Compiler Design
Chapter 1: Compiler Intro

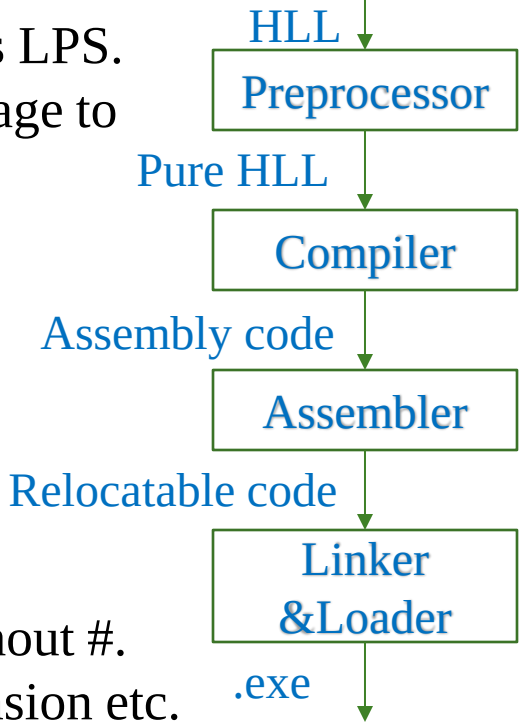
GATE CS Lectures
by Monalisa

- **Section 7: Compiler Design(≅5 mark)**
- Lexical analysis, parsing, syntax-directed translation. Runtime environments. Intermediate code generation . Local optimization, Data flow analyses: constant propagation, liveness analysis, common subexpression elimination.

- Chapter 1: Introduction to Compiler [Language processing System ,Compiler ,Phases of Compiler , Lexical Analysis]
- Chapter 2: Parsing
- Chapter 3: SDT,Code optimization &Runtime environments

Language Processing System

- The Software that compile & generate a .exe file is called as LPS.
- The software that convert source code from one form language to another form language is called as LPS.
- Basic function of LPS
- 1.Preprocessor
- 2.Compiler
- 3.Assembler
- 4.Linker & Loader



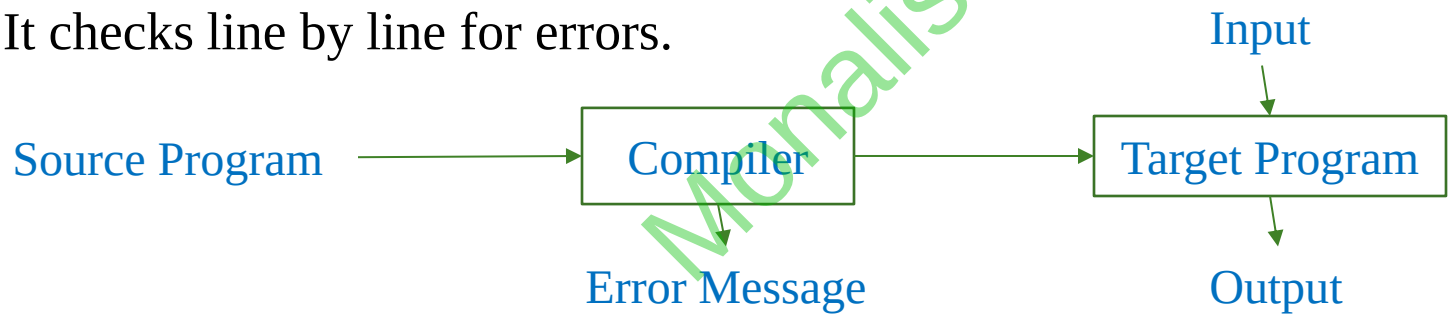
- **1.Preprocessor:**
- Preprocessor is a tool that produces input for compilers without #.
- It deals with macro-processing,file inclusion,language extension etc.
- Preprocessor also called as macro evaluator.
- It is optional ,preprocessing is not required for language which doesn't supports #include & macro . Ex :Pascal.

