GLOSSARY

1. Compiler - a program that reads a program written in one language and translates it in to an equivalent program in another language.

2. Analysis part - breaks up the source program into constituent pieces and creates an intermediate representation of the source program.

3. Synthesis part - constructs the desired target program from the intermediate representation.

4. Structure editor - takes as input a sequence of commands to build a source program. Pretty printer - analyses a program and prints it in such a way that the structure of the program becomes clearly visible.

5. Static checker - reads a program, analyses it and attempts to discover potential bugs without running the program.

6. 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.

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

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

9. Loader - is a program that performs the two functions: Loading and Link editing

10. Loading - taking relocatable machine code, altering the relocatable address and placing the altered instructions and data in memory at the proper locations.

11. Link editing - makes a single program from several files of relocatable machine code.

12. Preprocessor - produces input to compilers and expands macros into source language statements.

13. Symbol table - a data structure containing a record for each identifier, with fields for the attributes of the identifier. It allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

14. Assembler - a program, which converts the source language in to assembly languageLexeme - a sequence of characters in the source program that is matched by the pattern for a token.

16. Regular set - language denoted by a regular expression.

17. Sentinel - a special character that cannot be part of the source program. It speeds-up the lexical analyzer.

18. Regular expression - a method to describe regular language Rules:

19. Recognizers - machines which accept the strings belonging to certain language.

20. Parser - is the output of syntax analysis phase.

21. Error handler - report the presence of errors clearly and accurately and recover from each error quickly enough to be able to detect subsequent errors.

22. Context free grammar - consists of terminals, non-terminals, a start symbol, and productions.

23. Terminals - basic symbols from which strings are formed. It is a synonym for terminal. Nonterminals - syntactic variables that denote sets of strings, which help define the language generated by the grammar.

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

Context free language - a language that can be generated by a grammar.

Left most derivations – the leftmost nonterminal in any sentential form is replaced at each step.

25. Canonical derivations - rightmost nonterminal is replaced at each step are termed.

26. Parse tree - a graphical representation for a derivation that filters out the choice regarding replacement order.

27. Ambiguous grammar - A grammar that produces more than one parse tree for some sentence is said to be ambiguous.

Left recursive$_{\Rightarrow}$- A grammar is a left recursive if it has a nonterminal A such that there is a derivation A $\Rightarrow$ Aα for some string α.

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

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

34. Top Down parsing - Starting with the root, labeled, does the top-down construction of a parse tree with the starting nonterminal.

35. Recursive Descent Parsing - top down method of syntax analysis in which we execute a set of recursive procedures to process the input.

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

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

38. 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.

39. Handle - 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.

40. canonical derivations - process of obtaining rightmost derivation in reverse.

41. Viable prefixes - the set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser.

42. Operator grammar - A grammar is operator grammar if no production rule involves “ε” on the right side.

43. LR parsing method - most general nonbacktracking shift-reduce parsing method.

44. goto function - takes a state and grammar symbol as arguments and produces a state.

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

47. Kernel - The set of items which include the initial item, S S, and all items not at the left end are known as kernel items.

48. Non kernel items - The set of items, which have their dots at the left end, are known as non kernel items.

49. Postfix notation - is a linearized representation of a syntax tree50. Syntax directed definition - Syntax trees for assignment statement are produced by the syntax directed definition.

51. 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.

52. Quadruple - is a record structure with four fields, op, arg1, arg2 and result.

53. 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.

54. Triples - is a record structure with three fields op, arg1, arg2

55. Declaration - The process of declaring keywords, procedures, functions, variables, and statements with proper syntax.

56. Boolean Expression - Expressions which are composed of the Boolean operators (and, or, and not) applied to elements that are Boolean variables or relational expressions.

57. Calling sequence - A sequence of actions taken on entry to and exit from each procedure.

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

59. 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.

60. Flow graph - The basic block and their successor relationships shown by a directed graph is called a flow graph.

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

62. Back-end - Intermediate to binary translation is usually done by a separate compilation pass called back end.

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

64. Multiregister operations – operations that requires more than one register to perform.

65. Cost of an instruction - one plus the costs associated with the source and destination address modes. This cost corresponds to the length of the instructionRecursive procedure - A procedure is recursive a new activation can begin before an earlier activation of the same procedure has ended.

66. DAG - a directed acyclic graph with the labels on nodes:

67. Memory management - Mapping names in the source program to addresses of data object in run time memory.

68. Backpatching - Process of leaving a blank slot for missing information and fill in the slot when the information becomes available.

69. Algebraic transformation - change the set of expressions computed by a basic blocks into an algebraically equivalent set.

70. Register descriptor - keeps track of what is currently in each register.

71. Address descriptor - keeps track of the location where the current value of the name can be found at run time.

72. Local optimization - The optimization performed within a block of code.

73. Constant folding - Deducing at compile time that the value of an expression is a constant and using the constant instead.

74. 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.

75. 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.

76. Reduction in strength - replaces an expensive operation by a cheaper one such as a multiplication by an addition.

77. Loop invariant computation - An expression that yields the same result independent of the number of times the loop is executed.

78. Static allocation - the position of an activation record in memory is fixed at run time

79. Activation tree - A tree which depicts the way of control enters and leaves activations.

80. Control stack - A stack which is used to keep track of live procedure actions.

81. Heap - A separate area of run-time memory which holds all other informationPadding - Space left unused due to alignment consideration.

83. Call sequence - allocates an activation record and enters information into its fields

84. Return sequence - restores the state of the machine so that calling procedure can continue execution.

85. Dangling reference - occurs when there is storage that has been deallocated.

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

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

88. Block - a statement containing its own data declaration.

89. Access link - a pointer to each activation record which obtains a direct implementation of lexical scope for nested procedure.

90. Environment - refers to a function that maps a name to a storage location.

91. State - refers to a function that maps a storage location to the value held there.

92. Peephole optimization - a technique used in many compliers, in connection with the optimization of either intermediate or object code.