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
LAB-NO. : 6

PREPARED BY

Lecturer:
Mohanad Dawood Al-Roomi

&

Assistant Lecturer:
Luay Ibrahim Klalif

(3) 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_ASS	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".

(1) A Program checks keywords.

الطريقة الاولى

Program (27.1)

```cpp
#include <iostream> //Program.4.1 checks or validates keywords.
#include <cstdio>
#include <string.h>
using namespace std;
int main()
{
    char K_W [5][7]={"for","if","while","switch","else"};
    char Lexeme[10];
    cout << "inter your Lexeme: " ;
    gets(Lexeme);
    int S = 0;
    for (int i = 0 ; i < 5 ; i++)
            if (strcmp(Lexeme , K_W[i])== 0)
            {
                S = 1 ;
                break;
            }
    if (S == 1)
         cout << "Accepted. The Lexeme is keyword" << endl;
    else
         cout <<"Rejected.The Lexeme is not keyword"<<endl;
    return 0;
}
```

الطريقة الثانية

Program (27.2)

```cpp
#include <iostream>//Program.4.2 checks or validates keywords.
#include <cstring>// جلب زميلكم عبد الوهاب فند
#include <cstdio>
using namespace std;
char keywords[6][10]={"for","if","while","switch","else"};
int main()
{
    cout << "Enter a lexeme: ";
    char x[10];
    gets(x);
    bool isKW = false;
    for (int i = 0; i < 5; i++)
        if (strcmp(x, keywords[i]) == 0)
            {
            isKW = true;
            break;
            }
    if (isKW)
        cout << "Accepted The Lexeme is keyword." << endl;
    else
        cout <<"Rejected.The Lexeme is not keyword."<<endl;
    return 0;
}
```

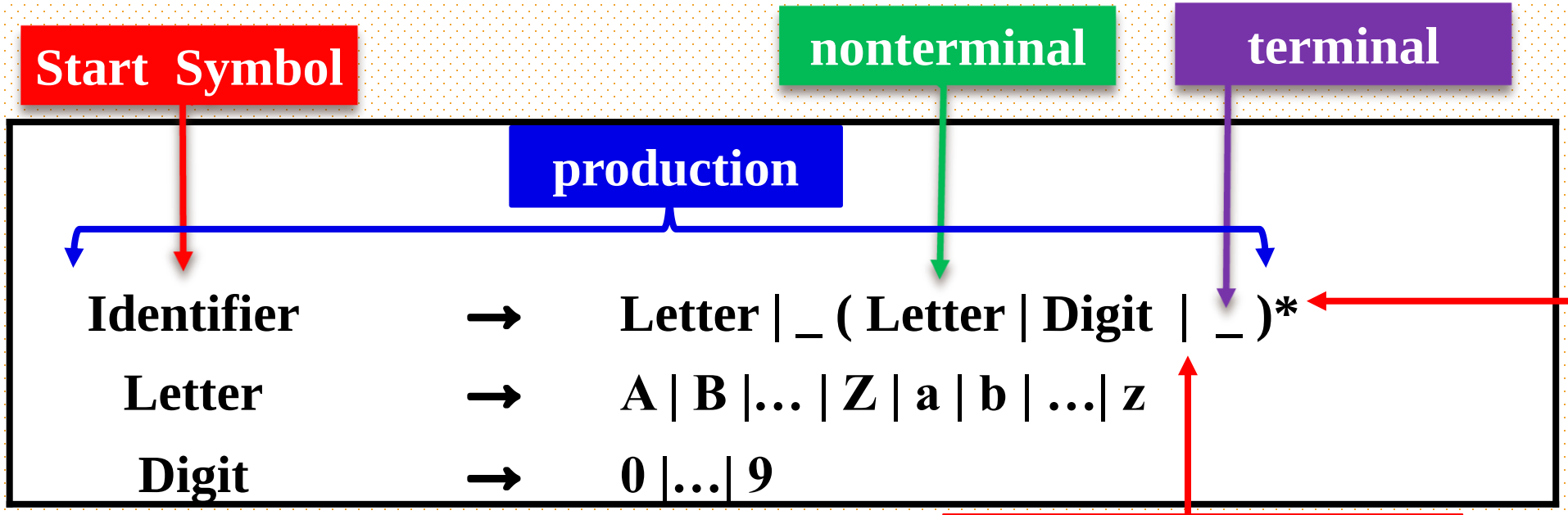
لاحظ تم استخدام كلمة Break الفائدة منه هي أن الشرط اذا وجد الكلمة المحجوزة تشابه الرموز lexeme المدخلة سوف يتوقف عن الاستمرار في البحث في مصفوفة الكلمات المحجوزة وهذا يقلل في وقت التنفيذ.

```
inter your Lexeme: if
Accepted. The Lexeme is keyword
```

```
inter your Lexeme: ifif
Rejected.The Lexeme is not keyword
```

Regular definition of Id (Grammars)

