03/03/22

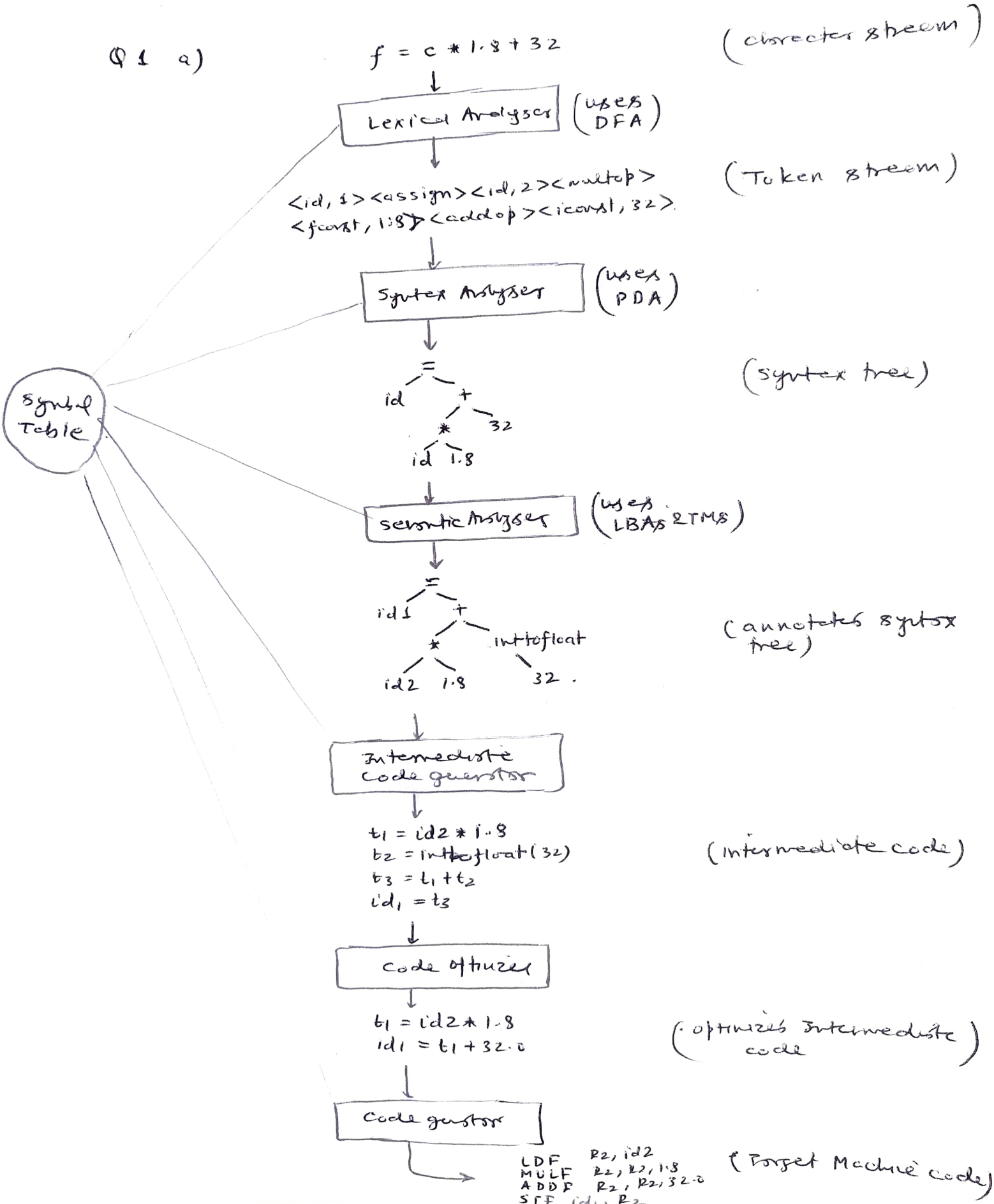
1st Sessional

Compiler Design

YASH VINAYVANSHI

19BCS081.

Q1 a)



Lexical Analyser
- checks misspelling

Syntax Analyser
- checks structure of sentence

Semantic Analyser.
- checks logic of program.

Intermediate Code Generator

Code optimizer

code generator.

Q1. b) Importance of two buffer scheme of Lexical Analysis.

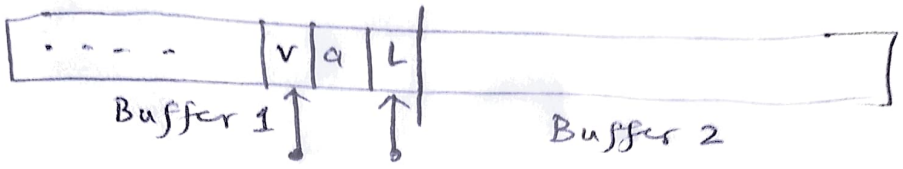
→ If a token is not completely brought into the buffer, it presents the problem of wrong lexical analysis.

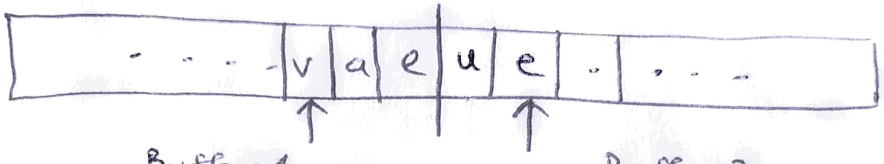
→ To solve this problem without requiring dynamic allocation of buffer, two buffer scheme is adopted

→ The buffer part on which lexical analyser is currently working is not replaced immediately, rather the remaining part is brought into the second buffer & the token which is broken into two parts between two buffers can be

correctly analysed using caterpillar movement of pointers as long as length of token is not more than size of second buffer is remains

.... value ....





