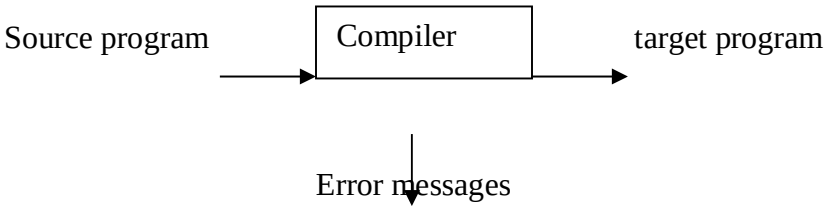
PRINCIPLES OF COMPILER CONSTRUCTION

UNIT I

2 MARK QUESTIONS WITH ANSWERS

1. 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



2. State some software tools that manipulate source program?

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

3. What are the cousins of compiler? April/May 2004, April/May 2005

The following are the cousins of compilers

i. Preprocessors
ii. Assemblers
iii. Loaders
iv. Link editors.

4. 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

5. 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 editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.

6. 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.

7. How many phases does analysis consists?

Analysis consists of three phases

i .Linear analysis

ii .Hierarchical analysis

iii. Semantic analysis

8. 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.

9. What happens in Hierarchical analysis?

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

10. 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.

11. State some compiler construction tools?Arpil /May 2008

i. Parse generator

ii. Scanner generators

iii. Syntax-directed translation engines

iv. Automatic code generator

v. Data flow engines.

12. 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 relocatable address and placing the altered instructions and data in memory at the proper locations.

13. What does the Link Editing does?

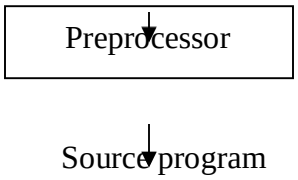
Link editing: This allows us to make a single program from several files of relocatable machine code. These files may have been the result of several compilations, and one or more may be library files of routines provided by the system and available to any program that needs them.

14. What is a preprocessor? Nov/Dev 2004

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



15. State some functions of Preprocessors

i) Macro processing
ii) File inclusion
iii) Relational Preprocessors
iv) Language extensions

16. 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.

17. 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

18. What is an assembler?

Assembler is a program, which converts the source language in to assembly language.

19. 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.

20. What is Lexical Analysis?

The first phase of compiler is Lexical Analysis. This is also known as linear analysis in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.

21. What is a lexeme? Define a regular set. Nov/Dec 2006

- 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

22. What is a sentinel? What is its usage? April/May 2004

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.

23. 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).

24. 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

25. Construct Regular expression for the language

L= {w ε{a,b}/w ends in abb}

Ans: {a/b}*abb.

26. What is recognizer?

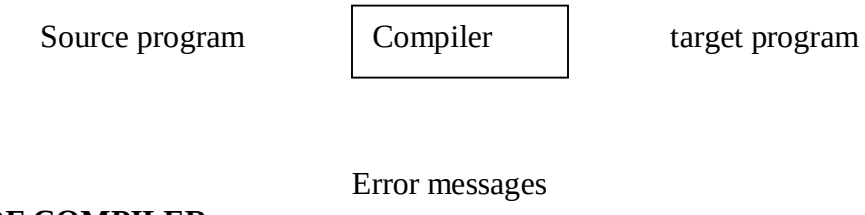
Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine, otherwise it is rejected.

PART-B QUESTIONS (16 MARKS)

27. What is a compiler? State various phases of a compiler and explain them in detail.(16)

COMPILER

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



PHASES OF COMPILER

A Compiler operates in phases, each of which transforms the source program from one representation into another. The following are the phases of the compiler:

Main phases:
1) Lexical analysis
2)Syntax analysis
3)Semantic analysis
4) Intermediate code generation
5)Code optimization
6)Code generation

Sub-Phases:
1) Symbol table management
2)Error handling

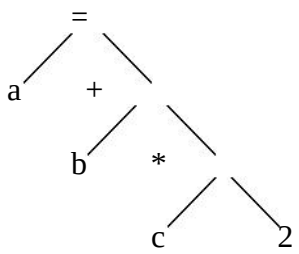
LEXICAL ANALYSIS:

- It is the first phase of the compiler. It gets input from the source program and produces tokens as output.
- It reads the characters one by one, starting from left to right and forms the tokens.
- **Token** : It represents a logically cohesive sequence of characters such as keywords, operators, identifiers, special symbols etc.
  Example: a +b =20
  Here, a,b,+,=,20 are all separate tokens.
  Group of characters forming a token is called the **Lexeme**.

- The lexical analyser not only generates a token but also enters the lexeme into the symbol table if it is not already there.

SYNTAX ANALYSIS:

- It is the second phase of the compiler. It is also known as parser.
- It gets the token stream as input from the lexical analyser of the compiler and generates syntax tree as the output.
- Syntax tree:
  It is a tree in which interior nodes are operators and exterior nodes are operands.
- Example: For a=b+c*2, syntax tree is



SEMANTIC ANALYSIS:

- It is the third phase of the compiler.
- It gets input from the syntax analysis as parse tree and checks whether the given syntax is correct or not.
- It performs type conversion of all the data types into real data types.

