

La transpilation de Fortran vers C

Présentation de **Erwan FALAUX-BACHELOT**

travail réalisé avec **Yann MIQUEL-ERDMANN**

Introduction

Le Fortran : 

Créé par : IMB en 1954

Utilisé surtout dans les supercalculateurs

Problématique

Comment implémenter une conversion rapide de programmes Fortran en programmes C ?

1. Analyse Lexicale

Les expressions régulières

Les automates

La détermination

2. Analyse Syntaxique

3. Conversion vers la syntaxe abstraite

4. Traduction vers le langage de sortie

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset, \varepsilon, a \in \Sigma$$

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les expressions régulières

- Expressions régulières

définies inductivement sur :

- Automates

$$\emptyset, \varepsilon, a \in \Sigma$$

avec les règles usuelles :

$$\cdot, |, *$$

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset, \varepsilon, a \in \Sigma$$

avec les règles usuelles :

$$\cdot, |, *$$

et des additionnelles :

$$+, ?, [a - z], \sim$$

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset, \varepsilon, a \in \Sigma$$

avec les règles usuelles :

$$\cdot, |, *$$

et des additionnelles :

$$+, ?, [a - z], \sim$$

exemple :

$$[0 - 9]^+ ([0 - 9]^+)^?$$

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset, \varepsilon, a \in \Sigma$$

avec les règles usuelles :

$$\cdot, |, *$$

et des additionnelles :

$$+, ?, [a - z], \sim$$

exemple :

[0 - 9]+ (. [0 - 9])+

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset, \varepsilon, a \in \Sigma$$

avec les règles usuelles :

$$\cdot, |, *$$

et des additionnelles :

$$+, ?, [a - z], \sim$$

exemple :

$[0 - 9]^+ ([\cdot [0 - 9]^+])^?$

Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les automates

- Expressions régulières
- Automates

$(\Sigma, Q, I, F, \delta)$

Analyse Lexicale

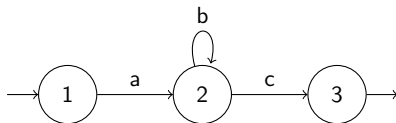
Analyse Syntaxique

Syntaxe Abstraite

Conversion

Les automates

- Expressions régulières
- Automates

 $(\Sigma, Q, I, F, \delta)$ automate pour $a \cdot b^* \cdot c$

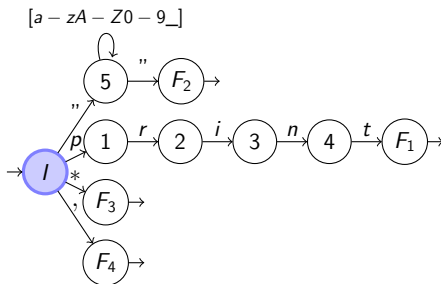
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

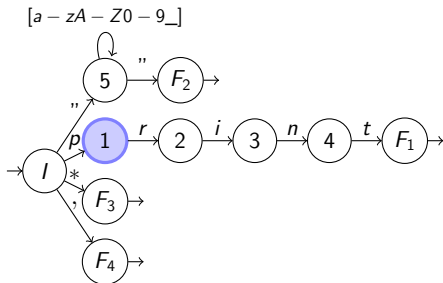
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

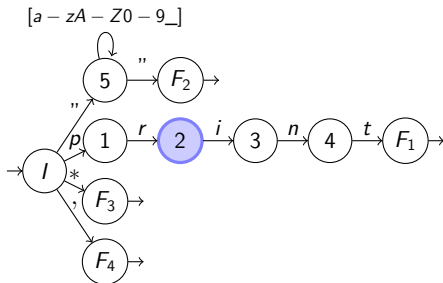
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

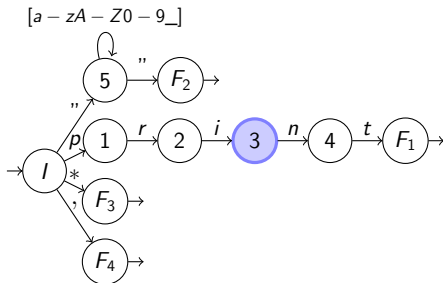
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

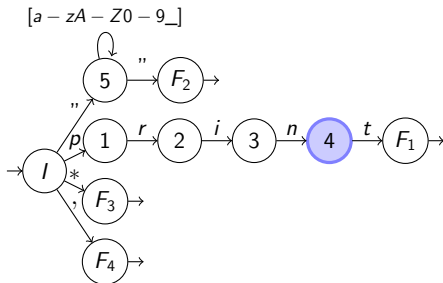
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

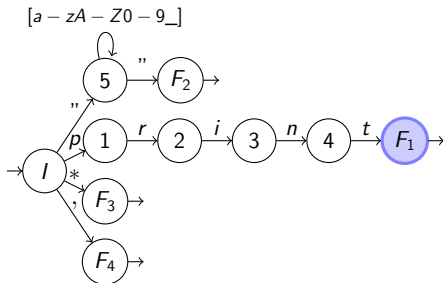

Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

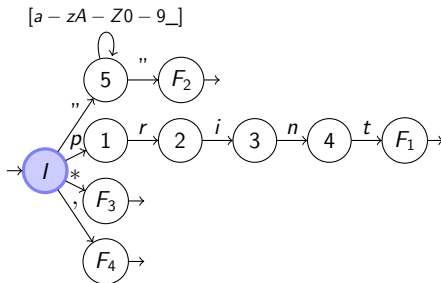
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ ]
```

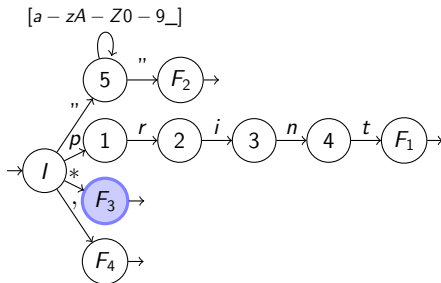
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' ]
```

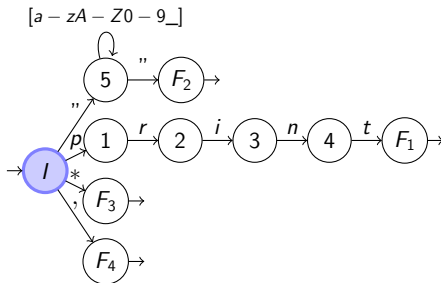
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' ]
```

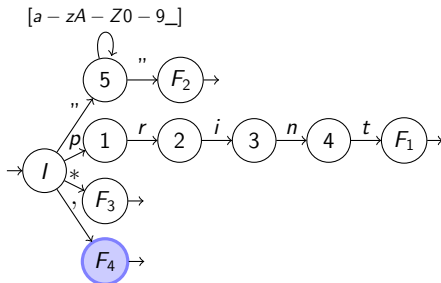
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' ]
```

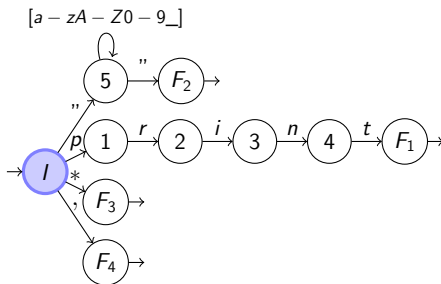
Les automates : étude d'un exemple



```
print*,["Hello World"]
```

```
res = [ 'print' , '*' ]
```

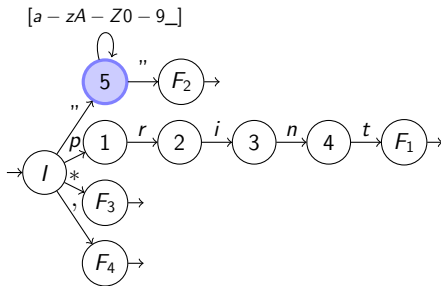
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

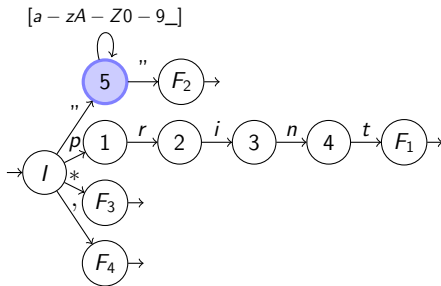
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

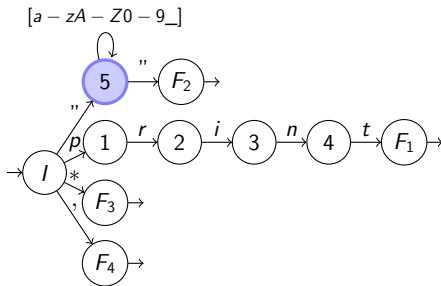

Les automates : étude d'un exemple



```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

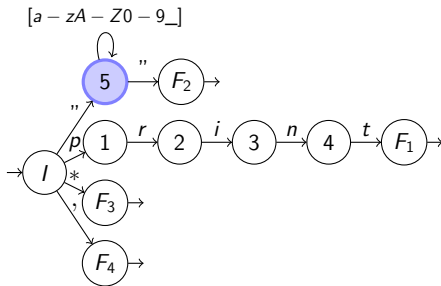
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

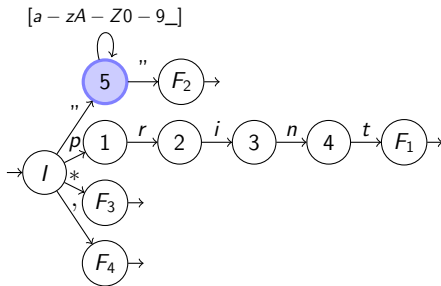
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

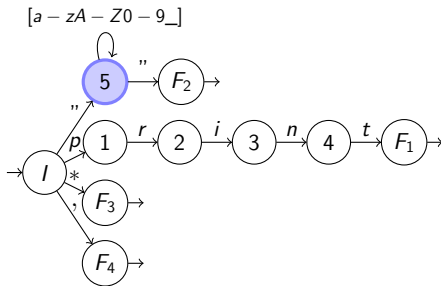
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

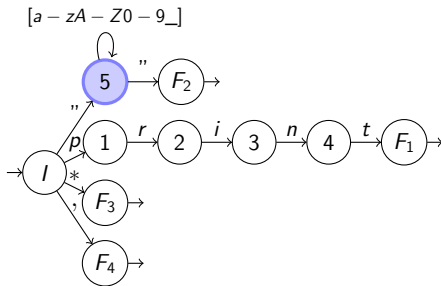
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

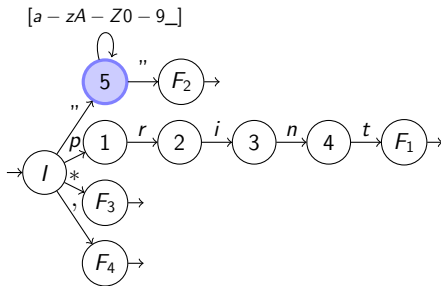
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

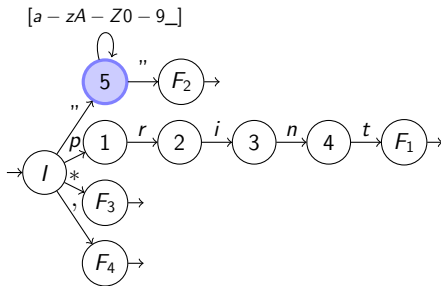
Les automates : étude d'un exemple



```
print*, "Hello Wprld"
```

```
res = [ 'print' , '*' , ',' ]
```

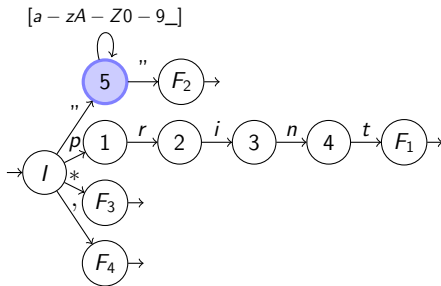
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

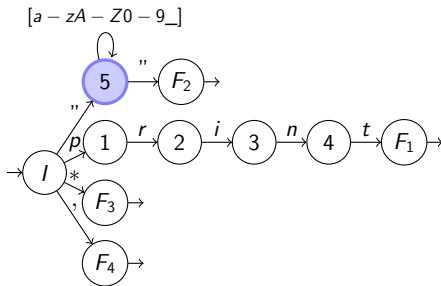

Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

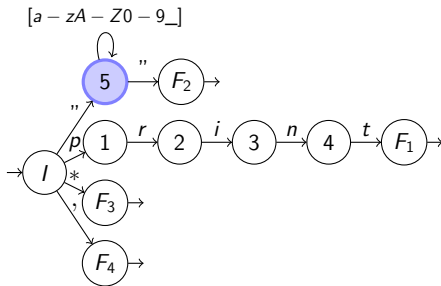
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

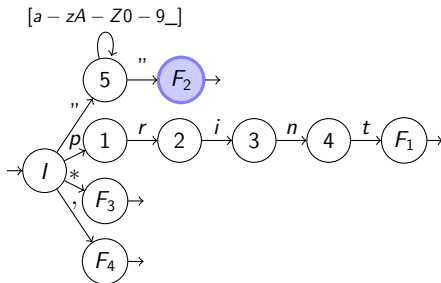
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

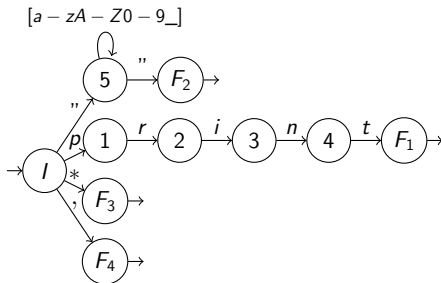
Les automates : étude d'un exemple



```
print*, "Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

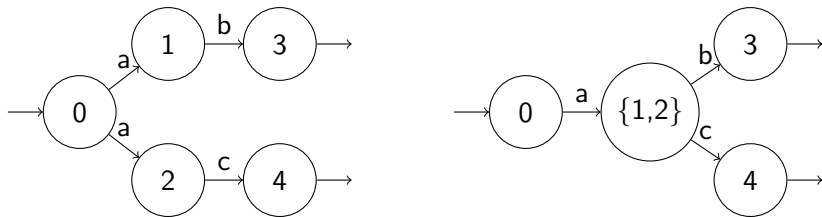
Les automates : étude d'un exemple



```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' , '"Hello World"' ]
```

La détermination



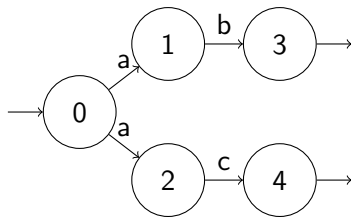
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

La détermination



```
1  type automate = {  
2      nodes : int list;  
3      debut_l : int list;  
4      fin : (int * terminal)  
        ↳ list;  
5      transitions : (char  
        ↳ option * int) list  
        ↳ array;  
6  }
```

Analyse Lexicale

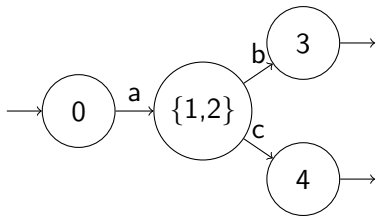
Analyse Syntaxique

Syntaxe Abstraite

Conversion

La détermination

```
1  type automate_det = {  
2    nodes : int list;  
3    debut : int;  
4    fin : terminal option  
    ↪ array;  
5    transitions : int array  
    ↪ array;  
6  }
```



Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

1. Analyse Lexicale

2. Analyse Syntaxique

La grammaire

L'algorithme LL1

3. Conversion vers la syntaxe abstraite

4. Traduction vers le langage de sortie

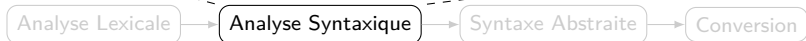
Analyse Syntaxique

- Grammaire
- LL1

Règles de production :

$A \rightarrow B$

$A \rightarrow c, c \in \Sigma^*$



Analyse Syntaxique

- Grammaire
- LL1

Règles de production :

$A \rightarrow B$

$A \rightarrow c, c \in \Sigma^*$

Exemple :

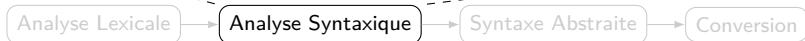
$S \rightarrow A$

$A \rightarrow aA$

$A \rightarrow B$

$B \rightarrow bB$

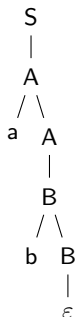
$B \rightarrow \text{epsilon}$



Analyse Syntaxique

- Grammaire

- LL1



Arbre de dérivation de ab

Règles de production :

$A \rightarrow B$

$A \rightarrow c, c \in \Sigma^*$

Exemple :

$S \rightarrow A$

$A \rightarrow aA$

$A \rightarrow B$

$B \rightarrow bB$

$B \rightarrow \text{epsilon}$

Analyse Lexicale

Analyse Syntaxique

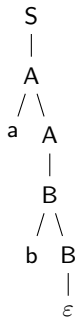
Syntaxe Abstraite

Conversion

• Grammaire

• LL1

[a, b] + $\begin{array}{l} S \rightarrow A \\ A \rightarrow aA \\ A \rightarrow B \\ B \rightarrow bB \\ B \rightarrow \text{epsilon} \end{array}$ \Rightarrow



Analyse Lexicale

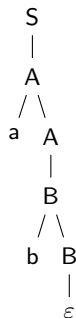
Analyse Syntaxique

Syntaxe Abstraite

Conversion

1. Analyse Lexicale
2. Analyse Syntaxique
3. Conversion vers la syntaxe abstraite
 - Fonctionnement
 - Un exemple
4. Traduction vers le langage de sortie

Syntaxe abstraite



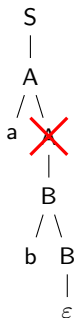
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite



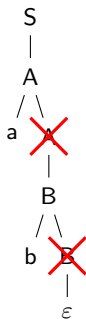
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite



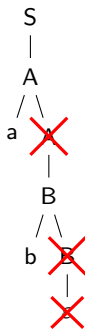
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite



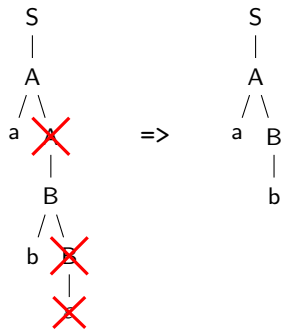
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite



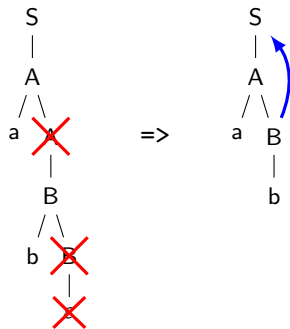
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite



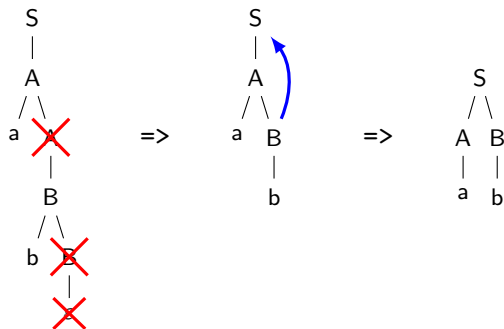
Analyse Lexicale

Analyse Syntaxique

Syntaxe Abstraite

Conversion

Syntaxe abstraite

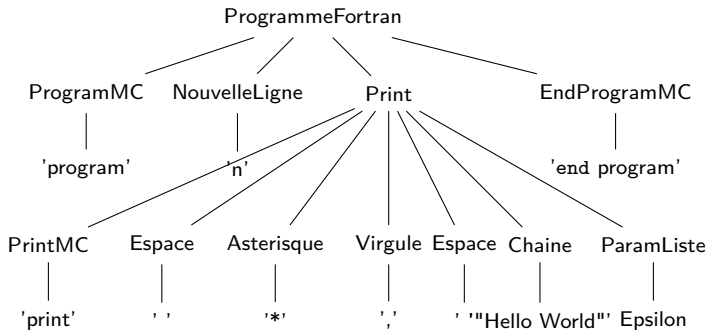


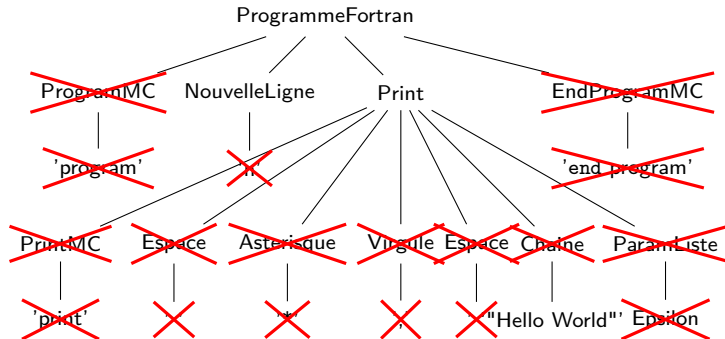
Analyse Lexicale

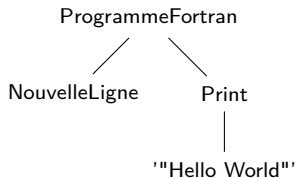
Analyse Syntaxique

Syntaxe Abstraite

Conversion







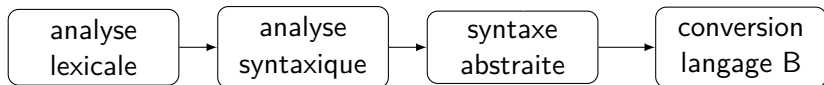
1. Analyse Lexicale
2. Analyse Syntaxique
3. Conversion vers la syntaxe abstraite
4. Traduction vers le langage de sortie

Traduction

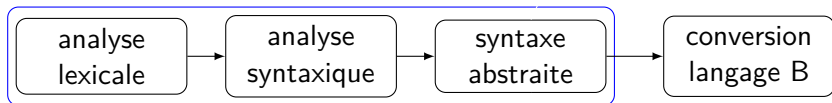
parcours en profondeur
conversion en chaîne



Transpileur

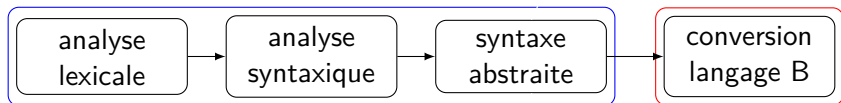


Transpileur



module du langage d'entrée A

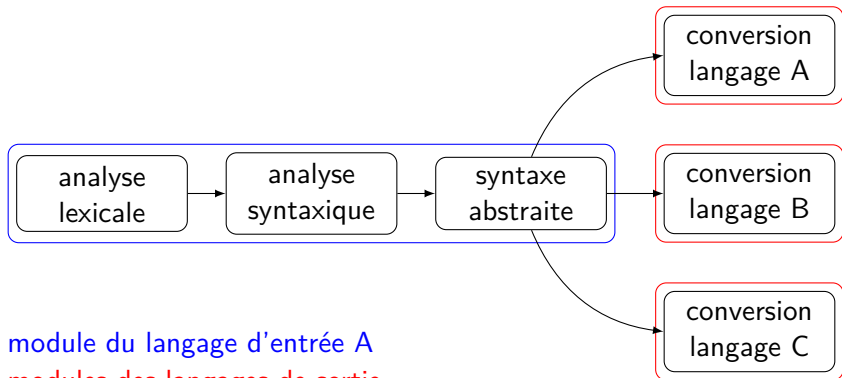
Transpileur



module du langage d'entrée A

modules du langage de sortie

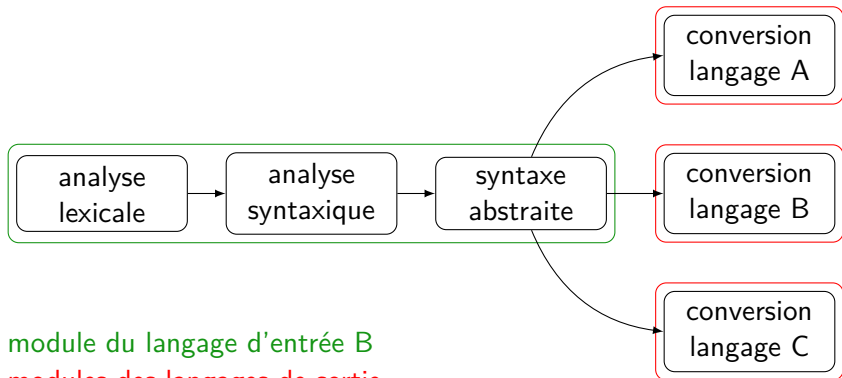
Transpileur



module du langage d'entrée A

modules des langages de sortie

Transpileur



module du langage d'entrée B

modules des langages de sortie

Conclusion

- Conversion rapide
- Partie automatisée rend la création moins pénible
- Processus interchangeable avec les langages souhaités

5- Annexe

```
1  type operateur = Plus | Moins | Foix | Division |  
   ↪  Puissance | Assignment  
2  
3  type compareur =  
4    | PlusPetit  
5    | PlusGrand  
6    | StrictPlusPetit  
7    | StrictPlusGrand  
8    | Egal  
9    | NonEgal  
10  
11 type operateur_logique = Et | Ou | Non | Equivalent  
   ↪  | NonEquivalent  
12  
13 type syntax =  
14    | Any  
15    | Character
```

5- Annexe

16		Complex
17		Constant
18		Do
19		Double_precision
20		Else
21		Else_if
22		Function
23		If
24		End_if
25		Integer
26		Logical
27		Program
28		Real
29		While
30		Call
31		Print
32		Return

