COMPILER DESIGN

1. What is a compiler?

A compiler is a program that reads a program written in one language –the source language and translates it into an equivalent program in another language-the target language. The compiler reports to its user the presence of errors in the source program.

2. What are the two parts of a compilation? Explain briefly.

Analysis and Synthesis are the two parts of compilation.

The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.

The synthesis part constructs the desired target program from the intermediate representation.

3. List the subparts or phases of analysis part.

Analysis consists of three phases:

- Linear Analysis.
- Hierarchical Analysis.
- Semantic Analysis.

4. Depict diagrammatically how a language is processed.

Skeletal source program

↓

Preprocessor

↓

Source program

↓

Compiler

↓

Target assembly program

↓

Assembler

↓

Relocatable machine code

↓

Loader/ link editor ←library, relocatable object files

↓

Absolute machine code

5. What is linear analysis?

Linear analysis is one 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.

Also called lexical analysis or scanning.

6. List the various phases of a compiler.

The following are the various phases of a compiler:

- Lexical Analyzer
- Syntax Analyzer
- Semantic Analyzer
- Intermediate code generator
- Code optimizer
- Code generator

7. What are the classifications of a compiler?

Compilers are classified as:

- Single- pass
- Multi-pass
- Load-and-go

- Debugging or optimizing

8. 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.

Whenever an identifier is detected by a lexical analyzer, it is entered into the symbol table. The attributes of an identifier cannot be determined by the lexical analyzer.

9. Mention some of the cousins of a compiler.

Cousins of the compiler are:

- Preprocessors
- Assemblers
- Loaders and Link-Editors

10. List the phases that constitute the front end of a compiler.

The front end consists of those phases or parts of phases that depend primarily on the source language and are largely independent of the target machine. These include

- Lexical and Syntactic analysis
- The creation of symbol table
- Semantic analysis
- Generation of intermediate code

A certain amount of code optimization can be done by the front end as well. Also includes error handling that goes along with each of these phases.

11. Mention the back-end phases of a compiler.

The back end of compiler includes those portions that depend on the target machine and generally those portions do not depend on the source language, just the intermediate language. These include

- Code optimization

- Code generation, along with error handling and symbol- table operations.

12. Define compiler-compiler.

Systems to help with the compiler-writing process are often been referred to as compiler-compilers, compiler-generators or translator-writing systems.

Largely they are oriented around a particular model of languages , and they are suitable for generating compilers of languages similar model.

13. List the various compiler construction tools.

The following is a list of some compiler construction tools:

- Parser generators
- Scanner generators
- Syntax-directed translation engines
- Automatic code generators
- Data-flow engines

14. Differentiate tokens, patterns, lexeme.

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

15. List the operations on languages.

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

- **Positive Closure –** L+ (one or more concatenations of L)

16. Write a regular expression for an identifier.

An identifier is defined as a letter followed by zero or more letters or digits.

The regular expression for an identifier is given as

letter (letter | digit)*

17. Mention the various notational shorthands for representing regular expressions.

- One or more instances (+)
- Zero or one instance (?)
- Character classes ([abc] where a,b,c are alphabet symbols denotes the regular expressions a | b | c.)
- Non regular sets

18. What is the function of a hierarchical analysis?

Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning.

Also termed as Parsing.

19. What does a semantic analysis do?

Semantic analysis is one in which certain checks are performed to ensure that components of a program fit together meaningfully.

Mainly performs type checking.

20. List the various error recovery strategies for a lexical analysis.

Possible error recovery actions are:

- Panic mode recovery
- Deleting an extraneous character

- Inserting a missing character
- Replacing an incorrect character by a correct character
- Transposing two adjacent characters

UNIT 2

1. **Define parser.**

Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning.

Also termed as Parsing.

2. **Mention the basic issues in parsing.**

There are two important issues in parsing.

- Specification of syntax
- Representation of input after parsing.

3. **Why lexical and syntax analyzers are separated out?**

Reasons for separating the analysis phase into lexical and syntax analyzers:

- Simpler design.
- Compiler efficiency is improved.
- Compiler portability is enhanced.

4. **Define a context free grammar.**

A context free grammar G is a collection of the following

- V is a set of non terminals
- T is a set of terminals
- S is a start symbol
- P is a set of production rules

G can be represented as G = (V,T,S,P)

Production rules are given in the following form

Non terminal → (V U T)*

5. **Briefly explain the concept of derivation.**

Derivation from S means generation of string w from S. For constructing derivation two things are important.

i) Choice of non terminal from several others.

ii) Choice of rule from production rules for corresponding non terminal.

Instead of choosing the arbitrary non terminal one can choose

i) either leftmost derivation – leftmost non terminal in a sentinel form

ii) or rightmost derivation – rightmost non terminal in a sentinel form

6. **Define ambiguous grammar.**

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

i.e. both leftmost and rightmost derivations are same for the given sentence.

