Nazia Mam

Book Ref:

1. Compilers Principles, Techniques, Tools
   by Jeffrey Ullman, Alfred Aho

Topic:

Scanned by CamScanner

Lecture-01

28.08.2019

Compiler:

High level language → Machine language

source code →

```
# include <stdio.h>
int main()
{
  ...
  return 0;
}
```

↓

Preprocessor

↓

compiler

↓ assembly language

Assembler

↓

Linker → static
Linker → dynamic

↓

Loader → exe

* Scanner:

* Parser: grammar কে parse করা

Scanned by CamScanner

* lexical analyzer:

Keyword, variable etc আলাদা করা।

প্রতিটি keyword-কে আলাদা token number দেয়া হয়।

* Syntactic Analysis:

statement validate করা।

* Semantic Analysis:

meaningful কি না তা check করা।

Scanned by CamScanner

Lecture-02

04.09.2019

Chap-02

A simple one-pass compiler

One-pass?

%% >> . print char %%

>> abc
>> char
>> char
>> char

প্রতিটি character-এর জন্য print হয়, এটিই one-pass.
one by one analyse করে।

Entire compiler Process:

→ Grammar for Syntax
→ Syntax-directed
⋮

Grammars for Syntax Definition:

→ CFG: is utilized to describe the syntactic structure of a language.

Scanned by CamScanner

characterized by-

1. Tokens / Terminal
2. Non-terminals
3. Production rules
4. Start symbol

let a CFG be:

*list → list + digit
| list - digit
| digit

digit → 0|1|2| ... |9

eg., 9-5+2

⇒ digit - digit + digit

⇒ list - digit + digit

⇒ list + digit

⇒ list (so grammar matched)

[0-9] digit

[+,-,*,/] +- */

Scanned by CamScanner

- Parse tree generate:

input: 9-5+2

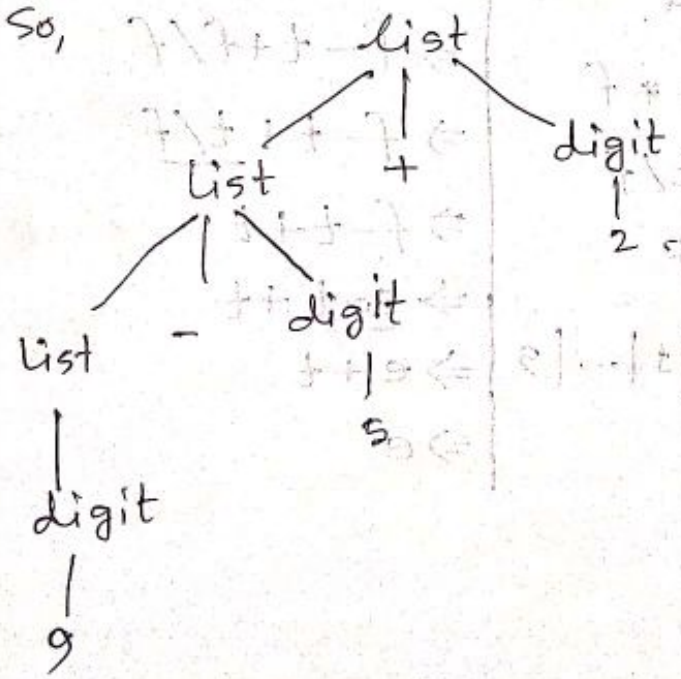
List = List + digit

= (List - digit) + digit

= digit - digit + digit

= 9 - 5 + 2

So,



[I] Ambiguity:

Same grammar এর জন্য একাধিক parse tree. থাকলে, compiler decision নিতে পারে না।

Scanned by CamScanner

* Associativity of operators → left, right

* Operator precedence

input: 9-5 * 2+6/8

grammar:

$e \rightarrow e+t$
$\quad | e-t$
$\quad | +$

$t \rightarrow t*f$
$\quad | t/f$
$\quad | f$

$f \rightarrow 0|1|\ldots|9$

অপারেটর precedency বেশি হলে আগে parse করতে হবে

Then,

$f-f*f+f/f$

$\Rightarrow f-\underline{t*f}+f/f$

$\Rightarrow f-t+f/f$

$\Rightarrow f-t+\underline{t/f}$

$\Rightarrow f-t+t$

$\Rightarrow \underline{e-t}+t$

$\Rightarrow e+t$

$\Rightarrow e$

Scanned by CamScanner

Lecture-03

11.09.2019

Syntax for statement:

statement → assignment থাকবে ।

Syntax-Directed Translation:

