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. : 2

PREPARED BY

Lecturer:
Mohanad Dawood Al-Roomi

&

Assistant Lecturer:
Luay Ibrahim Klalif

Language-processing system (Compilation):

- **WHAT IS THE PROBLEM?**
- A computer's CPU can understand and execute directly language written software of Machine language (0 , 1) only.
- Computer programs implemented in programming languages with the help of software programs called Language - processing system. (compilation).
- In general, a compiler as a single box that converted a source program into a semantically equivalent target program.

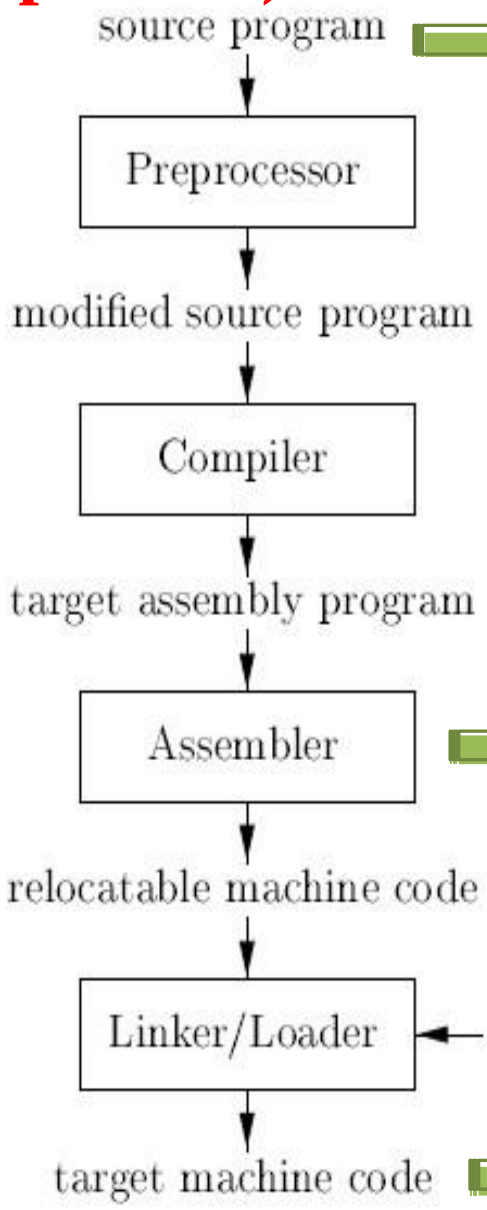


Figure 1.5: A language-processing system

العملية	وصف العملية
Z= X+ Y	أجمع محتوى المتغير X مع محتوى المتغير Y ووضع الناتج في محتوى المتغير Z

العملية	وصف العملية بلغة التجميع (ثلاثي العنوان)
LDA R1, id1	نقل محتوى الموقع id1 الى المسجل رقم R1
ADD R1, R1, id2	جمع محتوى المسجل R1 مع محتوى الموقع id2 والنتيجة في محتوى المسجل R1
ST id3, R1	خزن محتوى المسجل R1 في محتوى موقع id3

شفره العملية		عنوان في الذاكرة			
1100	1110	0000	0000	0001	0001
0001	0000	1101	0100	0000	0000
0010	0000	0010	0011	0000	0000

Language-processing system (Compilation): (3)

It is several programs may be required to create an executable target program are:

A preprocessor:

is a program that processes source language to produce a modified source program that is used as input to another program. The task of collecting the source program is sometimes entrusted to a separate program.(File Inclusion.) The preprocessor may also expand shorthand's, called macros, into source language statements.(Macro processing.)

A compiler

The modified source program is then fed to a compiler. The compiler may produce an assembly-language program as its output.

An assembler

The assembly language is then processed by a program called an assembler that produces relocatable machine code as its output.

A Linker / Loader:

Large programs are often compiled in pieces, so the relocatable machine code may have to be linked together with other relocatable object files and library files into the code that actually runs on the machine. The linker resolves external memory addresses, where the code in one file may refer to a location in another file. The loader then puts together all of the executable object files into memory for execution.

PHASES OF COMPILER

The following are the phases of the compiler:

Main phases:

1. Lexical analysis phase.((طور تحليل المفردات (الفقرات)
2. Syntax analysis phase.((طور تحليل القواعد اللغوية (العالقات اللغوية)
3. Semantic analysis phase. ((طور تحليل المعاني (الدلالي)

Analysis

4. Intermediate code generation phase. (طور توليد الشفرة الوسطية)
5. Code optimization phase. (طور تحسين الشفرة)
6. Code generation phase. (طور توليد الشفرة)

Synthesis

Sub-Phases:

