DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THIRD YEAR (SIXTH SEMESTER)

CS8602- COMPILER DESIGN

UNIT I –Introduction To compilers

1.What is a Translator?

Translator is a program which converts a program written in any source Language into any other Destination language. The source and destination languages may be any High Level Language or Low Level Languages like assembly language or machine language.

2. What is a Complier?

A Complier is a program that reads a program written in one language-the source language- and translates it in to an equivalent program in another language-the target language . As an important part of this translation process, the compiler reports to its user the presence of errors in the source program

Source program→compiler→ target program
(Error messages)

3. State some software tools that manipulate source program ?

i. Structure editors
ii. Pretty printers
iii. Static checkers
iv. Interpreters.

4. What are the cousins of compiler?

The following are the cousins of compilers
i. Preprocessors
ii. Assemblers
iii. Loaders
iv. Link editors.

5. What are the main two parts of compilation? What are they performing?

The two main parts are
•Analysis -part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.
•Synthesis-part constructs the desired target program from the intermediate representation

6. What is a Structure editor?

A structure editor takes as input a sequence of commands to build a source program .The structure editor not only performs the text creation and modification functions of an ordinary text Compiler . editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.

7. What are a Pretty Printer and Static Checker?

•A Pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visible.
•A static checker reads a program, analyses it and attempts to discover potential bugs with out running the program.

8. How many phases does analysis consists?

Analysis consists of three phases

i .Linear analysis
ii .Hierarchical analysis
iii. Semantic analysis

9. What happens in linear analysis?

This is the phase in which the stream of characters making up the source program is read from left to right and grouped in to tokens that are sequences of characters having collective meaning.

10. What happens in Hierarchical analysis?

This is the phase in which characters or tokens are grouped hierarchically in to nested collections with collective meaning.

11. What happens in Semantic analysis?

This is the phase in which certain checks are performed to ensure that the components of a program fit together meaningfully.

12. State some compiler construction tools?

i. Parse generator
ii. Scanner generators
iii. Syntax-directed translation engines
iv. Automatic code generator
v. Data flow engines.

13. What is a Loader? What does the loading process do?

A Loader is a program that performs the two functions

i. Loading
ii .Link editing

The process of loading consists of taking relocatable machine code, altering the reloadable address and placing the altered instructions and data in memory at the proper locations.

14. What does the Link Editing does?

Link editing:

This allows us to make a single program from several files of reloadable machine code. These files may have been the result of several compilations, and one or more may be library files of routines provided by the system and available to any program that needs them.

15. What is a preprocessor?

A preprocessor is one, which produces input to compilers. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a distinct program called a preprocessor. The preprocessor may also expand macros into source language statements.

Skeletal source program

Source program

16. State some functions of Preprocessors
    i)Macro processing
    ii)File inclusion
    iii)Relational Preprocessors
    iv)Language extensions

17. What is a Symbol table?
    A Symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

18. State the general phases of a compiler
    i)Lexical analysis
    ii)Syntax analysis
    iii)Semantic analysis
    iv)Intermediate code generation
    v)Code optimization
    vi)Code generation

19. What is an assembler?
    Assembler is a program, which converts the source language in to assembly language.

20. What is the need for separating the analysis phase into lexical analysis and parsing?
(Or) What are the issues of lexical analyzer?
    •Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases.
    •Compiler efficiency is improved.
    •Compiler portability is enhanced.

PART –B

1. What is a compiler? State various phases of a compiler and explain them in detail.
2. Explain the various phases of a compiler in detail. Also write down the output for the following expression after each phase a:= b*c-d.
3. What are the cousins of a Compiler? Explain them in detail.
4. Describe how various phases could be combined as a pass in a compiler? Also briefly explain Compiler construction tools .
5. For the following expression Position:=initial+ rate*60 Write down the output after each phase .

UNIT II-Lexical Analyser

1. What is Error Handler?

   The error handler invokes error diagnostic routines and also generates error messages. It works in conjunction with all the phases to handle the errors at the respective phases.

2. Define Tokens.

   The token can be defined as a meaningful group of characters over the character set of the programming language like identifiers, keywords, constants and others.

3. Define Symbol Table.

   A Symbol Table is a data structure containing a record for each identifier, with fields for the attributes of the identifier.

4. Define lexeme?

   The character sequence forming a token is called lexeme for the token.

5. Write a short note on LEX.

   A LEX source program is a specification of lexical analyzer consisting of set of regular expressions together with an action for each regular expression. The action is a piece of code, which is to be executed whenever a token specified by the corresponding regular expression is recognized. The output of a LEX is a lexical analyzer program constructed from the LEX source specification.

6. What is the role of lexical analysis phase?

   Lexical analyzer reads the source program one character at a time, and grouped into a sequence of atomic units called tokens. Identifiers, keywords, constants, operators and punctuation symbols such as commas, parenthesis, are typical tokens.

7. Mention the role of semantic analysis?

   - Semantic analysis checks the source program for semantic errors and gathers information for the subsequent code-generation phase.
   - It uses hierarchical structure to identify the operators and operands of expressions and statements.
   - An important component of semantic analysis is type checking .In type checking the compiler checks that each operator has operands that are permitted by the source language specification.
   - In such cases, certain programming language supports operand coercion or type coercion also.

8. Name some variety of intermediate forms.

   - Postfix notation or polish notation.
   - Syntax tree
   - Three address code
   - Quadruple
   - Triple

9. What are the techniques behind code optimization phase?

- Constant folding
- Loop constant code motion
- Induction variable elimination
- Common sub expression elimination
- Strength reduction
- Mathematical identities
-

10. Write the syntax for three-address code statement, and mention its properties.

Syntax: A= B op C

- Three-address instruction has at most one operator in addition to the assignment symbol. The compiler has to decide the order in which operations are to be done.
- The compiler must generate a temporary name to hold the value computed by each instruction.
- Some three-address instructions can have less than three operands.
-

11. What is a lexeme? Define a regular set.

•A Lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

•A language denoted by a regular expression is said to be a regular set

12. What is a sentinel? What is its usage?

A Sentinel is a special character that cannot be part of the source program. Normally we use 'eof' as the sentinel. This is used for speeding-up the lexical analyzer.

13. What is a regular expression? State the rules, which define regular expression?

Regular expression is a method to describe regular language

Rules:

1) ε-is a regular expression that denotes {ε} that is the set containing the empty string
2) If a is a symbol in ∑,then a is a regular expression that denotes {a}
3)Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then,
   a)(r)/(s) is a regular expression denoting L(r) U L(s).
   b)(r)(s) is a regular expression denoting L(r)L(s)
   c)(r)* is a regular expression denoting L(r)*.
   d)(r) is a regular expression denoting L(r).

14. What are the Error-recovery actions in a lexical analyzer?

1. Deleting an extraneous character
2. Inserting a missing character
3. Replacing an incorrect character by a correct character
4. Transposing two adjacent characters

15. Construct Regular expression for the language

L= {w ε{a,b}/w ends in abb}

Ans: {a/b}*abb.

16. What is recognizer?

Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine,otherwise it is rejected

17. Define prefix of S.

A string obtained by removing zero or more trailing symbols of string s; Ex.,ban is a prefix of banana.

18. Define suffix of S.

A string obtained by removing zero or more leading symbols of string s; Ex.,nana is a suffix of banana.

19. Write notational shorthand.

One or more instances (+)
Zero or one instance (?)
Zero or more instances (*)

20. What is Finite Automata?

These are language recognizers. Generally a recognizer will identify an entity which is familiar to that and say it is the same entity or it is not that entity. Similarly the automata will take the input token and say 'yes' if it is a sentence of the language and 'no' if it is not a sentence of the language.

