Class Test #1 (Set-A)
CSTE-4105 (Compiler Construction)
Date: 21/05/2024
SOLUTIONS

Answer the following questions: (Time: 45 minutes)

1. Differentiate between loader and linker. **5**

S. No.	Linker	Loader
1	A linker is an important utility program that takes the object files, produced by the assembler and compiler, and other code to join them into a single executable file.	A loader is a vital component of an operating system that is accountable for loading programs and libraries.
2	It uses an input of object code produced by the assembler and compiler.	It uses an input of executable files produced by the linker.
3	The foremost purpose of a linker is to produce executable files.	The foremost purpose of a loader is to load executable files to memory.
4	Linker is used to join all the modules.	Loader is used to allocate the address to executable files.
5	It is accountable for managing objects in the program's space.	It is accountable for setting up references that are utilized in the program.

2. Consider the following ambiguous grammar: **5**

Stmt →if expr than stmt | if expr then stmt else stmt | other

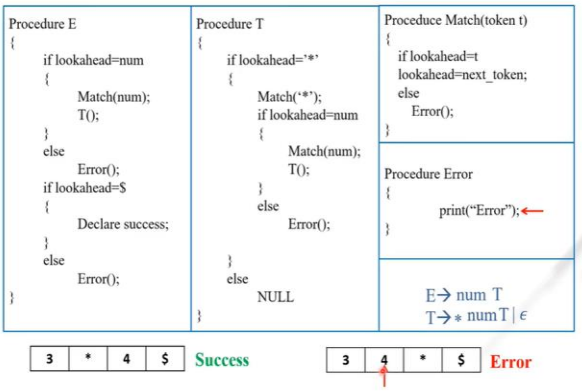
Rewrite the above grammar to eliminate ambiguity and then show the derivation for the compound conditional statement "**if E1 then S1 else if E2 then S2 else S3**".

Answer:

Stmt → *matchedstmt*
| openstmt
***matchedstmt* → if expr then matchedstmt else matchedstmt**
| other
openstmt → if expr then stmt
| if expr then matchedstmt else openstmt

3. Explain about recursive descent parsing with an example. **5**

1. Procedure begins with start symbol of the grammar
2. Scan the input left to right
3. **Non-Terminal** - Recursively replace NT with Production by checking with next input symbol (lookahead)
4. If more than one alternative production rule available for a non-terminal, then decision is based on comparison with lookahead symbol
5. **Terminal** – Matching with input string, advance the pointer to check with next input symbol
6. Procedure continues until it derives entire input string
7. Any step if it does not match to derive input string, apply backtracking



4. Construct LL(1) parsing table for the following grammar: **5**

S →(L)|a
L → SL'
L' → ,SL'|ε

Answer:

	First	Follow
S	(, a	$,)
L	(, a	)
L'	), ε	)

	(	)	a	$
S	S -> (L)		S -> a	
L	L -> SL'		L -> SL'	
L'		L'->(SL' L'->ε		

5. Explain the error recovery strategies in syntax analysis phase with appropriate examples. **5**

Answer:

1. Panic mode recovery
2. Phrase level
3. Error production
4. Global correction

Check slide no: 5 for more details.

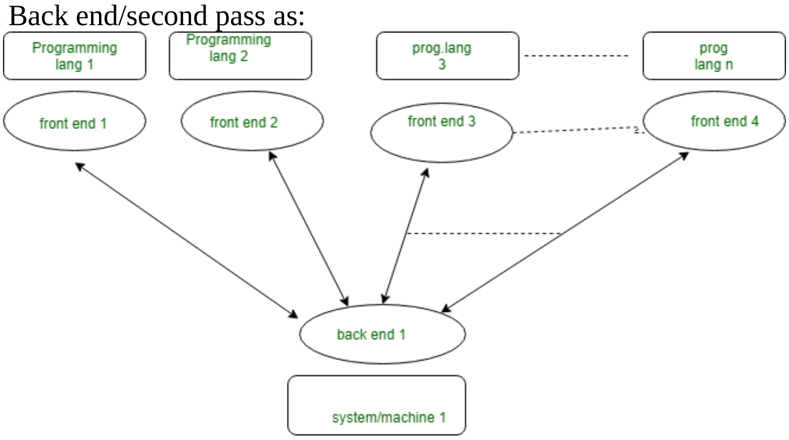
Class Test #1 (Set-B)
CSTE-4105 (Compiler Construction)
Date: 21/05/2024