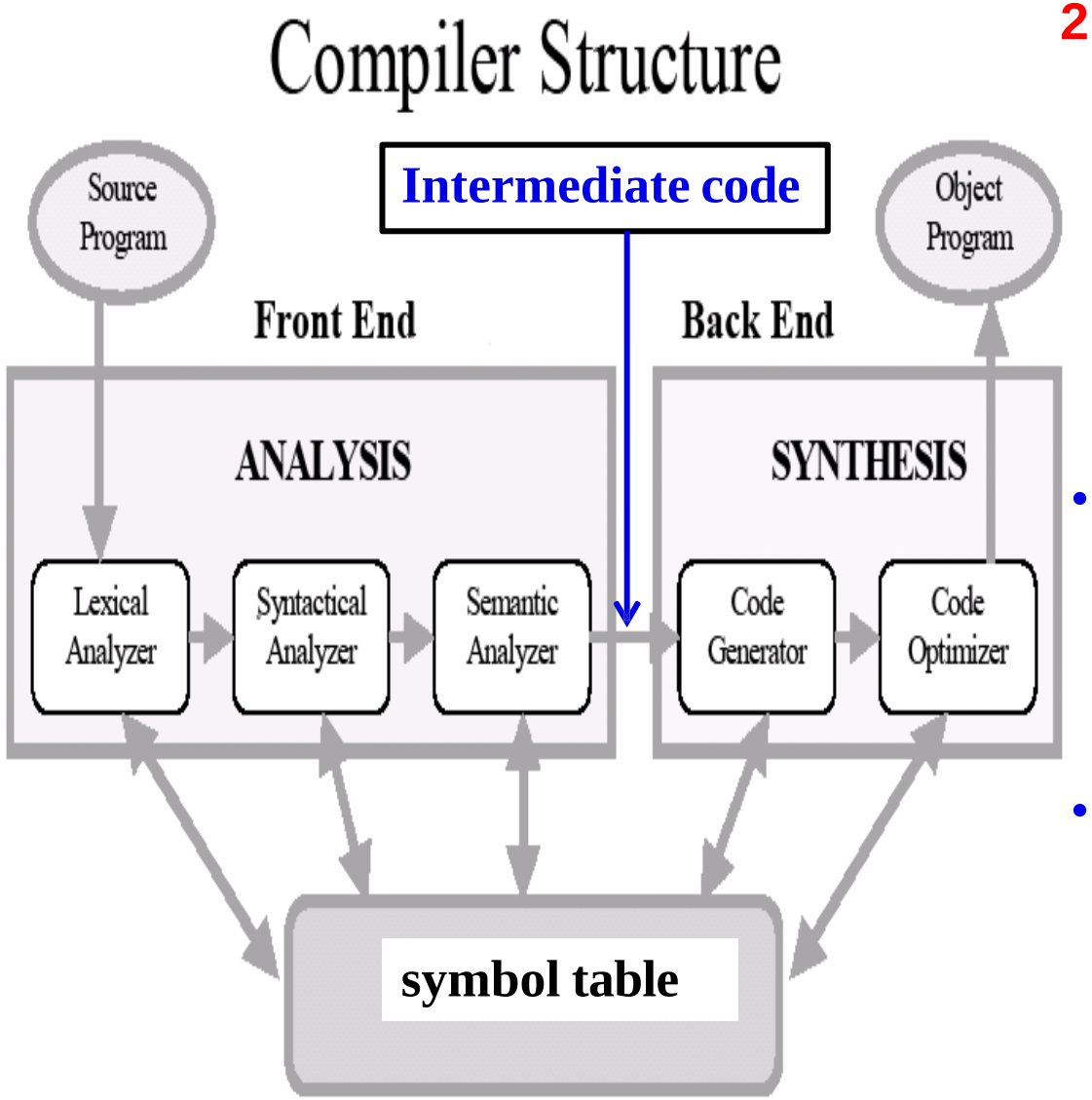
1. Symbol table management.(ادارة جدول الرموز)
2. Error handling. (معالجة االخطاء)

The Structure of a Compiler

There are two parts to this to the compiler:

1. Analysis: التحليل

- The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them.
- It then uses this structure to create an intermediate representation of the source program.
- It collects information about the source program and stores it in a data structure called a symbol table.
- detects some errors (either syntactically ill formed or semantically unsound) in the source program.