21. Define character class.

[A-Za-z][A-Za-z0-9]*

22. Define nonregular set.

Repeating strings cannot be described by regular expression. The set
{wcw | w is a string of a's and b's}

23. Define identifier.

letter→A | B | ……| Z | a | b | …..| z
digit → 0 | 1 |…….|9
id → letter (letter | digit) *

24. Define num.

digit → 0 | 1 |…….|9
digits → digit digit*
optional_fraction → .digits | €
optional_exponent → (E(+|-| €)digits) | €
num → digits optional_fraction optional_exponent
id → letter (letter | digit) *

PART –B [Text Book:Alfred Aho,Ravi Sethi,V.Jeffery Ullman]

1. Explain Role of Lexical Analyzer. **(Page No.84)**
2. Explain Tokens with its attributes.**(Page No.85)**
3. Explain Input Buffering with example. **(Page No.88)**
4. Explain the role Lexical Analyzer and issues of Lexical Analyzer.(8)
5. Explain the specification of tokens. (8)

UNIT III-Syntax Analysis

PART –A

1.What is the output of syntax analysis phase? What are the three general types of parsers for grammars?

Parser (or) parse tree is the output of syntax analysis phase.
General types of parsers:
1)Universal parsing
2)Top-down
3)Bottom-up

2.What are the different strategies that a parser can employ to recover from a syntactic error?

- Panic mode
- Phrase level
- Error productions
- Global correction

3.What are the goals of error handler in a parser?

The error handler in a parser has simple-to-state goals:
- It should report the presence of errors clearly and accurately.
- It should recover from each error quickly enough to be able to detect subsequent errors.
- It should not significantly slow down the process ing of correct programs.

4.What is phrase level error recovery?

On discovering an error, a parser may perform localcorrection on the remaining input; that is, it may replace a prefix of the remaining input by some string that allows the parser to continue. This is known as phrase level error recovery.

5.How will you define a context free grammar?

A context free grammar consists of terminals, non-terminals, a start symbol, and productions.

i.Terminals are the basic symbols from which strings are formed. “Token” is a synonym for terminal. Ex: if, then, else.

ii.Nonterminals are syntactic variables that denote sets of strings, which help define the language generated by the grammar. Ex: stmt, expr.

iii.Start symbol is one of the nonterminals in a grammar and the set of strings it denotes is the language defined by the grammar. Ex: S.

iv.The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings Ex: expr id

6.Define context free language. When will you say that two CFGs are equal?

•A language that can be generated by a grammar is said to be a context free language.

•If two grammars generate the same language, the grammars are said to be equivalent.

7. Give the definition for leftmost and canonical derivations.

•Derivations in which only the leftmost nonterminal in any sentential form is replaced at each step are termed leftmost derivations

•Derivations in which the rightmost nonterminal is replaced at each step are termed canonical derivations.

8.What is a parse tree?

A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding replacement order. Each interior node of a parse tree is labeled by some non terminal A and that the children of the node are labeled from left to right by symbols in the right side of the production by which this A was replaced in the derivation. The leaves of the parse tree are terminal symbols.

9.What is an ambiguous grammar? Give an example.

•A grammar that produces more than one parse tree for some sentence is said to be ambiguous

•An ambiguous grammar is one that produces more than one leftmost or rightmost derivation for the same sentence.

Ex:E E+E / E*E / id

10.Why do we use regular expressions to define the lexical syntax of a language?

i.The lexical rules of a language are frequently quite simple, and to describe them we do not need a notation as powerful as grammars.

ii.Regular expressions generally provide a more concise and easier to understand notation for tokens than grammars.

iii.More efficient lexical analyzers can be constructed automatically from regular expressions than from arbitrary grammars.

iv.Separating the syntactic structure of a language into lexical and non lexical parts provides a convenient way of modularizing the frontend of a compiler into two manageable-sized components.

11.When will you call a grammar as the left recursive one?

A grammar is a left recursive if it has a nonterminal A such that there is a derivation $A \Rightarrow A\alpha$ for some string α.

12.Define left factoring.

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing. The basic idea is that when it is not clear which of two alternative productions to use to expand a nonterminal "A", we may be able to rewrite the "A" productions to refer the decision until we have seen enough of the input to make the right choice.

13.Left factor the following grammar:

S →iEtS | iEtSeS |a

E →b.
Ans:
The left factored grammar is,
S →iEtSS′| a
S′→eS | ε
E →b

14.What is parsing?
Parsing is the process of determining if a string of tokens can be generated by a grammar.

15.What is Top Down parsing?
Starting with the root, labeled, does the top-down construction of a parse tree with the starting nonterminal, repeatedly performing the following steps.
i.At node n, labeled with non terminal "A", select one of the productions for "A"and construct children at n for the symbols on the right side of the production.
ii.Find the next node at which a sub tree is to be constructed.

16.What do you mean by Recursive Descent Parsing?
Recursive Descent Parsing is top down method of syntax analysis in which we execute a set of recursive procedures to process the input. A procedure is associated with each nonterminal of a grammar.

17.What is meant by Predictive parsing?
A special form of Recursive Descent parsing, in which the look-ahead symbol unambiguously determines the procedure selected for each nonterminal, where no backtracking is required.

18.Define Bottom Up Parsing.
Parsing method in which construction starts at the leaves and proceeds towards the root is called as Bottom Up Parsing.

19.What is Shift-Reduce parsing?
A general style of bottom-up syntax analysis, which attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root.

20.Define handle. What do you mean by handle pruning?
•An Handle of a string is a sub string that matches the right side of production and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.
•The process of obtaining rightmost derivation in reverse is known as Handle Pruning.

21.Define LR (0) items.
An LR (0) item of a grammar G is a production of G with a dot at some position of the right side. Thus the production A →XYZ yields the following four items,
A →.XYZ
A →X.YZ
A →XY.Z
A →XYZ.

22.What do you mean by viable prefixes?

•The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes.

•A viable prefix is that it is a prefix of a right sentential form that does not continue the past the right end of the rightmost handle of that sentential form.

23.What is meant by an operator grammar? Give an example.

A grammar is operator grammar if,

i.No production rule involves "εεεε"on the right side.

ii.No production has two adjacent nonterminals on the right side.

Ex:E →E+E | E-E | E*E | E/E | E↑E | (E) | -E | id

24.What are the disadvantages of operator precedence parsing?

i.It is hard to handle tokens like the minus sign, which has two different precedences.

ii.Since the relationship between a grammar for the language being parsed and the operator – precedence parser itself is tenuous, one cannot always be sure the parser accepts exactly the desired language.

iii.Only a small class of grammars can be parsed using operator precedence techniques.

25.State error recovery in operator-Precedence Parsing.

There are two points in the parsing process at which an operator-precedence parser can discover the syntactic errors:

i.If no precedence relation holds between the terminal on top of the stack and the current input.

ii.If a handle has been found, but there is no production with this handle as a right side.

26.LR (k) parsing stands for what?

The "L" is for left-to-right scanning of the input,the "R" for constructing a rightmost derivation in reverse, and the k for the number of input symbols of lookahead that are used in making parsing decisions.

27.Why LR parsing is attractive one?

•LR parsers can be constructed to recognize virtually all programming language constructs for which context free grammars can be written.

•The LR parsing method is the, most general nonbacktracking shift-reduce parsing method known, yet it can be implemented as efficiently as other shift reduce methods.

•The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with predictive parsers.

•An LR parser can detect a syntactic error as soon as it is possible to do so on a left-to-right scan of the input.

28.What is meant by goto function in LR parser? Give an example.

•The function goto takes a state and grammar symbol as arguments and produces a state.

•The goto function of a parsing table constructed from a grammar G is the transition function of a DFA that recognizes the viable prefixes of G.

Ex:goto(I,X) Where I is a set of items and X is a grammar symbol to be the closure of the set of all items [A→αX.β] such that [A→α.Xβ] is in I .

