Pattern

- It specifies a set of rules that a scanner follow to create a token.
- It specifies a set of rules that a scanner follows ~~in order to~~ to create a token.

Q1. Define lexeme, token, pattern.

lexeme: A sequence of characters in the source code matched by given predefined language ~~for~~ rules for every lexeme to be specified ~~to be~~ as a valid token.

eg: ~~int is a~~ main is a lexeme of type identifier.

Token: A sequence of characters considered as an unit and cannot be broken down further.

eg int a = 10;

int (identifier), a (identifier), = (operator), 10 (const)

Pattern: It specifies ~~is~~ set of rules that the scanner follows to create a token.

Q2. Describe various Compilier Tools.

1. Parser Generator:
   - It produces a syntax analyzer from the input that is based on context free grammer.
   - It is useful as syntax analysis phase is highly complex and consume more manual and compilation time.

   eg: PIC, EQM
2. Scanner Generator:
   - Accepts Regular Expression as an input and produces lexical analyzer as an output.

- ~~lexical analyzer reads the character from the source code and~~
lexical analyzer reads the source code character by character and produces a stream of tokens.
eg: ~~LEX~~ Tool

Q Describe various Compiler Tools.

1 Scanner Generator.
- It takes Regular Expression as an input and produces lexical analyzer as an output.
- Lexical analyzer reads the source code character by character and produces a string of tokens.
eg: LEX Tool.

2 Parser Generator
- It checks whether the token's syntax is correct or not.
- If the syntax is correct then it produces parse string.
~~- If the syntax is incorrect then it will send a message to the user.~~
- If there are any errors in the token then it will send a message to the user.
eg: YACC tool.

3. Syntax directed Translation Engine.
- It contants a collection of routines which are useful to traverse the parse tree and produces intermediate code as an output.
- It is useful in intermediate code representation.
eg:

4. Data flow Analysis Engine
- It is useful in code optimization phase.
- It is a key part of code optimization that gathers the information.

5. Code Generator

- It accepts optimized intermediate code as an input and produces machine code as an output.

6. Compiler Construction Toolkits

- It contains integrated collection of routines in order to construct a compiler.

Q3. What is ~~ambigous~~ ambiguous grammer? What are the ways to deal with this grammer?

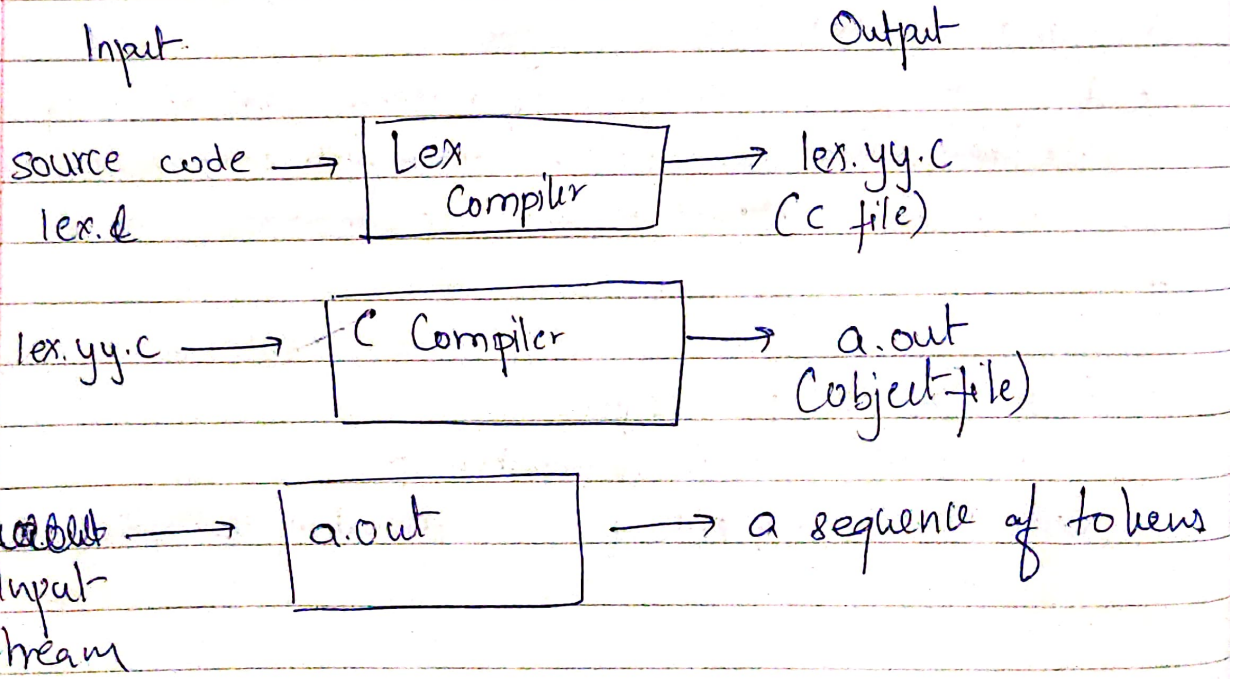
Ambiguous Grammer:

A grammer is said to be ambigaaus if it generates more than one phase tree. for the corresponding input ~~aaaaa~~ string

The ways we can deal with this grammer is:

1) left-most derivation
2) Right-most derivation

Q4. Describe the specification and compilation process of LEX Tools.



