HindPhotostat

Hind Photostat & Book Store

Best Quality Classroom Topper Hand Written Notes to Crack GATE, IES, PSU's & Other Government Competitive/ Entrance Exams

MADE EASY
COMPUTER SCIENCE
Topper Handwritten Notes
COMPILER DESIGN
BY-PRASAD SIR

- Theory
- Explanation
- Derivation
- Example
- Shortcuts
- Previous Years Question With Solution

Visit us:-**www.hindphotostat.com**

Courier Facility All Over India
(DTDC & INDIA POST)
Mob-9311989030

HindPhotostat

GET IT ON Google Play

ALL NOTES BOOKS AVAILABLEALL STUDY MATERIALAVAILABLECOURIERS SERVICE AVAILABLE

MADE EASY , IES MASTER, ACE ACADEMY, KREATRYX

ESE , GATE,PSU BEST QUALITYTOPPER HAND WRITTEN NOTESMINIMUM PRICE AVAILABLE @ OUR WEBSITE

1. ELECTRONICS ENGINEERING
2. ELECTRICAL ENGINEERING
3. MECHANICAL ENGINEERING
4. CIVIL ENGINEERING
5. INSTRUMENTION ENGINEERING
6. COMPUTER SCIENCE

IES ,GATE , PSU TEST SERIES AVAILABLE @ OUR WEBSITE

- IES –PRELIMS & MAINS
- GATE

➢ NOTE;- ALL ENGINEERING BRANCHS

➢ ALL PSUs PREVIOUS YEAR QUESTION PAPER @ OUR WEBSITE

PUBLICATIONS BOOKS -

MADE EASY , IES MASTER ,ACE ACADEMY ,KREATRYX ,GATE ACADEMY, ARIHANT ,GK

RAKESH YADAV, KD CAMPUS ,FOUNDATION , MC –GRAW HILL (TMH) ,PEARSON...OTHERS

HEAVY DISCOUNTS BOOKSAVAILABLE @ OUR WEBSITE

HIND PHOTOSTAT AND BOOK CENTER F230, LadoSarai New Delhi-110030 Phone: 9311 989 030 9560 163 471	Shop No: 46 100 Futa M.G. Rd Near Made Easy Ghitorni, New Delhi-30 Phone:	F518 Near Kali MaaMandir LadoSarai New Delhi-110030 Phone:	Shop No.7/8 Saidulajab Market Neb Sarai More, Saket, New Delhi-30

Website: www.hindPhotostat.com

Contact Us: 9311 989 030

Compiler Design.

- Basis of a compiler.
- Lexical Analysis
- Syntax Analysis
- Syntax Directed Translation.
- Intermediate code generation.
- code Organisation.
- Run time Environment.

* Basis of a Compiler.

- Language Translator
- Language Processing System.
- Types of Compiler
- Phases of Compiler.
- Single Pass & Multipass Compiler.

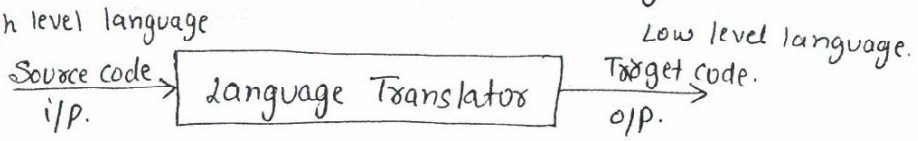
* Lexical Analysis:

- Divides into tokens
- Recognisation of tokens
- Create Symbol Table
- Error Recovery Methods.
- Interacts with Syntax Analyzer.
- Lex Tool.

BASICS OF COMPILER.

LANGUAGE TRANSLATOR.

Language Translator takes one language as input and produces other languages as output. Generally the i/p is Source code and output is Target code.



Types of language Translator:-

- Compiler
- Interpreter.
- Assembler.

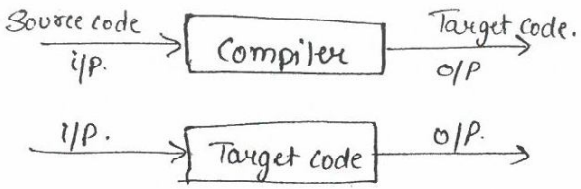
Compiler.

Compiler takes the Source code as input and Produces the target code as output. if that target code is an executable code then it will called by the user to process the inputs for producing the output.

- Compiler executes the entires code at a time, if any error occurs at any line all of them will be given at a time.
- Error diagonsis is not easy in compiler than compare to interpretor but output generation by compiler is faster than interpreter.
- Compiler is an offline process.
- Compiler requires more memory than the interpreter.

- The end user cannot make the modifications easily to the compiled program.

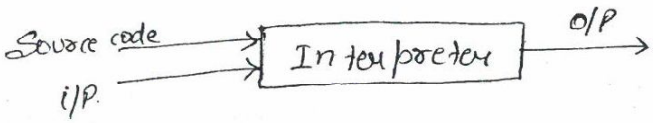
Eg:- C, C++, Pascal etc.



Interpreter.