5- Annexe

```
33 | Size
34 | For
35 | Step
36 | (* v---only for post-processing---v *)
37 | Out
38 | Subroutine
39
40 type token =
41 | Operateur of operateur
42 | Compareteur of compareteur
43 | OperateurLogique of operateur_logique
44 | Syntax of syntax
45 | NewLine
46 | Name of string
47 | Integer of string
48 | Floating of string
49 | Double of string
```

5- Annexe

```
50 |   Chaîne of string
51 |   Commentaire of string
52 |   Booleen of bool
53 |   Parentheseouvrante
54 |   Parenthesefermante
55 |   ProgramRoot
56 |   ToFlatten
57
58 |   (* paramètres et enfants confondus dans la liste
59 |   ↪ *)
60
61 |   type ast = Noeud of token * ast list
62
63 |   let string_of_token (t : token) : string =
64 |     match t with
65 |     | Operateur Plus -> "Operateur Plus"
66 |     | Operateur Moins -> "Operateur Moins"
67 |     | Operateur Fois -> "Operateur Fois"
```

5- Annexe

```
66 | Operateur Division -> "Operateur Division"
67 | Operateur Puissance -> "Operateur Puissance"
68 | Operateur Assignment -> "Operateur
   ↳ Assignment"
69 | Comparateur PlusPetit -> "Comparateur
   ↳ PlusPetit"
70 | Comparateur PlusGrand -> "Comparateur
   ↳ PlusGrand"
71 | Comparateur StrictPlusPetit -> "Comparateur
   ↳ StrictPlusPetit"
72 | Comparateur StrictPlusGrand -> "Comparateur
   ↳ StrictPlusGrand"
73 | Comparateur Egal -> "Comparateur Egal"
74 | Comparateur NonEgal -> "Comparateur NonEgal"
75 | OperateurLogique Et -> "OperateurLogique Et"
76 | OperateurLogique Ou -> "OperateurLogique Ou"
77 | OperateurLogique Non -> "OperateurLogique Non"
```

5- Annexe

```
78 | OperateurLogique Equivalent ->  
   | ↪ "OperateurLogique Equivalent"  
79 | OperateurLogique NonEquivalent ->  
   | ↪ "OperateurLogique NonEquivalent"  
80 | Syntax Any -> "Syntax Any"  
81 | Syntax Character -> "Syntax Character"  
82 | Syntax Complex -> "Syntax Complex"  
83 | Syntax Constant -> "Syntax Constant"  
84 | Syntax Do -> "Syntax Do"  
85 | Syntax Double_precision -> "Syntax  
   | ↪ Double_precision"  
86 | Syntax Else -> "Syntax Else"  
87 | Syntax Else_if -> "Syntax Else_if"  
88 | Syntax Function -> "Syntax Function"  
89 | Syntax If -> "Syntax If"  
90 | Syntax End_if -> "Syntax End_if"  
91 | Syntax Out -> "Syntax Out"
```

5- Annexe

```
92 | Syntax Integer -> "Syntax Integer"
93 | Syntax Logical -> "Syntax Logical"
94 | Syntax Program -> "Syntax Program"
95 | Syntax Real -> "Syntax Real"
96 | Syntax Subroutine -> "Syntax Subroutine"
97 | Syntax While -> "Syntax While"
98 | Syntax Call -> "Syntax Call"
99 | Syntax Print -> "Syntax Print"
100 | Syntax Return -> "Syntax Return"
101 | Syntax Size -> "Syntax Size"
102 | Syntax For -> "Syntax For"
103 | Syntax Step -> "Syntax Step"
104 | NewLine -> "NewLine"
105 | Name s -> "Name " ^ s
106 | Integer s -> "Integer " ^ s
107 | Floating s -> "Floating " ^ s
108 | Double s -> "Double " ^ s
```

5- Annexe

```
109 | Chaîne s -> "Chaîne " ^ s
110 | Commentaire s -> "Commentaire " ^ s
111 | Booleen b -> "Booleen " ^ string_of_bool b
112 | Parentheseouvrante -> "Parentheseouvrante"
113 | Parenthesefermante -> "Parenthesefermante"
114 | ProgramRoot -> "ProgramRoot"
115 | ToFlatten -> "ToFlatten"
116
117 let print_token (t : token) : unit = print_endline
    ↪ (string_of_token t)
```


5- Annexe

```
1  type libs = string list
2
3  let add_lib (l : libs) (name : string) : libs =
4    if not (List.mem name l) then name :: l else l
```

5- Annexe

```
1  open Abstract_tokens
2
3  type environnement = (string, token) Hashtbl.t
4  type intent = In | Out | InOut
5  type var_type_subroutine = (string, intent)
   ↪   Hashtbl.t
6
7  let print_env (env: environnement): unit =
8      Hashtbl.iter (fun k v -> print_string k ;
   ↪   print_string " -> " ; print_token v;
   ↪   print_newline ()) env
9
10
11 let rec last_of_list (l: 'a list): 'a =
12     match l with
13     | [] -> failwith "Liste vide "
14     | e::[] -> e
```

5- Annexe

```

15 | e::q -> last_of_list q
16
17
18 | (** crée un environnement à partir de l'ast [t] *)
19 let create_env_from_ast (t : ast) : environnement =
20   let env = Hashtbl.create 0 in
21   (** ajoute récursivement sur [t] les variables
22   ↪ dans l'environnement *)
23   let rec add_env (t : ast) : unit =
24     match t with
25     | Noeud (Syntax Double_precision, [ Noeud (Name
26       ↪ s, []) ]) )
27     | Noeud
28       ( Syntax Double_precision,
29         [ Noeud (Opérateur Assignment, [ Noeud
30           ↪ (Name s, []); _ ]) ] ) ->
31       Hashtbl.add env s (Syntax Double_precision)

```

5- Annexe

```

29 | Noeud (Syntax Integer, [ Noeud (Name s, [])
    | ↪ ])
30 | Noeud
31 |   ( Syntax Integer,
32 |     [ Noeud (Opérateur Assignment, [ Noeud
    |     ↪ (Name s, []); _ ]) ] ) ->
33 |     Hashtbl.add env s (Syntax Integer)
34 | Noeud (Syntax Real, [ Noeud (Name s, []) ])
35 | Noeud
36 |   ( Syntax Real,
37 |     [ Noeud (Opérateur Assignment, [ Noeud
    |     ↪ (Name s, []); _ ]) ] ) ->
38 |     Hashtbl.add env s (Syntax Real)
39 | Noeud (Syntax Logical, [ Noeud (Name s, [])
    | ↪ ])
40 | Noeud
41 |   ( Syntax Logical,

```

5- Annexe

```

42      [ Noeud (Opérateur Assignment, [ Noeud
      ↪ (Name s, []); _ ]) ] ) ->
43      Hashtbl.add env s (Syntax Real)
44      | Noeud (Syntax Character, [ Noeud (Syntax
      ↪ Size, _); Noeud (Name s, []) ])
45      | Noeud (Syntax Character, [ Noeud (Name s, [])
      ↪ ])
46      | Noeud
47        ( Syntax Character,
48          [
49            Noeud
50              ( Opérateur Assignment,
51                [ Noeud (Syntax Size, _); Noeud
                  ↪ (Name s, []); _ ] );
52          ] )
53      | Noeud
54        ( Syntax Character,

```

5- Annexe

```

55         [ Noeud (Opérateur Assignment, [ Noeud
           ↪ (Name s, []); _ ]) ] ) ->
56         Hashtbl.add env s (Syntax Real)
57     | Noeud (Syntax Function, Noeud(Name n, []))::l)
       ↪ ->(
58         List.iter add_env l;
59         match last_of_list l with
60         | Noeud(Syntax Return , Noeud(Name
           ↪ v, []))::_) -> Hashtbl.add env n
           ↪ (Hashtbl.find env v)
61         | _ -> () (* pas de return à la fin de la
           ↪ fct donc c'est une subroutine *)
62     )
63
64     | Noeud (_, l) -> List.iter add_env l
65 in
66 add_env t;

```

5- Annexe

```
67   env
68
69   (** déduis de l'utilisation des variables dans [t]
   ↪ si elles sont in, out out *)
70   let create_subroutine_intent (t : ast) :
   ↪ var_type_subroutine =
71     let sub_intent = Hashtbl.create 0 in
72     (** traite récursivement les variables sur [t]
   ↪ *)
73     let rec subroutine_intent_aux (t : ast) : unit =
74       (* TODO ajouter lorsque la syntaxe abstraite
   ↪ pour le in/out/inout est ok *)
75       match t with
76       | Noeud (Opérateur Assignment, Noeud (Name s,
   ↪ [])) :: _
77         when not (Hashtbl.mem sub_intent s) ->
78           Hashtbl.add sub_intent s Out
```

5- Annexe

```

79 | Noeud (Opérateur Assignment, Noeud (Name s,
    ↳ []) :: _)
80 |   when Hashtbl.find sub_intent s = In ->
81 |     Hashtbl.replace sub_intent s InOut
82 | Noeud (Opérateur Assignment, Noeud (Name s,
    ↳ []) :: _)
83 |   when Hashtbl.find sub_intent s = Out ||
    ↳   Hashtbl.find sub_intent s = InOut
84 |   ->
85 |     ()
86 | Noeud (Name s, []) when not (Hashtbl.mem
    ↳ sub_intent s) ->
87 |   Hashtbl.add sub_intent s In
88 | Noeud (Name s, []) when Hashtbl.find
    ↳ sub_intent s = Out ->
89 |   Hashtbl.add sub_intent s InOut
90 | Noeud (Name s, [])

```


5- Annexe

```
91      when Hashtbl.find sub_intent s = Out ||  
          ↪ Hashtbl.find sub_intent s = InOut  
92      ->  
93      ()  
94      | Noeud (_, l) -> List.iter  
          ↪ subroutine_intent_aux l  
95  
96      in  
97      subroutine_intent_aux t;  
98      sub_intent
```

5- Annexe

```
1  type terminal =
2      | EOF
3      | E
4      | PowerOp
5      | NotOp
6      | AndOp
7      | OrOp
8      | Dcon
9      | Rcon
10     | Icon
11     | SconSingle
12     | SconDouble
13     | Ident
14     | EOS
15     | Return
16     | Result
17     | Contains
```

5- Annexe

18		True
19		False
20		Program
21		Function
22		Subroutine
23		EndProgram
24		EndFunction
25		EndSubroutine
26		EndDo
27		EndIf
28		Colon
29		Comma
30		Equal
31		Asterisk
32		LParenthesis
33		RParenthesis
34		Integer

5- Annexe

35		Real
36		Double
37		Complex
38		Character
39		Logical
40		Parameter
41		Intent
42		In
43		Out
44		InOut
45		Call
46		Print
47		Do
48		While
49		If
50		Else
51		Then

5- Annexe

```
52 | Divide
53 | Plus
54 | Minus
55 | IsEqual
56 | NotEqual
57 | StrictLess
58 | LessEqual
59 | StrictGreater
60 | GreaterEqual
61 | Equivalent
62 | NotEquivalent
63 | Space
64 | Recursive
65
66 type non_terminal =
67 | ExecutableProgram
68 | StartCommentBlock
```

5- Annexe

69		Function_or_Subroutine_star_MainProgram
70		Function_or_Subroutine_star
71		Recursive_opt_Function_or_Subroutine
72		Function_or_Subroutine
73		MainProgram
74		MainRange
75		Contains_Function_opt_EndProgramStmt
76		Contains_Function
77		FunctionSubprogram_star
78		BodyConstruct_star
79		ProgramStmt
80		EndProgramStmt
81		FunctionSubprogram
82		FunctionPrefix
83		FunctionRange
84		FunctionParList
85		FunctionPar_Comma_FunctionPar_star_opt

5- Annexe

86		Comma_FunctionPar_star
87		FunctionPar
88		FunctionResult_opt
89		EndFunctionStmt
90		SubroutineSubprogram
91		SubroutineRange
92		SubroutineParList_opt
93		SubroutinePar_Comma_SubroutinePar_star_opt
94		Comma_SubroutinePar_star
95		SubroutinePar
96		EndSubroutineStmt
97		EndName_opt
98		BodyConstruct
99		SpecificationPartConstruct
100		DeclarationConstruct
101		TypeDeclarationStmt
102		Comma_AttrSpec_star

5- Annexe

103		AttrSpec
104		Intent_in_out
105		In_out
106		TypeDecl_Assignment
107		Comma_ObjectName_star
108		Comma_EntityDecl_star
109		EntityDecl
110		Equal_Expr_opt
111		Asterisk_CharLength_opt
112		CharLength
113		TypeParamValue
114		Expr_Or_Asterisk
115		TypeSpec
116		KindSelector_opt
117		ExecutableConstruct
118		ReturnStmt
119		ActionStmt

5- Annexe

120		AssignmentStmt
121		PrintStmt
122		Comma_OutputItemList_opt
123		FormatIdentifier
124		OutputItemList
125		Comma_OutputItem_star
126		OutputItem
127		DoConstruct
128		BlockDoConstruct
129		LoopControl_opt
130		EndDoStmt
131		Name_opt
132		LoopControl
133		Comma_IntRealDpExpression_opt
134		IntRealDpExpression
135		IfConstruct
136		ElseIfStmt_ExecutionPartConstruct_star_star

5- Annexe

137		ExecutionPartConstruct_star
138		ElseStmt_ExecutionPartConstruct_star_opt
139		IfThenStmt
140		ElseIfStmt
141		ElseStmt
142		EndIfStmt
143		ExecutionPartConstruct
144		ScalarLogicalExpr
145		Expr
146		Level5Expr
147		EquivOp_EquivOperand_star
148		EquivOperand
149		OrOp_OrOperand_star
150		OrOperand
151		AndOp_AndOperand_star
152		AndOperand
153		NotOp_opt

5- Annexe

```
154 | Level4Expr
155 | RelOp_Level3Expr_star
156 | Level3Expr
157 | Level2Expr
158 | AddOp_Sign_opt_AddOperand_star
159 | Sign_opt_AddOperand
160 | Sign_opt
161 | AddOperand
162 | MultOp_MultOperand_star
163 | MultOperand
164 | PowerOp_Level1Expr_star
165 | Level1Expr
166 | Primary
167 | FunctionReference_opt
168 |
    ↪ FunctionArg_Comma_FunctionArg_star_opt_RParenthes
169 | Comma_FunctionArg_star
```

5- Annexe

170		FunctionArg
171		Name
172		ArrayName
173		ComponentName
174		EndName
175		DummyArgName
176		FunctionName
177		ImpliedDoVariable
178		ProgramName
179		SubroutineName
180		SubroutineNameUse
181		VariableName
182		ObjectName
183		LogicalConstant
184		MultOp
185		AddOp
186		Sign

5- Annexe

```
187 | RelOp
188 | EquivOp
189 | ScalarIntLiteralConstant
190 | Scon
191 type symbol = | Terminal of terminal | NonTerminal
    ↪ of non_terminal
192 let safe_token = Ident
193 let unparsed_tokens = [Space; ]
194
195 let repr_of_terminal (t : terminal) : string =
196     match t with
197     | EOF -> "End of file"
198     | E -> "Epsilon"
199     | PowerOp -> "\\*\\*"
200     | NotOp -> "\\..not\\.."
201     | AndOp -> "\\..and\\.."
202     | OrOp -> "\\..or\\.."
```

5- Annexe

```

203 | Dcon ->
    ↪ "(([0-9]+\.\.[0-9]*)|(\.\.[0-9]+))(d(\\+|-)?[0-9]+)"
204 | Rcon ->
    ↪ "(([0-9]+\.\.[0-9]*)|(\.\.[0-9]+))(e(\\+|-)?[0-9]+)"
205 | Icon -> "[0-9]+(e(\\+|-)?[0-9]+)?"
206 | SconSingle -> "'(~[']|'')*'"
207 | SconDouble -> "\"(~[\"]|\"\\\")*\""
208 | Ident -> "[A-Za-z][A-Za-z0-9_]*"
209 | EOS -> "((!~[\\n]*)?\\n[ ]*)+"
210 | Return -> "return"
211 | Result -> "result"
212 | Contains -> "contains"
213 | True -> "\\true\\"
214 | False -> "\\false\\"
215 | Program -> "program"
216 | Function -> "function"
217 | Subroutine -> "subroutine"

```

5- Annexe

```
218 | EndProgram -> "end program"
219 | EndFunction -> "end function"
220 | EndSubroutine -> "end subroutine"
221 | EndDo -> "end do"
222 | EndIf -> "end if"
223 | Colon -> ":"
224 | Comma -> ","
225 | Equal -> "="
226 | Asterisk -> "\\*"
227 | LParenthesis -> "\\("
228 | RParenthesis -> "\\)"
229 | Integer -> "integer"
230 | Real -> "real"
231 | Double -> "double precision"
232 | Complex -> "complex"
233 | Character -> "character"
234 | Logical -> "logical"
```

5- Annexe

```
235 | Parameter -> "parameter"
236 | Intent -> "intent"
237 | In -> "in"
238 | Out -> "out"
239 | InOut -> "inout"
240 | Call -> "call"
241 | Print -> "print"
242 | Do -> "do"
243 | While -> "while"
244 | If -> "if"
245 | Else -> "else"
246 | Then -> "then"
247 | Divise -> "/"
248 | Plus -> "\\+"
249 | Minus -> "- "
250 | IsEqual -> "(==) | (\\.eq\\.)"
251 | NotEqual -> "(/=) | (\\.ne\\.)"
```


5- Annexe

```
252 | StrictLess -> "<)|(\.lt\\.)"
253 | LessEqual -> "<=)|(\.le\\.)"
254 | StrictGreater -> ">)|(\.gt\\.)"
255 | GreaterEqual -> ">=)|(\.ge\\.)"
256 | Equivalent -> "\\.eqv\\."
257 | NotEquivalent -> "\\.neqv\\."
258 | Space -> " "
259 | Recursive -> "recursive"
260
261 let string_of_terminal (t : terminal) : string =
262     match t with
263     | EOF -> "EOF"
264     | E -> "E"
265     | PowerOp -> "PowerOp"
266     | NotOp -> "NotOp"
267     | AndOp -> "AndOp"
268     | OrOp -> "OrOp"
```

5- Annexe

```
269 | Dcon -> "Dcon"
270 | Rcon -> "Rcon"
271 | Icon -> "Icon"
272 | SconSingle -> "SconSingle"
273 | SconDouble -> "SconDouble"
274 | Ident -> "Ident"
275 | EOS -> "EOS"
276 | Return -> "Return"
277 | Result -> "Result"
278 | Contains -> "Contains"
279 | True -> "True"
280 | False -> "False"
281 | Program -> "Program"
282 | Function -> "Function"
283 | Subroutine -> "Subroutine"
284 | EndProgram -> "EndProgram"
285 | EndFunction -> "EndFunction"
```

5- Annexe

```
286 | EndSubroutine -> "EndSubroutine"
287 | EndDo -> "EndDo"
288 | EndIf -> "EndIf"
289 | Colon -> "Colon"
290 | Comma -> "Comma"
291 | Equal -> "Equal"
292 | Asterisk -> "Asterisk"
293 | LParenthesis -> "LParenthesis"
294 | RParenthesis -> "RParenthesis"
295 | Integer -> "Integer"
296 | Real -> "Real"
297 | Double -> "Double"
298 | Complex -> "Complex"
299 | Character -> "Character"
300 | Logical -> "Logical"
301 | Parameter -> "Parameter"
302 | Intent -> "Intent"
```

5- Annexe

```
303 | In -> "In"
304 | Out -> "Out"
305 | InOut -> "InOut"
306 | Call -> "Call"
307 | Print -> "Print"
308 | Do -> "Do"
309 | While -> "While"
310 | If -> "If"
311 | Else -> "Else"
312 | Then -> "Then"
313 | Divise -> "Divise"
314 | Plus -> "Plus"
315 | Minus -> "Minus"
316 | IsEqual -> "IsEqual"
317 | NotEqual -> "NotEqual"
318 | StrictLess -> "StrictLess"
319 | LessEqual -> "LessEqual"
```

5- Annexe

```
320 | StrictGreater -> "StrictGreater"
321 | GreaterEqual -> "GreaterEqual"
322 | Equivalent -> "Equivalent"
323 | NotEquivalent -> "NotEquivalent"
324 | Space -> "Space"
325 | Recursive -> "Recursive"
326
327 let string_of_non_terminal (nt : non_terminal) :
    ↪ string =
328     match nt with
329     | ExecutableProgram -> "ExecutableProgram"
330     | StartCommentBlock -> "StartCommentBlock"
331     | Function_or_Subroutine_star_MainProgram ->
        ↪ "Function_or_Subroutine_star_MainProgram"
332     | Function_or_Subroutine_star ->
        ↪ "Function_or_Subroutine_star"
```

5- Annexe

```
333 | Recursive_opt_Function_or_Subroutine ->  
    ↪ "Recursive_opt_Function_or_Subroutine"  
334 | Function_or_Subroutine ->  
    ↪ "Function_or_Subroutine"  
335 | MainProgram -> "MainProgram"  
336 | MainRange -> "MainRange"  
337 | Contains_Function_opt_EndProgramStmt ->  
    ↪ "Contains_Function_opt_EndProgramStmt"  
338 | Contains_Function -> "Contains_Function"  
339 | FunctionSubprogram_star ->  
    ↪ "FunctionSubprogram_star"  
340 | BodyConstruct_star -> "BodyConstruct_star"  
341 | ProgramStmt -> "ProgramStmt"  
342 | EndProgramStmt -> "EndProgramStmt"  
343 | FunctionSubprogram -> "FunctionSubprogram"  
344 | FunctionPrefix -> "FunctionPrefix"  
345 | FunctionRange -> "FunctionRange"
```

5- Annexe

```
346 | FunctionParList -> "FunctionParList"
347 | FunctionPar_Comma_FunctionPar_star_opt ->
    ↪ "FunctionPar_Comma_FunctionPar_star_opt"
348 | Comma_FunctionPar_star ->
    ↪ "Comma_FunctionPar_star"
349 | FunctionPar -> "FunctionPar"
350 | FunctionResult_opt -> "FunctionResult_opt"
351 | EndFunctionStmt -> "EndFunctionStmt"
352 | SubroutineSubprogram -> "SubroutineSubprogram"
353 | SubroutineRange -> "SubroutineRange"
354 | SubroutineParList_opt ->
    ↪ "SubroutineParList_opt"
355 | SubroutinePar_Comma_SubroutinePar_star_opt ->
    ↪ "SubroutinePar_Comma_SubroutinePar_star_opt"
356 | Comma_SubroutinePar_star ->
    ↪ "Comma_SubroutinePar_star"
357 | SubroutinePar -> "SubroutinePar"
```

5- Annexe

```
358 | EndSubroutineStmt -> "EndSubroutineStmt"
359 | EndName_opt -> "EndName_opt"
360 | BodyConstruct -> "BodyConstruct"
361 | SpecificationPartConstruct ->
    ↪ "SpecificationPartConstruct"
362 | DeclarationConstruct -> "DeclarationConstruct"
363 | TypeDeclarationStmt -> "TypeDeclarationStmt"
364 | Comma_AttrSpec_star -> "Comma_AttrSpec_star"
365 | AttrSpec -> "AttrSpec"
366 | Intent_in_out -> "Intent_in_out"
367 | In_out -> "In_out"
368 | TypeDecl_Assignment -> "TypeDecl_Assignment"
369 | Comma_ObjectName_star ->
    ↪ "Comma_ObjectName_star"
370 | Comma_EntityDecl_star ->
    ↪ "Comma_EntityDecl_star"
371 | EntityDecl -> "EntityDecl"
```


5- Annexe

```
372 | Equal_Expr_opt -> "Equal_Expr_opt"
373 | Asterisk_CharLength_opt ->
    ↪ "Asterisk_CharLength_opt"
374 | CharLength -> "CharLength"
375 | TypeParamValue -> "TypeParamValue"
376 | Expr_Or_Asterisk -> "Expr_Or_Asterisk"
377 | TypeSpec -> "TypeSpec"
378 | KindSelector_opt -> "KindSelector_opt"
379 | ExecutableConstruct -> "ExecutableConstruct"
380 | ReturnStmt -> "ReturnStmt"
381 | ActionStmt -> "ActionStmt"
382 | AssignmentStmt -> "AssignmentStmt"
383 | PrintStmt -> "PrintStmt"
384 | Comma_OutputItemList_opt ->
    ↪ "Comma_OutputItemList_opt"
385 | FormatIdentifier -> "FormatIdentifier"
386 | OutputItemList -> "OutputItemList"
```

5- Annexe

```
387 | Comma_OutputItem_star ->  
    ↪ "Comma_OutputItem_star"  
388 | OutputItem -> "OutputItem"  
389 | DoConstruct -> "DoConstruct"  
390 | BlockDoConstruct -> "BlockDoConstruct"  
391 | LoopControl_opt -> "LoopControl_opt"  
392 | EndDoStmt -> "EndDoStmt"  
393 | Name_opt -> "Name_opt"  
394 | LoopControl -> "LoopControl"  
395 | Comma_IntRealDpExpression_opt ->  
    ↪ "Comma_IntRealDpExpression_opt"  
396 | IntRealDpExpression -> "IntRealDpExpression"  
397 | IfConstruct -> "IfConstruct"  
398 | ElseIfStmt_ExecutionPartConstruct_star_star ->  
    ↪ "ElseIfStmt_ExecutionPartConstruct_star_star"  
399 | ExecutionPartConstruct_star ->  
    ↪ "ExecutionPartConstruct_star"
```