Answer the following questions: (Time: 45 minutes)

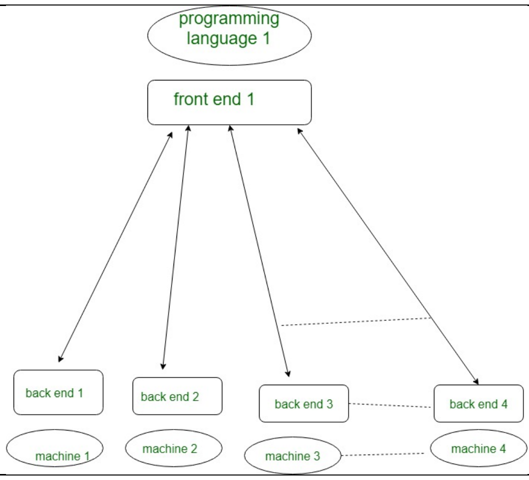
1. "Multi-pass compiler can solve two basic problems."-what are they? Explain with examples. **5**

Answer:

1. If we want to design a compiler for different programming language for same machine. In this case for each programming language there is requirement of making Front end/first pass for each of them and only one Back end/second pass as:

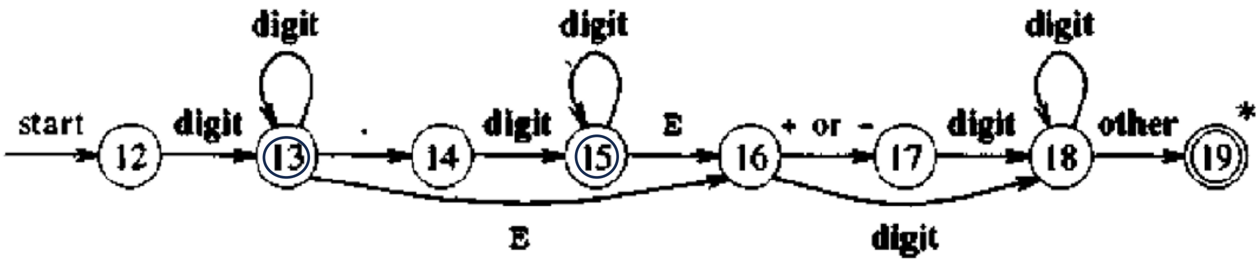


2. If we want to design a compiler for same programming language for different machine/system. In this case we make different Back end for different Machine/system and make only one Front end for same programming language as:



2. Draw a transition diagram accepting both integer and floating-point numbers with exponentiation. **5**

Answer:



3. What is left recursion of a grammar? Eliminate left recursion from the following grammar: **1+4=**

$A \to B\ C \mid a$

$B \to C\ A \mid A\ b$

$C \to A\ B \mid C\ C \mid a$

Answer:

A Grammar G is left recursive Grammar if the non-terminal A in the derivation is of the form:

$$A \overset{+}{\Rightarrow} A\alpha$$

Where α is a string of terminals and non-terminals.

Whenever the first symbol in the **right hand side of the production is same as the left hand side variable**, then the grammar is said to be a **left recursive grammar**.