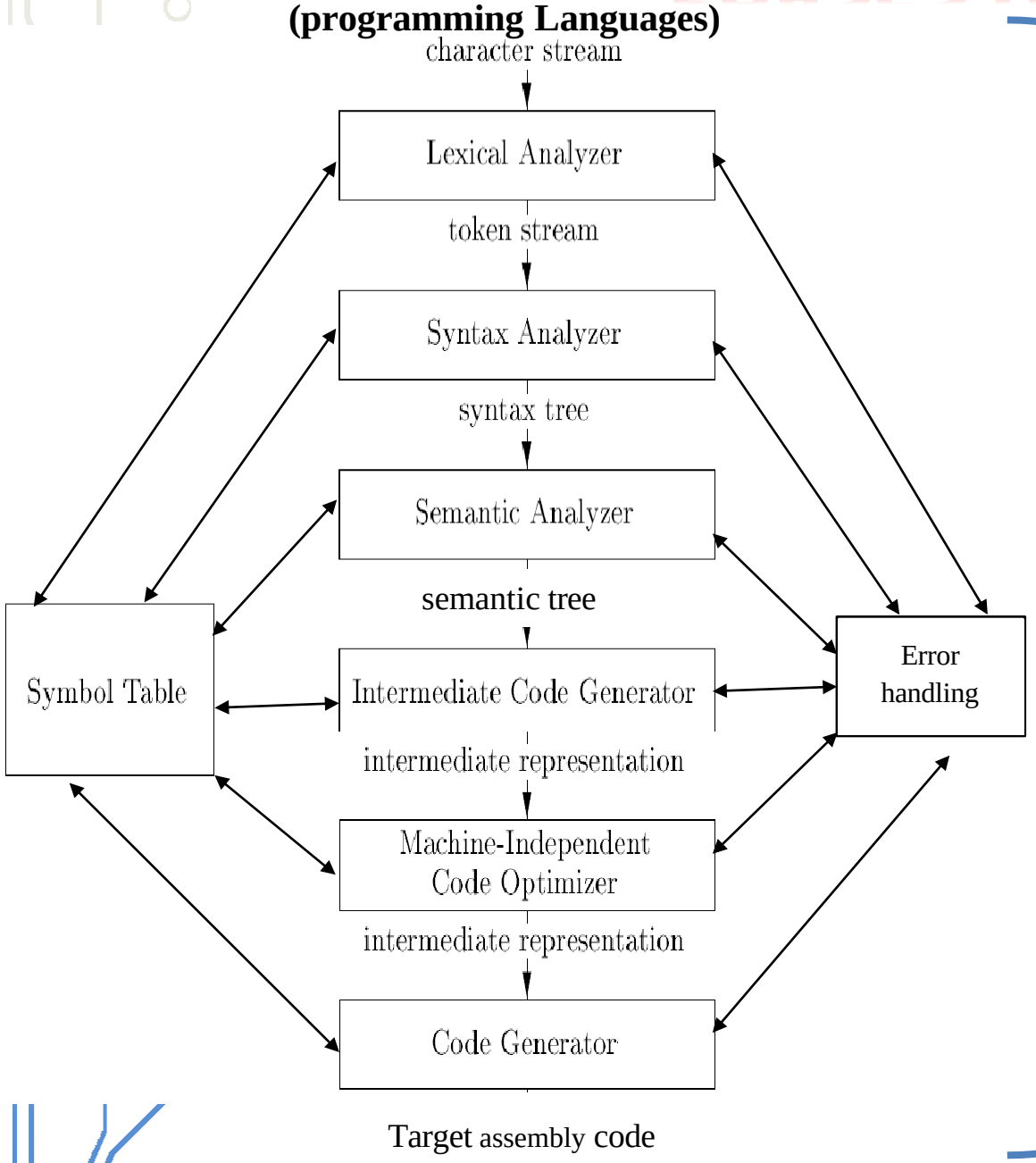


2. Synthesis: التوليف (التركيب)

The synthesis part constructs the desired target program from the intermediate representation and the information in the symbol table.

- The symbol table, which stores information about the entire source program, is used by all phases of the compiler.
- The analysis part is often called the front end of the compiler; the synthesis part is the back end.

PHASES OF COMPILER



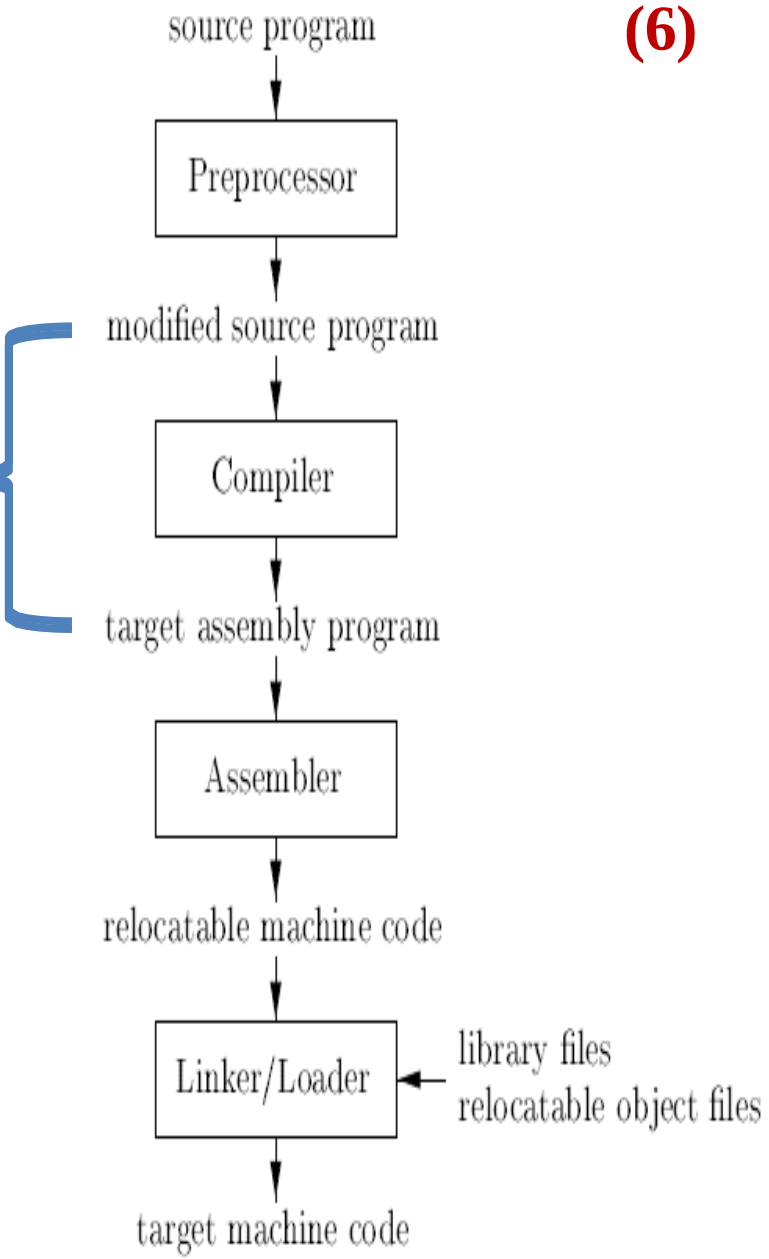


Figure 1.5: A language-processing system

Lexical Analyzer phase:

Software system and first phase of the compiler. It reads and scans the input characters (ASCII) making up the source program and groups the characters into meaningful sequences called lexemes and it produces as output tokens table (token-name; attribute-value), symbol table and tokens stream. It sends the tokens stream to the parser. This phase identifies errors according to a misspellings of lexeme.