INTERMEDIATE CODE GENERATION:

- It is the fourth phase of the compiler.
- It gets input from the semantic analysis and converts the input into output as intermediate code such as three-address code.
- The three -address code consists of a sequence of instructions, each of which has atmost three operands.
  Example: t1=t2+t3

CODE OPTIMIZATION:

- It is the fifth phase of the compiler.
- It gets the intermediate code as input and produces optimized intermediate code as output.
- This phase reduces the redundant code and attempts to improve the intermediate code so that faster-running machine code will result.
- During the code optimization, the result of the program is not affected.
- To improve the code generation, the optimization involves
  - deduction and removal of dead code (unreachable code).
  - calculation of constants in expressions and terms.
  - collapsing of repeated expression into temporary string.
  - loop unrolling.
  - moving code outside the loop.
  - removal of unwanted temporary variables.

CODE GENERATION:

- It is the final phase of the compiler.
- It gets input from code optimization phase and produces the target code or object code as result.
- Intermediate instructions are translated into a sequence of machine instructions that perform the same task.
- The code generation involves
  - allocation of register and memory
  - generation of correct references
  - generation of correct data types
  - generation of missing code

SYMBOL TABLE MANAGEMENT:

- Symbol table is used to store all the information about identifiers used in the program.
- It is a data structure containing a record for each identifier, with fields for the attributes of the identifier.
- It allows to find the record for each identifier quickly and to store or retrieve data from that record.
- Whenever an identifier is detected in any of the phases, it is stored in the symbol table.

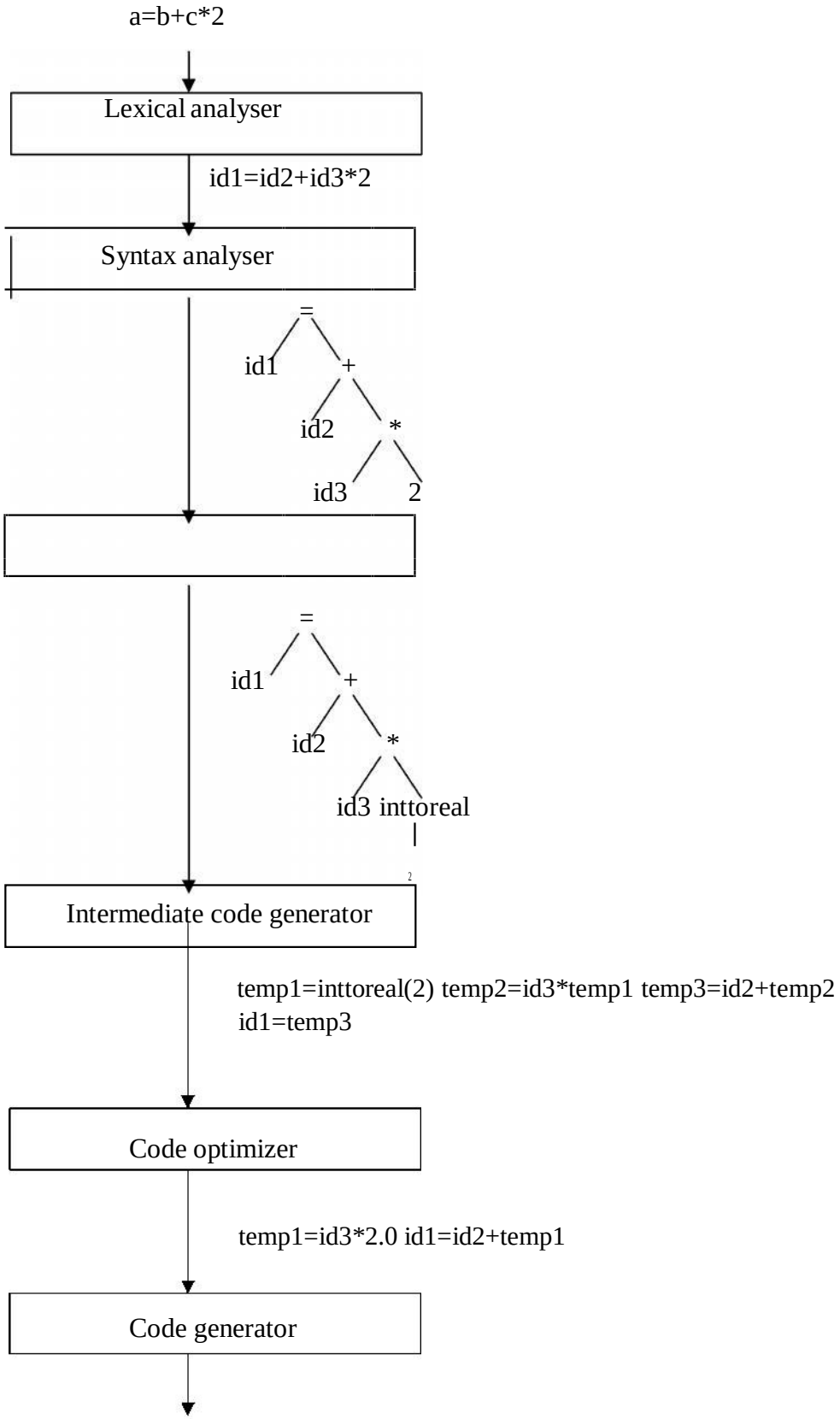
ERROR HANDLING:

