CS6660-COMPILER DESIGN

UNIT I

INTRODUCTION TO COMPILERS

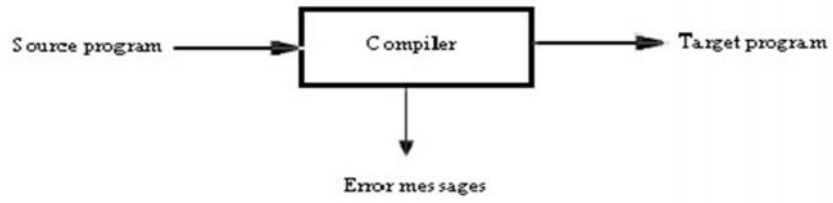
SYLLABUS: Translators-Compilation and Interpretation-Language processors -The Phases of Compiler-Errors Encountered in Different Phases-The Grouping of Phases-Compiler Construction Tools - Programming Language basics.

COURSE OBJECTIVE: To learn the basic concepts, phases and types of various translators along with several representations, specification and construction formats for language

PART – A

1. What does translator mean?
A translator is a program that takes a input program on one programming language (source language) and produces output in another language (object language or target language).

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



3. Give the analysis –synthesis model of compilation?
The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program. The synthesis part constructs the desired target program from the intermediate representation.

4. Define interpreter. (APRIL/MAY 2011)
Interpreter is a language processor program that translates and executes source code directly, without compiling it to machine code.

5. Define linker.
Linker is a program that combines (multiple) objects files to make an executable. It converts names of variables and functions to numbers (machine addresses).

6. Define editor.
Editors may operate on plain text, or they may be wired into the rest of the complier, highlighting syntax errors as you go, or allowing you to insert or delete entire syntax constructed at a time.

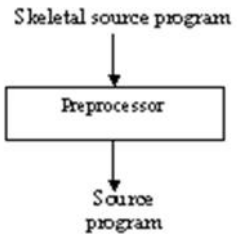
7. Define profiler.
Profiler is a program to help you see where your program is spending its time, so you can tell where you need to speed it up.

8. What is literal table?
Literal table is a table that stores "literal" values, mainly strings and variables names that may occur in many places. Only one copy of each unique string and name needs to be allocated in memory.

9. Define Assembler.
Assembler is a translator from human readable (ASCII text) files of machine instructions into the actual binary code (object files) of a machine.

10. Define Loader.
Loader is a program that performs the functions of loading and linkage editing. The process of loading consists of taking re-locatable machine code, altering the re-locatable address and placing the altered instruction and data in memory at the proper location.

11. Define Preprocessor. What are its functions?
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.



The functions of a preprocessor are:

- Macro processing.
- File inclusion.
- Rational preprocessors.
- Language extensions

.

12. Define Debugger.
Debugger is a program to help you see what is going on when your program runs. It can print the values of variables, show what procedure called what procedure to get where you are, run up to a particular line, run until a particular variables gets a special value etc.

13. What is front-end and back-end of the compiler?
Often the phases of a compiler are collected into a front-end and back-end. Front-end consists of those phases that depend primarily on the source program and largely independent of the target machine. Back-end consists

of those phases that depend on the target machine language and generally those portions do not depend on the source language, just the intermediate language. In back end we use aspects of code optimization, code generation, along with error handling and symbol table operations.

14. Define Passes.

In an implementation of a compiler, portion of one or more phases are combined into a module called pass. A pass reads the source program or the output of the previous pass, makes the transformation specified by its phases and writes output into an intermediate file, which is read by subsequent pass.

15. What is the main difference between phase and pass of a compiler?

A phase is a sub process of the compilation process whereas combination of one or more phases into a module is called pass.

16. Define syntax and semantics.

The rules and regulations used to form a language are known as syntax. The meaning given to a programming construct is known as semantics.

17. What are the classifications of a compiler?

The classifications of compiler are:

- Single-pass compiler.
- Multi-pass compiler.
- Load and go compiler.
- Debugging compiler.
- Optimizing compiler.

18. What is linear analysis?

The stream of characters making up the source program is read from left-toright and grouped into tokens that are sequence of characters having a collective meaning.