- Syntax-Directed definition:

  ↳ set of semantic rules থাকবে

  ↳ set of attributes

Attribute

↳ Synthesized attribute

e.g., $e \to \underline{e.t} \parallel t.t \parallel '+'$

↳

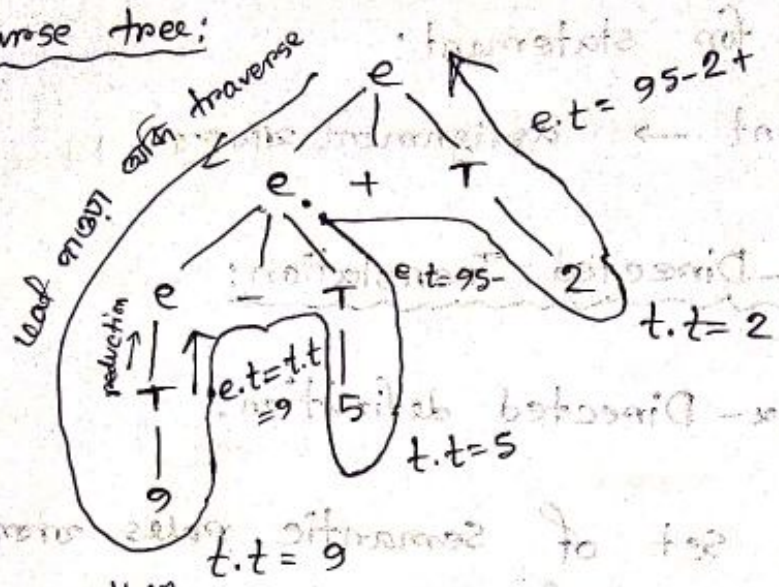
Each production rule of CFG has a semantic Rule:

PR	SR
e.g., $e \to e + T$	1. $e.t = e.t \parallel t.t \parallel '+'$
$\to e - T$	2. $e.t = e.t \parallel t.t \parallel '-'$
$\mid T$	3. $e.t = t.t$
$T \to 0 \mid \ldots \mid 9$	4. $t.t = '0' \parallel \ldots \parallel '9'$

Scanned by CamScanner

input : 9-5+2

parse tree:



For T = 9, then

T → e reduction
e → T production

Top down traverse করলে → Synthesized attribute (parent node-এ child-এর value)

Translation Scheme:

- Semantic rules is explicitly shown.

Parsing:

Two types → Top-down
→ Bottom-up

Scanned by CamScanner

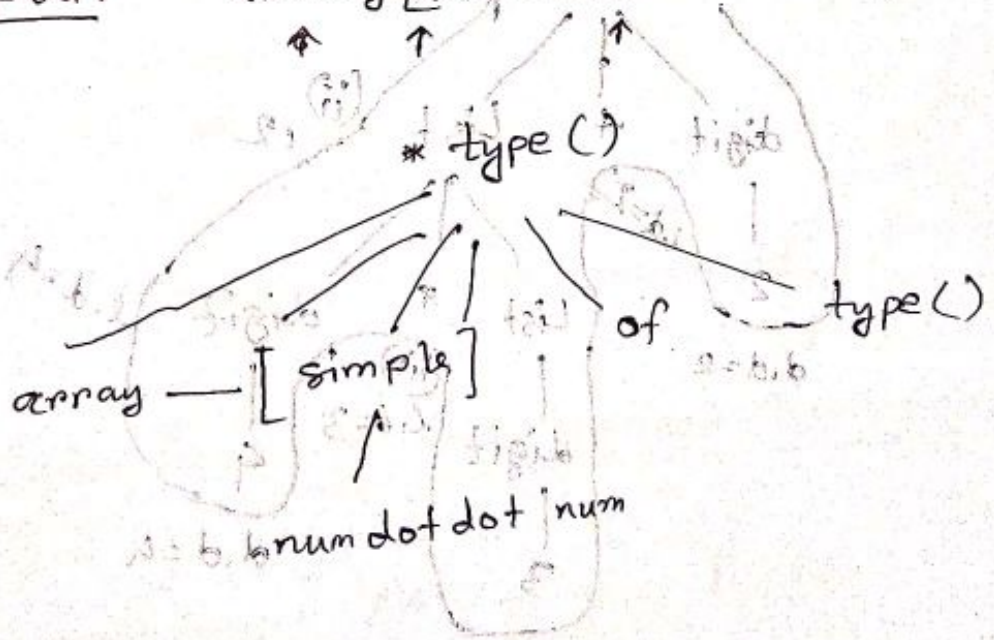
- **Top-down Parsing:**

CFG:

