- Programming language grammars are employed to specify a programming language.
- An assembly language is programming language

Context free grammars:
A *context-free* grammar has four components:

1. A set of *tokens*, known as *terminal symbols.*
2. A set of *nonterminal symbols* or *nonterminals*.
3. A set of productions, or rules, where each production consists of a nonterminal symbol, called the *left side* or the production, an arrow, and a sequence of tokens and nonterminal symbols, called the *right side* of the production.
4. A designation of one of the nonterminal symbols as the *start* symbol.

Notation:

1. Terminal symbols are expressed in **bold** print.
2. Nonterminal symbols are *italicized*.

Example 1:
Write a grammar for an arbitrarily long expression consisting of single digs separated by either the plus sign or the minus sign.

	left side		*right side*
1	*list*	→	*list* **+** *digit*
2	*list*	→	*list* **-** *digit*
3	*list*	→	*digit*
4	*digit*	→	**0**
5	*digit*	→	**1**
6	*digit*	→	**2**
7	*digit*	→	**3**
8	*digit*	→	**4**
9	*digit*	→	**5**
10	*digit*	→	**6**
11	*digit*	→	**7**
12	*digit*	→	**8**
13	*digit*	→	**9**

Table 1. P, the set of productions

For this grammar,

1. T, the set of terminal symbols, called tokens, $T = \{+, -, \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}\}$
2. N, the set of nonterminal symbol, $N = \{list, digit\}$
3. P, the set of productions. Refer to Table 1.
4. The starting nonterminal symbol is *list*.

Example 2:

Write a grammar for arithmetic expressions.

	left side		right side
1	*expression*	→	*expression* **+** *term*
2	*expression*	→	*expression* **-** *term*
3	*expression*	→	*term*
4	*term*	→	*term* ***** *factor*
5	*term*	→	*term* **/** *factor*
6	*term*	→	*factor*
7	*factor*	→	**(** *expression* **)**
8	*factor*	→	**id**

Table 2. Set of productions for *expression*s

For this grammar,

1. T, the set of terminal symbols, called tokens, $T = \{+, -, *, /, (,),$ **id**$\}$
2. N, the set of nonterminal symbol, $N =$ {expression,term,factor}
3. P, the set of productions. Refer to Table 2.
4. The starting nonterminal symbol is *expression*.

Example 3:

Perform left-most derivation of the arithmetic expression $\mathbf{id_1+id_2*id_3}$

String of terminals and nonterminals	Rule used to reduce a handle of the string			
$\mathbf{id_1+id_2*id_3}$	Terminal string			
factor $\mathbf{+ id_2*id_3}$	8	*factor*	→	**id**
term $\mathbf{+ id_2*id_3}$	6	*term*	→	*factor*
expression $\mathbf{+ id_2*id_3}$	3	*expression*	→	*term*
expression **+** *factor* $\mathbf{*id_3}$	8	*factor*	→	**id**
expression **+** *term* $\mathbf{*id_3}$	6	*term*	→	*factor*
expression **+** *term* ***** *factor*	8	*factor*	→	**id**
expression **+** *term*	4	*term*	→	*term* ***** *factor*
expression	1	*expression*	→	*expression* **+** *term*