29.Write the configuration of an LR parser?

A configuration of an LR parser is a pair whose first component is the stack contents and whose second component is the unexpended input: (s0X1 s1X2 s2 ...Xmsm , ai ai+1... an$)

30.Define LR grammar.

A grammar for which we can construct a parsing table is said to be an LR grammar.

31.What are kernel and non kernel items?

i.The set of items which include the initial item, S′→.S, and all items whose dots are not at the left end are known as kernel items.

ii.The set of items, which have their dots at the leftend, are known as non kernel items.

32.Why SLR and LALR are more economical to construct than canonical LR?

For a comparison of parser size, the SLR and LALR tables for a grammar always have the same number of states, and this number is typically several hundred states for a language like Pascal. The canonical LR table would typically have several thousand states for the same size language. Thus, it is much easier and more economical to construct SLR and LALR tables than the canonical LR tables.

33.What is Syntax Analyzer?

This receives valid tokens from the scanner and checks them against the Grammar and produces the valid syntactical constructs. This is also called Parser.

34.What is Left Recursion?

A grammar is left recursive if it has a non terminal 'A' such that, there is a derivation as follows A → AX for some input string, where X is a grammar symbol.

35.What are the steps involved in Non Recursive predictive parsing?

The steps involved are:

- Input buffer is filled with input string with ∃ as the right end marker.
- Stack is initially pushed with ∃
- Construction of Parsing Table T
- Parsing by parsing routine

36.What is LL(1) Grammar?

A grammar 'G' whose parsing table has no multiply defined entries, can be called as LL(1) grammar.

37.What are Operator Precedence relations?

There are three operator precedence relations and they are represented by <. , = and .>These have the following meanings:

T1 <. T2 t1 yields precedence to t2

T1 = t2 t1 has the same precedence as t2

T1 .> t2 t1 takes precedence over t2

Where t1,t2 are terminals.

38. What are the two functions of parser?

- It checks the tokens appearing in its input, which is output of the lexical analyzer.

➢ It involves grouping the tokens of source program into grammatical phrases that are used by the compiler to synthesize the output. Usually grammatical phrases of the source program are represented by tree like structure called parse tree.

39. specify demerits ambiguous grammar

- It is difficult to select or determine which parse tree is suitable for an input string.

40.Mention the properties of parse tree.

- The root is labeled by the start symbol.
- Each leaf is labeled by a token or by□ε□
- Each interior node is labeled by a non terminal
- If A is the Non terminal, labeling some interior node and x1, x2, x3 … .xn

are the labels of the children.

41.What do you mean by a syntax tree?

Syntax tree is a variant of a parse tree in which each leaf represents an operand and each interior node represents an operator.

42.Define Handle pruning.

A technique to obtain the rightmost derivation in reverse (called canonical reduction sequence) is known as handle pruning (i.e.) starting with a string of terminals w to be parsed. If w is the sentence of the grammar then□α=α□ n where□ α□ n is the nth right sentential form of unknown right most derivation.

43.What are the demerits of SLR?

- ➢ It will not produce uniquely defined parsing action tables for all grammars.
- ➢ Shift-Reduce conflict.

44.Why LR parsing is good and attractive?

- LR parsers can be constructed for all programming language constructs for which CFG can be written.
- LR parsing is Non-backtracking Shift-Reduce parsing.
- Grammars parsed using LR parsers are super set of the class of grammar.
- LR parser can detect syntactic error as soon as possible, when left-to-right
- Scan of the input.
-

45.What are the rules for " Closure operation" in SLR parsing?

- If 'I' is a set of items for grammar G then Closure (I) is the set of items constructed from I by the following 2 rules.
- Initially every item in I is added to Closure (I)
- If A → α .Bβ is in Closure(I) and BÆα to I , then add the item BÆ.α to I, if it is not

Already there. Apply this until no more new items can be added to Closure (I).

46.Mention the demerits of LALR parser.

- Merger will produce reduce / reduce conflict.
- On erroneous input, LALR parser may proceed to do some reductions

- After the LR parser has declared an error, but LALR parser never shift a
- Symbol after the LR parser declares an error.
-

47.What are Terminals and Non Terminals?

Terminals are symbols which are used to make strings in any language. Non terminals are the symbols which are used to specify the strings.

48.What is a production rule?

Production rules define the way in which the syntactical constructs may be built, from the terminals. Each Production rule consists of non terminal followed by a string of non terminals and terminals.

49.What are the ways of constructing LR parsing tables?

The different ways of constructing LR Parsing tables are:

- SLR (Simple LR)
- CLR (Canonical LR)
- LALR (look ahead LR)
-

50.What are the conflicts during Shift reduce parsing?

There are context free grammars for which shift reduce parsing cannot be used. Every shift reduce parser for such a grammar can reach a configuration in which the parser knowing the entire stack contents and the next input symbol cannot decide whether to shift or to reduce a shift/reduce conflict or cannot decide which of several reductions to make a reduce/reduce conflict.

51.What are the components involved in the construction of SLR Parsing Table?

The components involved are:

- Construction of sets of LR(O) items collection using CLOSURE function and goto function.
- Construction of parsing table using LR(O) items
-

52.What are the actions performed on a SLR Parsing Table?

The Parsing table has two parts. They are Action entries and Goto Entries. The actions performed are shift, reduce, accept and error. The Goto entries will be having state numbers like 1,2,3 etc.

53.What is Data flow engine?

Much of the information needed to perform good code optimization involves “data flow analysis,” the gathering of information about how values are transmitted from one part of a program to each other part. This is handled by the data flow engines.

54.What are Syntax directed translation engines?

These produce collections of routines that walk the parse tree generating intermediate code.

55. What is the syntax for YACC source specification program?

Declarations
%%
Translation rules
%%
Supporting C-routines

56.How YACC resolves parsing action conflicts?

- A reduce/reduce conflict is resolved by choosing the conflicting production listed first in the YACC specification.
- A shift/reduce conflict is resolved by shifting action or introduces associativity and precedence.

57.What is the new production added to the YACC on error recovery?

When parser generated by YACC encounters errors, then YACC pops symbol from its stack until it finds the topmost state on its stack whose underlying set of items includes an item of the form A → Error □ □α□

PART-B QUESTIONS (16 MARKS)

1.What are preliminary steps that are to be carried out during parsing? Explain with suitable examples.

2.Explain the error recovery in predictive parsing.

3.What are the necessary conditions to be carried outbefore the construction of predictive parser?

4.i) Construct the predictive parser for the following grammar:
S →(L) | a
L →L,S | S.
ii) Construct the behavior of the parser on the sentence (a, a) using the grammar specified above.

5.i) Give an algorithm for finding the FIRST and FOLLOW positions for a given non-terminal.
ii) Consider the grammar, E →TE′
E →+TE′| ε
T →FT′
T′→*FT′| ε
F →(E) | id. Construct a predictive parsing table for the grammar given above. Verify whether the input string id + id * id is accepted by the grammar or not.

6.Construct the predictive parser for the following grammar:
S →a | ↑| (T)
T →T, S | S .Write down the necessary algorithms and define FIRST and FOLLOW. Show the behavior of the parser in the sentences, i.(a,(a,a)) ii.(((a,a), ↑,(a),a) .

7. Check whether the following grammar is SLR (1) or not. Explain your answer with reasons.

S →L = R

S →R
L →* R
L →id
R →L.
8.For the operators given below, calculate the operator-precedence relations and operator precedence function id, +, *, $.

9.Check whether the following grammar is a LL(1) grammar
S →iEtS | iEtSeS | a
E →b
Also define the FIRST and FOLLOW procedures.

10.Consider the grammar
E →E + E | E * E | (E) | id
Show the sequence of moves made by the shift-reduce parser on the input
id1+ id2* id3 and determine whether the given string is accepted by the parser or not.

