TIKRIT UNIVERSITY
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS
DEPARTMENT OF COMPUTER SCIENCE

SUBJECT OF COMPILER1

DATE OF ISSUE: 2024 - 2025

CLASS: 3TH STAGE

SEMESTER 1

LECTURE NO. : 4

PREPARED BY

Lecturer:
Mohanad Dawood Al-Roomi

&

Assistant Lecturer:
Luay Ibrahim Klalif

Writing a grammar

(2)

1. * zero or more instances.

L* is the set of all strings of letters, including ε the empty string.

L (L U D)* is the set of all strings of letters and digits beginning with a letter.

2. + One or more instances.

$r^* = r^+ \mid \varepsilon$ and $r^+ = rr^* = r^*r$

+ and * operators has the same precedence and associativity.

D^+ is the set of all strings of one or more digits.

3. ? Zero or one instance.

r? is equivalent to r | εzero or one occurrence

L(r?) = L(r) U {ε}.

? operator has the same precedence and associativity.

4. L^4 is the set of all 4-letter strings.

Extensions of Regular Expressions?

Expression	Matches	Example
^	beginning of a line	^abc
$	end of a line	abc$
[s]	any one of the characters in string s	[abc]
[^s]	any one character not in string s	[^abc]
r*	zero or more strings matching r	a*
r^+	one or more strings matching r	a+
r?	zero or one r	a?
r {m , n}	between m and n occurrences of r	a{1 , 5}
r_1 r_2	an r_1 followed by an r_2	ab
r_1 | r_2	an r_1 or an r_2	a | b
(r)	same as r	(a | b)
r_1 / r_2	r_1 when followed by r_2	abc / 123
[A-Z]	Known sequence	A|…|Z
[0-9]	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |9	0 |…|9

❑ Regular Expressions: التعابير المنتظمة

are commonly used to describe patterns, The regular expressions are built recursively out of smaller regular expressions. They are built from single characters, using union , concatenation, and the Kleene closure and positive closure.

❑ Precedence rules of Regular Expressions : قواعد الأسبقية للتعابير المنتظمة

regular expression r ; language *L*(r) ; recursively subexpressions r's ; alphabet Σ

a. The unary operator * has highest precedence and is left associative. المرفوع لقوه معينه يحل أولاً

b. Concatenation has second highest precedence and is left associative. التتابع يحل ثانياً

c. | has lowest precedence and is left associative. علامة أو تكون الأخيرة

Ex: (a)|((b)*(c)) by a|b*c = a|c = a|bc = a|bbc = a|bbbc = …

Both expressions denote the set of strings that are either a single a or are zero or more b's followed by one c.

Example: Let Σ = {a , b}.

1. The regular expression a | b denotes the language {a , b}.
2. (a|b)(a|b) denotes {aa; ab; ba; bb}, the language of all strings of length two over the alphabet Σ. Another regular expression for the same language is aa|ab|ba|bb.
3. a* denotes the language consisting of all strings of zero or more a's, that is, {ε, a, aa, aaa, …}.
4. (a|b)* denotes the set of all strings consisting of zero or more instances of a or b, that is, all strings of a's and b's: { ε, a, b, aa, ab, ba, bb, aaa, …}. Another regular expression for the same language is (a*b*)*.
5. a|a*b denotes the language {a, b, ab, aab, aaab, …}, that is, the string a and all strings consisting of zero or more a's and ending in b.

❑ Algebraic laws of Regular Expressions : القوانين الجبر الرياضي للتعابير المنتظمة

Figure shows some of the algebraic laws that hold for arbitrary regular expressions r, s, and t.

Sq.	LAW	DESCRIPTION
(1)	r|s = s|r	| is commutative تَبادُلِيّ
(2)	r|(s|t) = (r|s)|t = r|s|t	| is associative ترابطي
(3)	r(st) = (rs)t = rst	Concatenation is associative التتابع هو ترابطي
(4)	r(s|t) = rs|rt ; (s|t)r = sr|tr	Concatenation distributes over | التتابع يوزع اكثر
(5)	rε = εr = r	ε is the identity for concatenation هي للتتابع Empty
(6)	r* = (r| ε)* = ε | r | rr |…	ε is guaranteed in a closure
(7)	r** = r*	* is idempotent تدل على عنصر من مجموعة لم يتغير في القيمة عند ضربه أو تشغيله بنفسه.

