## Backus-Naur Form (BNF): First assignment

### Abstract

This assignment provided us with the opportunity to create BNF grammar descriptions for a given set of rules and also make parse trees for certain given sentences. This allowed us to understand how various languages work with a given set of syntax. The main objective of this assignment was to familiarize oneself with structuring BNF and to construct sentences based on the grammar. Here are solutions to the given set of problems.

Problem 1: Laughter

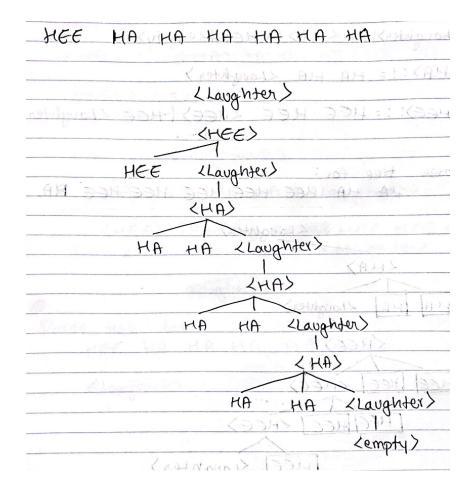
```
BNF Grammar:
```

```
<Laughter> ::= <HA> | <HEE> | <empty>
<HA> ::= HA HA <Laughter>
<HEE> ::= HEE HEE <HEE> | HEE <Laughter>
```

Parse tree for:

1. HA HA HEE HEE HEE HEE HEE HA HA

< Laughte	r>
C DUN PUBLY	AH AH
<b>LHA</b>	
1 < A142	
HA HA Laughters	
Cro Agun Me AH	<u>n-1</u>
(HEE)	
CIT STATE	
HEE HEE (HEE)	MIN
(R) LA LOUGHARY	
[HEE] HEE] KHEE]	>
(vtame)	
[HEE] <1	aughters
and the second second second second second second	
	(MA)
HA H	A (Laughter)
	Kempty>
	f $f$

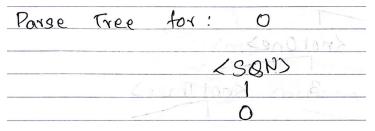


Problem 2: SQN(Special Quaternary Numbers)

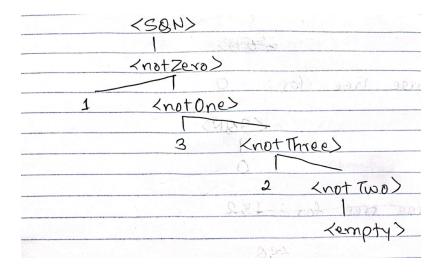
BNF Grammar:

```
<SQN> ::= 0 | <notZero>
<notZero> ::= 1 <notOne> | 2 <notTwo> | 3 <notThree> | <empty>
<notOne> ::= 0 <notZero> | 2 <notTwo> | 3 <notThree> | <empty>
<notTwo> ::= 0 <notZero> | 1 <notOne> | 3 <notThree> | <empty>
<notThree> ::= 0 <notZero> | 1 <notOne> | 2 <notTwo> | <empty>
```

1.0



### 2.132



#Question: Explain, in precise terms, why you cannot draw a parse tree, consistent with the BNF grammar that you crafted, for the string: 1223

In the given string, there are two adjacent occurrences of '2', which contradicts the syntax that this grammar is based on. If we give it a try, after occurrence of one '2', the grammar leads to <notTwo> that makes it impossible for us to get another '2' and this applies to other adjacent occurrences too. BNF Grammar:

<BXR> ::= #t | #f | <operators> <operators> ::= <and> | <or> | <not> <and> ::= ( and <boolean> ) | ( and ) <or> ::= ( or <boolean> ) | (or) <not> ::= ( not #t ) | ( not #f ) | ( not #t ) <boolean> | ( not #f ) <boolean> <boolean> ::= #t <boolean> | #f <boolean> | <operators> | <empty>

Parse trees for:

1. ( or #t )

Parse free for '. (or #t)
L'in Laraga > 1000 L
< BXR>
Mr. < brank 20
<operators></operators>
( Land Stocoled 1)
<01>
Le colorado -
( Or <boolean 1)<="" td=""></boolean>
< tons
[#t] zbooleans
KNO910002> ( +H HON D)
<empty></empty>
1600 17HI

# 2.( and ( not #t ) #f )

	Charleans is Ht < booleans 14+ <b< th=""></b<>
rowse	tree for : (and (not #+) #f)
	- Jose -
Realized	< B×P>
	Ranse here from ( or 4+)
	<operators></operators>
	LAXA>
	Kand >
	A DE ARA DE COM
	[ [and <booleans ]]<="" td=""></booleans>
	1<02
	<operators></operators>
	Classical Ind
	<not></not>
	Langelood And
	( not [#t]) <boolean></boolean>
	< Ptgmpty>
	[#f] <boolean></boolean>
	< empty>