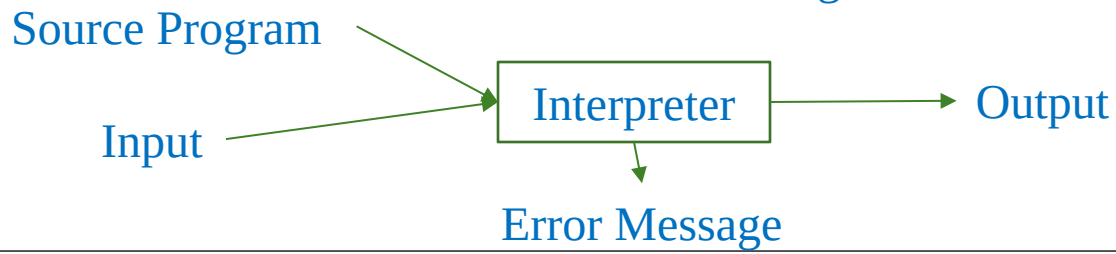
- Macro processing:
- A macro is a rule or pattern that specifies how a certain input sequence should be mapped to an output sequence according to a defined procedure.
- The mapping process that instantiates a macro into a specific output sequence is known as macro expansion . Ex-#define.
- File Inclusion:
- 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.
- Language extension :
- These processors attempt to add capabilities to the language by built-in macros.
- For example, the language Equal is a database query language embedded in C.
- **2.Compiler:**
- The SW system that convert source code into Assembly language instruction .It is optional not required for language like HTML, DHTML,JavaScript etc.

- **3.Assembler:**
- Assembler creates object code by translating assembly instruction into machine code.
- Object code is hexadecimal operated by Executable File Format.
- Object code is relocatable.
- **4.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 :
- Allocation : Getting memory partition from OS to store object code.
- Relocation : Mapping of relocatable object code in the physical location.
- Linker : Combine all the external (.dll) file to the object code & generate .exe file.
- System wide start up file , system library file , system I/O file etc will be added to object code to generate .exe file.
- Loader :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.

Compiler vs Interpreter

- **Compiler**: Translation of a program written in a source language into a semantically equivalent program written in a target language .
- Generate new program that runs without compiler.
- It displays the errors after the whole program is executed .
- **Interpreter**: a program that reads an *executable* program and produces the results of running that program
- Interpreters run programs “as is” ,Little or no preprocessing
- It checks line by line for errors.