Buffer 1
(Not overwritten)

Buffer 2
(Brought in next block containing 'ue' part of 'value')

Q2. Rules to compute First().

1. If X is a terminal → FIRST(X) = {X}.
2. If X is a non terminal
3. If X is nonterminal & X → $Y_1 Y_2 \dots Y_k$, then place 'a' in FIRST(X) if for some i, 'a' is in First(Y), and ε is in all of FIRST(Y_i) ... FIRST(Y_{i-1}) ie $Y_1 \dots Y_{i-1} \Rightarrow \varepsilon$. If ε is in FIRST($y_j$) for all j = 1 to k. Then add ε to FIRST(X)
4. If X → ε is a production, then add ε to FIRST(X)

Rules to Compute Follow()

1. Place $ in FOLLOW(S) where S is ~~start~~ symbol. ($ is the input right endmarker)
2. If there is a production A → α B β, then everything is in FIRST(β) except ε is in FOLLOW(B).
3. If there is a production A → α B, or a production A → α B β where FIRST(β) contains ε, then everything in FOLLOW(A) is in FOLLOW(B)

Example.

G : S → A B C
A → a | b | ε
B → c | d | ε
C → e | f | ε.

First(A) = First(a) ∪ First(b) ∪ First(ε) = {a, b, ε}
First(B) = First(c) ∪ First(d) ∪ First(ε) = {c, d, ε}
First(C) = First(e) ∪ First(F) ∪ First(ε) = {e, f, ε}
First(S) = First(A)
but if A → ε, then First(B)
but if A, B → ε then First(C).

= First(A) ∪ First(B) ∪ First(C)
= {a, b, c, d, e, f, ε}

Follow(S) = {$} (bcz S doesn't occur in RHS of any production)

Follow(A) = First(B)
but if B → ε, then First(C)
but if B, C → ε, then Follow(S).

= First(B) ∪ First(C) ∪ Follow(S)
= {c, d, e, f, $}

Follow(B) = First(C)
but if C → ε, then Follow(S)

= First(C) ∪ Follow(S)
= {e, f, $}.

Follow(C) = Follow(S)
= {$}.

∴ Finally,

First(A) = {a, b, ∈}
First(B) = {c, d, ∈}
First(C) = {e, f, ∈}.
First(S) = {a, b, c, d, e, f, ∈}.

Follow(A) = {c, d, e, f, $}
Follow(B) = {e, f, $}
Follow(C) = {$}
Follow(S) = {$}.

Q3 Given Grammar :

G : $S \rightarrow SbA \mid Ba$.
$B \rightarrow bB \mid t$
$A \rightarrow aA \mid ab$.

(i) Making grammar suitable for parsing.

step 1 : Removing left recursion
ie productions of type $A \rightarrow A\alpha \mid \beta$ $\alpha, \beta \in (V \cup T)^*$
replaced with $A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \epsilon$.

There is left recursion in production $S \rightarrow SbA$ in G ~ modified grammar will be

G' : $S \rightarrow BaS'$
$S' \rightarrow bAS' \mid \epsilon$.
$B \rightarrow bB \mid t$
$A \rightarrow aA \mid ab$

step 2 : Application of Left Factoring.
ie productions of type $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots$
replaced with $A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \ldots$.

Left factoring can be applied to production $A \rightarrow aA \mid ab$ of G' ~ modified grammar will be

G'' : $S \rightarrow BaS'$
$S' \rightarrow bAS' \mid \epsilon$.
$B \rightarrow bB \mid t$
$A \rightarrow aA'$
$A' \rightarrow A \mid b$

(ii) Producing Parsing Table of G''.

G'' : $S \to BaS'$
$S' \to bAS' \mid \epsilon$
$B \to bB \mid t$
$A \to aA'$
$A' \to A \mid b$

Step 1: calculating First() & Follow() sets

First(A) = {a}.
First(B) = {b, t}.
First(A') = {b}.
First(S') = {b, ε}.
First(S) = First(B) = {b} & $B \xrightarrow{*} \epsilon$ doesn't occur

Follow(S) = {$} S don't occur in RHS of any prodn
Follow(S') = Follow(S) = {$}.
Follow(B) = First(a) = {a}.
Follow(A) = ——
Follow(A') = ——

Step 2: construction of parsing table

	a	b	t	$
S		$S \to BaS'$	$S \to BaS'$	
S'		$S' \to bAS'$		$S' \to \epsilon$.
B		$B \to bB$.	$B \to t$.	
A	$A \to aA'$			
A'	$A' \to A$	$A' \to b$.		

$S \to BaS' \rightsquigarrow$ First(RHS) = First(B) = {b, t}.
$S' \to bAS' \rightsquigarrow$ First(RHS) = first(b) = {b}.
$B \to bB$ Pd $\rightsquigarrow$ First(RHS) = first(b) = {b}.
$B \to t \rightsquigarrow$ First(RHS) = first(t) = {t}.
$A \to aA' \rightsquigarrow$ First(RHS) = first(a) = {a}.
$A' \to A \rightsquigarrow$ First(A) = {a}.

$S' \to \epsilon \rightsquigarrow$ Follow(LHS)
$\rightsquigarrow$ Follow(S')
= {$}

(iii) Taking a random example & parsing it.

sentence : bata.

STACK		INPUT	OUTPUT
$S		bata.	S → baS'
$S'aB.		bata.	B → bB.
$S'aBb	← match →	bata	
$S'aB		ata	X
			dead.

∴ string Not Parsed.
bata ∉ Language generated by G''