- Each phase can encounter errors. After detecting an error, a phase must handle the error so that compilation can proceed.
- In lexical analysis, errors occur in separation of tokens.
- In syntax analysis, errors occur during construction of syntax tree.
- In semantic analysis, errors occur when the compiler detects constructs with right syntactic structure but no meaning and duringtype conversion.
- In code optimization, errors occur when the result is affected by the optimization.
- In code generation, it shows error when code is missing etc.

To illustrate the translation of source code through each phase, consider the statement a=b+c*2. The figure shows the representation of this statement after each phase:

Symbol Table

a	id1
b	id2
c	id3



```
MOVF id3,R2
MULF #2.0,R2
MOVF id2,R1
ADDF R2,R1
MOVF R1,id1
```

28. 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.(16)

Refer Q27.

29. What are the cousins of a Compiler? Explain them in detail.

COUSINS OF COMPILER

1. Preprocessor
2. Assembler
3. Loader and Link-editor

PREPROCESSOR

A preprocessor is a program that processes its input data to produce output that is used as input to another program. The output is said to be a preprocessed form of the input data, which is often used by some subsequent programs like compilers.

They may perform the following functions :

1. Macro processing
2. File Inclusion
3. Rational Preprocessors
4. Language extension

1. Macro processing:

A macro is a rule or pattern that specifies how a certain input sequence should be mapped to an output sequence according to a defined procedure. The mapping process that instantiates a macro into a specific output sequence is known as macro expansion.

2. File Inclusion:

Preprocessor includes header files into the program text. When the preprocessor finds an #include directive it replaces it by the entire content of the specified file.

3. Rational Preprocessors:

These processors change older languages with more modern flow-of-control and data-structuring facilities.

4. Language extension :

These processors attempt to add capabilities to the language by what amounts to built-in macros. For example, the language Equel is a database query language embedded in C.

ASSEMBLER

Assembler creates object code by translating assembly instruction mnemonics into machine code. There are two types of assemblers:

- One-pass assemblers go through the source code once and assume that all symbols will

be defined before any instruction that references them.

- Two-pass assemblers create a table with all symbols and their values in the first pass, and then use the table in a second pass to generate code.

LINKER AND LOADER

A **linker** or **link editor** is a program that takes one or more objects generated by a compiler and combines them into a single executable program.

Three tasks of the linker are :

1. Searches the program to find library routines used by program, e.g. printf(), math routines.
2. Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references
3. Resolves references among files.

A **loader** is the part of an operating system that is responsible for loading programs in memory, one of the essential stages in the process of starting a program.

30. Describe how various phases could be combined as a pass in a compiler? Also briefly explain Compiler construction tools. April/May 2008

GROUPING OF THE PHASES

Compiler can be grouped into front and back ends:

- **Front end**: analysis (machine independent)

These normally include lexical and syntactic analysis, the creation of the symbol table, semantic analysis and the generation of intermediate code. It also includes error handling that goes along with each of these phases.

- **Back end**: synthesis (machine dependent)

It includes code optimization phase and code generation along with the necessary error handling and symbol table operations.

Compiler passes

A collection of phases is done only once (single pass) or multiple times (multi pass)

- Single pass: usually requires everything to be defined before being used in source program.
- Multi pass: compiler may have to keep entire program representation in memory.

Several phases can be grouped into one single pass and the activities of these phases are interleaved during the pass. For example, lexical analysis, syntax analysis, semantic analysis and intermediate code generation might be grouped into one pass.

COMPILER CONSTRUCTION TOOLS

These are specialized tools that have been developed for helping implement various phases of a compiler. The following are the compiler construction tools:

1) Parser Generators:

-These produce syntax analyzers, normally from input that is based on a context-free grammar.

-It consumes a large fraction of the running time of a compiler. -

Example-YACC (Yet Another Compiler-Compiler).

2) Scanner Generator:

-These generate lexical analyzers, normally from a specification based on regular expressions.

-The basic organization of lexical analyzers is based on finite automation.

3) Syntax-Directed Translation:

-These produce routines that walk the parse tree and as a result generate intermediate code.

-Each translation is defined in terms of translations at its neighbor nodes in the tree.

4) Automatic Code Generators:

-It takes a collection of rules to translate intermediate language into machine language. The rules must include sufficient details to handle different possible access methods for data.

5) Data-Flow Engines:

-It does code optimization using data-flow analysis, that is, the gathering of information about how values are transmitted from one part of a program to each other part.