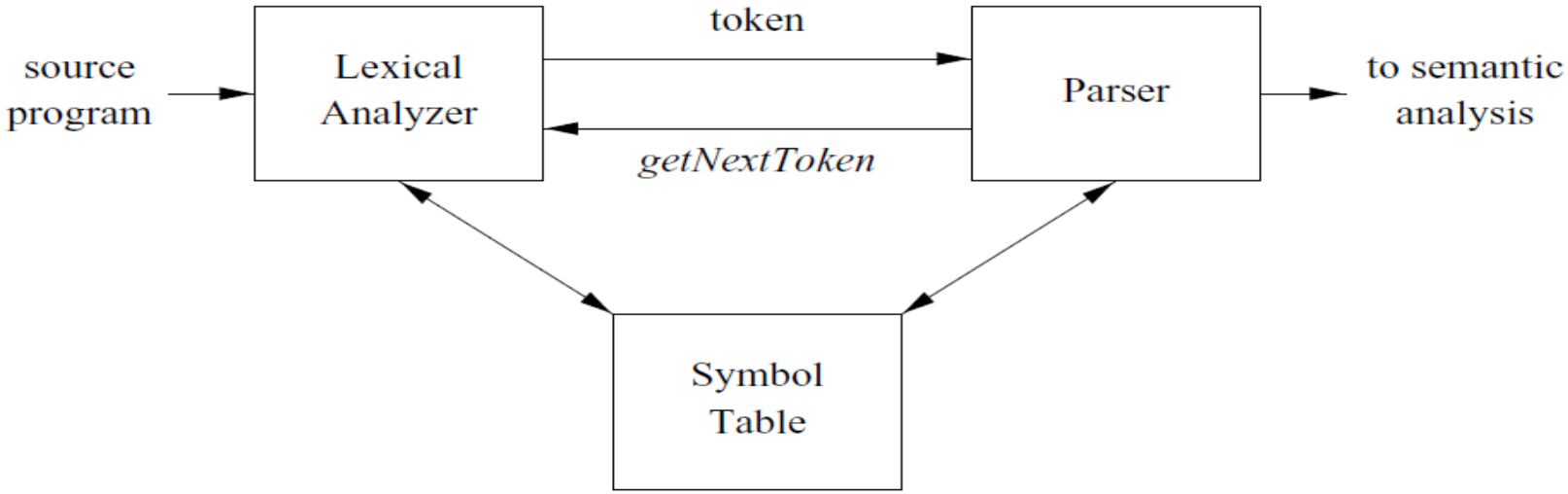


Figure 3.1: Interactions between the lexical analyzer and the parser

The Role of the Lexical Analyzer

(1) The main task of the lexical analyzer is **to read** the input characters of the source program, **group them** into lexemes, and **produce** as output a sequence of tokens for each lexeme in the source program. The stream of tokens is **sent to** the parser for syntax analysis.

(2) When the lexical analyzer **discovers** a lexeme constituting an identifier, it **needs to enter** that lexeme into the symbol table.

(3) There are **suggested** interactions between the lexical analyzer and the symbol table. When, the parser call the lexical analyzer to suggest by the get **NextToken** command, causes the lexical analyzer to read characters from its input until it can identify the next lexeme and produce for it the next token, which it returns to the parser.

(4) One such task is stripping out comments and whitespace (blank, newline, tab, and perhaps other characters that are used to separate tokens in the input).

(5) Another task is correlating error messages generated by the compiler with the source program. For example, the lexical analyzer may keep track of the number of newline characters seen, so it can associate a line number with each error message.

(6) If the source program uses a macro-preprocessor, the expansion of macros may also be performed by the lexical analyzer.

The Role of the Lexical Analyzer

Sometimes, lexical analyzers are divided into a cascade of two processes:

(a) Scanning consists of the simple processes that do not require tokenization of the input, such as deletion of comments and compaction of consecutive whitespace characters into one.

(b) Lexical analysis proper is the more complex portion, which produces tokens from the output of the scanner.

The Role of Lexical analyzer phase

نفترض أن لدينا برنامج المصدر (source program) المكتوب بواسطة لغة C ++. كيف يتم تحليل البرنامج المصدر من قبل طور تحليل الفقرات (Lexical analyzer) لتمريرها إلى طور تحليل القواعد اللغوية (المعرب) ((Syntax analysis phase (parser)؟ وكيف يتم تصحيح أخطاء طور تحليل الفقرات أن وجدت قبل تمريرها الى طور المعرب.

عندما نقوم بالضغط على مفتاح تنفيذ البرنامج (Run) بمعنى اننا نقوم بالطلب من المترجم (Compiler) بترجمة البرنامج المصدري والمكتوب بلغة C++ وبما ان طور تحليل الفقرات -المفردات- (Lexical analyzer) هو اول أطوار المترجم (Compiler) والذي يقوم بصورة عامه بعملية المسح والقراءة لـ(source program) وتحويله الى (token streams) وهي كما يلي:

١. يقوم بعملية بمسح وقراءة رمز بعد رمز (Character by Character) للبرنامج المصدري (والمكتوب بشكل رموز شفرة (ASCII)) بأكمله وخزنه في ذاكرة مستقلة كما هو مكتوب دون تغيير أو حذف او تعديل للاستفادة منه في تحديد موقع الخطأ اذا تطلب ذلك (It can associate a line number with each error message).

٢. ضغط الفراغات المتسلسلة في فراغ واحد (Compaction of consecutive whitespace characters into one).

The Role of Lexical analyzer phase

٣. حذف التعليقات (Deleting comments) وكما يلي:

A. يهمل طور تحليل الفقرات الرمزين المتتاليين // وما بعدهما من جملة التعليقات الى نهاية السطر لا يقوم بتحليل أو تقطيع رموزهما ولكن تبقى مخزونة في البرنامج المصدري الاصلي كما هو مكتوب.

```
// this is a single-line comment
```

B. لكن اذا كان اكثر من سطر نستخدم

```
/* this is a
multi-line
comment */
```

٤. ثم يقوم بتقطيع وتجميع (characters) المحصورة بين كل فراغين (Two blank spaces) ومن اليسار الى اليمين مع بعضها وانتاج Lexeme (وهو اصغر مفردة في اللغة ما ربما يتكون من رمز شفرة ASCII واحد او اكثر). وهكذا يعمل لجميع رموز برنامج المصدر.

٥. تعطى جميع (Lexemes) في البداية نوع بياني واحد هو characters لذلك المشكلة تكمن في كيفية تحديد قيمة هذه الـــ(lexemes) وجعل المترجم يتعرف على قيمتها البيانية.

٦. يرسل كل الـــ(lexeme) الى A macro-preprocessor ليقوم بالتحقق من قيمتها وانتاج (Token) له. حسب قواعد المايكروية (برامج مخزنه مسبقاً وفق قواعد اللغة المراد العمل في مجالها) التي تتعرف عليها وخزنها في (Token table). كل Token يتكون من (token- name) & (token- attribute).

The Role of Lexical analyzer phase

Table of operation description		
Operation	Description	Meaning
+	OP_ADD	addition
-	OP_SUB	subtraction
*	OP_MUL	multiplication
/	OP_DIV	division
=	Op_ASSIGN	assignment
<	OP_LT	less than
>	Op_GT	greater than
<=	OP_LE	less or equal
>=	OP_GE	greater or equal
==	OP_EQ	equal
!=	Op_NOT EQ	not equal
&&	OP_AND	and
||	OP_OR	or
!	OP_NOT	not

8. انواع القواعد المايكروية:

A. القاعدة المايكروية المسؤولة عن تحديد كلمات المفاتيح Keywords.

مثل if ، main ،for ، while ...

