

Noorul Islam College of Engineering
Department of Software Engineering
Part A Questions and Answers
XCS 362 Compiler Design

M. MERLIN JOHNSY
Unit I

1. Define compilers and translators?

A translator is a program that takes as input a program written in one programming language and produces as output a program in another language. If the source language is a high level language and the object language is a low-level language then such a translator is called a compiler.

2. What are the phases of a compiler?

- i) Lexical analysis.
- ii) Syntax analysis.
- iii) Intermediate code generation.
- iv) Code optimization.
- v) Code generation.

3. 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 transformations specified by its phases and writes output into an intermediate file, which is read by subsequent pass.

4. Define Lexical Analysis?

The lexical analyzer reads the source program one character at a time, carving the source program into a sequence of atomic units called tokens. Identifiers, keywords, constants, operators and punctuation symbols are typical tokens.

5. Write notes on syntax analysis?

Syntax analysis is also called parsing. It involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output.

6. What is meant by semantic analysis?

The semantic analysis phase checks the source program for semantic errors and gathers type information for the subsequent code generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operand of expressions and statements.

7. Define optimization?

Certain compilers apply transformations to the output of the intermediate code generator. It is used to produce an intermediate-language from which a faster or smaller object program can be produced. This phase is called optimization phase. Types of optimization are local optimization and loop optimization.

8. What is cross compiler?

A compiler may run on one machine and produce object code for another machine is called cross compiler.

9. Define semantics of a programming language?

The rules that tell whether a string is a valid program or not are called syntax of the language. The rules that give meaning to programs are called the semantics of a programming language.

10. What are the data elements of a programming language?

- a) Numerical data.
- b) Logical data.
- c) Character data.
- d) Pointers.
- e) Labels.

11. Define binding?

The act of associating attributes to a name is referred to as binding the attributes to the name. Most binding done at compile time called static binding. Some languages, such as SNOBOL allow dynamic binding, binding done at run time.

12. What is coercion of types?

The translation of the operator, which the compiler must provide, includes any necessary conversion from one type to another, and this implied change in type is called coercion.

13. What is meant by loaders and link-editors?

A program called a loader performs the two function of loading and link-editing. The process of loading consists of taking relocatable machine code, altering the relocatable addresses and placing the altered instruction and data in memory at the proper locations.

14. Write down the various compiler construction tools?

Some of the useful compiler construction tools are

- a) Parser generator
- b) Scanner generators
- c) Syntax-directed translation engines
- d) Automatic code generators
- e) Data-flow engines

15. What are the possible error recovery actions in lexical analysis:

- a) Deleting an extraneous character
- b) Inserting a missing character
- c) Replacing an incorrect character by a correct character
- d) Transposing two adjacent characters

16. Define regular expressions?

Regular expressions are the notation we shall use to define the class of languages known as regular sets. It is used to describe tokens. In regular expression notation we could write

identifier = letter (letter | digit)*

17. Write the regular expression for denoting the set containing the string a and all strings consisting of zero or more a's followed by a b.

a | a * b

18. Describe the language generated by the regular expressions?a) $0(011)^*0$

The set of zero or more number of zeroes and ones prefixed by zero and suffixed by 0.

19. What is a regular definition?

If Σ is an alphabet of basic symbols, then a regular definition is a sequence of definition of the form

 $d_1 \rightarrow r_1$ $d_2 \rightarrow r_2$

....

 $d_n \rightarrow r_n$

Where each d_i is a distinct name, and each r_i is a regular expression over the symbol in $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$

20. Define finite automata?

A better way to convert a regular expression to a recognizer is to construct a generalized transition diagram from the expression. This diagram is called a finite automaton.

21. What is Deterministic Automata?

A finite automaton is deterministic if

- a. It has no transition of input ϵ .
- b. For each state s and input symbol a , there is at most one edge labeled a leaving s .

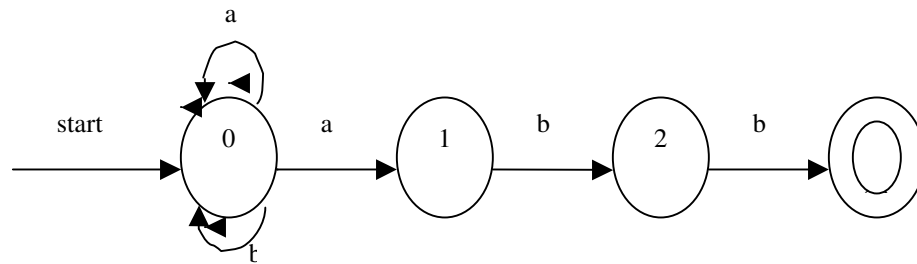
22. Write the algorithm for simulating a DFA?

```

s := s0;
c := nextchar
while c ≠ eof do
    s := move(s,c)
    c := nextchar
end
if s is in F then
    return "yes"
else return "no";

```

23. Write the transition graph for an NFA that recognizes the language $(ab)^*abb$?



24. Define LEX?

LEX is a tool for automatically generating lexical analyzers. A LEX source program is a specification of a lexical analyzer, consisting of a set of regular expressions together with an action for each regular expression. The output of LEX is a lexical analyzer program.