→ type → simple
| ↑ id
| array [simple] of type

simple → integer
| char
| num dot dot num

input: array [num dot dot num] of integer



- **Predictive Parsing:**

Backtrack না করে procedure appling call করে।

Scanned by CamScanner

Task: 2 + 3 * 4

CFG:

i. list → ~~list~~ digit + list

ii. | list * digit

iii. | digit

iv. digit → 0 | .. | 9

Semantic rule

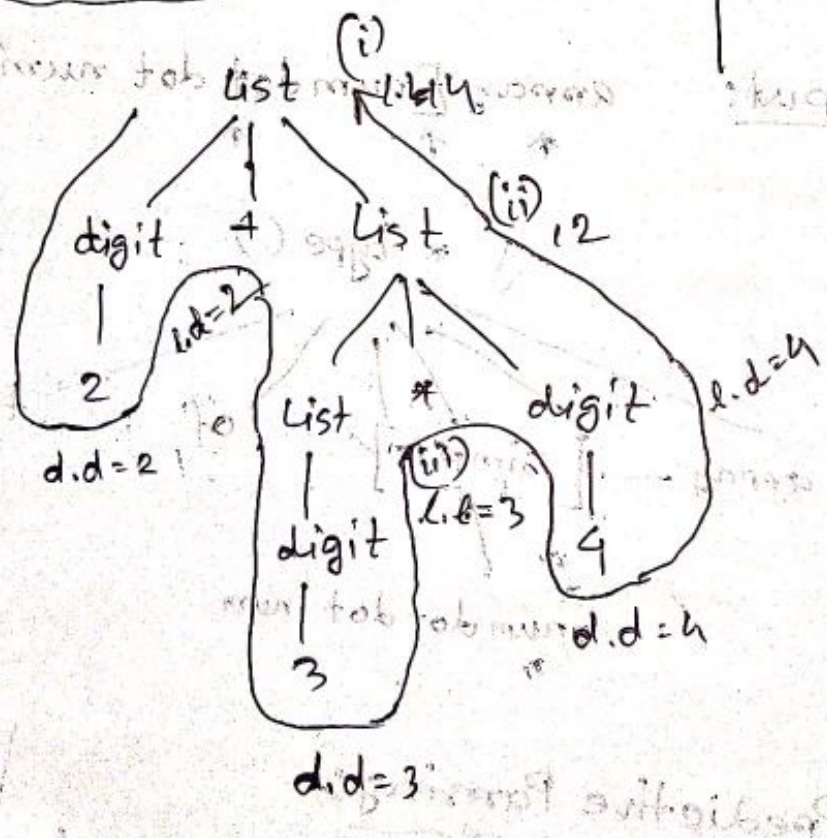
i. l.l = l.d ~~+ l.d~~ ~~l.d~~ + l.l

ii. l.l = l.l ~~+~~ * ~~+~~ l.d

iii. l.l = l.d

iv. d.d = '0' || .. || '9'

Annotated PT:



[প্রতি node-এ value assign করতে হবে]

Lecture - 04

18.09.2019

Functional Responsibilities:

1. White space & comments are filtered
2. Constant identify.

   eg. 32 + 57 + 17

   → <num, 32> + <num, 57> + <num, 17>
3. Recognizing identifiers and keywords.

   ↳ variable

   ↳ Arrays

   ↳ Functions

   & Keywords are reserved.

Interface to the Lexical Analyzer:

input → LA → Parser

← Push-back

LA:

↳ Read character

↳ Group them into lexeme

↳ Passes the token with attribute values

Scanned by CamScanner

* Why push back?