B. القاعدة المايكروية المسؤولة عن تحديد المعرفات Identifiers.

مثل x ، y ، x1 ، line1 ...

C. القاعدة المايكروية المسؤولة عن تحديد الارقام Number.

مثل 3 , 3.5 , 3.542 , 72.4E10 , 10.55E+6 , 836E-4

D. القاعدة المايكروية المسؤولة عن تحديد العوامل Operators.

i. العوامل الحسابية + ، - ، * ، /.

ii. العوامل الارتباط < ، > ، = =، <= ، >= ، != .

iii. العوامل المنطقية And (&&) ، OR(||) ، Not (!).

iv. علامة الاحلال = Assignment.

v. العوامل المركبة += , -= , *= , /= , %= , >= , >= , &= , |= , -- , ++

E. القاعدة المايكروية المسؤولة عن تحديد الفواصل والتنقيط Punctuations.

مثل (،) ، { ، } ، , ، ; ، . ، : ، ' ، [،] ، "

F. القاعدة المايكروية المسؤولة عن تحديد الحرفي Literal (أي شيء محصور بين علامتي الاقتباس الثنائية " ").

مثل "Student".

H.W.4.3. The macro-preprocessor

< Keyword> قاعدة مايكروية تختبر الكلمات المحجوزة

< Identifier> قاعدة مايكروية تختبرالمعرف

<Number> قاعدة مايكروية تختبر الرقم يصورة عامة

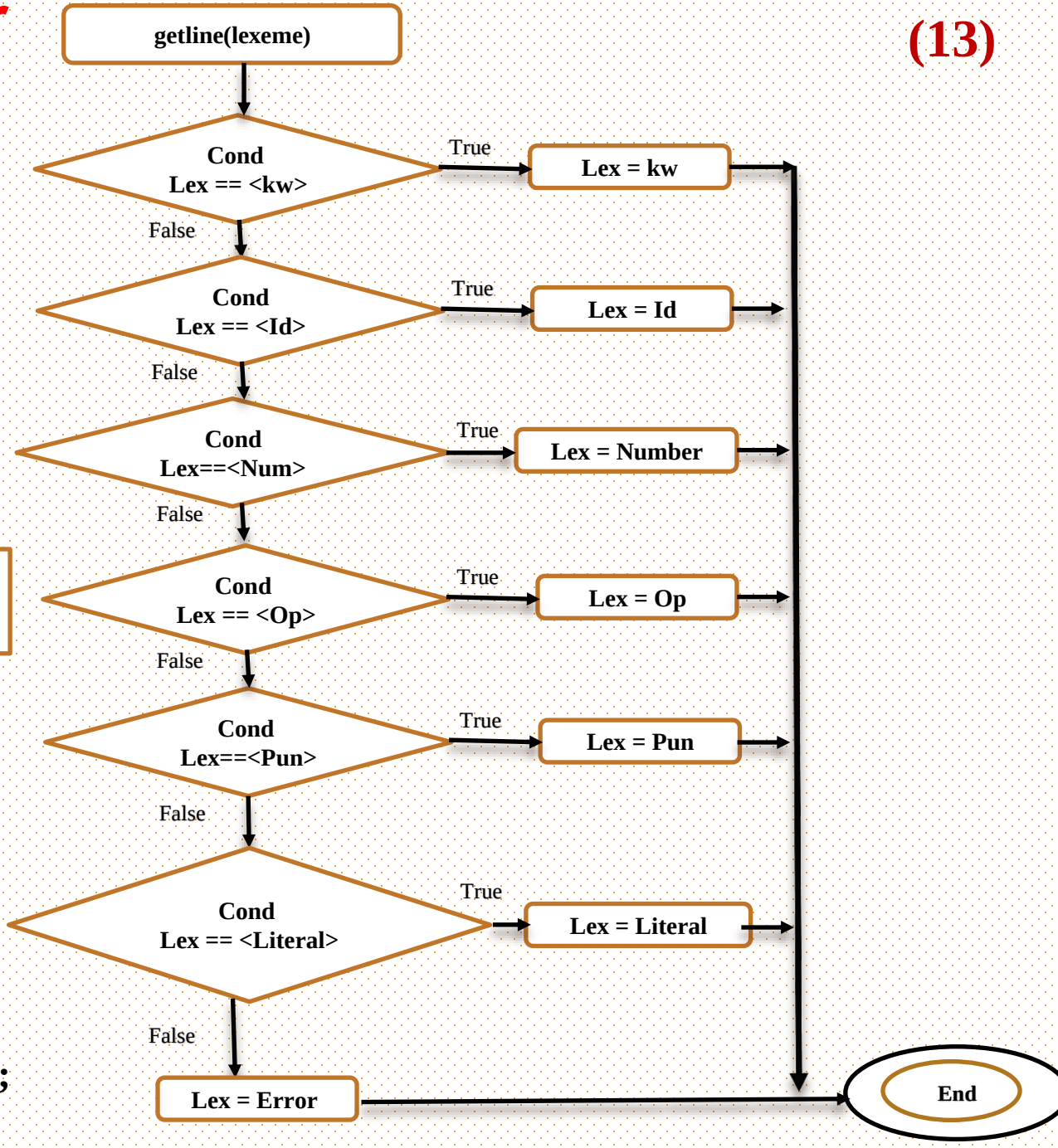
<Operator> قاعدة مايكروية تختبر العمليات الحسابية