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



*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

x
X1
Student
mark_3
_
Paracetamol_2020

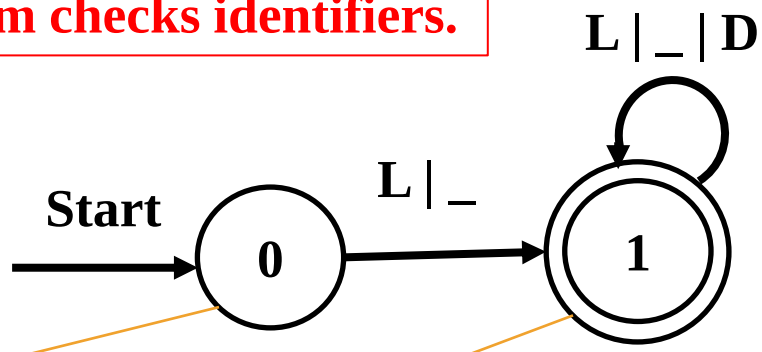
Ex: Identifier is rejected

@x
1X
Stu.dent
Mark+3
_!
Paracetamol_2020#

(2) A Program checks identifiers.

Program (28)

```
#include <iostream> //program 4.3.
#include <cstdio> //to check the lexeme is ID
#include <cstring>
using namespace std;
int main()
{
int state = 0 , Error = 0 , i = 0 , L;
char Lex[10];
cout << "enter your lexeme: ";
gets(Lex); L = strlen(Lex);
do
  {
    switch (state)
      {
        case 0: if (((Lex[i]>='a')&&(Lex[i]<='z')) || ((Lex[i]>='A')&&(Lex[i]<='Z')) || (Lex[i]=='_'))
                    state = 1;
                else  Error = 1;
                break;
        case 1: if (((Lex[i]>='a')&&(Lex[i]<= 'z')) || ((Lex[i]>='A')&&(Lex[i]<= 'Z')) || (Lex[i]=='_')
                   || ((Lex[i] >= '0')&&(Lex[i] <= '9')))
                    state = 1;
                else  Error = 1;
                break;
      }
     i++;
  }
while ((i < L) && (Error == 0));
if ((state == 1) && (Error == 0))
   cout<< "The Lexeme is Accepted ID \n";
else
   cout<< "The Lexeme is Rejected ID \n";
return 0 ;
}
```



Transition diagram of Identifier in C++

يمكن استخدام دالة isalpha لفحص الحروف الصغيرة والكبيرة، ودالة isdigit لفحص الأرقام من 0 – 9

هذا بدلاً من هذا

```
case 0: if (isalpha(Lex[i]) || (Lex[i]=='_'))
            state = 1;
        else  Error = 1;
        break;
case 1: if (isalpha(Lex[i]) || (Lex[i]=='_') || isdigit(Lex[i]))
            state = 1;
        else  Error = 1;
        break;
```

How to write a program by C++ to check the lexeme is keyword or identifier or not both?

Program (29.1)

```
#include <iostream>//program check the lexeme is K.W or ID or not both?
#include <cstring>//
using namespace std;
int main() {
    char k_w[5][10] = {"for","if","while","switch","else"};
    char lex[10];  int state = 0, Error  = 0, i = 0, L, t = 0 ;
    cout << "enter your lexeme: "; cin.getline(lex,11); L =strlen(lex) ;
    cout << "Length the lexeme: "<< L << endl;
    do {
        switch(state) {
            case 0: if (isalpha(lex[i])||(lex[i] == '_'))
                          state = 1 ;
                    else Error = 1 ;  break;
            case 1: if ((isalpha(lex[i]))||(lex[i] == '_')||(isdigit(lex[i])))
                          state = 1 ;
                    else Error = 1 ;   break;
          }
        i++ ;
      }
    while ((i < L) && (Error == 0));
    if ((state == 1) && (Error == 0))
      {
       for (int j = 0 ; j < 5 ; j++)
           if (strcmp(k_w[j], lex)== 0)
              { t = 1 ; break ; }
       if (t == 0) cout << "The lexeme is Identifier \n" ;
       else        cout << "The lexeme is keyword \n" ;
      }
    else cout << "The lexeme is not K.W or ID \n" ;
    return 0 ; }
```

لاحظ تم استخدام ملف الصديرة <cstring> واستخدامنا مصفوفة ثنائية للكلمات المحجوزة نوعها البياني char وتم قراءة الـ Lexeme المدخل مصفوفة أحادية ذو نوع بياني char .

لاحظ تم فحص الـ Lexeme في البداية على قانون الـ ID وهل هو ID ام لا . ثم يتم فرز K.W. عن ID .

لاحظ تم استخدام دالة strcmp() عند المقارنة لأن العنصرين نوعهما البياني مصفوفة رموز char .