19. What are the phases of a compiler?

- Lexical analysis phase or scanning phase
- Syntax analysis phase
- Intermediate code generation
- Code optimization
- Code generation

20. Name minimum 4 compiler construction tools?(Nov/Dec-16)

- LEX – Scanner generator
- YACC-Parser generator
- Syntax directed translation scheme.
- Automatic code generator
- Data flow engines

21. Mention few cousins of compiler. (MAY/JUNE 2012)

- Preprocessors

- Assemblers
- Loaders
- Link editors
-

22. What are the Error-recovery actions in a lexical analyzer? (APRIL/MAY 2015)

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

23. What are the two parts of a compilation ? Explain briefly(MAY/JUNE 2016)

The two main parts are

- **Analysis**
- **Synthesis**

• **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

24. Illustrate diagrammatically how a language is processed(MAY/JUNE 2016)

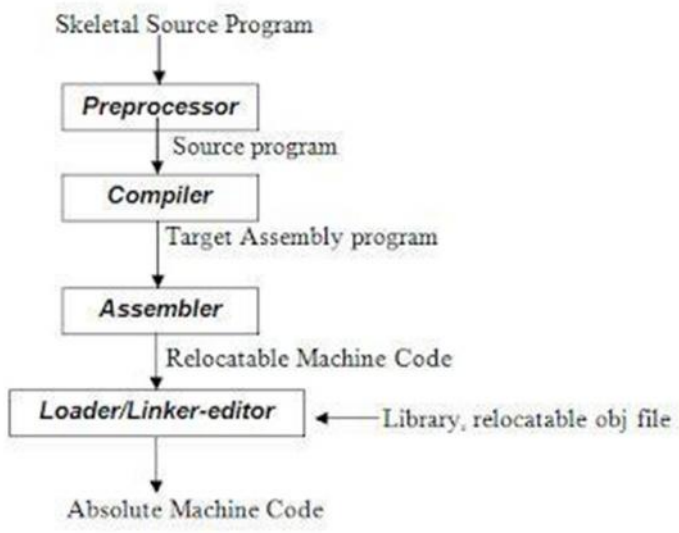


Fig 1.1 Language –processing System

PART – B

1. Describe phases of compilers with an example? (April/May 2011, May/June 2012) **(Nov/Dec-16)** (U)
2. Explain in detail about cousins of compiler? (Nov/Dec 2013)(U)
3. Write short notes on compiler construction tools? (April/May 2011, May/June 2012)**(Nov/Dec-16)** (U)
4. Explain in details about Grouping of phases. (Nov/Dec 2014) **(Nov/Dec-16)** (U)
5. Define the terms : Compiler, Interpreter, Translator and differentiate between them. (MAY/JUNE 2014)(U)
6. Describe error recovery schemes in lexical phase of the compiler. (U) (APRIL/MAY 2015)
7. Describe the various phases of compiler and trace it with the program segment (position:=initial+rate*60). (MAY/JUNE-16) (Ap)
8. (i) Explain language processing system with neat diagram(U)

   (ii) Explain the need for grouping phases.(U)

   (iii) Explain various Error encountered in different phases of compiler. (May/June-16) **(Nov/Dec-16)** (R)

COURSE OUTCOME: Acquire knowledge in different phases and passes of Compiler, and specifying different types of tokens by lexical analyzer.

UNIT II

LEXICAL ANALYSIS

SYLLABUS: Need and Role of Lexical Analyzer-Lexical Errors-Expressing Tokens by Regular Expressions-Converting Regular Expression to DFA- Minimization of DFA-Language for Specifying Lexical Analyzers-LEX-Design of Lexical Analyzer for a sample Language.

COURSE OBJECTIVE: To design and implement a lexical analyzer

PART-A

1. What is the role of lexical analysis phase? (Nov/Dec 2014)
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.

2. Define lexeme? (MAY/JUNE 2014)
The character sequence forming a token is called lexeme for the token.

3. 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.

4. 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.

5. 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).

6. List the rules for constructing regular expressions?

The rules are divided into two major classifications

- Basic rules
- Induction rules

Basic rules:

i) € is a regular expression denoting {€} (i.e.) the language contains only an empty string.
ii) For each a in ∑, a is a regular expression denoting {a}, the language with only one string , which consists of a single symbol a.

Induction rules:

i) If R and S are regular expressions denoting languages LR and LS respectively then,
ii)(R)/(S) is a regular expression denoting L(R) U L(S)
iii)(R).(S) is a regular expression denoting L(R) . L(S)
iv)(R)* is a regular expression denoting L(R)*.

7. 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.

8. Write the algebraic laws obeyed by the regular expression.

For any regular expressions R, S, T the following laws hold:
i. R|S = S|R (commutative)
ii. R|(S|T) = (R|S)|T (associative)
iii. R.(S.T) = (R.S).T (associative)
iv. R(S|T) = RS|RT (distributive)

9. Define DFA.

A DFA is an acceptor for which any state and input character has utmost one transition state that the acceptor changes to. If no transition state is specified the input string is rejected.

10. 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.

10. What is symbol table? What are its contents? (May/June 2014)(Nov/Dec-16)

Symbol table is a data structure that contains all variables in the program and temporary storage and any information needed to reference or allocate storage for them. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

11. What is the demerit in uniform structure of symbol table?

- Length of the name should not exceed upper bound or limit of name field.
- If length of name is small, then remaining space was wasted.