Problem 4: LSS(Line Segment Sequences)

```
<LSS> ::= ( <distance> <angle> <color> | <empty>
<distance> ::= <num>
<angle> ::= <num>
<color> ::= BLUE ) <LSS> | BLACK ) <LSS> | RED ) <LSS>
```

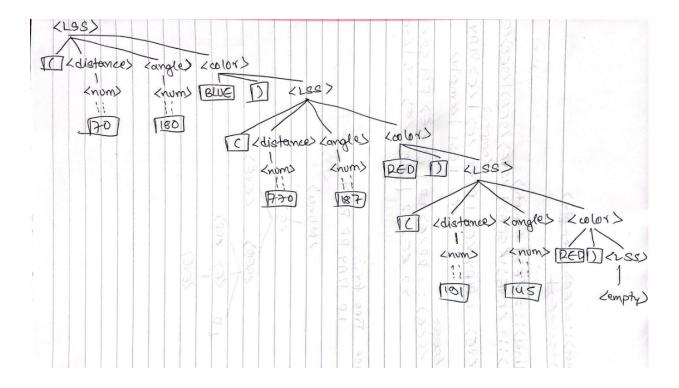
Parse tree for:

BNF Grammar

1. ( 120 95 BLACK )

Parse free for : 120 95 BLACK) Smotel < LSS <distance) <angle> 2 color> BLACK Lnum> LLSS> Lnum> Lem 120

2. ( 70 180 BLUE ) ( 770 187 RED ) ( 191 145 RED )



Problem 5: M-Lines

BNF Grammar

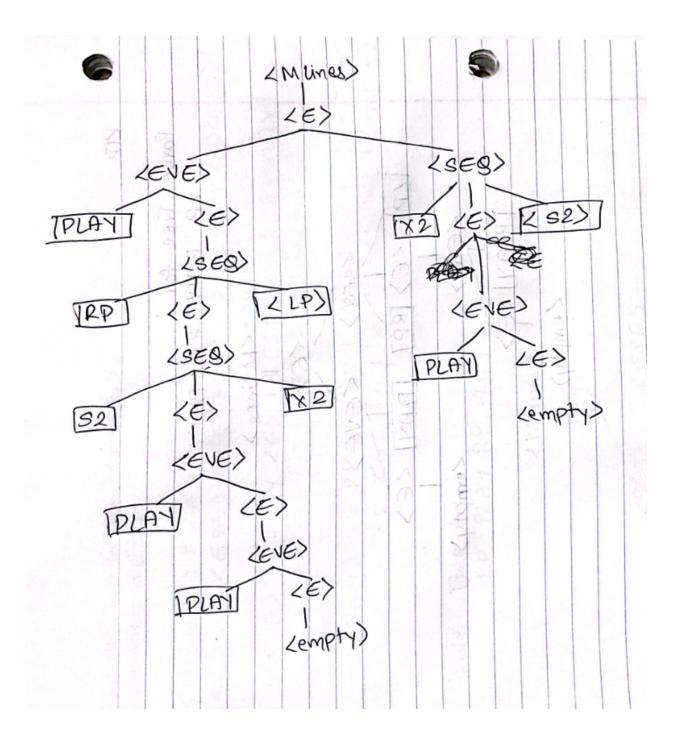
<Mlines> ::= <E> | <empty>
<E> ::= <EVE> | <SEQ> | <EVE> <SEQ> | <SEQ> <EVE> | <empty>
<EVE> ::= PLAY <E> | REST <E> | <E> PLAY | <E> REST | <empty>
<SEQ> ::= RP <E> LP | LP <E> RP | S2 <E> X2 | X2 <E> S2 | S3 <E>
X3 | <empty>

Parse tree for :

1. LP PLAY RP PLAY

Payse Tree for RP PLAY PLAN LP < M lines <E> <EVE> <SE9 <E> RP <E> 1P PAN <empty. <EVE> PLA emot





#### Problem 6: What is BNF?

BNF(Backus-Naur Form) is simply a mathematical way to define syntax of a programming language. It consists of specific symbols or strings, which are called terminals and non-terminals. We use those, along with a set of rules to create a set of grammar for a language. BNF has been used to develop various languages. The set of rules is written as such:

Left-Hand side ::= Right-Hand side

There can only be one non-terminal on the left-hand side but there can be multiple terminals or non-terminals on the right-hand side.