5- Annexe

```
400 | ElseStmt_ExecutionPartConstruct_star_opt ->
    ↳ "ElseStmt_ExecutionPartConstruct_star_opt"
401 | IfThenStmt -> "IfThenStmt"
402 | ElseIfStmt -> "ElseIfStmt"
403 | ElseStmt -> "ElseStmt"
404 | EndIfStmt -> "EndIfStmt"
405 | ExecutionPartConstruct ->
    ↳ "ExecutionPartConstruct"
406 | ScalarLogicalExpr -> "ScalarLogicalExpr"
407 | Expr -> "Expr"
408 | Level5Expr -> "Level5Expr"
409 | EquivOp_EquivOperand_star ->
    ↳ "EquivOp_EquivOperand_star"
410 | EquivOperand -> "EquivOperand"
411 | OrOp_OrOperand_star -> "OrOp_OrOperand_star"
412 | OrOperand -> "OrOperand"
```

5- Annexe

```
413 | AndOp_AndOperand_star ->  
    ↳ "AndOp_AndOperand_star"  
414 | AndOperand -> "AndOperand"  
415 | NotOp_opt -> "NotOp_opt"  
416 | Level4Expr -> "Level4Expr"  
417 | RelOp_Level3Expr_star ->  
    ↳ "RelOp_Level3Expr_star"  
418 | Level3Expr -> "Level3Expr"  
419 | Level2Expr -> "Level2Expr"  
420 | AddOp_Sign_opt_AddOperand_star ->  
    ↳ "AddOp_Sign_opt_AddOperand_star"  
421 | Sign_opt_AddOperand -> "Sign_opt_AddOperand"  
422 | Sign_opt -> "Sign_opt"  
423 | AddOperand -> "AddOperand"  
424 | MultOp_MultOperand_star ->  
    ↳ "MultOp_MultOperand_star"  
425 | MultOperand -> "MultOperand"
```

5- Annexe

```
426 | PowerOp_Level1Expr_star ->  
    ↳ "PowerOp_Level1Expr_star"  
427 | Level1Expr -> "Level1Expr"  
428 | Primary -> "Primary"  
429 | FunctionReference_opt ->  
    ↳ "FunctionReference_opt"  
430 |  
    ↳ FunctionArg_Comma_FunctionArg_star_opt_RParenthes  
    ↳ ->  
    ↳ "FunctionArg_Comma_FunctionArg_star_opt_RParenthes  
431 | Comma_FunctionArg_star ->  
    ↳ "Comma_FunctionArg_star"  
432 | FunctionArg -> "FunctionArg"  
433 | Name -> "Name"  
434 | ArrayName -> "ArrayName"  
435 | ComponentName -> "ComponentName"  
436 | EndName -> "EndName"
```

5- Annexe

```
437 | DummyArgName -> "DummyArgName"
438 | FunctionName -> "FunctionName"
439 | ImpliedDoVariable -> "ImpliedDoVariable"
440 | ProgramName -> "ProgramName"
441 | SubroutineName -> "SubroutineName"
442 | SubroutineNameUse -> "SubroutineNameUse"
443 | VariableName -> "VariableName"
444 | ObjectName -> "ObjectName"
445 | LogicalConstant -> "LogicalConstant"
446 | MultOp -> "MultOp"
447 | AddOp -> "AddOp"
448 | Sign -> "Sign"
449 | RelOp -> "RelOp"
450 | EquivOp -> "EquivOp"
451 | ScalarIntLiteralConstant ->
    ↪ "ScalarIntLiteralConstant"
452 | Scon -> "Scon"
```

5- Annexe

```
1  type string_or_string_list = S of string | L of
   ↪  string_or_string_list list
2
3  (** crée une chaîne de [n] tabulations *)
4  let tabs_to_string (n : int) :
   ↪  string_or_string_list = S (String.make n '\t')
5
6  (** crée une chaîne de [n] retours à la ligne *)
7  let rec n_new_lines (n : int) :
   ↪  string_or_string_list = S (String.make n '\n')
8
9
10 let string_of_string_or_string_list (sosl :
   ↪  string_or_string_list) : string =
11   let rec aux (sosl : string_or_string_list) (acc :
   ↪  string list) : string list =
12     match sosl with
```

5- Annexe

```

13 | S s -> s :: acc
14 | L [] -> acc
15 | L (e :: q) ->
16 |     let acc2 = aux e acc in
17 |     aux (L q) acc2
18 in
19 String.concat "" (List.rev (aux sosl []))
20
21
22 let rec last_of_list (l: 'a list): 'a =
23     match l with
24     | [] -> failwith "Liste vide "
25     | e::[] -> e
26     | e::q -> last_of_list q
27
28 let print_sosl (sosl: string_or_string_list): unit
    ↪ =

```


5- Annexe

```
29 | print_string (string_of_string_or_string_list  
   | ↪  sosl);  
30 | print_newline ()
```

5- Annexe

```
1  open Symbols
2
3  type regex =
4      (* cas de base *)
5      | Epsilon
6      | Caractere of char
7      | AllChars
8      | Range of char * char
9      (* opérations sur les regex*)
10     | Concat of regex * regex
11     | Ou of regex * regex
12     | UnPlus of regex
13     | ZeroPlus of regex
14     | Vide
15     | Facultatif of regex
16     | AllBut of bool array
17
```

5- Annexe

```
18  (** affiche l'expression [c] en argument *)
19  let print_reg_list (c : regex list) : unit =
20    (** affiche en vidant au fur et à mesure la
      ↪ liste *)
21  let rec print_list_aux (r : regex list) : unit =
22    match r with
23    | [] -> ()
24    | [ Vide ] -> print_char '_'
25    | Vide :: q ->
26      print_char '_';
27      print_char ' ';
28      print_list_aux q
29    | [ Epsilon ] -> print_char '#'
30    | Epsilon :: q ->
31      print_char '#';
32      print_char ' ';
33      print_list_aux q
```

5- Annexe

```
34 | [ Caractere c ] -> print_char c
35 | Caractere c :: q ->
36 |     print_char c;
37 |     print_char ' ';
38 |     print_list_aux q
39 | [ AllChars ] -> print_char '.'
40 | AllChars :: q ->
41 |     print_char '.';
42 |     print_char ' ';
43 |     print_list_aux q
44 | [ Range (s, e) ] ->
45 |     print_char '[';
46 |     print_char s;
47 |     print_char '-';
48 |     print_char e;
49 |     print_char ']'
50 | Range (s, e) :: q ->
```

5- Annexe

```
51     print_char '[';
52     print_char s;
53     print_char '-';
54     print_char e;
55     print_char ']';
56     print_char ' ';
57     print_list_aux q
58 | [ Concat (e1, e2) ] ->
59     print_list_aux [ e1 ];
60     print_list_aux [ e2 ]
61 | Concat (e1, e2) :: q ->
62     print_list_aux [ e1 ];
63     print_list_aux [ e2 ];
64     print_char ' ';
65     print_list_aux q
66 | [ Ou (e1, e2) ] ->
67     print_char '(';
```

5- Annexe

```
68         print_list_aux [ e1 ];
69         print_char '|';
70         print_list_aux [ e2 ];
71         print_char ')'
72 | Ou (e1, e2) :: q ->
73         print_char '(';
74         print_list_aux [ e1 ];
75         print_char '|';
76         print_list_aux [ e2 ];
77         print_char ')';
78         print_char ' ';
79         print_list_aux q
80 | [ UnPlus e ] ->
81         print_char '(';
82         print_list_aux [ e ];
83         print_string ")+";
84 | UnPlus e :: q ->
```

5- Annexe

```
85     print_char '(';
86     print_list_aux [ e ];
87     print_string ")+";
88     print_char ' ';
89     print_list_aux q
90 | [ ZeroPlus e ] ->
91     print_char '(';
92     print_list_aux [ e ];
93     print_string ")*"
94 | ZeroPlus e :: q ->
95     print_char '(';
96     print_list_aux [ e ];
97     print_string ")*";
98     print_char ' ';
99     print_list_aux q
100 | [ Facultatif e ] ->
101     print_char '(';
```

5- Annexe

```
102     print_list_aux [ e ];
103     print_string ")?"
104 | Facultatif e :: q ->
105     print_char '(';
106     print_list_aux [ e ];
107     print_string ")?";
108     print_char ' ';
109     print_list_aux q
110 | [ AllBut _ ] ->
111     print_string "~(...)";
112     print_string ")"
113 | AllBut _ :: q ->
114     print_string "~(...)";
115     print_string ")";
116     print_char ' ';
117     print_list_aux q
118 in
```


5- Annexe

```
119   if List.length c = 0 then print_string "[]" else  
    ↪   print_list_aux c  
120  
121   (** convertit la chaîne de caractères s à partir de  
    ↪   l'index index et l'ajoute à  
122   la liste c *)  
123   let rec string_to_char_2 (s : string) (c : char  
    ↪   list) (index : int) : char list  
124       =  
125       if index = String.length s then List.rev c  
126       else string_to_char_2 s (s.[index] :: c) (index +  
    ↪   1)  
127  
128   exception Invalid_syntax  
129   exception Empty_pile  
130
```

5- Annexe

```
131  (** teste si la liste référencée dans [l] est vide  
    ↪  *)  
132  let is_empty (l : 'a list ref) : bool = List.length  
    ↪  !l == 0  
  
133  
134  (** dépile un élément de la pile référencée par  
    ↪  [l] *)  
135  let pop (l : 'a list ref) : 'a =  
136      match !l with  
137      | [] -> raise Empty_pile  
138      | x :: q ->  
139          l := q;  
140          x  
  
141  
142  (** convertit l'entier [n] en booléen *)  
143  let bool_of_int (n : int) : bool = if n == 0 then  
    ↪  false else true
```

5- Annexe

```

144
145  (** renvoie la disjonction de toutes les
    ↪  expressions régulières dans la liste
146      [l] *)
147  let or_reg (l : regex list) : regex =
148      let rec or_reg_aux (l : regex list) (out : regex)
149          ↪ : regex =
150          match (l, out) with
151          | [], _ -> out
152          | x :: q, Epsilon -> or_reg_aux q x
153          | x :: q, _ -> or_reg_aux q (Ou (x, out))
154      in
155      or_reg_aux l Epsilon
156
157  (** renvoie la concaténation de toutes les
    ↪  expressions régulières de [l] *)
158  let concat_reg (l : regex list) : regex =

```

5- Annexe

```
158   let rec concat_reg_aux (l : regex list) (out :  
    ↪ regex) : regex =  
159       match (l, out) with  
160       | [], _ -> out  
161       | x :: q, Epsilon -> concat_reg_aux q x  
162       | x :: q, _ -> concat_reg_aux q (Concat (x,  
    ↪ out))  
163   in  
164   concat_reg_aux l Epsilon  
165  
166   (** transforme, si c'est possible, la chaine de  
    ↪ caractères [s] en une expression  
    ↪ régulière (regex) *)  
167  
168   let rec gen_regex (s : string) : regex =  
169       let characters = ref (List.of_seq (String.to_seq  
    ↪ s)) in  
170
```

5- Annexe

```
171      (** fonction auxiliaire qui permet de générer le  
      ↪ regex entre parenthèses *)  
172      let parenthesis () : regex =  
173          let l = ref [] in  
174          try  
175              let count = ref 0 in  
176              let c = ref (pop characters) in  
177              let ignore = ref false in  
178              (* boucle sur le contenu de la parenthèse et  
              ↪ ne s'arrete pas si elle est ignorée *)  
179              while not (!c = ')') && !count = 0 && not  
              ↪ !ignore) do  
180                  if not !ignore then (  
181                      if  
182                          (* si on rencontre un \, on ignore le  
                          ↪ caractère suivant *)  
183                          !c = '\\'
```

5- Annexe

```

184         then ignore := true
185         else if !c = '(' then count := !count + 1
186         else if !c = ')' then count := !count -
            ↪ 1)
187         else ignore := false;
188         l := !c :: !l;
189         c := pop characters
190     done;
191     let s1 = String.of_seq (List.to_seq (List.rev
            ↪ !l)) in
192     gen_regex s1
193     with Empty_pile -> raise Invalid_syntax
194 in
195
196     (** fonction auxiliaire qui permet de générer le
            ↪ regex entre crochets *)
197     let crochet () : regex =

```

5- Annexe

```

198   let l = ref [] in
199   try
200     let c = ref (pop characters) in
201     let ignore_left = ref 0 in
202     let pile1 = ref [] in
203     (* on boucle tant que l'on a pas le crochet
      ↪ final et tant que l'on ignore pas *)
204     while not (!c = ']' && not (bool_of_int
      ↪ !ignore_left)) do
205       (* si on ignore pas encore, on ignore pour
        ↪ 2 tours si on rencontre un \ *)
206       if (not (bool_of_int !ignore_left)) && !c =
        ↪ '\\' then ignore_left := 2;
207       (if !ignore_left <> 2 then
208         match !pile1 with
209         | [ Caractere '-'; Caractere x ] ->
          ↪ pile1 := [ Range (x, !c) ]

```

5- Annexe

```

210         | Caractere '-' :: Caractere x :: q ->
           ↪ pile1 := Range (x, !c) :: q
211         | _ ->
           if !ignore_left = 1 && !c = 'n' then
212             pile1 := Caractere '\n' :: !pile1
213         else pile1 := Caractere !c ::
           ↪ !pile1;
214             l := !c :: !l);
215     c := pop characters;
216     if !ignore_left > 0 then ignore_left :=
           ↪ !ignore_left - 1
217 done;
218 or_reg !pile1
219 with Empty_pile -> raise Invalid_syntax
220 in
221
222

```


5- Annexe

```
223      (** fonction auxiliaire générale qui génère le  
      ↪ regex à partir de la chaîne,  
224      qui stocke au fur et à mesure dans la pile et  
      ↪ qui ignore les caractères à  
225      effets lorsque ignore est à vrai *)  
226  let rec gen_regex_2 (pile : regex list) (ignore :  
      ↪ bool) : regex =  
227      (* s'il n'y a plus rien à convertir, on  
      ↪ concatène les expressions régulières*)  
228  if is_empty characters then concat_reg pile  
229  else  
230      let c = pop characters in  
231      if ignore then  
232          if c = 'n' then gen_regex_2 (Caractere '\n'  
          ↪ :: pile) false  
233      else gen_regex_2 (Caractere c :: pile)  
          ↪ false
```

5- Annexe

```

234     else
235         match c with
236         | '\\\' -> gen_regex_2 pile true
237         | '(' -> gen_regex_2 (parenthesis () ::
238           ↪ pile) false
239         | '[' -> gen_regex_2 (crochet () :: pile)
240           ↪ false
241         | ')' | ']' -> raise Invalid_syntax
242         | '|' -> (
243             let left = gen_regex_2 [] false in
244             match (pile, left) with
245             | [], _ -> raise Invalid_syntax
246             | [ Caractere c ], AllChars | [
247               ↪ AllChars ], Caractere c ->
248                 if c = '\\n' then
249                     gen_regex_2 [ Ou (Caractere c,
250                       ↪ AllChars) ] false

```

5- Annexe

```

247         else gen_regex_2 [ AllChars ] false
248     | Caractere c :: q, AllChars | AllChars
    ↪ :: q, Caractere c ->
249         if c = '\n' then
250             gen_regex_2 (Ou (Caractere c,
    ↪ AllChars) :: q) false
251         else gen_regex_2 (AllChars :: q)
    ↪ false
252     | [ Range (d, f) ], AllChars | [
    ↪ AllChars ], Range (d, f) ->
253         if int_of_char d < 32 ||
    ↪ int_of_char f < 32 then
254             gen_regex_2 [ Ou (Range (d, f),
    ↪ AllChars) ] false
255         else gen_regex_2 [ AllChars ] false
256     | Range (d, f) :: q, AllChars |
    ↪ AllChars :: q, Range (d, f) ->

```

5- Annexe

```

257         if int_of_char d < 32 ||
           ↪ int_of_char f < 32 then
258             gen_regex_2 (Ou (Range (d, f),
           ↪ AllChars) :: q) false
259         else gen_regex_2 (AllChars :: q)
           ↪ false
260     | [ x ], _ -> gen_regex_2 [ Ou (x,
           ↪ left) ] false
261     | x :: q, _ -> gen_regex_2 (Ou (x,
           ↪ left) :: q) false)
262 | '#' -> (
263     match pile with
264     | [] -> gen_regex_2 [ Epsilon ] false
265     | Concat (_, _) :: _
266     | Caractere _ :: _
267     | Ou (_, _) :: _
268     | Range (_, _) :: _ ->

```

5- Annexe

```
269         gen_regex_2 pile false
270     | _ -> gen_regex_2 (Epsilon :: pile)
        ↪ false)
271 | '+' -> (
272     match pile with
273     | [] -> raise Invalid_syntax
274     | Epsilon :: q -> gen_regex_2 pile
        ↪ false
275     | [ x ] -> gen_regex_2 [ UnPlus x ]
        ↪ false
276     | x :: q -> gen_regex_2 (UnPlus x :: q)
        ↪ false)
277 | '*' -> (
278     match pile with
279     | [] -> raise Invalid_syntax
280     | Epsilon :: q -> gen_regex_2 pile
        ↪ false
```

5- Annexe

```
281         | [ x ] -> gen_regex_2 [ ZeroPlus x ]  
           ↪ false  
282         | x :: q -> gen_regex_2 (ZeroPlus x ::  
           ↪ q) false)  
283     | '.' -> gen_regex_2 (AllChars :: pile)  
           ↪ false  
284     | '?' -> (  
285         match pile with  
286         | [] -> raise Invalid_syntax  
287         | Epsilon :: q -> gen_regex_2 pile  
           ↪ false  
288         | [ x ] -> gen_regex_2 [ Facultatif x ]  
           ↪ false  
289         | x :: q -> gen_regex_2 (Facultatif x  
           ↪ :: q) false)  
290     | '~' -> (  
291         let c = ref (pop characters) in
```

5- Annexe

```

292     match !c with
293     | '[' ->
294         let reg = crochet () in
295         let rec recon_crochet e =
296             match e with
297             | Caractere c ->
298                 gen_regex_2
299                     (AllBut (Array.init 128 ((
300                         ↪ <> ) (int_of_char c)))
301                         :: pile)
302                     false
303             | Range (b1, b2) ->
304                 gen_regex_2
305                     (AllBut
306                         (Array.init 128 (fun i
307                             ↪ ->

```

5- Annexe

```

306             i < int_of_char b1
               ↪ || i >
               ↪ int_of_char
               ↪ b2))

307         :: pile)
308         false
309     | Ou (e1, e2) -> (
310         match (recon_crochet e1,
               ↪ recon_crochet e2) with
311     | AllBut l1, AllBut l2 ->
312         gen_regex_2
313         (AllBut (Array.map2 (
               ↪ && ) l1 l2) ::
               ↪ pile)
314         false
315         | _ -> failwith "impossible")
316     | _ -> raise Invalid_syntax

```


5- Annexe

```

317         in
318         recon_crochet reg
319     | '(' ->
320         let reg = parenthesis () in
321         let rec recon_parntethis e =
322             match e with
323             | Caractere c ->
324                 gen_regex_2
325                     (AllBut (Array.init 128 ((
326                         ↪ <> ) (int_of_char c)))
327                       :: pile)
328                       false
329             | Range (b1, b2) ->
330                 gen_regex_2
331                     (AllBut
332                         (Array.init 128 (fun i
333                             ↪ ->

```

5- Annexe

```

332                                     i < int_of_char b1
                                     ↪ || i >
                                     ↪ int_of_char
                                     ↪ b2))

333                                 :: pile)
334                             false
335 | AllChars ->
336     gen_regex_2
337     (AllBut (Array.init 128 ((
338         ↪ < ) 32)) :: pile)
339     false
340 | Ou (e1, e2) -> (
341     match (recon_parntethis e1,
342         ↪ recon_parntethis e2) with
343 | AllBut l1, AllBut l2 ->
344     gen_regex_2

```

5- Annexe

```

343         (AllBut (Array.map2 (
                ↪  && ) 11 12) ::
                ↪  pile)
344         false
345         | _ -> failwith "impossible")
346     | _ -> raise Invalid_syntax
347     in
348     recon_parntethis reg
349 | '\\' ->
350     gen_regex_2
351     (AllBut
352         (Array.init 128 (( <> )
                ↪  (int_of_char (pop
                ↪  characters))))
353         :: pile)
354     false
355 | ')' | ']' -> raise Invalid_syntax

```

5- Annexe

```

356         | c1 ->
357             gen_regex_2
358             (AllBut (Array.init 128 (( <> )
359                 ↪ (int_of_char c1))) :: pile)
360             false)
361         | _ -> gen_regex_2 (Caractere c :: pile)
362         ↪ false
363 in
364 let reg = gen_regex_2 [] false in
365 if not (is_empty characters) then raise
366     ↪ Invalid_syntax else reg

```

5- Annexe

```
1  open Symbols
2
3  type pattern = symbol list
4  type rule = symbol * pattern list
5  type rule_hashtable = (symbol, pattern list)
   ⇨ Hashtbl.t
6  type grammar = { rules_htbl : rule_hashtable;
   ⇨ start_symbol : Symbols.symbol }
7
8  (** pour un symbole [s], renvoie sa représentation
   ⇨ sous forme de chaîne *)
9  let string_of_symbol (s : symbol) : string =
10     match s with
11     | Terminal t -> string_of_terminal t
12     | NonTerminal nt -> string_of_non_terminal nt
13
14  (** affiche le symbole [s] *)
```

5- Annexe

```
15  let print_symbol (s : symbol) : unit = print_string
    ↪  (string_of_symbol s)
16
17  (** affiche chaque pattern de [patterns] *)
18  let print_patterns (patterns : pattern list) : unit
    ↪  =
19      List.iter
20          (fun pattern ->
21              List.iter (fun s -> print_string
22                  ↪  (string_of_symbol s ^ " ")) pattern;
23                  print_newline ())
24          patterns
25
26  (** affiche chaque règle de [r] *)
27  let print_rule_list (r : rule_hashtable) : unit =
28      Hashtbl.iter
          (fun s patterns ->
```

5- Annexe

```
29     print_string (string_of_symbol s ^ " -> \n");
30     print_patterns patterns;
31     print_newline (())
32     r
33
34     (** affiche la grammaire [g] *)
35     let print_grammar (g : grammar) =
36         print_string "{\nrules: ";
37         print_rule_list g.rules_htbl;
38         print_string "\nstart_symbol: ";
39         print_symbol g.start_symbol;
40         print_string "\n}\n"
41
42     (** teste si le symbole [s] est terminal *)
43     let is_terminal_symbol (s : symbol) : bool =
44         match s with Terminal _ -> true | NonTerminal _
            ↪ -> false
```

5- Annexe

```
45
46  (** teste si le symbole [s] est non terminal *)
47  let is_non_terminal_symbol (s : symbol) : bool =
    ↪  not (is_terminal_symbol s)
48
49  (** teste si la règle associe un symbole [s] est
    ↪  terminal *)
50  let is_terminal ((s, _) : rule) : bool =
    ↪  is_terminal_symbol s
51
52  (** teste si la règle [r] est non terminale*)
53  let is_non_terminal (r : rule) : bool = not
    ↪  (is_terminal r)
54
55  (** renvoie la liste des terminaux de [r] *)
56  let terminals (r : rule list) : rule list =
    ↪  List.filter is_terminal r
```


5- Annexe

```
57
58  (** renvoie la liste des non terminaux de [r] *)
59  let non_terminals (r : rule list) : rule list =
    ↪ List.filter is_non_terminal r
60
61  (** renvoie la règle associée au symbole [s] dans
    ↪ la grammaire [g] *)
62  let rule_of_symbol (g : grammar) (s : symbol) :
    ↪ rule =
63    if Hashtbl.mem g.rules_htbl s then (s,
        ↪ Hashtbl.find g.rules_htbl s)
64    else
65      failwith
66        ("the hashed grammar does not contain " ^
        ↪ string_of_symbol s
67        ^ " as a key\n")
```

5- Annexe

```

1  open Grammar_functions
2  open Symbols
3  let grammar = { start_symbol = NonTerminal
    ↪ ExecutableProgram;
4  rules_htbl = Hashtbl.of_seq (List.to_seq
    ↪ [(NonTerminal ExecutableProgram, [[NonTerminal
    ↪ StartCommentBlock; NonTerminal
    ↪ Function_or_Subroutine_star_MainProgram;]; [NonTerminal
    ↪ Function_or_Subroutine_star_MainProgram;];]);
5  (NonTerminal StartCommentBlock, [[Terminal EOS;];]);
6  (NonTerminal
    ↪ Function_or_Subroutine_star_MainProgram, [[NonTerminal
    ↪ Recursive_opt_Function_or_Subroutine; NonTerminal
    ↪ Function_or_Subroutine_star_MainProgram;]; [NonTerminal
    ↪ MainProgram; NonTerminal
    ↪ Function_or_Subroutine_star;];]);

```

5- Annexe

```

7  (NonTerminal
    ↪  Function_or_Subroutine_star, [[NonTerminal
    ↪  Recursive_opt_Function_or_Subroutine;NonTerminal
    ↪  Function_or_Subroutine_star;];[Terminal E;];]);
8  (NonTerminal
    ↪  Recursive_opt_Function_or_Subroutine, [[Terminal
    ↪  Recursive;NonTerminal
    ↪  Function_or_Subroutine;];[NonTerminal
    ↪  Function_or_Subroutine;];]);
9  (NonTerminal Function_or_Subroutine, [[NonTerminal
    ↪  FunctionSubprogram;];[NonTerminal
    ↪  SubroutineSubprogram;];]);
10 (NonTerminal MainProgram, [[NonTerminal
    ↪  ProgramStmt;NonTerminal MainRange;];]);

