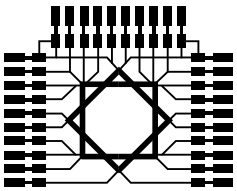


المرحلة الأولى من المشروع - أخر موعد للتسليم 11- أيار - 2025

بناء محلل قواعدي للغة Mini Pascal

نعالج في هذا الوظيفة مجموعة جزئية من لغة Pascal ندعوها MiniPascal، سنهتم خلال هذه المرحلة ببناء الجزء الأول من المترجم والذي يتضمن التحليل المفرداتي والقواعدي لبرامج مكتوبة بلغة MiniPascal. نستعرض في هذا القسم مفردات اللغة وقواعدها ونحدد البرامج التي يتوجب على الطالب تقديمها في نهاية هذه المرحلة وذلك قبل الموعد المحدد في العنوان.

تحليل المفردات (Lexical Analysis)

تم إنجاز هذه المرحلة في الوظيفة الأولى، ولكن نذكر شروطها مرة أخرى، بالإضافة على انه يجب ربط المحلل المفرداتي الذي تم غنجازه مع المحلل القواعدي الذي سيتم إنجازه. نستعرض في هذا القسم مفردات اللغة. **لا يتم التمييز في باسكال بين الأحرف الصغيرة والكبيرة**، مثلا" يمكن كتابة if كما يلي: IF If iF if وجميعها تعتبر مقبولة، سنستعرض المفردات التي تؤلف نص برنامج مكتوب بلغة MiniPascal:

تعتبر الأحرف الناتجة عن المفاتيح Space، Tabulation، Return، فراغات.

يمكن استخدام نمطين من التعليقات: تبدأ بـ { وتنتهي بـ } وتمتد على عدة أسطر، وتعليقات لها شكل تعليقات لغة C++ تبدأ بـ // وتمتد على سطر واحد فقط من نص البرنامج.

تتبع المعرّفات (Identifier) التعبير المنتظم <id> التالي:

```
<Digit>      ::=    0-9
<Alpha>      ::=    a-z | A-Z
<id>   ::=    (<Alpha> | _ ) (<Alpha> | <Digit> | _ )*
```

تتبع الثوابت الرقمية الصحيحة التعبير المنتظم <Integer> التالي:

```
<int_num>    ::=    0 | 1-9 <Digit>*
```

تتبع الثوابت الرقمية الحقيقية التعبير المنتظم <Real> التالي:

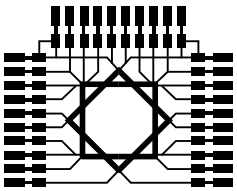
```
<real_num>   ::=    <Digit>+ . <Digit>* ((e | E) (- | +) ? <Digit>+) ?
             |      . <Digit>+ ((e | E) (- | +) ? <Digit>+) ?
             |      <Digit>+ (e | E) (- | +) ? <Digit>+
```

تعتبر الكلمات التالية، كلمات مفتاحية:

program var integer real function procedure while do begin end if then else array of div not or and

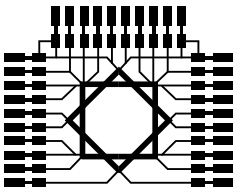
يتحليل القواعد الصرفية (Syntax Analysis):

نعتمد على القواعد الصرفية التالية، حيث نعتبر الرمز اللانهائي <program> مدخلاً للقواعد الصرفية الناظمة للغة Minipascal:



```
<program>          ::=   program id ;
                         <declarations>
                         <subprogram_declarations>
                         <compound_statement>
<identifier_list>  ::=   id
                         | <identifier_list> , id

<declarations>     ::=   <declarations> var <identifier_list> : <type> ;
                         | /* empty */
<type>             ::=   <standard_type>
                         | array [ <int_num> .. <int_num> ] of
                           <standard_type>
<standard_type>    ::=   integer
                         | real
                         | boolean
<subprogram_declarations>     ::=   <subprogram_declarations>
                                    <subprogram_declaration> ;
                                    | /* empty */
<subprogram_declaration>::=<subprogram_head><compound_statement>
<subprogram_head>  ::=   function id <arguments> : <standard_type> ;
                         | procedure id <arguments> ;
<arguments>        ::=   ( <parameter_list> )
                         | /* empty */
<parameter_list>   ::=   <identifier_list> : <type>
                         | <parameter_list> ; <identifier_list> : <type>
<compound_statement> ::= begin <optional_statements> end
<optional_statements> ::= <statement_list>
                           | /* empty */
<statement_list>   ::=   <statement>
                         | <statement_list> ; <statement>
<statement>        ::=   <variable> := <expression>
                         | <procedure_statement>
                         | <compound_statement>
                         | if <expression> then <statement>
                         | if <expression> then <statement> else
                           <statement>
                         | while <expression> do <statement>
<variable>         ::=   id
                         | id [ <expression> ]
<procedure_statement> ::= id
                           | id ( <expression_list> )
<expression_list>  ::=   <expression>
                         | <expression_list> , <expression>
<expression>       ::=   id
                         | <int_num>
                         | <real_num>
                         | true
                         | false
                         | id (expression_list> )
                         | ( <expression> )
                         | <expression> <unary_operator> <expression>
                         | not <expression>
<unary_operator>   ::=   + | - | * | / | div | > | < | >= | <= | = | <> | not | or | and
```