Grammar for MARIE

Rule	Left Side		Right Side
1	*program*	→	*statement-list*
2	*statement-List*	→	*statement*
3	*statement-List*	→	*statement-list statement*
4	*statement*	→	*directive*
5	*statement*	→	*labeled-item*
6	*statement*	→	*item*
7	*directive*	→	**org hexlit**
8	*labeled-item*	→	*label item*
9	*label*	→	**identifier ,**
10	*item*	→	*instruction*
11	*item*	→	*data-definition*
12	*instruction*	→	**JnS** *operand*
13	*instruction*	→	**Load** *operand*
14	*instruction*	→	**Store** *operand*
15	*instruction*	→	**Add** *operand*
16	*instruction*	→	**Subt** *operand*
17	*instruction*	→	**Input**
18	*instruction*	→	**Output**
19	*instruction*	→	**Halt**
20	*instruction*	→	**Skipcond** *operand*
21	*instruction*	→	**Jump** *operand*
22	*instruction*	→	**Clear**
23	*instruction*	→	**AddI** *operand*
24	*instruction*	→	**JumpI** *operand*
25	*instruction*	→	**LoadI** *operand*
26	*instruction*	→	**StoreI** *operand*
27	*instruction*	→	**END**
28	*operand*	→	**hexlit**
29	*operand*	→	**identifier**
30	*data-definition*	→	**HEX hexlit**
31	*data-definition*		**DEC hexlit**

Token	**Definition**
identifier	[a-zA-Z][a-zA-Z0-9]*
intlit	[1-9][0-9]*
hexlit	[0][0-9a-fA-F]*

Rule	**Left Side**		**Right Side**	**Action**
1	*program*	→	*statement-list*	Write the memory image to file *source*.mex.
2	*statement-list*	→	*statement*	
3	*statement-list*	→	*statement-list statement*	
4	*statement*	→	*directive*	
5	*statement*	→	*labeled-item*	
6	*statement*	→	*item*	
7	*directive*	→	**org hexlit**	• Convert the integer to 16-bit two's complement form. • Assign the integer to the initial address. • Assign the integer to the current address.
8	*labeled-item*	→	*label item*	
9	*label*	→	**identifier ,**	Define the identifier in the Label Table by assigning the current address to the identifier.
10	*item*	→	*instruction*	Increment the current address
11	*item*	→	*data-definition*	Increment the current address
12	*instruction*	→	**JnS** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **JnS** opcode (0000) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
13	*instruction*	→	**Load** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Load** opcode (0001) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
14	*instruction*	→	**Store** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Store** opcode (0010) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
15	*instruction*	→	**Add** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Add** opcode (0011) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.

Rule	Left Side		Right Side	Action
16	*instruction*	→	**Subt** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Subt** opcode (0100) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
17	*instruction*	→	**Input**	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Input** opcode (0101) to the operator field of the instruction. Assign the **0x000** to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
18	*instruction*	→	**Output**	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Output** opcode (0110) to the operator field of the instruction. Assign the **0x000** to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
19	*instruction*	→	**Halt**	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Halt** opcode (0111) to the operator field of the instruction. Assign the **0x000** to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
20	*instruction*	→	**Skipcond** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Skipcond** opcode (1000) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
21	*instruction*	→	**Jump** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Jump** opcode (1001) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
22	*instruction*	→	**Clear**	Create 16-bit instruction. Bits 15-12 are operator. Assign the **Clear** opcode (1010) to the operator field of the instruction. Assign the **0x000** to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.

Rule	Left Side		Right Side	Action
23	*instruction*	→	**AddI** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **AddI** opcode (1011) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
24	*instruction*	→	**JumpI** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **JumpI** opcode (1100) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
25	*instruction*	→	**LoadI** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **LoadI** opcode (1101) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
26	*instruction*	→	**StoreI** *operand*	Create 16-bit instruction. Bits 15-12 are operator. Assign the **StoreI** opcode (1110) to the operator field of the instruction. Assign the *operand* to the remaining 12 bits of the instruction, bits 11-0. Assign the instruction to the current address in memory.
27	*Instruction*	→	**END**	
28	*operand*	→	**hexlit**	Convert the **hexlit** to a 16-bit two's complement integer.
29	*operand*	→	**identifier**	Reference the identifier in the Label Table by assigning the current address to the identifier's reference list.
30	*data-definition*	→	**hex hexlit**	Convert the **hexlit** to a 16-bit two's complement integer.
31	*data-definition*	→	**dec intlit**	Convert the **intlit** to a 16-bit two's complement integer.