$i = 1$: nothing to do

$i = 2, j = 1: B \to C\ A \mid \underline{A}\ \mathbf{b}$

$\Rightarrow\ B \to C\ A \mid \underline{B\ C}\ \mathbf{b} \mid \underline{\mathbf{a}}\ \mathbf{b}$

$\Rightarrow_{\text{(imm)}}\ B \to C\ A\ B_R \mid \mathbf{a}\ \mathbf{b}\ B_R$

$B_R \to C\ \mathbf{b}\ B_R \mid \varepsilon$

$i = 3, j = 1: C \to \underline{A}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow\ C \to \underline{B\ C}\ B \mid \underline{\mathbf{a}}\ B \mid C\ C \mid \mathbf{a}$

$i = 3, j = 2: C \to \underline{B}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow\ C \to \underline{C\ A\ B_R}\ C\ B \mid \underline{\mathbf{a}\ \mathbf{b}\ B_R}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow_{\text{(imm)}}\ C \to \mathbf{a}\ \mathbf{b}\ B_R\ C\ B\ C_R \mid \mathbf{a}\ B\ C_R \mid \mathbf{a}\ C_R$

$C_R \to A\ B_R\ C\ B\ C_R \mid C\ C_R \mid \varepsilon$

4. Prove that the following grammar is not LL(1): **5**

$S \to \mathbf{i}E\mathbf{t}SS' \mid \mathbf{a}$

$S' \to \mathbf{e}S \mid \varepsilon$

$E \to \mathbf{b}$

Answer:

FIRST(S)={i, a}	FOLLOW(S)={e, $}
FIRST(S')={e, ε}	FOLLOW(S')={e, $}
FIRST(E)={b}	FOLLOW(E)={t}

1) FIRST(iEtSS') ∩ FIRST(a) = ϕ

$\{i\} \cap \{a\} = \phi$

2) FIRST(eS) ∩ FIRST(ε) = ϕ

$\{e\} \cap \{\varepsilon\} = \phi$

3) FIRST(eS) ∩ FOLLOW(S') = ϕ

$\{e\} \cap \{e, \$\} \neq \phi$

Or,

	a	b	e	i	t	$
S	$S \rightarrow \mathbf{a}$			$S \rightarrow \mathbf{i}\ E\ \mathbf{t}\ S\ S_R$		
S_R			$S_R \rightarrow \varepsilon$ $S_R \rightarrow \mathbf{e}\ S$			$S_R \rightarrow \varepsilon$
E		$E \rightarrow \mathbf{b}$				

5. Show the comparisons among error recovery strategies in a lexical analyzer with examples. **5**

Answer:

1. Removes one character from the remaining input;
2. In the panic mode, the successive characters are always ignored until we reach a well-formed token;
3. By inserting the missing character into the remaining input;
4. Replace a character with another character;
5. Transpose two serial characters.

Class Test #1 (Set-C)
CSTE-4105 (Compiler Construction)
Date: 21/05/2024

Answer the following questions: (Time: 45 minutes)

1. Show the comparisons among lexeme, pattern and token with examples. **5**

Answer:

A token is a pair consisting of a token name and an optional attribute value.