11.i) What is a shift-reduce parser? Explain in detail the conflicts that may occur during shift-reduce parsing.
ii) For the grammar given below, calculate the operator precedence relation and the precedence functions
E →E + E | E - E | E * E | E / E | E ↑E | (E) | -E | id

12.i) Consider the grammar given below.
E →E + T
E →T
T →T * F
T →F
F →(E)
F→id Construct an LR parsing table for the above grammar. Give the moves of LR parser on id*id+id .
ii) Briefly explain error recovery in LR parsing.

13.Explain the LR parsing algorithm with an example.

14.Construct a canonical parsing table for the grammar given below
S →CC
C →cC | d
15.Explain the Stack Implementation of Shift Reduce Parsing. **(Page No.198)**

16.Explain Operator –Precedence Parsing. **(Page No.203)**

17.Explain error recovery in Operator- Precedence Parsing. **(Page No.210)**

18.Explain LR Parsing Algorithm. **(Page No.216)**

19.Explain Parser Generators. **(Page No.257)**

UNIT – IV-SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT

PART – A

1.What are the two standard storage allocation strategies?

The two standard allocation strategies are
1.Static allocation.
2.Stack allocation

2. Discuss about static allocation.

In static allocation the position of an activation record in memory is fixed at run time.

3.Write short notes on activation tree.

•A tree which depicts the way of control enters and leaves activations.
•In an activation tree
i.Each node represents an activation of an procedure
ii.The root represents the activation of the main program.
iii.The node for a is the parent of the node for b , if and only if control flows from activation a to b
iv.Node for a is to the left of the node for b, if and only if the lifetime of a occurs before the lifetime of b.

4.Define control stack.

A stack which is used to keep track of live procedure actions is known as control stack.

5.Define heap.

A separate area of run-time memory which holds all other information is called a heap.

6.Give the structure of general activation record

Returned value
Actual parameters
Optional control link
Optional access link
Saved machine status
Local data
Temporaries

7.Discuss about stack allocation.

In stack allocation a new activation record is pushed on to the stack for each execution of a procedure. The record is popped when the activation ends.

8.What are the 2 approaches to implement dynamic scope?

•Deep access
•Shallow access

9.What is padding?

Space left unused due to alignment consideration is referred to as padding.

10.What are the 3 areas used by storage allocation strategies?

•Static allocation

•Stack allocation
•Heap allocation

11.What are the limitations of using static allocation?

•The size of a data object and constraints on its position in memory must be known at compile time.

•Recursive procedure are restricted, because all activations of a procedure use the same bindings for local name

•Data structures cannot be created dynamically since there is no mechanism for storage allocation at run time

12.Define calling sequence and return sequence.

•A call sequence allocates an activation record and enters information into its fields

•A return sequence restores the state of the machine so that calling procedure can continue execution.

13.When dangling reference occurs?

•A dangling reference occurs when there is storage that has been deallocated.

•It is logical error to use dangling references, since the value of deallocated storage is undefined according to the semantics of most languages.

14.Define static scope rule and dynamic rule .

•Lexical or static scope rule determines the declaration that applies to a name by a examining the program text alone.

•Dynamic scope rule determines the declaration applicable to name at runtime, by considering the current activations.

15.What is block? Give its syntax.

•A block is a statement containing its own data declaration.

•Syntax:

{
Declaration statements
}

16.What is access link?

•An access link is a pointer to each activation record which obtains a direct implementation of lexical scope for nested procedure.

17.What is known as environment and state?

•The term environment refers to a function that maps a name to a storage location.

•The term state refers to a function that maps a storage location to the value held there.

18.How the run-time memory is sub-divided?

•Generated target code
•Data objects
•A counterpart of the control stack to keep track of procedure activations.

19. What are the types of intermediate representations?

The types of intermediate representations are Syntax trees, postfix notation, three address code.

20. What is a symbolic label?

A symbolic label represents the index of a three address statement in the array holding intermediate code.

21. What is Quadruple?

A Quadruple is a record structure with four fields, which we call op, arg1, arg2 and result. The op field contains an internal code for the operator.

22. What are the functions used for manipulating the list of labels in Backpatching?

The lists of functions are

a. makelist(i) - creates a new list containing only I,an index into the array of quadruples.
b. merge(p1,p2) – concatenates the lists pointed to by p1 and p2
c. backpatch(p,i) – inserts I as the target label for each of the statements on the list pointed to by p.

23. What is type system?

The type system is a collection of rules for assigning type expressions in the program. The type checker implements the type system.

24. Define lifetime.

The life time of any procedure refers to the consecutive sequence of steps from beginning to end of that procedure during the execution of the program.

25. What is activation record?

An activation record is a record which keeps the information necessary for an activation of a procedure and also manages that information.

26.. How the value of synthesized attribute is computed?

It was computed from the values of attributes at the children of that node in the parse tree.

27.How the value of inherited attribute is computed?

It was computed from the value of attributes at the siblings and parent of that node.

28. Define backpatching.

To generate three address code, 2 passes are necessary. In first pass, labels are not specified. These statements are placed in a list. In second pass, these labels are properly filled, is called backpatching.

29. What are the three functions used for backpatching?

- Makelist (i)
- Merge (p1, p2)
- Backpatch (p, i)

30. When procedure call occurs, what are the steps to be takes placed?
- State of the calling procedure must be saved, so that it can resume after completion of procedure.
- Return address is saved, in this location called routine must transfer after completion of procedure

31. What are the demerits in generating 3-address code for procedure calls?
Consider, b=abc (I, J) .It is very difficult to identify whether it is array reference or it is a call to the procedure abc.

32. Define backpatch.
backpatch(p,i) – inserts I as the target label for each of the statements on the list pointed to by p.

33. What is the usage of syntax directed definition.
Syntax trees for assignment statement are produced by the syntax directed definition

34. Give the syntax directed definition for if-else statement.

Ans:

Production	Semantic rule
S →if E then S1 else S2	E.true := newlabel; E.false := newlabel; S1.next := S.next S2.next := S.next S.code := E.code || gen(E.true ':') || S1.code || gen('goto' S.next) || gen(E.false ':') || S2.code

PART –B [Text Book:Alfred Aho,Ravi Sethi,V.Jeffery Ullman]

1. Explain Symbol tables. **(Page No.429)**
2. Explain the different types of Three – address Statements. **(Page No.467)**
3. Explain Syntax Directed Translation into Three Address Code. **(Page No.468)**
4. Explain the operations used in Declarations. **(Page No.476)**
5. Explain Procedure Calls. **(Page No.506)**

UNIT – V-CODE OPTIMIZATION &CODE GENERATION

PART – A

1. Define basic blocks.
   a. A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.

2. What are the structure preserving transformations?
   The structure preserving transformations are:
   - Common sub expression elimination
   - Dead code elimination
   - Renaming of temporary variables
   - Interchange of two independent adjacent statements
   -

3. What is a register descriptor?
   A register descriptor keeps track of what is currently in each register. It is consulted whenever a new register is needed.

4. What are the nodes on a directed acylic graph?
   The nodes are:
   - Leaves are labeled by unique identifiers, either variable names or constants.
   - Interior nodes are labeled by an operator symbol.
   - Nodes are also optionally given a sequence of identifiers for labels.
   -

5. What is local and global program?
   A transformation of a program is called local if it can be performed by looking only at the statements in a basic block; otherwise it is called global.

6. What is dead code elimination?
   A variable is live at a point in a program if its value can be used subsequently; otherwise it is dead at that point.

7. What is dominator tree?
   In a dominator tree, the initial node is the root and each node dominates only to its descendants.

8. When is a flow graph reducible?
   A flow graph is reducible if and only if we can partition the edges into two disjoint groups often called the forward edges and back edges.

9. What is induction variable?
   A variable is called an induction variable of a loop if every time the variable changes values, it is incremented or decremented by some constant.

