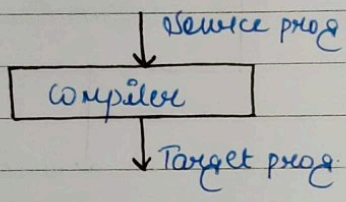
10th Jan'17
Lecture-1

Introduction

Language Processors :

The compiler is a program that can read a program in one language the source language and translate it into an equivalent program in another language the target language.



An important role of the compiler is to report any errors in the source program that it detects during the translation process.

If the target program is an executable machine language program it can then be called by the user to process inputs & produce outputs.

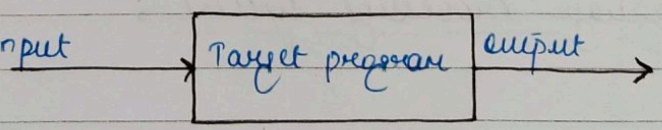
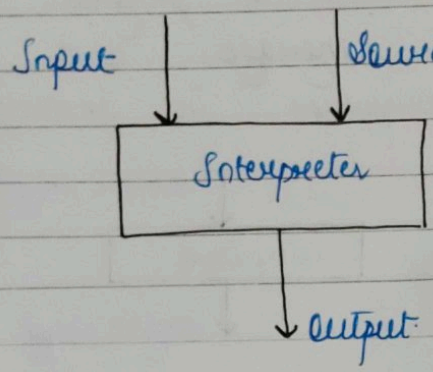


Fig : Running the target prog.

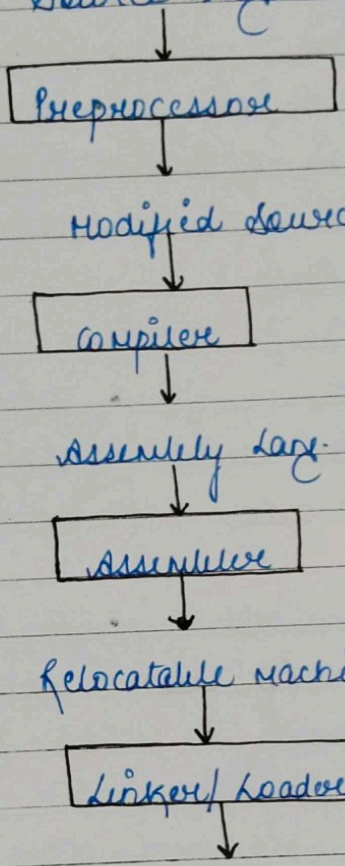
An interpreter is another kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operation specified in the source program on inputs supplied by the user.



* Note :

The machine language target program produced by the compiler is much faster than interpreter in mapping i/ps & O/ps. An interpreter gives better error diagnostic than a compiler because it executes the source program statement by statement.

Language Processing System :



11th Jan '17
Lecture-2

Difference b/w Compiler & Interpreter :

Compiler	Interpreter
• Compiler works on the complete program at once. It takes the entire program as input.	• It works line by line which takes one statement at a time as input.
• Compiler generates intermediate code called object/ machine code.	• Interpreter does not generate intermediate object or machine code.
• It executes conditional control statements (if, else) and logic constructs faster than interpreter.	• It executes conditional control statements at a much slower speed.
• Compiled programs take more memory because the entire object code has to reside in memory.	• Interpreter does not generate intermediate object code as a result ~~interpreter~~ interpreted programs are more memory efficient.
• Compiled once & run any time. Compiled programs do not need to be compiled everytime.	• Interpreted programs are interpreted line by line everytime they are run.