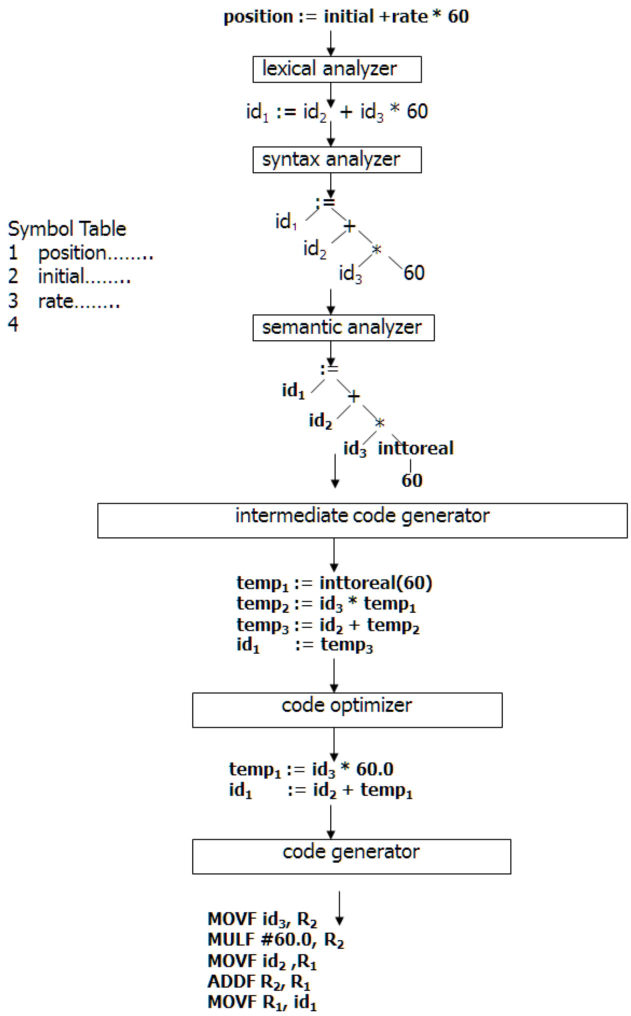
A pattern is a description of the form that the lexemes of a token may take [or match].

A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but ", surrounded by "'s	`"core dumped"`

2. Show the phases of compiler for the statement *position :=initial + rate*60.* **5**

Answer:



3. Calculate FIRST and Follow for the following grammar: **5**

$S \rightarrow$ **(**L**)|a**

$L \rightarrow L, S \mid S$

Answer: The given grammar is left recursive. First, we need to eliminate left recursion from the grammar. So, after eliminating left recursion, we will get the following grammar:

S →(L)|a

L → SL'

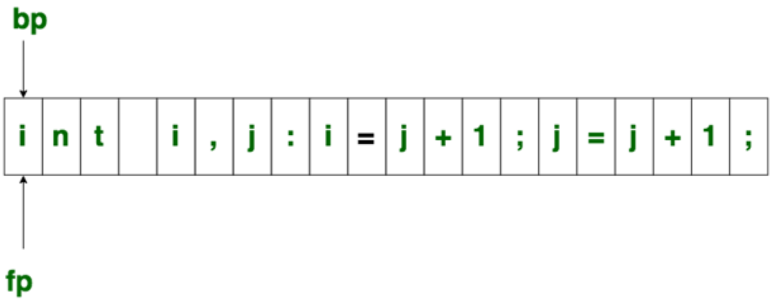
L' → ,SL'|ε

	First	Follow
S	(, a	$,)
L	(, a	)
L'	), ε	)

4. What is input buffering? “Buffer pair with sentinels optimizes a code by reducing the number of tests”- do you agree with the statement? Justify your answer accordingly with an example. 5

Answer:

- A buffer contains data that is stored for a short amount of time, typically in the computer's memory (RAM).
  - The purpose of a buffer is to hold data right before it is used.



Initial Configuration

Yes. The statement is true. Because, buffer pair contains the following code implements:

```
if (fwd at end of first half)
   reload second half;
   set fwd to point to beginning of second half;
else if (fwd at end of second half)
   reload first half;
   set fwd to point to beginning of first half;
else
   fwd++;
```

it takes two tests for each advance of the fwd pointer

Whereas, buffer pair with sentinels can optimize the above code as follows:

```
fwd++;
if ( *fwd == EOF )
{
  if (fwd at end of first half)
        . . .
  else if (fwd at end of second half)
        . . .
  else      /* end of input */
        terminate processing.
}
```

5. How to recover error using panic mode error recovery in LL(1) parser? Explain. 5

Answer:

Explain it with any relevant example. See below as an example:

E→TE'　　E'→+TE'|ε

T→FT'　　T'→*FT'|ε

F→(E)|id

Non-terminal	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'	***synch***	***synch***
E'		E'→+TE'			E'→ε	E'→ε
T	T →FT'	***synch***		T→FT'	***synch***	***synch***
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id	***synch***	***synch***	F→(E)	***synch***	***synch***