25. Write notes on auxiliary definitions?

The auxiliary definitions are statements of the form

- a. $D1 = R1$
- b. $D2 = R2$
- c.
- d.
- e. $Dn = Rn$

Where each D_i is a distinct name, and each R_i is a regular expression.

Unit II

26. Define context-free grammar?

The syntactic specification of a programming language can be formed by a notation called a context-free grammar, which is also called a BNF (Backus-Naur form) description. Context-free grammars are capable of describing most, but not all, of the syntax of programming languages.

27. Define parse trees?

The graphical representation for derivations that filters out the choice regarding replacement order. This representation is called the parse trees. It represents the hierarchical syntactic structure of sentences that is implied by the grammar.

28. What are the various types of errors in program?

- a) Lexical, such as misspelling an identifier, keyword, or operator.
- b) Syntactic, such as an arithmetic expression with unbalanced parenthesis.
- c) Semantic, such as an operator applied to an incompatible operand.
- d) Logical, such as an infinitely recursive call.

29. What are the various error-recovery strategies?

- a) Panic mode - On discovering this error, the parser discards the input symbols one at a time until one of a designated set of synchronized tokens is found.
- b) Phrase level - On discovering an error, a parser performs local correction on the remaining input; that is, it may replace a prefix or the remaining input by some string that allows the parser to continue.
- c) Error production and - If we have a good idea of error we recover it.
- d) Global correction - Use the compiler to make as few changes as possible in processing an input string.

30. Write a grammar to define simple arithmetic expression?

$$\begin{aligned} \text{expr} &\rightarrow \text{expr op expr} \\ \text{expr} &\rightarrow (\text{expr}) \\ \text{expr} &\rightarrow - \text{expr} \\ \text{expr} &\rightarrow \text{id} \\ \text{op} &\rightarrow + | - | * | / | ^ \end{aligned}$$

31. Define context-free language?

Given a grammar G with start symbol S , we can use the \Rightarrow relation to define $L(G)$, the language generated by G . We say a string of terminals w is in $L(G)$ if and only if $S \Rightarrow w$. The string w is called a sentence of G . The language that can only be generated by a grammar is said to be a context-free language.

$L(G)$ if and only if $S \Rightarrow w$. The string w is called a sentence of G . The language that can only be generated by a grammar is said to be a context-free language.

32. Define ambiguity?

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 more than one rightmost derivation for some sentence.

33. What is meant by left recursion?

A grammar is left recursive if it has a nonterminal A such that there is a derivation $A \Rightarrow A \alpha$ for some string α . Top down parsing methods cannot

handle left-recursion grammars, so a transformation that eliminates left recursion in needed.

Ex:-

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid id \end{aligned}$$

34. Write the algorithm to eliminate left recursion from a grammar?

1. Arrange the non terminals in some order A_1, A_2, \dots, A_n
- 2 for $i := 1$ to n do begin
 - for $j := 1$ to $i-1$ do begin
 - replace each production of the form $A_i \rightarrow A_j \gamma$ by the productions $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$
 - end
 - eliminate the immediate left recursion among the A_i productions
- end.

35. Write is meant by 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 production to defer the decision until we have seen enough of the input to make the right choice.

35. Define parser?

A parser for grammar G is a program that takes as input a string w and produces as output either a parse tree for w , if w is a sentence of G , or an error message indicating that w is not a sentence of G .

36. What is shift_reduce parsing?

The bottom_up style of parsing is called shift_reduce parsing. This parsing method is bottom_up because it attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root.

37. Define Handles?

A handle of a right-sentential form γ is a production $A \rightarrow \beta$ and a position of γ where the string β may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of γ .

38. What are the four possible action of a shift_reduce parser?

- a) Shift action – the next input symbol is shifted to the top of the stack.
- b) Reduce action – replace handle.
- c) Accept action – successful completion of parsing.
- d) Error action- find syntax error.

39. What is an operator grammar?

The grammars have the property that no production right side is ϵ or has two adjacent nonterminals is called operator grammar.

40. What are the problems in top down parsing?

- a) Left recursion.
- b) Backtracking.
- c) The order in which alternates are tried can affect the language accepted.

41. Define recursive-descent parser?

A parser that uses a set of recursive procedures to recognize its input with no backtracking is called a recursive-descent parser. The recursive procedures can be quite easy to write.