12. What are semantic errors? Give example.

The error s detectable by the compiler is termed as semantic error. These errors are detected both at compile time and runtime. Semantic errors that can be detected at compile time errors of declaration and scope.
Semantic errors that can be detected at runtime are range-checking foe certain values, particularly array subscripts and case statement selectors.

13. What are the drawbacks of using buffer pairs?

- This buffering scheme works quite well most of the time but with it amount of lookahead is limited.
- Limited look ahead makes it impossible to recognize tokens in situations where the distance, forward pointer must travel is more than the length of buffer.

14. Define NFA.

A NFA is an acceptor for which any state and input character has more than one transition state that the acceptor changes to. If no transition state is specified the input string is rejected.

15. What is meant by epsilon move?

It is a finite automata with empty move. i.e. NFA with empty move. If we want to give transition without taking any input symbol then epsilon move is used. It is denoted by the symbol €.

16. What is the need for separating the analysis phase into lexical analysis and parsing? (Or) What are the issues of lexical analyzer? (MAY/JUNE 2012)

- 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.

17. Write a grammar for branching statements.

- S → if E then S_1
- S → if E then S_1 else S_2
- S → while E do S_1

18. List the Operations on languages(MAY/JUNE 2016)

- **Union -** L U M ={s | s is in L or s is in M}
- **Concatenation** – LM ={st | s is in L and t is in M}
- **Kleene Closure –** L* (zero or more concatenations of L)
- **Positive Closure –** L+ (one or more concatenations of L)

19. Differetiate Between lexeme,token and pattern.(Nov/Dec-16)

Tokens- Sequence of characters that have a collective meaning.
Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token
Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.

PART – B

1. Explain the role of the lexical analyzer? **(Nov/Dec-16)** (U)
2. Write short notes on input buffering? (R)
3. Design a lexical analyzer generator?(Ap)
4. Construct NFA,DFA from NFA and optimized DFA for the following
    1. aa*|bb*
    2. (a|b)*abb
    3. (a*|b*)*
    4. 0(0|1)*0
    5. ((ab)*)*
    6. (011)*001
    7. ((€/a)b*)* (NOVEMBER/DECEMBER 2012)
    8. ab*/ab (MAY/JUNE 2014) (Ap**) (Nov/Dec-16)**
5. Design a Lexical analyzer for a sample language. (Ap)
6. Discuss about LEX tool.?(U)
7. (i) What are the issues in lexical analysis? (R))
   (ii)Elaborate in detail the recognition of tokens. (MAY/JUNE 2012) (U)

8. (i) Differetiate Between lexeme,token and pattern.(An)

   (ii) What are the issues in lexical analysis? (R)

   (iii) Write notes on regular expressions(MAY/JUNE 2016). (U)

9. (i) Write notes on regular expression to NFA.(Un) Construct Regular expression to NFA for the sentence(a|b)*a. (Ap)

   (ii) Construct DFA to recognize the language (a/b)*ab. (MAY/JUNE 2016).(Ap)

10. Discuss how finite automata is used to represent tokens and perform lexical analysis with example **(Nov/Dec-16)** (U)
11. Write an algorithm for minimizing the number of states of a DFA(Nov/Dec-16) (U)

COURSE OUTCOME: Can be able to use the Compiler tools like LEX, YACC, etc

UNIT III

SYNTAX ANALYSIS

SYLLABUS: Need and Role of the Parser-Context Free Grammars -Top Down Parsing -General Strategies-Recursive Descent Parser Predictive Parser-LL(1) Parser-Shift Reduce Parser-LR Parser-LR (0)Item-Construction of SLR Parsing Table -Introduction to LALR Parser - Error Handling and Recovery in Syntax Analyzer-YACC-Design of a syntax Analyzer for a Sample Language

COURSE OBJECTIVE: To learn the role of a parser and to study the different ways of recognizing and parsing of tokens

PART – A

1. What do you mean by context free grammar?

Context free grammar is a notation for specifying the syntax of a language.
A context free grammar has four components:
i. A set of tokens, known as terminal symbols.
ii. A set of non-terminals.
iii. A set of productions where each production consists of a non-terminal, called the left side of the production, an arrow, and a sequence of tokens and/or non-terminals, called the right side of the production.
iv. A designation of one of the non-terminals as the start symbol.

2. Define terminal and non-terminal.

Terminals are the basic symbols from which the strings are formed.
Nonterminal are syntactic variables that denote set of strings.

3. What are the advantages of Grammar?

- Easy to understand.
- Imparts a structure to a programming language.
- Acquiring new constructs.

4. What are the different strategies that a parser can recover from a syntactic error?

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

5. Define ambiguous grammar, and specify it demerits. (May/June-16)

If a grammar produces more than one parse tree for the given input string then it is called ambiguous grammar. Its demerit is difficult to select or determine which parse tree is suitable for an input string.