→ same type বারবার পেলে buffer-এ পাঠাও।

'>=' → ২টা character মিলে ১টি operator বোঝাও। push back না করলে '>', '=' আলাদা আলাদা mean করত / tokenise করত।

* Token handle করতে LA-তে একটি Token ~~no~~ & Symbol table থাকে।

let, Var [a-z]+

Digit [0-9]+

Type "int" | "float"

Assigop "="

তাহলে,

int a=10

lexptr	token	attribute
	type	257
	Var	10
	Assig op	258
	Digit	10

Scanned by CamScanner

Attribute-গুলো semantic rule অনুসারে হয়।

e.g., Var.t ⇒ Digit.t

Digit.t → '0' | '1' | ..... | '9'

Abstract Stack Machines:

↳ instruction memory

↳ Data "

↳ Arithmatic ...

* L-value & R-value:

Location value:

Real value:

e.g., I := 5 → real value

↓

Loc. value

* Stack Manipulation:

→ push v

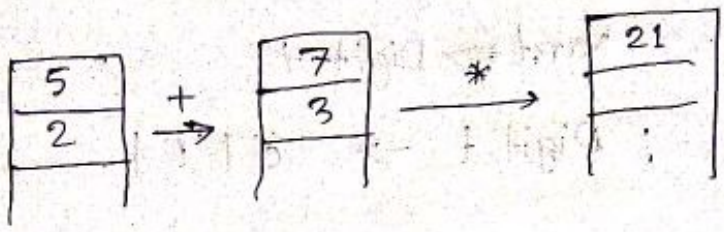
→ rvalue I

→ lvalue I

→ pop

Scanned by CamScanner

example: 52+3 *



Control Flow:

- label
- goto
- gofalse
- gotrue

details পড়তে হবে না।

*Review previous lecture:

left factoring : List → list + digit
→ list − digit + digit

Right " : List → digit + list
→ digit + list − digit

* 2+3−8+2

= digit + d − d + d

= list + d − d + d

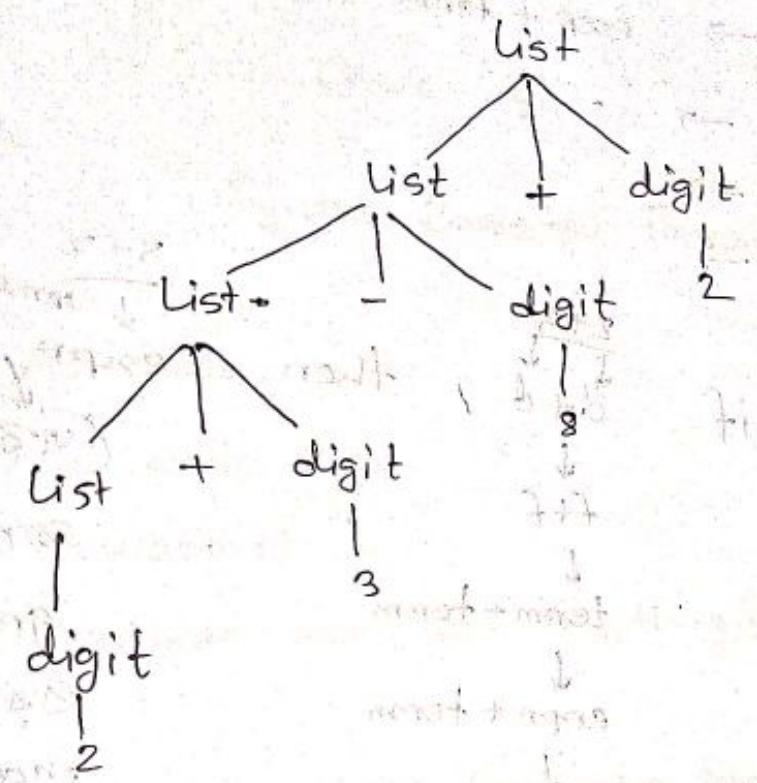
= list − d + d

= list + d

= list

Scanned by CamScanner

Parse tree:



Task:

Expression: if 2+3 then 5+7 else 2+5

Syntax for Statement:

stmt → id := expr
| if expr then stmt
| if expr then stmt else stmt
| while expr do stmt
| begin opt_stms end

Scanned by CamScanner

expr → exp. + term

term →

f →

d →

Solⁿ: if $\overset{2+3}{d+d}$ then $\overset{5+7}{\text{expr}}$ same way

$d+d$
↓
$f+f$
↓
$term+term$
↓
$expr+term$
↓
$expr$

expr ↓ ...