42. Define predictive parsers?

A predictive parser is an efficient way of implementing recursive_descent parsing by handling the stack of activation records explicitly. The predictive parser has an input, a stack, a parsing table and an output.

43. Define FIRST in predictive parsing?

- a) If X is terminal, then $FIRST(X)$ is $\{X\}$.
- b) If $X \rightarrow \epsilon$ is a production, then add ϵ to $FIRST(X)$.
- c) If X is non terminal and $X \rightarrow Y_1Y_2\dots Y_k$ is a production, then place a in $FIRST(X)$ if for some i , a is in $FIRST(Y_i)$, and ϵ is in all of $FIRST(Y_1), \dots, FIRST(Y_{i-1})$;

*

that is, $Y_1\dots Y_{i-1} \Rightarrow \epsilon$. If ϵ is in $FIRST(Y_j)$ for all $j = 1, 2, \dots, k$, then add ϵ to $FIRST(X)$.

44. Define FOLLOW in predictive parsing?

The $FOLLOW(A)$, for some non terminal A , to be the set of terminals a that can appear immediately to the right of A in some sentential form, that is, $S \xrightarrow{*} \alpha A a \beta$ for some α and β . If A can be the rightmost symbol in some sentential form, then we add $\$$ to $FOLLOW(A)$.

45. Write the algorithm for the construction of a predictive parsing table?

Input : Grammar G

Output : Parsing table M

Method :

- a) For each production $A \rightarrow \alpha$ of the grammar, do step b and c.
- b) For each terminal a in $FIRST(\alpha)$ and $A \rightarrow \alpha$ to $M[A, a]$
- c) If ϵ is in $FIRST(\alpha)$ add $A \rightarrow \alpha$ to $M[A, b]$ for each terminal b is $FOLLOW(A)$. If ϵ is in $FIRST(\alpha)$ and $\$$ is in $FOLLOW(A)$, and $A \rightarrow \alpha$ to $M[A, \$]$
- d) Make each undefined entry of M be error.

46. What is LL(1) grammar?

A grammar whose parsing table has no multiply-defined entries is said to be LL(1).

47. What are LR parsers?

LR(k) parsers scan the input from (L) left to right and construct a (R) rightmost derivation in reverse. LR parsers consist of a driver routine and a parsing table. The k is for the number of input symbols of lookahead that are used in making parsing decisions. When k is omitted, k is assumed to be 1. It is attractive because

- a) It is constructed to recognize virtually all programming language constructs for which context-free grammars can be written.
- b) It is the most general nonbacktracking shift-reduce method.
- c) 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.

d) It can detect syntactic error as soon as possible.

48. Define LR grammar?

A grammar for which we can construct a parsing table in which every entry is uniquely defined is said to be an LR grammar.

49. What is augmented grammar?

If G is a grammar with start symbol S , then G' , the augmented grammar for G , is G with a new start symbol S' and production $S' \rightarrow S$. It is to indicate the parser when it should stop and announce acceptance of the input.

Unit III

1. Define procedure definition?

A procedure definition is a declaration that, in its simplest form, associates an identifier with a statement. The identifier is the procedure name, and the statement body. Some of the identifiers appearing in a procedure definition are special and are called formal parameters of the procedure. Arguments, known as actual parameters may be passed to a called procedure; they are substituted for the formal in the body.

2. Define activation trees?

A recursive procedure p need not call itself directly; p may call another procedure q , which may then call p through some sequence of procedure calls. We can use a tree called an activation tree, to depict the way control enters and leaves activation. In an activation tree

- a) Each node represents an activation of a procedure,
- b) The root represents the activation of the main program
- c) The node for a is the parent of the node for b if and only if control flows from activation a to b , and
- d) The 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 .

3. Write notes on control stack?

A control stack is to keep track of live procedure activations. The idea is to push the node for activation onto the control stack as the activation begins and to pop the node when the activation ends.

4. Write the scope of a declaration?

A portion of the program to which a declaration applies is called the scope of that declaration. An occurrence of a name in a procedure is said to be local to the procedure if it is in the scope of a declaration within the procedure; otherwise, the occurrence is said to be nonlocal.

5. Define binding of names?

When an environment associates storage location s with a name x , we say that x is bound to s ; the association itself is referred to as a binding of x . A binding is the dynamic counterpart of a declaring.

6. What is the use of run time storage?

The run time storage might be subdivided to hold

- a) The generated target code
- b) Data objects, and
- c) A counterpart of the control stack to keep track of procedure activation.