Compiler	Interpreter
• Errors are reported after the entire program is checked for syntactical & other errors.	• Error is reported as soon as the first error is encountered, rest of the error will not be checked until the existing error is removed.
• A compiled language is more difficult to debugg.	• Debugging is easy because interpreter stops & report error as it encounters them.
• They does not allow a program to run until it is completely error-free	• It runs the program from the first line & stops execution only if it encounters an error.
• Examples of programming languages that use compiler are C, C++ & COBOL	• Examples are VB, Python, MATLAB, PERL, LISP etc

Structure (Phases) of a compiler :

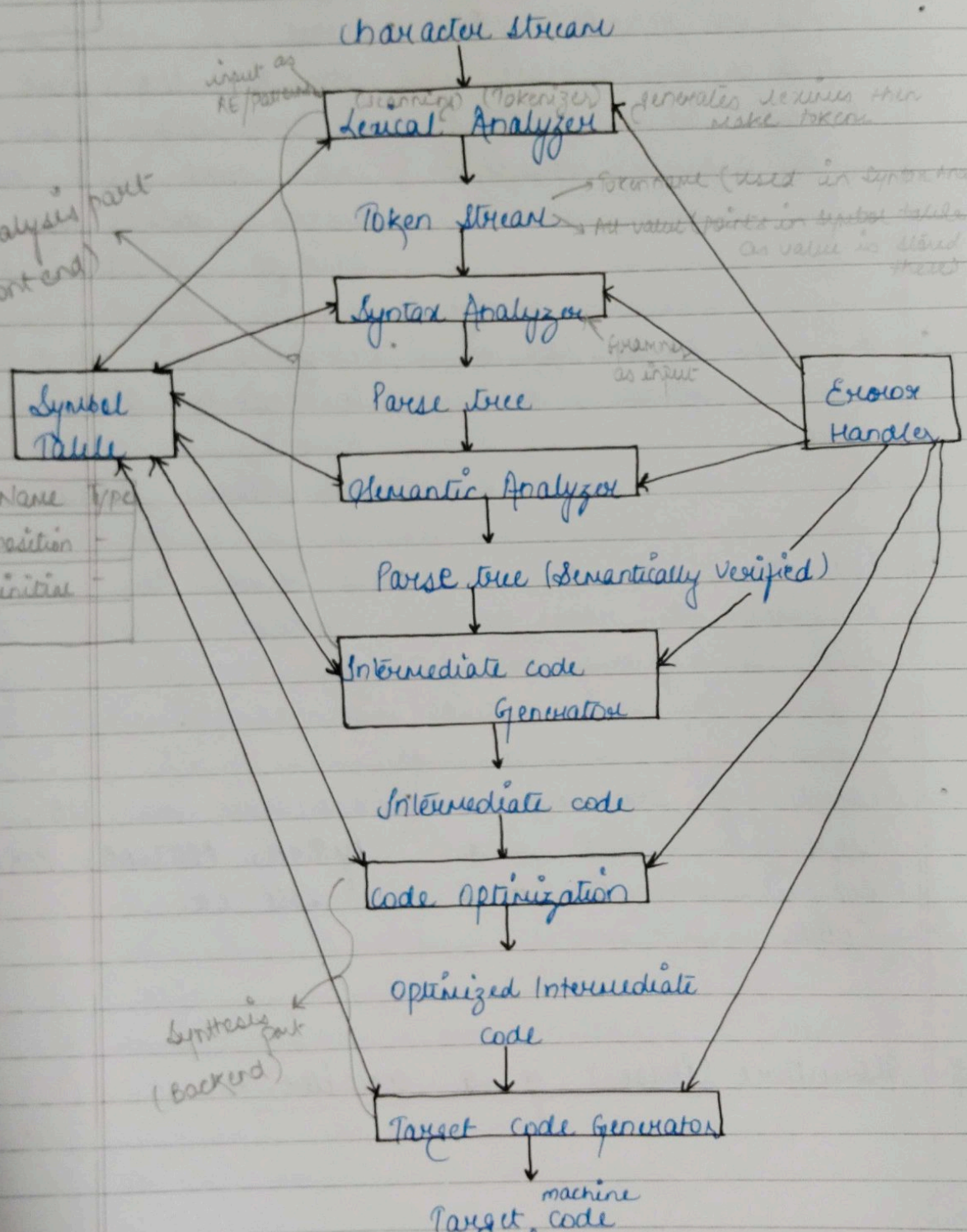
Example : Position = initial + rate * 60