❑ Regular Definitions

is the patterns that describe the tokens of a Complex collections of programming language and is a sequence of statements that each define one variable to stand for some regular expression.

$$
\begin{aligned}
d_1 &\longrightarrow r_1\\
d_2 &\longrightarrow r_2\\
\cdots &\longrightarrow \cdots\\
d_n &\longrightarrow r_n
\end{aligned}
$$

Example: C identifiers are strings of letters, digits, and underscores.

Letter_ ⟶ A | B | … | Z | a | b | … | z | _

Digit ⟶ 0 | 1 | … | 9

Identifier ⟶ Letter_ (letter_ | digit)*

Context-Free Grammar (Definition of Grammars)

a context-free grammar (has four components) consists of :

1. **Terminals are the basic symbols from which strings are formed. Ex: the terminals are the keywords if and else and the symbols "(" and ")". A set of terminal symbols, sometimes referred to as "tokens". The terminals are the elementary symbols of the language defined by the grammar.**

2. **Nonterminals are syntactic variables that denote sets of strings. They help define the language generated by the grammar. Nonterminals impose a hierarchical structure on the language that is key to syntax analysis and translation. Ex: stmt and expr are nonterminals. A set of nonterminals, sometimes called "syntactic variables". Each nonterminal represents a set of strings of terminals, in a manner we shall describe.**

3. **In a grammar, one nonterminal is distinguished as the start symbol, and the set of strings it denotes is the language generated by the grammar. A designation of one of the nonterminals as the start symbol.**

4. **The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings. A set of productions, where each production consists of a nonterminal, called the head or left side of the production, an arrow, and a sequence of terminals and/or nonterminals, called the body or right side of the production. The intuitive intent of a production is to specify one of the written forms of a construct; if the head nonterminal represents a construct, then the body represents a written form of the construct.**

❑ The Formal Definition of a Context-Free Grammar

Each production consists of:

(a) A nonterminal called the head or left side of the production; this production defines some of the strings denoted by the head.

(b) The symbol → Sometimes ::= has been used in place of the arrow.

(c) A body or right side consisting of zero or more terminals and non-terminals. The components of the body describe one way in which strings of the nonterminal at the head can be constructed.

❑ Notational Conventions (اتفاقيات التدوين أو التشكيلي)

1. These symbols are terminals:

(a) Lowercase letters early in the alphabet, such as a, b, c.

(b) Operator symbols such as +, *, and so on.

(c) Punctuation symbols such as parentheses (, comma , , and so on.

(d) The digits 0, 1, … , 9.

(e) Boldface strings such as id or if, each of which represents a single terminal symbol.

Notational Conventions (اتفاقيات التدوين أو التشكيلي)

2. These symbols are nonterminals:

(a) Uppercase letters early in the alphabet, such as A, B, C.

(b) The letter S, which, when it appears, is usually the start symbol.

(c) Lowercase, italic names such as *expr* or *stmt*.

(d) When discussing programming constructs, uppercase letters may be used to represent nonterminals for the constructs. For example, non-terminals for *expressions*, *terms*, and *factors* are often represented by *E*, *T*, and *F*, respectively.

3. Uppercase letters late in the alphabet, such as X, Y, Z, represent grammar symbols; that is, either nonterminals or terminals.

4. Lowercase letters late in the alphabet, chiefly u, v, ... , z, represent (possibly empty) strings of terminals.

5. Lowercase Greek letters, α, β, γ, for example, represent (possibly empty) strings of grammar symbols. Thus, a generic production can be written as A → α , where A is the head and α the body.