7. What is an activation record?

Information needed by a single execution of a procedure is managed using a contiguous block of storage called an activation record or frame, consisting of the collection of fields such as

- a) Return value
- b) Actual parameters

- c) Optional control link
- d) Optional access link
- e) Saved machine status
- f) Local data
- g) Temporaries

8. What are the storage allocation strategies?

- a) Static allocation lays out storage for all data objects at compile time.
- b) Stack allocation manages the run-storage as a stack.
- c) Heap allocation allocates and deallocates storage as needed at run time from a data area known as heap.

9. What is static allocation?

In static allocation, names are bound to storage as the program is compiled, so there is no need for a run-time support package. Since the bindings do not change at run time, every time a procedure is activated, its names are bound to the same storage location.

10. What is stack allocation?

Stack allocation is based on the idea of a control stack; storage is organized as a stack, and activation records are pushed and popped as activations begin and end respectively.

11. What are the limitations of static allocation?

- a) The size of a data object and constraints on its position in memory must be known at compile time.
- b) Recursive procedure is restricted.
- c) Data structures cannot be created dynamically.

12. What is dangling references?

Whenever storage can be deallocated, the problem of dangling references arises. A dangling reference occurs when there is a reference to storage that has been deallocated.

13. What is heap allocation?

Heap allocation parcels out pieces of contiguous storage, as needed for activation records or other objects. Pieces may be deallocated in any order, so over time the heap will consist of alternate areas that are free and in use.

14. Define displays?

Faster access to nonlocals than with access links can be obtained using an array of pointers to activation records, called display. The display changes when a new activation occurs and it must be reset when control returns from the new activation.

15. Write notes on call-by-value?

This is the simplest method for passing parameters. The actual parameters are evaluated and their r-values are passed to the called procedure. Call-by-value can be implemented as follows

- a) A formal parameter is treated just like a local name, so the storage for the formals is in the activation record of the called procedure.
- b) The caller evaluates the actual parameters and places their r-values in the storage for the formals.

16. What is meant by call-by-reference?

When parameters are passed by reference, the caller passes to the called procedure a pointer to the storage address of each actual parameter.

a) If an actual parameter is a name or an expression having l-value, then that l-value itself is passed.

b) However, if the actual parameter is an expression, then the expression is evaluated in a new location, and address of that location is passed.

17. What is meant by copy-restore?

A hybrid between call-by-value and call by reference is copy-restore linkage.

1. Before control flows to the called procedure, the actual parameters are evaluated.

2. When control returns, the current r-values of the formal parameters are copied back into the l-values of the actual, using the l-values computed before the call.

18. Write notes on call-by-name?

Call-by-name is traditionally defined by the copy-rule of Algol, which is

a) The procedure is treated as if it were a macro; that is, its body is substituted for the call in the caller, with the actual parameters literally substituted for the formals. Such a literal substitution is called macro-expansion or in-line expansion.

b) The local named of the called procedure are kept distinct from the names of the calling procedure.

c) The actual parameters are surrounded by parenthesis if necessary to preserve their integrity.

19. Define symbol tables?

A compiler uses a symbol table to keep track of scope and binding information about names. The symbol table is searched every time a name is encountered in the source text. Two symbol table mechanisms are linear list and hash tables.

20. Define Garbage?

Dynamically allocated storage can become unreachable. Storage that a program allocates but cannot refer to is called garbage. Lisp performs garbage collection that reclaims inaccessible storage.

21. What are the dynamic storage allocation techniques?

a) Explicit allocation of Fixed-sized blocks.

b) Explicit allocation of Variable-sized blocks – one method is first-fit method, in this when a block of size s is allocated; we search for the first free block that is of size $f \geq s$.

c) Implicit Deallocation – it requires cooperation between the user program and the run-time package.

22. What is memory map?

For each data area the compiler creates a memory map, which is a description of the contents of the area. This memory map might simply consist of

an indication, in the symbol-table entry for each name in the area, of its offset in the area.

23. Write notes on COMMON statement?

When processing a declaration like

COMMON / BLOCK1 / NAME 1,NAME 2

a) In the table for COMMON block names, create a record for BLOCK 1, if one does not already exist.

b) In the symbol table entries for NAME1 and NAME2, set a pointer to the symbol table entry for BLOCK1, indicating that these are in COMMON and members of BLOCK1.

24. Write notes on equivalence statement?

A sequence of EQUIVALENCE statements groups names into equivalence sets whose positions relative to one another are all defined by the EQUIVALENCE statements. For example

EQUIVALENCE A,B+100

EQUIVALENCE C,D-100

EQUIVALENCE A,C+30

EQUIVALENCE E,F

Groups names into the sets {A,B,C,D} and {E,F} where E and F denote the same location.

25. Write the algorithm for adjustment of offsets?

Begin

 h := offset(nk-1);

 for i := k-2 downto 1 do begin

 parent(ni) := nk;

 h := h + offset(ni);

 offset(ni) := h

 End

End

Unit IV

50. Define intermediate code?

In many compilers the source code is translated into a language which is intermediate in complexity between a high-level programming language and machine code. Such a language is therefore called intermediate code or intermediate text.