RUN

```
enter your lexeme: else
Length the lexeme: 4
The lexeme is keyword
```

```
enter your lexeme: Mark_1
Length the lexeme: 6
The lexeme is Identifier
```

```
enter your lexeme: Mark_@
Length the lexeme: 6
The lexeme is not K.W or ID
```

Program (29.2)

```
1  #include <iostream>//program check the lexeme is K.W or ID or not both?
2  #include <string>//
3  using namespace std;
4  int main() {
5      string k_w[5] = {"for","if","while","switch","else"};
6      string lex;  int state = 0, Error  = 0, i = 0, L, t = 0 ;
7      cout << "enter your lexeme: ";  getline(cin,lex); L =lex.length() ;
8      cout << "Length the lexeme: "<< L << endl;
9      do {
10         switch(state) {
11             case 0: if (isalpha(lex[i])||(lex[i] == '_'))
12                             state = 1 ;
13                     else Error = 1 ;  break;
14             case 1: if ((isalpha(lex[i]))||(lex[i] == '_')||(isdigit(lex[i])))
15                             state = 1 ;
16                     else Error = 1 ;   break;
17           }
18         i++ ;
19       }
20     while ((i < L) && (Error == 0));
21     if ((state == 1) && (Error == 0))
22       {
23        for (int j = 0 ; j < 5 ; j++)
24            if (k_w[j] == lex)
25               { t = 1 ; break ; }
26        if (t == 0) cout << "The lexeme is Identifier \n" ;
27        else        cout << "The lexeme is keyword \n" ;
28      }
29     else cout << "The lexeme is not K.W or ID \n" ;
30     return 0 ; }
```

لاحظ تم استخدام ملف الصديرة <string>
كما تم استخدام مصفوفة أحادية للكلمات المحجوزة نوعها البياني string بينما تم قراءة الـ Lexeme المدخل ذو نوع بياني string .

لاحظ تم فحص الـ Lexeme في البداية على قانون الـ ID وهل هو ID ام لا . ثم يتم فرز ID عن K.W. .

لاحظ لم نستخدم دالة strcmp() عند المقارنة لأن العنصرين نوعهما البياني ليس مصفوفة رموز char .

RUN

```
enter your lexeme: else
Length the lexeme: 4
The lexeme is keyword
```

```
enter your lexeme: Mark_1
Length the lexeme: 6
The lexeme is Identifier
```

```
enter your lexeme: Mark_@
Length the lexeme: 6
The lexeme is not K.W or ID
```

(9)

Program (29.3)

لاحظ تم استخدام مصفوفة أحادية للكلمات المحجوزة نوعها البياني string . بينما تم قراءة الـ Lexeme المدخل ذو نوع بياني مصفوفة أحادية char

```
#include <iostream>//program 4.5.
#include <string.h> //to check the lexeme is keyword or identifier or not both?
#include <cstdio>
using namespace std;
int main() {
 int state = 0 , Error = 0 , i = 0, L , t = 0 ;
 string k_w[10] ={"for","if","while","switch","else"};
 char lex [10]; cout << "enter your lexeme: ";
 gets(lex); L =strlen(lex); cout << "Length the lexeme: "<< L << endl;
 do {
       switch(state)
          {
        case 0: if (((lex[i]>='a')&&(lex[i]<='z'))||((lex[i]>='A')&&(lex[i]<= 'Z'))
                     ||(lex[i] == '_'))
                        state = 1 ;
                else  Error = 1 ;
                break;
        case 1: if (((lex[i]>='a')&&(lex[i]<='z'))||((lex[i]>='A')&&(lex[i]<='Z'))
                     ||((lex[i]>='0')&&(lex[i]<='9'))||(lex[i]== '_'))
                        state = 1 ;
                else  Error = 1 ;
                break;
          }
       i++ ;
      }
 while ((i < L) && (Error == 0));
 if ((state == 1) && (Error == 0))
   {
     for (int j = 0 ; j < 5 ; j++)
         if (k_w [j] == lex)
            {
              t = 1 ; break ;
            }
     if (t == 0)   cout << "The lexeme is Identifier \n" ;
     else          cout << "The lexeme is keyword \n" ;
   }
 else cout << "The lexeme is not K.W or ID \n" ;
 return 0 ;   }
```

لن نحتاج استخدام strcmp() عند المقارنة لان العنصرين نوعهما البياني ليس مصفوفة char

```
enter your lexeme: else
Length the lexeme: 4
The lexeme is keyword
```

```
enter your lexeme: Mark_1
Length the lexeme: 6
The lexeme is Identifier
```

```
enter your lexeme: Mark_@
Length the lexeme: 6
The lexeme is not K.W or ID
```

