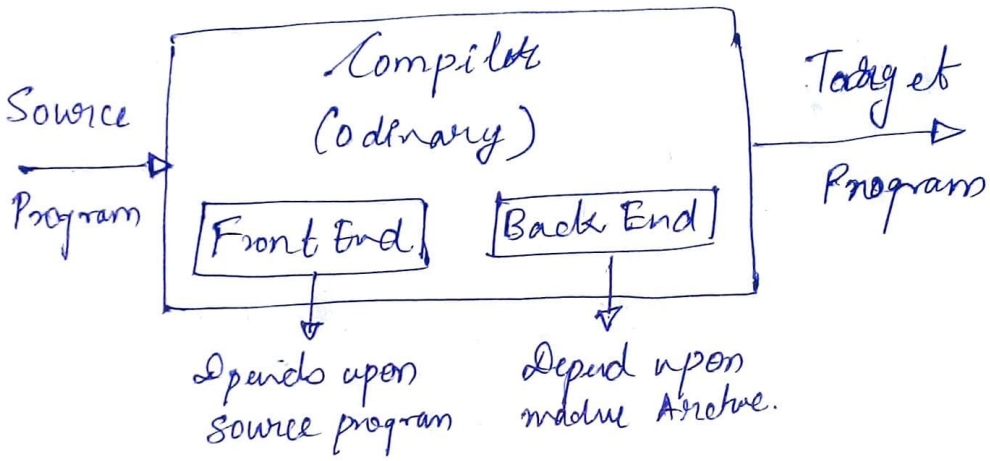
29-04-20

Revision

Day - 1

Q.1 Differentiate between front end and back end of the compiler.

Ans.



- There are mainly 6 phases of a compiler which are :

1. Lexical Analyser
2. Syntax Analyser
3. Semantic Analyser
4. Intermediate Code generator
5. Code optimizer
6. Target Code generator

→ Front End

→ Back End

Front End of a Compiler

- It consists those phase which work upon the source language.
- The front end depend upon the high level langage eg:- C, C++, Python – in which the soure program is written.
- This part is mostly systems independend.

Back end of a Compiler

- It consists of those phase which work to produce target language
- The back end depend upon the low-level language eg:- Binary/ Machine Langage, Assembly langage.
- Back end is usually depends upon the system arctue to produce efficient code.

Q2. Define Lexem, Token and Pattern.

Ans

~~Lexems~~ Token : Token is a sequence of characters that can be treated as a single local entity
eg:- Identifier, keywords, operator etc.

Pattens : Pattens are rules that describe the set of strings associated with the tokens.

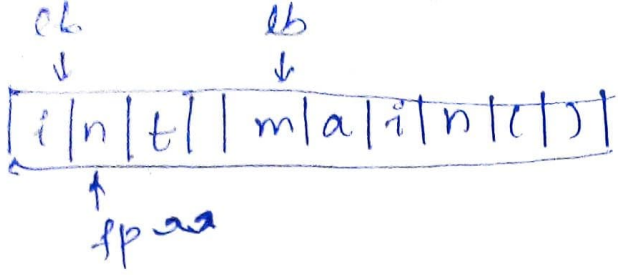
Lexeme: A lexem is a sequence of charaks in the source program that is matched by the pattern for a token.

eg:- for, while, ~~3.14~~ 3.14 "hello world"

Q3. Describe the Input Buffering Schem in Lexical Analyser.

Ans Input Buffering Scheme

- It is the ~~schem~~ buffering schem employed to ~~ef~~ efficiently manage memory.
- The source program is written ~~in~~ and saved in the hard disk
- To execute it it must be brought to the main memory.
- Lex Analyser scans the source code with two pointers say "lp" & "fp" (lexem begin & forward pointer) & bring them into a buffer.
- The buffer is used because ~~direct~~ a system call to just bring one character to the main memory is ~~tedious~~ tedious & expensive.
- With buffer chunks of source code can be brought in to main memory.



One buffer scheme

- This is used when the source size is small in blocks.
- If buffer overflows it will corrupt the earlier data.

Two buffer scheme

- Two buffer are used alternatively
- If one buffer reaches eof the other buffer can be used.

Q4. Uses of Symbol Table

Ans - The symbol table is a data structure containing a record for each variable with fields for the attributes of the name.