<token name, attr value>

<id, 1> <=> <id, 2> <+> <id, 3> <*> <60>

Symbol Table

S NO	Name	Type
1	Position	-
2	initial	-



S NO	Name	Type
1	position	-
2	initial	-

12th Jan'17
Lecture-3

* Phase I : Lexical Analyzer / Scanning / Tokenizer :

LA reads the stream of characters making up the source program & groups the characters into meaningful sequences called "lexemes". For each lexeme, the LA produces as output a token of the form <token_name, attribute_value>.

- token_name is used in phase 2 Syntax analyzer.
- attribute_value contains a value that will be stored in symbol table.

Example : Position = initial + rate * 60

<id, 1> <=> <id, 2> <+> <id, 3> <*> <60>

* Phase II : Syntax Analyzer (Parsing) :

The parser uses the first component of the tokens produced by LA to create a tree like intermediate representation that depicts the grammatical structure of the token stream.

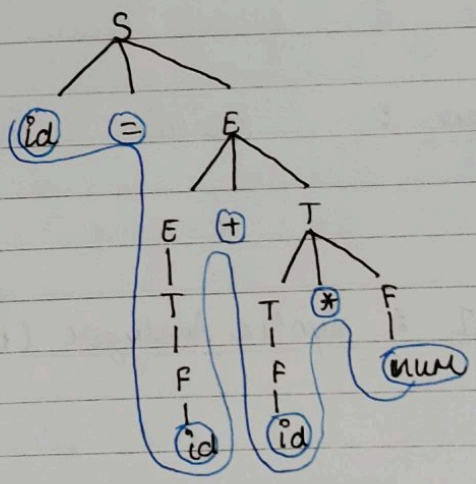
A typical representation is a syntax tree in which each interior node represents an operation and the children of the node represents arguments of the operations.

Example : For the same example mentioned previously.

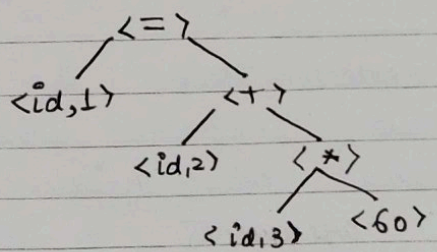
The give productions are :

$$S \rightarrow id = E$$
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow id \mid num$$

1st parse tree is made :



Now syntax tree is :

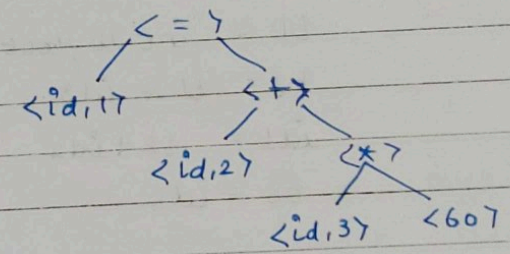


* Phase III : Semantic Analyzer :

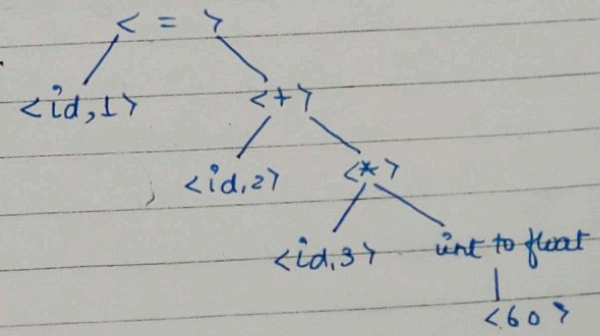
Semantic Analyzer uses syntax tree & the information in the symbol table to check source program for semantic consistency with the language defination.