Program (29.4)

```
#include <iostream>//program check the lexeme is K.W or ID or not both?
#include <cstring>
#include <string>
using namespace std;
int main() {
    char k_w[5][10] = {"for","if","while","switch","else"};
    string lex;  int state = 0, Error  = 0, i = 0, L, t = 0 ;
    cout << "enter your lexeme: "; getline(cin,lex);
    L =lex.length(); cout << "Length the lexeme: "<< L << endl;
    do {
        switch(state)
          {
            case 0: if (isalpha(lex[i])||(lex[i] == '_'))
                          state = 1 ;
                    else Error = 1 ;  break;
            case 1: if ((isalpha(lex[i]))||(lex[i] == '_')||(isdigit(lex[i])))
                       state = 1 ;
                    else Error = 1 ;  break;
          }
        i++ ;
      }
    while ((i < L) && (Error == 0));
    if ((state == 1) && (Error == 0))
      {
       for (int j = 0 ; j < 5 ; j++)
           if (k_w[j]== lex)
              { t = 1 ; break ; }
       if (t == 0) cout << "The lexeme is Identifier \n" ;
       else        cout << "The lexeme is keyword \n" ;
      }
    else cout << "The lexeme is not K.W or ID \n" ;
    return 0 ; }
```

لاحظ تم استخدام مصفوفة ثنائية للكلمات المحجوزة نوعها البياني char بينما تم قراءة الـ Lexeme المدخل ذو نوع بياني string .

لاحظ لم نستخدم دالة strcmp() عند المقارنة لأن احد العنصرين نوعه البياني ليس مصفوفة رموز char .

RUN

```
enter your lexeme: else
Length the lexeme: 4
The lexeme is keyword
```

```
enter your lexeme: Mark_1
Length the lexeme: 6
The lexeme is Identifier
```

```
enter your lexeme: Mark_@
Length the lexeme: 6
The lexeme is not K.W or ID
```

Type	Typical Size in Bits	Minimal Range
char	8	-127 to 127
unsigned char	8	0 to 255
signed char	8	-127 to 127
int	16 or 32	-32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	same as int
short int	16	-32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	same as short int
long int	32	-2,147,483,647 to 2,147,483,647
signed long int	32	same as long int
unsigned long int	32	0 to 4,294,967,295
float	32	Six digits of precision
double	64	Ten digits of precision
long double	80	Ten digits of precision

All Data Types Defined by the ANSI/ISO C Standard

Example	حدود النوع	النوع البياني في لغة C	
		النوع العددي الصحيح (يكون بدون الكسور العشريه)	1
int x; int w , y , z;	$\pm 32,767$	الصحيح	
long int x; long x;	$\pm 2,147,483,648$	الصحيح الطويل	
unsigned int x;	0 الى 65535	صحيح بدون اشارة	
unsigned long x;	0 الى 4 بليون	صحيح الطويل بدون اشارة	
		النوع الحقيقي (يكون مع الكسورالعشريه)	2
float y;	10^{-38} الى 10^{38}	حقيقي	
double z;	10^{-308} الى 10^{308}	حقيقي دقه مضاعفه	
Long double i;	10^{-4932} الى 10^{4932}	حقيقي طويل دقه مضاعفه	
		النوع الرمزي	3
char ch; ch = 'a';	يقبل رمز واحد فقط طولة 1 بايت		
char name [10]; name="Iraq-----/0";	خيط رمزي طوله 10 بايت		

Regular definition of Unsigned numbers (Grammars) قواعد كتابة التعريف المنتظم للأرقام بدون الاشارة

- General grammar Unsigned numbers (integer or floating point) are strings such as 3 , 10 , 5280 , 0.01234 , 6.336E4 , or 1.89E-4 the regular definition c , c++ , Pascal language.

(12)

A. How to write grammar of unsigned number in C++ language:

digit → 0 |...| 9

digits → digit (digit)*

OptionalFraction → . digits | ε

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

Number → digits OptionalFraction OptionalExponent

Number → digits (. digits | ε) ((E (+|-| ε) digits) | ε)

| معنها أو بحيث يجب ان يأتي احد الأرقام ومره واحدة فقط.

*معنها الحالة ما داخل القوس لا تأتي ولا مره مطلقاً أو تتكرر مرة او أكثر.

ε معنها حالة نفي أي انها لا تأتي ولا المرة

Another way: How to write unsigned number in C++ language:

D	→	0 |...| 9
DS	→	DD*
Num	→	DS (. DS)? (E [+|-]? DS)?

أو تستطيع ان تكتب هذه الخطوة بالصيغة التالية كذلك :

DS → D^+

+ معنها يجب ان تأتي أما مرة واحدة على الاقل أو عدد من المرات

? معنها القوس الذي قبلها يأتي مرة واحدة فقط أو لا يأتي ولا مرة

Regular definition of Unsigned numbers (Grammars) قواعد كتابة التعريف المنتظم للأرقام

❑ B. How to draw one transition diagram for unsigned number instruction?

