS | €

   Check whether the grammar is ambiguous or not
7. What is Recursive Descent parsing? List the problems faced in designing such a parser.
8. Find the FIRST and FOLLOW of the non-terminals in the grammar

   S->aABe A->Abc|b B->d
9. Design a recursive descent parser for the grammar

   E->E + T | T

   T->T*F | F

   F->(E) | id
10. What is left recursive grammar? Give an example. What are the steps in removing left recursion?

DECEMBER 2019

11. Differentiate leftmost derivation and rightmost derivation. Show an example for each.

12. Find out context free language for the grammar given below:

    S -> abB A -> aaBb | ε B -> bbAa

13. Given a grammar :
    S-> (L)|a L-> L,S | S
    (i) Is the grammar ambiguous? Justify
    (ii) give the parse tree for the string (a,((a,a), (a,a)))

14. Construct the predictive parsing table for the following grammar:

    S -> (L) | a L -> L,S | S

15. Can recursive descent parsers used for left recursive grammars? Justify your answer. Give the steps in elimination of left recursion in a grammar

MODULE III

1. Find the LR(0) items for the grammar S->SS | a| €.

2.What is handle pruning? Find the handles in the reduction of the right sentential form

S S+ a * to the start symbol using the grammar below:

S→ S S+ | S S * | a

3. Left factor the following grammar and then obtain LL(1) parsing table

E → T+E | T

T → float | float * T | (E)

4. Construct canonical LR(0) collection of items for the grammar below.

S -> L = R

S -> R

L -> * R

L -> id

R -> L

Also identify a shift reduce conflict in the LR(0) collection constructed above

5. Show that the input string id1 *id2 is accepted by the Shift reduce parser for the following grammar

E→ E + T | T

T→ T * F | F

F→ (E) | id

6.Compare different bottom-up parsing techniques.

7. What are the possible actions of a shift reduce parser.

8.(a) Construct the LR(0) set of items and their GOTO function for the grammar

S→S S + | S S * | a

(b) Is the grammar SLR? Justify your answer

9.(a) Identify LR(1) items for the grammar

S→ CC

C→ cC | d

(b) Construct LALR table for the above grammar

10. Explain operator grammar and operator precedence parsing with example

11. Explain viable prefixes? For the given grammar S → 0 S 1 |0 1

write all the viable prefixes for the string 00001111

APRIL 2018

1. Demonstrate the identification of handles in operator precedence parsing?
2. Construct canonical LR(0) collection of items for the grammar below.

   S -> L = R
   S -> L = R

   S -> R

   L -> * R

   L -> id

   R -> L

   Also identify a shift reduce conflict in the LR(0) collection constructed above

3. Construct LALR parse table for the grammar S->CC,C->cC|d

MAY 2019

4. Explain the main limitaitions in a shift reduce parser
5. What are different parsing conflicts in SLR parsing table?
6. Find the LR(0) items for the grammarS->SS | a| €.
7. Derive LALR (1) parsing algorithm for following grammarS→AS/b A→SA/a
8. Explain operator grammar and operator precedence parsing

DECEMBER 2019

9. Compute FIRST and FOLLOW for the grammar:
   S -> SS+ | SS* | a
10. Write the algorithm to construct LR(1) collection for a grammar.
11. Write algorithm for SLR paring table construction
12. Construct the SLR table for the grammar:
    S -> aSbS | a
13. Differentiate CLR and LALR parsers.

MODULE IV

1.Design a Syntax Directed Definition for a Desk calculator that prints the result.

2 What are annotated parse trees? Give examples

3. With an SDD for a desk calculator, give the appropriate code to be executed at each reduction in the LR parser designed for the calculator.

4.Also give the annotated parse tree for the expression (3*5) –2.

5. Give the annotated parse tree for the expression: 1*2*3*(4+5) n

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

symbol

E → E1 # T {E.value=E1.value x 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. lvalue ;}

Compute E.value for the root of the parse tree for the expression 2#3 & 5# 6 &7

7 Explain bottom- up evaluation of S- attributed definitions.

8.Write Syntax Directed Translator (SDT) and parse tree for infix to postfix translation of an expression.

9.Explain the storage allocation strategies.

10. Design a Syntax Directed Translator(SDT) for the arithmetic expression

(4 * 7 + 19) * 2 and draw an annotated parse tree for the same

11. Differentiate synthesized and inherited attributes with examples.

12. Translate a[i] = b * c – b * d, to quadruple

13. Construct the DAG and three address code for the expression a+a*(b-c)+(b-c)*d

MODULE V

1. Construct the optimization of basic blocks with examples.

2. Generate target code sequence for the following statement

   d := (a-b)+(a-c)+(a-c).

3. Construct the syntax tree and then draw the DAG for the statement

   e := (a*b) + (c-d) * (a*b)

4.Explain the code generation algorithm. Illustrate with an example

5.Construct the DAG and three address code for the expression

   a+a*(b-c)+(b-c)*d

6.Describe the principal sources of optimization

7.Illustrate the optimization of basic blocks with examples.

8.Write the Code Generation Algorithm and explain the getreg function

9. What is the role of peephole optimization in the compilation process

10. What are the issues in the design of a code generator

MODULE 4&5

APRIL 2018

1. Write syntax directed definitions to construct syntax tree and three address code for assignment statements.
2. Explain quadruples and triples with an example each.
3. Construct the syntax tree and then draw the DAG for the statement
   e := (a*b) + (c-d) *(a*b)
4. Explain static allocation and heap allocation strategies.
5. With an example each explain the following loop optimization techniques: (i) Codemotion (ii) Induction variable elimination and (iii) strength reduction
6. Explain any two issues in the design of a code generator.
7. Explain the optimization of basic blocks.
8. Write the Code Generation Algorithm and explain the *getreg* function.
9. Generate a code sequence for the assignment d=(a-b)+(a-c)+(a-c)

MAY 2019

10. Explain storage organization and storage allocation strategies
11. Explain intermediate code generation of an assignment statement
12. Explain quadruples, triples and dags with an example each

13. Explain the principal sources of optimization
14. Explain optimization of basic blocks
15. With suitable examples explain loop optimization
16. Explain issues in design of a code generator
17. Explain simple code generation algorithm

DECEMBER 2019

18. Explain how DAGs help in intermediate code generation?
19. Explain the code generation algorithm. Illustrate with an example
20. Define the following and show an example for each.
    i). Three-address code iii). Triples ii). Quadruples iv). Indirect triples
21. State the issues in design of a code generator
22. Explain different stack allocation strategies with suitable examples.
23. Explain different code optimization techniques available in local and globaloptimizations?
24. How is storage organization and management done during runtime?
25. How the optimization of basic blocks is done by a compiler?
26. Write the algorithm for partitioning a sequence of three-address instructions into basic blocks