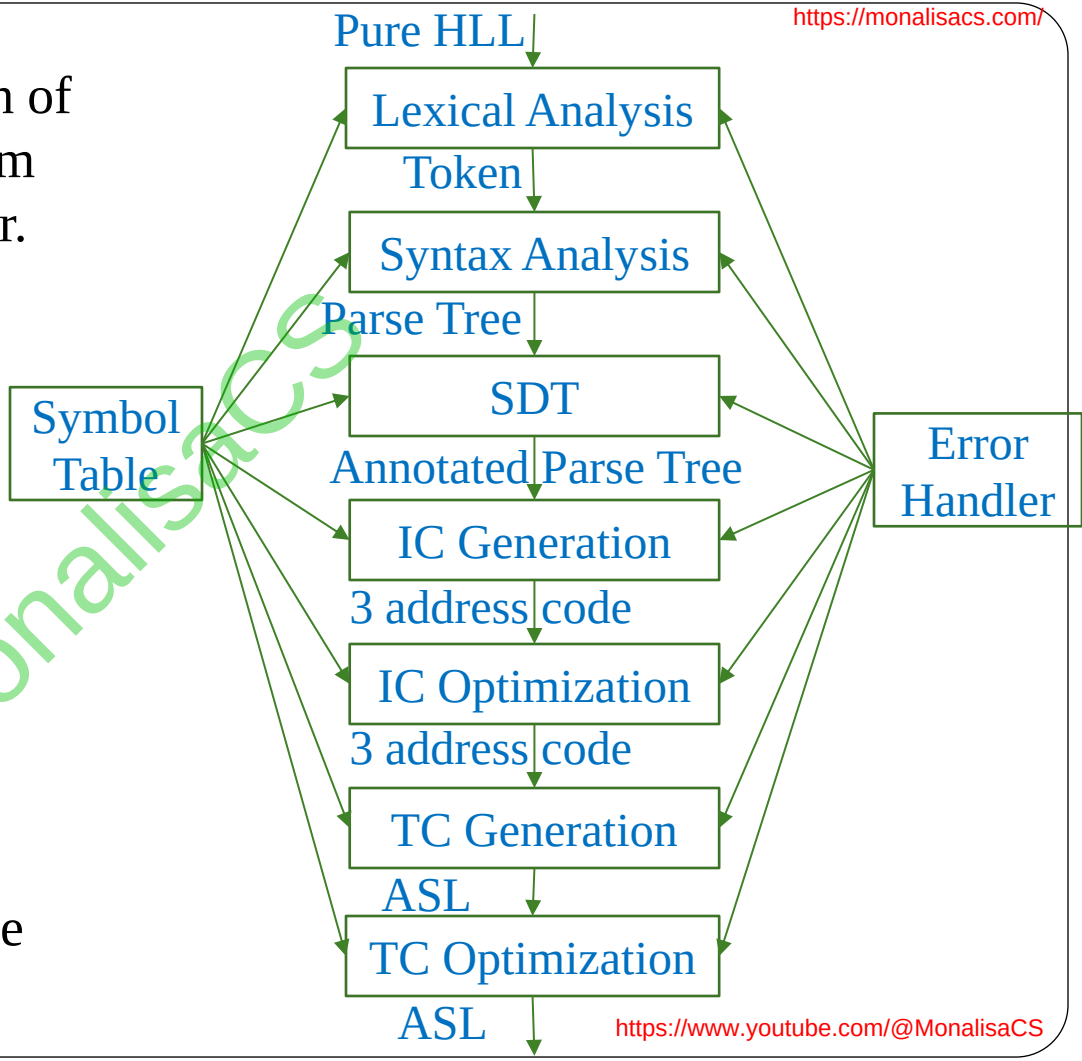


Compiler	Interpreter
➢ Process of translation	➢ Process of execution
➢ Scan entire text at a time	➢ Scan the text line by line
➢ Request more memory but less time	➢ Request less memory but more time
➢ Performance is high	➢ Performance is low
➢ Reusability of structure & code possible	➢ No concept of reusability
➢ Code optimization is possible	➢ Code optimization is not possible
➢ Debugging is difficult	➢ Debugging is easy
➢ Fortran is 1st language of compiler	➢ Basic is 1st language of interpreter
➢ Ex:C,C++,Fortran ,Pascal ,Cobol ,Smalltalk	➢ Ex:SQL ,PLSQL ,Basic ,Lisp , Prolog, Matlab , Perl