6. Mention the properties of parse tree?

- The root is labeled by the start symbol.
- Each leaf is labeled by a token.
- 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

7. When does a parser detect an error?

A parser detects an error when it has no legal move from its current configuration, which is determined by it state, its stack contents and the current input symbol. To recover from an error a parser should ideally locate the position of the error, correct the error, revise its current configuration, and resume parsing.

8. What is meant by left most derivation?

In derivation, only the leftmost nonterminal in any sentential form is replaced at each step. Such derivations are termed leftmost derivation.

9. Define left factoring.

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

10. Define predictive parsing.

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

11. What you mean by nonrecursive predictive parser?

A nonrecursive predictive parser is a program that matches the terminal symbols against the input by maintaining a stack rather than using recursive calls.

12. What is meant by shift-reduce parsing?

Shift reduce parsing constructs a parse tree for an input string beginning at the leaves and working up towards the root i.e., " reducing" a string w to the start symbol of a grammar.

13. Define Handle.

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

14. What are the possible actions of a shift reduce parser?

The possible actions are:

- Shift.
- Reduce.
- Accept.
- Error.

15. Define 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.

16. List down the conflicts during shift-reduce parsing.

- Shift/reduce conflict.

Parser cannot decide whether to shift or reduce.

- Reduce/reduce conflict.

Parser cannot decide which of the several reductions to make.

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

A grammar is operator grammar if,

i. No production rule involves "ε**"** on the right side.

ii. No production has two adjacent nonterminal on the right side.. Ex:

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

18. What are the disadvantages of operator precedence parsing?(May/June 2007)

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.

19. 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.

20. Differentiate Kernel and non-Kernel items.

Kernel items, which include the initial item, S' -> .S and all items whose dots are not at the left end. Whereas the non-kernel items have their dots at the left end.

21. What are the components of LR parser?

The components of LR parser are:

- An input.
- An output.
- A stack.
- A driver program.
- A parsing table.

22. List the different techniques to construct an LR parsing table?

The different techniques to construct an LR parsing table are:

- Simple LR(SLR).
- Canonical LR.
- Lookahead LR (LALR).

23. Define LR (0) item.

An LR(0) item of a grammar G is a production of G with a dot at some position of the right side.

Eg: A->. XYZ

A->X .YZ

A->XY. Z

A->XYZ .

24. Compare production with reduction.

The rules that define the ways in which the syntactic categories can be built are productions whereas the replacement of the string by an non-terminal according to a grammar production is called reduction.

25. Define handle pruning.(Nov/Dec-16)

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.

(OR)

What is meant by handle pruning?

A rightmost derivation in reverse can be obtained by handle pruning.

If w is a sentence of the grammar at hand, then w = γn, where γn is the nth rightsentential form of some as yet unknown rightmost derivation

S = γ0 => γ1…=> γn-1 => γn = w

26. List various types of grammars.

- Phase sensitive grammar
- Context sensitive grammar
- Context-free grammar
- Regular grammar

27. 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 superset of the class of grammar.LR parser can detect syntactic error as soon as possible, when left-to-right scan of the input.

28. Specify the advantages of LALR.

Merging of states with common cores can never produce a shift/reduce conflict that was not present in any one of the original states. Because shift actions depends only one core, not the look ahead.

29. 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.

30. What is meant by YACC?

YACC is a parser tool. It is Yet Another Compiler Compiler. It is used to generate the Parser automatically.

31. Write the Algorithm for FIRST and FOLLOW in parser.(MAY/JUNE 2016)

FIRST

1. If X is terminal, then FIRST(X) IS {X}.
2. If X → e is a production, then add e to FIRST(X).
3. If X is non terminal and X → Y1,Y2..Yk is a production, then place a in FIRST(X) if for some i , a is in FIRST(Yi) , and e is in all of FIRST(Y1),…FIRST(Yi-1);

FOLLOW

1. Place $ in FOLLOW(S),where S is the start symbol and $ is the input right endmarker.
2. If there is a production A → aBß, then everything in FIRST(ß) except for e is placed in FOLLOW(B).
3. If there is a production A → aB, or a production A→ aBß where FIRST(ß) contains e , then everything in FOLLOW(A) is in FOLLOW(B).

32. Construct a parse tree for -(id+id) (Nov/Dec-16)

parse tree

PART – B

1. Explain the role of parser? (U)
2. Eliminate left recursion from the following grammar:

   S->Aa|b

   A->Ac|Sd| ε (Ap)
3. Generate SLR parse table for the grammar (Ap)

S->Aa|bAc|Bcb|Ba

A->d

B->d

And parse the sentence “bdc” and “dd” **(April/May 2015)**

4. Draw and discuss the model of Non-Recursive predictive parsing? (U)

5. Consider the grammar

E->E+T|T

T->T*F|F

F->(E)|id

Construct predictive parser table and validate the input string id*id+(id*+id) (Ap)