51. What are the benefits of using a machine-independent intermediate form?

- a) 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.
- b) A machine-independent code optimizer can be applied to the intermediate representation.

52. What are the various kinds of intermediate representations for intermediate code generation?

- a) Syntax trees
- b) Postfix notation
- c) Three address code

53. What is syntax directed translation scheme?

A syntax directed translation scheme is merely a context-free grammar in which a program fragment called an output action (or sometimes a semantic action or semantic rule) is associated with each production.

54. Write the syntax-directed translation scheme for infix-postfix translation?

PRODUCTION	SEMANTIC ACTION
$E \rightarrow E^{(1)} \text{ op } E^{(2)}$	$E.CODE := E^{(1)}.CODE \parallel E^{(2)}.CODE \parallel \text{op}$
$E \rightarrow (E^{(1)})$	$E.CODE := E^{(1)}.CODE$
$E \rightarrow \text{id}$	$E.CODE := \text{id}$

55. Define parse trees and syntax trees.

The parse tree itself is a useful intermediate language representation for a source program. A parse tree, however often contains redundant information which can be eliminated. A variant of a parse tree is what is called an syntax tree, a tree in which each leaf represents an operand and each interior node an operator.

56. What is a three-address code?

Three-address code is a sequence of statements, typically of the general form $A := B \text{ op } C$, where A,B and C are either programmer-defined names, constants or compiler-generated temporary names; op stands for any operator, such as fixed- or floating-point arithmetic operator, or a logical operator on Boolean-valued data.

57. Write the three address code for the assignment statement $a := b * -c + b * -c$

```

t1 := -c
t2 := b * t1
t3 := -c
t4 := b * t3
t5 := t2 + t4

```

a := t5

58. Name any four types of three-address statements?

- Assignment statements of the form $x := y \text{ op } z$
- Assignment instruction of the form $x := \text{op } y$
- Copy statement of the form $x := y$
- The unconditional jump goto L.

59. What are the representations of three-address statements?

A three address statement is an abstract form of intermediate code. There are three representation are available. They are

- Quadruples
- Triples
- Indirect triples

60. Write notes on quadruples?

A quadruple is a record structure with four fields, which we call op, arg1, arg2 and result. The op field contains an internal code for the operator. The three address statement $x := y \text{ op } z$ is represented by placing y in arg1, z in arg2, and x in result. The quadruple representation for $x := y * z$ is shown as below

	Op	Arg1	Arg2	Result
(0)	*	Y	z	x

The quadruple representation for $x := -y$ is shown as below

	Op	Arg1	Arg2	Result
(0)	-	Y		x

61. Write notes on triples?

To avoid entering temporary names into the symbol table, we might refer to a temporary value by the position of the statement that computes it. The three address codes are represented in triples in three fields as op, arg1 and arg2. The triple representation for the assignment statement $a := b * -c + b * -c$ is shown as below

	Op	arg1	arg2
(0)	Uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

62. Define indirect triples?

An implementation of three-address code, which has been considered, is that of listing pointers to triples, rather than listing the triples themselves. This implementation is naturally called indirect triples.

63. Write the translation scheme for the statement

If $A < B$ then 1 else 0
 if $A < B$ goto (4)
 $T := 0$
 goto (5)
 $T := 1$

64. Write a grammar for array reference?

$A \rightarrow L := E$
 $L \rightarrow \text{id}[\text{elist}] \mid \text{id}$
 $\text{elist} \rightarrow \text{elist}, E \mid E$
 $E \rightarrow E + E \mid (E) \mid L$

65. Write a grammar for simple procedure call statement?

$S \rightarrow \text{call id} (\text{elist})$
 $\text{elist} \rightarrow \text{elist}, E$
 $\text{elist} \rightarrow E$

66. Write a grammar for declaration statement?

$D \rightarrow \text{integer namelist} \mid \text{namelist}$
 $\text{namelist} \rightarrow \text{id}, \text{namelist} \mid \text{id}$

67. Write the grammar for Boolean expression?

$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E \mid (E) \mid \text{id relop id} \mid \text{true} \mid \text{false}$

We use the attribute *op* to determine which of the comparison operators $<$, \geq , \leq , \neq , or $>$ is represented by *relop*.

68. What are the methods of translating Boolean expression?

There are two principal methods of representing the value of a Boolean expression. The first method is to encode true or false numerically and to evaluate a Boolean expression analogously to an arithmetic expression. The second principle 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.