digit → 0 |...| 9

digits → digit (digit)*

OptionalFraction → . digits | ε

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

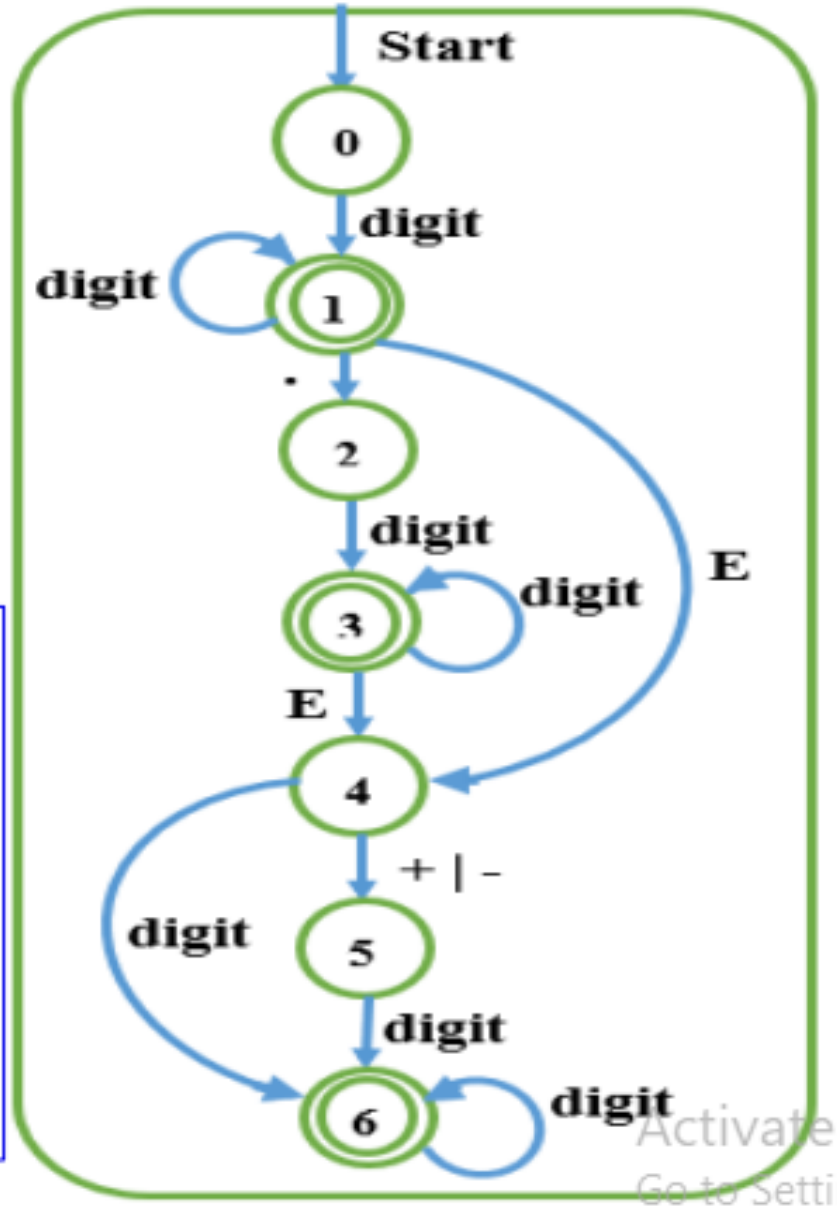
Number → digits (. digits)? (E [+|-]? digits)?

	Ex: Cases of accepted	Ex: Cases of rejected
1.	7	7w
2.	93.5	.45
3.	12E+4	E+12
4.	77.3E-10	77.E-10
5.	Etc.	

ملاحظة:

لايتم حفظ الـــ numbersالموجودة في العالم الحقيقي بشكل مسبق في ذاكرة البرنامج (كما يتم ذلك في الكلمات المحجوزة Keywordsوالتي يتم تدوينها وحجزها مسبقا داخل هيكل بياني في البرنامج).

وانما يتم خزن برنامج يختبر المدخل (characters) والتي تم تقطيعه وتجميعه بشكل lexemeسوف يولد number أم لا بالاعتماد على قواعد الجبر الرياضي.

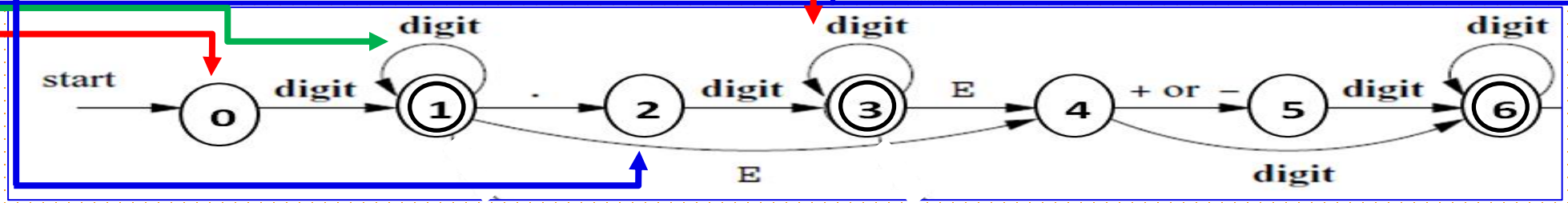


(3) A Program check the number.