<Punctuation> قاعدة مايكروية تختبر الفاصلة والتنقيط

< Literal > قاعدة مايكروية تختبر الجملة بين علامتي التنصيص

التسلسل (K.W. وبعده ID) هنا مطلوب

```
Main (Algorithm)
getline(lexeme);
 if (lexeme == < Keyword>) then lexeme = K.W;
    else if (lexeme == < Identifier>) then lexeme = Id;
       else if (lexeme == <Number>) then lexeme = number;
          else if (lexeme == <Operator>) then lexeme = Op;
             else if (lexeme == <Punctuation>) then lexeme = Pun;
                else if (lexeme == < Literal >) then lexeme = Literal;
                   else lexeme = error;
```



The Role of Lexical analyzer phase

٩. يقوم كذلك بأنشاء جدول الرموز (symbol table) وهو هيكل بياني يحتوي على التسلسل (Sq.) و(Token- name) و(Types). والذي يضع فيه الـــــ(Lexeme) عندما يجد بأن الـــــ(Lexeme) يشير الى معرف (Id) فانه يقوم بخزن اسم النموذج (Token- name) في جدول الرموز (Symbol Table)، واعطاءه تسلسل ولا يقوم بخزنه مره ثانية عندما يتكرر نفس (Id) في برنامج المصدر وإنما يشير الى موقعه في (Token Table)، اما حقل (Types) يبقى فارغ في هذه المرحلة.

١٠. ثم يقوم Lexical analyzer بأنشاء هيكل بياني (Tokens stream) يوضع فيه اسماء – النماذج (Token- name) حسب ترتيب ظهورها في الجدول الفقرات (Tokens table) باستثناء المعرف بدل ان يضع اسم النموذج يوضع بدلاً عنه نوع – النموذج (Token - attribute value) الموجود في جدول الفقرات (Tokens-table) وتسلسل ظهور المعرف الموجود في جدول الرموز (Symbol table).

Definition an identifier of tokens stream, that is consist of two parts are: token – attribute of this identifier in token table and sequence of this identifier in symbol table.

١١. ثم يمرر هذا الهيكل البياني (Tokens stream) الى (syntax analysis phase).

The Role of Lexical analyzer phase

- إذا وجد (Lexical analyzer phase) بأن (Lexeme) لا ينتمي الى أي من القواعد المايكروية سوف يقوم بأرسال رسالة خطأ الى المستخدم ويخبره بأن الـ(Lexeme) الحالي غير معرف بالنسبة إليه (Unknown identifier) وتحديد السطر الذي يقع فيه الـ(Lexeme) موضع الخلاف ليقوم المستخدم بتصليح الـ(Lexeme) في البرنامج المصدري الاصلي الموجود على الشاشة الادخال.
- أما الأخطاء الاخرى الموجودة في البرنامج المصدر الاصلي (اخطاء تركيب الفقرات أو اخطاء المعاني (syntactically ill formed or semantically unsound فهو غير مسؤول عنها في هذه المرحلة.
- الاخطاء التي يكتشفها هذا الطور تكون محدده لأنه لا يكتشف الاخطاء المتعلقة بتركيب الفقرات فعندما يصادف فقره لا تنتمي الى اي نموذج يقوم بإصدار رسالة خطأ.

Ex1: int x x x ; سوف لن تظهر رسالة خطا

Token Table	
اسم الفقرة	نوع الفقرة
int	K.W
x	Id
x	Id
x	Id
;	Pun

Ex2: سوف تظهر رسالة خطأ

int @x ;

Unknown identifier

The Role of Lexical analyzer phase

مخرجات طور تحليل الفقرات هي:

الجدول الأول: يسمى بجدول (Lexemes table) توضع به كل (Lexemes) التي تم تقطيعها بالاعتماد على الفراغ بعد دمج الفراغات المتعاقبة في واحد وحذف التعليقات.

الجدول الثاني: يسمى جدول الفقرات (المفردات - النماذج) (Tokens table) تدرج فيه كل فقرات البرنامج مع تحديد نوع كل فقره.

الجدول الثالث: هو جدول الرموز (Symbol table) حيث يفتح مدخل لكل معرف عند اول ظهور له في جدول النموذج وعند تكرار ظهور المعرف يتم عمل مؤشر الى مدخله في جدول الرموز.

سلسلة النماذج: وهي هيكل بياني (Tokens stream) يوضع فيه اسماء – النماذج (name-token) حسب ترتيب ظهورها في الجدول الفقرات (Tokens table) باستثناء المعرف بدل وضع اسم النموذج يوضع بدلاً عنه نوع – النموذج (attribute value) الموجود في جدول الفقرات (Tokens-table) وتسلسل ظهور المعرف الموجود في جدول الرموز (Symbol table).

Lexical analyzer phase

Source program

Ex.1:

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

1

Lexemes
if
(
x
>
y
)
x
=
10
;
while

2

Token table

Name	Attribute
if	Kw
(	Pun
x	Id
>	Op_GT
y	Id
)	Pun
x	Id
=	Op
10	Num
;	Pun
while	K.w