69. Define short-circuit 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.

70. Write the grammar for if-then, if-then-else, and while-do statements?

$S \rightarrow \text{if } E \text{ then } S1$
 $\quad \mid \text{if } E \text{ then } S1 \text{ else } S2$
 $\quad \mid \text{while } E \text{ do } S1$

71. Write the three address code for the following statements?

While $a < b$ do
 If $c < d$ then
 $x := y + z$
 Else
 $x := y - z$

Answer

L1 : if $a < b$ go to L2
 goto Lnext
 L2 : if $c < d$ goto L3
 goto L4
 L3 : t1 := $y + z$
 $x := t1$

```

        goto L1
L4 : t2 := y - z
      x := t2
      goto L1
Lnext :

```

72. Define back patching?

The main problem with generating code for Boolean expressions and flow-of-control statements in a single pass is that during one single pass we may not know the labels that control must go to at the time the jump statements are generated. We can get around this problem by generating a series of branching statements with the targets of the jumps unspecified. The labels will be filled in when the proper label can be determined. We call this subsequent filling in of labels back-patching.

73. Write the three address code for the expression given below

$a < b$ or $c < d$ and $e < f$

Answer

```

100 : if a < b goto 103
101 : t1 := 0
102 : goto 104
103 : t1 := 1
104 : if c < d goto 107
105 : t2 := 0
106 : goto 108
107 : t2 := 1
108 : if e < f goto 111
109 : t3 := 0
110 : goto 112
111 : t3 := 1
112 : t4 := t2 and t3
113 : t5 := t1 or t4

```

74. Write the grammar for procedure calls?

The grammar for a simple procedure call statement is

```

S → call id (Elist)
Elist → Elist, E
Elist → E

```

Unit V

1. What is meant by code generation phase?

The final phase in compiler model is the code generator. It takes as input an intermediate representation of the source program and produces as output an equivalent target program. In code generation the output code must be correct and of high quality.

2. What code to generate to manage activation records at run time?

Two standard storage-allocation strategies were presented, namely, static allocation and stack allocation. In static allocation, the position of an activation record in memory is fixed at compile time. In stack allocation, a new activation record is pushed onto the stack for each execution of a procedure.

3. Write the various three address modes together with their assembly-language forms and associated costs?

MODE	FORM	ADDRESS	ADDED COST
Absolute	M	M	1
Register	R	R	0
Indexed	c(R)	c+ contents(R)	1
Indirect register	*R	contents(R)	0
Indirect indexed	*c(R)	contents(c+contents(R))	1

4. What is the instruction cost of the following instruction?

MOV b, R0

MOV c, R0

MOV R0,a

The instruction cost is 6

5. What are the two standard storage allocation strategies?

- a) Static allocation.
- b) Stack allocation.

6. Write the three address statements related to procedure calls?

- a) Call
- b) Return
- c) halt
- d) action

7. Write notes on stack allocation?

In stack allocation the position of the record for an activation of a procedure is usually stored in a register, so words in the activation record can be accessed as offsets from the value in the register. The indexed addressed mode of the target machine is convenient for this purpose.

8. Define code optimization?

The term code optimization refers to techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program.

9. Define basic blocks?

The sequences of consecutive statements which may be entered only at the beginning, and when entered are executed in sequence without halt or possibly of branch.

10. What are the structure preserving transformations of basic blocks?

- a) common sub expression elimination.
- b) dead-code elimination.
- c) renaming of temporary variables.
- d) interchange of two independent adjacent statements.

11. Define flow graph?

The basic blocks and their successor relationships are portrayed by a directed graph called a flow graph. The nodes of the flow graph are the basic blocks.

12. Define loops?

- Loop is a collection of nodes that
- (i) It is strongly connected.
 - (ii) It has a unique entry.

13. What is a preheader?

Several transformations require us to move statements “before the header”. We therefore begin treatment of a loop L by creating a new block, called the preheader. The preheader has only the header as successor, and all edges which is formerly entered the header L from outside L instead enter the preheader.

14. Define code motion?

An important source of modification is called code motion, where we take a computation that yields the same result independent of the number of times through the loop and place it before the loop.

15. What is reduction in strength?

The replacement of an expensive operation by a cheaper one is termed reduction in strength. That is replacing a multiplication operation by an addition operation.

16. Define DAG?

A useful data structure for automatically analyzing basic blocks is a directed acyclic graph (DAG). A DAG is a directed graph with no cycles.

17. What are the applications of DAG's?

- a) Automatically detect common subexpressions.
- b) Determine which identifiers have their values used in the block.
- c) Determine which statements compute values which could be used outside the block.

18. Define Dominators?

A flow graph dominates node n, written $d \text{ DOM } n$, if every path from the initial node of the flow graph to n goes through d. Under this definition, every node dominates itself, and the entry of a loop dominates all nodes in the loop.

19. What is reducible flow graphs?

A flow graph G is reducible if and only if we can partition the edges into two disjoint groups, forward edges and backward edges, with the following two properties.