Program (30)

```
#include <iostream>//program for unsigned numbers in c++
#include <cstdio>
#include <cstring>  //#include <string>
using namespace std;
int main()
{
int i = 0 , Error = 0 , state = 0, L ;
char Lex [10] ;                          // string Lex;
cout<< "Enter a lexeme to check of number: ";
gets(Lex);                               // getline(cin,Lex);
L= strlen(Lex);                          // L=Lex.length();
do
  {
   switch(state)
        {
          case 0: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 1;
                  else Error = 1;
                  break;
          case 1: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 1;
                  else if (Lex[i] == '.')
                       state = 2;
                  else if (Lex[i] == 'E')
                       state = 4;
                  else Error = 1;
                  break;
          case 2: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 3;
                  else Error = 1;
                  break;
          case 3: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 3;
                  else if (Lex[i] == 'E')
                       state = 4;
                  else Error = 1;
                  break;
          case 4: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 6;
                  else if ((Lex[i]=='+') || (Lex[i]=='-'))
                       state = 5;
                  else Error = 1;
                  break;
          case 5: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 6;
                  else Error = 1;
                  break;
          case 6: if ((Lex[i] >= '0') && (Lex[i] <= '9'))
                       state = 6;
                  else Error = 1;
                  break;
        }
    i++;
  }
while ((i < L) && (Error == 0));
if (((state == 1)||(state == 3)||(state == 6)) && (Error == 0))
     cout<< "Accepted. The lexeme is number /n";
else
     cout<< "Rejected. The lexeme is not number /n";
return 0;
}
```

يمكن استبدال بالإيعاز التالي
(isdigit (Lex[i]))
isdigit دالة تفحص الرمز
هل هو digit بين 0 – 9

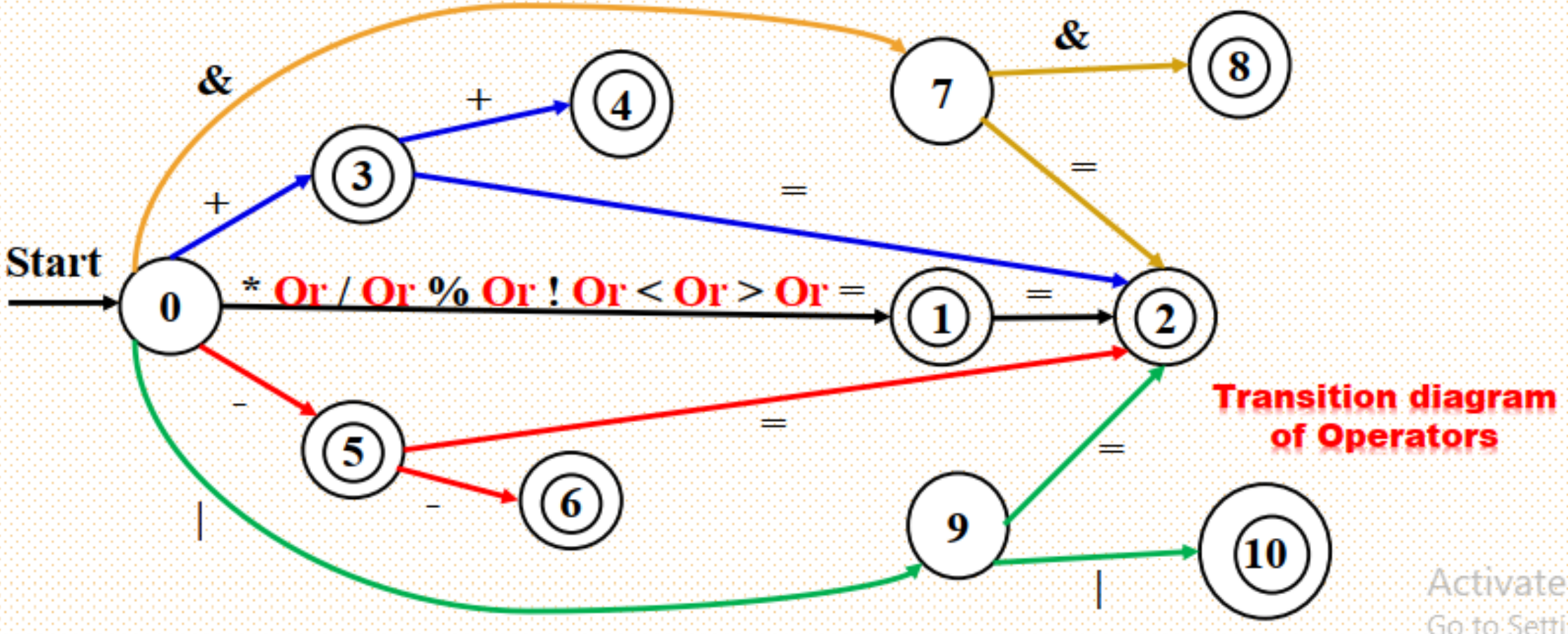


Grammar of Operator

OP → + | - | * | / | % | = | > | < | ! | += | -= | *= |/= | %= |== |>= |<= |!= |&= | |= |++| -- | &&| ||

Shorthand

OP → (* | / | % | = | > | < | !)(=)? | +(+|=)? | - (-|=)? | &(&|=)| |(| | =)