6. Write the stack implementation of shift-reduce parsing? (U)

7. Draw and discuss the model of LR parser? (U)

8. Construct operator precedence functions for the operators +,*,id,$. (Ap)

9. Every SLR (1) grammar is unambiguous, but there are many unambiguous grammars that are not SLR(1).Justify it.(E)

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

S →CC, C→cC|d (Ap)

11. i) Give an algorithm for finding the FIRST and FOLLOW positions for a given non-terminal. (An)

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. (Ap)

12. Construct the predictive parser for the following grammar**. (May/June 2012)**
S->(L)/a , L->L,S/S (Ap)

13. Describe the conflicts that may occur during shift reduce parsing.(U) **(May/June 2012)**

14. (i) Construct Sack implementation of shift reduce parsing for the grammar

E → E+E
E→ E*E
E→(E)
E→ id and the input string id1+id2*id3 (Ap)

(ii) Explain LL(1) grammar for the sentences S→iEts|iEtSeS |a E→ b.**(May/June 2016)**(U)

15. (i)Write an a algorithm for Non recursive predictive parsing. (U)
(ii)Explain Context free grammars with examples **(May/June 2016)** (U)

16.(i) Construct parse tree for the input string **w = cad** using top down parser. (Ap) **(Nov/Dec-16)**

S→cAd

A→ab|a

(ii) Construct parsing table for the grammar and find moves made by predictive parser on input id * id + id and find FIRST and FOLLOW **(Nov/Dec-16)** (Ap)

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

17.(i) Explain ambiguous grammer G:E→E +E | E* E | (E) | -E | ID for the sentence id+id*id.(U)
(ii) Construct SLR parsing table for the following grammer: **(Nov/Dec-16)** (Ap)

G: E→ E +T|TT →T*F|FF → (E)| id.

COURSE OUTCOME: Acquire knowledge about top-down parsing and bottom-up parsing and able to construct parse table.

UNIT IV

SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT

SYLLABUS: Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-Attribute Definitions- Design of predictive translator - Type Systems-Specification of a simple type checker-Equivalence of Type Expressions-Type Conversions.

RUN-TIME ENVIRONMENT: Source Language Issues-Storage Organization-Storage Allocation-Parameter Passing-Symbol Tables-Dynamic Storage Allocation-Storage Allocation in FORTAN.

COURSE OBJECTIVE: To learn about syntax directed translation and intermediate languages, and able to know about the various storage allocation strategies

PART – A

1. What is mean by syntax directed definition?
It is a generalization of a CFG in which each grammar symbol has an associated set of attributes like, synthesized attribute and inherited attribute.

2. What is synthesized attributes?
An attribute is said to be synthesized, if its value at a parse tree node is determined from the attribute values at the children.

3. What is inherited attributes?
An inherited attribute is one whose value at a node in a parse tree is defined in terms of attributes at the parent and/or siblings of that node.

4. What is annotated parse tree?
A parse tree showing the values of attributes at each node is called an annotated parse tree. The process of computing the attribute values at the nodes is called an annotating or decorating the parse tree.

5. List three kinds of intermediate representation?
The three kinds of intermediate representations are:

- Syntax tree
- Postfix notation
- Three address code.

6. Define syntax tree representation?
An abstract or syntax tree is a condensed form of parse tree useful for representing language constructs. A syntax tree depicts the natural hierarchical structure of a source program.

7. Explain postfix notation representation?
It is the linearized representation of syntax tree .It is a list of nodes of the syntax tree in which a node appears immediately after its children.

8. Define three address code representations?
Three –address code is a sequence of statements of the form x: =y op z, Where x, y, z are names, constants or compiler generated temporaries and op stands for an operator. Since a statement involves not more than three references, it is called three address statements and hence a sequence of such statement is called three address codes.

9. List three types of representations of three address statements? (May/June 2014)

- Quadruples
- Triples
- Indirect triples.

10. Define quadruple and give one example?
A quadruple is a data structure with 4 fields like operator, argument-1, argument-2 and result. Example: a=b*-c

11. Define triple and give one example?
A triple is a data structure with 3 fields like operator, argument-1, and argument-2.
Example: a=b*-c

12. Define back patching. (May/June 2014)
To generate three address codes, 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 back patching.

13. What are the three functions used for back patching?

Three functions used in back patching are

- Make list (i).
- Merge (p1, p2).
- Back patch (p, i).

14. 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

15. 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.

16. Which information's are entered into the symbol table?

The information entered in the symbol table are

- The string of characters denoting the name.
- Attributes of the name.
- Parameters.
- An offset describing the position in storage to be allocated for the name.

17. What are the different data structures used for symbol table?

The data structures used in symbol table are

- Lists
- Self-organizing lists.
- Search tree.
- Hash table

18. Define Boolean expression?

Boolean expressions are composed of the Boolean operators (and, or, not) applied to elements. That is Boolean variables or relational expressions.