10. What is static memory allocation?
    This allocation is done at the compile time and the data for the procedure can be in advance known and the sizes are also known at compile time, so these memories can be allocated statically.

11.Mention some of the major optimization techniques.

- Local optimization
- Loop optimization
- Data flow analysis
- Function preserving transformations
- Algorithm optimization.
-

12. What are the methods available in loop optimization?

- Code movement
- Strength reduction
- Loop test replacement
- Induction variable elimination

13. What is the step takes place in peephole optimization?

It improves the performance of the target program by examining a short sequence of target instructions. It is called peephole. Replace this instructions by a shorter or faster sequence whenever possible. It is very useful for intermediate representation.

14. What are the characteristics of peephole optimization?

a. Redundant instruction elimination.
b. Flow of control optimization
c. Algebraic simplifications
d. Use of machine idioms

15. What are the rules to find " leader" in basic block?

- It is the first statement in a basic block is a leader.
- Any statement which is the target of a conditional or unconditional goto is a leader.
- Any statement, which immediately follows a conditional goto, is a leader.
-

16. Define flow graph.

Relationships between basic blocks are represented by a directed graph called flow graph.

17. What are the principle sources of optimization?

The principle sources of optimization are,

- Optimization consists of detecting patterns in the program and replacing these patterns by equivalent but more efficient constructs.
- The richest source of optimization is the efficient utilization of the registers and instruction set of a machine.

18. What is a cross complier?

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. (ie). A compier may run on one machine and produce target code for another machine.

19. What are the properties of optimizing compilers?

Compilers that apply code- improving transformations are called **optimizing compilers.** The properties of optimizing compilers are:

- Transformation must preserve the meaning of programs.
- Transformation must, on the average, speed up programs by a measurable amount.
- Transformation must be worth the effect.
-

20. What does the runtime storage hold?

Runtime storage holds

1) The generated target code
2) Data objects
3) A counter part of the control stack to keep track of procedure activations.

21.Define code generation.

The code generation is the final phase of the compiler. It takes an intermediate representation of the source program as the input and produces an equivalent target program as the output.

22. State the problems in code generation.

Three main problems in code generation are,

- Deciding what machine instructions to generate.
- Deciding in what order the computations should be done and
- Deciding which registers to use.

23. How the quality of object program is measured?

The quality of an object program is measured by its Size or its running time. For large computation running time is particularly important.For small computations size may be as important or even more.

24. .What is the more accurate term for code optimization?

The more accurate term for code optimization would be “code improvement”

25. .Explain the principle sources of optimization.

Code optimization techniques are generally applied after syntax analysis, usually both before and during code generation. The techniques consist of detecting patterns in the program and replacing these patterns by equivalent and more efficient constructs.

26. What are the patterns used for code optimization?

The patterns may be local or global and replacement strategy may be a machine dependent or independent

27. .What are the 3 areas of code optimization?

•Local optimization
•Loop optimization
•Data flow analysis

28. Define local optimization.

The optimization performed within a block of code is called a local optimization.

29. .Define constant folding.

Deducing at compile time that the value of an expression is a constant and using the constant instead is known as constant folding.

30. .What do you mean by inner loops?

The most heavily traveled parts of a program, the inner loops, are an obvious target for optimization. Typical loop optimizations are the removal of loop invariant computations and the elimination of induction variables.

31. .What is code motion?

Code motion is an important modification that decreases the amount of code in a loop.

32. .What are the properties of optimizing compilers?

•Transformation must preserve the meaning of programs.
•Transformation must, on the average, speed up the programs by a measurable amount
•A Transformation must be worth the effort.

33. .Give the block diagram of organization of code optimizer.

34. What are the advantages of the organization of codeoptimizer?

a.The operations needed to implement high level constructs are made explicit in the intermediate code, so it is possible to optimize them.
b.The intermediate code can be independent of the target machine, so the optimizer does not have to change much if the code generator is replaced by one for a different machine

35. Define Local transformation & Global Transformation

. A transformation of a program is called Local, if it can be performed by looking only at the statements in a basic block otherwise it is called global.

36. .Give examples for function preserving transformations.

•Common subexpression elimination
•Copy propagation
•Dead – code elimination
•Constant folding

37. .What is meant by Common Subexpressions?

An occurrence of an expression E is called a common subexpression, if E was previously computed, and the values of variables in E have not changed since the previous computation.

38. .What is meant by Dead Code?

A variable is live at a point in a program if its value can be used subsequently otherwise, it is dead at that point. The statement that computes values that never get used is known Dead code or useless code.

39. What are the techniques used for loop optimization?

i)Code motion
ii)Induction variable elimination
iii)Reduction in strength

40 .What is meant by Reduction in strength?

Reduction in strength is the one which replaces an expensive operation by a cheaper one such as a multiplication by an addition.

41.What is meant by loop invariant computation?

An expression that yields the same result independent of the number of times the loop is executed is known as loop invariant computation.

42.Define data flow equations.

A typical equation has the form Out[S] = gen[S] U (In[S] – kill[S]) and can be read as, " the information at the end ofa statement is either generated within the statement, or enters at the beginning and is not killed as control flows through the statement". Such equations are called data flow equations.

PART –B [Text Book:Alfred Aho,Ravi Sethi,V.Jeffery Ullman]

1. Explain the issues in the design of a Code Generator. **(Page No.514)**
2. Explain the Code Generation Algorithm. **(Page No.537)**
3. How do you construct a directed Acyclic Graph? **(Page No.546)**
4. Explain Peephole Optimization. **(Page No.554)**
5. Explain the principle sources of Optimization. **(Page No.592)**
6. Explain loops in Flow Graph. **(Page No.602)**

PREPARED BY

(UMADEVI.V)

(S.DHANALAKSHMI)

B.E/B.Tech.DEGREE EXAMINATION, MAY/JUNE 2007

Sixth semester

Computer Science and Engineering

CS337-PRINCIPLES OF COMPILER DESIGN

(Regulation 2004)

Time : Three hours **Maximum : 100 marks**

Part A --(10*2=20 marks)

Answer All questions

1. Define Compiler.

   Compiler is a program that can read a program in one language(source language) and translate it into an equivalent program in another language(target language).

2. What is Regular Expression?

   A regular expression, often called a **pattern**, is an expression that describes a set of strings.

3. What are Handles?

   A handle of a string is a substring that matches the right side of a production and whose reduction to the non terminal on the left side of the production represents one step along the reverse of a rightmost derivation.

4. List out the actions involved in shift reduce parsing.
   a. Shift.
   b. Reduce.
   c. Accept.
   d. Error.

5. Eliminate Left Recursion for the given grammar.

   $E \rightarrow E + T \mid T$
   $T \rightarrow T * F \mid F$
   $F \rightarrow (\ E\ )\mid$ id

   E → TE'

E' → +TE' | ϵ
T → FT'
T' → *FT' | ϵ
F → (E) | id

6. What is three address code(TAC) and generate TAC for the statement:

   a=2*b+3*c

   Three-address code (often abbreviated to TAC or 3AC) is a form of representing intermediate code used by compilers to aid in the implementation of code-improving transformations. Each instruction in three-address code can be described as a 4-tuple: (operator, operand1, operand2, result).

   t1=3*c

   t2=2*b

   t3=t1+t2

7. What is SDT?

   Syntax-directed translation (SDT) is a method of translating a string into a sequence of actions by attaching one such action to each rule of a grammar. Thus, parsing a string of the grammar produces a sequence of rule applications. And SDT provides a simple way to attach semantics to any such syntax.

8. Define Directed Acyclic Graph?

   A **directed acyclic graph** (commonly abbreviated to **DAG**), is a directed graph with no directed cycles. That is, it is formed by a collection of vertices and directed edges, each edge connecting one vertex to another. In this common sub expressions are eliminated. So it is a compact way of representation.