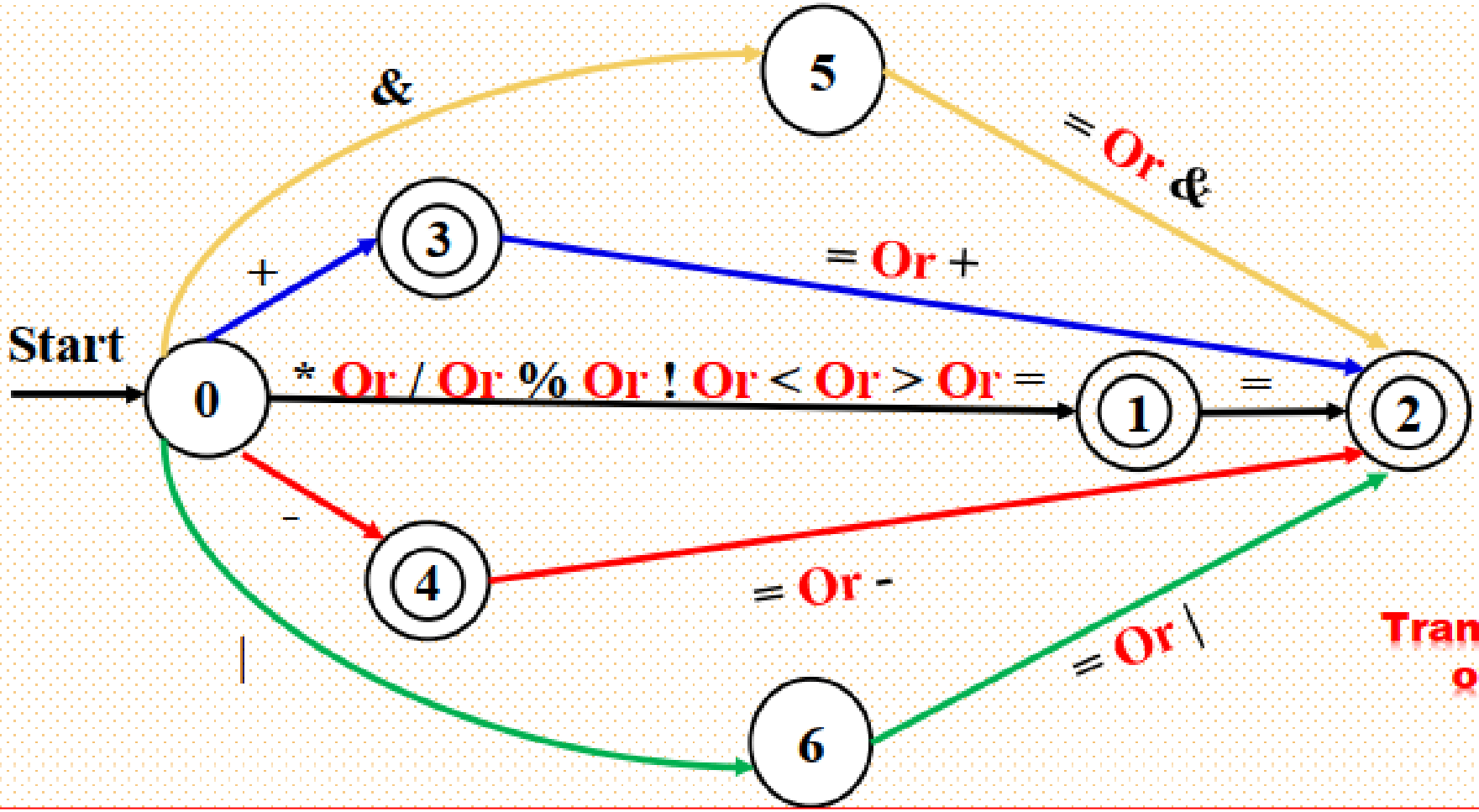


Grammar of Operator

OP → + | - | * | / | % | = | > | < | ! | += | -= | *= |/= | %= |== |>= |<= |!= |&= | |= |++| -- | &&| ||

Shorthand

OP → (* | / | % | = | > | < | !)(=)? | +(+|=)? | - (-|=)? | &(&|=) | |(| | =)