7. **What is a operator precedence parser?**

A grammar is said to be operator precedence if it possess the following properties:

1. No production on the right side is ε.

2. There should not be any production rule possessing two adjacent non terminals at the right hand side.

8. **List the properties of LR parser.**

1. LR parsers can be constructed to recognize most of the programming languages for which the context free grammar can be written.

2. The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.

3. LR parsers work using non backtracking shift reduce technique yet it is efficient one.

9. **Mention the types of LR parser.**

- SLR parser- simple LR parser
- LALR parser- lookahead LR parser
- Canonical LR parser

10. **What are the problems with top down parsing?**

The following are the problems associated with top down parsing:

- Backtracking
- Left recursion
- Left factoring
- Ambiguity

11. **Write the algorithm for FIRST and FOLLOW.**

FIRST

1. If X is terminal, then FIRST(X) IS {X}.

2. If X → ε is a production, then add ε to FIRST(X).

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

FOLLOW

1. Place $ in FOLLOW(S),where S is the start symbol and $ is the input right endmarker.

2. If there is a production A → αBβ, then everything in FIRST(β) except for ε is placed in FOLLOW(B).

3. If there is a production A → αB, or a production A→ αBβ where FIRST(β) contains ε , then everything in FOLLOW(A) is in FOLLOW(B).

12. **List the advantages and disadvantages of operator precedence parsing.**

Advantages

This typeof parsing is simple to implement.

Disadvantages

1. The operator like minus has two different precedence(unary and binary).Hence it is hard to handle tokens like minus sign.

2. This kind of parsing is applicable to only small class of grammars.

13. **What is dangling else problem?**

Ambiguity can be eliminated by means of dangling-else grammar which is show below:

stmt → if expr then stmt

| if expr then stmt else stmt

| other

14. **Write short notes on YACC.**

YACC is an automatic tool for generating the parser program.

YACC stands for Yet Another Compiler Compiler which is basically the utility available from UNIX.

Basically YACC is LALR parser generator.

It can report conflict or ambiguities in the form of error messages.

15. **What is meant by handle pruning?**

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

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

S = γ0 => γ1…=> γn-1 => γn = w

16. **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, production A → XYZ yields the four items

A→.XYZ

A→X.YZ

A→XY.Z

A→XYZ.

17. **What is meant 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. An equivalent definition of a viable prefix is that it is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

18. **Define handle.**

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

A handle of a right – sentential form γ is a production A→β 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. That is , if S =>αAw =>αβw,then A→β in the position following α is a handle of αβw.

19. **What are kernel & non-kernel items?**

Kernel items, whish include the initial item, S'→ **.**S, and all items whose dots are not at the left end.

Non-kernel items, which have their dots at the left end.

20. **What is phrase level error recovery?**

Phrase level error recovery is implemented by filling in the blank entries in the predictive parsing table with pointers to error routines. These routines may change, insert, or delete symbols on the input and issue appropriate error messages. They may also pop from the stack.

UNIT 3

1. **What are the benefits of intermediate code generation?**
   - A Compiler for different machines can be created by attaching different back end to the existing front ends of each machine.
   - A Compiler for different source languages can be created by proving different front ends for corresponding source languages t existing back end.
   - A machine independent code optimizer can be applied to intermediate code in order to optimize the code generation.

2. **What are the various types of intermediate code representation?**

There are mainly three types of intermediate code representations.

- Syntax tree
- Posix
- Three address code

3. **Define backpatching.**

Backpatching is the activity of filling up unspecified information of labels using appropriate semantic actions in during the code generation process.In the semantic actions the functions used are mklist(i),merge_list(p1,p2) and backpatch(p,i)

4. **Mention the functions that are used in backpatching.**

- mklist(i) creates the new list. The index i is passed as an argument to this function where I is an index to the array of quadruple.
- merge_list(p1,p2) this function concatenates two lists pointed by p1 and p2. It returns the pointer to the concatenated list.
- backpatch(p,i) inserts i as target label for the statement pointed by pointer p.

5. **What is the intermediate code representation for the expression a or b and not c?**

The intermediate code representation for the expression a or b and not c is the three address sequence

t1 := not c

t2 := b and t1

t3 := a or t2

6. **What are the various methods of implementing three address statements?**

The three address statements can be implemented using the following methods.

- Quadruple : a structure with atmost four fields such as operator(OP),arg1,arg2,result.
- Triples : the use of temporary variables is avoided by referring the pointers in the symbol table.
- Indirect triples : the listing of triples has been done and listing pointers are used instead of using statements.