```

5- Annexe

```
11 (NonTerminal MainRange, [[NonTerminal
    ↳ BodyConstruct;NonTerminal
    ↳ BodyConstruct_star;NonTerminal
    ↳ Contains_Function_opt_EndProgramStmt;]; [NonTerminal
    ↳ Contains_Function_opt_EndProgramStmt;];]);
12 (NonTerminal
    ↳ Contains_Function_opt_EndProgramStmt, [[NonTerminal
    ↳ Contains_Function;NonTerminal
    ↳ EndProgramStmt;]; [NonTerminal
    ↳ EndProgramStmt;];]);
13 (NonTerminal Contains_Function, [[Terminal
    ↳ Contains;Terminal EOS;NonTerminal
    ↳ FunctionSubprogram_star;];]);
```

5- Annexe

```
14 (NonTerminal FunctionSubprogram_star, [[Terminal
    ↪ Recursive;NonTerminal
    ↪ FunctionSubprogram;NonTerminal
    ↪ FunctionSubprogram_star;]]; [NonTerminal
    ↪ FunctionSubprogram;NonTerminal
    ↪ FunctionSubprogram_star;]]; [Terminal E;]];]);
15 (NonTerminal BodyConstruct_star, [[NonTerminal
    ↪ BodyConstruct;NonTerminal
    ↪ BodyConstruct_star;]]; [Terminal E;]];]);
16 (NonTerminal ProgramStmt, [[Terminal
    ↪ Program;NonTerminal ProgramName;Terminal
    ↪ EOS;]];]);
17 (NonTerminal EndProgramStmt, [[Terminal
    ↪ EndProgram;NonTerminal EndName_opt;Terminal
    ↪ EOS;]];]);
```

5- Annexe

```
18 (NonTerminal FunctionSubprogram, [[NonTerminal
    ↪ FunctionPrefix;NonTerminal
    ↪ FunctionName;NonTerminal FunctionRange;];]);
19 (NonTerminal FunctionPrefix, [[NonTerminal
    ↪ TypeSpec;Terminal Function;];[Terminal
    ↪ Function;];]);
20 (NonTerminal FunctionRange, [[NonTerminal
    ↪ FunctionParList;NonTerminal
    ↪ FunctionResult_opt;Terminal EOS;NonTerminal
    ↪ BodyConstruct_star;NonTerminal
    ↪ EndFunctionStmt;];]);
21 (NonTerminal FunctionParList, [[Terminal
    ↪ LParenthesis;NonTerminal
    ↪ FunctionPar_Comma_FunctionPar_star_opt;Terminal
    ↪ RParenthesis;];]);
```

5- Annexe

```

22  (NonTerminal
    ↪  FunctionPar_Comma_FunctionPar_star_opt, [[NonTerminal
    ↪  FunctionPar;NonTerminal
    ↪  Comma_FunctionPar_star;]; [Terminal E;];]);
23  (NonTerminal Comma_FunctionPar_star, [[Terminal
    ↪  Comma;NonTerminal FunctionPar;NonTerminal
    ↪  Comma_FunctionPar_star;]; [Terminal E;];]);
24  (NonTerminal FunctionPar, [[NonTerminal
    ↪  DummyArgName;];]);
25  (NonTerminal FunctionResult_opt, [[Terminal
    ↪  Result;Terminal LParenthesis;NonTerminal
    ↪  VariableName;Terminal RParenthesis;]; [Terminal
    ↪  E;];]);
26  (NonTerminal EndFunctionStmt, [[Terminal
    ↪  EndFunction;NonTerminal EndName_opt;Terminal
    ↪  EOS;];]);

```

5- Annexe

```

27 (NonTerminal SubroutineSubprogram, [[Terminal
    ↳ Subroutine;NonTerminal
    ↳ SubroutineName;NonTerminal
    ↳ SubroutineRange;];]);
28 (NonTerminal SubroutineRange, [[NonTerminal
    ↳ SubroutineParList_opt;Terminal EOS;NonTerminal
    ↳ BodyConstruct_star;NonTerminal
    ↳ EndSubroutineStmt;];]);
29 (NonTerminal SubroutineParList_opt, [[Terminal
    ↳ LParenthesis;NonTerminal
    ↳ SubroutinePar_Comma_SubroutinePar_star_opt;Terminal
    ↳ RParenthesis;];[Terminal E;];]);
30 (NonTerminal
    ↳ SubroutinePar_Comma_SubroutinePar_star_opt, [[NonTerminal
    ↳ SubroutinePar;NonTerminal
    ↳ Comma_SubroutinePar_star;];[Terminal E;];]);

```


5- Annexe

```
31 (NonTerminal Comma_SubroutinePar_star, [[Terminal
    ↪ Comma;NonTerminal SubroutinePar;NonTerminal
    ↪ Comma_SubroutinePar_star;];[Terminal E;];]);
32 (NonTerminal SubroutinePar, [[NonTerminal
    ↪ DummyArgName;];]);
33 (NonTerminal EndSubroutineStmt, [[Terminal
    ↪ EndSubroutine;NonTerminal EndName_opt;Terminal
    ↪ EOS;];]);
34 (NonTerminal EndName_opt, [[NonTerminal
    ↪ EndName;];[Terminal E;];]);
35 (NonTerminal BodyConstruct, [[NonTerminal
    ↪ SpecificationPartConstruct;];[NonTerminal
    ↪ ExecutableConstruct;];]);
36 (NonTerminal
    ↪ SpecificationPartConstruct, [[NonTerminal
    ↪ DeclarationConstruct;];]);
```

5- Annexe

```
37 (NonTerminal DeclarationConstruct, [[NonTerminal
    ↳ TypeDeclarationStmt;];]);
38 (NonTerminal TypeDeclarationStmt, [[NonTerminal
    ↳ TypeSpec;NonTerminal
    ↳ Comma_AttrSpec_star;NonTerminal
    ↳ TypeDecl_Assignment;Terminal EOS;];]);
39 (NonTerminal Comma_AttrSpec_star, [[Terminal
    ↳ Comma;NonTerminal AttrSpec;NonTerminal
    ↳ Comma_AttrSpec_star;];[Terminal E;];]);
40 (NonTerminal AttrSpec, [[Terminal
    ↳ Parameter;];[NonTerminal Intent_in_out;];]);
41 (NonTerminal Intent_in_out, [[Terminal
    ↳ Intent;Terminal LParenthesis;NonTerminal
    ↳ In_out;Terminal RParenthesis;];]);
42 (NonTerminal In_out, [[Terminal In;];[Terminal
    ↳ Out;];[Terminal InOut;];]);
```

5- Annexe

```
43 (NonTerminal TypeDecl_Assignment, [[Terminal
    ↪ Colon;Terminal Colon;NonTerminal
    ↪ EntityDecl;NonTerminal
    ↪ Comma_EntityDecl_star;];[NonTerminal
    ↪ ObjectName;NonTerminal
    ↪ Comma_ObjectName_star;];]);
44 (NonTerminal Comma_ObjectName_star, [[Terminal
    ↪ Comma;NonTerminal ObjectName;NonTerminal
    ↪ Comma_ObjectName_star;];[Terminal E;];]);
45 (NonTerminal Comma_EntityDecl_star, [[Terminal
    ↪ Comma;NonTerminal EntityDecl;NonTerminal
    ↪ Comma_EntityDecl_star;];[Terminal E;];]);
46 (NonTerminal EntityDecl, [[NonTerminal
    ↪ ObjectName;NonTerminal
    ↪ Asterisk_CharLength_opt;NonTerminal
    ↪ Equal_Expr_opt;];]);
```

5- Annexe

```
47 (NonTerminal Equal_Expr_opt, [[Terminal
    ↪ Equal;NonTerminal Expr;];[Terminal E;];]);
48 (NonTerminal Asterisk_CharLength_opt, [[Terminal
    ↪ Asterisk;NonTerminal CharLength;];[Terminal
    ↪ E;];]);
49 (NonTerminal CharLength, [[Terminal
    ↪ LParenthesis;NonTerminal
    ↪ TypeParamValue;Terminal
    ↪ RParenthesis;];[NonTerminal
    ↪ ScalarIntLiteralConstant;];]);
50 (NonTerminal TypeParamValue, [[NonTerminal
    ↪ Expr_Or_Asterisk;];]);
51 (NonTerminal Expr_Or_Asterisk, [[NonTerminal
    ↪ Expr;];[Terminal Asterisk;];]);
```

5- Annexe

```

52 (NonTerminal TypeSpec, [[Terminal
    ↪ Integer;NonTerminal
    ↪ KindSelector_opt;]; [Terminal Double;]; [Terminal
    ↪ Complex;NonTerminal
    ↪ KindSelector_opt;]; [Terminal
    ↪ Logical;NonTerminal
    ↪ KindSelector_opt;]; [Terminal Real;NonTerminal
    ↪ KindSelector_opt;]; [Terminal Character;];]);
53 (NonTerminal KindSelector_opt, [[Terminal
    ↪ LParenthesis;NonTerminal Expr;Terminal
    ↪ RParenthesis;]; [Terminal E;];]);
54 (NonTerminal ExecutableConstruct, [[NonTerminal
    ↪ ActionStmt;]; [NonTerminal
    ↪ DoConstruct;]; [NonTerminal
    ↪ IfConstruct;]; [NonTerminal ReturnStmt;];]);
55 (NonTerminal ReturnStmt, [[Terminal Return;Terminal
    ↪ EOS;];]);

```

5- Annexe

```
56 (NonTerminal ActionStmt, [[NonTerminal  
    ↪ AssignmentStmt;];[NonTerminal PrintStmt;];]);  
57 (NonTerminal AssignmentStmt, [[NonTerminal  
    ↪ Name;Terminal Equal;NonTerminal Expr;Terminal  
    ↪ EOS;];]);  
58 (NonTerminal PrintStmt, [[Terminal Print;NonTerminal  
    ↪ FormatIdentifier;NonTerminal  
    ↪ Comma_OutputItemList_opt;Terminal EOS;];]);  
59 (NonTerminal Comma_OutputItemList_opt, [[Terminal  
    ↪ Comma;NonTerminal OutputItemList;];[Terminal  
    ↪ E;];]);  
60 (NonTerminal FormatIdentifier, [[Terminal  
    ↪ Asterisk;];]);  
61 (NonTerminal OutputItemList, [[NonTerminal  
    ↪ OutputItem;NonTerminal  
    ↪ Comma_OutputItem_star;];]);
```

5- Annexe

```
62 (NonTerminal Comma_OutputItem_star, [[Terminal
    ↪ Comma;NonTerminal OutputItem;NonTerminal
    ↪ Comma_OutputItem_star;];[Terminal E;];]);
63 (NonTerminal OutputItem, [[NonTerminal Expr;];]);
64 (NonTerminal DoConstruct, [[NonTerminal
    ↪ BlockDoConstruct;];]);
65 (NonTerminal BlockDoConstruct, [[Terminal
    ↪ Do;NonTerminal LoopControl_opt;NonTerminal
    ↪ ExecutionPartConstruct_star;NonTerminal
    ↪ EndDoStmt;];]);
66 (NonTerminal LoopControl_opt, [[NonTerminal
    ↪ LoopControl;Terminal EOS;];[Terminal EOS;];]);
67 (NonTerminal EndDoStmt, [[Terminal EndDo;NonTerminal
    ↪ Name_opt;Terminal EOS;];]);
68 (NonTerminal Name_opt, [[NonTerminal
    ↪ Name;];[Terminal E;];]);
```

5- Annexe

```
69 (NonTerminal LoopControl, [[Terminal While;Terminal  
    ↳ LParenthesis;NonTerminal Expr;Terminal  
    ↳ RParenthesis;]; [NonTerminal  
    ↳ VariableName;Terminal Equal;NonTerminal  
    ↳ IntRealDpExpression;Terminal Comma;NonTerminal  
    ↳ IntRealDpExpression;NonTerminal  
    ↳ Comma_IntRealDpExpression_opt;];]);  
70 (NonTerminal  
    ↳ Comma_IntRealDpExpression_opt, [[Terminal  
    ↳ Comma;NonTerminal  
    ↳ IntRealDpExpression;]; [Terminal E;];]);  
71 (NonTerminal IntRealDpExpression, [[NonTerminal  
    ↳ Expr;];]);
```


5- Annexe

72

```
(NonTerminal IfConstruct, [[NonTerminal
  ↳ IfThenStmt;NonTerminal
  ↳ ExecutionPartConstruct_star;NonTerminal
  ↳ ElseIfStmt_ExecutionPartConstruct_star_star;NonTerminal
  ↳ ElseIfStmt_ExecutionPartConstruct_star_opt;NonTerminal
  ↳ EndIfStmt;];]);
```

73

```
(NonTerminal
  ↳ ElseIfStmt_ExecutionPartConstruct_star_star, [[NonTerminal
  ↳ ElseIfStmt;NonTerminal
  ↳ ExecutionPartConstruct_star;NonTerminal
  ↳ ElseIfStmt_ExecutionPartConstruct_star_star;]; [Terminal
  ↳ E;];]);
```

74

```
(NonTerminal
  ↳ ExecutionPartConstruct_star, [[NonTerminal
  ↳ ExecutionPartConstruct;NonTerminal
  ↳ ExecutionPartConstruct_star;]; [Terminal E;];]);
```

5- Annexe

75

`(NonTerminal` `↪ ElseStmt_ExecutionPartConstruct_star_opt, [[NonTerminal` `↪ ElseStmt;NonTerminal` `↪ ExecutionPartConstruct_star;];[Terminal E;];]);`

76

`(NonTerminal IfThenStmt, [[Terminal If;Terminal` `↪ LParenthesis;NonTerminal` `↪ ScalarLogicalExpr;Terminal` `↪ RParenthesis;Terminal Then;Terminal EOS;];]);`

77

`(NonTerminal ElseIfStmt, [[Terminal Else;Terminal` `↪ If;Terminal LParenthesis;NonTerminal` `↪ ScalarLogicalExpr;Terminal` `↪ RParenthesis;Terminal Then;Terminal EOS;];]);`

78

`(NonTerminal ElseStmt, [[Terminal Else;Terminal` `↪ EOS;];]);`

79

`(NonTerminal EndIfStmt, [[Terminal EndIf;Terminal` `↪ EOS;];]);`

5- Annexe

```
80 (NonTerminal ExecutionPartConstruct, [[NonTerminal
    ↳ ExecutableConstruct;];]);
81 (NonTerminal ScalarLogicalExpr, [[NonTerminal
    ↳ Expr;];]);
82 (NonTerminal Expr, [[NonTerminal Level5Expr;];]);
83 (NonTerminal Level5Expr, [[NonTerminal
    ↳ EquivOperand;NonTerminal
    ↳ EquivOp_EquivOperand_star;];]);
84 (NonTerminal
    ↳ EquivOp_EquivOperand_star, [[NonTerminal
    ↳ EquivOp;NonTerminal EquivOperand;NonTerminal
    ↳ EquivOp_EquivOperand_star;]; [Terminal E;];]);
85 (NonTerminal EquivOperand, [[NonTerminal
    ↳ OrOperand;NonTerminal OrOp_OrOperand_star;];]);
86 (NonTerminal OrOp_OrOperand_star, [[Terminal
    ↳ OrOp;NonTerminal OrOperand;NonTerminal
    ↳ OrOp_OrOperand_star;]; [Terminal E;];]);
```

5- Annexe

```
87 (NonTerminal OrOperand, [[NonTerminal
    ↪ AndOperand;NonTerminal
    ↪ AndOp_AndOperand_star;];]);
88 (NonTerminal AndOp_AndOperand_star, [[Terminal
    ↪ AndOp;NonTerminal AndOperand;NonTerminal
    ↪ AndOp_AndOperand_star;];[Terminal E;];]);
89 (NonTerminal AndOperand, [[NonTerminal
    ↪ NotOp_opt;NonTerminal Level4Expr;];]);
90 (NonTerminal NotOp_opt, [[Terminal NotOp;];[Terminal
    ↪ E;];]);
91 (NonTerminal Level4Expr, [[NonTerminal
    ↪ Level3Expr;NonTerminal
    ↪ RelOp_Level3Expr_star;];]);
92 (NonTerminal RelOp_Level3Expr_star, [[NonTerminal
    ↪ RelOp;NonTerminal Level3Expr;NonTerminal
    ↪ RelOp_Level3Expr_star;];[Terminal E;];]);
```

5- Annexe

```
93 (NonTerminal Level3Expr, [[NonTerminal
    ↳ Level2Expr;];]);
94 (NonTerminal Level2Expr, [[NonTerminal
    ↳ Sign_opt_AddOperand;NonTerminal
    ↳ AddOp_Sign_opt_AddOperand_star;];]);
95 (NonTerminal
    ↳ AddOp_Sign_opt_AddOperand_star, [[NonTerminal
    ↳ AddOp;NonTerminal
    ↳ Sign_opt_AddOperand;NonTerminal
    ↳ AddOp_Sign_opt_AddOperand_star;]; [Terminal
    ↳ E;];]);
96 (NonTerminal Sign_opt_AddOperand, [[NonTerminal
    ↳ Sign_opt;NonTerminal AddOperand;];]);
97 (NonTerminal Sign_opt, [[NonTerminal
    ↳ Sign;]; [Terminal E;];]);
```

5- Annexe

```
108 (NonTerminal AddOperand, [[NonTerminal
    ↳ MultOperand;NonTerminal
    ↳ MultOp_MultOperand_star;];]);
109 (NonTerminal MultOp_MultOperand_star, [[NonTerminal
    ↳ MultOp;NonTerminal MultOperand;NonTerminal
    ↳ MultOp_MultOperand_star;];[Terminal E;];]);
110 (NonTerminal MultOperand, [[NonTerminal
    ↳ Level1Expr;NonTerminal
    ↳ PowerOp_Level1Expr_star;];]);
111 (NonTerminal PowerOp_Level1Expr_star, [[Terminal
    ↳ PowerOp;NonTerminal Level1Expr;NonTerminal
    ↳ PowerOp_Level1Expr_star;];[Terminal E;];]);
112 (NonTerminal Level1Expr, [[NonTerminal Primary;];]);
```

5- Annexe

```
103 (NonTerminal Primary, [[Terminal Icon;]; [Terminal
    ↪ Rcon;]; [Terminal Dcon;]; [NonTerminal
    ↪ Name; NonTerminal
    ↪ FunctionReference_opt;]; [NonTerminal
    ↪ Scon;]; [NonTerminal LogicalConstant;]; [Terminal
    ↪ LParenthesis; NonTerminal Expr; Terminal
    ↪ RParenthesis;];]);
104 (NonTerminal FunctionReference_opt, [[Terminal
    ↪ LParenthesis; NonTerminal
    ↪ FunctionArg_Comma_FunctionArg_star_opt_RParenthesis
    ↪ E;];]);
105 (NonTerminal
    ↪ FunctionArg_Comma_FunctionArg_star_opt_RParenthesis
    ↪ FunctionArg; NonTerminal
    ↪ Comma_FunctionArg_star; Terminal
    ↪ RParenthesis;]; [Terminal RParenthesis;];]);
```

5- Annexe

```
106 (NonTerminal Comma_FunctionArg_star, [[Terminal
    ↪ Comma;NonTerminal FunctionArg;NonTerminal
    ↪ Comma_FunctionArg_star;]; [Terminal E;];]);
107 (NonTerminal FunctionArg, [[NonTerminal Expr;];]);
108 (NonTerminal Name, [[Terminal Ident;];]);
109 (NonTerminal ArrayName, [[Terminal Ident;];]);
110 (NonTerminal ComponentName, [[Terminal Ident;];]);
111 (NonTerminal EndName, [[Terminal Ident;];]);
112 (NonTerminal DummyArgName, [[Terminal Ident;];]);
113 (NonTerminal FunctionName, [[Terminal Ident;];]);
114 (NonTerminal ImpliedDoVariable, [[Terminal
    ↪ Ident;];]);
115 (NonTerminal ProgramName, [[Terminal Ident;];]);
116 (NonTerminal SubroutineName, [[Terminal Ident;];]);
117 (NonTerminal SubroutineNameUse, [[Terminal
    ↪ Ident;];]);
118 (NonTerminal VariableName, [[Terminal Ident;];]);
```


5- Annexe

```
119 (NonTerminal ObjectName, [[Terminal Ident;];]);
120 (NonTerminal LogicalConstant, [[Terminal
    ↪ True;]; [Terminal False;];]);
121 (NonTerminal MultOp, [[Terminal Asterisk;]; [Terminal
    ↪ Divise;];]);
122 (NonTerminal AddOp, [[NonTerminal Sign;];]);
123 (NonTerminal Sign, [[Terminal Plus;]; [Terminal
    ↪ Minus;];]);
124 (NonTerminal RelOp, [[Terminal IsEqual;]; [Terminal
    ↪ NotEqual;]; [Terminal StrictLess;]; [Terminal
    ↪ LessEqual;]; [Terminal StrictGreater;]; [Terminal
    ↪ GreaterEqual;];]);
125 (NonTerminal EquivOp, [[Terminal
    ↪ Equivalent;]; [Terminal NotEquivalent;];]);
126 (NonTerminal ScalarIntLiteralConstant, [[Terminal
    ↪ Icon;];]);
```

5- Annexe

```
127  (NonTerminal Scon, [[Terminal SconSingle;]; [Terminal  
    ↪   SconDouble;];]);  
128  ]))}
```

5- Annexe

```
1  open Regex
2  open Symbols
3  open Vector
4  module IntSet = Set.Make (Int)
5
6  (* différentes représentations d'automates utiles
   ↪ lors de sa transformation en automate
   ↪ déterministe *)
7  type automate = {
8      nodes : int list;
9      debut_l : int list;
10     fin : (int * terminal) list;
11     transitions : (char option * int) list array;
12 }
13
14 type automate_sans_eps = {
15     nodes : int list;
```

5- Annexe

```
16   debut_l : int list;  
17   fin : (int * terminal) list;  
18   transitions_sans_eps : (char * int) list array;  
19 }  
  
20  
21 type pre_automate_det = {  
22     mutable nodes : int list;  
23     debut : int;  
24     mutable fin : terminal option array;  
25     mutable pre_transitions : int array Vector.t;  
26 }  
  
27  
28 (* automate de sortie utile pour la transpilation  
    ↪ *)  
29 type automate_det = {  
30     nodes : int list;  
31     debut : int;
```

5- Annexe

```
32   fin : terminal option array;  
33   transitions : int array array;  
34       (* transitions.(i).(j), i le sommet de  
        ↪ départ, j l'entier du caractère *)  
35 }  
36  
37 (** Renvoie la liste de toutes les lignes dans le  
    ↪ fichier [file_name]. *)  
38 let read_file (file_name : string) : string list =  
39     let rec lire file liste =  
40         let line = input_line file in  
41         ();  
42         try lire file (line :: liste)  
43         with End_of_file ->  
44             close_in file;  
45             line :: liste  
46     in
```

5- Annexe

```
47   List.rev (lire (open_in file_name) [])
48
49   (** Renvoie la liste [[0...n-1]] *)
50   let range_list (n : int) : int list =
51     let l = ref [] in
52     for i = 0 to n - 1 do
53       l := i :: !l
54     done;
55     !l
56
57   (** crée l'automate de stage 1 à partir d'une
58   ↪ expression régulière [reg] et d'un
59   terminal [t] *)
60   let automate_gen (reg : regex) (t : terminal) :
61     ↪ automate =
62     let dico = Hashtbl.create 0 in
63     let a =
```

5- Annexe

```
62     ref { nodes = []; debut_1 = [ 0 ]; fin = [ (1,  
        ↪ t) ]; transitions = [[]] }  
63 in  
64  
65     (** ajoute la transition [n1 -> n2] étiquetée par  
        ↪ [c] à l'automate *)  
66 let add_transition ((n1, c, n2) : int * char  
        ↪ option * int) =  
67     if Hashtbl.mem dico n1 then  
68         Hashtbl.replace dico n1 ((c, n2) ::  
            ↪ Hashtbl.find dico n1)  
69     else Hashtbl.add dico n1 [ (c, n2) ]  
70 in  
71  
72 let next_node = ref 2 in  
73
```

5- Annexe

```
74  let rec automate_gen_aux (reg : regex)
    ↪  ((node_before, node_after) : int * int)
75      : unit =
76      match reg with
77      | Vide -> ()
78      | Epsilon -> add_transition (node_before, None,
    ↪  node_after)
79      | AllChars ->
80          for i = 32 to 127 do
81              add_transition (node_before, Some
    ↪  (char_of_int i), node_after)
82          done
83      | Caractere x -> add_transition (node_before,
    ↪  Some x, node_after)
84      | Concat (g, d) ->
85          let node = !next_node in
86          next_node := !next_node + 1;
```


5- Annexe

```
87      automate_gen_aux g (node_before, node);
88      automate_gen_aux d (node, node_after)
89  | Ou (g, d) ->
90      automate_gen_aux g (node_before,
91        ↪ node_after);
92      automate_gen_aux d (node_before,
93        ↪ node_after)
94  | Range (a1, a2) ->
95      for i = int_of_char a1 to int_of_char a2 do
96          automate_gen_aux (Caractere (char_of_int
97            ↪ i)) (node_before, node_after)
98      done
99  | ZeroPlus e ->
100      automate_gen_aux e (node_before,
101        ↪ node_before);
102      automate_gen_aux Epsilon (node_before,
103        ↪ node_after)
```

5- Annexe

```
99      | UnPlus e ->
100         automate_gen_aux e (node_before,
101           ↪ node_before);
102         automate_gen_aux e (node_before,
103           ↪ node_after)
104      | Facultatatif e ->
105         automate_gen_aux e (node_before,
106           ↪ node_after);
107         automate_gen_aux Epsilon (node_before,
108           ↪ node_after)
109      | AllBut e ->
110         for i = 0 to 127 do
111             if e.(i) then
112                 add_transition (node_before, Some
113                   ↪ (char_of_int i), node_after)
114             done
115 in
```