➢ Ex of both :Java ,Python , scala

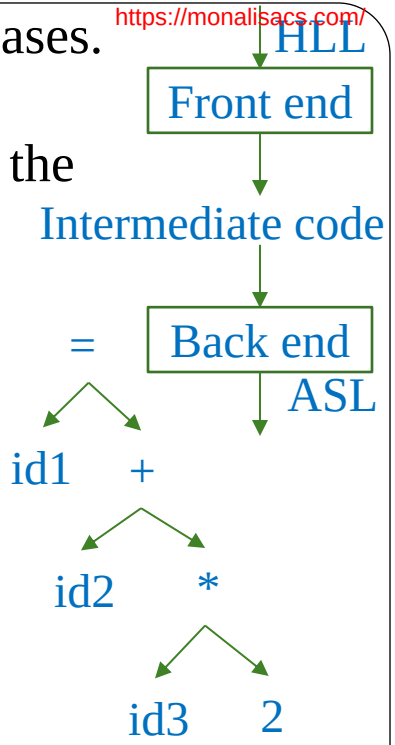
Phases of Compiler

- A Compiler operates in phases, each of which transforms the source program from one representation into another.
- 1.Lexical Analysis
- 2.Syntax Analysis/Parsing
- 3.Syntax directed Translation(SDT) /Semantic Analysis
- 4.Intermediate code generation
- 5.Intermediate code optimization / Machine Independent Code optimization
- 6.Target code Generation/Machine Dependent code generation
- 7.Target code optimization /Machine dependent code optimization

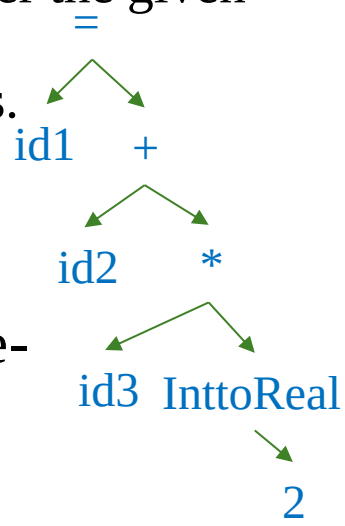


- **Pass:**
- Number of time source code is scan during the process of compilation .
- The Compiler can be single pass or multi pass.
- Single pass: usually requires everything to be defined before being used in source program.
- Single pass require more memory but less time.
- Multi pass: compiler may 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.
- Multi pass require less memory more time.
- In general the compiler is 2 pass
- The process of translation is divided into two part :1. Front end 2.Back end
- **Front end :** analysis (machine independent)
- These include lexical and syntactic analysis, the creation of the symbol table ,semantic analysis and the generation of intermediate code.
- It convert source code into 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.
- It convert 3 address code into assembly language.
- **Lexical Analysis**:
- It scan the source code & divide into tokens where token is basic programming unit.Use Regular Expression for token identification.
- Ex: a=b+c*2 ⇒<id1><=><id2><+><id3><*><2>
- **Syntax Analysis:**
- It gets the token stream as input and generates syntax tree/parse tree as the output.
- Syntax tree :It is a tree in which interior nodes are operators and exterior nodes are operands . Same as derivation tree of CFG.
- It verify sentence is according to grammar or not.
- Ex:id1=id2+id3*2

