

Compte rendu du mini-projet (analyseur syntaxique)

**Réalisé par : Ghammam anwar
Ichraf ben Fadhel
Insaf Khorcheni**

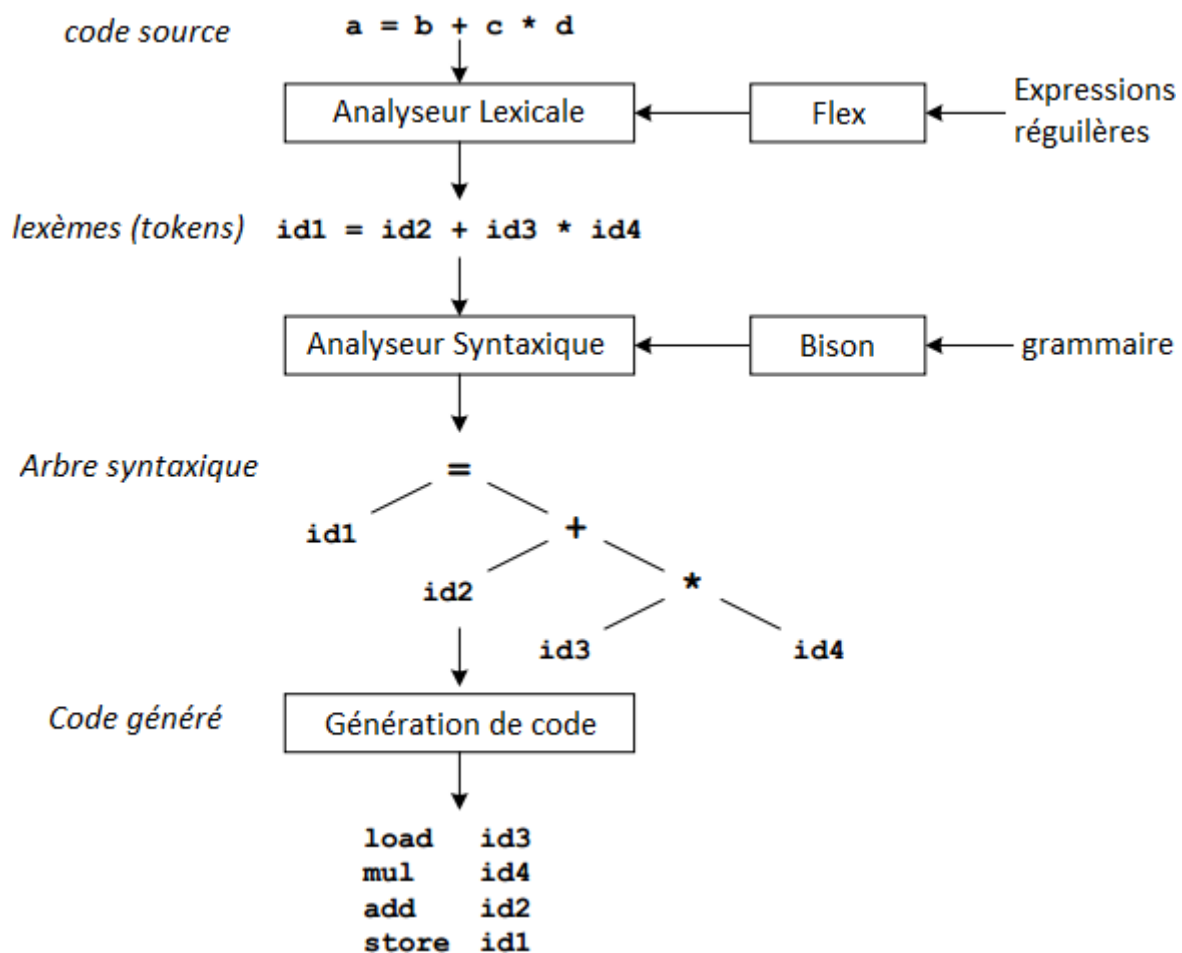
GL4–groupe2

Introduction

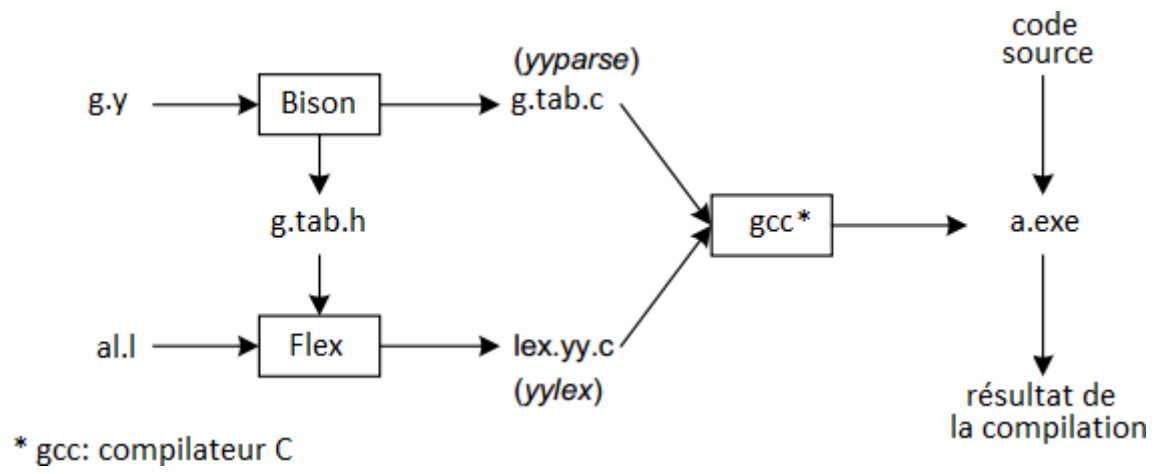
L'intérêt de ce mini-projet est de créer une version simplifiée d'un compilateur du langage Pascal.

Pour cela, on est amené à créer un analyseur lexicale : pour reconnaître les différents lexèmes (*tokens*) du code source et un analyseur syntaxique : afin de vérifier l'ordre des lexèmes reconnues dans le code. (qui est le but de notre compte rendu)

Les outils utilisés sont : *bison* pour la génération de l'analyseur syntaxique et *code::blocks* pour compiler les fichiers C générés par *flex*.



Flex doit obtenir un fichier (.l) pour générer un analyseur lexicale, ce fichier contient les expressions régulières qui définissent les différents lexèmes du langage, *bison* doit obtenir un fichier (.y) qui contient la description de la grammaire du langage.



Analyse syntaxique

Pour l'analyse syntaxique on doit définir la grammaire qu'utilisent les compilateurs du langage pascal pour vérifier la syntaxe des codes entrés.

Le fichier *bison(.y)* a le même format que le fichier *flex(.l)*

Définitions: Où on peut inclure les librairies C et déclarer des Variables auxiliaires ou alors des signatures de fonctions à Implémenter dans la dernière partie du fichier.

Définition de la grammaire: Dans cette partie on doit inscrire les Différents règles de production de la grammaire.