19. Define short-circuit code?

It translates a Boolean expression into three-address code without generating code for any of the Boolean operators and without having the node necessarily evaluate the entire expression. This type of evaluation is called short circuit or jumping code.

20. What is the need for type checking and type analysis?

The semantic analysis is done in order to obtain the precise meaning of programming construct. The need for semantic analysis is to build a symbol table that keeps track of names established in declarations. The data type of identifier is obtained and type checking of the whole expression and statements is done in order to follow the type rules of the language. It also identifies the scope of identifiers.

21. What do you meant by type expressions?

The systematic way of expressing type of language construct is called type expression. It can be defined as the basic type is called type expression. Hence int, char, float, double, enum are type expressions.

22. What is meant by type conversion?

It is the process of converting one type to another. There are two types of conversion. Implicit conversion and explicit conversion. If the conversion is done automatically by the compiler then it is called implicit conversion. If the conversion is done by the programmer by writing something for converting one type to another then it is called explicit conversion.

23. Explain with some suitable example type checker.

In this language construct the type of the identifier must be declared before the use of that identifier. The type checker is a translator scheme in which the type of each expression from the types of sub expressions is obtained. The type checker can decide the types for arrays, pointers, statements and functions.

24. What are the source language issues?

The following are the source language issues:

1. Recursion
2. How the parameters are passed to the procedure.
3. Does the procedure refer nonlocal names

25. What are the limitations of static allocation? (April/May-11, Nov/Dec-06, 07)

- The size of data objects is known at compile time.
- It uses static allocation
- The compiler can determine the amount of storage required by each data object.

26. Draw the diagram of the activation record and give purpose of any two fields. (Or) Define activation tree.

The activation record is a block of memory used for managing information needed by a single execution of a procedure.

27. Define actual parameters?

The arguments passed to a called procedure are known as actual parameters.

28. Define activation tree?

An activation tree depicts the way control enters and leaves activations.

29. Define activation record? (Nov/Dec 2013)

Information needed by a single execution of a procedure is managed using a contiguous block of storage called an activation record.

Various fields of activation records:

- Temporary values
- Local values
- Saved machine registers
- Control link
- Access link
- Actual parameters
- Return values

30. What are the different storage allocation strategies?

1) Static allocation.
It lays out storage for all data objects at compile time.
2) Stack allocation.
It manages the runtime storage as a stack.
3) Heap allocation.
It allocates and deallocates storage as needed at runtime from a data area.

31. What do you mean by dangling reference? (MAY/JUNE 2012)
A dangling reference occurs when there is a reference to storage that has been deallocated.

32. When does Dangling references occur?(MAY/JUNE 2016)
Whenever storage is deallocated, the problem of dangling references arises
- Occurs when there is a reference to storage that has been deallocated
- Logical error

(OR)

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.

33. Write down syntax directed definition of a simple desk calculator.(Nov/Dec-16)
Syntax directed definition specifies the values of attributes by associating semantic rules with the grammar productions. **Syntax directed definition of simple desk calculator** :

Production	Semantic rules
L ---> E_n	*L.val = E.val*
E ---> E_1+ T	*E.val = E_1.val+ T.val*
E ---> T	*E.val = T.val*
T---> T_1*F	*T.val = T_i.val x F.val*
T ---> F	*T.val = F.val*
F ---> (E)	*F.val = E.val*
F ---> digit	*F.val = digit.lexval*

34. List Dynamic Storage allocation Techniques.(Nov/Dec-16)

- **Stack**: Manage activation of procedures at runtime.
- **Heap:** holds variables created dynamically

PART – B

1. Translate the arithmetic expression a*-(b+c) into

a. Syntax tree (Ap)

b. Postfix Notation (An)

c. Three-Address code. (An)

2. Write syntax directed definition for flow-of-control statements? (R)

3. Write syntax directed definition for Booleans? (R)

4. Write syntax directed definition for case statements?(R) **(Nov/Dec 2014)**

5. Explain procedure calls with neat example? (U)

6. Explain in detail about the storage organization. (U) **(April/May 2015)**

7. What are different storage allocation strategies? Explain. (U) **(Apr/May 2011, Nov/Dec 2013)**

8. How names can be looked up in the symbol table? Discuss. (U)**(May/June-12)**

9. Explain about source language issues? (U)

10. Explain in detail about parameters passing methods? (U) **(Nov/Dec 2013)**

11. i) Explain the specification for a simple type checker. (U)

ii) Explain run time environment with suitable example. (U) **(Nov/Dec 2014)**

12. (i) Construct a syntax directed definition for constructing a syntax tree for assignment statements.

S→id:=E

E→ E1+E2

E→ E1*E2

E→-E1

E→(E1)

E→id (Ap)

(ii) Discuss specification of a simple type checker. (An) **(May/June-16)**

OR

13. Discuss different storage allocation strategies. (U) **(May/June-16)**

14. A Syntax-Directed Translation scheme that takes strings of a's ,b's and c's as input and produces as output the number of substrings in the input and produces as output the number of substrings in the input string that correspond to pattern a(a|b)*c+(a|b)*b.