31. For the following expression Position:=initial+ rate*60. Write down the output after each phase.

Refer Q27

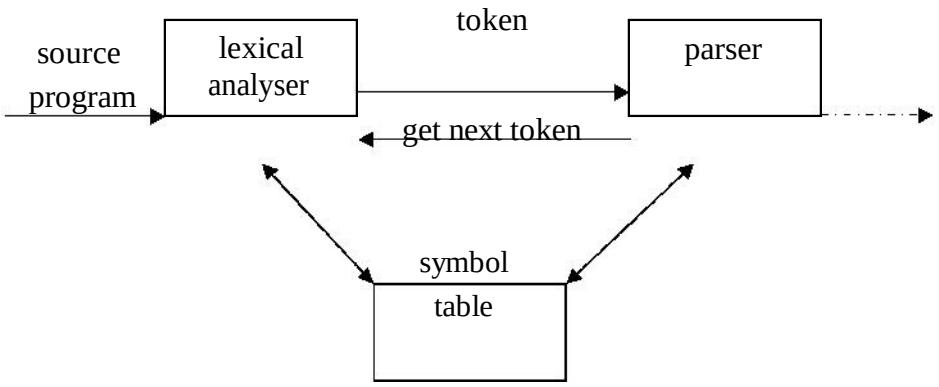
32. Explain the role Lexical Analyzer and issues of Lexial Analyzer.(8) Nov/Dec 2004

LEXICAL ANALYSIS

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a lexical analyzer or scanner. A lexer often exists as a single function which is called by a parser or another function.

THE ROLE OF THE LEXICAL ANALYZER

- The lexical analyzer is the first phase of a compiler.
- Its main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis.



- Upon receiving a "get next token" command from the parser, the lexical analyzer reads input characters until it can identify the next token.

ISSUES OF LEXICAL ANALYZER

There are three issues in lexical analysis:

- To make the design simpler.
- To improve the efficiency of the compiler.
- To enhance the computer portability.

33. Explain the specification of tokens. (8) April/May 2008

TOKENS

A token is a string of characters, categorized according to the rules as a symbol (e.g., IDENTIFIER, NUMBER, COMMA). The process of forming tokens from an input stream of characters is called **tokenization**.

A token can look like anything that is useful for processing an input text stream or text file. Consider this expression in the C programming language: sum=3+2;

Lexeme	Token type
sum	Identifier
=	Assignment operator
3	Number
+	Addition operator
2	Number
;	End of statement

LEXEME:

Collection or group of characters forming tokens is called Lexeme.

PATTERN:

A pattern is a description of the form that the lexemes of a token may take.

In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword. For identifiers and some other tokens, the pattern is a more complex structure that is matched by many strings.

Attributes for Tokens

Some tokens have attributes that can be passed back to the parser. The lexical analyzer collects information about tokens into their associated attributes. The attributes influence the

translation of tokens.

i) Constant : value of the constant
ii) Identifiers: pointer to the corresponding symbol table entry.

ERROR RECOVERY STRATEGIES IN LEXICAL ANALYSIS:

The following are the error-recovery actions in lexical analysis:

1)Deleting an extraneous character.

2) Inserting a missing character.

3)Replacing an incorrect character by a correct character.

4)Transforming two adjacent characters.

5) **Panic mode recovery**: Deletion of successive characters from the token until **error** is resolved.

UNIT II SYNTAX ANALYSIS

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 processing of correct programs.

4. **What is phrase level error recovery?**
   On discovering an error, a parser may perform local correction 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. Differentiate sentence and sentential form.

Sentence	Sentential form
• If $S \Rightarrow w$ then the string w is called Sentence of G. • Sentence is a string of terminals. Sentence is a sentential form with no nonterminals.	• If $S \Rightarrow \alpha$ then α is a sentential form of G. • Sentential form may contain non terminals.

8. 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.

9. 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 nonterminal 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.

10. 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:

$E \longrightarrow E+E / E*E /$ **id**

11. 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 front end of a compiler into two manageable-sized components.

12. 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.

13. 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.

14. Left factor the following grammar:

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b.$

Ans:

The left factored grammar is,

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \varepsilon$

$E \rightarrow b$

15. What is parsing?

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

16. 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.

17. 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.

18. What is meant by Predictive parsing? Nov/Dec 2007

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.

19. Define Bottom Up Parsing.

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

20. 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.

21. Define handle. What do you mean by handle pruning? Nov/Dec 2004, April/May 2005

- 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.

22. 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**.**

23. 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.

24. 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 nonterminals on the right side..

Ex:

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

25. 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.

26. 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.

27. 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.

28. 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.

29. 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 $[A \rightarrow \alpha X.\beta]$ such that $[A \rightarrow \alpha.X\beta]$ is in I

30. 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:

$(s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m, a_i a_{i+1} \ldots a_n \$)$

31. Define LR grammar.

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

32. What are kernel and non kernel items?Nov/Dec 2005

i. The set of items which include the initial item, $S' \rightarrow .S$, 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 left end, are known as non kernel items.

33. 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.

34. What is ambiguous grammer? Give an example. Nov/Dec 2005, Nov/Dec 2007

A grammer G is said to be ambiguous if it generates more than one parse trees for sentence of language L(G).