3

Symbol table

Sq.	Token
1	x
2	y

4

Token stream

التمثيل العملي في البرنامج

if	Kw
(	Pun
Id	1
>	Op
Id	2
)	Pun
Id	1
=	Op
10	Num
;	Pun
while	K.w

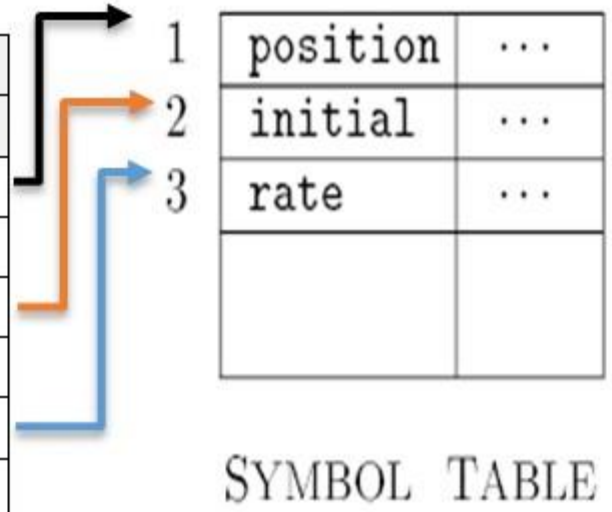
4

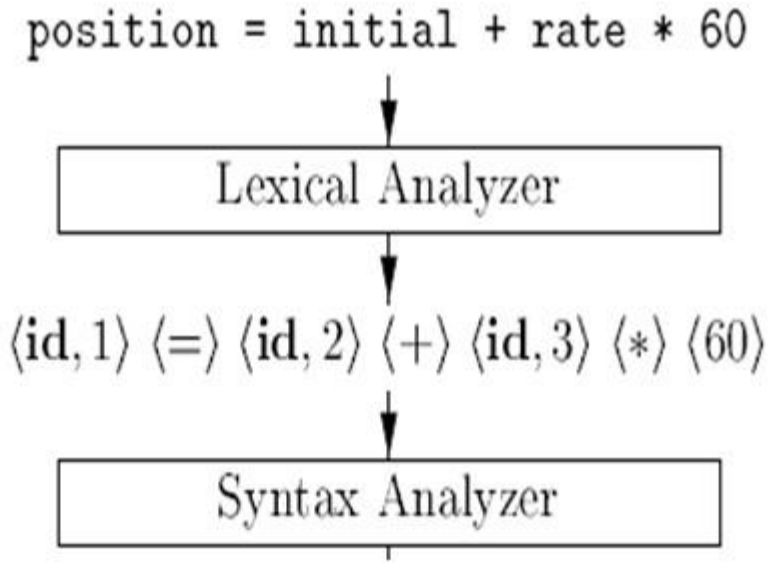
Token stream: التمثيل النظري على الورق

<if,k.w><(,pun> <Id,1> <>,Op_GT> <Id,2> <),pun> <Id,1> <=,Op_Assign.>
<10,Num> <;,pun> < while,k.w>

Ex.2:

Token Table		
Sq.	**Token**	**Token type**
1.	**Position**	**Id**
2.	**=**	**Op**
3.	**Initial**	**Id**
4.	**+**	**Op**
5.	**rate**	**Id**
6.	*****	**Op**
7.	**60**	**Number**





EXAMPLE 3

If we have part from the source program by C++ in below. How to analyze it by the lexical analyzer to pass on to the syntax analyzer. Find the following:

(1) Lexemes before correction?
(2) Corrections in the source program those the lexical analyzer phase?
(3) Lexemes after correction.
(4) Token table.
(5) Symbol table.
(6) Tokens stream.

```
1.   int  main ( )
2.   {
3.       int   x  ,  Ahmad  ,  float   y
4.       char  Array  [  10  ,  10  ]  ; // Array is string.
5.       x_y  :=                  5  ;
6.       switch  (  ch  )
7.       case  0  :  for  (  int  i  =  0  ;  i  <= 9  ;  I  ++
8.       Array  [  i  ]  =  8i  +  yx  *  *  @x  ;
9.       Cout  <<< "ahmad"  ;
10.      print  <<  Ahmed
11.  }
```

Solution (1): Lexemes before correction?

Sq.	Lexemes
1.	int
2.	main
3.	(
4.	)
5.	{
6.	int
7.	x
8.	,
9.	Ahmad
10.	,
11.	float
12.	y
13.	char
14.	Array
15.	[
16.	10
17.	,
18.	10
19.	]
20.	;
21.	x_y
22.	:=
23.	5
24.	;
25.	switch
26.	(
27.	ch
28.	)
29.	case
30.	0
31.	:
32.	for

Sq.	Lexemes
33.	(
34.	int
35.	i
36.	=
37.	0
38.	;
39.	i
40.	<=
41.	9
42.	;
43.	I
44.	++
45.	Array
46.	[
47.	i
48.	]
49.	=
50.	8i
51.	+
52.	yx
53.	*
54.	*
55.	@x
56.	;
57.	Cout
58.	<<<
59.	"ahmad"
60.	;
61.	print
62.	<
63.	<
64.	Ahmed
65.	}

	Source program before correction
1.	int main ( )
2.	{
3.	int x , Ahmad , float y
4.	char Array [10 , 10] ; // Array is string.
5.	x_y := 5 ;
6.	switch (ch)
7.	case 0 : for (int i = 0 ; i <= 9 ; I ++
8.	Array [i] = 8i + yx * * @x ;
9.	Cout <<< "ahmad" ;
10	print << Ahmed
11	}

Source program after correction

```
int main ( )
{
    int x , Ahmad , float y
    char Array [ 10 , 10 ] ; // Array is string.
    x_y = 5 ;
    switch ( ch )
    case 0 : for ( int i = 0 ; i <= 9 ; I ++
    Array [ i ] = i + yx * * x ;
    Cout << "ahmad" ;
    print << Ahmed
}
```

Solution (2): Corrections in the source program those the lexical analyzer phase

الأخطاء التي سوف يكتشفها طور تحليل الفقرات (Lexical analyzer) بعدما يقوم بسمح وقراءة البرنامج المصدري في هذا السؤال هي:

1-الـ(Lexeme) ← := والموجودة في السطر الخامس، حيث يظهر Lexical analyzer رسالة خطأ للمبرمج يخبره فيها بان الـ(Lexeme) := غير معرفه لديه ولا تنتمي الى اي قاعدة مايكروية والموجودة في السطر الخامس وانه لا يستطيع تنفيذ البرنامج مالم يتم تعديلها وربما يقترح عليه تعديلها الى = ومن ثم يقوم المبرمج بتعديلها الى = ليستمر تنفيذ البرنامج.

2-الـ(Lexeme) ← 8i والموجودة في السطر الثامن ويقوم المستخدم بتعديلها الى 8 (نفس الاجراء المتبع كما في الخطورة رقم واحد).

3-الـ(Lexeme) ← @x والموجودة في السطر الثامن ويقوم المستخدم بتعديلها الى i (نفس الاجراء المتبع كما في الخطورة رقم واحد).

4-الـ(Lexeme) ← <<< والموجودة في السطر التاسع ويقوم المستخدم بتعديلها الى << (نفس الاجراء المتبع كما في الخطورة رقم واحد).