তেরকম

আর কোনো grammar এর সাথে match করেনা।

—: Chap - 02 শেষ :—

Scanned by CamScanner

Lecture - 05

24.09.2019

Chap - 05

Syntax Directed Translation

- Attribute
- Semantic Rule
- Evaluation of "
- Attribute can hold anything, such as: String, int...

SDT vs TS (translation Scheme)

↓

(operation order বলে দেয় না)

Annotated PT : প্রতি node-এ value semantic rule অনুসারে। এই property কে "annodity" বলে।

* $A \rightarrow \alpha$, where A হল parrent, α হল child.

Scanned by CamScanner

It is associated with a set of semantic rule — of the form

$$b = f(c_1, c_2, \ldots, c_n)$$

↓

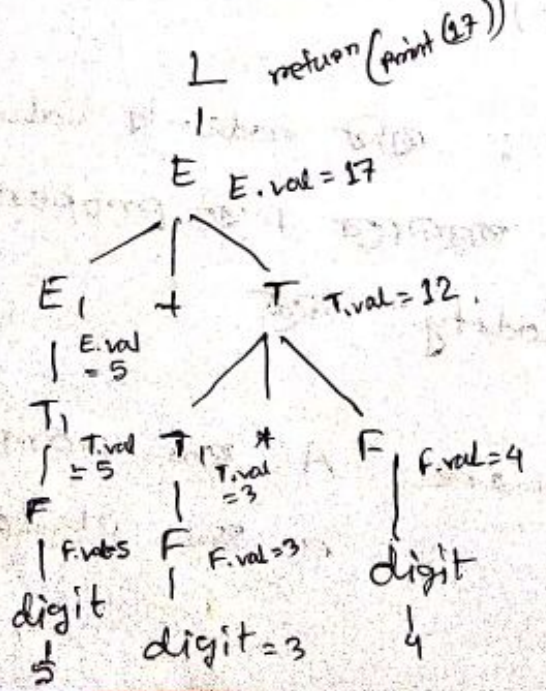
Synthesized

where, c_1, c_2 are symbols in production and b is attribute

Ques: *Task: $5+3*4$,

Create an annodity parse tree.

Soln:



Productions:

$L \rightarrow E$ return

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

$T \rightarrow F$

$F \rightarrow (E)$

F — digit

Scanned by CamScanner

Now, Semantic rule use করে annodity PT create:

* প্রতি node-এ value দিতে হবে।

Dependency Graph:

- bottom up.
- operator থাকবে না

Scanned by CamScanner

25.09.2019

Lecture-06

Syntax Directed Definition

- example 2:
- Inherited Attributes:

input: real id1, id2, id3.