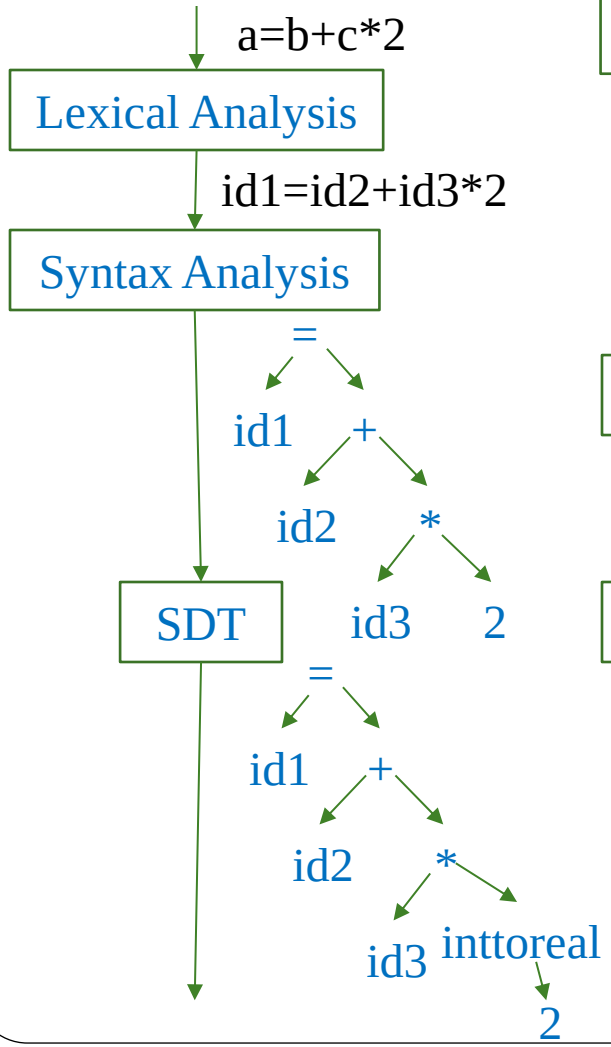


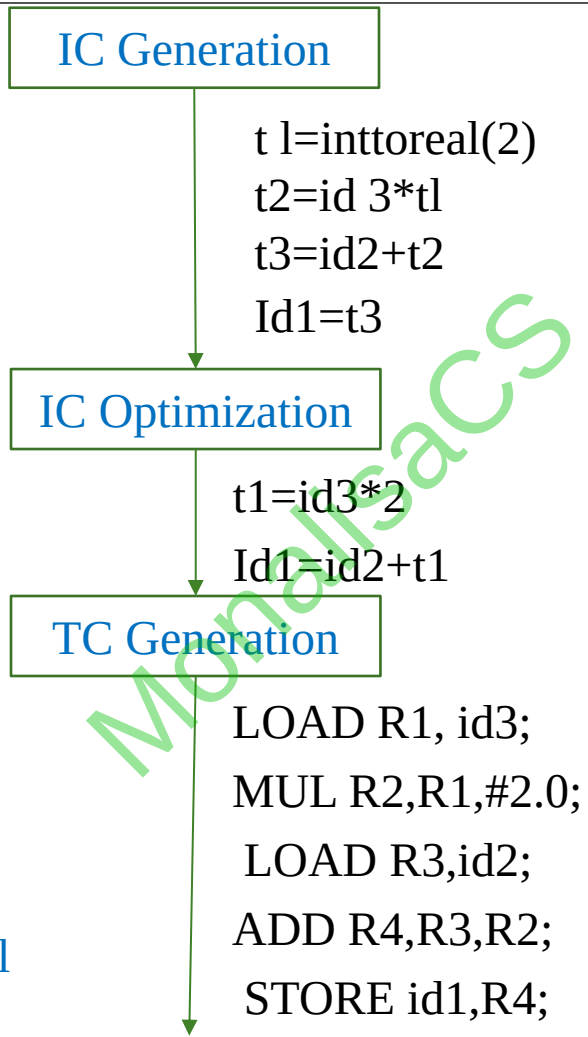
- **Semantic analysis:**
- It gets input from the syntax analysis as parse tree and checks whether the given syntax is correct or not.
- It performs type conversion of all the data types into same data types.
- It verify semantic of sentence .The sentence is meaningful or not.
- Ex:id1=id2+id3*2

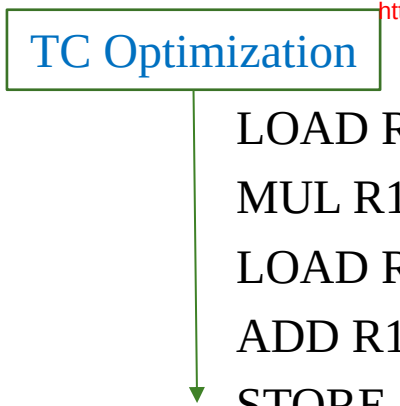


- **Intermediate code generation:**
- It converts input into output as intermediate code such as three-address code.
- The three-address code consists of a sequence of instructions , each of which has atmost three operands
- each three-address assignment instruction has atmost one operator on the right side .
- t l=inttoreal(2)
- t2=id 3*tl
- t3=id2+t2
- Id1=t3
- “t” is used for temporary value.