The important part of semantic analysis is Type checking where the compiler checks that each operator has matching operands.

Syntax tree (Syntax Analyzer) :



Syntax tree (semantically verified) :



* Phase IV : Intermediate code Generator :

We consider an intermediate form called 3-address code which consists of a sequence of assembly like instruction with three operands / ~~registers~~ instruction.

Each operand act like a register.

Some 3-address instruction have fewer than 3 operands.

$$t1 = int\ to\ float\ (60)$$
$$t2 = t1 * id3$$
$$id1 = t2 + id2$$
$$t3 = id1$$

13th Jan'17
Lecture-4

* Phase V : code optimization

In this phase, the above formed intermediate code is optimised. The code is optimised by running the code by removing some steps that are of no importance.

The optimised code is :

$$t1 = id3 * 60.0$$
$$id1 = id2 + t1$$

* Phase VI : Target code Generator :

```
LDF   R1, id3
MULF  R1, R1, #60.0      → immediate constant.
LDF   R2, id2
ADDF  R1, R1, R2
STF   id1, R1
```

The above mentioned code is the target code. It is generated from the optimised code formed previously.

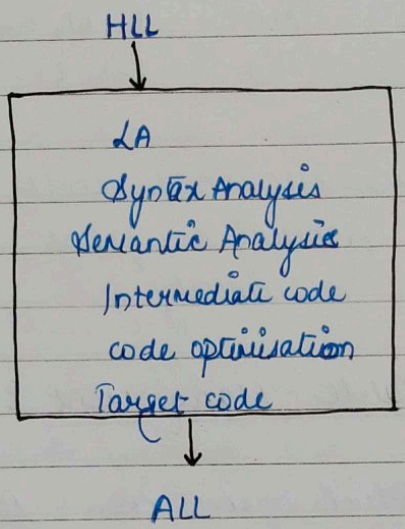
* Symbol table management :

An essential function of a compiler is to record the variable's name used in the source program & collect information about various attributes of each name. These attributes may provide information about the storage allocation for a name, its type, its scope & in the case of procedure name such thing as the number & types of its arguments, the method of passing each argument & type return so the symbol table is a data structure containing a record for -

each variable name with fields for the attributes for the name.

Passes :

* Single pass :