Production

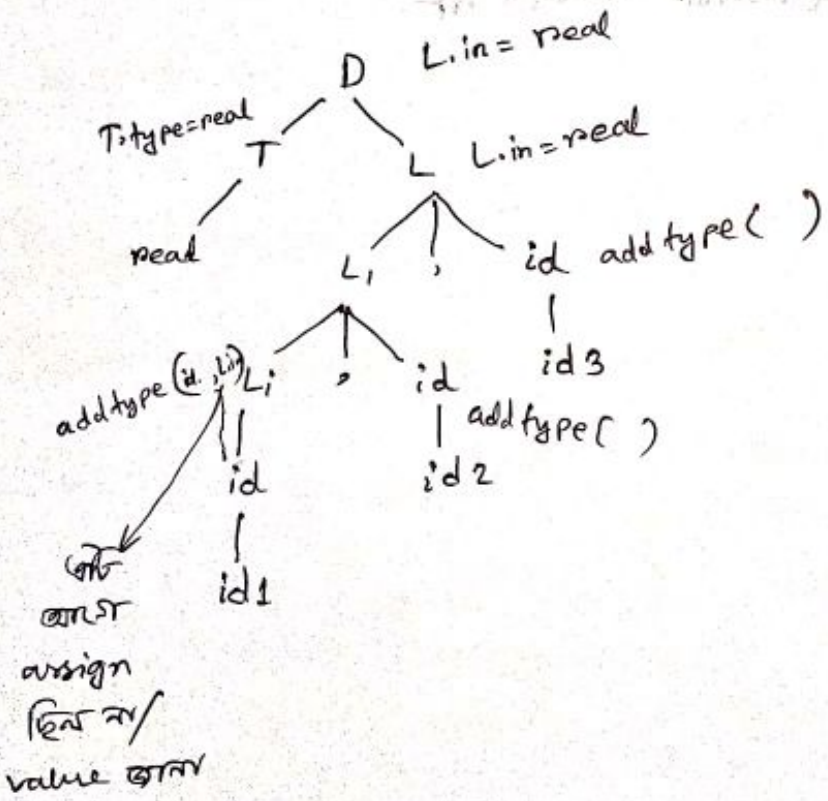
$D \rightarrow TL;$

$T \rightarrow int$

$T \rightarrow real$

$L \rightarrow L_1, id$

$L \rightarrow id$



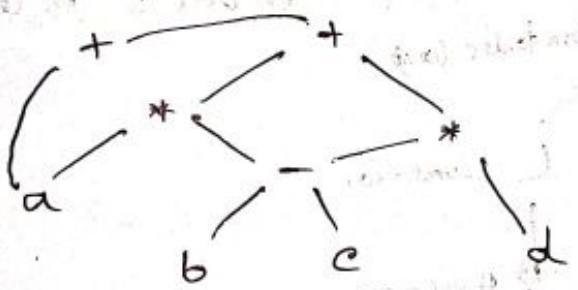
ছিল না জন্যে আবার Traverse করে L, D তে গেছে L.in এর value determine করে then পুনরায় ফিরে আসতে হয়, func-এ L.in এর value বসাতে গিয়ে হয়। এটিই inherit করে পাওয়া attribute

Syntax Tree:

non terminal হলে node তৈরি করা হয়।
" " leaf " " "

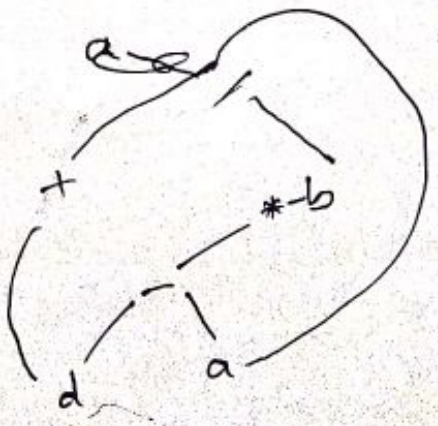
Directed Acyclic Graph (DAG):

expr: $a + a * (b-c) + (b-c) * d$



*operator অনেক বার আসতে পারে কিন্তু, variable একবার করেই থাকবে।

expr: $d + a / b * (d-a)$



Scanned by CamScanner

* Spot test Solution:

CFG:

$S \rightarrow L.L \mid L$

$L \rightarrow LB \mid B$

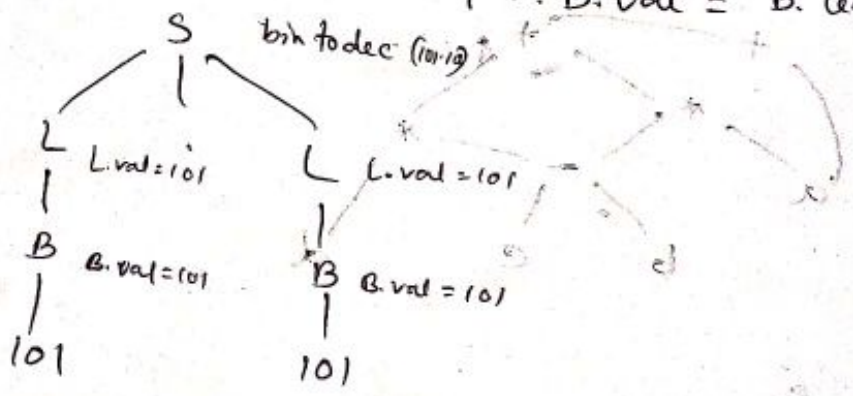
$B \rightarrow [0|1]+$

SR

1. S.val = bin_to_dec(L.val '.' L.val) | bin_to_dec(L.val)

2. L.val = L.val || B.val | B.val

3. B.val = B.lex val = [0|1]



Scanned by CamScanner

Lecture - 07

26.09.2019

Ch - 06

Type Checking

Static Checking:

- Type checks : '%' operation শুধু int type এর জন্য কাজ করে - এসব ব্যাপার check
- Flow control check : break (outside while?)
- Uniqueness : labels in case statement
- Variable .... : same?