5-أما الفراغات الموجودة في السطر الخامس والفراغات الموجودة في البرنامج ككل فهي لا تظهر كأخطاء للمستخدم ولكن البرنامج (Lexical analyzer) يقوم بدمجها بفراغ واحد – أي يعتبرها فراغ واحد – بين كل Two Lexemes متتاليين (ليعود مره اخرى بخزن البرنامج المصدري كبرنامج جديد بين كل Lexeme ولـ Lexeme فراغ واحد فقط).

6-أما بقية الاخطاء الموجودة فعلاً في البرنامج فأن الـ(Lexical analyzer) لا يعتبرها أخطاء في هذا المرحلة.

Solution (3): Lexemes after correction?

Sq.	Lexemes
1.	int
2.	main
3.	(
4.	)
5.	{
6.	int
7.	x
8.	,
9.	Ahmad
10.	,
11.	float
12.	y
13.	char
14.	Array
15.	[
16.	10
17.	,
18.	10
19.	]
20.	;
21.	x_y
22.	=
23.	5
24.	;
25.	switch
26.	(
27.	ch
28.	)
29.	case
30.	0
31.	:
32.	for

Sq.	Lexemes
33.	(
34.	int
35.	i
36.	=
37.	0
38.	;
39.	i
40.	<=
41.	9
42.	;
43.	I
44.	++
45.	Array
46.	[
47.	i
48.	]
49.	=
50.	8
51.	+
52.	yx
53.	*
54.	*
55.	x
56.	;
57.	Cout
58.	<<
59.	"ahmad"
60.	;
61.	print
62.	<
63.	<
64.	Ahmed
65.	}

في هذا المثال سوف يتم اعتبار ++ كـ(One lexeme) في هذا المثال سوف يتم اعتبار هذه الحالة.
أما إذا قام المستخدم منذ البداية في البرنامج المصدري بفصلها بفراغ سوف يتم اعتبارها (Two lexemes).

هنا المستخدم قام بالتعديل عليها من >>> الى >> كـ(One lexeme). في المثال سوف نأخذ هذا الجانب.
أما اذا قام بالتعديل عليها من >>> الى > > بينهم فراغ كـ (Two lexemes) أيضا تعتبر صحيحة في هذه المرحلة.

Solution (4 & 5): Token table and Symbol Table.

Table of Token			الارقام تشير الى التسلسل Id في جدول الرموز
Sq.	Token-name	Token-value	
1.	int	K.W	
2.	main	K.W	
3.	(	Pun	
4.	)	Pun	
5.	{	Pun	
6.	int	K.W	
7.	x	Id	1
8.	,	Pun	
9.	Ahmad	Id	2
10.	,	Pun	
11.	float	K.W	
12.	y	Id	3
13.	char	K.W	
14.	Array	Id	4
15.	[	Pun	
16.	10	Num	
17.	,	Pun	
18.	10	Num	
19.	]	Pun	
20.	;	Pun	
21.	x y	Id	5
22.	=	Op	
23.	5	Num	
24.	;	Pun	
25.	switch	K.W	
26.	(	Pun	
27.	ch	Id	6
28.	)	Pun	
29.	case	K.W	
30.	0	Num	
31.	:	Pun	
32.	for	K.W	

Symbol Table					
Sq.	Token-name				
1.	x				
2.	Ahmad				
3.	y				
4.	Array				
5.	x_y				
6.	ch				
7.	i				
8.	I				
9.	yx				
10.	Cout				
11.	print				
12.	Ahmed				

Table of Token			الارقام تشير الى التسلسل Id في جدول الرموز
Sq.	Token-name	Token-value	
33.	(	Pun	
34.	int	K.W	
35.	i	Id	7
36.	=	Op	
37.	0	Num	
38.	;	Pun	
39.	i	Id	7
40.	<=	Op	
41.	9	Num	
42.	;	Pun	
43.	I	Id	8
44.	++	Op	
45.	Array	Id	2
46.	[	Pun	
47.	i	Id	7
48.	]	Pun	
49.	=	Op	
50.	i	Id	7
51.	+	Op	
52.	yx	Id	9
53.	*	Op	
54.	*	Op	
55.	x	Id	1
56.	;	Pun	
57.	Cout	Id	10
58.	<<	Op	
59.	"ahmad"	Literal	
60.	;	Pun	
61.	print	Id	11
62.	<	Op	
63.	<	Op	
64.	Ahmed	Id	12
65.	}	Pun	

Solution (6): Tokens stream.

<int,k.w> <main,k.w> <(,pun> <),pun> <{,pun> <int,k.w> <Id,1>
<,,pun> <Id,2> <,,pun> <float,k.w> <Id,3> <char,k.w> <Id,4> <[,pun>
<10,number> <,,pun> <10,number> <],pun> <;,pun> <Id,5> <=,op>
<5,number> <;,pun> <switch,k.w> <(,pun> <Id,6> <),pun> <case,k.w>
<0,number> <:,pun> <for,k.w> <(,pun> <int,k.w> <Id,6> <=,op>
<0,number> <;,pun> <Id,6> <<=,GE> <9,number> <;,pun> <Id,8>
<++,op> <Id,4> <[,pun> <Id,7> <],pun> <=,op> <Id,7> <+,op> <Id,9>
<*,op> <*,op> <Id,1> <;,pun> <Id,10> <<<,pun> < "ahmad" , Literal >
<;,pun> <Id,11> <<,pun> <<,pun> <Id,12 > <},pun>

H.W. **(24)**

Suppose that we have the following source program by C++. How to analyze it by the lexical analyzer to pass on to the syntax analyzer? After correct the errors in the lexical analyzer.

1	#include <iostream.h>
2	int main()
3	{
4	int xy , Ahmad ,
5	char Array [10] ; // Array is string
6	xy = 5 ;
7	x.1 = 7 ;
8	fot (int i = 0 ; i <= 9 ; I + +)
9	Array [i] = i + yx * * x 1 ;
10	Ahmad = x ,
11	Cout << Ahmad ;
12	}

THANK YOU