7. **Give the syntax-directed definition for if-else statement.**

1. S → if E then S1

E.true := new_label()

E.false :=S.next

S1.next :=S.next

S.code :=E.code | | gen_code(E.true ': ') | | S1.code

2. S → if E then S1 else S2

E.true := new_label()

E.false := new_label()

S1.next :=S.next

S2.next :=S.next

S.code :=E.code | | gen_code(E.true ': ') | | S1.code| | gen_code('go to',S.next) | |gen_code(E.false ':') | | S2.code

UNIT 4

1. **Mention the properties that a code generator should possess.**

- The code generator should produce the correct and high quality code. In other words, the code generated should be such that it should make effective use of the resources of the target machine.

- Code generator should run efficiently.

2. **List the terminologies used in basic blocks.**

    - **Define and use –** the three address statement a:=b+c is said to define a and to use b and c.

    - **Live and dead –** the name in the basic block is said to be live at a given point if its value is used after that point in the program. And the name in the basic block is said to be dead at a given point if its value is never used after that point in the program.

3. **What is a flow graph?**

    A flow graph is a directed graph in which the flow control information is added to the basic blocks.

    - The nodes to the flow graph are represented by basic blocks

    - The block whose leader is the first statement is called initial block.

    - There is a directed edge from block B1 to block B2 if B2 immediately follows B1 in the given sequence. We can say that B1 is a predecessor of B2.

4. **What is a DAG? Mention its applications.**

    Directed acyclic graph(DAG) is a useful data structure for implementing transformations on basic blocks.

    DAG is used in

    - Determining the common sub-expressions.

    - Determining which names are used inside the block and computed outside the block.

    - Determining which statements of the block could have their computed value outside the block.

- Simplifying the list of quadruples by eliminating the common su-expressions and not performing the assignment of the form x := y unless and until it is a must.

5. **Define peephole optimization.**

Peephole optimization is a simple and effective technique for locally improving target code. This technique is applied to improve the performance of the target program by examining the short sequence of target instructions and replacing these instructions by shorter or faster sequence.

6. **List the characteristics of peephole optimization.**

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

7. **How do you calculate the cost of an instruction?**

The cost of an instruction can be computed as one plus cost associated with the source and destination addressing modes given by added cost.

MOV R0,R1 1

MOV R1,M 2

SUB 5(R0),*10(R1) 3

8. **What is a basic block?**

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

Eg. t1:=a*5

t2:=t1+7

t3:=t2-5

t4:=t1+t3

t5:=t2+b

9. **How would you represent the following equation using DAG?**

a:=b*-c+b*-c

UNIT 5

1. **Mention the issues to be considered while applying the techniques for code optimization.**

   - The semantic equivalence of the source program must not be changed.

   - The improvement over the program efficiency must be achieved without changing the algorithm of the program.

2. **What are the basic goals of code movement?**
   - To reduce the size of the code i.e. to obtain the space complexity.
   - To reduce the frequency of execution of code i.e. to obtain the time complexity.

3. **What do you mean by machine dependent and machine independent optimization?**

   - The machine dependent optimization is based on the characteristics of the target machine for the instruction set used and addressing modes used for the instructions to produce the efficient target code.

   - The machine independent optimization is based on the characteristics of the programming languages for appropriate programming structure and usage of efficient arithmetic properties in order to reduce the execution time.

4. **What are the different data flow properties?**

   - Available expressions

   - Reaching definitions

   - Live variables

   - Busy variables

5. **List the different storage allocation strategies.**

   The strategies are:

   - Static allocation

- Stack allocation
- Heap allocation

6. **What are the contents of activation record?**

The activation record is a block of memory used for managing the information needed by a single execution of a procedure. Various fields f activation record are:

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

7. **What is dynamic scoping?**

In dynamic scoping a use of non-local variable refers to the non-local data declared in most recently called and still active procedure. Therefore each time new findings are set up for local names called procedure. In dynamic scoping symbol tables can be required at run time.

8. **Define symbol table.**

Symbol table is a data structure used by the compiler to keep track of semantics of the variables. It stores information about scope and binding information about names.

9. **What is code motion?**

Code motion is an optimization technique in which amount of code in a loop is decreased. This transformation is applicable to the expression that yields the same result independent of the number of times the loop is executed. Such an expression is placed before the loop.

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

The source code should be such that it should produce minimum amount of target code.

There should not be any unreachable code.

Dead code should be completely removed from source language.

The optimizing compilers should apply following code improving transformations on source language.

i) common subexpression elimination

ii) dead code elimination

iii) code movement

iv) strength reduction

11. **What are the various ways to pass a parameter in a function?**

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

12. **Suggest a suitable approach for computing hash function.**

Using hash function we should obtain exact locations of name in symbol table.

The hash function should result in uniform distribution of names in symbol table.

The hash function should be such that there will be minimum number of collisions. Collision is such a situation where hash function results in same location for storing the names.