- **Intermediate code optimization:**
- It gets the intermediate code as input and produces optimized intermediate code as output without affecting outcome of source code.
- Ex:t1=id3*2
- Id1=id2+t1
- **Target code generation:**
- It convert 3 address code into target code/assembly language.
- Ex: LOAD R1, id3; MUL R2,R1,#2.0;
- LOAD R3,id2; ADD R4,R3,R2;
- STORE id1,R4;
- **Target code optimization:**
- Reducing number of register and instruction without affecting outcome
- Ex: LOAD R1, id3; MUL R1,R1,#2.0;
- LOAD R2,id2; ADD R1,R1,R2;
- STORE id1,R1;







Symbol Table	
a	Id1
b	Id2
c	id3

Symbol Table :

- It is the abstract data structure use by compiler to store all the information about identifiers used in the program.
- Every phases of compiler interact with symbol table.
- Whenever an identifier is detected in any of the phases, it is stored in the symbol table.
- During first 2 phases information store in symbol table & in remaining phases the information of the symbol table will be used.
- It allows to find the record for each identifier quickly and to store or retrieve data from that record.
- Information of identifier store into symbol table are name ,value ,type ,size ,offset or address ,scope ,lifetime ,token ,other information.
- Function or operation of symbol table:1.insert,2.Modify,3.Lookup,4.Delate
- Implementation of symbol table: Hash table is the suitable DS for symbol table because of fastness & lookup operation .

L no	Name	value	Type	Size	offset	Scope	lifetime	token	Other info
1	a	10	Int	2	X100	Global	Fun	id1	…….
2	b	20	Int	2	X110	Local	Fun	id2	…….
3	c	30	int	2	X120	Local	program	id3	…….

Error Handling:

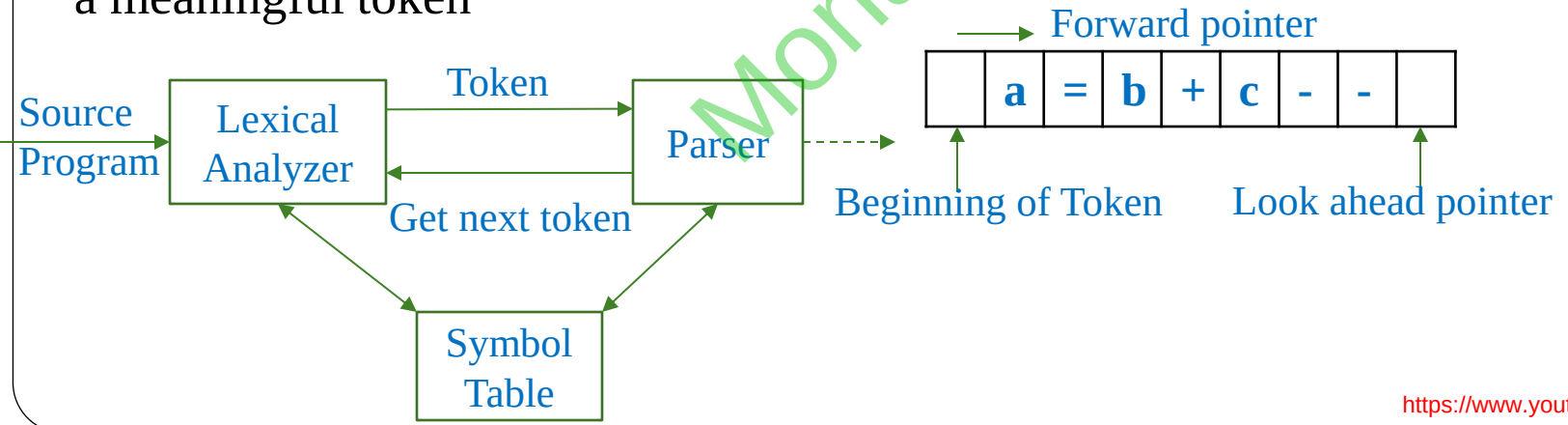
- Each phase can encounter errors. After detecting an error, a phase must handle the error so that compilation can proceed.
- Lexical errors occur in separation of tokens ,declaration of variable , exceeding length ,unmatched string.
- Ex:int x y=10;
- Syntax errors occur during construction of syntax tree and grammar of language.
- Ex: int x=20(no semicolon) ,x=20 int;
- Semantic errors occur due to meaning of sentence and type conversion.
- It also check any variable must be declared before its use.
- Ex: Int x=‘toc’;
- Code optimization errors occur when the result is affected by the optimization.
- Code generation error shows when code is missing etc.
- The error that can be handled during process of compilation called as exception.
- Programmer is responsible for handling exception.
- The error that can occur during process of execution is called as fatal error & admin is responsible for this.