5- Annexe

```
111 automate_gen_aux reg (0, 1);
112
113 let l1 = List.of_seq (Hashtbl.to_seq_keys dico)
    ↪ in
114 let rec build_trans (l : int list) (arr : (char
    ↪ option * int) list array) :
115     unit =
116     match l with
117     | [] -> ()
118     | x :: q ->
119         arr.(x) <- Hashtbl.find dico x;
120         build_trans q arr
121 in
122 let arr = Array.make !next_node [] in
123 build_trans l1 arr;
124 {
125     nodes = range_list !next_node;
```

5- Annexe

```

126     debut_l = !a.debut_l;
127     fin = !a.fin;
128     transitions = arr;
129 }

130
131 (** construit la disjonction des automates de
132 ↪ [l_a], qui reconnaît donc l'union
133 des langage des automates de [l_a] *)
134 let ou_automates (l_a : automate list) : automate =
135     (** construit la disjonction de la liste [l] avec
136     ↪ un automate [out] *)
137     let rec ou_automate_aux (l : automate list) (out
138     ↪ : automate) =
139         match l with
140         | [] -> out
141         | x :: q ->
142             let inc = List.length out.nodes in

```

5- Annexe

```
140     let x2 =
141         {
142             nodes = List.map (( + ) inc) x.nodes;
143             debut_l = List.map (( + ) inc)
144                 ↪ x.debut_l;
145             fin = List.map (fun (x, y) -> (x + inc,
146                 ↪ y)) x.fin;
147             transitions = [||];
148         }
149     in
150     ou_automate_aux q
151     {
152         nodes = out.nodes @ x2.nodes;
153         debut_l = out.debut_l @ x2.debut_l;
154         fin = out.fin @ x2.fin;
155         transitions =
156             Array.init
```

5- Annexe

```
155             (inc + Array.length x.transitions)
156             (fun i ->
157                 if i < inc then
158                     ↪ out.transitions.(i)
159                 else
160                     List.map
161                         (fun (c, x) -> (c, x + inc))
162                         x.transitions.(i - inc));
163
164 in
165 let a =
166     ou_automate_aux l_a
167     { nodes = []; debut_l = []; fin = [];
168       ↪ transitions = [[]] }
169
170 in
171 let nouv_d = List.length a.nodes in
```

5- Annexe

```
170      (** Ajoute le début unique à l'automate par une  
    ↪ transition [k -> x] étiquetée  
171 par epsilon (où k est l'état de départ, x est  
    ↪ un ancien départ d'un  
172 élément de [l]) à out *)  
173 let rec ajouter_debut (l : int list) (out : (char  
    ↪ option * int) list array) :  
174   (char option * int) list array =  
175   Array.append out [| List.map (fun x -> (None,  
    ↪ x)) l |]  
176 in  
177 {  
178   nodes = nouv_d :: a.nodes;  
179   debut_l = [ nouv_d ];  
180   fin = a.fin;  
181   transitions = ajouter_debut a.debut_l  
    ↪ a.transitions;
```

5- Annexe

```
182   }
183
184   (** enlève les doublons dans la liste [l] *)
185   let remove_duplicates (l : 'a list) : 'a list =
186     let tbl = Hashtbl.create 0 in
187     let rec aux (l : 'a list) (out : 'a list) : 'a
188       ↪ list =
189       match l with
190       | [] -> out
191       | x :: q ->
192         if Hashtbl.mem tbl x then aux q out
193         else (
194           Hashtbl.add tbl x 0;
195           aux q (x :: out))
196     in
197     aux l []
```


5- Annexe

```
198  (*
199      Étapes pour enlever les epsilon-transitions:
200      - On prend un sommet
201      - On regarde toutes les epsilon-transitions
202        ↪ sortantes (pas les boucles)
203      - On regarde tous les transitions entrantes
204      - On ajoute des nouvelles transitoins des
205        ↪ entrantes vers les sortantes
206      - On supprime l'epsilon-transition
207
208      Requis:
209      - transitions entrantes de chaque sommet =>
210        ↪ précalculé
211
212  *)
213
214  (** enlève les epsilon-transitions dans l'automate
215      ↪ [a] en effectuant les étapes
```

5- Annexe

```
211      ci-dessus *)
212  let enleve_epsilon_trans (a : automate) :
    ↪ automate_sans_eps =
213      let len = List.length a.nodes in
214      (* on stocke les degrés entrants et sortants de
    ↪ chaque sommet *)
215      (* les éléments stockés ne sont pas linéarisés
    ↪ *)
216      let entrants = Array.make len [] in
217
218      (* degré entrant de chaque sommet *)
219      let degres = Array.make len 0 in
220
221      let trans_temp = ref a.transitions in
222      let fin_temp = ref a.fin in
223      let deg_traite = ref 0 in
224
```

5- Annexe

```
225      (** trouve le premier sommet de degré zéro *)
226      let rec find_premier_deg_zero () : int =
227          let deg = !deg_traite in
228          if degres.(!deg_traite) <= 0 then incr
229              ↪ deg_traite;
230          while !deg_traite < len && degres.(!deg_traite)
231              ↪ > 0 do
232              incr deg_traite
233          done;
234          if deg == !deg_traite then (
235              incr deg_traite;
236              find_premier_deg_zero ())
237          else if !deg_traite >= len then -1
238          else !deg_traite
239      in
240
241      (* construit les listes d'entrants de chaque
242      ↪ sommet *)
```

5- Annexe

```
240   Array.iteri
241     (fun e l ->
242       List.iter
243         (fun (c, s) ->
244           entrants.(s) <- (e, c) :: entrants.(s);
245           degrees.(s) <- degrees.(s) + 1)
246       l)
247   a.transitions;
248   (* ajouter un de degré entrant pour chaque entrée
249    ↪ *)
249   List.iter (fun x -> degrees.(x) <- degrees.(x) + 1)
250             ↪ a.debut_l;
251
252   (* applique l'algorithme de suppression des
253    ↪ epsilon transitions sur chacun des sommets *)
252   List.iter
253     (fun node_i ->
```

5- Annexe

```
254      (* récupération les transitions entrantes
      ↪ avec epsilon transitions qui ne sont pas
      ↪ des boucles + enlever du degré entrant
      ↪ pour chaque epsilon transition *)
255  let res =
256      remove_duplicates
257      (List.fold_left
258       (fun acc x ->
259         match x with
260         | _, Some x -> acc
261         | node, None ->
262           degrees.(node_i) <-
263             ↪ degrees.(node_i) - 1;
264             if node == node_i then acc else
265             ↪ node :: acc)
266       [] entrants.(node_i))
267  in
```

5- Annexe

```
266      (* on enlève les epsilon entrants de chaque
      ↪  epsilon transition *)
267  entrants.(node_i) <-
268      List.filter (fun (_, c) -> c != None)
      ↪  entrants.(node_i);
269  (* on enlève les sortants de chaque epsilon
      ↪  transition *)
270  !trans_temp.(node_i) <-
271      List.filter
272          (fun (c, x) -> not (c == None && x ==
      ↪  node_i))
273          !trans_temp.(node_i);
274  List.iter
275      (fun x ->
276          !trans_temp.(x) <-
277              List.filter
```

5- Annexe

```
278      (fun (c, x1) -> not (c == None && x1
      ↪ == node_i))
279      !trans_temp.(x))
280      res;
281      (* ajoute les transitions des entrants avec
      ↪ epsilon transitions vers les sortants et
      ↪ actualise les entrants/sortants/degré de
      ↪ deux sommets *)
282      List.iter
283      (fun x ->
284        List.iter
285        (fun (c, node) ->
286          !trans_temp.(x) <- (c, node) ::
287          ↪ !trans_temp.(x);
288          entrants.(node) <- (x, c) ::
289          ↪ entrants.(node);
290          degres.(node) <- degres.(node) + 1)
```

5- Annexe

```
289         (List.filter
290          (fun (c, n) -> not (n == node_i && c
291                               ↪ = None))
292          !trans_temp.(node_i)))
293
294     res;
295
296     (* récupération des états de fin du sommet de
297     ↪ départ *)
298     let fins =
299       List.fold_left
300         (fun acc (x, t) -> if x == node_i then t
301                               ↪ :: acc else acc)
302         [] a.fin
303   in
```


5- Annexe

```
301      (* applique l'ensemble des états de fin aux  
      ↪ sommets qui avaient une epsilon  
      ↪ transition *)  
302      List.iter  
303      (fun x -> List.iter (fun t -> fin_temp :=  
      ↪ (x, t) :: !fin_temp) fins)  
304      res)  
305      a.nodes;  
306  
307      (* enlever les nodes qui ne sont plus atteintes  
      ↪ *)  
308      let todo = ref (find_premier_deg_zero ()) in  
309      while !todo <> -1 do  
310          List.iter (fun (_, x) -> degres.(x) <-  
          ↪ degres.(x) - 1) !trans_temp.(!todo);  
311          todo := find_premier_deg_zero ()  
312      done;
```

5- Annexe

```
313
314  (* enlever les option car toutes les transition
    ↪  (devraient être) sans epsilon transition *)
315  {
316    nodes = List.filter (fun x -> degres.(x) > 0)
    ↪  a.nodes;
317    debut_l = a.debut_l;
318    fin = List.filter (fun (x, _) -> degres.(x) >
    ↪  0) !fin_temp;
319    transitions_sans_eps =
320      Array.map
321        (fun x ->
322          List.map
323            (fun (c, n) ->
324              match c with None -> failwith "pas
    ↪  correct" | Some c1 -> (c1, n))
```

5- Annexe

```

325         (List.filter (fun (_, n) -> degrees.(n)
326             ↪ > 0) x))
327     !trans_temp;
328 }
329 (** Donne à chaque ensemble d'éléments [elem] un
330 ↪ entier à partir de la table de
331 linéarisation [lin_tbl] et cette de
332 ↪ délinéarisation [delin_tbl] *)
333 let lin (elem : IntSet.t) (lin_tbl : (IntSet.t,
334     ↪ int) Hashtbl.t)
335     (delin_tbl : IntSet.t Vector.t) : int =
336     if not (Hashtbl.mem lin_tbl IntSet.empty) then
337         Hashtbl.add lin_tbl IntSet.empty 0;
338
339     if Hashtbl.find lin_tbl IntSet.empty <>
340         ↪ Vector.length delin_tbl then

```

5- Annexe

```
337     failwith "La taille des deux tables n'est pas  
    ↪     la même";  
  
338  
339     if not (Hashtbl.mem lin_tbl elem) then (  
340         Hashtbl.add lin_tbl elem (Hashtbl.find lin_tbl  
    ↪     IntSet.empty);  
341         Hashtbl.replace lin_tbl IntSet.empty  
    ↪     (Hashtbl.find lin_tbl IntSet.empty + 1);  
342         Vector.push delin_tbl elem);  
343     Hashtbl.find lin_tbl elem  
344  
345     (** Pour chaque entier [elem], renvoie l'ensemble  
    ↪     de sommets correspondant, à  
346     l'aide de la table de délinéarisation  
    ↪     [delin_tbl] *)  
347     let delin (elem : int) (delin_tbl : IntSet.t  
    ↪     Vector.t) : IntSet.t =
```

5- Annexe

```
348   if elem = -1 then IntSet.empty
349   else if elem >= Vector.length delin_tbl then
350       failwith "L'élément demandé n'est pas dans la
           ↪ table"
351   else Vector.get delin_tbl elem
352
353   (** détermine l'automate [a] *)
354   let determinise (a : automate_sans_eps) :
           ↪ automate_det =
355       (* crée les deux table utiles pour la
           ↪ linéarisation / délinéarisation *)
356       let lin_tbl = Hashtbl.create (List.length
           ↪ a.nodes) in
357       let delin_tbl = Vector.create ~dummy: IntSet.empty
           ↪ in
358
```

5- Annexe

```
359   let start_node = lin (IntSet.of_list a.debut_l)
      ↪ lin_tbl delin_tbl in
360   let todo = ref [ start_node ] in
361
362   let a_det =
363     {
364       nodes = [ start_node ];
365       debut = start_node;
366       fin = [[]];
367       pre_transitions = Vector.create
          ↪ ~dummy:(Array.make 1 0);
368     }
369   in
370
371   Vector.push a_det.pre_transitions (Array.make 128
      ↪ (-1));
372   (* pour le début *)
```

5- Annexe

```
373   let fin = IntSet.of_list (List.map (fun (a, b) ->
    ↪   a) a.fin) in
374
375   (** trouve les successeurs de tous les sommets de
    ↪   [l] et on les stocke dans
376     [arr] *)
377   let rec trouver_suivants (l : IntSet.t) (arr :
    ↪   int array) : unit =
378     let len = Array.length arr in
379
380     (* on rassemble les éléments accessibles depuis
    ↪   tous les sommets de l *)
381     let storage = Array.make len IntSet.empty in
382     IntSet.iter
383       (fun x ->
384         List.iter
385           (fun (c, e) ->
```

5- Annexe

```
386         storage.(int_of_char c) <- IntSet.add e
           ↪ storage.(int_of_char c))
387     a.transitions_sans_eps.(x))
388     l;
389
390     (* on linéarise les sommets obtenus et on les
       ↪ stocke dans arr *)
391     for i = 0 to len - 1 do
392         if not (IntSet.is_empty storage.(i)) then
393             arr.(i) <- lin storage.(i) lin_tbl
               ↪ delin_tbl
394     done
395 in
396
397     (** teste si le sommet [elem] linéarisé contient
       ↪ des éléments finaux et
398         l'ajoute aux finaux si c'est le cas *)
```


5- Annexe

```
399   let ajouter_fin (elem : int) : unit =
400     let res =
401       remove_duplicates
402         (List.map
403           (fun e -> List.assoc e a.fin)
404           (IntSet.to_list (IntSet.inter fin (delin
405             ↪ elem delin_tbl))))))
406   in
407   match res with
408   | [] -> ()
409   | [ e ] -> a_det.fin.(elem) <- Some e
410   | [ e1; e2 ] ->
411     (* L'un des deux est un safe_token, elle
412       ↪ peut être vue autrement donc elle est
413       ↪ ignorée *)
414     if e1 = safe_token then a_det.fin.(elem) <-
415       ↪ Some e2
```

5- Annexe

```

412     else if e2 = safe_token then
        ↪ a_det.fin.(elem) <- Some e1
413     else (
414         print_char "";
415         print_string (repr_of_terminal e1);
416         print_string "\" \"";
417         print_string (repr_of_terminal e2);
418         print_char "";
419         print_string "on un état final commun";
420         print_newline ();
421         failwith "" (* il a plus d'un élément
        ↪ final *)
422     | _ -> failwith "A syntax can't have more than
        ↪ one output"
423 in
424
425 let finished = ref false in

```

5- Annexe

```
426   let seen = ref IntSet.empty in
427   (* on construit l'automate déterministe *)
428   while not !finished do
429     match !todo with
430     | [] -> finished := true
431     | x :: q ->
432       let init_len = Hashtbl.find lin_tbl
433         ↪ IntSet.empty in
434       todo := q;
435       let suivants = Array.make 128 (-1) in
436       trouver_suivants (delin x delin_tbl)
437         ↪ suivants;
438
439       let arr = Vector.get a_det.pre_transitions
440         ↪ x in
441       for i = 0 to 127 do
442         if suivants.(i) >= init_len then
```

5- Annexe

```
440      (* si c'est un nouveau noeud, on
      ↪ l'ajoute a la liste de traitement,
441      on l'ajoute dans l'automate et on
      ↪ vérifie s'il est final *)
442      if not (IntSet.mem suivants.(i) !seen)
      ↪ then (
443          seen := IntSet.add suivants.(i)
      ↪ !seen;
444          todo := suivants.(i) :: !todo;
445          Vector.push a_det.pre_transitions
      ↪ (Array.make 128 (-1));
446          a_det.nodes <- suivants.(i) ::
      ↪ a_det.nodes);
447      (* noeud déjà existant/complétion nouveau
      ↪ noeud, comme on ne traite qu'une fois
      ↪ chaque sommets,
```

5- Annexe

```
448         on sait que les sommets trouvé sont les
           ↪ bons, on les remplace *)
449         if suivants.(i) <> -1 then arr.(i) <-
           ↪ suivants.(i)
450         done;
451         Vector.set a_det.pre_transitions x arr
452     done;
453
454     (* gérer le cas des mots vides, qui sont donc à
       ↪ la fois initiaux et finaux *)
455     a_det.fin <- Array.make (List.length a_det.nodes)
       ↪ None;
456     List.iter ajouter_fin a_det.nodes;
457     {
458         nodes = a_det.nodes;
459         debut = a_det.debut;
460         fin = a_det.fin;
```

5- Annexe

```
461     transitions = Vector.to_array
      ↪     a_det.pre_transitions;
462 }
463
464 (** effectue le delta 1 sur l'automate [a] à partir
      ↪ de [node] avec l'étiquette
465 [c] *)
466 let exec_char (a : automate_det) (node : int) (c :
      ↪ char) : int =
467     a.transitions.(node).(int_of_char c)
468
469 (** effectue le delta étoile sur l'automate [a]
      ↪ avec le texte [texte] *)
470 let execution_mot (a : automate_det) (texte : char
      ↪ list) :
471     int * char list * char list =
472     let node = ref a.debut in
```

5- Annexe

```
473   let last_found = ref (-1) in
474   let text_as_last = ref texte in
475   let texte = ref texte in
476   let text_read = ref [] in
477   let last_read = ref [] in
478
479   (* tant que l'on est pas dans le puit *)
480   while !node != -1 do
481     match !texte with
482     | [] -> node := -1 (* forcer la fin de la
483       ↪ boucle *)
484     | c :: q ->
485       (* on effectue le delta 1 *)
486       text_read := c :: !text_read;
487       node := exec_char a !node c;
488       texte := q;
489       (* on note si on passe par un état final *)
```

5- Annexe

```
489      if !node = -1 then (if !last_found = -1
    ↪      then last_read := !text_read)
490  else (
491      (match a.fin.(!node) with
492      | None -> ()
493      | Some t ->
494          last_found := !node;
495          text_as_last := !texte);
496      last_read := !text_read)
497  done;
498  (* on renvoie le dernier état final trouvé*)
499  (!last_found, !text_as_last, !last_read)
500
501  (** exécute l'automate [a] en boucle sur le texte
    ↪  [txt] pour créer une liste de
502      lexèmes *)
503  let exec (a : automate_det) (txt : string) :
    ↪  (symbol * string) list =
```


5- Annexe

```
504      (** exécute l'automate [a] en boucle sur le texte  
      ↪ [texte] et concatène le  
505      résultat dans [out] *)  
506  let rec exec_aux (a : automate_det) (texte : char  
      ↪ list)  
507      (out : (terminal * string) list) : (terminal  
      ↪ * string) list =  
508  match texte with  
509  | [] -> List.rev out  
510  | _ -> (  
511      match execution_mot a texte with  
512      | -1, _, s ->  
513          print_string "Le lexème ";  
514          print_string (String.of_seq  
              ↪ (List.to_seq (List.rev s)));  
515          print_string "' n'est pas un lexème  
              ↪ reconnu\n";
```

5- Annexe

```

516         failwith ""
517     | x, q, s -> (
518         let s = String.of_seq (List.to_seq
519             ↪ (List.rev s)) in
520         match a.fin.(x) with
521         | None ->
522             print_string "Le lexème ";
523             print_string s;
524             print_string "' n'est pas un lexème
525             ↪ reconnu\n";
526             failwith ""
527         | Some t -> exec_aux a q ((t, s) ::
528             ↪ out)))
529
530 in
531 let res = exec_aux a (List.of_seq (String.to_seq
532     ↪ txt)) [] in

```

5- Annexe

```

528   let tbl = Hashtbl.create (List.length
    ↪   unparsed_tokens) in
529   List.iter (fun x -> Hashtbl.add tbl x ())
    ↪   unparsed_tokens;
530   List.map
531     (fun (t, s) -> (Terminal t, s))
532     (List.filter (fun (x, _) -> not (Hashtbl.mem
    ↪   tbl x)) res)
533
534   (** exécute l'automate [a] sur le fichier [f_name]
    ↪   *)
535   let exec_of_file (a : automate_det) (f_name :
    ↪   string) : (symbol * string) list =
536     exec a
537     (List.fold_left (fun acc line -> acc ^ line ^
    ↪   "\n") "" (read_file f_name))

```

5- Annexe

```
1  open Grammar_functions
2  open Symbols
3
4  module SymbolSet = Set.Make (struct
5      type t = symbol
6
7      let compare = compare
8  end)
9
10 (* arbre de syntaxe (non abstraite) *)
11 type at = Noeud of (symbol * string) * at list
12 type symbol_SS_Htbl = (symbol, SymbolSet.t)
13     ↪ Hashtbl.t
14
15 let print_SymbolSet (ss : SymbolSet.t) : unit =
16     print_endline "SymbolSet(";
17     SymbolSet.iter
```

5- Annexe

```

17      (fun s ->
18          print_symbol s;
19          print_newline ())
20      ss;
21      print_endline ")\\n"
22
23  let print_SymbolSet_Hastable (h : symbol_SS_Htbl) :
24  ↪ unit =
25      Hashtbl.iter
26      (fun s ss ->
27          print_string (string_of_symbol s ^ " -> ");
28          print_SymbolSet ss;
29          print_newline ())
30      h
31
32  (* returns a hashtable containing the first set of
33  ↪ every non terminal *)

```

5- Annexe

```
32 let first (g : grammar) : symbol_SS_Htbl =  
33   (* prog dyn, filling first_h *)  
34   let first_h = Hashtbl.create (Hashtbl.length  
    ↪ g.rules_htbl) in  
  
35  
36   let rec first_of_rule ((s, patterns) : rule) :  
    ↪ unit =  
37     if is_terminal (s, patterns) then  
38       Hashtbl.add first_h s (SymbolSet.singleton s)  
39     else if  
40       (* checking if already computed *)  
41       not (Hashtbl.mem first_h s)  
42     then  
43       Hashtbl.add first_h s  
44       (List.fold_left  
45         (fun acc d ->  
46           let f = first_of_pattern s d in
```

5- Annexe

```
47         if SymbolSet.disjoint f acc then
48             ↪ SymbolSet.union f acc
49         else (
50             print_symbol s;
51             print_newline ();
52             print_SymbolSet f;
53             prerr_newline ();
54             print_SymbolSet acc;
55             print_newline ();
56             failwith "the first set is not
57                 ↪ disjoint"))
58         SymbolSet.empty patterns)
59     else ()
60 and first_of_pattern (s : symbol) (d : pattern) :
61     ↪ SymbolSet.t =
62     match d with
63     | [] -> failwith "empty pattern"
```

5- Annexe

```
61 | Terminal t :: q -> SymbolSet.singleton
   | ↪ (Terminal t)
62 | fst_symbol :: q ->
63 |   first_of_rule (fst_symbol, Hashtbl.find
   |   ↪ g.rules_htbl fst_symbol);
64 |   if
65 |     SymbolSet.mem (Terminal E) (Hashtbl.find
   |     ↪ first_h fst_symbol)
66 |     && q <> []
67 |   then
68 |     SymbolSet.union (first_of_pattern s q)
69 |     (SymbolSet.remove (Terminal E)
   |     ↪ (Hashtbl.find first_h fst_symbol))
70 |   else Hashtbl.find first_h fst_symbol
71 | in
72 | Hashtbl.iter (fun name pattern -> first_of_rule
   | ↪ (name, pattern)) g.rules_htbl;
```


5- Annexe

```

73     first_h
74
75     let rec first_of_pattern (fst_ss_htbl :
    ↪ symbol_SS_Htbl) (p : pattern) :
76         SymbolSet.t =
77         (* print_endline ("first_of_pattern ");
    ↪ print_patterns [p]; print_newline(); *)
78     match p with
79     | [] -> failwith "empty pattern in
    ↪ first_of_pattern"
80     | Terminal t :: _ -> SymbolSet.singleton
    ↪ (Terminal t)
81     | NonTerminal nt :: q ->
82         if
83             SymbolSet.mem (Terminal E) (Hashtbl.find
    ↪ fst_ss_htbl (NonTerminal nt))
84             && q <> []

```

5- Annexe

```
85         then
86             SymbolSet.union
87                 (first_of_pattern fst_ss_htbl q)
88                 (SymbolSet.remove (Terminal E)
89                     (Hashtbl.find fst_ss_htbl (NonTerminal
90                         ↪ nt)))
91         else Hashtbl.find fst_ss_htbl (NonTerminal
92             ↪ nt)
93
94     (* returns a hashtable containing the follow set of
95     ↪ every non terminal (all the terminals that can
96     ↪ occur after a rule) *)
97 let follow (g : grammar) : symbol_SS_Htbl =
98     (* contains the terminals following the key in
99     ↪ the hashtable *)
100     let follow_non_terminal =
101         Hashtbl.of_seq
```

5- Annexe

```
97         (Seq.map
98           (fun (name, pattern) -> (name,
99             ↪ SymbolSet.empty))
100           (Hashtbl.to_seq g.rules_htbl))
101
102   in
103
104   (* contains the rule names in which the key
105     ↪ appears in at the end *)
106   let parent_on_end =
107     Hashtbl.of_seq
108       (Seq.map
109         (fun (name, pattern) -> (name,
110           ↪ SymbolSet.empty))
111         (Hashtbl.to_seq g.rules_htbl))
112
113   in
```

