04/21/2025

2. assign values to data areas

Values. scope local
global

lifetime: automatic
static
dynamic

automatic: lifetime of a variable matches the single allocaton of the scope that declares it.

```
void f (int j )
{   int i = 0;   ]  scope of i
        :
}
```

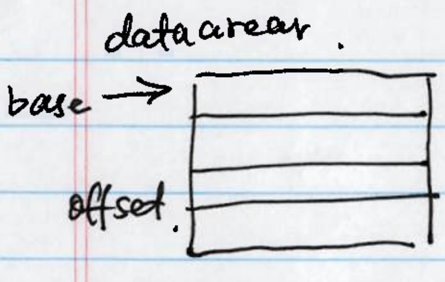
static. lifetime spans multiple activatns of its declaraton.

```
void f (  )
{ static int c=0;
   c++;
}
```

dynamic : lifetime depends on explicit allocation/ deallocation.

```
Person *p = new person(  ) ;
delete p ;
```

scope.	lifetime.	locaton .
local.	automatic	registers . or local area of . declaring scope
local	static	procedure/file static data area .
global	static .	global data area .



ch5. syntax driven translation (SDT).

The first translation from source code to IR

① . The compiler writer specifies. action that should be taken when the parser reduces by a given ~~specifies~~. production

② . the parser generator arranges for action to execute at the appropriate ~~parsing~~ points in the parsing process.

5.3.

ex1. compute the value of a num.

1. num → DList
2 DList → DList digit
3. | digit

state	Action		goto
	eof	digit	DList
0		s2	1
1	acc	s3	
2	r3	r3	
3	r2	r2	

MSD LSD .
175

Hornor's rule $((1 \times 10) + 7) \times 10 + 5$

associate a value with each symbol used in the parser. encode this strategy into .each rule.

Notation: yacc bison.

$$: value of LHS of the rule

$1 : value of the first symbol on the RHS.

$2 : second.

```
1.  num → DIST                $$ = $1
                                              ctoI
2.  DIST → DIST digit         $$ = $1 *10 + ($2)
          | digit             $$ = ctoi ($1)
```

change to skeleton parser

push (symbol, state) . ⟹ push (symbol,
state,
value).

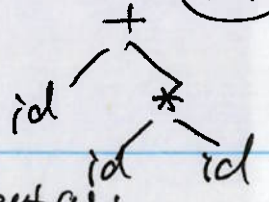
$1 \times \underline{\underline{100}} + 7 \underline{\underline{\times 10}} + 5$

"175"

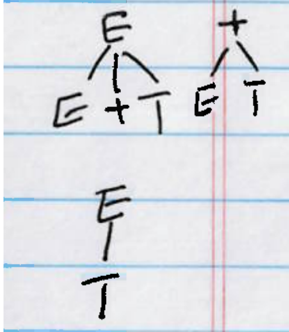
123.

state	word	stack	action	
—	d_1	< n, 0, null>	—	
0	d_1	< n, 0, null>	s2	
2	d_7	<n,0, null> <d_1, 2, "1">	r3	DList → digit
1	d_7	<n,0, null> <DL, 1, 1>	s3	
3.	d_5	<n,0, null> <DL,1,1> <d_7, 3, "7"> .	r2	DL → DL d .
		\$1 x10 + ctoi(\$2)		
1	d_5	<n,0,null> < DL, 1, 17 >	s3	.
3	eof	<n,0, null> <DL, 1, 17> <d_5, 3, "5">.	r2 .	
		\$1x10 $\underline{\underline{\$2}}$		
1	eof	<n, 0, null> < DL, 1, $\underline{\underline{175}}$ >	accp.	

~~id + id * id~~,



ex2. build AST.   value: root of current ast. "+"



$1 $2 $3

E → E + T $$ = make node2 (plus, $1, $3)

E − T $$ = make node 2 (minus, $1, $3).

T $$ = $1

T → T*F ;

T/F ;

F ; $$ = $1

F → num $$. makeleaf (num, lexem).

name   ~~$$~~= makleaf (name, lexeme)