Structure of LEX

i) Declaration
ii) Translation Rules
iii) Auxiliary functions.

1) Declaration: It is mainly used to declare C variables and consts.

```
%{
    variables, consts;
%}
```

Useful for declaring regular expressions.

2) Translation Rule:

Syntax: pattern {Action}
RE ↓ ↳ C language statements.

```
%%
pattern1 {Action1}
pattern2 {Action2}
pattern3 {Action3}
%%
```

3) Auxiliary functions:

Q. Explain the syntax directed translation for flow of control statement.

Q. Illustrate the structure of an activation record with diagram.

1. Return Value
2. Actual Parameters
3. Control Link
4. Access Link.
5. Saved Machine Status
   Local Data
   Temporaries

Q. Illustrate the structure of Activation Record with diagram.

- Structure of Activation
  1. Return Value
  2. Actual Parameter
  3. Control link
  4. Access link
  5. Saved Machine status
  6. Local Data
  7. Temporaries.

1. Return Value: It is used by calling procedure to return a value to calling procedure.

2. Actual Parameters: It is used by calling procedures to supply parameters to the called procedures.

3. Control Link: It points to activation Record of the caller.

4. Access link: It is used to refer non-local data held in other activation records.

5. Saved Machine Status: It holds the information about the status of the machine before the procedure is called.

6. Local Data: It holds the data that is local to the execution of the procedure.

7. Temporaries: It stores the value that arises in the evaluation of an expression.

Q. Describe the translation scheme for assignment statement.

eg:
Example by which we can describe the translation scheme for assignment statements:

$S \rightarrow id := E$

$E \rightarrow E_1 + E_2$

$E \rightarrow E_1 * E_2$

$E \rightarrow (E_1)$

$E \rightarrow id$

Production Rule	Semantic Actions
① $S \rightarrow id := E$	{ P = look_up (id.name) if P ≠ nil then Emit (P = E.place) else Error }
② $E \rightarrow E_1 + E_2$	{ E.place = newtemp(); Emit (E.place = E_1.place + E_2.place) }
③ $E \rightarrow E_1 \times E_2$	{ E.place = newtemp(); Emit (E.place = E_1.place × E_2.place) }
④ $E \rightarrow (E_1)$	{ E.place = E_1.place }
⑤ $E \rightarrow id$	{ P = look_up (id.name) if P ≠ nil then Emit (P = E.place) else Error }

Q. Describe various storage allocation strategies.

1. Static Storage Allocation:
- names are bound to storage locations.
- Supports Dynamic data structure that means memory is created only at compile time and deallocated after program completion.
- If memory is created at compile time then the memory will be created in static area and only once.

2. Stack Storage Allocation
- Storage is organized as a stack.
- An activation record is pushed into the stack when activation begins and it is popped when the activation ends.
- Works on the basis of last-in-first-out (LIFO) and this allocation supports the recursion process.

3. Heap Storage Allocation
- Most flexible allocation scheme.
- Allocation and deallocation of memory can be done at any time and at any place depending upon the user's requirement.
- Supports the recursion process.

Q. Describe various issues in the design of code generator.

1. Input to Code generator:
- The input to the code generator is the intermediate code generated by the front-end, along with the information in the symbol table that determines the run-time addresses of the data objects denoted by the names in the intermediate representation.
- Intermediate code may be represented mostly in quadriples, triples, indirected triples, Postfix notation, syntax trees, DAG etc.

- The code generation phase just proceeds on an assumption that the input is free from all syntactic and state semantic errors, the necessary type checking has taken place and the type-conversion operators have been inserted wherever necessary.

2. Target Program:
- The target program is the output of the code generator.
- The output may be absolute machine language, relocatable machine language, or assembly language.

3. Memory Management:
- Mapping the name in the source program to the addresses of data objects is done by the front end and code generator.
- A name in the three address statements refers to the symbol table entry for name.
- Then from the symbol table entry, a relative address can be determined for the name.

4. Instruction Selection:
- Selecting the best instruction will improve the efficiency of the program.
- It includes the instructions that should be complete and uniform.
- Instruction speeds and machine idioms also play a major role when efficiently considered.

5. Register Allocation:
- Use of registers make the computation faster in comparision to that of memory, so efficient utilization of registers is important.
- Use of registers is divided into two subproblems:
  i) Register assignment
  ii) Evaluation Order.