5- Annexe

```

110      (* fills the follow_non_terminal and
      ↪ parent_on_end *)
111  let rec pattern_follow (s : symbol) (d : pattern)
      ↪ : unit =
112      match d with
113      | NonTerminal s1 :: s2 :: q ->
114          Hashtbl.replace follow_non_terminal
      ↪ (NonTerminal s1)
115          (SymbolSet.add s2 (Hashtbl.find
      ↪ follow_non_terminal (NonTerminal
      ↪ s1)));
116          pattern_follow s (s2 :: q)
117      | Terminal s1 :: q -> pattern_follow s q
118      | NonTerminal s1 :: [] ->
119          if NonTerminal s1 <> s && NonTerminal s1 <>
      ↪ Terminal E then

```

5- Annexe

```

120      Hashtbl.replace parent_on_end
      ↪ (NonTerminal s1)
121      (SymbolSet.add s (Hashtbl.find
      ↪ parent_on_end (NonTerminal s1)))
122      else ()
123      | [] -> ()
124  in
125
126  let patterns_follow (s : symbol) (patterns :
  ↪ pattern list) : unit =
127      List.iter (pattern_follow s) patterns
128  in
129  Hashtbl.iter patterns_follow g.rules_htbl;
130
131  let first_h = first g in
132  (* contains the terminals that can appear after a
  ↪ key in the grammar*)

```

5- Annexe

```
133   let follow_h = Hashtbl.create (Hashtbl.length
    ↪   first_h) in
134
135   let rec union_and_next_on_epsilon (next_hashtable
    ↪   : symbol_SS_Htbl)
136     (s : symbol) (acc : SymbolSet.t) :
    ↪   SymbolSet.t =
137     if is_non_terminal_symbol s then
138       if SymbolSet.mem (Terminal E) (Hashtbl.find
    ↪   next_hashtable s) then (
139         follow_of_symbol s;
140         SymbolSet.remove (Terminal E)
141           (SymbolSet.union acc
142             (SymbolSet.union (Hashtbl.find
    ↪   follow_h s)
143               (Hashtbl.find next_hashtable s))))
```

5- Annexe

```
144     else SymbolSet.union acc (Hashtbl.find
      ↪ next_hashtable s)
145   else SymbolSet.singleton s
146 and follow_of_symbol (s : symbol) : unit =
147   (* checking if already computed *)
148   if not (Hashtbl.mem follow_h s) then
149     let follow_non_terminal_set =
150       SymbolSet.fold
151         (union_and_next_on_epsilon first_h)
152         (Hashtbl.find follow_non_terminal s)
153         (SymbolSet.singleton (Terminal EOF))
154   in
155   let follow_parent_on_end_set =
156     SymbolSet.fold
157       (fun s ->
158         follow_of_symbol s;
159         union_and_next_on_epsilon follow_h s)
```

5- Annexe

```

160         (Hashtbl.find parent_on_end s)
161         (SymbolSet.singleton (Terminal EOF))
162     in
163
164     Hashtbl.replace follow_h s
165     (SymbolSet.union follow_non_terminal_set
166       ↪ follow_parent_on_end_set)
167   else ()
168   in
169   Hashtbl.iter (fun s _ -> follow_of_symbol s)
170     ↪ g.rules_htbl;
171   follow_h
172
173 let analyse_LL1_of_symbol (g : grammar) (text :
174   ↪ (symbol * string) list)
175   (s : symbol) : at * (symbol * string) list =
176   let follow_sshtbl = follow g in

```


5- Annexe

```
174   let first_sshdbl = first g in
175
176   let rec analyse_LL1_of_pattern (text : (symbol *
    ↪ string) list) (s : symbol)
177     (p : pattern) : at * (symbol * string) list =
178     (* print_endline "analyse_LL1_of_pattern"; *)
179     if p = [ Terminal E ] then
180       (Noeud ((s, ""), [ Noeud ((Terminal E, ""),
    ↪ [] ) ]), text)
181   else
182     let txt = ref text in
183     let t =
184       Noeud
185         ( (s, ""),
186           List.map
187             (fun (s_p : symbol) ->
```

5- Annexe

```

188         let tree, txt' =
           ↪ analyse_LL1_of_symbol_aux !txt
           ↪ s_p in
189         txt := txt';
190         tree)
191     p )
192     in
193     (t, !txt)
194 and analyse_LL1_of_symbol_aux (text : (symbol *
   ↪ string) list) (s : symbol) :
195     at * (symbol * string) list =
196     (* print_endline ("analyse_LL1_of_symbol_aux "
   ↪ ^ string_of_symbol s); *)
197     (*print_patterns [ List.map fst text ];
198     print_newline ();*)
199     if is_terminal (s, []) then
200         (* print_endline "is terminal"; *)

```

5- Annexe

```

201     match text with
202     | [] -> failwith "no text to match in
        ↳ analyse_LL1, text is empty"
203     | t1 :: q ->
204         if fst t1 = s then (Noeud (t1, []), q)
205         else failwith "the expected terminal does
        ↳ not match the text"
206     else
207         (* print_endline "is non_terminal"; *)
208         match text with
209         | [] ->
210             if
211                 (* print_endline "Terminal E"; *)
212                 SymbolSet.mem (Terminal E)
213                 ↳ (Hashtbl.find first_sshtbl s)
                && SymbolSet.mem (Terminal EOF)
                ↳ (Hashtbl.find follow_sshtbl s)

```

5- Annexe

```
214         then (  
215             let l =  
216                 List.filter  
217                     (fun p ->  
218                         SymbolSet.mem (Terminal E)  
219                             ↪ (first_of_pattern  
220                             ↪ first_sshtbl p))  
219                 (Hashtbl.find g.rules_htbl s)  
220         in  
221         if l = [] then (  
222             print_string  
223                 ("on " ^ string_of_symbol s ^ ",  
224                 ↪ expected "  
225                 ^ string_of_symbol (Terminal E)  
226                 ^ " in first of patterns but first  
227                 ↪ of patters is\n");
```

5- Annexe

```

226         print_SymbolSet (Hashtbl.find
           ↪ first_sshtbl s);
227         failwith "error 1")
228     else ();
229     analyse_LL1_of_pattern text s (List.nth
           ↪ 1 0))
230     else failwith "text is epsilon but more
           ↪ symbols are expected"
231 | _ ->
232     if
233         (*print_SymbolSet (Hashtbl.find
           ↪ first_sshtbl s);
234         print_SymbolSet (Hashtbl.find
           ↪ follow_sshtbl s);*)
235         SymbolSet.mem (Terminal E)
           ↪ (Hashtbl.find first_sshtbl s)
236         && SymbolSet.mem

```

5- Annexe

```
237         (fst (List.nth text 0))
238         (Hashtbl.find follow_sshtbl s)
239     then (
240         let l =
241             List.filter
242             (fun p ->
243                 SymbolSet.mem (Terminal E)
244                     ↪ (first_of_pattern
245                     ↪ first_sshtbl p))
246             (Hashtbl.find g.rules_htbl s)
247         in
248         if l = [] then (
249             print_string
250             ("on " ^ string_of_symbol s ^ ",
251             ↪ expected "
252             ^ string_of_symbol (Terminal E)
```

5- Annexe

```

250      ~ " in first of patterns but first
      ~ of patters is\n");
251      print_SymbolSet (Hashtbl.find
      ~ first_sshtbl s);
252      failwith "error 2")
253      else ();
254      analyse_LL1_of_pattern text s (List.nth
      ~ l 0))
255      else (
256        (* print_endline "cas 2";
257        print_symbol (fst (List.nth text 0));
258        print_newline (); *)
259        let l =
260          List.filter
261            (fun p ->
262              SymbolSet.mem
263                (fst (List.nth text 0))

```

5- Annexe

```

264         (first_of_pattern first_sshtbl
           ↪ p))
265     (Hashtbl.find g.rules_htbl s)
266 in
267 if l = [] then (
268     print_string
269     ("on " ^ string_of_symbol s ^ ",
       ↪ expected "
       ^ string_of_symbol (fst (List.nth
       ↪ text 0))
       ^ " in first of patterns but first
       ↪ of patters is\n");
271     print_SymbolSet (Hashtbl.find
       ↪ first_sshtbl s);
272     failwith "error 3")
273 else ();
274

```


5- Annexe

```
275         analyse_LL1_of_pattern text s (List.nth
           ↪ 1 0))
276     in
277     analyse_LL1_of_symbol_aux text s
278
279 let analyse_LL1 (g : grammar) (text : (symbol *
           ↪ string) list) : at =
280     fst (analyse_LL1_of_symbol g (text @ [ (Terminal
           ↪ EOF, "") ]) g.start_symbol)
```

5- Annexe

```

1  open Abstract_tokens
2  open LL1
3  open Symbols
4
5  (** écrase les noeuds à écraser à partir de [t] et
6  ↳ l'ajoute à ceux de la liste
7  [l] (pour remonter des arguments par exemple)
8  ↳ *)
9
10 let flatten (t : ast) (l : ast list) : ast list =
11   let rec aux (t : ast) (out : ast list) : ast list
12     ↳ =
13     match t with
14     | Noeud (ToFlatten, []) -> out
15     | Noeud (ToFlatten, x :: q) ->
16       (* vide l'entièreté des éléments de x dans
17       ↳ out puis ceux de q *)
18       let out = aux x out in

```

5- Annexe

```

14         aux (Noeud (ToFlatten, q)) out
15     | _ -> t :: out
16     in
17     List.rev_append (aux t []) l
18
19     (** convertis l'arbre de syntaxe abstrait [t] pour
20     ↪ que les fonctions, dont le
21     nom est stocké au fur et à mesure dans
22     ↪ [curr_func], aient leur valeur de
23     retour *)
24 let rec link_return_function (t : ast) (return_val
25     ↪ : string option) : ast =
26     match t with
27     | Noeud (Syntax Function, l) ->
28         let name, l =
29             List.fold_left
30                 (fun (_name, _l) (x : ast) ->

```

5- Annexe

```

28         match x with
29         | Noeud (Name s, []) when _name = "" ->
           ↪ (s, x :: _l)
30         | Noeud (Syntax Out, [ Noeud (Name s,
           ↪   []) ]) -> (s, _l)
31         | _ -> (_name, x :: _l))
32     ("", []) l
33 in
34 let l =
35     match l with
36     | Noeud (Syntax Return, []) :: q ->
           ↪ List.rev l
37     | _ ->
38         let (l : ast list) =
39             Noeud (Syntax Return, [ Noeud (Name
           ↪   name, []) ]) :: l
40         in

```

5- Annexe

```

41         List.rev l
42     in
43     Noeud
44         ( Syntax Function,
45           List.map (fun x -> link_return_function x
46                    ↪ (Some name)) l )
47 | Noeud (Syntax Subroutine, l) ->
48     Noeud
49         ( Syntax Function,
50           match List.rev l with
51           | Noeud (Syntax Return, []) :: q ->
52             ↪ List.rev q
53           | _ -> l )
54 | Noeud (Syntax Return, []) -> (
55     match return_val with
56     | None -> failwith "Return doit être dans une
57       ↪ fonction"

```

5- Annexe

```

55 | Some s -> Noeud (Syntax Return, [ Noeud
    |↪ (Name s, []) ]))
56 | Noeud (x, l) ->
57 | Noeud (x, List.map (fun x ->
    |↪ link_return_function x return_val) l)
58
59 (** Si [t] est un commentaire, le convertit pour
    ↪ séparer les différentes lignes,
60 sinon renvoie une erreur *)
61 let convert_comments (t : ast) : ast list =
62   let nb = ref 0 in
63   match t with
64   | Noeud (Commentaire s, []) ->
65       let l = List.map String.trim
        |↪ (String.split_on_char '\n' s) in
66       let l2 =
67         List.map

```

5- Annexe

```

68         (fun (s : string) : ast ->
69             if s = "" then Noeud (NewLine, [])
70             else (
71                 incr nb;
72                 Noeud (Commentaire (String.sub s 1
73                     ↪ (String.length s - 1)), [])))
74     in
75     if !nb = 0 then match l2 with [] -> l2 | _ ::
76         ↪ q -> q else l2
77 | _ -> failwith "l'argument donné n'est pas un
78     ↪ commentaire"
79
80 (** Enlève le niveau d'abstraction de l'arbre de
81     ↪ syntaxe [t] créé avec LL1 *)
82 let convert_to_abstract (t : at) : ast =
83     let rec convert_to_abstract_aux (t : at) : ast =

```

5- Annexe

```

81      match t with
82      | Noeud ((NonTerminal ExecutableProgram, _), l)
      ↪ -> (
83          match l with
84          | [
85              Noeud ((NonTerminal StartCommentBlock,
      ↪ s1), l1);
86              Noeud ((NonTerminal
      ↪ Function_or_Subroutine_star_MainProgram,
      ↪ s), l);
87          ] ->
88              Noeud
89                  ( ProgramRoot,
90                    convert_to_abstract_aux (Noeud
      ↪ ((Terminal EOS, s1), []))
91                    :: flatten
92                      (convert_to_abstract_aux

```


5- Annexe

```

93         (Noeud
94             ( ( NonTerminal
95
96                 ↪ Function_or_Subrou
97                 s ),
98                 1 )))
99         [] )
100     | [
101         Noeud ((NonTerminal
102             ↪ Function_or_Subroutine_star_MainProgram,
103             ↪ s), 1);
104     ] ->
105         Noeud
106             ( ProgramRoot,
107                 flatten
108                     (convert_to_abstract_aux
109                         (Noeud

```

5- Annexe

```

107         ( ( NonTerminal
              ↪ Function_or_Subroutine_star
108             s ),
109             1 )))
110     [] )
111     | _ -> failwith "ExecutableProgram")
112 | Noeud ((NonTerminal
  ↪ Function_or_Subroutine_star_MainProgram,
  ↪ _), 1) -> (
113     match 1 with
114     | [
115         Noeud ((NonTerminal
              ↪ Recursive_opt_Function_or_Subroutine,
              ↪ s), 1);
116         Noeud ((NonTerminal
              ↪ Function_or_Subroutine_star_MainProgram,
              ↪ s1), 11);

```

5- Annexe

```

117         ] ->
118         Noeud
119         ( ToFlatten,
120         [
121             convert_to_abstract_aux
122             (Noeud
123                 ((NonTerminal
124                     ↪ Recursive_opt_Function_or_Su
125                     ↪ s), 1)));
126             convert_to_abstract_aux
127             (Noeud
128                 ( ( NonTerminal
129                     ↪ Function_or_Subroutine_star_l
130                     s1 ),
131                     11 ));
132         ] )
133     | [

```

5- Annexe

```

131         Noeud ((NonTerminal MainProgram, s), 1);
132     Noeud
133         ( (NonTerminal
            ↪ Function_or_Subroutine_star, _),
            [ Noeud ((Terminal E, _), []) ] );
134 ] ->
135     convert_to_abstract_aux (Noeud
            ↪ ((NonTerminal MainProgram, s), 1))
136 | [
137     Noeud ((NonTerminal MainProgram, s), 1);
138     Noeud ((NonTerminal
            ↪ Function_or_Subroutine_star, s1), 11);
139 ] ->
140     Noeud
141         ( ToFlatten,
142         [
143             convert_to_abstract_aux
144 
```

5- Annexe

```

145         (Noeud ((NonTerminal
           ↪ MainProgram, s), 1));
146     convert_to_abstract_aux
147     (Noeud ((NonTerminal
           ↪ Function_or_Subroutine_star,
           ↪ s1), 11));
148     ] )
149     | _ -> failwith
       ↪ "Function_or_Subroutine_star_MainProgram")
150 | Noeud ((NonTerminal
       ↪ Function_or_Subroutine_star, _), 1) -> (
151     match 1 with
152     | [
153         Noeud ((NonTerminal
           ↪ Recursive_opt_Function_or_Subroutine,
           ↪ s), 1);
154         Noeud

```

5- Annexe

```

155      ( (NonTerminal
      ↪   Function_or_Subroutine_star, _),
156      [ Noeud ((Terminal E, _), []) ] );
157  ] ->
158      convert_to_abstract_aux
159      (Noeud ((NonTerminal
      ↪   Recursive_opt_Function_or_Subroutine,
      ↪   s), 1))
160  | [
161      Noeud ((NonTerminal
      ↪   Recursive_opt_Function_or_Subroutine,
      ↪   s), 1);
162      Noeud ((NonTerminal
      ↪   Function_or_Subroutine_star, s1), 11);
163  ] ->
164      Noeud
165      ( ToFlatten,

```

5- Annexe

```

166         [
167             convert_to_abstract_aux
168             (Noeud
169                 ((NonTerminal
170                     ↪ Recursive_opt_Function_or_Sul
171                     ↪ s), 1));
172             convert_to_abstract_aux
173             (Noeud ((NonTerminal
174                 ↪ Function_or_Subroutine_star,
175                 ↪ s1), 1));
176         ] )
177     | _ -> failwith
178         ↪ "Function_or_Subroutine_star")
179 | Noeud
180     ( (NonTerminal
181         ↪ Recursive_opt_Function_or_Subroutine,
182         ↪ _),

```

5- Annexe

```

176         [
177             Noeud ((Terminal Recursive, _), []);
178             Noeud ((NonTerminal
179                 ↪ Function_or_Subroutine, s), l);
180         ] )
181 | Noeud
182     ( (NonTerminal
183         ↪ Recursive_opt_Function_or_Subroutine,
184         ↪ _),
185     [ Noeud ((NonTerminal
186         ↪ Function_or_Subroutine, s), l) ] ) ->
187     convert_to_abstract_aux
188     (Noeud ((NonTerminal
189         ↪ Function_or_Subroutine, s), l))
190 | Noeud
191     ( (NonTerminal Function_or_Subroutine, _),

```


5- Annexe

```

187      [ Noeud ((NonTerminal FunctionSubprogram,
      ↪ s), 1) ] ) ->
188      convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal FunctionSubprogram, s),
      ↪ 1))
189  | Noeud
190      ( (NonTerminal Function_or_Subroutine, _),
191        [ Noeud ((NonTerminal
      ↪ SubroutineSubprogram, s), 1) ] ) ->
192      convert_to_abstract_aux
193      (Noeud ((NonTerminal
      ↪ SubroutineSubprogram, s), 1))
194  | Noeud
195      ( (NonTerminal FunctionSubprogram, _),
196        [
197          Noeud ((NonTerminal FunctionPrefix, s),
      ↪ 1);

```

5- Annexe

```

198         Noeud ((NonTerminal FunctionName, s1),
              ↪ 11);
199         Noeud ((NonTerminal FunctionRange, s2),
              ↪ 12);
200     ] ) -> (
201 match 1 with
202 | [ Noeud ((Terminal Function, _), []) ] ->
203     Noeud
204         ( Syntax Function,
205           flatten
206             (Noeud
207               ( ToFlatten,
208                 [
209                   convert_to_abstract_aux
210                     (Noeud ((NonTerminal
                          ↪ FunctionName, s1),
                          ↪ 11));

```

5- Annexe

```

211         convert_to_abstract_aux
212         (Noeud ((NonTerminal
                ↪ FunctionRange, s2),
                ↪ 12));
213     ] ))
214     [ ] )
215     | _ ->
216         Noeud
217         ( Syntax Function,
218         flatten
219         (Noeud
220         ( ToFlatten,
221         [
222         convert_to_abstract_aux
223         (Noeud ((NonTerminal
                ↪ FunctionPrefix, s),
                ↪ 1));

```

5- Annexe

```

224         convert_to_abstract_aux
225         (Noeud ((NonTerminal
                ↪ FunctionName, s1),
                ↪ l1));
226     convert_to_abstract_aux
227     (Noeud ((NonTerminal
                ↪ FunctionRange, s2),
                ↪ l2));
228         ] ))
229     [ ] ))
230 | Noeud
231     ( (NonTerminal FunctionPrefix, _),
232       [
233         Noeud ((NonTerminal TypeSpec, s), l);
234         Noeud ((Terminal Function, _), [ ] );
235       ] ) ->

```

5- Annexe

```
236      convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal TypeSpec, s), 1))
237 | Noeud
238   ( (NonTerminal FunctionRange, _),
239     [
240       Noeud ((NonTerminal FunctionParList,
241               ↪ s), 1);
242       Noeud ((NonTerminal FunctionResult_opt,
243               ↪ s4), 14);
244       Noeud ((Terminal EOS, s1), 11);
245       Noeud ((NonTerminal BodyConstruct_star,
246               ↪ s3), 13);
247       Noeud ((NonTerminal EndFunctionStmt,
248               ↪ s2), 12);
249     ] ) ->
250 let n = ref [] in
251 let n3 = ref [] in
```

5- Annexe

```

248     let n4 = ref [] in
249     (match l3 with
250     | [ Noeud ((Terminal E, _), []) ] -> ()
251     | _ ->
252         n3 :=
253             [
254                 convert_to_abstract_aux
255                     (Noeud ((NonTerminal
256                         ↪ BodyConstruct_star, s3),
257                         ↪ l3)));
256             ]);
257     (match l with
258     | [
259         Noeud ((Terminal LParenthesis, _), []);
260         Noeud

```

5- Annexe

```

261      ( (NonTerminal
      ↪  FunctionPar_Comma_FunctionPar_star_opt,
      ↪  _),
262      [ Noeud ((Terminal E, _), []) ] );
263  Noeud ((Terminal RParenthesis, _), []);
264  ] ->
265      ()
266  | _ ->
267      n :=
268      [
269          convert_to_abstract_aux
270          (Noeud ((NonTerminal
      ↪  FunctionParList, s), 1));
271      ]);
272  (match 14 with
273  | [ Noeud ((Terminal E, _), []) ] -> ()
274  | _ ->

```

5- Annexe

```
275         n4 :=
276         [
277             convert_to_abstract_aux
278             (Noeud ((NonTerminal
279                 ↪ FunctionResult_opt, s4),
280                 ↪ l4));
281         ]);
282     Noeud
283     ( ToFlatten,
284       flatten
285       (Noeud (ToFlatten, !n))
286       (flatten
287         (Noeud (ToFlatten, !n4))
288         (convert_to_abstract_aux (Noeud
289             ↪ ((Terminal EOS, s1), l1))
290         :: flatten
291           (Noeud (ToFlatten, !n3))
```


5- Annexe

```

289         [
290             convert_to_abstract_aux
291             (Noeud ((NonTerminal
                     ↪ EndFunctionStmt, s2),
                     ↪ 12));
292         ])) )
293 | Noeud
294     ( (NonTerminal FunctionResult_opt, _),
295       [
296         Noeud ((Terminal Result, _), []);
297         Noeud ((Terminal LParenthesis, _), []);
298         Noeud ((NonTerminal VariableName, s),
                ↪ 1);
299         Noeud ((Terminal RParenthesis, _), []);
300       ] ) ->
301     Noeud
302         ( Syntax Out,

```

5- Annexe

```

303         [
304             convert_to_abstract_aux (Noeud
305                 ↪ ((NonTerminal VariableName, s),
306                 ↪ l));
307         ] )
308 | Noeud
309     ( (NonTerminal FunctionParList, _),
310         [
311             Noeud ((Terminal LParenthesis, _), []);
312             Noeud ((NonTerminal
313                 ↪ FunctionPar_Comma_FunctionPar_star_opt,
314                 ↪ s), l);
315             Noeud ((Terminal RParenthesis, _), []);
316         ] ) ->
317     convert_to_abstract_aux

```

5- Annexe

```

314         (Noeud ((NonTerminal
           ↪ FunctionPar_Comma_FunctionPar_star_opt,
           ↪ s), 1))
315 | Noeud
316   ( (NonTerminal
     ↪ FunctionPar_Comma_FunctionPar_star_opt,
     ↪ _),
317   [
318     Noeud ((NonTerminal FunctionPar, s),
           ↪ 1);
319     Noeud
320       ( (NonTerminal
         ↪ Comma_FunctionPar_star, _),
         [ Noeud ((Terminal E, _), []) ] );
321   ] ) ->
322
323 convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal FunctionPar, s), 1))

```

5- Annexe

```

324 | Noeud
325   ( (NonTerminal
    ↪ FunctionPar_Comma_FunctionPar_star_opt,
    ↪ _),
326   [
327     Noeud ((NonTerminal FunctionPar, s),
    ↪ 1);
328     Noeud ((NonTerminal
    ↪ Comma_FunctionPar_star, s1), l1);
329   ] ) ->
330 Noeud
331   ( ToFlatten,
332   [
333     convert_to_abstract_aux (Noeud
    ↪ ((NonTerminal FunctionPar, s),
    ↪ 1));
334     convert_to_abstract_aux

```

5- Annexe

```

335         (Noeud ((NonTerminal
               ↪ Comma_FunctionPar_star, s1),
               ↪ l1));
336     ] )
337 | Noeud
338     ( (NonTerminal Comma_FunctionPar_star, _),
339     [
340         Noeud ((Terminal Comma, _), []);
341         Noeud ((NonTerminal FunctionPar, s),
               ↪ l);
342         Noeud
343             ( (NonTerminal
               ↪ Comma_FunctionPar_star, _),
               [ Noeud ((Terminal E, _), []) ] ) );
344     ] ) ->
345 convert_to_abstract_aux (Noeud
346     ↪ ((NonTerminal FunctionPar, s), l))

```

5- Annexe

```

347 | Noeud
348   ( (NonTerminal Comma_FunctionPar_star, _),
349     [
350       Noeud ((Terminal Comma, _), []);
351       Noeud ((NonTerminal FunctionPar, s),
352             ↪ 1);
353       Noeud ((NonTerminal
354             ↪ Comma_FunctionPar_star, s1), l1);
355     ] ) ->
356   Noeud
357     ( ToFlatten,
358       [
359         convert_to_abstract_aux (Noeud
360           ↪ ((NonTerminal FunctionPar, s),
361           ↪ 1));
362         convert_to_abstract_aux