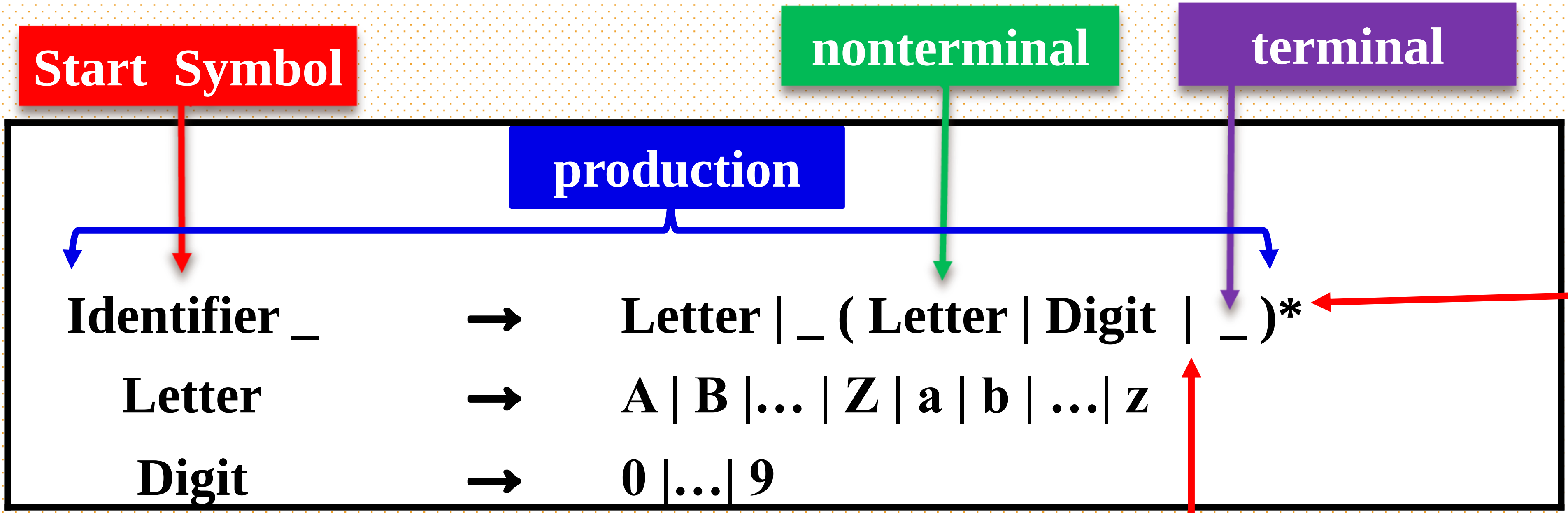
6. A set of productions A → α1, A→ α2, ..., A→ αk with a common head A (call them A-productions), may be written A → α1 |α2|... |αk. Call α1, α2, ..., αk the alternatives for A.

7. Unless stated otherwise, the head of the first production is the start symbol.

Regular definition of Id (Grammars)

❑ C, C++ identifiers are strings of letters, digits, and underscores.

ملاحظة: المخطط الانتقالي وبرمجته للمعرف سوف تكون في محاضرة العملي.



Identifier _ → Letter | _ (Letter | Digit | _)*

Letter → A | B |… | Z | a | b | …| z

Digit → 0 |…| 9

*رمز النجمة تشير ان ما داخل القوس يمكن ان يتكرر كثيرا او صفر من المرات.

رمز | تشير الى معنى أو OR

Shorthand يمكن ان نكتبه بهذه الصيغة

*letter_* → [A-Z a-z_]

digit → [0-9]