For example the translation of the input string "abbcabcababc" is "3".

(1) Write a context-free grammar that generate all strings of a's b's and c's. (U)

(2) Give the semantic attributes for the grammar symbols. (U)

(3) For each production of the grammar present a set of rules for evaluation of the semantic attributes. (U)

(4) Illustrate type checking with necessary diagram. (U) **(Nov/Dec-16)**

15. Explain the following with respect to code generation phase.

(i) Input to code generator

(ii) Target program

(iii) Memory management

(iv) Instruction selection

(v) Register allocation

(vi) Evaluation order. **(Nov/Dec-16)** (U)

COURSE OUTCOME: Can be able to translate the statement and implement the storage allocation strategies.

UNIT – V

CODE OPTIMIZATION & CODE GENERATION

SYLLABUS: Principal Sources of Optimization-DAG- Optimization of Basic Blocks-Global Data Flow Analysis-Efficient Data Flow Algorithms-Issues in Design of a Code Generator - A Simple Code Generator Algorithm

COURSE OBJECTIVE: To study the concepts of code generation and concepts of Code Optimization and about various code improving transformations.

PART – A

1. 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.

2. Define Target machine?

The target computer is byte-addressable machine with four bytes to a word and n-general purpose registers. R0, R1… … … .Rn-1. It has two address instructions of the form Op, source, destination in which Op is an op-code, and source and destination are data fields.

3. 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.

4. Define basic block? (Nov/Dec 2014)

A basic block contains sequence of consecutive statements, which may be entered only at the beginning and when it is entered it is executed in sequence without halt or possibility of branch.

5. Define flow graph? (Nov/Dec 2014)

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

6. What do you mean by DAG?(May/June-16)

It is Directed Acyclic Graph. In this common sub expressions are eliminated. So it is a compact way of representation.

7. Specify some advantages of DAGs? (April/May 2015)

- It automatically detects common sub expression.
- We can determine which identifiers have their values used in the block.
- We can determine which statements compute values, and which could be used outside the block.
- It reconstruct a simplified list of quadruples taking advantage of common sub expressions and not performs assignments of the form a=b unless necessary.

8. What are the characteristics of peephole optimization?

- Redundant instruction elimination.
- Flow of control optimization
- Algebraic simplifications
- Use of machine idioms

9. 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, which is useful for intermediate representation.

10. 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.

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 meant by U-D chaining?

It is Use-Definition chaining. It is the process of gathering information about global data flow analysis can be used id called as use-definition (ud) chaining.

14. What do you mean by induction variable elimination?

It is the process of eliminating all the induction variables, except one when there are two or more induction variables available in a loop is called induction variable elimination.

15. What are dominators?

A node of flow graph is said to be a dominator, i.e. one node dominates the other node if every path from the initial node of the flow graph to that node goes through the first node.(d Dom n).when d-node dominates n-node.

16. What is meant by constant folding?

Constant folding is the process of replacing expressions by their value if the value can be computed at complex time.

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 are the various types of optimization?

The various type of optimization is,

- Local optimization
- Loop optimization
- Data flow analysis
- Global optimization

19. List the criteria for selecting a code optimization technique?

The criteria for selecting a good code optimization technique are,

- It should capture most of the potential improvement without an unreasonable amount of effort.
- It should preserve the meaning of the program
- It should reduce the time or space taken by the object program.

20. What is data flow analysis? (Nov/Dec 2012)

The data flow analysis is the transmission of useful relationships from all parts of the program to the places where the information can be of use.

21. Define code motion and loop-variant computation?

Code motion: It is the process of taking a computation that yields the same result independent of the number of times through the loops and placing it before the loop.Loop –variant computation: It is eliminating all the

induction variables, except one when there are two or more induction variables available in a loop represent, not locations in memory, but logical signals (0 or 1) or group of signals in a switching circuit. The output is a circuit design in an appropriate language.

22. How do you calculate the cost of an instruction?

We take the cost of an instruction to be one plus the costs associated with the source and destination address modes. This cost corresponds to the length (in words) of the instruction. Address modes involving registers have cost zero, while those with a memory location or literal in them have cost one, because such operands have to be stored with the instruction.

23. What is meant by optimization?

It is a program transformation that made the code produced by compiling algorithms run faster or takes less space.

24. Define optimizing compilers?

Compilers that apply code-improving transformations are called optimizing compilers.

25. When do you say a transformation of a program is local?

A transformation of a program is called local, if it can be performed by looking only at the statement in a basic block.

26. Define common sub expression?

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

27. Define live variable? (Nov/Dec 2012)

A variable is live at a point in a program if its value can be used subsequently.

28. Define formal parameters?

The identifiers appearing in procedure definitions are special and are called formal parameters.