9. Optimize following the code by eliminating common sub expression:

   Y=A+B*x+C*(x**2)+D*(x**3)

   (Refer Book)

10. What is peephole optimization?

    Peephole optimization is a kind of optimization performed over a very small set of instructions in a segment of generated code. The set is called a "peephole" or a "window". It

works by recognizing sets of instructions that don't actually do anything, or that can be replaced by a leaner set of instructions.

PART – B (5 X 16 = 80 MARKS)

11. (a) Explain the various phases of compiler in detail with neat sketch. **(Page No:5)**

[OR]

(b) (i) Explain with neat sketch implementation of Lexical analyzer. **(Page No:76)**

(ii) Describe in detail about any three compiler construction tools. **(Page No:12)**

12. (a) (i) Explain the working model of top down parsing and bottom up parsing?

(Page No:217,233)

(ii) Describe the error detection and recovery process involved in the lexical phase. **(Page No:195)**

[OR]

(b) Write the algorithm for predictive parser and parse the input expression: x-2*y using the below given grammar G: **(Page No:217)**

E→E+T | E-T | T

T→T*F | T/F | F

F→num | id

13. (a) (i) Perform LR parsing and drive the input a (a , a (a , a)) using the below given grammar: **(Page No:242)**

S→S

S→a (L)

S→a

L→S, L

L→S

(ii) Perform shift reduce parsing for the input 2 * (1+3) usin the grammar:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (\ E\ ) \mid$ id

[OR]

(b) (i) Write an algorithm for generating LR item sets and constructing SLR Parsing table. **(Page No:245, 253)**

(ii) Write about LALR Parser. **(Page No:270)**

14. (a) Brief Intermediate code generation for Basic block, Control flow and Boolean expressions. **(Page No:399)**

[OR]

(b) (i) Explain the data structure used for implementing symbol table. **(Page No:85)**

(ii) Write about Quadruple and Triple with its structure. **(Page No:366)**

15. (a) (i) Explain various code optimization techniques in detail. **(Page No:533)**

(ii) Generate target code for the given program segment:

```
main()
{
        int i,j;  i=4; j=i+5;
}
```

[OR]

(b) (i) Explain the various issues involved in the design of Code generation. **(Page No:506)**

(ii) Describe in detail about Run time storage Management. **(Page No:427)**

B.E/B.Tech.DEGREE EXAMINATION, NOVEMBER/DECEMBER 2007
Sixth semester

Computer Science and Engineering

CS1352-PRINCIPLES OF COMPILER DESIGN

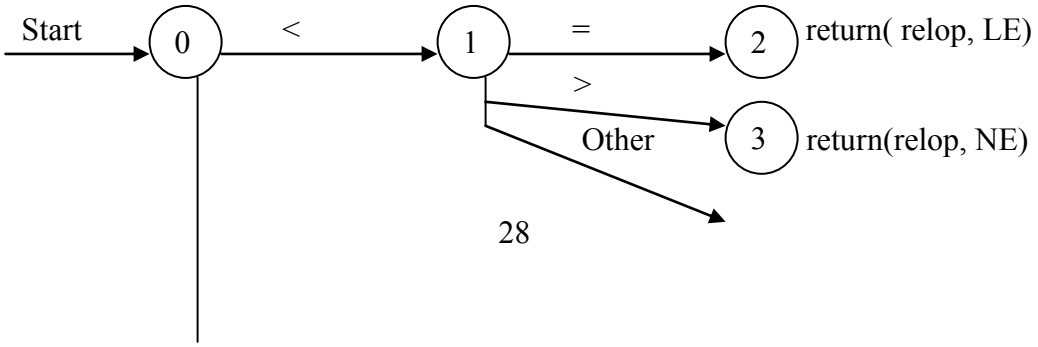
Time : Three hours **Maximum : 100 marks**

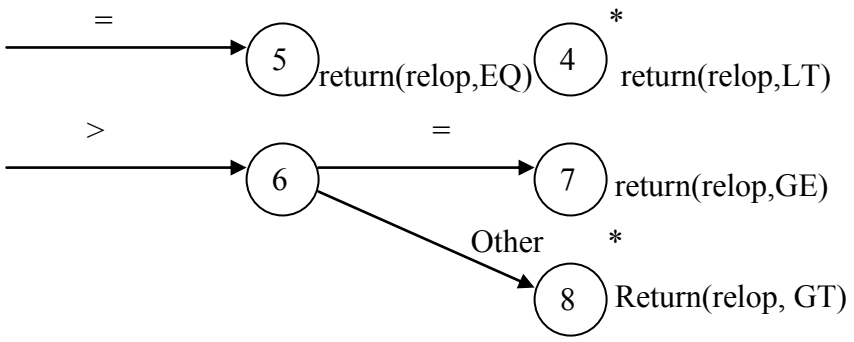
Part A --(10*2=20 marks)

1. **What is an ambiguous grammar?**

If a grammar produces more than one parse tree for the given input string, then it can be called as ambiguous grammar.

2. **Draw a transition diagram to represent relational operators**.





3. What is left factoring? Give an example.

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.

4. What is a predictive parser?

Predictive parser is a program based on a transition diagram attempts to match the terminal symbols against the input.

5. **What are the goals of error handler in a parser**?

The goals of error handler in a parser are

- To maximize input text
- To report actual and presence of errors clearly and accurately.
- To recover from each error quickly enough to be able to detect – subsequent errors.
- Not to significantly slow down the processing of correct programs.

6. **State the parts of a Yacc source program**.

The Yacc source program has 3 parts:
Declarations
%%
Translation rules
%%
Supporting C-routines

7. What are the notations used to represent an intermediate languages?

The notations used to represent intermediate languages are

Syntax trees.

Postfix notation

Three – address code

8. **What is the sequence of actions taking place on entry to a procedure**?

The sequence of actions taking place on entry to a procedure includes calling sequence, translation for a call. A call sequence allocates an activation record and enters information

into its fields. The code in calling sequence is often divided between the calling procedure (the caller) and the procedure it calls (the callee)

9. **What is a cross complier?**

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. (ie). A compier may run on one machine and produce target code for another machine.

10. **What are the properties of optimizing compilers**?

Compilers that apply code- improving transformations are called **optimizing compilers.** The properties of optimizing compilers are:

- Transformation must preserve the meaning of programs.
- Transformation must, on the average, speed up programs by a measurable amount.
- Transformation must be worth the effect.

PART – B (5 X 16 = 80 MARKS)

11. a. (i) What is a compiler? Explain the various phases of compiler in detail.

Complier is a program that reads a program written in one language (source language)- and translates it into an equivalent program in another language-(target language). In this translation process, the complier reports to its user the presence of the errors in the source program.

Refer. Page. No. 10

(ii) What is a regular expression? Explain the rules with an example.

Refer. Page. No. 94

b. (i) Give a detailed note on the compiler- construction tools.

Refer. Page. No. 22

(ii) Explain briefly about input buffering in reading the source program for finding the tokens.

Refer. Page. No. 88

12. a. (i) Explain in detail about design of a lexical analyzer generator.

Refer. Page. No. 198

(ii) Explain the non-recursive predictive parsing with its algorithm.

Refer. Page. No. 187

b. (i) Construct a minimum state DFA for the regular expression:

(a b) * a b c

Refer. Page. No. 136

(ii) List and explain the different types of error recovery strategies.

Refer. Page. No. 192

13. a. (i) For the operators given below, calculate the operator precedence relations and operator-precedence function.

id, +, *,$

Refer. Page. No. 204 & 205

(ii) Explain the LR parsing algorithm in detail.

Refer. Page. No. 218

b. Construct a canonical parsing table for the grammar given below.

S → CC C → cC | d

Refer. Page. No. 231 -235

14 a. (i) Explain the various implementation of three address statements.