- Interpreter takes the source code as input and produces the direct output. It will not produce any intermediate language as in the case of compiler.
- Interpreter executes the code line by line, if any error occurs at any line then immediately it will produces that error to the user.
- Until we resolve that ~~error~~ error the interpreter will not move to the next line.
- Interpreter executes every statements of the program and simultaneously it process the inputs also, thus interpreter is online process. After completing the execute of the last line of the program, immediately it will produce the output.
- If we change the input then interpreter again execute the total program from top to bottom. Therefore output generation by interpreter is slower than the compiler.

- The interpreted program can be modified easily by the end user.
- Interpreter takes less memory than the Compiler.
- Eg:- LISP, Python, PERL, RUBY.

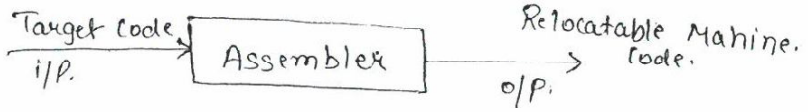


* Assembler:-

- Assembler takes the assembly language as input and produces relocatable machine code as o/p which has to be loaded in main memory which ready for execution.
- Assembler is a compiler of assembly language. Assembly languages uses opcode for its instructions. An opcode basically gives the information about the particular instruction.
- The Symbolic Representation of OPcode is called as Mnemonics.

Eg:-

```
ADD  A,B.
 ↑   ↑ ↑
opcode  operands.
```



Eg:- GAS, GNU.

One Pass Assembler:-

In this the whole conversion of assembly language code into machine code will be done in one step.

Two Pass Assembler / Multipass Assembler.

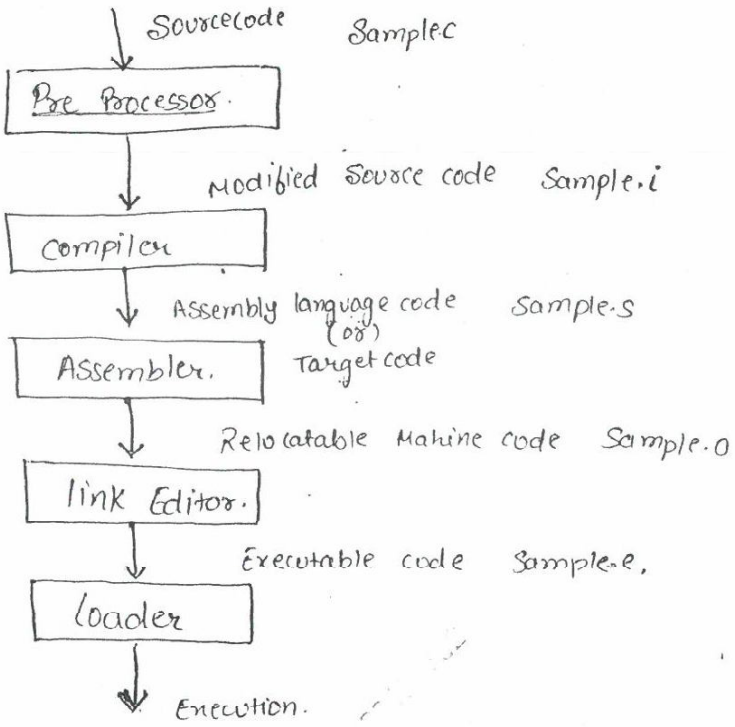
In this assembler 1st process the assembly language and creates the values in Symbol table & opcode table and then in the 2nd step it generates the machine code using these values.

Symbol tables.

It stores the information about the variable & their attributes

OPcode Tables:- It stores the mnemonics and their corresponding values.

LANGUAGE PROCESSING SYSTEM



Pre-Processor

- The Pre-processor takes the source code as input and produces modified source code as output.
- If the program is too big, it may be divided into small programs (i.e modules) and will be stored in different files.
- The Preprocessor takes care of all these modules.
- The command used in Pre-processor are called as Pre-Processor directory and they begin with the Symbol hash (#).

Lexical Analysis :-

- Functions of Lexical Analysis
- Lexical Errors.
- Finding of the tokens
- LEX Tool.

- A Lexical analyzer Scans every character of the Source code and the following will be done by it
  - Divides into tokens
  - Ignores comments
  - Ignores white Space characters like blank space, tabspace & new line characters.
  - Counts the no. of lines

Syntax Analysis

- Ambiguity.
- Left- Recursion
- Left Factoring
- Top Down parsing
    - Back Tracking
    - Recursive Descent Parsing
    - LLC(1) Parsing
    - FIRST & FOLLOW
- Bottom up parsing
    - operation Precedence Parsing.
    - SLR(1), LR(0), LALR(1), CLR(1)
- YACC Tool.

Source code

- The tokens produces by lexical Analyzer will be given to the Syntax analyzer to form a parse tree.
- In Syntax analysis the token will be grouped together as a parse tree by using some rules will be supplied by seperated by CFG. If tokens are not well formed then the Syntax Analyzer produces a Syntax error.

If these tokens are Syntically correct then they will be grouped together