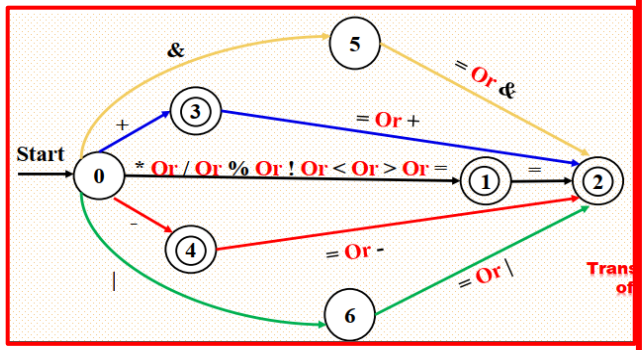
Transition diagram of Operators

(4) A Program checks operations.
الطريقة الاولى

Program (31.1)

```
#include <iostream>//program to check the lexeme is Operator or not.
#include <string.h>
#include <cstdio>
using namespace std;
int main()
{
int state = 0, Error = 0, i = 0, L ; char lex[10];
cout << "Enter the lexeme of Operator to check it: ";
gets(lex); L = strlen(lex);
do
  {
   switch (state)
      {
      case 0 : if ((lex[i]=='*')||(lex[i]=='/')||(lex[i]=='%')||(lex[i]=='!')
                 ||(lex[i]=='>')||(lex[i]=='<')||(lex[i]=='='))
                    state = 1;
               else if (lex[i] =='+')
                    state = 3;
               else if (lex[i] =='-')
                    state = 4;
               else if (lex[i] =='&')
                    state = 5;
               else if (lex[i] =='|')
                    state = 6;
               else Error = 1;
               break;
      case 1 : if ((lex[i] =='='))
                    state = 2;
               else Error = 1;
               break;
      case 3 : if ((lex[i]=='+')||(lex[i]=='='))
                    state = 2;
               else Error = 1;
               break;
      case 4 : if ((lex[i] =='-') || (lex[i] =='='))
                    state = 2;
               else Error = 1;
               break;
      case 5 : if ((lex[i] =='&') || (lex[i] =='='))
                    state = 2;
               else Error = 1;
               break;

      case 6 : if ((lex[i] =='|') || (lex[i] =='='))
                    state = 2;
               else Error = 1;
               break;
      default : Error = 1 ;
      }
   i++;
  }
while ((i < L) && (Error == 0));
if (((state==1)||(state==2)||(state==3)||(state ==4))&&(Error==0))
    cout<< "Accepted. The lexeme is operator in C++ language.\n";
else
    cout<< "Rejected. The lexeme is not operator in C++ language.\n";
return 0;
}
```



(4) A Program checks operations.
الطريقة الثانية

Program (31.2)

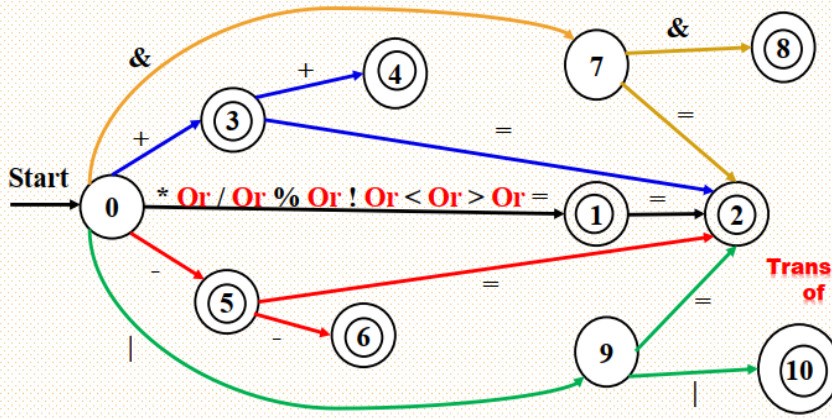
```cpp
#include <iostream>//program to check the lexeme is Operator or not.
#include <cstring>
#include <cstdio>
using namespace std;
char arr[26][3] = {"+", "-", "++", "--", "&", "&&", "!", "*", "|", "||",
                   "/", "%", "<", ">", "=", "==", "&=", "!=", "-=", "+=",
                   "*=", "/=", "%=", "<=", ">=", "|="};
int main()
{
    char lex[10];
    cout << "Enter a Lexeme: ";
    gets(lex);
    int s = 0;
    for (int i = 0; i < 26; i++)
        if (strcmp(lex, arr[i]) == 0)
        {
            s = 1;
            break;
        }
    if (s == 1)
        cout << "Accepted. The lexeme is Operator" << endl;
    else
        cout << "Rejected. The lexeme is not Operator" << endl;
    return 0;
}
```

Grammar of Operator

OP → + | - | * | / | % | = | > | < | ! | += | -= | *= |/= | %= |== |>= |<= |!= |&= | |= |++| -- | &&| ||

Shorthand

OP → (* | / | % | = | > | < | !)(=)? | +(+|=)? | - (-|=)? | &(&|=)| |(| | =)



A Program checks punctuation codes. **H.W. 1.**

Program (32)

A program checks literal. **H.W.2.**

Program (33)

H.W. 3. Write programs by C++ language, which call the grammars of macro-preprocessor and identify the errors.

< 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

Program (34)

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;
```