29. What do you mean by data flow equations?

A typical equation has the form

out[s] = gen[s] U (in[s]-kill[s])

It can be read as information at the end of a statement is either generated within the statement or enters at the beginning and is not killed as control flows through the statement.

30. State the meaning of in[s], out[s], kill[s], gen[s]?

in[s]-The set of definitions reaching the beginning of S.
out[s]-End of S.
gen [s]-The set of definitions generated by S.
kill[s]-The set of definitions that never reach the end of S.

31. What are the techniques used for loop optimization? (MAY/JUNE 2014)

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

32. What are the properties of optimizing compiler? (MAY/JUNE 2016)

(i) Transformation must preserve the meaning of programs.
(ii) Transformation must, on the average, speed up the programs by a measurable amount
(iii)A Transformation must be worth the effort.

33. Write three address code sequence for the assignment statement d:=(a-b) +(a-c) +(a-c). (MAY/JUNE 2016)

Statement	Code generation	Register descriptor	Address descriptor
t:=a-b	MOV a,R0 SUB b,R0	R0 contains t	t in R0
u:=a-c	MOV a,R1 SUB c,R1	R0 contains t R1 contains u	t in R0 u in R1
v:=t+u	ADD R1,R0	R0 contains v R1 contains u	u in R1 v in R0
d:=v+u	ADD R1,R0 MOV R0,d	R0 contains d	d in R0 d in R0 and memory

34. Identify the constructs for optimization in basic block.(Nov/Dec-16)

Optimization Of Basic Blocks

There are two types of basic block optimizations. They are :

- Structure-Preserving Transformations
- Algebraic Transformations

35. Write the characteristics of peephole optimization? (Nov/Dec-16)

- Redundant-instruction elimination
- Flow-of-control optimizations.
- Algebraic simplifications
- Use of machine idioms

PART – B

1. Explain the various issues involved in the design of a code generator? (U) **(May/June 2014)**
2. Describe in detail about the various allocation strategies in memory management? (U)
3. Design a simple code generator? (Ap)
4. Explain in detail about register allocation and assignment? (U) (**Nov/Dec 2012**)
5. Discuss briefly about DAG representation of basic blocks? (U)
6. Construct DAG and optimal target code for the expression

   X=((a+b)/(b-c)-(a+b)*(b-c)+f) **(April/May 2015)** (Ap)
7. Explain the algorithm to partition into basic block? (U)

8. Construct the DAG for the following basic block

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

9. Explain the various transformations can be applied to basic block? (**May/June 2014**) (U)

10. Explain the principal sources of optimization?(U)

11. Write short notes on

A.Lexical scope. B.Dynamic scope.(U)

12. How do you reduce a flow graph to get optimized code? (An)

13. Explain about global flow analysis? (**May/June 2014, Nov/Dec 2013**) (U)

14.Explain Principal sources of optimization with examples. **(May/june-16)** (U)

15.(i)Explain various issues in the design of code generator (**May/june-16)** (U)

(ii)Write note on simple code generator. (**May/june-16)** (U)

16.(i) Write an algorithm for constructing natural loop of a back edge . **(Nov/Dec-16)** (U)
(ii) Explain any four issues that crop up when designing a code generator (**Nov/Dec-16)** (U)

17. Explain global data flow analysis with necessary equations. **(Nov/Dec-16)** (U)

COURSE OUTCOME: Can be able to apply the various optimization techniques

COURSE OUTCOMES

COURSE NAME : CS6660-COMPILER DESIGN

YEAR/SEMESTER : III / VI

YEAR OF STUDY : 2016 –2017 EVEN (R – 2013)

On Completion of this course student will gain

C313.1	**An Ability to Acquire knowledge in different phases and passes of Compiler, and specifying different types of tokens by lexical analyzer**
C313.2	**An Ability to use the Compiler tools like LEX, YACC, etc.**
C313.3	**An Ability to acquire knowledge about top-down parsing and bottom-up parsing and able to construct parse table.**
C313.4	**An Ability to translate the statement and implement the storage allocation strategies.**
C313.5	**An Ability to apply the various optimization techniques.**

CO-PO MATRICES

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C313.1	3	1	-	-	-	-	-	-	-	-	-	-
C313.2	2	3	-	-	3	-	-	-	-	-	-	-
C313.3	3	3	-	-	3	-	-	-	-	-	-	-
C313.4	3	3	-	-	3	-	-	-	-	-	-	-
C313.5	2	2	-	-	1	-	-	-	-	-	-	-
C313	2.6	2.4	-	-	2.5	-	-	-	-	-	-	-

CO-PSO MATRIC OF COURSE

CO	PSO1	PSO2	PSO3
C313.1	-	2	-
C313.2	-	2	-
C313.3	-	-	2
C313.4	-	2	-
C313.5	-	-	2
C313	-	2	2