- These attribute may provide information about the storage allocated for a name - its type, scope, procedure name, arguments, return type etc.
- The data structure must be efficient to find ~~out~~, retrive and store data efficiently.

- This symbol table is used with each of the stages.

Q5. Explain the any four compiler writing tools.

- These are the four compiler writing tools.

1. Parser Generator:

- helps to write a software which can parse other languages.

Input : rules for a language

output : syntax.

2. Scanner Generator

- The input to a scanner generator is regular expression.
- while the output is a lexical Analyser.

3. Syntax Directed Translation Engine

- Helps in generating the regex engine with :

input : parse tree

Output : Intermediate code generator.

4. Data flow Analysis Engine

~ This tools is used for advanced code optimization which will help to realize the possibilities of code refactoring when and where required.

Q6. Discuss the importance of bootstrapping to develope a compiler.

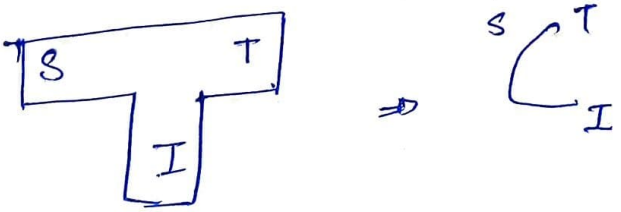
Ans

~ Bootstrapping is the process in which a simple language is used to translate more complicated program.

~ This intern may help to handle an even more complicated program.

~ It helps in producing a self-hosting compiler compiler – the one that can compile its own source code.

~ The compiler is defined by three languages (i) Source Language (ii) Target Language (iii) Implementation Language



~ Thus bootstrap significally reduces the complexity of writing a ~~tri~~ tual. high level compiler,

~ The above is true given that we already possess the lower level cascaded compilers.

Q7. Define Cross compiler & compiler compiler.

Ans.

Cross compiler

~ These are those types of compiler that that can generate code for multiple platformes.

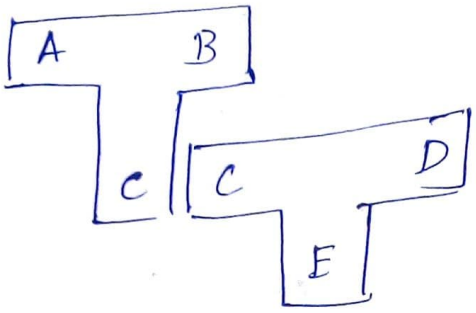
~ For this the front end is same but the backend is modified to enable code genration for various platoforms (being platfome agnostic)

eg:- GNU cross compiler
Flutter - it uses a buildengine to develop binary for linux, widows android and iOS.

Compiler Compiler

- A bootstrapped compiler compiles the compiler.

eg:-



- Here E is the comiler compiler
- The ultimate source large is A & the ultimate target large is D.
- C compiles A to gmake B
- E compiles C to gerate D

- This is the process of bootstrapping and it helps in compiling the compiler.

Q8. Explain the differt Phases in the design of a compiler.

A8. There are mainly 6 phases in the design of a compiler:

1.) Lexical Analyser

- This phase analyses the sourcode and converts it into tokens.
- At the sanae time it it uses the symbol table to ~~reduce~~ store these token with their respective ids for later user.

input : sowrce code output : tokens, lexems

2.) Syntax Analyser

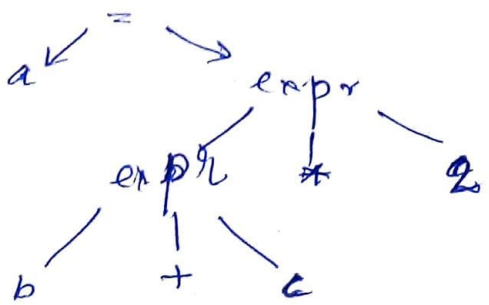
- This phase cheks for the syntax error in the given sowrce code.
- It takes input from the lexical analyser and genrates a syntax tree