Fonctions en C: La dernière partie du fichier *bison* est consacré à l'implémentation des fonctions principales et optionnels utiles dans la phase de l'analyse sémantique.

... définitions ...

%%

... définition de la grammaire ...

%%

... fonctions en C ...

```
%{
```

```
#include <stdio.h>
```

```
int yyerror(char const *msg);
```

```
int yylex(void);
```

```
extern int yylineno;
```

```
%}
```

```
|
```

Partie définitions du fichier exemple.y (1/2)

```

/* l'axiome de la grammaire */
%token ID
%token NB
%token OPADD
%token OPAFFECT
%token OPMUL
%token OPREL
|
    /* Mots clés */
%token BEGIN_TOKEN
%token DO
%token ELSE
%token END
%token FUNCTION
%token IF
%token NOT
%token PROCEDURE
%token PROGRAM
%token THEN
%token VAR
%token WHILE
%token INTEGER
%token READ
%token WRITE
%token DEUX_POINTS
%token ARRAY
%token POINT_VIRGULE
%token VIRGULE
%token CHAINE
%token INT
%token PARENTHESE_OUVRANTE
%token PARENTHESE_FERMANTE
%token COMMENTAIRE

```

Partie définitions du fichier pascal.y (2/2)

%start programmes

%%

```
programmes          : PROGRAM
                     ID
                     POINT_VIRGULE
                     | PROGRAM
                     ID
                     POINT_VIRGULE
                     declaration
                     | PROGRAM
                     ID
                     POINT_VIRGULE
                     declaration
                     instruction_composee
                                     { output_syn("fin du programme"); }
                     |
                     PROGRAM
                     ID
                     POINT_VIRGULE
                     instruction_composee

                     |error ID POINT_VIRGULE      {yyerror (" program attendu on line : "); }
                     |PROGRAM error POINT_VIRGULE  {yyerror (" nom du programme invalide on line : "); }
                     |PROGRAM ID error             {yyerror (" point virgule attendu on line : "); };
```

declaration:

```
    declaration_var
    declarations_sous_programmes
```

;

declaration_var:

```
    declaration_var
```

Activer Windows

Accédez aux paramètres pour activer Windows

Partie définition de la grammaire du fichier pascal.y (1/4)


```

declaration_var:
    declaration_var
    VAR
    liste_identificateurs
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE

|      /* chaîne vide */

|  declaration_var
    error { yyerror("mot cle var introuvable !"); }
    liste_identificateurs
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE
|

    declaration_var
    VAR
    error { yyerror("identificateur de variable manquante ");}
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE
|  declaration_var
    VAR
    liste_identificateurs
    error { yyerror(": manquante !"); }
    INTEGER
    POINT_VIRGULE

```

```

        INTEGER
        error { yyerror(" ; manquante !"); }

;

liste_identificateurs:
    ID
|    liste_identificateurs
    POINT_VIRGULE
    ID

|    liste_identificateurs
    error {yyerror(" ; manquante !");}
    ID
|    liste_identificateurs
    POINT_VIRGULE
    error {yyerror(" id introuvable!");}

;

declarations_sous_programmes:
    declarations_sous_programmes
    declarations_sous_programme
    POINT_VIRGULE
|    /* cha ne vide */
|
    declarations_sous_programmes

```

Partie d finition de la grammaire du fichier pascal.y (2/4)

```

declarations_sous_programmes:
    declarations_sous_programmes
    declarations_sous_programme
    POINT_VIRGULE
|
    /* chaÃªne vide */
|
    declarations_sous_programmes
    declarations_sous_programme
    error {yyerror(" ; manquante!");}
;

declarations_sous_programme:
    entete_sous_programme
    declaration
    instruction_composee
|
    entete_sous_programme
    instruction_composee

|error {yyerror(" entete de programme manquante!");}
    declaration
    instruction_composee

|entete_sous_programme
    declaration
    error {yyerror(" corps de fonction / procedure manquant manquante!");}

;

```

```

entete_sous_programme:
    FUNCTION
    ID
    arguments
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE

    |

    error {yyerror(" mot cle function manquant !");}
    ID
    arguments
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE

    |
    FUNCTION
    error {yyerror(" nom de fonction manquant !");}
    arguments
    DEUX_POINTS
    INTEGER
    POINT_VIRGULE
|
    FUNCTION
    ID
    arguments
    error {yyerror(" : manquant !");}
    INTEGER
    POINT_VIRGULE
    |
    FUNCTION

```

<

```

FUNCTION
ID
arguments
DEUX_POINTS
error {yyerror(" return type de fonction manquant !");}
POINT_VIRGULE
|
FUNCTION
ID
arguments
DEUX_POINTS
INTEGER
error {yyerror(" ;manquante!");}
|
PROCEDURE
ID
arguments
POINT_VIRGULE
|
PROCEDURE
error {yyerror("nom e procedure manquant !");}
arguments
POINT_VIRGULE

|
PROCEDURE
ID
arguments
error {yyerror(";manquante!");}

```

```

arguments:
    PARENTHESE_OUVRANTE
    liste_parametres
    PARENTHESE_FERMANTE
|   error{ yyerror("parenthese ouvrante manquante"); }
    liste_parametres
    PARENTHESE_FERMANTE
|   PARENTHESE_OUVRANTE
    liste_parametres
    error{ yyerror("parenthese fermante manquante"); }
|   /* chaÃne vide */
;

liste_parametres:
    parametre
|   liste_parametres
    POINT_VIRGULE
    parametre
|   liste_parametres
    error{ yyerror("point virgule manquante"); }
    parametre
|   error{ yyerror("parametre manquante"); }
    POINT_VIRGULE
    parametre
|   liste_parametres
    POINT_VIRGULE
    error{ yyerror("parametre manquante"); }

```

```

parametre:
    ID
    DEUX_POINTS
    INTEGER
|   error{ yyerror("identifiant manquant"); }
    DEUX_POINTS
    INTEGER
|   ID
    DEUX_POINTS
    error{ yyerror("integer manquant"); }
|   VAR
    ID
    DEUX_POINTS
    INTEGER
;

instruction_composee:
    BEGIN_TOKEN
    instructions_optionnelles
    END|
    error{ yyerror("Begin manquante"); }
    instructions_optionnelles
    END
|   BEGIN_TOKEN
    instructions_optionnelles
    error{ yyerror("End manquante"); }
;

instructions_optionnelles:
    liste_instructions
;

```

<

```

liste_instructions:
    instruction POINT_VIRGULE
|
    liste_instructions
    instruction POINT_VIRGULE
|
    instruction
    error { yyerror("point virgule d'instruction omis"); }
;

instruction:
    variable
    OPAFFECT
    expression
|
    appel_de_procedure
|
    instruction_composee
|
    IF
    expression
    THEN
    instruction
    ELSE
    instruction
|
    error { yyerror("IF manquante"); }
    expression
    THEN
    instruction
    ELSE
    instruction
|
    IF
    expression
    error { yyerror("THEN manquante"); }
    instruction
    ELSE
    instruction