Refer. Page. No. 466 & 467

(ii) Explain the operations that are used to define the semantic rules.

Refer. Page. No. 286 & 287

b. (i) Describe the method of generating syntax-directed definition for control statements.

Refer. Page. No. 280

(ii) Explain the various methods for implementing symbol table and compare them.

Refer. Page. No. 429

15. a. Explain the various issues involved in the design of a code generator.

Refer. Page. No. 514

b. Explain the following regarding runtime storage management:

(i) Static allocation

Refer. Page. No. 522

(ii) Stack allocation

Refer. Page. No. 524

B.E/B.Tech.DEGREE EXAMINATION, APRIL/MAY 2005

Sixth semester

Computer Science and Engineering

CS1352-PRINCIPLES OF COMPILER DESIGN

Time : Three hours **Maximum : 100 marks**

PART – A (10 X 2 = 20 Marks)

1. **What are the cousins of the compiler?**

The cousins of the compiler are

- Preprocessors
  - Macro processing.
  - File inclusion.
  - Rational preprocessors
  - Language Extensions.
- Assemblers.
- Loaders and Link editors

2. **What are the error recovery actions in a lexical analyzer?**

- Panic mode recovery.
- Deleting an extraneous character
- Inserting a missing character.
- Replacing an incorrect character by a correct character.
- Transposing two adjacent characters.

3. **What are the algebraic properties of regular expressions?**

- Union
- Kleene Closure
- Positive Closure

4. **What is a finite automation?**

Finite automation are language recognizers. Generally a recognizer will identify an entity, which is familiar to that, and say it is the same entity or it is not that entity. Similarly the automata will take the input token and say 'yes' if it is a sentence of the language and 'no' if it is not a sentence of the language.

5. **What are the goals of error handler in a parser?**

The goals of error handler in a parser are

- To maximize input text
- To report actual and presence of errors clearly and accurately.

- To recover from each error quickly enough to be able to detect – subsequent errors.
- Not to significantly slow down the processing of correct programs.

6. Define Handle.

A handle of a string is a sub string that matches the right side of a production. This reduction helps in constructing the parse tree or right most derivation.

7. What is a basic block?

A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.

8. What are the various ways to pass parameters in a function?

- Call by value
- Call by reference

9. How would you calculate the cost of an instruction?

10. How would you map names to values?

PART-B(5 X 16 = 80 Marks)

11. **(i)** What are the various phases of the compiler? Explain each phase in detail. Write down the output of each phase for the expression. a: = b + c * 50.

Refer. Page. No. 10

(ii) Briefly explain the Compiler- Construction tools.

Refer. Page. No. 22

12. a. (i) Explain the process of constructing an NFA from a regular expression. Find NFA for the expression (a/b) * a(a | b) (a | b). Convert the obtained NFA into DFA.

Refer. Page. No. 117,118,121 - 123

(ii) What are the issues of the lexical analyzer?

Refer. Page. No. 84 & 85

b. (i) How would you construct a DFA directly from a regular expression? Explain with a suitable example.

Refer. Page. No. 135 & 136

(ii) What are the preliminary steps that are to be carried out during parsing? Explain with suitable example.

13. a. Construct the predictive parser for the following grammar:

$S \rightarrow a \mid \uparrow \mid (T)$

T → T , S | S

Write down the necessary algorithms and define FIRST and FOLLOW. Show the behavior of the parser in the sentences;

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

(ii) (((a, a), ↑, (a) ,a).

Refer. Page. No. 182 & 188

b. (i) Check whether the following grammar is a LL(1) grammar.

S → iEtS | iEtSeS | a

E → b

Refer. Page. No. 191

(ii). Explain the error recovery strategies on Predictive parsing.

Refer. Page. No. 192

(iii)Explain the LR parsing algorithm with an example.

Refer. Page. No. 218 - 220

14. a. (i) What is a three address code? What are its types? How it is implemented?

Refer. Page. No. 466 & 467

(ii) Explain how declarations are done in a procedure using syntax directed translation.

Refer. Page. No. 468

b. Explain the various data structures used for implementing the symbol table and compare them.

Refer. Page. No. 429

15. a. Explain the principle sources of code optimization in detail.

Refer. Page. No. 592

b. (i) What are the issues in the design of the code generator? Explain.

Refer. Page. No. 514

(ii) Explain the DAG representation of the basic block with an example.

Refer. Page. No. 598

B.E/B.Tech.DEGREE EXAMINATION, MAY/JUNE 2009
Sixth semester

Computer Science and Engineering

CS1352-PRINCIPLES OF COMPILER DESIGN

Time : Three hours **Maximum : 100 marks**

Part A --(10*2=20 marks)

1. What are the issues to be considered in the design of lexical analyzer?

How to Precisely Match Strings to Tokens

How to Implement a Lexical Analyzer

2. Define concrete and abstract syntax with example.

The grammar is mainly used for parsing only - the key is to resolve all ambiguities. This grammar is called **Concrete Syntax**.

- **Abstract Syntax** is used to characterize the essential structure of the program - the key is to be as simple as possible; it may contain ambiguities.
- **Traditional Compilers** do semantic analysis on Concrete Syntax
- **Modern Compilers do** the semantic analysis on Abstract Syntax tree

3. Derive the string and construct a syntax tree for the input string ceaedbe using the grammar S->SaA|A,A->AbB|B,B->cSd|e

Refer class notes.

4. List the factors to be considered for top-down parsing.

We begin with the start symbol and at each step, expand one of the remaining nonterminals by replacing it with the right side of one of its productions. We repeat until only terminals remain. The top-down parse produces a leftmost derivation of the sentence.

5. Why is it necessary to generate intermediate code instead of generating target program itself?

Intermediate code is simple enough to be easily converted to any target code.

Complex enough to represent the entire complex structure of high level language

6. Define back patching.

Backpatch (X , Y) : replace the line numbers listed in 'X' with the value 'Y'

7. List the issues in code generation.

Code generator tries to optimize the generated code in some way. The generator may try to use faster instructions, use fewer instructions, exploit available registers, and avoid redundant computations.

8. Write the steps for constructing leaders in basic blocks.

Determine leaders (first statements of BBs)

The first statement is a leader

The target of a conditional is a leader

A statement following a branch is a leader

For each leader, its basic block consists of the leader and all the statements up to but not including the next leader

9. What are the issues in static allocation?

Resources needed to meet the deadlines of safety-critical tasks are typically preallocated. Also, these tasks are usually statically scheduled such that their deadlines will be met even under worst-case conditions. Besides periodicity constraints, algorithms that do allocation and scheduling must be able to handle the resource, precedence, communication, and fault etc.

10. What is meant by copy-restor?

Refer class notes.

PART B--(5*16=80)

11. (a) (i) Explain the need for dividing the compilation process into various phases and explain its functions.

Answer the following with an example,

(ii) Explain how abstract stack machine can be used as translators.

Refer Principles of compiler design -aho , Page No.62

(or)

(b) What is syntax directed translation? How it is used for translation of expresions? (16)

Refer Principles of compiler design -aho , Page No.33

12. (a) Given the following grammar S->AS|b,A->SA|a Construct a SLR parsing table for the string baab (16)

Refer class notes.

(or)

(b) Consider the grammar E->E+T,T->T*F,F->(E)\id.Using predictive parsing the string id+id*id. (16)

Refer class notes.

13. (a) Explain in detail how three address code are generated and implemented.

Highlight the following

Quadraples, Binary Operator,Unary Operator, Move Operator, Unconditional Jumps, Conditional Jumps, Procedure Parameters, Procedure Calls, Indexed Assignments, Address and Pointer Assignments, Syntax-Directed Translation into Three-Address Code.

(or)

(b) Explain the role of declaration statements in intermediate code generation.