- a) The forward edges form an acyclic graph in which every node can be reached from the initial node of G.
- b) The back edges consist only of edges whose heads dominates their tails.

20. Define depth-first ordering?

The depth first ordering of the nodes is the reverse of the order in which we last visit the nodes in the preorder traversal.

21. Define induction variable?

Induction variable of loop L to be either a basic induction variable or a name J for which there is a basic induction variable I such that each time J is assigned in L, J's value is the same linear function of the value of I.

22. Define loop unrolling?

Loop unrolling is to avoid a test at every iteration by recognizing that the number of iteration is constant and replication the body of the loop.

23. What are the problems in code generation?

- a) What instruction should we generate?
- b) In what order should we perform computation.
- c) What registers should we use?

24. What is peephole optimization?

Peephole optimization is a technique used in many compilers, 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.

25. Write the algorithm for constructing the natural loop?

```

Procedure INSERT(m);
If M is not in Loop then
Begin
Loop:= Loop U {m};
Push m onto stack
End
Main()
STACK:=empty;
Loop:={d}
INSERT(n)
While STACK is not empty do
Begin
Pop m, the first element of STACK, off STACK
For each predecessor p of do INSERT (P)
END

```

Unit I

1. Explain the phases of the compiler?

A compiler operates in phases, each of which transforms the source program from one representation to another. The various phases are

- a) lexical analyzer
- b) syntax analyzer
- c) semantic analyzer

- d) intermediate code generator
- e) code optimizer
- f) code generator

Two other activities, symbol table management and error handling interact with the six phases.

Lexical analyzer:-

The lexical analyzer reads the source program one character at a time, carving the source program into a sequence of atomic units called tokens. Identifiers, keywords, constants, operators and punctuation symbols are typical tokens.

Syntax analyzer:-

Syntax analysis is also called parsing. It involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output.

Semantic analyzer:-

The semantic analysis phase checks the source program for semantic errors and gathers type information for the subsequent code generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operand of expressions and statements.

Intermediate code generator:-

After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program. This intermediate representation should have two important properties; it should be easy to produce and easy to translate into the target program.

Code optimizer:-

Certain compilers apply transformations to the output of the intermediate code generator. It is used to produce an intermediate-language from which a faster or smaller object program can be produced. This phase is called optimization phase. Types of optimization are local optimization and loop optimization.

Code generator:-

The final phase of the compiler is the generation of target code, consisting normally of relocatable machine code or assembly code.

Symbol Table management:-

A symbol table is a data structure containing a record for each identifier with fields for the attribute of the identifier. The data structure allows us to find the record for each identifier quickly to store and retrieve data from that record quickly.

Error Detection and Reporting:-

The lexical phase can detect errors where the characters remaining in the input do not form any token of the language. Errors where the token stream violates the structure rules of the language are determined by the syntax analysis phase.

2. Briefly discuss on the cousins of the compiler?

The cousins of the compiler are

- a) Preprocessors
- b) Assemblers
- c) Two-pass assembly
- d) Loaders and link-editors

Preprocessors:-

Preprocessors produce input to compilers. They may perform the following functions.

Macro processing : A preprocessor may also allow a user to define macros that are shorthand's for longer constructs.

File inclusion: A preprocessor may include header files into the program text.

“Rational” preprocessors: These processors augment older languages with more modern flow-of-control and data-structuring facilities.

Language extensions: These processors attempt to add capabilities to the language by what amounts to built-in macros.

Assemblers:-

Some compilers produce assembly code that is passed to an assembler for further processing.

Assembly code is a mnemonic version of machine code, which names are used instead of binary codes for operations and names are also given to memory addresses.

Two-pass assembly:-

The simplest form of assembler makes two passes over the input, where a pass1 consist of reading an input file once. In the first pass, all the identifiers that denote the storage locations are found and stored in a symbol table. Identifiers are assigned storage location.

In the second pass, the assembler scans the input again. It translates each operation code into the sequence of bits representing that operation in machine language. The output of second pass is usually a relocatable machine code.

Loaders and link-editors:-

A program called a loader performs the two function of loading and link-editing. The process of loading consists of taking relocatable machine code, altering the relocatable addresses and placing the altered instruction and data in memory at the proper locations.

3. Explain about input buffering?
 1. Buffer pairs
 2. Sentinels
4. How to specify the tokens in a language?
 - Strings and languages
 - Operations on languages
 - Regular expressions
 - Regular definitions
 - Notational Shorthands
 - Non regular sets

Unit II

5. Explain about context free grammars?

Non-terminals are special symbols that denote set of strings. “Syntactic variable” and “syntactic category” is other names for non-terminals. One non-terminal is select as start symbol. The productions (rewriting rules) define the ways in which the syntactic categories may be built from one another. Production consists of a non-terminal followed by an arrow, followed by a string of non-terminals.