Example: E-> E+E|E*E|id

PART-B QUESTIONS (16 MARKS)

1. What are preliminary steps that are to be carried out during parsing? Explain with suitable examples.(6)*****

2. Explain the error recovery in predictive parsing.(8)*****

3. What are the necessary conditions to be carried out before the construction of predictive parser? (6)

4. i) Construct the predictive parser for the following grammar: (12)

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S.$

ii) Construct the behavior of the parser on the sentence (a, a) using the grammar specified above. (4)

5. i) Give an algorithm for finding the FIRST and FOLLOW positions for a given non-terminal.(4)

ii) Consider the grammar,

$E \rightarrow TE'$

$E \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id.$

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. (12)

6. Construct the predictive parser for the following grammar: (16)

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

$T \rightarrow T, S \mid S$

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. i) What is an operator grammar? Draw the precedence function graph for the following table.(8)

	a	(	)	,	$
a			>	>	>
(	<	<	=	<	
)			>	>	>
,	<	<	>	>	
$	<	<			

ii) Check whether the following grammar is SLR (1) or not. Explain your answer with reasons.(8)

$S \rightarrow L = R$
$S \rightarrow R$
$L \rightarrow * R$
$L \rightarrow id$
$R \rightarrow L.$

8. For the operators given below, calculate the operator-precedence relations and operator precedence function

id, +, *, $ (8)

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

$S \rightarrow$ iEtS | iEtSeS | a

$E \rightarrow$ b

Also define the FIRST and FOLLOW procedures. (16)*****

10. Consider the grammar

$E \rightarrow$ E + E | E * E | (E) | **id**

Show the sequence of moves made by the shift-reduce parser on the input $id_1 + id_2 * id_3$ and determine whether the given string is accepted by the parser or not. (8)

11. i) What is a shift-reduce parser? Explain in detail the conflicts that may occur during shift-reduce parsing. (6)

ii) For the grammar given below, calculate the operator precedence relation and the precedence functions (10)

$E \rightarrow$ E + E | E - E | E * E | E / E | E $\uparrow$E | (E) | -E | **id**

12. i) Consider the grammar given below. (12)

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

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. (4)

13. Explain the LR parsing algorithm with an example. (6)*****

14. Construct a canonical parsing table for the grammar given below (16)

$S \rightarrow CC$

$C \rightarrow cC \mid d$

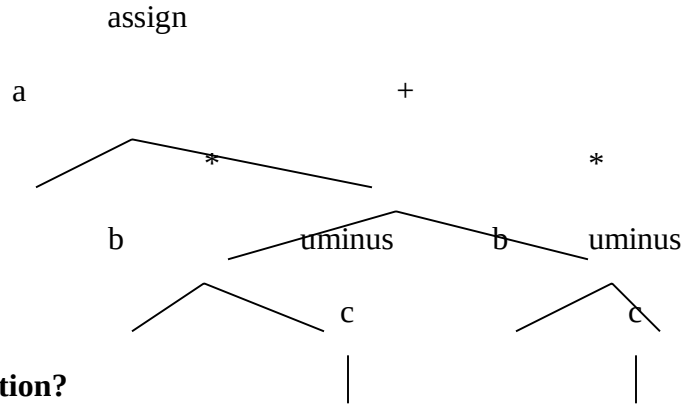
UNIT III INTERMEDIATE CODE GENERATION

1. **What are the benefits of using machine-independent intermediate form?**
   - Retargeting is facilitated; a compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.
   - A machine-independent code optimizer can be applied to the intermediate representation.

2. **List the three kinds of intermediate representation.**
   The three kinds of intermediate representations are
   i. Syntax trees
   ii. Postfix notation
   iii. Three address code

3. **How can you generate three-address code?**
   The three-address code is generated using semantic rules that are similar to those for constructing syntax trees for generating postfix notation.

4. **What is a syntax tree? Draw the syntax tree for the assignment statement**
   a **:=** b * -c + b * -c.
   - A syntax tree depicts the natural hierarchical structure of a source program.
   - Syntax tree:



5. **What is postfix notation?**
   A Postfix notation is a linearized representation of a syntax tree. It is a list of nodes of the tree in which a node appears immediately after its children.

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

7. **Why "Three address code" is named so?**
   The reason for the term "Three address code" is that each usually contains three addresses, two for operands and one for the result.

8. Define three-address code.

- Three-address code is a sequence of statements of the general form

  x **:=** y *op* z

  where x, y and z are names, constants, or compiler-generated temporaries; op stands for any operator, such as fixed or floating-point arithmetic operator, or a logical operator on boolean-valued data.
- Three-address code is a linearized representation of a syntax tree or a dag in which explicit names correspond to the interior nodes of the graph.

9. State quadruple

A quadruple is a record structure with four fields, which we call op, arg1, arg2 and result.

10. What is called an abstract or syntax tree?

A tree in which each leaf represents an operand and each interior node an operator is called as abstract or syntax tree.