```



```

        instruction
    |
        IF
        expression
        THEN
        instruction
        error { yyerror("ELSE manquante"); }
        instruction
    |
        WHILE
        expression
        DO
        instruction
    |
        /* chaÃ©ne vide */
;

variable:
    ID
;

appel_de_procedure:
    ID
    |
    ID
    error { yyerror("PARENTHESE OUVRANTE manquante"); }
    liste_expressions
    PARENTHESE_FERMANTE
    |
    ID
    PARENTHESE_OUVRANTE
    liste_expressions
    PARENTHESE_FERMANTE
    |
    ID
    PARENTHESE_OUVRANTE
    liste_expressions
    error { yyerror("PARENTHESE_FERMANTE manquante"); }
;

```

```

liste_expressions:
    expression
|
    liste_expressions
    VIRGULE
    expression
|
    liste_expressions
    error { yyerror("virgule manquante"); }
    expression
|
    liste_expressions
    VIRGULE
    error { yyerror("expression manquante"); }
;

expression:
    expression_simple
|
    expression_simple
    OPREL
    expression_simple
|
    error { yyerror("expression simple manquante"); }
    OPREL
    expression_simple
|
    expression_simple
    error { yyerror("OPREL manquant"); }
    expression_simple
|
    expression_simple
    OPREL
    error { yyerror("expression simple manquante"); }
;

expression_simple:
    terme
|
    signe

```

```

expression_simple:
    terme
|   signe
    error { yyerror("terme manquant"); }
|   terme
    error { yyerror("signe manquant"); }
|   expression_simple
    OPADD
    terme
|   error { yyerror("expression simple manquante"); }
    OPADD
    terme
|   expression_simple
    error { yyerror("OPADD manquante"); }
    terme
|   expression_simple
    OPADD
    error { yyerror("terme manquante"); }
;

terme:
    facteur
|   terme
    OPMUL
    facteur
|   error { yyerror("terme manquant"); }
    OPMUL
    facteur
|   terme
    error { yyerror("OPMUL manquant"); }
    facteur
|   terme
    OPMUL
    error { yyerror("facteur manquant"); }
;

```

```

facteur:
    ID
|    ID
    PARENTHESE_OUVRANTE
    liste_expressions
    PARENTHESE_FERMANTE
|    ID
    error { yyerror("parenthese ouvrante manquante"); }
    liste_expressions
    PARENTHESE_FERMANTE
|    ID
    PARENTHESE_OUVRANTE
    liste_expressions
    error { yyerror("parenthese fermante manquante"); }
|
    ID
    PARENTHESE_OUVRANTE

    PARENTHESE_FERMANTE
|    NB
|
    PARENTHESE_OUVRANTE
    expression
    PARENTHESE_FERMANTE
|    error { yyerror("parenthese ouvrante manquante"); }
    expression
|
    PARENTHESE_OUVRANTE
    expression
    error { yyerror("parenthese fermante manquante"); }
|
    PARENTHESE_OUVRANTE
    error { yyerror("expression manquante"); }
|
    PARENTHESE_FERMANTE
<
|    PARENTHESE_OUVRANTE
    error { yyerror("expression manquante"); }
    PARENTHESE_FERMANTE
|    NOT
    facteur
|    CHAINE
;

signe:
    '+'
|    '-'
;
;
%%

```

Partie définition de la grammaire du fichier pascal.y (3/4)

```
int yyerror(char const *msg) {  
  
    fprintf(stderr, "%s %d\n", msg,yylineno);  
    return 0;  
  
}  
  
extern FILE *yyin;  
  
int main()  
{  
    yyparse();  
    return 1 ;  
}
```

Partie fonctions en C du fichier pascal.y

NB: la ligne (`#include "lex.yy.c"`) permet d'inclure le fichier (.c) généré par *flex* pour qu'il soit compilé avec le fichier (.c) généré par *bison*, plus précisément, le fichier *lex.yy.c* contient la fonction *yylex()* vu dans la partie de l'analyse lexicale qui lit le texte en entrée caractère par caractère, reconnaît et retourne une suite d'unités lexicales, cette fonction sera appelée par la fonction *yyparse()* de l'analyseur syntaxique pour obtenir l'unité lexicale suivante. On peut remarquer que la signature de la fonction *yylex()* se trouve dans la première partie du fichier *bison* alors que son implémentation est dans le fichier *lex.yy.c*

A ce niveau, on doit apporter quelques modifications sur le fichier *flex* pour permettre la coopération entre l'analyseur lexicale et l'analyseur syntaxique: l'option `-d` de *bison* permet de générer un fichier (.h) contenant les différents symboles terminaux codés chacun par un nombre afin de les identifier, ce fichier doit être inclus dans le fichier spécification de l'analyseur lexicale, ensuite, à chaque identification d'une unité lexicale, on doit retourner son code correspondant pour qu'il soit reconnu par *bison*, et si le symbole terminal est écrit à la main (entre guillemets), dans le fichier (.l) on doit retourner le caractère lui-même.

```
/* Tokens. */
#define ID 258
#define NB 259
#define OPADD 260
#define OPAFFECT 261
#define OPMUL 262
#define OPREL 263
#define BEGIN 264
#define DO 265
#define ELSE 266
#define END 267
#define FUNCTION 268
#define IF 269
#define NOT 270
#define PROCEDURE 271
#define PROGRAM 272
#define THEN 273
#define VAR 274
#define WHILE 275
#define INT 276
```

Les différents codes de symboles terminaux dans le fichier *pascal.tab.h*

```
/* Définitions */

%{
    /* Code C */
    int outputLexical = 1; // flag qui permet d'afficher(1)/masquer(0)
    /* fonction qui affiche un message passé en paramètre */
    void output(const char* msg) {
        if(outputLexical == 1) {
            printf("Analyseur lexicale: %s\n", msg);
        }
    }
    /* chaîne de caractères utilisée avec la fonction output() */
    char buffer[50];

    #include "pascal.tab.h"
}
```

Partie définitions du fichier (.l) après modification
(Inclusion du fichier *pascal.tab.h*)

```

[bB][eE][gG][iI][nN]      { output("mot cle: BEGIN");      return BEGIN;      }
[dD][oO]                    { output("mot cle: DO");          return DO;          }
[eE][lL][sS][eE]            { output("mot cle: ELSE");        return ELSE;        }
[eE][nN][dD]                { output("mot cle: END");          return END;          }
[fF][uU][nN][cC][tT][iI][oO][nN] { output("mot cle: FUNCTION"); return FUNCTION; }
[iI][fF]                    { output("mot cle: IF");           return IF;           }
[iI][nN][tT]                { output("mot cle: INT");          return INT;          }
[pP][rR][oO][cC][eE][dD][uU][rR][eE] { output("mot cle: PROCEDURE"); return PROCEDURE; }
[pP][rR][oO][gG][rR][aA][mM] { output("mot cle: PROGRAM"); return PROGRAM; }
[tT][hH][eE][nN]            { output("mot cle: THEN");        return THEN;        }
[vV][aA][rR]                { output("mot cle: VAR");          return VAR;          }
[wW][hH][iI][lL][eE]        { output("mot cle: WHILE");       return WHILE;       }