id → *letter_ (letter_ | digit)**

Shorthand يمكن ان نكتبه بهذه الصيغة

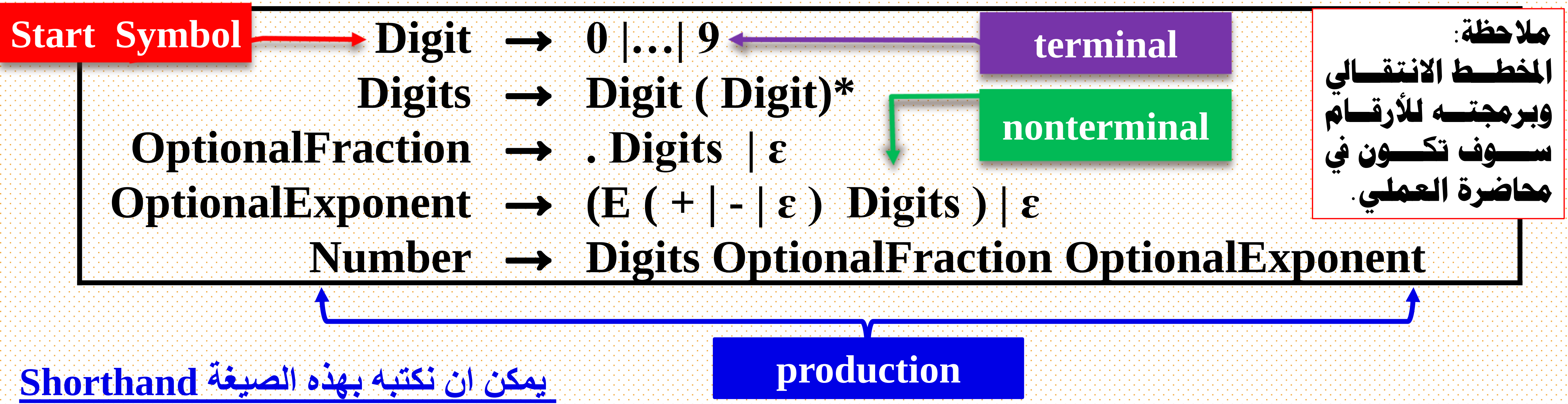
L_ → [A-Z a-z_]

D → [0-9]

Id → L_ (L _ | D)*

Ex: Identifier is accepted	Ex: Identifier is rejected
x	@x
X1	1X
Student	Stu.dent
mark_3	Mark+3
_	_!
Paracetamol_2020	Paracetamol_2020#

Regular definition of unsigned number (Grammars)



Start Symbol

Digit → 0 |…| 9

Digits → Digit (Digit)*

OptionalFraction → . Digits | ε

OptionalExponent → (E (+ | - | ε) Digits) | ε

Number → Digits OptionalFraction OptionalExponent

terminal

nonterminal

production

ملاحظة: المخطط الانتقالي وبرمجته للأرقام سوف تكون في محاضرة العملي.

Shorthand يمكن ان نكتبه بهذه الصيغة

D → 0 |…| 9

DS → DD*

Num → DS (. DS)? (E [+-]? DS)?