Lexical Analysis

- Lexical analysis is the process of converting a sequence of characters into tokens.
- Lexical analysis is also called a lexical analyzer , scanner, Lexer ,Tokenizer .
- We can also produce a lexical analyzer automatically by specifying the patterns to a lexical-analyzer generator.
- lexical-analyzer generator called Lex (or Flex) .
- **Lexeme:**
- Collection or group of characters forming tokens is called Lexeme.
- It is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token .
- **Token:**
- The process of forming tokens from stream of characters is called **tokenization.**
- A token is a string of characters, categorized according to the rules.
- The sequence of char having logical meaning is called as token.
- Token is basic programming unit.

- Rules for token:
- 1.One token for each keyword , identifiers, constants ,such as numbers and literal.
- 2.One tokens for the operators ,either individually or in classes.
- 3.One tokens for each punctuation symbol ,such as left and right parentheses ,comma ,and semicolon .
- 4.One token for statement written “ ” .
- **Pattern:**
- A pattern is a description of the form that the lexemes of a token may take.
- Pattern is the rule to describe RE & to recognize string.
- 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.
- Ex: Printf ("Total=%d\n", score) ;
- <Printf> <(> <“Total=%d\n”> <,> <id1> <)><;>

- **The Role Of The Lexical Analyzer** :
- 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.
- Store attribute information in symbol table then send to parser.
- For LA stream of char is input & sequence of token is output.
- LA generate token depending on next symbol.
- Buffering technique is use to read the group of char at a time instead of char by char.
- As characters are read from left to right, each character is stored in the buffer to form a meaningful token



- **Secondary Function Of Lexical Analyzer:**
- 1.Stripping out comments and whitespace (blank ,newline , tab) .
- 2.Keep track of the number of new line characters seen ,so it can associate a line number with each error message.
- 3.Creation of symbol table & store attribute information in symbol table
- Token representation require less memory than ASCII representation.
- Token format is convenient structure to verify the structure of program using CFG.
- As CFG contain only terminal & nonterminal & all programming language can be defined by CFG.
- **Lexical errors** occur in separation of tokens ,declaration of variable , exceeding length ,unmatched string ,illegal char.
- **Error Recovery Strategies In Lexical Analysis:**
- 1. Panic mode recovery: Deletion of successive characters from the token until error is resolved
- 2. Deleting an extraneous character.
- 3. Inserting a missing character.
- 4. Replacing an incorrect character by a correct character.
- 5. Transforming two adjacent characters.

- GATE 2000-Q18, ISRO 2015-Q25:The number of tokens in the following C statement is **printf** ("i = %d, &i = %x", i , &i) ;
- (A) 3 (B) 26 (C) 10 (D) 21
- Ans :10
- ISRO CS 2017 – May: The output of a lexical analyzer is
- (A) A parse Tree
- (B) Intermediate Code
- (C)Machine Code
- (D)A stream of Token
- Ans :(D)A stream of Token