11. Construct Three address code for the following

position := initial + rate * 60

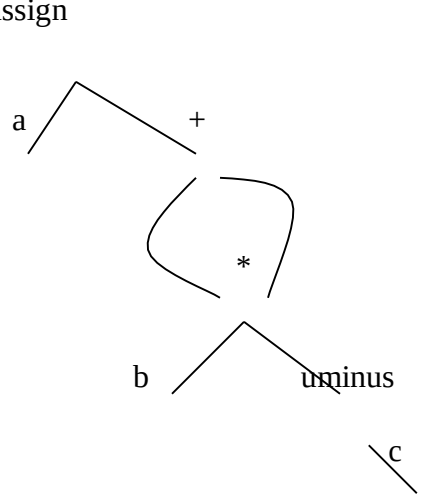
Ans:

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1  := temp3
```

12. What are triples?

- The fields arg1,and arg2 for the arguments of op, are either pointers to the symbol table or pointers into the triple structure then the three fields used in the intermediate code format are called triples.
- In other words the intermediate code format is known as triples.

13. Draw the DAG for a := b * -c + b * -c



14. List the types of three address statements.

The types of three address statements are

a. Assignment statements
b. Assignment Instructions
c. Copy statements
d. Unconditional Jumps
e. Conditional jumps
f. Indexed assignments
g. Address and pointer assignments
h. Procedure calls and return

15. What are the various methods of implementing three-address statements?

i. Quadruples
ii. Triples
iii. Indirect triples

16. What is meant by declaration?

The process of declaring keywords, procedures, functions, variables, and statements with proper syntax is called declaration.

17. How semantic rules are defined?

The semantic rules are defined by the following ways

a. mktable(previous)
b. enter(table,name,type,offset)
c. addwith(table,width)
d. enterproc(table,name,newtable)

18. What are the two primary purposes of Boolean Expressions?

- They are used to compute logical values
- They are used as conditional expressions in statements that alter the flow of control, such as if-then, if-then-else, or while-do statements.

19. Define Boolean Expression.

Expressions which are composed of the Boolean operators (and, or, and not) applied to elements that are Boolean variables or relational expressions are known as Boolean expressions

20. What are the two methods to represent the value of a Boolean expression?

i. The first method is to encode true and false numerically and to evaluate a Boolean expression analogously to an arithmetic expression.
ii. The second principal method of implementing Boolean expression is by flow of control that is representing the value of a Boolean expression by a position reached in a program.

21. What do you mean by viable prefixes. Nov/Dec 2004

Viable prefixes are the set of prefixes of right sentinels forms that can appear on the stack of shift/reduce parser are called viable prefixes. It is always possible to add terminal symbols to the end of the viable prefix to obtain a right sentential form.

22. What is meant by Shot-Circuit or jumping code?

We can also translate a Boolean expression into three-address code without generating code for any of the Boolean operators and without having the code necessarily evaluate the entire expression. This style of evaluation is sometimes called "short-circuit" or "jumping" code.

23. What is known as calling sequence?

A sequence of actions taken on entry to and exit from each procedure is known as calling sequence.

24. What is the intermediate code representation for the expression a or b and not c? (Or) Translate a or b and not c into three address code.

Three-address sequence is

t_1 := not c
t_2 := b and t_1
t_3 := a or t_2

25. Translate the conditional statement if *a<b then 1 else 0* into three address code.

Three-address sequence is