Q. Define Address Descriptors And Register Descriptors.

i) Address Descriptors:

- Values of the ~~names~~ identifies used in the program might be stored at different locations while in execution.
- Address Descriptor are used to keep track of the memory locations where the values of identifiers are stored.

ii) Register Descriptors:

- Used to inform the code generator about the availability of registers.
- It keeps track of values stored in each register.

Q. Explain input buffering technique used for lexical analysis phase of compilation.

i) Buffering Pairs:

- A specialized buffering techniques is used to reduce the amount of overhead which is required to process an input character in moving ~~characters~~ characters.
- Consist of two buffers, each consists of N - character size which are reloaded alternatively.
- Two pointers lexemeBegin and forward are maintained.
- Lexeme Begin points to the beginning of the current lexeme which is yet to be found.
- forward scan ahead until a match for a pattern is found.
- Once a lexeme is found, lexeme begin is set to the character immediately after the lexeme which is just found and forward is set to the character at its right ends.
- Current lexeme is the set of characters between two pointers.

Sentinels:

- Sentinels is used to make a check, each time when the forward pointer is moved, a check is done to ensure that one half of the buffer has not moved off.
- If it is done, then the other half must be reloaded.
- Therefore the ends of the buffer ~~have~~ halves require two tests for each advance of the forward pointer.
- Test 1: for end of buffer.
- Test 2: To determine what character is read.
- The usage of sentinel reduces the two tests to one by extending each buffer half to hold a sentinel character at the end.
- The sentinel is a special character that cannot be part of the source program.

Q) Differentiate between:

i) front-end and Back-end of ~~computer~~ compiler

ii) Analysis phase and Synthesis phase.

iii) Linkers and Loader.

i) front-end	Back-end
- It consist of those phases that depend primarily on the source program.	- It includes the code optimization phase and final code generation phase, along with the necessary error handling and symbol table operation.
- These normally include lexical and Syntactic analysis, Semantic analysis, and the generation of intermediate code.	- synthesizes the target program from the intermediate code.
- A certain amount of code optimization can be done by front end as well.	-

ii) Analysis
- known as frontend of compiler
- doesnot require specialized technique
- The analysis part breakup the source program into consistent pieces & creates an intermediation representation of source program.

Synthesis
- known as back-end of compiler.
- Synthesis requires the most specialised technique
- The synthesis part construct the desired target program from intermediate representation

iii) Loaders
- It is a part of OS that is responsible for loading program & libraries.
- It is one of the essential stages in the process of starting a program, as it places program into memory & prepares them for execution

Linkers
- Linker or link editor is a computer system program that takes one or more object file & combines them into a single executable file, library file or another object.

Q. Briefly describe the various data structures used for symbol table.

1. List:
- In this method, an array is used to store names and associated information
- It takes a minimum amount of space.
- A pointer "available" is maintained at end of all stored record records and new names are added in the order as they arrive.

2. Linked list:
- This implementation is used a linked list. A link field is added to each record.
- Searching of names is done in order pointed by the link of the link field.

- A pointer "First" is maintained to point to the first record of the symbol table.

3. Harsh Table:
- In harshing scheme, two tables are maintained - a harsh table and a symbol table.
- A harsh table is an array with an index range 0 to table size - 1. These entries are pointers pointing to the names of the symbol table.
- It is quick to search.

4. Binary Search Tree:
- Another approach to implimenting a symbol table is to use a binary search tree i.e we add two link fields ie left and right child.
- All names are created as child of the root node that always follows the property of the binary search tree.

6. Write an algorithm to construct basic blocks form a sequence of 3-address statement.

→ Rule 1: Determining the leader.
- first statement of a block is always treated as a leader.
- The target of conditional or unconditional jump is a leader
- The statement following conditional or unconditional jump is a leader.

→ Rule 2: Determining the basic block;
- Start at leader & ends before the next leader
- Beginning at the leader statements and ends before the next leader.

Q. Explain the following code optimization techniques.

i) Common sub-expression elimination:

- It is an expression appearing repeatedly in the program which is ~~computed~~ computed previously.
- Then if the operands of this sub expression do not get changed at all then result of such sub expression is used instead of recomputing it each time.

ii) Constant folding:

- Expressions with constant operands can be evaluated at compile time, thus improving run-time performance and reducing code size by avoiding evaluation at compile time.

Q. Explain in detail Peephole optimization.

- This technique work locally on source code to transform it into an optimized code.
- The peephole optimization is a short sequence of target instructure into otherless expensive one such as turning multiplication of x by 2 into an addition of x with itself.

Characteristics

i) Reductant Instruction elimination

ii) Unreachable code:

- It is a part of program code that is never accessed because of program construct.
- Programmer may have accidently written a piece of code that can never be reached.

iii) Flow of control Optimization: These are instances in a code where the program control jump back & forth without performing an significant task, these jump can be removed.

iv) Algebric Simplication

these are occasions where algebric expressions can be made simple.