```

5- Annexe

```

359             (Noeud ((NonTerminal
                    ↪ Comma_FunctionPar_star, s1),
                    ↪ l1));
360         ] )
361     | Noeud
362       ( (NonTerminal FunctionPar, _),
363         [ Noeud ((NonTerminal DummyArgName, s),
                 ↪ l) ] ) ->
364     convert_to_abstract_aux (Noeud
                               ↪ ((NonTerminal DummyArgName, s), l))
365 | Noeud
366   ( (NonTerminal EndFunctionStmt, _),
367     [
368       Noeud ((Terminal EndFunction, _), []);
369       Noeud ((NonTerminal EndName_opt, _),
               ↪ _);
370       Noeud ((Terminal EOS, s), l);

```

5- Annexe

```

371         ] ) ->
372         convert_to_abstract_aux (Noeud ((Terminal
          ↪ EOS, s), 1))
373     | Noeud
374     ( (NonTerminal SubroutineSubprogram, _),
375       [
376         Noeud ((Terminal Subroutine, _), []);
377         Noeud ((NonTerminal SubroutineName, s),
          ↪ 1);
378         Noeud ((NonTerminal SubroutineRange,
          ↪ s1), 11);
379       ] ) ->
380     Noeud
381     ( Syntax Subroutine,
382       convert_to_abstract_aux (Noeud
          ↪ ((NonTerminal SubroutineName, s),
          ↪ 1))

```


5- Annexe

```

383         :: flatten
384         (convert_to_abstract_aux
385          (Noeud ((NonTerminal
386                  ↪ SubroutineRange, s1), l1)))
387         [] )
388 | Noeud
389   ( (NonTerminal SubroutineRange, _),
390     [
391       Noeud ((NonTerminal
392               ↪ SubroutineParList_opt, s), l);
393       Noeud ((Terminal EOS, s1), l1);
394       Noeud ((NonTerminal BodyConstruct_star,
395               ↪ s2), l2);
396       Noeud ((NonTerminal EndSubroutineStmt,
397               ↪ s3), l3);
398     ] ) ->
399 let n = ref [] in

```

5- Annexe

```

396     let n2 = ref [] in
397     (match l with
398     | [ Noeud ((Terminal E, _), []) ] -> ()
399     | _ ->
400         n :=
401         [
402             convert_to_abstract_aux
403             (Noeud ((NonTerminal
404                 ↪ SubroutineParList_opt, s),
405                 ↪ l));
406             (match l2 with
407             | [ Noeud ((Terminal E, _), []) ]
408             | [
409                 Noeud ((Terminal LParenthesis, _), []);
410                 Noeud

```

5- Annexe

```

410      ( (NonTerminal
      ↪ SubroutinePar_Comma_SubroutinePar_star
      ↪ _),
411      [ Noeud ((Terminal E, _), []) ] );
412      Noeud ((Terminal RParenthesis, _), []);
413  ] ->
414      ()
415  | _ ->
416      n2 :=
417      [
418          convert_to_abstract_aux
419          (Noeud ((NonTerminal
      ↪ BodyConstruct_star, s2),
      ↪ 12)));
420      ]);
421      Noeud
422      ( ToFlatten,

```

5- Annexe

```

423         flatten
424         (Noeud (ToFlatten, !n))
425         (convert_to_abstract_aux (Noeud
426         ↪ ((Terminal EOS, s1), l1))
427         :: flatten
428         (Noeud (ToFlatten, !n2))
429         [
430         convert_to_abstract_aux
431         (Noeud ((NonTerminal
432         ↪ EndSubroutineStmt, s3),
433         ↪ l3));
434         ]) )
435 | Noeud
436   ( (NonTerminal SubroutineParList_opt, _),
437     [
438       Noeud ((Terminal LParenthesis, _), []);
439       Noeud

```

5- Annexe

```

437         ((NonTerminal
           ↪ SubroutinePar_Comma_SubroutinePar_star
           ↪ s), 1);
438     Noeud ((Terminal RParenthesis, _), []);
439 ] ) ->
440 convert_to_abstract_aux
441     (Noeud ((NonTerminal
           ↪ SubroutinePar_Comma_SubroutinePar_star_opt
           ↪ s), 1))
442 | Noeud
443     ( (NonTerminal
       ↪ SubroutinePar_Comma_SubroutinePar_star_opt,
       ↪ _),
444     [
445         Noeud ((NonTerminal SubroutinePar, s),
           ↪ 1);
446         Noeud

```

5- Annexe

```

447         ( (NonTerminal
           ↪ Comma_SubroutinePar_star, _),
448         [ Noeud ((Terminal E, _), []) ] );
449     ] ) ->
450     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal SubroutinePar, s), l))
451 | Noeud
452     ( (NonTerminal
           ↪ SubroutinePar_Comma_SubroutinePar_star_opt,
           ↪ _),
453     [
454         Noeud ((NonTerminal SubroutinePar, s),
           ↪ l);
455         Noeud ((NonTerminal
           ↪ Comma_SubroutinePar_star, s1), l1);
456     ] ) ->
457     Noeud

```

5- Annexe

```
458         ( ToFlatten,  
459         [  
460             convert_to_abstract_aux  
461             (Noeud ((NonTerminal SubroutinePar,  
                     ↪ s), 1));  
462             convert_to_abstract_aux  
463             (Noeud ((NonTerminal  
                     ↪ Comma_SubroutinePar_star, s1),  
                     ↪ l1));  
464         ] )  
465 | Noeud  
466     ( (NonTerminal Comma_SubroutinePar_star,  
        ↪ _),  
467     [  
468         Noeud ((Terminal Comma, _), []);  
469         Noeud ((NonTerminal SubroutinePar, s),  
                ↪ 1);
```

5- Annexe

```

470         Noeud
471         ( (NonTerminal
           ↪ Comma_SubroutinePar_star, _),
472           [ Noeud ((Terminal E, _), []) ] );
473     ] ) ->
474     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal SubroutinePar, s), l))
475 | Noeud
476     ( (NonTerminal Comma_SubroutinePar_star,
           ↪ _),
477       [
478         Noeud ((Terminal Comma, _), []);
479         Noeud ((NonTerminal SubroutinePar, s),
           ↪ l);
480         Noeud ((NonTerminal
           ↪ Comma_SubroutinePar_star, s1), l1);
481       ] ) ->

```


5- Annexe

```

482         Noeud
483         ( ToFlatten,
484         [
485             convert_to_abstract_aux
486             (Noeud ((NonTerminal SubroutinePar,
487                     ↪ s), 1));
488             convert_to_abstract_aux
489             (Noeud ((NonTerminal
490                     ↪ Comma_SubroutinePar_star, s1),
491                     ↪ 11));
492         ] )
493 | Noeud
494     ( (NonTerminal SubroutinePar, _),
495       [ Noeud ((NonTerminal DummyArgName, s),
496               ↪ 1) ] ) ->
497     convert_to_abstract_aux (Noeud
498                             ↪ ((NonTerminal DummyArgName, s), 1))

```

5- Annexe

```
494 | Noeud
495   ( (NonTerminal EndSubroutineStmt, _),
496     [
497       Noeud ((Terminal EndSubroutine, _),
498             ↪ []);
499       Noeud ((NonTerminal EndName_opt, _),
500             ↪ _);
501       Noeud ((Terminal EOS, s), 1);
502     ] ) ->
503   convert_to_abstract_aux (Noeud ((Terminal
504     ↪ EOS, s), 1))
505 | Noeud
506   ( (NonTerminal MainProgram, _),
507     [
508       Noeud ((NonTerminal ProgramStmt, s),
509             ↪ 1);
```

5- Annexe

```

506         Noeud ((NonTerminal MainRange, s1),
               ↪ 11);
507     ] ) ->
508     Noeud
509     ( Syntax Program,
510       flatten
511       (Noeud
512        ( ToFlatten,
513         [
514           convert_to_abstract_aux
515           (Noeud ((NonTerminal
516                  ↪ ProgramStmt, s), 1));
517           convert_to_abstract_aux
518           (Noeud ((NonTerminal
519                  ↪ MainRange, s1), 11));
518         ] ))
519     [ ] )

```

5- Annexe

```

520 | Noeud
521   ( (NonTerminal Contains_Function, _),
522     [
523       Noeud ((Terminal Contains, _), []);
524       Noeud ((Terminal EOS, s), 1);
525       Noeud
526         ( (NonTerminal
527           ↪ FunctionSubprogram_star, _),
528           [ Noeud ((Terminal E, _), []) ] ) );
529     ] ) ->
530   convert_to_abstract_aux (Noeud ((Terminal
531     ↪ EOS, s), 1))
532 | Noeud
533   ( (NonTerminal Contains_Function, _),
534     [
535       Noeud ((Terminal Contains, _), []);
536       Noeud ((Terminal EOS, s), 1);

```

5- Annexe

```

535         Noeud ((NonTerminal
           ↪ FunctionSubprogram_star, s1), l1);
536     ] ) ->
537     Noeud
538     ( ToFlatten,
539     [
540         convert_to_abstract_aux (Noeud
           ↪ ((Terminal EOS, s), l));
541         convert_to_abstract_aux
542         (Noeud ((NonTerminal
           ↪ FunctionSubprogram_star, s1),
           ↪ l1));
543     ] )
544 | Noeud
545     ( (NonTerminal FunctionSubprogram_star, _),
546     [
547         Noeud ((Terminal Recursive, _), []);

```

5- Annexe

```

548         Noeud ((NonTerminal FunctionSubprogram,
           ↪ s), l);
549     Noeud
550         ( (NonTerminal
           ↪ FunctionSubprogram_star, _),
           [ Noeud ((Terminal E, _), []) ] );
551     ] )
552 | Noeud
553     ( (NonTerminal FunctionSubprogram_star, _),
554     [
555         Noeud ((NonTerminal FunctionSubprogram,
556             ↪ s), l);
557         Noeud
558             ( (NonTerminal
                 ↪ FunctionSubprogram_star, _),
                 [ Noeud ((Terminal E, _), []) ] );
559     ] ) ->
560

```

5- Annexe

```

561         convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal FunctionSubprogram, s),
           ↪ 1))
562     | Noeud
563         ( (NonTerminal FunctionSubprogram_star, _),
564         [
565             Noeud ((NonTerminal FunctionSubprogram,
                    ↪ s), 1);
566             Noeud ((NonTerminal
                    ↪ FunctionSubprogram_star, s1), 11);
567         ] )
568     | Noeud
569         ( (NonTerminal FunctionSubprogram_star, _),
570         [
571             Noeud ((Terminal Recursive, _), []);
572             Noeud ((NonTerminal FunctionSubprogram,
                    ↪ s), 1);

```

5- Annexe

```

573         Noeud ((NonTerminal
           ↪ FunctionSubprogram_star, s1), l1);
574     ] ) ->
575     Noeud
576     ( ToFlatten,
577     [
578         convert_to_abstract_aux
579         (Noeud ((NonTerminal
           ↪ FunctionSubprogram, s), l));
580         convert_to_abstract_aux
581         (Noeud ((NonTerminal
           ↪ FunctionSubprogram_star, s1),
           ↪ l1));
582     ] )
583 | Noeud
584     ( (NonTerminal ProgramStmt, _),
585     [

```


5- Annexe

```

586         Noeud ((Terminal Program, _), []);
587         Noeud ((NonTerminal ProgramName, s),
               ↪ 1);
588         Noeud ((Terminal EOS, s1), l1);
589     ] ) ->
590     Noeud
591     ( ToFlatten,
592     [
593         convert_to_abstract_aux (Noeud
               ↪ ((NonTerminal ProgramName, s),
               ↪ 1));
594         convert_to_abstract_aux (Noeud
               ↪ ((Terminal EOS, s1), l1));
595     ] )
596 | Noeud
597     ( (NonTerminal MainRange, _),

```

5- Annexe

```

598      [ Noeud ((NonTerminal
      ↪ Contains_Function_opt_EndProgramStmt,
      ↪ s), 1) ]
599    ) ->
600    convert_to_abstract_aux
601    (Noeud ((NonTerminal
      ↪ Contains_Function_opt_EndProgramStmt,
      ↪ s), 1))
602  | Noeud
603    ( (NonTerminal MainRange, _),
604    [
605      Noeud ((NonTerminal BodyConstruct, s),
      ↪ 1);
606      Noeud
607        ( (NonTerminal BodyConstruct_star,
      ↪ _),
608        [ Noeud ((Terminal E, _), [ ]) ] ) );

```

5- Annexe

609

```

    Noeud ((NonTerminal
        ↪ Contains_Function_opt_EndProgramStmt,
        ↪ s1), l1);

```

610

```

    ] ) ->

```

611

```

    Noeud

```

612

```

    ( ToFlatten,

```

613

```

    [

```

614

```

        convert_to_abstract_aux

```

615

```

        (Noeud ((NonTerminal BodyConstruct,
            ↪ s), l));

```

616

```

        convert_to_abstract_aux

```

617

```

        (Noeud

```

618

```

            ((NonTerminal
                ↪ Contains_Function_opt_EndProgramStmt,
                ↪ s1), l));

```

619

```

        ] )

```

620

```

    | Noeud

```

5- Annexe

```

621      ( (NonTerminal MainRange, _),
622        [
623          Noeud ((NonTerminal BodyConstruct, s),
624                ↪ 1);
625          Noeud ((NonTerminal BodyConstruct_star,
626                ↪ s2), 12);
627          Noeud ((NonTerminal
628                ↪ Contains_Function_opt_EndProgramStmt,
629                ↪ s1), 11);
630        ] ) ->
631      Noeud
632      ( ToFlatten,
633        [
634          convert_to_abstract_aux
635          (Noeud ((NonTerminal BodyConstruct,
636                ↪ s), 1));
637          Noeud

```

5- Annexe

```

633         ( ToFlatten,
634         [
635             convert_to_abstract_aux
636             (Noeud ((NonTerminal
637                     ↪ BodyConstruct_star, s2),
638                     ↪ 12)));
639         ] );
640     convert_to_abstract_aux
641     (Noeud
642     ((NonTerminal
643         ↪ Contains_Function_opt_EndProgram
644         ↪ s1), 11)));
645 ] )
646 | Noeud
647 ( (NonTerminal BodyConstruct_star, _),
648 [

```

5- Annexe

```

645         Noeud ((NonTerminal BodyConstruct, s),
               ↪ 1);
646     Noeud
647         ( (NonTerminal BodyConstruct_star,
               ↪ _),
648           [ Noeud ((Terminal E, _), []) ] );
649     ] ) ->
650     Noeud
651         ( ToFlatten,
652           [
653             convert_to_abstract_aux
654               (Noeud ((NonTerminal BodyConstruct,
655                     ↪ s), 1));
656           ] )
657 | Noeud
658     ( (NonTerminal BodyConstruct_star, _),
        [

```

5- Annexe

```

659         Noeud ((NonTerminal BodyConstruct, s),
              ↪ 1);
660         Noeud ((NonTerminal BodyConstruct_star,
              ↪ s1), l1);
661     ] ) ->
662     Noeud
663     ( ToFlatten,
664     [
665         convert_to_abstract_aux
666         (Noeud ((NonTerminal BodyConstruct,
              ↪ s), l));
667         Noeud
668         ( ToFlatten,
669         [
670             convert_to_abstract_aux

```

5- Annexe

```

671             (Noeud ((NonTerminal
                    ↪ BodyConstruct_star, s1),
                    ↪ l1)));
672             ] );
673     ] )
674 | Noeud
675     ( (NonTerminal
        ↪ Contains_Function_opt_EndProgramStmt,
        ↪ _),
676     [ Noeud ((NonTerminal EndProgramStmt, s),
        ↪ l) ] ) ->
677     convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal EndProgramStmt, s), l))
678 | Noeud
679     ( (NonTerminal
        ↪ Contains_Function_opt_EndProgramStmt,
        ↪ _),

```


5- Annexe

```

680         [
681             Noeud ((NonTerminal Contains_Function,
682                 ↪ s1), l1);
683             Noeud ((NonTerminal EndProgramStmt, s),
684                 ↪ l);
685         ] ) ->
686     Noeud
687     ( ToFlatten,
688         [
689             convert_to_abstract_aux
690             (Noeud ((NonTerminal
691                 ↪ Contains_Function, s1), l1));
692             convert_to_abstract_aux
693             (Noeud ((NonTerminal
694                 ↪ EndProgramStmt, s), l));
695         ] )
696 | Noeud

```

5- Annexe

```

693      ( (NonTerminal EndProgramStmt, _),
694        [
695          Noeud ((Terminal EndProgram, _), []);
696          Noeud ((NonTerminal EndName_opt, _),
697                ↪ _);
698          Noeud ((Terminal EOS, s), []);
699        ] ) ->
700      convert_to_abstract_aux (Noeud ((Terminal
701        ↪ EOS, s), []))
702      | Noeud
703        ( (NonTerminal BodyConstruct, _),
704          [ Noeud ((NonTerminal
705            ↪ SpecificationPartConstruct, s), 1) ]
706          ↪ ) ->
707      Noeud
708        ( ToFlatten,
709          [

```

5- Annexe

```

706         convert_to_abstract_aux
707         (Noeud ((NonTerminal
              ↪ SpecificationPartConstruct, s),
              ↪ l));
708     ] )
709 | Noeud
710     ( (NonTerminal BodyConstruct, _),
711       [ Noeud ((NonTerminal
              ↪ ExecutableConstruct, s), l) ] ) ->
712     convert_to_abstract_aux
713     (Noeud ((NonTerminal ExecutableConstruct,
              ↪ s), l))
714 | Noeud
715     ( (NonTerminal SpecificationPartConstruct,
              ↪ _),
716       [ Noeud ((NonTerminal
              ↪ DeclarationConstruct, s), l) ] ) ->

```

5- Annexe

```

717         Noeud
718         ( ToFlatten,
719         [
720             convert_to_abstract_aux
721             (Noeud ((NonTerminal
722                 ↪ DeclarationConstruct, s), 1)));
723         ] )
724 | Noeud
725     ( (NonTerminal DeclarationConstruct, _),
726     [ Noeud ((NonTerminal
727         ↪ TypeDeclarationStmt, s), 1) ] ) ->
728     Noeud
729     ( ToFlatten,
730     [
731         convert_to_abstract_aux
732         (Noeud ((NonTerminal
733             ↪ TypeDeclarationStmt, s), 1)));

```

5- Annexe

```

731         ] )
732     | Noeud
733         ( (NonTerminal TypeDeclarationStmt, _),
734         [
735             Noeud
736                 ((NonTerminal TypeSpec, _), [ Noeud
737                     ↪ ((Terminal Double, _), []) ]);
738             Noeud ((NonTerminal
739                 ↪ Comma_AttrSpec_star, _), l1);
740             Noeud ((NonTerminal
741                 ↪ TypeDecl_Assignment, s), l);
742             Noeud ((Terminal EOS, s2), []);
743         ] ) ->
744         (match l1 with
745         | [ Noeud ((Terminal E, _), []) ]
746         | [
747             Noeud ((Terminal Comma, _), []);

```

5- Annexe

```

745         Noeud
746         ( (NonTerminal AttrSpec, _),
747           [ Noeud ((NonTerminal
748                 ↪ Intent_in_out, _), _) ] );
749         Noeud
750         ( (NonTerminal Comma_AttrSpec_star,
751           ↪ _),
752           [ Noeud ((Terminal E, _), []) ] );
753     ] ->
754     ()
755 | _ -> failwith "TypeDeclarationStmt 1");
756 Noeud
757     ( ToFlatten,
758       [
759         Noeud
760         ( Syntax Double_precision,
761           [

```

5- Annexe

```

760         convert_to_abstract_aux
761         (Noeud ((NonTerminal
                  ↪ TypeDecl_Assignment, s),
                  ↪ 1));
762     ] );
763     convert_to_abstract_aux (Noeud
                              ↪ ((Terminal EOS, s2), []));
764 ] )
765 | Noeud
766   ( (NonTerminal TypeDeclarationStmt, _),
767     [
768       Noeud
769         ( (NonTerminal TypeSpec, _),
770           [
771             Noeud ((Terminal Integer, _),
772                   ↪ []);
772             Noeud

```

5- Annexe

```

773         ( (NonTerminal
              ↪ KindSelector_opt, _),
774         [ Noeud ((Terminal E, _), [])
              ↪ ] );
775     ] );
776     Noeud ((NonTerminal
              ↪ Comma_AttrSpec_star, _), l1);
777     Noeud ((NonTerminal
              ↪ TypeDecl_Assignment, s), l);
778     Noeud ((Terminal EOS, s2), []);
779     ] ) ->
780     (match l1 with
781     | [ Noeud ((Terminal E, _), []) ]
782     | [
783         Noeud ((Terminal Comma, _), []);
784         Noeud
785         ( (NonTerminal AttrSpec, _),

```


5- Annexe

```

786         [ Noeud ((NonTerminal
              ↪ Intent_in_out, _), _) ] );
787     Noeud
788     ( (NonTerminal Comma_AttrSpec_star,
        ↪ _),
789       [ Noeud ((Terminal E, _), []) ] );
790   ] ->
791     ()
792 | _ -> failwith "TypeDeclarationStmt 2");
793 Noeud
794   ( ToFlatten,
795     [
796       Noeud
797       ( Syntax Integer,
798         flatten
799         (convert_to_abstract_aux

```

5- Annexe

```

800             (Noeud ((NonTerminal
                    ↪ TypeDecl_Assignment, s),
                    ↪ 1)))
801         [] );
802     convert_to_abstract_aux (Noeud
        ↪ ((Terminal EOS, s2), []));
803 ] )
804 | Noeud
805     ( (NonTerminal TypeDeclarationStmt, _),
806     [
807         Noeud
808             ( (NonTerminal TypeSpec, _),
809             [ Noeud ((Terminal Character, _),
                    ↪ [] ) ] );
810         Noeud ((NonTerminal
        ↪ Comma_AttrSpec_star, s1), l1);

```

5- Annexe

```

811         Noeud ((NonTerminal
           ↪ TypeDecl_Assignment, s), l);
812         Noeud ((Terminal EOS, s2), []);
813     ] ) ->
814     let n1 = ref [] in
815     (match l1 with
816     | [ Noeud ((Terminal E, _), []) ]
817     | [
818         Noeud ((Terminal Comma, _), []);
819         Noeud
820             ( (NonTerminal AttrSpec, _),
821               [ Noeud ((NonTerminal
822                     ↪ Intent_in_out, _), _) ] ) );
822     Noeud
823         ( (NonTerminal Comma_AttrSpec_star,
824           ↪ _),
824         [ Noeud ((Terminal E, _), []) ] );

```

5- Annexe

```

825         ] ->
826         (
827     | _ ->
828         n1 :=
829         [
830             convert_to_abstract_aux
831             (Noeud ((NonTerminal
832                 ↪ Comma_AttrSpec_star, s1),
833                 ↪ l1)));
834     ];
835     Noeud
836     ( ToFlatten,
837     [
838         Noeud
839         ( Syntax Character,
840         flatten
841         (Noeud (ToFlatten, !n1))

```

5- Annexe

```

840         (flatten
841             (convert_to_abstract_aux
842                 (Noeud ((NonTerminal
843                     ↪ TypeDecl_Assignment,
844                     ↪ s), 1)))
845                 [])) );
846     convert_to_abstract_aux (Noeud
847         ↪ ((Terminal EOS, s2), []));
848 ] )
849 | Noeud
850     ( (NonTerminal Comma_AttrSpec_star, _),
851     [
852         Noeud ((NonTerminal Intent_in_out, _),
853             ↪ _);
854         Noeud ((NonTerminal
855             ↪ Comma_AttrSpec_star, s), 1);
856     ] ) ->

```

5- Annexe

```

852         convert_to_abstract_aux
853         (Noeud ((NonTerminal Comma_AttrSpec_star,
854                 ↪ s), 1))
855 | Noeud
856   ( (NonTerminal Comma_AttrSpec_star, _),
857     [
858       Noeud ((Terminal Parameter, _), []);
859       Noeud ((NonTerminal
860               ↪ Comma_AttrSpec_star, _), _);
861       (* the rest must be intent_in_out if
862         ↪ not empty. either way, nothing to
863         ↪ be done *)
864     ] ) ->
865     Noeud (Syntax Constant, [])
866 | Noeud
867   ( (NonTerminal TypeDeclarationStmt, _),
868     [

```

5- Annexe

```

865         Noeud
866         ( (NonTerminal TypeSpec, _),
867         [
868             Noeud ((Terminal Complex, _),
869                 ↪ []);
870             Noeud
871             ( (NonTerminal
872                 ↪ KindSelector_opt, _),
873                 [ Noeud ((Terminal E, _), [])
874                 ↪ ] );
875             ] );
876         Noeud ((NonTerminal
877             ↪ Comma_AttrSpec_star, _), 11);
878         Noeud ((NonTerminal
879             ↪ TypeDecl_Assignment, s), 1);
880         Noeud ((Terminal EOS, s2), []);
881     ] ) ->

```

5- Annexe

```

877     (match l1 with
878     | [ Noeud ((Terminal E, _), []) ]
879     | [
880         Noeud ((Terminal Comma, _), []);
881         Noeud
882             ( (NonTerminal AttrSpec, _),
883               [ Noeud ((NonTerminal
884                     ↪ Intent_in_out, _), _) ] ) );
884         Noeud
885             ( (NonTerminal Comma_AttrSpec_star,
886               ↪ _),
887               [ Noeud ((Terminal E, _), []) ] ) );
887     ] ->
888     ()
889 | _ -> failwith "TypeDeclarationStmt 3");
890 Noeud
891     ( ToFlatten,