نلاحظ مما سبق أنه من الضروري إلغاء حالات الغموض (ambiguities) التي تتسم بها الحالات السابقة بحيث يتوجب في النهاية تقديم ملف MiniPascal.y خالٍ من أي حالات conflicts.

يعطي الجدول التالي أسلوب تجميع العملية الواحدة بالإضافة إلى أفضليات العمليات المختلفة وذلك بترتيبها من العمليات الأضعف إلى العمليات الأقوى:

العملية	التجميع
or	يساري
and	يساري
not	يساري
=, <>	يساري
>, >=, <, <=	يساري
+, -	يساري
*, /, div	يساري

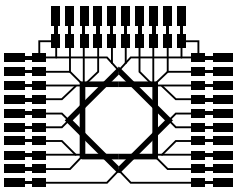
معالجة الأخطاء

عند تشغيل المترجم، يجب أن يكون قادراً، على تحديد الأخطاء المفرداتية والقواعدية الموجودة في الملف المصدري (ملف test.pas مثلاً) عبر تحديد طبيعة وموقع الخطأ في الملف المصدري. يمكن مثلاً الاعتماد على الصيغة التالية لرسائل الخطأ:

File "test.pas", line 4, characters 5-6
Syntax error

المطلوب:

١. كتابة مترجم، يدعى MiniPasC، يقبل كدخل، ملف مصدري مكتوب بلغة MiniPascal



٢. تقديم تقرير بسيط يحتوي، فقط، على التصميم التفصيلي لبنية المعطيات المستخدمة التي يقوم المترجم ببنائها لتمثيل البرنامج المصدري المكتوب بلغة MinPascal. سيتم استخدام هذه البنية، لاحقاً، في عمليات التحليل الدلالي. أي بنية الشجرة كما تم شرحها، ويفضل البدء بتنفيذها مستفيدين من المثال الموجود على eclass

٣. تقديم البرامج والتقارير المتعلقة بهذه المرحلة ضمن ملف zip.* يمثل أرشيفاً لمجلد يحتوي على الرماز المصدري للبرامج التي تشكل المترجم MiniPasC مع توضيح طريقة ترجمة وتنفيذ وتجريب هذا المترجم من خلال ملف README.TXT. يجب أن يحتوي المجلد أيضاً، على بضعة برامج بسيطة مكتوبة بلغة MiniPascal لتجريبها مع المترجم MiniPasC بالإضافة إلى التقرير المذكور في الفقرة السابقة والذي يوضح بنية المعطيات الناتجة عن هذه المرحلة. كما يجب أن تكون البرامج المصدرية حاوية على تعليقات واضحة ومفصلة لمختلف أقسامها.

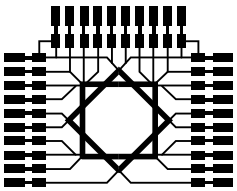
أمثلة بلغة باسكال:

```
01.  program HelloWorld;
02.
03.  begin
04.     writeln('Hello, world!');
05.  end.
```

```
01.  program Factorial
02.  var
03.    Counter, Factorial: integer;
04.  begin
05.    Counter := 5;
06.    Factorial := 1;
07.    while Counter > 0 do begin
08.      Factorial := Factorial * Counter;
09.      Counter := Counter - 1;
10.    end;
11.    Write(Factorial);
12.  end.
```

مثال مصفوفات

```
Program ArrayExample; Var
    myVar   : Integer;
    myArray : Array[1..5] of
Integer;  Begin
 myArray[2] := 25;
myVar := myArray[2];
End.
```



مثال إجرائية

```
Program Lesson7_Program2;
Uses Crt;
Procedure DrawLine(X : Integer; Y : Integer);
{the decleration of the variables in brackets are
called  parameters or arguments} Var Counter : Integer;
 {normally this is called a local variable}
Begin
 GotoXy(X,Y); {here I use the parameters}
 textcolor(green);
 For Counter := 1 to 10 do
Begin
   write(chr(196));
  End;
End;
Begin
 DrawLine(10,5);
 DrawLine(10,6);
 DrawLine(10,7);

DrawLine(10,10);
Readkey; End.
```