Highlights the important of declaration part in a program

Nested Procedure Declarations

14. (a) Design a simple code generator and explain with example.

Refer Principles of compiler design -aho , Page No.535

(or)

(b) Write short notes on:

1: Peep hole optimization **Refer Principles of compiler design -aho , Page No.554**

2: Issues in code generation **Refer Principles of compiler design -aho , Page No.514**

15. (a) Explain with an example how basic blocks are optimized.
Refer Principles of compiler design -aho , Page No.598

(or)

(b) Explain the storage allocation strategies used in run time environments.
Refer Principles of compiler design -aho , Page No.522

B.E/B.Tech.DEGREE EXAMINATION,APRIL/MAY 2008
CS1352-PRINCIPLES OF COMPILER DESIGN
Answer All questions

Part A --(10*2=20 marks)

1. Differentiate compiler and interpreter

An interpreter is a translator of high level language program.The function of these translator is to convert high level program into machine code.These translators translate one instruction at a time.A Compiler is a translator which is used to translate the whole source of a program at a time.

2.Write short notes on buffer pair.

Refer class notes.

3.Construct a parse tree of (a+b)*c for the grammer E->E+E/E*E/(E)/id.

Refer class notes

4.Eliminate immediate left recursion for the following grammer

E->E+T/T,T->T*F,F->(E)/id.

Refer class notes

5.Write short notes on global data flow analysis.

Data-flow analysis is a collection of techniques for compile-time reasoning about the run-time flow of values

6.Define back patching with an example.

Backpatch (X , Y) : replace the line numbers listed in 'X' with the value 'Y'

7.Give syntax directed translation for the following statement Call p1(int a, int b).

Refer class notes

8.How can you find the leaders in basic block.

Determine leaders (first statements of BBs)
The first statement is a leader
The target of a conditional is a leader
A statement following a branch is a leader
For each leader, its basic block consists of the leader and all the statements up to but not including the next leader

9.Define code motion.

In computer programming, **loop-invariant code** consists of statements (in an imperative programming language) which can be moved outside the body of a loop without affecting the semantics of the program.

Loop-invariant code motion (also called **hoisting** or **scalar promotion**) is a compiler optimization which performs this movement automatically.

10.Define basic block and flow graph.

A **basic block** is code that has one entry point (i.e., no code within it is the destination of a jump instruction), one exit point and no jump instructions contained within it. The start of a basic block may be jumped to from more than one location. The end of a basic block may be a jump instruction or the statement before the destination of a jump instruction. Basic blocks are usually the basic unit to which compiler optimizations are applied.

In a **control flow graph** each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow. There are, in most presentations, two specially designated blocks: the *entry block*, through which control enters into the flow graph, and the *exit block*, through which all control flow leaves.

PART B--(5*16=80)

11. a) i) Explain the phases of compiler with the neat schematic.

Answer the following with an example,

Lexical Analyzer → Syntax Analyzer → Semantic Analyzer → Intermediate Code Generator → Code Optimizer → Code Generator

ii) Write short notes on compiler construction tools.

Refer Principles of compiler design -aho , Page No.22

(or)

b) i) Explain grouping of phases.

Refer Principles of compiler design -aho , Page No.20

ii) Explain specification of tokens.

Refer Principles of compiler design -aho , Page No.92

12. a) Find the SLR parsing table for the given grammer and parse the sentence (a+b)*c.

E->E+E/E*E/(E)/id.

Refer class notes

(or)

b) Find the predictive parser for the given grammer and parse the sentence (a+b)*c.

E->E+T/T,T->T*F/F,F->(E)/id.

Refer class notes

13. a) Generate intermediate code for the following code segment along with the required syntax directed translation scheme.

i) if(a>b)
x=a+b
else
x=a-b
where a & x are of real and b of int type data.

ii) int a,b;
float c;
a=10;
switch(a)
{
case 10: c=1;
case 20: c=2;
}

Refer class notes

(or)

b) i) Generate intermediate code for the following code segment along with the required syntax directed translation scheme.

I=1;s=0;

While(i<=10)

s=s+a[i][i][i]

i=i+1; **Refer class notes**

ii) Write short notes on back-patching**.**

Refer Principles of compiler design -aho , Page No.500

14. a) i) Explain the various issues in the design of code generation,

Refer Principles of compiler design -aho , Page No.514

ii) Explain code generation phase with simple code generation algorithm.

Refer Principles of compiler design -aho , Page No.535

(or)

b) i) Generate DAG representation of the following code and list out the applications of DAG representation.

```
i=;s=0;
 while(i<=10)
  s=s+a[i][i];
  i=i+1;
```

Refer class notes and Refer Principles of compiler design -aho , Page No.550

ii) Write short notes on next-use information with suitable example.

Refer Principles of compiler design -aho , Page No.534

15. a) i) Explain principle sources of optimization.

Refer Principles of compiler design -aho , Page No.592

ii) Write short notes on.

1) Storage Organization

Refer Principles of compiler design -aho , Page No.396

2) Parameter Passing.

Refer Principles of compiler design -aho , Page No.424

(or)

b) i) Optimize the following code using various optimization technique.

```
    i=1;s=0;
for(i=1;i<=3;i++)
  for(j=1;j<=3;j++)
    c[i][j]=c[i][j]+a[i][j]+b[i][j]
```

Refer Class notes

ii) Write short notes on access to non-local names. **Refer Class notes**

B.E/B.Tech. DEGREE EXAMINATION, APRIL/MAY 2010.

Eighth semester

Information Technology

CS1352 – PRINCIPLES OF COMPILER DESIGN

(Common to Sixth Semester Computer Science and Engineering)

(Regulation 2004)

(Also Common to B.E.(Part Time) Fifth Semester, Regulation 2005)

Time : Three Hours **Maximum : 100 Marks**

PART A – (10 X 2 = 20 MARKS)

1. What are the two types of analysis of the source programs by compiler?
2. How is the token structure is specified?
3. Give two examples for each of top down parser and bottom up parser?
4. What is the significance of look ahead in LR(1) items?
5. What is the meaning of inherited translation?
6. What is meant by short circuit code?
7. What is meant by a basic block?
8. Which variable is to be made live and why?
9. Give any two examples for strength reduction.
10. What are the fields available in activation records?

PART B – (5 X 16 = 80 MARKS)

11. (a) (i) Discuss any four compiler construction tools. (8)

(ii) Write down the exclusive functions carried out by the analysis phases of any compiler. (8)

[OR]

(b) (i) Write briefly on the issues in input buffering and how they are handled. (8)

(ii) Give the rules for generating regular expressions for describing languages. (8)

12. (a) Construct predictive parsing table for the following grammar. Show how the string (a, a) is parsed by the predictive parser? (16)

S → a | ^ | (T)

T → T,SS

[OR]

(b) Construct the CLR parsing table for the following Grammar. (16)

S → CC

C → cC

C → d

13. (a) Describe the translation scheme for array translation and translate the following integer array operation into three address code.

A[i, j] = B[i, j] + C[k] where A and B are of 10 X 20 size and C contains 50 elements. (16)

[OR]

(b) Translate (a or b) and (c<d) and (d<e) into three address statements using back patching. (16)

14. (a) Write on the issues in code generation and generate the assembly language for the statement W:= (A + B) + (A + C) + (A + C) (8+8)

[OR]

(b) (i) Construct DAG for the following basic block. (8)

T_1 := A + B

T_2 := C + D

T_3 := E - T_2

T_4 := T_1 - T_3

(ii) Explain the techniques used in peephole optimization. (8)

15. (a) For the following program segment construct the flow graph and apply the possible optimizations.

```
sum := 0
I := 1;
do
        sum := prod + A[I] * B[I];
        I := I + 1;
while I <= 20;
```

[OR]

(b) Discuss the various storage allocation strategies. (16)