```
100:  if a < b goto 103
101:  t := 0
102:  goto 104
103:  t := 1
104:
```

26. Explain the following functions:

i) makelist(*i*) ii) merge(p_1,p_2) iii) backpatch(*p*,*i*)

i. makelist(*i*) creates a new list containing only I, an index into the array of quadruples; makelist returns a pointer to the list it has made.
ii. merge(p_1,p_2) concatenates the lists pointed to by p_1 *and* p_2, and returns a pointer to the concatenated list.
iii. backpatch(*p*,*i*) inserts *i* as the target label for each of the statements on the list pointed to by *p*.

27. Define back patching. May/June 2007 & Nov/Dec 2007

Back patching is the activity of filling up unspecified information of labels using appropriate semantic actions in during the code generation process.

28. What are the methods of representing a syntax tree?

i. Each node is represented as a record with a field for its operator and additional fields for pointers to its children.
ii. Nodes are allocated from an array of records and the index or position of the node serves as the pointer to the node.

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

Ans:

Production	Semantic rule
$S \rightarrow$ if E then S_1 else S_2	E.true := newlabel; E.false := newlabel; S_1.next := S.next S_2.next := S.next S.code := E.code || gen(E.true ':') || S_1.code || gen('goto' S.next) || gen(E.false ':') || S_2.code

PART-B QUESTIONS (16 MARKS)

1. How would you generate the intermediate code for the flow of control statements? Explain with examples. (16) *
2. What are the various ways of calling the procedures? Explain in detail. (8)
3. What is the 3 address code? Mention its types. How would you implement the three address statements? Explain with suitable examples. (8) **
4. Explain how declaration is done in a procedure using syntax directed translation. (6)
5. Explain procedure calls with suitable example. (6)*Nov/Dec 2007
6. Describe the method of generating syntax directed definition for control statements.(8)
7. Brief Intermediate code generation for Basic block, Control Flow and Boolean Expressions.(16)
8. Write about Quadruple and Triple with its structure.(6)
9. Explain the data structure used for implementing Symbol Table.(6)** **Nov/Dec 2007**
10. Compare the various data structures used for Symbol Table construction.(4) **Nov/Dec 2007**

UNIT –IV CODE GENERATION

1. What are basic blocks?

A sequence of consecutive statements which may be entered only at the beginning and when entered are executed in sequence without halt or possibility of branch , are called basic blocks.

2. What is a flow graph?

- The basic block and their successor relationships shown by a directed graph is called a flow graph.
- The nodes of a flow graph are the basic blocks.

3. Mention the applications of DAGs.

- We can automatically detect common sub expressions.
- We can determine the statements that compute the values, which could be used outside the block.
- We can determine which identifiers have their values used in the block.

4. What are the advantages and disadvantages of register allocation and assignments?

- Advantages:
  i. It simplifies the design of a compiler
- Disadvantages:
  i. It is applied too strictly.
  ii. It uses registers in efficiently. Certain registers may go unused over substantial portions of the code, while unnecessary load and stores are generated.

5. What is meant by virtual machine?

An intermediate language as a model assembly language, optimized for a non-existent but ideal computer called a virtual machine.

6. Discuss back-end and front end?

- Back-end
  i. Intermediate to binary translation is usually done by a separate compilation pass called back end.
- Front end
  i. There are several back ends for different target machines, all of which use the same parser and code generator called front end.

7. Define relocatable object module.

The unpatched binary image is usually called a relocatable object module.

8. What is meant by multiregister operations?

We can modify our labeling algorithm to handle operations like multiplication, division, or function calls which normally requires more than one register to perform. Hence this operation is called multiregister operations.

9. What is meant by peephole optimization?

Peephole optimization is a technique used in many compliers, in connection with the optimization of either intermediate or object code. It is really an attempt to overcome the difficulties encountered in syntax directed generation of code.

10. List the types of addressing modes:-

i) Intermediate mode
ii) Direct mode
iii) Indirect mode
iv) Effective address mode

11. What is input to code generator?

The input to code generator consists of the intermediate representation of the source program produced by the front end together with information in the symbol table that is used to determine the run time addresses of the data objects denoted by the names in the intermediate representation.

12. How the use of registers is subdivided into 2 sub-problems?

- During register allocation we select the set of variables that will reside in registers at a point in the program.
- During a subsequent register assignment phase, we pick the specific register that a variable will reside in.

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

- 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 of the instruction.
- Address modes involving registers have cost zero, while those with a memory location or literal in them have cost one.

14. What are the primary structure preserving transformations on basic blocks?

- Common sub-expression elimination
- Dead-code elimination
- Renaming of temporary variable
- Interchange of 2 independent adjacent statements.

15. Give some examples for 3 address statements.

- Call
- Return
- Halt
- Action

16. What are the characteristics of peephole optimization?

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

17. What is a recursive procedure?

A procedure is recursive a new activation can begin before an earlier activation of the same procedure has ended.

18. What are the common methods for associating actual and formal parameters?

- Call-by-value
- Call-by-reference
- Copy-restore
- Call-by-name
- Macro-expansion

19. Define DAG. Nov/Dec 2007

A DAG for a basic block is a directed acyclic graph with the following labels on nodes:

i) Leaves are labeled by unique identifiers, either variable names or constants.
ii) Interior nodes are labeled by an operator symbol.
iii) Nodes are also optionally given a sequence of identifiers for labels.

20. What are the issues in the design of code generators? Nov/Dec 2007

i) Input to the code generator