input : tokens/lexems output : syntax tree

eg:- a = (b + c) * 2

Tokens: a, =, (, b, + (,), *, 2

Sytax tree

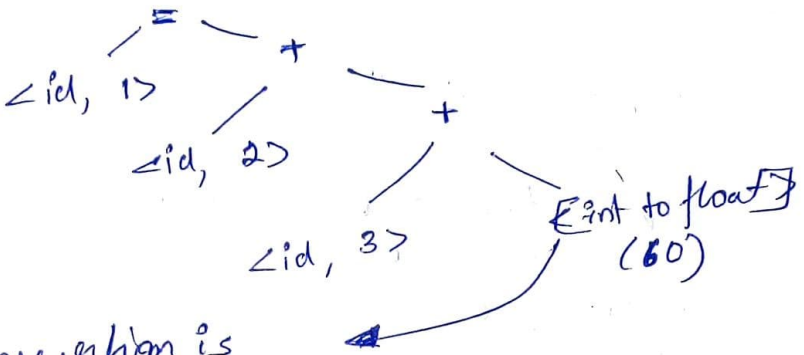


3) Semantic Analyser

~ This phase checks for semantic errors in the source code

input: syntax tree output: corrected code.

eg:-



This conversion is done to maintain the consistency of the program.

4) Intermediate code generation

~ This phase utilises the temporary variable to generate code.

~ It usually generates, three-address intermediate code

eg:- $t_1 = int\_to\_float(2)$
$t_2 = id3 * t1$
$t_3 = id2 + t2$
$id_1 = t3.$

5) Code optimization phase

~ This phase check for the ~~program~~ ~~in code~~ room for optimization in the code.

- This phase is necessary to avoid utilizing excess of resources

eg:- t1 = id3 * 60.0
id1 = id2 + t1.

6.) Code Generation

- This is the final phase which generates the the target code in the desired language.

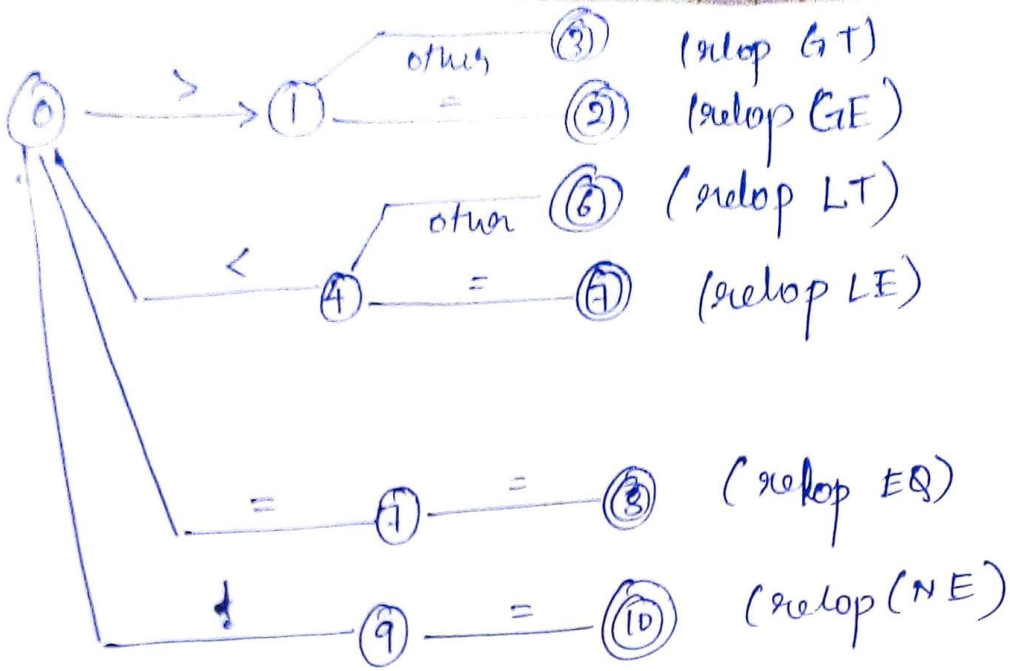
eg:-
```
LDF   R2, id3
MULF  R2, #60.0
LDF   R1, id2
ADDF  R1, R2
STF   id1, R1
```

Q.9. Construct a lexical analyser for the token of relational operators.

Sol. The relation operators are:

relop (>, >= , <, <= , ==, !=) GT, GE, LT, LE, EQ, NE

∴ The lexical analyser will relop portions check in this manner.



Q6. Construct a lexical analyser for the token conditional statements (if, else, for)

Sol The analyser will perform checks and move to state in this manner.