Type Systems:

* int + int = int
→ user defined type (class/struct) and built-in type (basic)

Scanned by CamScanner

→ typedef

→ Array

→ $T_1 \times T_2 = T_3$ (T = int)

→ Pointer

→ Record

→ Function

গুলো হল Type expression. (basic + constructed)

Ques 1. Type expr কি? কত প্রকার + example : 2 type

2. Static checking. Kinds of :

3. Type system

Scanned by CamScanner

Lecture-08

02.10.2019

Type expression ২ রকমের :

i. Basic type : int, float etc

ii. Type constructor : Array, class, struct ...

let, var A: array [1..10] of int

(a PASCAL declaration).

তব Type expression :

T.E = array(1..10, int)

For, c variable dec : int A[10],

তব Type expression :

T.E = array (10, int)

dec

- int *P ⟶ T.E = pointer (int)

- type row = record
  address: integer
  lexeme: array [1..15] of char
  ends;

  T.E = record (address × integer) × (lexeme × array (1..15, char))

Scanned by CamScanner

এইক্ষেত্রে type expr. -এ variable নামও দিতে হয়।

function এর জন্য:

→ $D \rightarrow R$ (domain → range)

* $f(\underbrace{a,b}_{input} : char) : \underbrace{\uparrow int}_{output}$ এর TE —

TE : (int x char) → pointer (int)

* type row = record

- - -

: : :

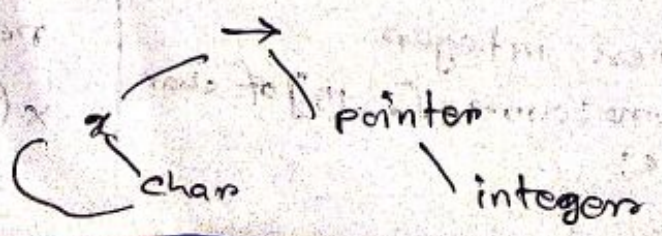
var table : [1 .. 10] of row

এটার Type Expression:

TE = array (10, row)

Representation of TE:

- (char x char) → pointer (int)

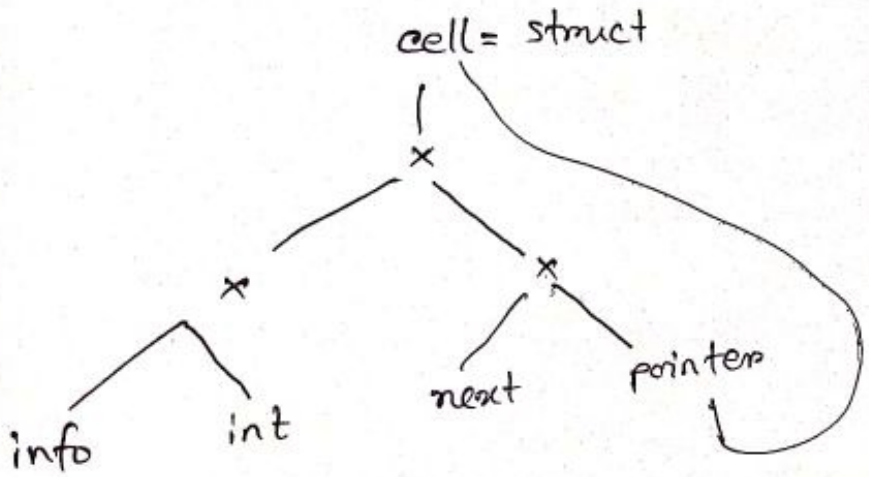


Scanned by CamScanner

```
struct cell
{
    int info;
    struct cell * next;
}
```

ও TE —

TE : struct ((info × int) × (next × pointer(cell)))



- Static Vs. Dynamic Checking } less imp.
- Error recovery
- Translation Scheme for Declaration :

(Simulation) → xm- ও বোঝাও