ii) Target programs
iii) Memory management
iv) Instruction selection
v) Register allocation
vi) Choice of evaluation order
vii) Approaches to code generation

21. What are the various forms of target programs?

i) Absolute machine language
ii) Relocatable machine language
iii) Assembly language

22. What is memory management?

Mapping names in the source program to addresses of data object in run time memory done comparatively by the front end and the code generator is called memory management

23. What is backpatching? April/May 2008

Process of leaving a blank slot for missing information and fill in the slot when the information becomes available is known as backpatching.

24. What are the rules to determine the leaders of basic blocks?

i) The first statement is a leader
ii) Any statement that is the target of a conditional or unconditional goto is a leader
iii) Any statement that immediately follows a goto or conditional goto statement is a leader.

25. What is the use of algebraic transformation?

Algebraic transformation can be used to change the set of expressions computed by a basic blocks into an algebraically equivalent set.

26. What is meant by loop?

A loop is a collection of nodes in a flow graph such that

i) All nodes in the collection are strongly connected i.e., from any node in the loop to any other, there is a path of length one or more, wholly within the loop
ii) The collection of nodes has a unique entry, i.e. a node in the loop such that the only way to reach a node of the loop from a node outside the loop is to first go through the entry.

27. What is register descriptor and address descriptor?

- A register descriptor keeps track of what is currently in each register.
- An address descriptor keeps track of the location where the current value of the name can be found at run time.

28. What are the characteristics of peephole optimization?

i) Redundant – instruction elimination
ii) Flow – of – control optimizations
iii) Algebraic simplifications
iv) Use of machine idioms

PART-B QUESTIONS (16 MARKS)

1. What are the issues in design of a code generator? Explain in detail(8)* **Nov/Dec 2007**

2. Discuss about the run time storage management of a code generator. (8)***Nov/Dec 2007**

3. Explain the simple code generator with a suitable example. (8) **Nov/Dec 2007**

4. Explain in the DAG representation of the basic block with example(8)*

5. Explain peephole optimization. (8)

UNIT –V CODE OPTIMIZATION

1. **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.

2. **What is the more accurate term for code optimization?**

   The more accurate term for code optimization would be "code improvement"

3. **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.

4. **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

5. **What are the 3 areas of code optimization?**
   - Local optimization
   - Loop optimization
   - Data flow analysis

6. **Define local optimization.**

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

7. **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.

8. **What do you mean by inner loops?**

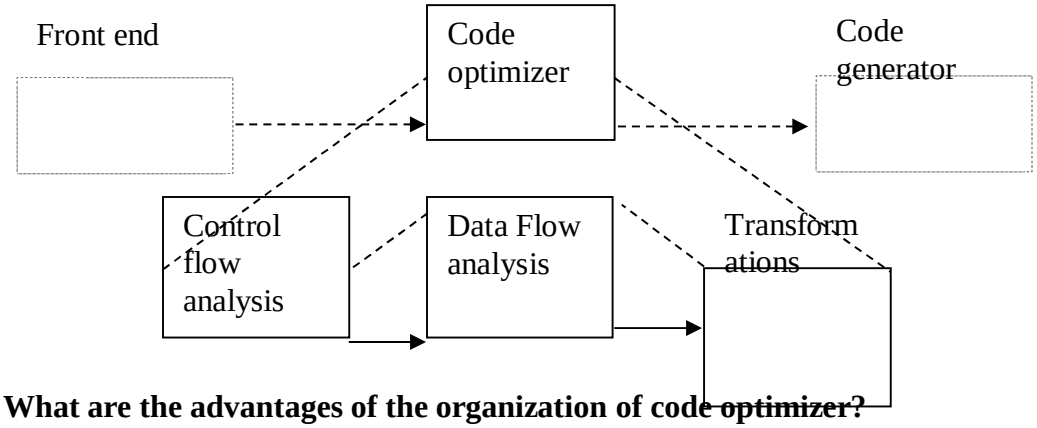
   The most heavily traveled parts of a program, the inner loops, are an obvious target for optimization. Typical loop optimizations are the removal of loop invariant computations and the elimination of induction variables.

9. **What is code motion? April/May 2004, May/June 2007, April/May-2008**

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

10. **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.

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



12. What are the advantages of the organization of code optimizer?

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

13. 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.

14. Give examples for function preserving transformations.

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

15. 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.

16. 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.

17. What are the techniques used for loop optimization?

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

18. 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.

19. 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.

20. 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 of a 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.

21. What are the two standard storage allocation strategies?

The two standard allocation strategies are

1. Static allocation.
2. Stack allocation

22. Discuss about static allocation.

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

23. Write short notes on activation tree. Nov/Dec 2007

- 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.

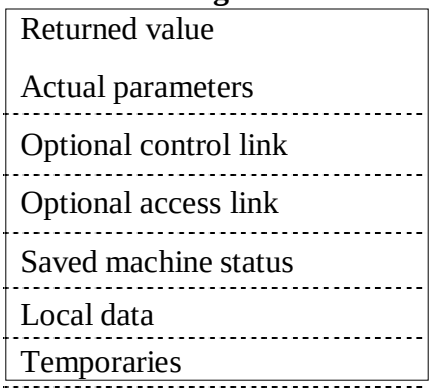
24. Define control stack.

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

25. Define heap.

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

26. Give the structure of general activation record



27. 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.

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

- Deep access
- Shallow access

29. What is padding?

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

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

- Static allocation
- Stack allocation
- Heap allocation

31. 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

32. 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.

33. 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.

34. 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.

35. What is block? Give its syntax.

A block is a statement containing its own data declaration.

Syntax:
{
Declaration statements
}

36. 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.

37. 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.

38. 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.

PART-B QUESTIONS (16 MARKS)

1. Explain the principle sources of code optimization in detail (16)***May/June 2007**
2. What is bootstrapping? Explain the approach to compiler development (10)
3. Discuss about the following : (6)

   i) Copy propagation **May/June 2006**

   ii) Dead code elimination

   iii) Code motion
4. Explain about parameter passing.(6) **April/May 2008**
5. What are the different storage allocation strategies? Explain.(10)
6. Explain the various source language issues.

7. Write about Data Flow Analysis of structural programs.(8)**May/June 2007**

8. Explain various code optimization techniques in detail.(10) **Nov/Dec 2007**

9. Generate target code for the given program segments:(6)

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