```

Partie définitions et expressions régulières après modification

```

/*      nb      (nombres: chiffre chiffre*)
*/
{nb}    {
        sprintf(buffer, "NB: %s (%d caractere(s))", yytext, yyleng);
        output(buffer);
        return NB;
}

/*      id      (identificateurs)
*/
{id}    {
        sprintf(buffer, "ID: %s (%d caractere(s))", yytext, yyleng);
        output(buffer);
        return ID;
}

/*      oprel   (opérateurs relationnels: == <> < > <= >=)
*/
==|<>|<|>|<=|>= {
        sprintf(buffer, "OPREL: %s (%d caractere(s))", yytext, yyleng);
        output(buffer);
        return OPREL;
}

[nN][oO][tT] {
        sprintf(buffer, "OPREL/NOT: %s (%d caractere(s))", yytext, yyleng);
        output(buffer);
        return NOT;
}

```

```

/*      opadd      (+ - or)
*/
\+
|
| [oO][rR]
|
{
    sprintf(buffer, "OPADD: %s (%d caractere(s))", yytext, yyleng);
    output(buffer);
    return OPADD;
}

```

```

/*      opaffect   (=)
*/
=
{
    sprintf(buffer, "OPAFFFECT: %s (%d caractere(s))", yytext, yyleng);
    output(buffer);
    return OPAFFECT;
}

```

```

/*      opmul      (* / div mod and)
*/
\*
|
| \
| [dD][iI][vV]
| [mM][oO][dD]
| [aA][nN][dD]
|
{
    sprintf(buffer, "OPMUL: %s (%d caractere(s))", yytext, yyleng);
    output(buffer);
    return OPMUL;
}

```

```

/*      * caractères blancs: espace, tabulation, retour chariot
*/
{blanc}
{
    /* les caractères blancs sont à ignorer, on ne retourne rien */
}

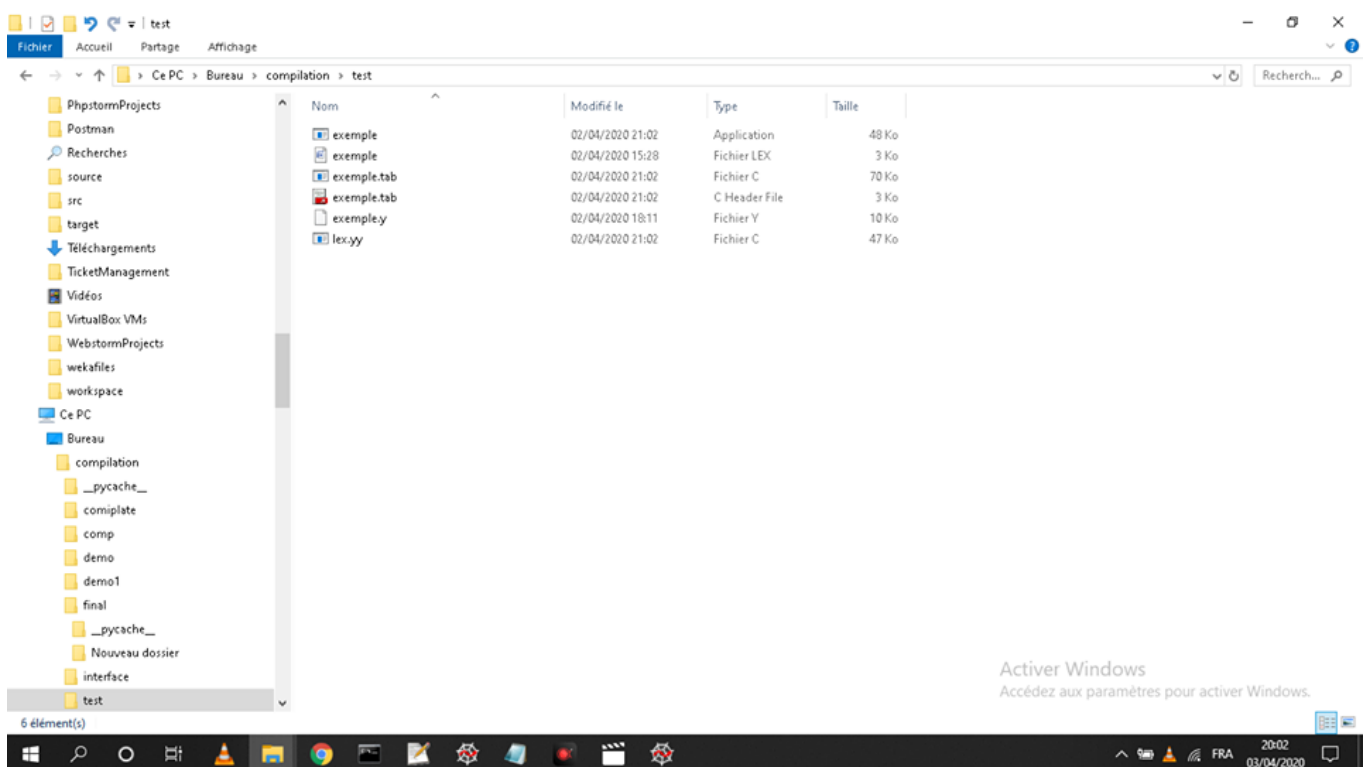
```

```

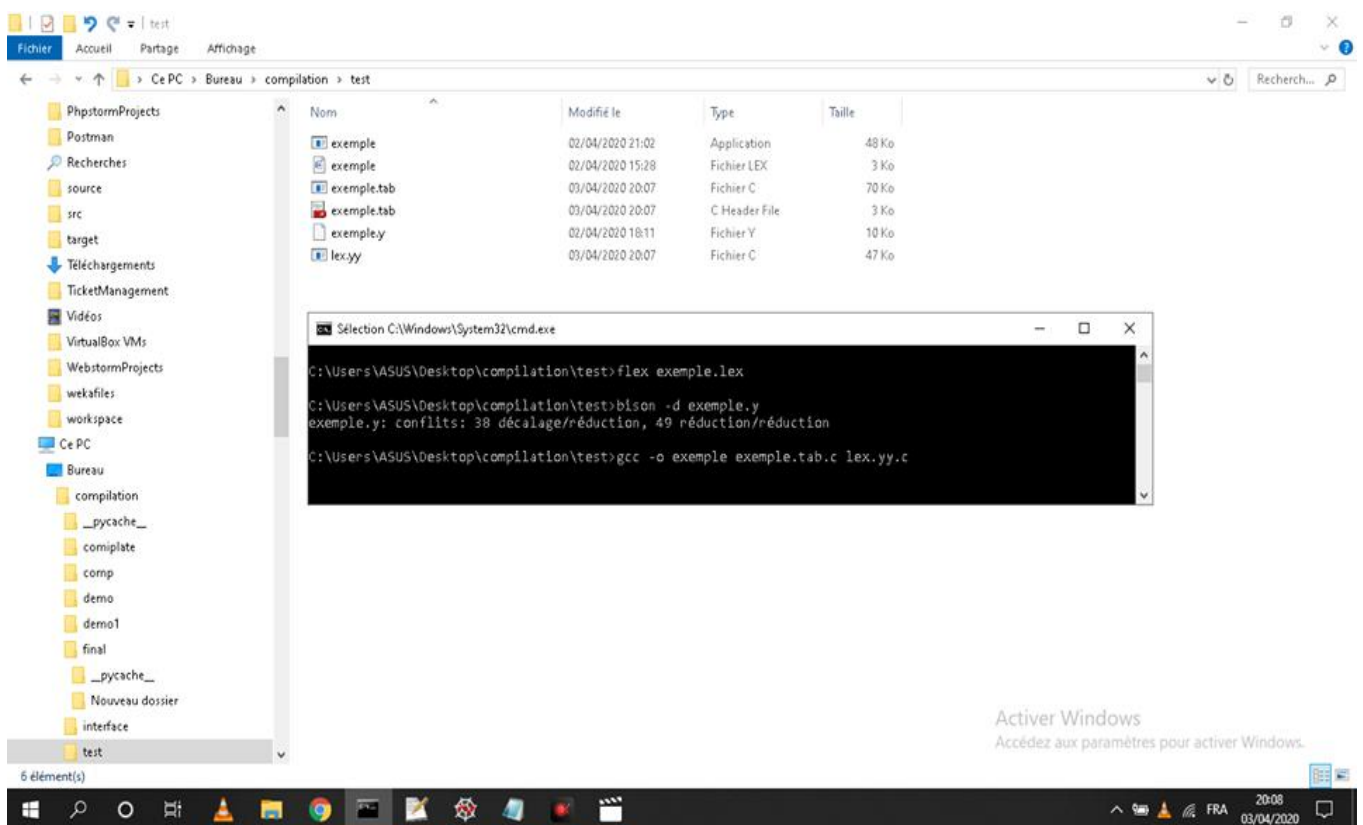
/*      * commentaires: placés entre { ... }
*/
{commentaire}
{
    sprintf(buffer, "COMMENTAIRE: %s (%d caractere(s))", yytext, yyleng);
    output(buffer);
    /* les commentaires sont à ignorer aussi */
}