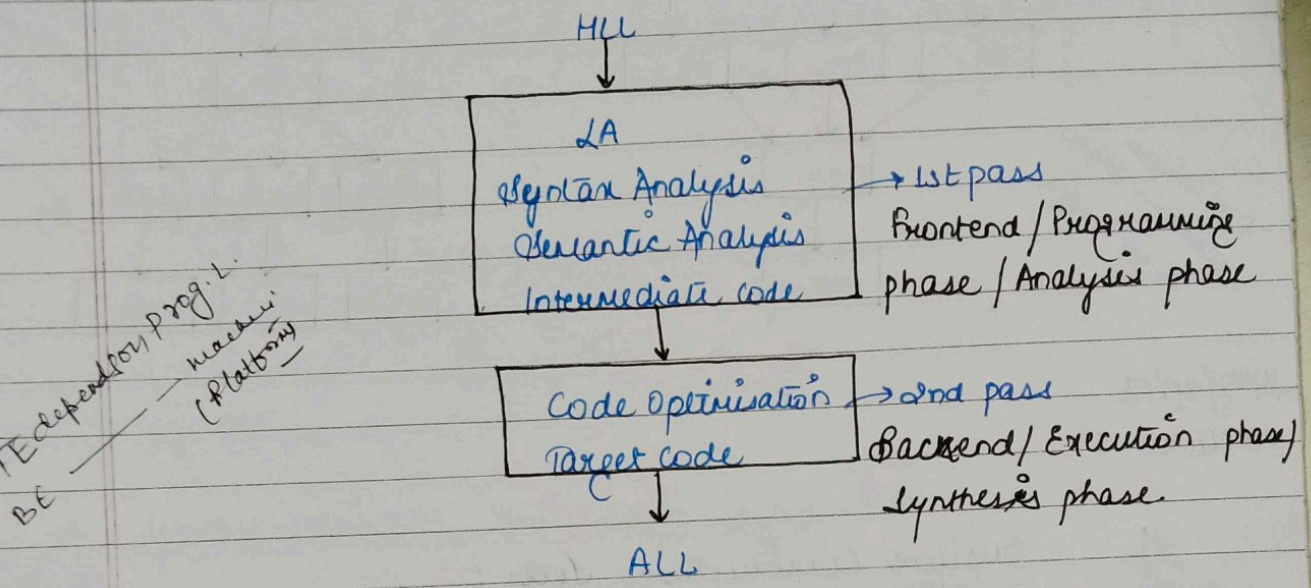


In this phase, all phases are in single module (pass) depending upon the implementation requirements.

- Advantage :
  - No communication gap (components are closely related)
  - Speed is fast
  - ~~to~~ No need to keep track of intermediate files.

- Disadvantage :
  - compiler tends to impose some restrictions to programmer. → Ex: data type, constants, procedures need to be define before use
  - Large amount of memory is required.
  - They may not allow for better provision of error handling, code optimisation & error reporting.

* 2 pass compiler :



* Multipass :

In this, compiler phases are grouped into multipass.

Advantages :

- Portable,

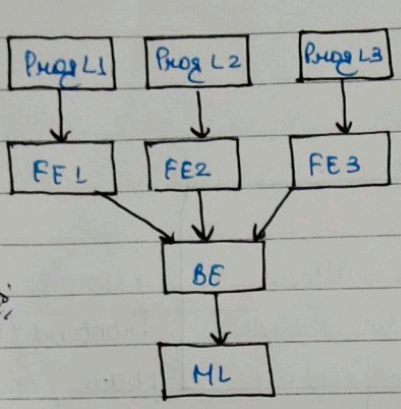
Disadvantages :

- less speed
- less space

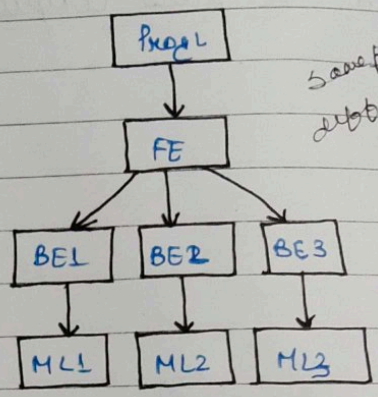
```
int max(x,y)
int x,y;
/* find max of x and y */
{
return((x>y)?x:y);
}
```

(28)

printf("%d", &a);



For same machine different prog languages.



Same prog L different machines

14th Jan'17
Lecture - 5.

5 marks.

Compiler Construction Tools :

The compiler writter like any software developer can use modern software development environment containing tools such as language editors, debuggers, version managers, profilers & so on.

In addition to these general software development tools other more sophisticated tools created to help various cases of a compiler.

Some commonly use compiler construction tools includes :

1) Scanner Generator :

They produce lexical analyzers from a RE descriptions of the token of the language.

2. Parser Generator :

That automatically produce syntax analyzers from a grammatical description of a programming language.

3. Syntax directed translation engines :

That produce collection of routines for walking a parse tree & generating a intermediate code.

4. code-Generator Generators :

That produce a code-generator from a collection of rules for translating each operation of the intermediate language into machine language for a target machine.

5. Data flow analysis engines :

That facilitates the gathering of information about how values are transmitted from one part of a program to another part.

Date: / /
Page:

14th La
Lectur

Data flow analysis is a key part of analysis.

6. Compiler construction tool kits :

That provide an integrated set of routines for various cases of compiler.