Figure in fig 4.1

Production rule for expression: -

The terminal symbols are

Id, +, -, *, /, ^, (,).

Productions are

Expression \rightarrow expression operator expression

Expression \rightarrow (expression)

Expression \rightarrow -expression

Expression \rightarrow id

Operator \rightarrow +

Operator \rightarrow -

Operator \rightarrow *

Operator \rightarrow /

Operator \rightarrow ^

Notational convention for production rule: -

1. Non-terminals are lowercase names, italic capital letters, the letter S (start symbol).
2. Terminals are lowercase letters a, b, c.. .. Operator symbol +, punctuation parenthesis, comma etc.
Digits -- 0,1...9.
3. X, Y, Z represent grammar symbols, that is either non-terminal or terminal.
4. α, β, γ represent strings of grammar symbols. Production rule $A \rightarrow \alpha$
5. If $A \rightarrow \alpha_1, A \rightarrow \alpha_2, A \rightarrow \alpha_k$, are may write $A \rightarrow \alpha_k$
6. The left side of the first production is the start symbol.

Ex: Using shorthands, the above grammars is written as

$$E \rightarrow EAE \mid (E) \mid \cdot E \mid id$$

$$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$$

E and A are non-terminals, E the start symbol.

Remaining are terminals

6. Briefly explain on the stack implementation of shift-reduce parsing?

Shift Reduce –Parsing

Bottom – up shift parsing is called shift-reduce-parsing. At each step a string matching on the right side of a production is replaced by the symbol on the left.

Ex: consider the grammar

$$S \rightarrow aACBe$$

$$A \rightarrow Ab \mid b$$

$$B \rightarrow d$$

Consider the string abcde

Abcde

$$A \rightarrow b: aAbcde$$

$$A \rightarrow Ab: aAcde$$

$$B \rightarrow d: aAcBe$$

$$S \rightarrow aAcBe: S$$

Shift reduce parsing is one of finding and reducing handles

Handles:

*

If $S \Rightarrow \alpha Aw \Rightarrow \alpha \beta$, then $A \rightarrow \beta w$

Handle Pruning:

The right derivation is reverse, often called a canonical reduction. Sequence is obtained by handle parsing.

Stack implementation of shift reduce parsing:

A convenient way to implement a shift reduce parser is to use a stack at an input buffer. Use \$ to mark the bottom of stack. The parser shift zero or more input symbols onto the stack until a handle β is on top of the stack. The parser then reduces β to the left side of the production.

Ex: Steps of shift reduced parser is passing id, +id2 x id3 according to the grammar

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

	Stack	Input	Action
(1)	\$	id ₁ + id ₂ * id ₃	Shift
(2)	\$ id ₁	\$	Reduce by E -> id
(3)	\$ E	+ id ₂ * id ₃ \$ ”	Shift
(4)	\$E+	id ₂ * id ₃ \$	”
(5)	\$E+ id ₂	* id ₃ \$	reduce by E->id

(6)	\$E+E	"	shift
(7)	\$E+E	id ₃ \$	"
(8)	\$E+E*	\$	reduce by E ->id
(9)	id ₃ \$E+E*E	\$	reduce by E->E * E
(10)	\$E+E	\$	" E -> E +
(11)	\$ E	\$	E accept

Four possible action of shift reduce parser are (1) shift (2) reduce (3) accept (4) error

1. In shift action, the near i/p symbol is shifted to the top of stack.
2. In reduce action; the parser knows the right end of the parser is at the top of the stack.
3. In accept action, the parser give successful completion.
4. In error action, the parser discovers syntax error.

Constructing a Parse Tree

The bottom – up tree construction has two aspects.

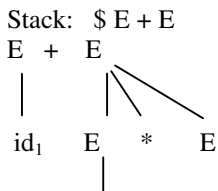
- (1) When we shift an input symbol a on to the stack we create a one node tree labeled a.
- (2) When we reduce X₁, X₂.... X_n to A, we create a new node labeled as A.

Ex:-

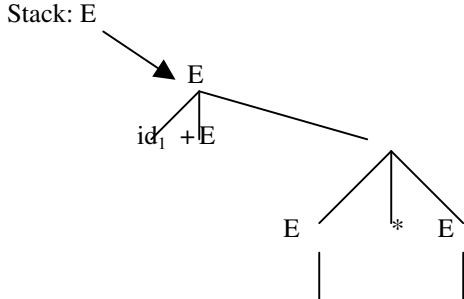
Stack: \$id,
o
id₁
fig (a)

stack: \$E
E
id₁

fig (b)



id₂
fig(c) after reducing id₁ + id₂* id₃to E + E
Stack: E



id_1 id_3