```


les fichiers générés :

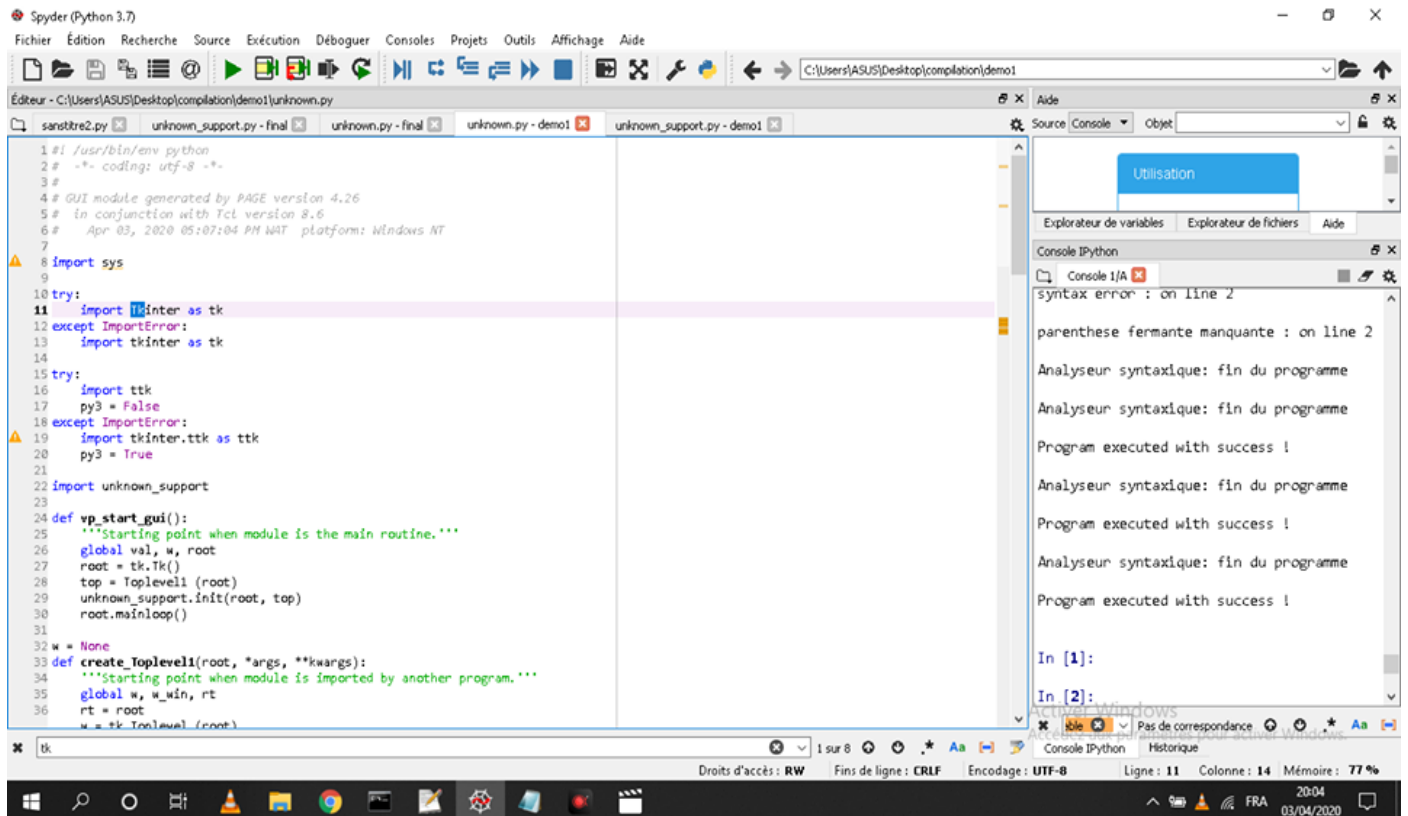


Ceci les etapes pour générer le fichier exemple .exe :



fichiers codes de l'interface :