(E) $$ = ~~n$~~ 2

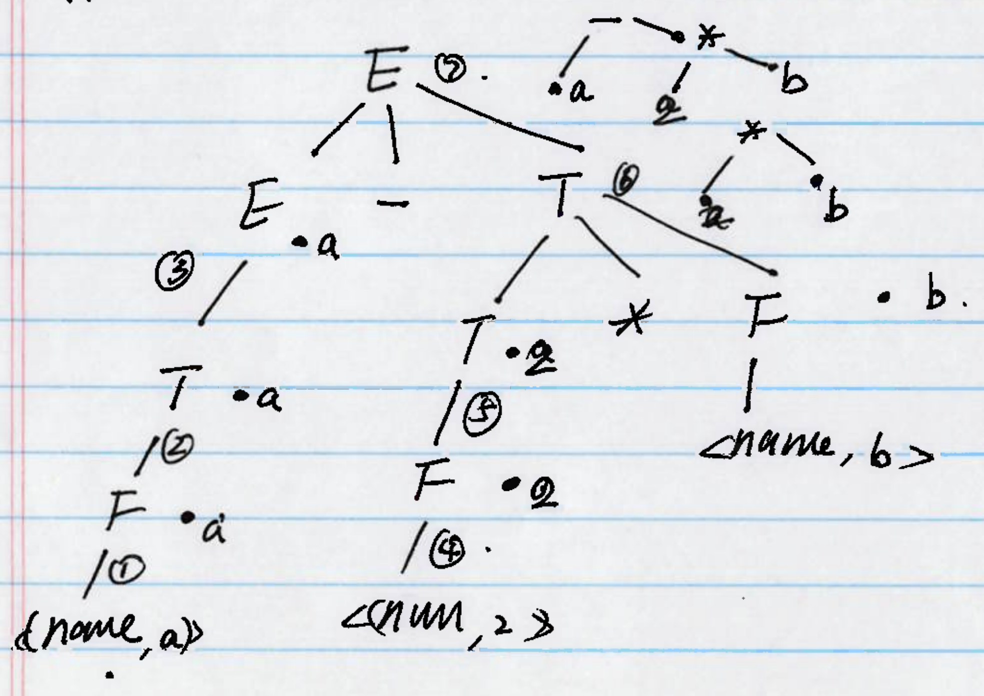
↑ ↑ ↑

$1 $2 $3

name − num * name .

E → E − T

T → T * F

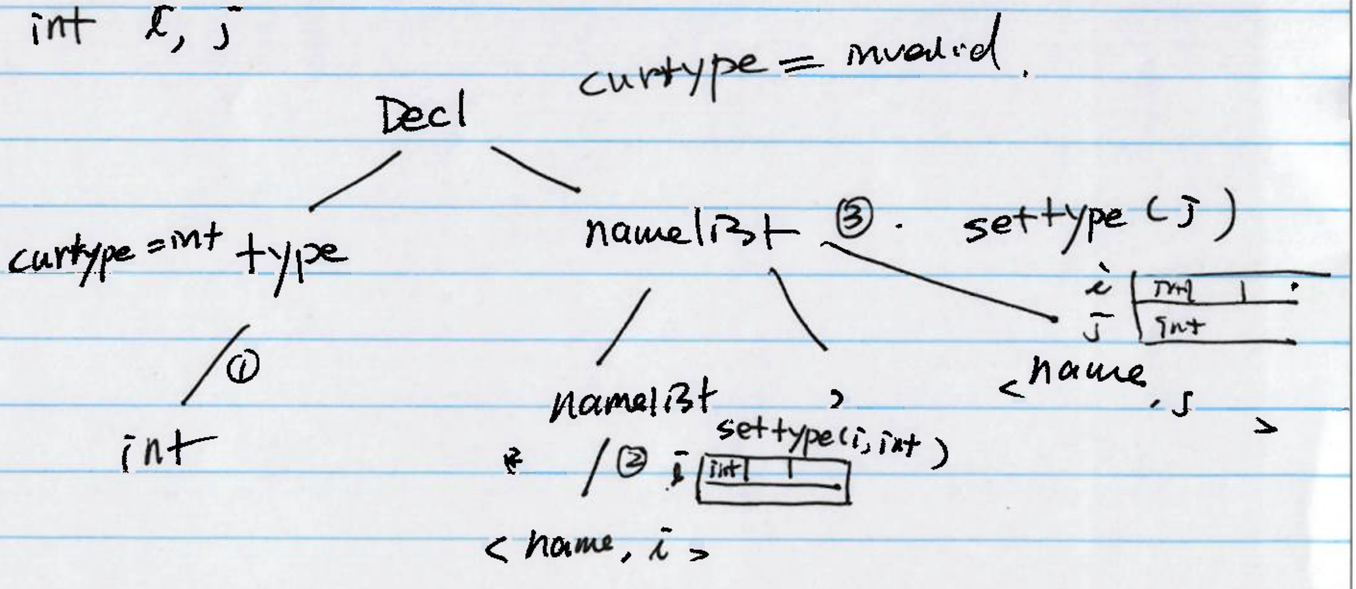


use global varriable ~~to~~ to comunicate info betwee the grammer rules.

ex:

```
Decl → type namelist            curent type ← invali
type → int                      ~~curre~~ curtype ← int
     | float                    curtype ← float
namelist → name .               settype($1, curtype).
         | namelist , name      settype( $3, curtype)
           ↑        ↑   ↑
           $1       $2  $3 .
```

int i, j


curtype = invalid.
Decl
curtype = int
type
①
int
namelist
③ .
settype (j)
i int
j int
<name, j>
namelist
,
②
settype(i, int)
i int
<name, i>