Shorthand يمكن ان نكتبه بهذه الصيغة

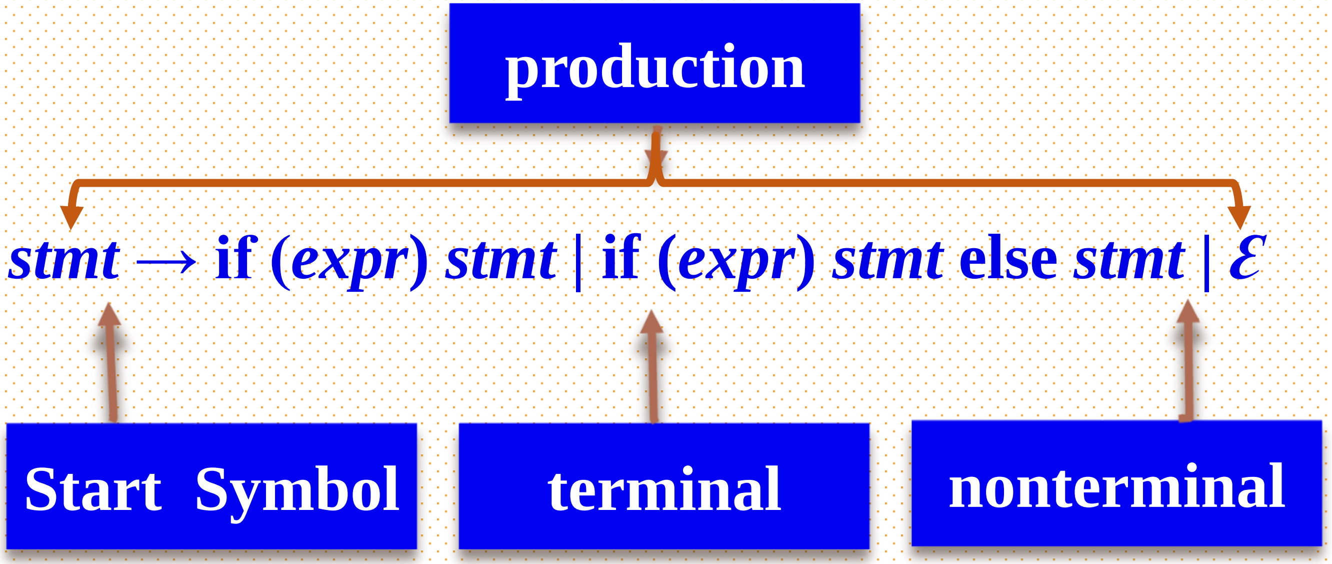
D → 0 |…| 9

DS → D^+

Num → DS (. DS)? (E [+-]? DS)?

Ex: Cases of accepted	Ex: Cases of rejected
7	7w
93.5	.45
12E+4	E+12
77.3E-10	77.E-10
0.0	9.5E+2E-10
Etc.	9.5.2

The Formal Definition of a Context-Free Grammar



stmt → if (*expr*) *stmt* | if (*expr*) *stmt* else *stmt* | *ℰ*

ملاحظة:

الكرامر الذي كتبت به تعريف if سوف يولد مشكلة التداخل من اليسار في المثال رقم 2 و 3 و 4 لذلك يجب ان تكتب بالصيغة التالية:

stmt → if (*expr*) *stmt* [else *stmt*]? | *ℰ*

Ex1:

```
if ( x >= y )
    x = 10 ;
```

Ex2:

```
if ( x >= y )
    x = 10 ;
else
    x = 5 ;
```

Ex3:

```
if ( x >= y )
    if ( z == 4 )
        x = 10 ;
        else
            x = 5 ;
```

الــ else في المثال الثالث المترجم سوف يعتبرها تابعة لأقرب (if)

Ex4:

```
if ( x >= y )
    x = 1 ;
   else if ( z == 4 )
       x = 10 ;
       else if ( s < 3 )
           x = 5 ;
             else x = 0 ;
```

The Formal Definition of a Context-Free Grammar

Example: The grammar in defines simple arithmetic expressions. In this grammar, the **terminal** symbols are id + - * / ( ) , The **nonterminal** symbols are *expression*, *term* and *factor*, and *expression* is the **start symbol**

expression → *expression + term*
expression → *expression - term*
expression → *term*
term → *term * factor*
term → *term / factor*
term → *factor*
factor → *(expression)*
factor → **Id**
factor → **number**

Ex: Cases of accepted	**Ex: Cases of rejected**
X + Y	**X * + Y**
Z – 5 * 6	**Z (– 5 * 6**
4 * 6 / 8.5 – 10	**4 * 6 / 8.5 - X 10**
X	
8	

❑ **Shorthand يمكن ان نكتبه بهذه الصيغة**

Expression → Operand (Operator Operand)*
Operand → Id | Num
Operator → + | - | * | /

✓ **Operator (عامل رياضيات أو العملية الرياضية)**
Ex: + , - , * , / , Etc.

✓ **operand (المعامل) (المعرف أو الرقم)**
Ex: x , y , mark_1, 1 , 2.5 , 7.4E+3 , Etc.

The Formal Definition of a Context-Free Grammar

Statement → if (Expression) Statement (else Statement)?

Expression → Operand (Operator Operand)*

Operand → Id | Num

Operator → + | - | * | /

Id_ → Letter | _ (Letter | Digit | _)*

Letter → A | B |… | Z | a | b | …| z

Digit → 0 |…| 9

Digits → Digit (Digit)*

OptionalFraction → . Digits | ε

OptionalExponent → (E (+ | - | ε) Digits) | ε

Num → Digits OptionalFraction OptionalExponent

Statement → Statement – if | Statement – Assignment | Statement – while | Statement – do_while; | for – Statement | … Etc

Statement – if → if (Expression) Statement (else Statement)?

Statement – Assignment → id = Expression ;

Statement – while → while (Expression) Statement

Statement – do_while; → do Statement while (Expression);

Statement - for → for (Expression ; Expression ; Expression) Statement

THANK YOU