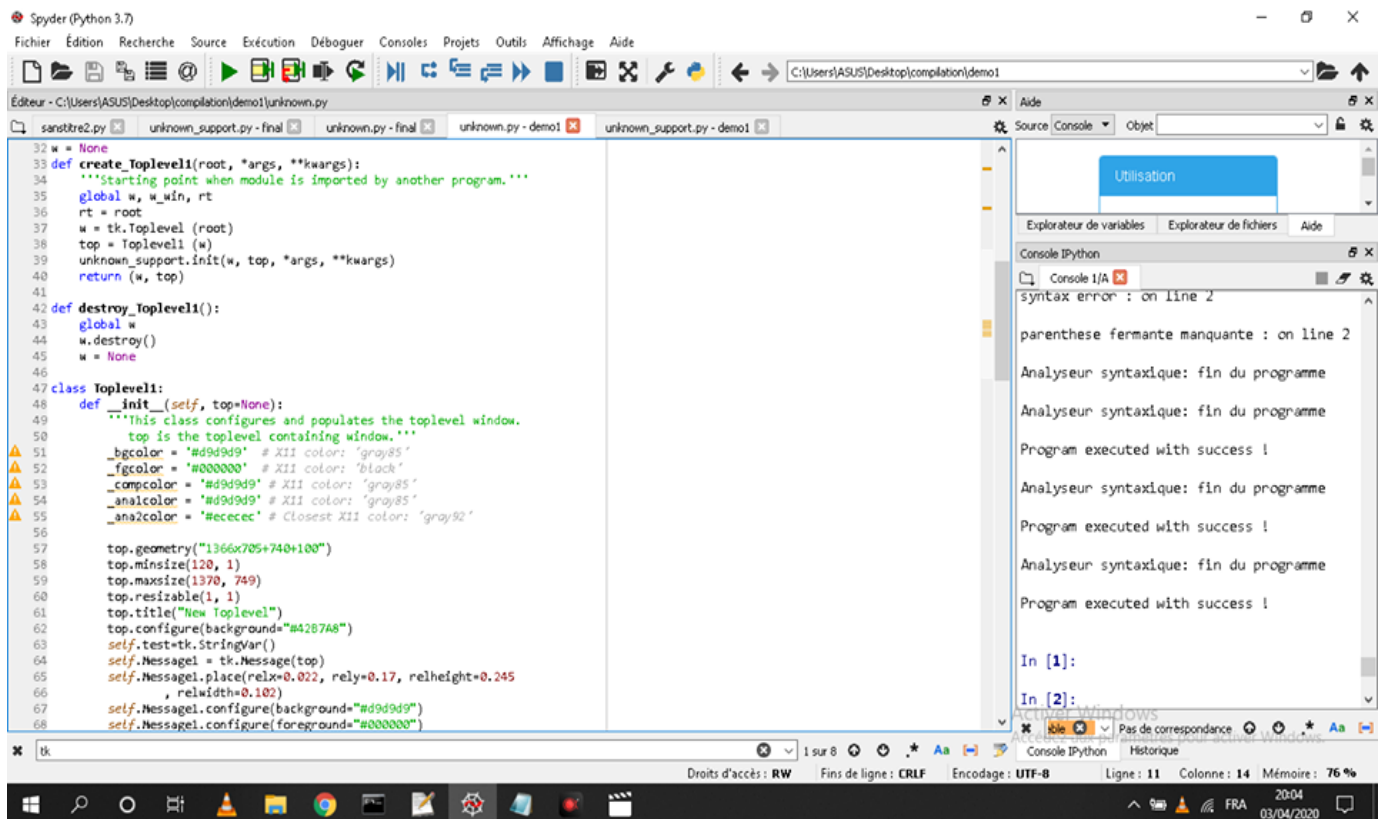
(notre interface a été codé en python) :



```
1 #! /usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 # GUI module generated by PAGE version 4.26
5 # in conjunction with Tcl version 8.6
6 # Apr 03, 2020 05:07:04 PM WAT platform: Windows NT
7
8 import sys
9
10 try:
11     import tkinter as tk
12 except ImportError:
13     import tkinter as tk
14
15 try:
16     import ttk
17     py3 = False
18 except ImportError:
19     import tkinter.ttk as ttk
20     py3 = True
21
22 import unknown_support
23
24 def vp_start_gui():
25     '''Starting point when module is the main routine.'''
26     global val, w, root
27     root = tk.Tk()
28     top = Toplevel(root)
29     unknown_support.init(root, top)
30     root.mainloop()
31
32 w = None
33 def create_Toplevel(root, *args, **kwargs):
34     '''Starting point when module is imported by another program.'''
35     global w, w_win, rt
36     rt = root
37     w = tk.Toplevel(root)
```

Console IPython

```
syntax error : on line 2
parenthese fermante manquante : on line 2
Analyseur syntaxique: fin du programme
Analyseur syntaxique: fin du programme
Program executed with success !
Analyseur syntaxique: fin du programme
Program executed with success !
Analyseur syntaxique: fin du programme
Program executed with success !
```



```
32 w = None
33 def create_Toplevel(root, *args, **kwargs):
34     '''Starting point when module is imported by another program.'''
35     global w, w_win, rt
36     rt = root
37     w = tk.Toplevel(root)
38     top = Toplevel(w)
39     unknown_support.init(w, top, *args, **kwargs)
40     return (w, top)
41
42 def destroy_Toplevel():
43     global w
44     w.destroy()
45     w = None
46
47 class Toplevel:
48     def __init__(self, top=None):
49         '''This class configures and populates the toplevel window.
50         top is the toplevel containing window.'''
51         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
52         _fgcolor = '#000000' # X11 color: 'black'
53         _compcolor = '#d9d9d9' # X11 color: 'gray85'
54         _anacolor = '#d9d9d9' # X11 color: 'gray85'
55         _ana2color = '#e0e0e0' # Closest X11 color: 'gray92'
56
57         top.geometry("1366x705+740+100")
58         top.minsize(120, 1)
59         top.maxsize(1370, 749)
60         top.resizable(1, 1)
61         top.title("New Toplevel")
62         top.configure(background="#d9d9d9")
63         self.test=tk.StringVar()
64         self.Message1 = tk.Message(top)
65         self.Message1.place(relx=0.022, rely=0.17, relheight=0.245
66                             , relwidth=0.102)
67         self.Message1.configure(background="#d9d9d9")
68         self.Message1.configure(foreground="#000000")
```

Console IPython

```
syntax error : on line 2
parenthese fermante manquante : on line 2
Analyseur syntaxique: fin du programme
Analyseur syntaxique: fin du programme
Program executed with success !
Analyseur syntaxique: fin du programme
Program executed with success !
Analyseur syntaxique: fin du programme
Program executed with success !
```

Spyder (Python 3.7)

Fichier Édition Recherche Source Exécution Débugger Consoles Projets Outils Affichage Aide

Éditeur - C:\Users\ASUS\Desktop\compilation\demo1\unknown.py

sanstire2.py unknown_support.py - final unknown.py - final unknown.py - demo1* unknown_support.py - demo1

```

89
90 self.Text2.configure(background="white")
91 self.Text2.configure(font="TkTextFont")
92 self.Text2.configure(foreground="black")
93 self.Text2.configure(highlightbackground="#d9d9d9")
94 self.Text2.configure(highlightcolor="black")
95 self.Text2.configure(insertbackground="black")
96 self.Text2.configure(selectbackground="#c4c4c4")
97 self.Text2.configure(selectforeground="black")
98 self.Text2.configure(wrap="word")
99
100 def update(event):
101
102     self.Text2.delete(1.0,100.0)
103     self.test=unknown_support.compiler(self.test)
104     print(self.test)
105     self.Text2.insert('insert',self.test)
106
107
108
109
110
111 self.Button1 = tk.Button(top)
112 self.Button1.place(relx=0.447, rely=0.44, height=24, width=197)
113 self.Button1.configure(activebackground="#e6e6ec")
114 self.Button1.configure(activeforeground="#000000")
115 self.Button1.configure(background="#f6f6f6")
116 self.Button1.bind("<Button-1>", update)
117 self.Button1.configure(command=lambda:update)
118 self.Button1.configure(disabledforeground="#a3a3a3")
119 self.Button1.configure(foreground="#000000")
120 self.Button1.configure(highlightbackground="#d9d9d9")
121 self.Button1.configure(highlightcolor="black")
122 self.Button1.configure(pady="0")
123 self.Button1.configure(text='''compiler''')
124 #self.Button1.event_add("<button>",update)
125 #self.Button1.bind("<button>", update)

```

tk

1 sur 8 Fins de ligne : CRLF Encodage : UTF-8 Ligne : 89 Colonne : 1 Mémoire : 76 %

Spyder (Python 3.7)

Fichier Édition Recherche Source Exécution Débugger Consoles Projets Outils Affichage Aide

Éditeur - C:\Users\ASUS\Desktop\compilation\demo1\unknown.py

sanstire2.py unknown_support.py - final unknown.py - final unknown.py - demo1* unknown_support.py - demo1

```

61 top.title("New TopLevel")
62 top.configure(background="#42b7a8")
63 self.test=tk.StringVar()
64 self.Message1 = tk.Message(top)
65 self.Message1.place(relx=0.022, rely=0.17, relheight=0.245
66 , relwidth=0.102)
67 self.Message1.configure(background="#d9d9d9")
68 self.Message1.configure(foreground="#000000")
69 self.Message1.configure(highlightbackground="#d9d9d9")
70 self.Message1.configure(highlightcolor="black")
71 self.Message1.configure(text='''Message''')
72 self.Message1.configure(width=140)
73
74 self.Text1 = tk.Text(top)
75 self.Text1.place(relx=0.007, rely=0.156, relheight=0.616, relwidth=0.42)
76 self.Text1.configure(background="white")
77 self.Text1.configure(font="TkTextFont")
78 self.Text1.configure(foreground="black")
79 self.Text1.configure(highlightbackground="#d9d9d9")
80 self.Text1.configure(highlightcolor="black")
81 self.Text1.configure(insertbackground="black")
82 self.Text1.configure(selectbackground="#c4c4c4")
83 self.Text1.configure(selectforeground="black")
84 self.Text1.configure(wrap="word")
85
86 self.Text2 = tk.Text(top)
87 self.Text2.place(relx=0.608, rely=0.156, relheight=0.573, relwidth=0.391)
88
89
90 self.Text2.configure(background="white")
91 self.Text2.configure(font="TkTextFont")
92 self.Text2.configure(foreground="black")
93 self.Text2.configure(highlightbackground="#d9d9d9")
94 self.Text2.configure(highlightcolor="black")
95 self.Text2.configure(insertbackground="black")
96 self.Text2.configure(selectbackground="#c4c4c4")
97 self.Text2.configure(selectforeground="black")

```

tk

1 sur 8 Fins de ligne : CRLF Encodage : UTF-8 Ligne : 89 Colonne : 1 Mémoire : 75 %

Utilisation

Explorateur de variables Explorateur de fichiers Aide

Console IPython

Console 1/A

syntax error : on line 2

parenthese fermante manquante : on line 2

Analyseur syntaxique: fin du programme

Analyseur syntaxique: fin du programme

Program executed with success !

Analyseur syntaxique: fin du programme

Program executed with success !

Analyseur syntaxique: fin du programme

Program executed with success !

In [1]:

In [2]:

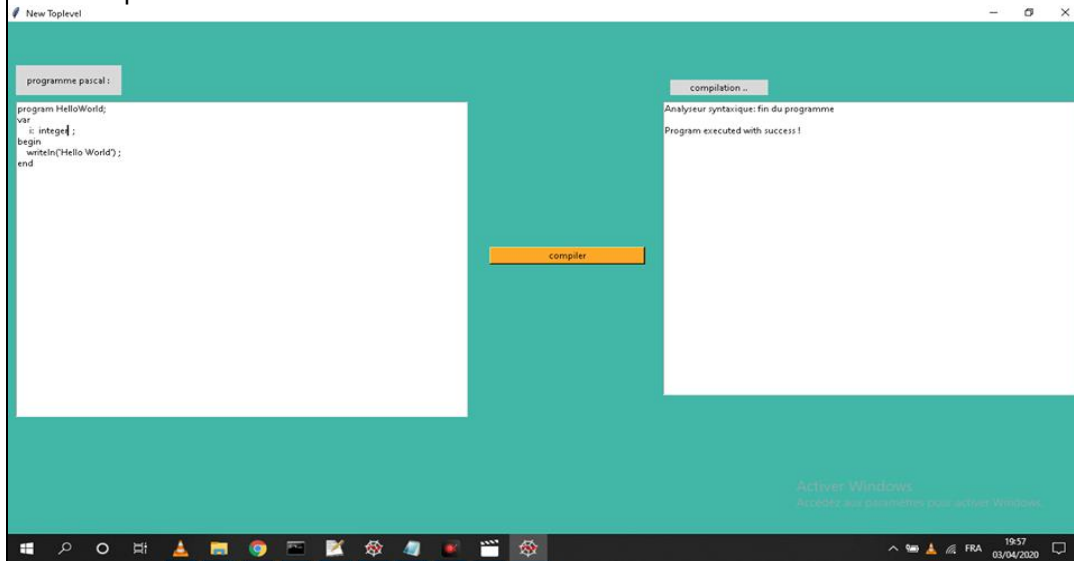
Pas de correspondance

Historique

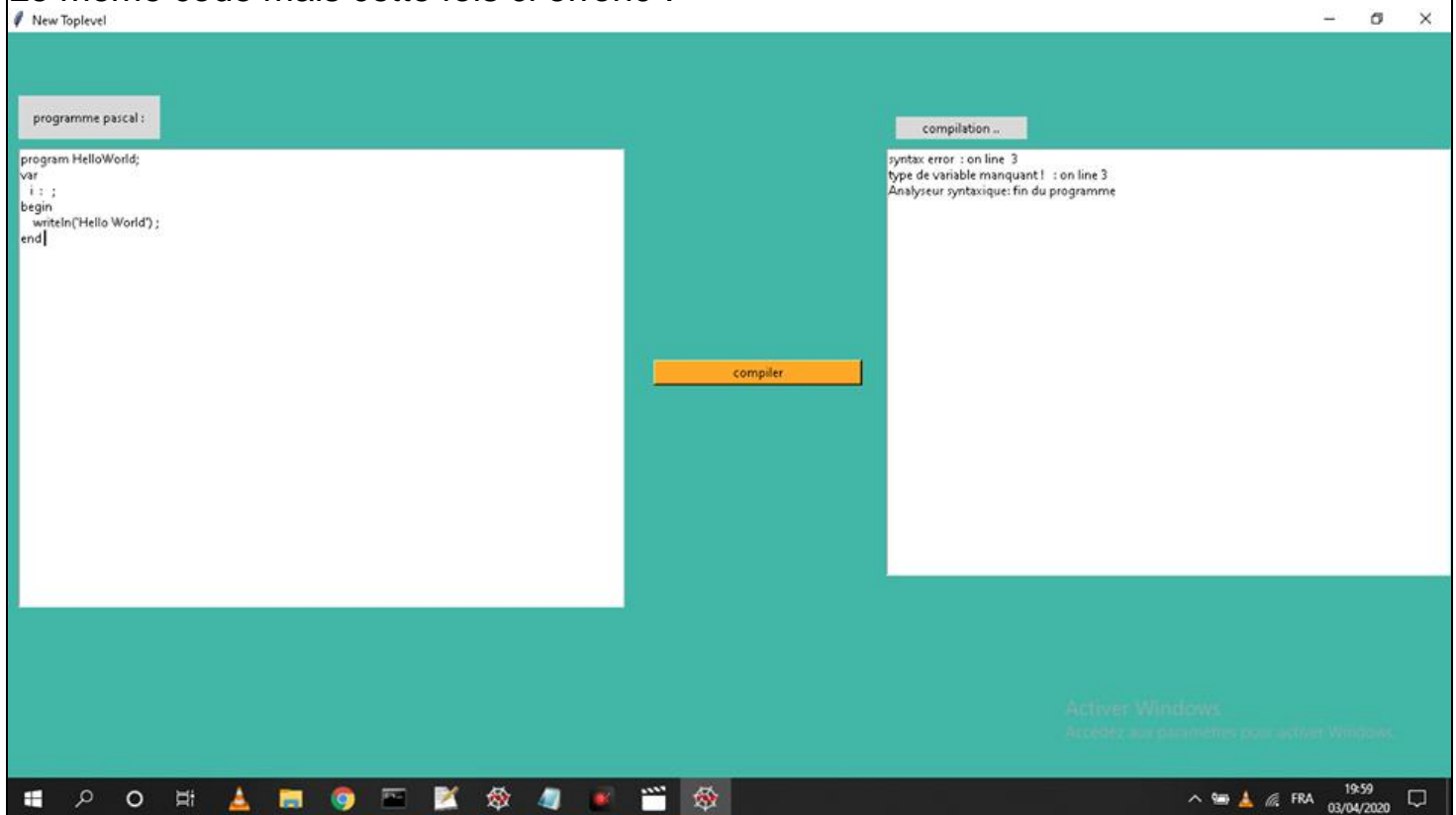
03/04/2020

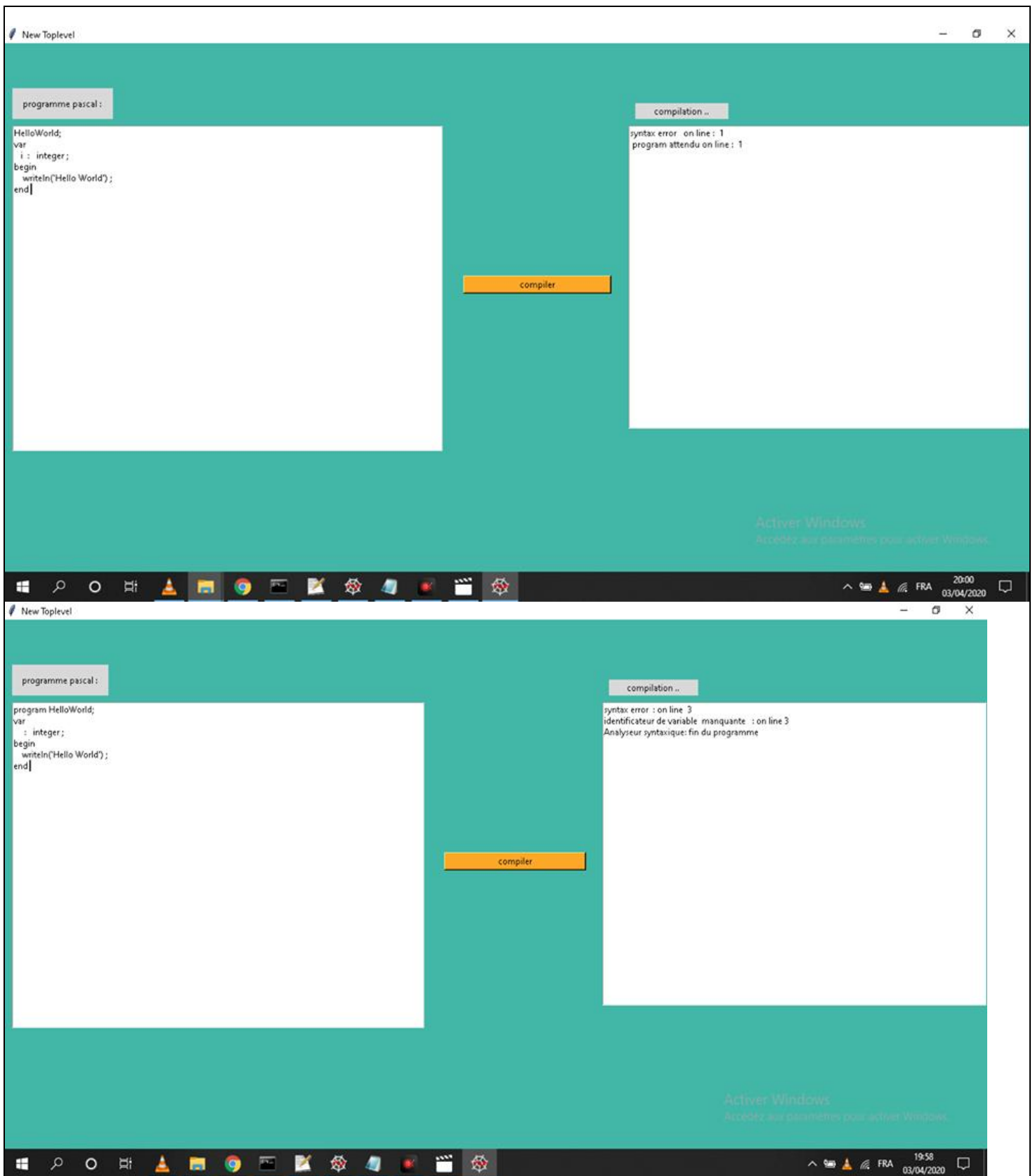
codes et leurs résultats de compilations:

Un code qui est correcte :



Le meme code mais cette fois ci erroné :





On a montré d'autres codes avec leur compilations dans notre video ,

Merci pour votre attention