```


5- Annexe

```

892         [
893         Noeud
894         ( Syntax Complex,
895         flatten
896         (convert_to_abstract_aux
897         (Noeud ((NonTerminal
898         ↪ TypeDecl_Assignment, s),
899         ↪ 1)))
900         [] );
901         convert_to_abstract_aux (Noeud
902         ↪ ((Terminal EOS, s2), []));
903     ] )
904 | Noeud
905     ( (NonTerminal TypeDeclarationStmt, _),
906     [
907         Noeud
908         ( (NonTerminal TypeSpec, _),

```

5- Annexe

```

906         [
907         Noeud ((Terminal Logical, _),
908             ↪  []);
909         Noeud
910         ( (NonTerminal
911             ↪  KindSelector_opt, _),
912             [ Noeud ((Terminal E, _), [])
913               ↪  ] );
914         ] );
915         Noeud ((NonTerminal
916             ↪  Comma_AttrSpec_star, _), l1);
917         Noeud ((NonTerminal
918             ↪  TypeDecl_Assignment, s), l);
919         Noeud ((Terminal EOS, s2), []);
920     ] ) ->
921     (match l1 with
922     | [ Noeud ((Terminal E, _), []) ]

```

5- Annexe

```

918 | [
919     Noeud ((Terminal Comma, _), []);
920     Noeud
921         ( (NonTerminal AttrSpec, _),
922           [ Noeud ((NonTerminal
923                 ↪ Intent_in_out, _), _) ] );
924     Noeud
925         ( (NonTerminal Comma_AttrSpec_star,
926           ↪ _),
927           [ Noeud ((Terminal E, _), []) ] );
928 ] ->
929     ()
930 | _ -> failwith "TypeDeclarationStmt 4");
931 Noeud
932     ( ToFlatten,
933       [
934         Noeud

```

5- Annexe

```

933         ( Syntax Logical,
934         flatten
935         (convert_to_abstract_aux
936         (Noeud ((NonTerminal
937                 ↪ TypeDecl_Assignment, s),
938                 ↪ 1)))
939         [] );
940     convert_to_abstract_aux (Noeud
941     ↪ ((Terminal EOS, s2), []));
942 ] )
943 | Noeud
944 ( (NonTerminal TypeDeclarationStmt, _),
945 [
946     Noeud
947     ( (NonTerminal TypeSpec, _),
948     [
949         Noeud ((Terminal Real, _), []);

```

5- Annexe

```

947         Noeud
948         ( (NonTerminal
949           ↪ KindSelector_opt, _),
950           [ Noeud ((Terminal E, _), [])
951             ↪ ] );
952     ] );
953     Noeud ((NonTerminal
954       ↪ Comma_AttrSpec_star, _), l1);
955     Noeud ((NonTerminal
956       ↪ TypeDecl_Assignment, s), l);
957     Noeud ((Terminal EOS, s2), []);
958   ] ) ->
959   (match l1 with
960   | [ Noeud ((Terminal E, _), []) ]
961   | [
962     Noeud ((Terminal Comma, _), []);
963     Noeud

```

5- Annexe

```

960         ( (NonTerminal AttrSpec, _),
961           [ Noeud ((NonTerminal
                    ↪ Intent_in_out, _), _) ] ) );
962     Noeud
963     ( (NonTerminal Comma_AttrSpec_star,
        ↪ _),
        [ Noeud ((Terminal E, _), []) ] );
964   ] ->
965     ()
966 | _ -> failwith "TypeDeclarationStmt 5");
967 Noeud
968   ( ToFlatten,
969     [
970       Noeud
971         ( Syntax Real,
972           flatten
973             (convert_to_abstract_aux
974
```

5- Annexe

```

975         (Noeud ((NonTerminal
                ↪ TypeDecl_Assignment, s),
                ↪ 1)))
976     [] );
977     convert_to_abstract_aux (Noeud
                ↪ ((Terminal EOS, s2), []));
978 ] )
979 | Noeud
980     ( (NonTerminal TypeDeclarationStmt, _),
981     [
982         Noeud
983             ( (NonTerminal TypeSpec, _),
984             [
985                 Noeud ((Terminal Integer, _),
                        ↪ []);
986                 Noeud ((NonTerminal
                        ↪ KindSelector_opt, s1), l1);

```

5- Annexe

```

987         ] );
988         Noeud ((NonTerminal
           ↪ Comma_AttrSpec_star, _), 13);
989         Noeud ((NonTerminal
           ↪ TypeDecl_Assignment, s), 1);
990         Noeud ((Terminal EOS, s2), []);
991     ] ) ->
992     (match 13 with
993     | [ Noeud ((Terminal E, _), []) ]
994     | [
995         Noeud ((Terminal Comma, _), []);
996         Noeud
997             ( (NonTerminal AttrSpec, _),
998               [ Noeud ((NonTerminal
999                   ↪ Intent_in_out, _), _) ] ) );

```


5- Annexe

```

1000      ( (NonTerminal Comma_AttrSpec_star,
        ↪ _),
1001      [ Noeud ((Terminal E, _), []) ] );
1002  ] ->
1003      ()
1004  | _ -> failwith "TypeDeclarationStmt 6");
1005  Noeud
1006      ( ToFlatten,
1007      [
1008          Noeud
1009              ( Syntax Integer,
1010                Noeud
1011                    ( Syntax Size,
1012                      [
1013                          convert_to_abstract_aux
1014                              (Noeud ((NonTerminal
                                  ↪ KindSelector_opt,
                                  ↪ s1), l1));

```

5- Annexe

```

1015         ] )
1016         :: flatten
1017         (convert_to_abstract_aux
1018         (Noeud ((NonTerminal
1019                 ↪ TypeDecl_Assignment,
1020                 ↪ s), 1)))
1019         [] );
1020         convert_to_abstract_aux (Noeud
1021         ↪ ((Terminal EOS, s2), []));
1021     ] )
1022 | Noeud
1023   ( (NonTerminal TypeDeclarationStmt, _),
1024     [
1025       Noeud
1026         ( (NonTerminal TypeSpec, _),
1027         [

```

5- Annexe

```

1028         Noeud ((Terminal Complex, _),
               ↪  []);
1029         Noeud ((NonTerminal
               ↪  KindSelector_opt, s1), l1);
1030     ] );
1031     Noeud ((NonTerminal
               ↪  Comma_AttrSpec_star, _), l3);
1032     Noeud ((NonTerminal
               ↪  TypeDecl_Assignment, s), l);
1033     Noeud ((Terminal EOS, s2), []);
1034     ] ) ->
1035     (match l3 with
1036     | [ Noeud ((Terminal E, _), []) ]
1037     | [
1038         Noeud ((Terminal Comma, _), []);
1039         Noeud
1040             ( (NonTerminal AttrSpec, _),

```

5- Annexe

```

1041         [ Noeud ((NonTerminal
               ↪ Intent_in_out, _), _) ] );
1042     Noeud
1043     ( (NonTerminal Comma_AttrSpec_star,
        ↪ _),
        [ Noeud ((Terminal E, _), []) ] );
1044     ] ->
1045     ()
1046 | _ -> failwith "TypeDeclarationStmt 7");
1047 Noeud
1048     ( ToFlatten,
        [
1049         Noeud
1050             ( Syntax Complex,
1051               Noeud
1052                   ( Syntax Size,
1053                     [

```

5- Annexe

```

1056         convert_to_abstract_aux
1057         (Noeud ((NonTerminal
                  ↪ KindSelector_opt,
                  ↪ s1), l1));
1058     ] )
1059     :: flatten
1060     (convert_to_abstract_aux
1061     (Noeud ((NonTerminal
              ↪ TypeDecl_Assignment,
              ↪ s), l)))
1062     [ ] );
1063     convert_to_abstract_aux (Noeud
                              ↪ ((Terminal EOS, s2), [ ]));
1064     ] )
1065 | Noeud
1066   ( (NonTerminal TypeDeclarationStmt, _),
1067     [

```

5- Annexe

```

1068         Noeud
1069         ( (NonTerminal TypeSpec, _),
1070         [
1071             Noeud ((Terminal Logical, _),
1072                 ↪ []);
1073             Noeud ((NonTerminal
1074                 ↪ KindSelector_opt, s1), l1);
1075             ] );
1076         Noeud ((NonTerminal
1077             ↪ Comma_AttrSpec_star, _), l3);
1078         Noeud ((NonTerminal
1079             ↪ TypeDecl_Assignment, s), l);
1080         Noeud ((Terminal EOS, s2), []);
1081     ] ) ->
1082     (match l3 with
1083     | [ Noeud ((Terminal E, _), []) ]
1084     | [

```

5- Annexe

```

1081         Noeud ((Terminal Comma, _), []);
1082     Noeud
1083         ( (NonTerminal AttrSpec, _),
1084           [ Noeud ((NonTerminal
1085                    ↪ Intent_in_out, _), _) ] ) );
1085     Noeud
1086         ( (NonTerminal Comma_AttrSpec_star,
1087            ↪ _),
1088           [ Noeud ((Terminal E, _), []) ] ) );
1088 ] ->
1089     ()
1090 | _ -> failwith "TypeDeclarationStmt 8");
1091 Noeud
1092     ( ToFlatten,
1093       [
1094         Noeud
1095             ( Syntax Logical,

```

5- Annexe

```

1096         Noeud
1097         ( Syntax Size,
1098         [
1099             convert_to_abstract_aux
1100             (Noeud ((NonTerminal
1101                 ↪ KindSelector_opt,
1102                 ↪ s1), l1));
1103         ] )
1104         :: flatten
1105         (convert_to_abstract_aux
1106         (Noeud ((NonTerminal
1107             ↪ TypeDecl_Assignment,
1108             ↪ s), l)))
1109         [] );
1110     convert_to_abstract_aux (Noeud
1111         ↪ ((Terminal EOS, s2), []));
1112 ] )

```


5- Annexe

```

1108 | Noeud
1109   ( (NonTerminal TypeDeclarationStmt, _),
1110     [
1111       Noeud
1112         ( (NonTerminal TypeSpec, _),
1113           [
1114             Noeud ((Terminal Real, _), []);
1115             Noeud ((NonTerminal
1116                   ↪ KindSelector_opt, s1), l1);
1117           ] );
1118       Noeud ((NonTerminal
1119             ↪ Comma_AttrSpec_star, _), l3);
1120       Noeud ((NonTerminal
1121             ↪ TypeDecl_Assignment, s), l);
1122       Noeud ((Terminal EOS, s2), []);
1123     ] ) ->
1124     (match l3 with

```

5- Annexe

```

1122 | [ Noeud ((Terminal E, _), []) ]
1123 | [
1124     Noeud ((Terminal Comma, _), []);
1125     Noeud
1126         ( (NonTerminal AttrSpec, _),
1127           [ Noeud ((NonTerminal
1128                 ↪ Intent_in_out, _), _) ] ) );
1129     Noeud
1130         ( (NonTerminal Comma_AttrSpec_star,
1131           ↪ _),
1132           [ Noeud ((Terminal E, _), []) ] ) );
1131 ] ->
1132 ()
1133 | _ -> failwith "TypeDeclarationStmt 9");
1134 Noeud
1135     ( ToFlatten,
1136     [

```

5- Annexe

```

1137         Noeud
1138         ( Syntax Real,
1139         Noeud
1140         ( Syntax Size,
1141         [
1142             convert_to_abstract_aux
1143             (Noeud ((NonTerminal
1144                 ↪ KindSelector_opt,
1145                 ↪ s1), l1)));
1146         ] )
1147         :: flatten
1148         (convert_to_abstract_aux
1149         (Noeud ((NonTerminal
1150             ↪ TypeDecl_Assignment,
1151             ↪ s), l)))
1152         [] );

```

5- Annexe

```

1149         convert_to_abstract_aux (Noeud
           ↪ ((Terminal EOS, s2), []));
1150     ] )
1151 | Noeud ((NonTerminal AttrSpec, _), [ Noeud
   ↪ ((Terminal Parameter, _), []) ])
1152     ->
1153     Noeud (Syntax Constant, [])
1154 | Noeud
   ( (NonTerminal Comma_EntityDecl_star, _),
1155   [
1156       Noeud ((Terminal Comma, _), []);
1157       Noeud ((NonTerminal EntityDecl, s), 1);
1158       Noeud
1159           ( (NonTerminal Comma_EntityDecl_star,
1160             ↪ _),
1161           [ Noeud ((Terminal E, _), []) ] );
1162   ] )

```

5- Annexe

```

1163 | Noeud
1164 |   ( (NonTerminal TypeDecl_Assignment, _),
1165 |     [
1166 |       Noeud ((Terminal Colon, _), []);
1167 |       Noeud ((Terminal Colon, _), []);
1168 |       Noeud ((NonTerminal EntityDecl, s), 1);
1169 |       Noeud
1170 |         ( (NonTerminal Comma_EntityDecl_star,
1171 |           ↪ _),
1172 |           [ Noeud ((Terminal E, _), []) ] ) );
1173 |   ] ) ->
1174 |   convert_to_abstract_aux (Noeud
1175 |     ↪ ((NonTerminal EntityDecl, s), 1))
1176 | | Noeud
1177 |   ( (NonTerminal Comma_EntityDecl_star, _),
1178 |     [
1179 |       Noeud ((Terminal Comma, _), []);

```

5- Annexe

```

1178         Noeud ((NonTerminal EntityDecl, s), 1);
1179         Noeud ((NonTerminal
           ↪ Comma_EntityDecl_star, s2), 12);
1180     ] )
1181 | Noeud
1182     ( (NonTerminal TypeDecl_Assignment, _),
1183     [
1184         Noeud ((Terminal Colon, _), []);
1185         Noeud ((Terminal Colon, _), []);
1186         Noeud ((NonTerminal EntityDecl, s), 1);
1187         Noeud ((NonTerminal
           ↪ Comma_EntityDecl_star, s2), 12);
1188     ] ) ->
1189     Noeud
1190         ( ToFlatten,
1191         [

```

5- Annexe

```

1192         convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal EntityDecl, s),
        ↪ 1));
1193     Noeud
1194         ( ToFlatten,
1195         [
1196             convert_to_abstract_aux
1197                 (Noeud ((NonTerminal
        ↪ Comma_EntityDecl_star,
        ↪ s2), 12));
1198         ] );
1199     ] )
1200 | Noeud
1201     ( (NonTerminal Comma_ObjectName_star, _),
1202     [
1203         Noeud ((Terminal Comma, _), []);
1204         Noeud ((NonTerminal ObjectName, s), 1);

```

5- Annexe

```

1205         Noeud
1206         ( (NonTerminal Comma_ObjectName_star,
           ↪ _),
1207           [ Noeud ((Terminal E, _), []) ] );
1208     ] )
1209 | Noeud
1210     ( (NonTerminal TypeDecl_Assignment, _),
1211       [
1212         Noeud ((NonTerminal ObjectName, s), 1);
1213         Noeud
1214         ( (NonTerminal Comma_ObjectName_star,
           ↪ _),
1215           [ Noeud ((Terminal E, _), []) ] );
1216       ] ) ->
1217     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal ObjectName, s), 1))
1218 | Noeud

```


5- Annexe

```

1219         ( (NonTerminal Comma_ObjectName_star, _),
1220         [
1221             Noeud ((Terminal Comma, _), []);
1222             Noeud ((NonTerminal ObjectName, s), 1);
1223             Noeud ((NonTerminal
                ↪ Comma_ObjectName_star, s1), 11);
1224         ] )
1225 | Noeud
1226     ( (NonTerminal TypeDecl_Assignment, _),
1227     [
1228         Noeud ((NonTerminal ObjectName, s), 1);
1229         Noeud ((NonTerminal
                ↪ Comma_ObjectName_star, s1), 11);
1230     ] ) ->
1231 Noeud
1232     ( ToFlatten,
1233     [

```

5- Annexe

```

1234         convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal ObjectName, s),
        ↪ 1));
1235     convert_to_abstract_aux
1236         (Noeud ((NonTerminal
        ↪ Comma_ObjectName_star, s1),
        ↪ 11));
1237     ] )
1238 | Noeud
1239     ( (NonTerminal EntityDecl, _),
1240     [
1241         Noeud ((NonTerminal ObjectName, s), 1);
1242         Noeud
1243             ( (NonTerminal
        ↪ Asterisk_CharLength_opt, _),
1244             [ Noeud ((Terminal E, _), []) ] );
1245         Noeud

```

5- Annexe

```

1246         ((NonTerminal Equal_Expr_opt, _), [
           ↪ Noeud ((Terminal E, _), [])]);
1247     ] ) ->
1248     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal ObjectName, s), l))
1249 | Noeud
           ( (NonTerminal EntityDecl, _),
1250           [
1251             Noeud
1252               ((NonTerminal ObjectName, _), [ Noeud
                 ↪ ((Terminal Ident, s), [])]);
1253             Noeud
1254               ( (NonTerminal
                 ↪ Asterisk_CharLength_opt, _),
1255               [
1256                 Noeud ((Terminal Asterisk, _),
1257                   ↪ []);

```

5- Annexe

```

1258         Noeud ((NonTerminal CharLength,
                ↪ s1), l1);
1259     ] );
1260     Noeud
1261     ((NonTerminal Equal_Expr_opt, _), [
        ↪ Noeud ((Terminal E, _), []) ]);
1262 ] ) ->
1263 Noeud
1264 ( ToFlatten,
1265 [
1266     Noeud
1267     ( Syntax Size,
1268     [
1269         convert_to_abstract_aux
1270         (Noeud ((NonTerminal
                ↪ CharLength, s1), l1));
1271     ] );

```

5- Annexe

```

1272         Noeud (Name s, []);
1273     ] )
1274 | Noeud
1275     ( (NonTerminal EntityDecl, _),
1276     [
1277         Noeud
1278         ((NonTerminal ObjectName, _), [ Noeud
1279             ↪ ((Terminal Ident, s), []) ]));
1280         Noeud
1281         ( (NonTerminal
1282             ↪ Asterisk_CharLength_opt, _),
1283             [ Noeud ((Terminal E, _), []) ] ) );
1284 | Noeud
1285     ( (NonTerminal Equal_Expr_opt, _),
1286     [
1287         Noeud ((Terminal Equal, _), []);
1288         Noeud ((NonTerminal Expr, s2),
1289             ↪ 12);

```

5- Annexe

```

1287         ] );
1288     ] ) ->
1289     Noeud
1290     ( Operateur Assignment,
1291     [
1292         Noeud (Name s, []);
1293         convert_to_abstract_aux (Noeud
1294             ↪ ((NonTerminal Expr, s2), 12));
1295     ] )
1296 | Noeud
1297     ( (NonTerminal EntityDecl, _),
1298     [
1299         Noeud
1300             ((NonTerminal ObjectName, _), [ Noeud
1301                 ↪ ((Terminal Ident, s), []) ]);
1302         Noeud
1303             ( (NonTerminal
1304                 ↪ Asterisk_CharLength_opt, _),

```

5- Annexe

```

1302         [
1303         Noeud ((Terminal Asterisk, _),
1304             ↪  []);
1305         Noeud ((NonTerminal CharLength,
1306             ↪  s1), l1);
1307     ] );
1308     Noeud
1309     ( (NonTerminal Equal_Expr_opt, _),
1310     [
1311         Noeud ((Terminal Equal, _), []);
1312         Noeud ((NonTerminal Expr, s2),
1313             ↪  l2);
1314     ] );
1315     ] ) ->
1316     Noeud
1317     ( Operateur Assignment,
1318     [

```

5- Annexe

```

1316         Noeud
1317         ( Syntax Size,
1318         [
1319             convert_to_abstract_aux
1320             (Noeud ((NonTerminal
1321                 ↪ CharLength, s1), 11));
1321             ] );
1322         Noeud (Name s, []);
1323         convert_to_abstract_aux (Noeud
1324             ↪ ((NonTerminal Expr, s2), 12));
1324     ] )
1325 | Noeud
1326     ( (NonTerminal CharLength, _),
1327     [
1328         Noeud ((Terminal LParenthesis, _), []);
1329         Noeud ((NonTerminal TypeParamValue, s),
1330             ↪ 1);

```


5- Annexe

```

1330         Noeud ((Terminal RParenthesis, _), []);
1331     ] ) ->
1332     convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal TypeParamValue, s), 1))
1333 | Noeud
        ( (NonTerminal CharLength, _),
1334     [ Noeud ((NonTerminal
        ↪ ScalarIntLiteralConstant, s), 1) ] )
1335     ↪ ->
1336     convert_to_abstract_aux
1337     (Noeud ((NonTerminal
        ↪ ScalarIntLiteralConstant, s), 1))
1338 | Noeud
        ( (NonTerminal TypeParamValue, _),
1339     [ Noeud ((NonTerminal Expr_Or_Asterisk,
1340     ↪ s), 1) ] ) ->

```

5- Annexe

```

1341     convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal Expr_Or_Asterisk, s), 1))
1342 | Noeud
      ( (NonTerminal Expr_Or_Asterisk, _),
1343       [ Noeud ((Terminal Asterisk, _), []) ] )
1344     ↪ ->
      Noeud (Syntax Any, [])
1345 | Noeud
      ((NonTerminal Expr_Or_Asterisk, _), [ Noeud
1346       ↪ ((NonTerminal Expr, s), 1) ])
1347     ->
1348     convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal Expr, s), 1))
1349 | Noeud
      ( (NonTerminal KindSelector_opt, _),
1350       [
1351         Noeud ((Terminal LParenthesis, _), []);
1352
1353

```

5- Annexe

```

1354         Noeud ((NonTerminal Expr, s), l);
1355         Noeud ((Terminal RParenthesis, _), []);
1356     ] ) ->
1357     convert_to_abstract_aux (Noeud
1358     ↪ ((NonTerminal Expr, s), l))
1359 | Noeud ((NonTerminal ExecutableConstruct, _),
1360 ↪ l) -> (
1361     match l with
1362     | [ Noeud ((NonTerminal ActionStmt, s1),
1363     ↪ l1) ] ->
1364         Noeud
1365         ( ToFlatten,
1366         [
1367             convert_to_abstract_aux
1368             (Noeud ((NonTerminal
1369             ↪ ActionStmt, s1), l1));
1370         ] )

```

5- Annexe

```
1367 | [ Noeud ((NonTerminal DoConstruct, s1),  
    ↪ 11) ] ->  
1368     Noeud  
1369       ( ToFlatten,  
1370         [  
1371           convert_to_abstract_aux  
1372             (Noeud ((NonTerminal  
    ↪ DoConstruct, s1), 11));  
1373           ] )  
1374 | [ Noeud ((NonTerminal IfConstruct, s1),  
    ↪ 11) ] ->  
1375     Noeud  
1376       ( ToFlatten,  
1377         [  
1378           convert_to_abstract_aux  
1379             (Noeud ((NonTerminal  
    ↪ IfConstruct, s1), 11));
```

5- Annexe

```

1380         ] )
1381     | [
1382         Noeud
1383         ( (NonTerminal ReturnStmt, _),
1384           [ Noeud ((Terminal Return, _), []);
1385             ↪ Noeud ((Terminal EOS, s), 1) ]
1386         );
1387     ] ->
1388         Noeud
1389         ( ToFlatten,
1390           [
1391             Noeud (Syntax Return, []);
1392             convert_to_abstract_aux (Noeud
1393               ↪ ((Terminal EOS, s), 1));
1394           ] )
1395     | _ -> failwith "ExecutableConstruct"
1396 | Noeud

```

5- Annexe

```

1395      ( (NonTerminal ActionStmt, _),
1396        [ Noeud ((NonTerminal AssignmentStmt, s),
1397                ↪ 1) ] ) ->
1398      Noeud
1399      ( ToFlatten,
1400        [
1401          convert_to_abstract_aux
1402          (Noeud ((NonTerminal
1403                ↪ AssignmentStmt, s), 1));
1404        ] )
1405  | Noeud
1406    ((NonTerminal ActionStmt, _), [ Noeud
1407      ↪ ((NonTerminal PrintStmt, s), 1) ] )
1408  ->
1409    Noeud
1410    ( ToFlatten,

```

5- Annexe

```

1408         [ convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal PrintStmt, s), 1)) ]
1409     )
1410 | Noeud
1411     ( (NonTerminal AssignmentStmt, _),
1412       [
1413         Noeud ((NonTerminal Name, s), 1);
1414         Noeud ((Terminal Equal, _), []);
1415         Noeud ((NonTerminal Expr, s1), 11);
1416         Noeud ((Terminal EOS, s2), []);
1417       ] ) ->
1418     Noeud
1419       ( ToFlatten,
1420         [
1421           Noeud
1422             ( Operateur Assignment,
1423               [

```

5- Annexe

```

1424         convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal Name, s),
        ↪ 1));
1425         convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal Expr, s1),
        ↪ 11));
1426     ] );
1427     convert_to_abstract_aux (Noeud
        ↪ ((Terminal EOS, s2), []));
1428 ] )
1429 | Noeud
1430 ( (NonTerminal PrintStmt, _),
1431 [
1432     Noeud ((Terminal Print, _), []);
1433     Noeud ((NonTerminal FormatIdentifier,
        ↪ _), _);
1434     Noeud

```


5- Annexe

```

1435         ( (NonTerminal
           ↪ Comma_OutputItemList_opt, _),
1436         [ Noeud ((Terminal E, _), []) ] );
1437     Noeud ((Terminal EOS, s2), []);
1438 ] ) ->
1439 Noeud
1440     ( ToFlatten,
1441     [
1442         Noeud (Syntax Print, [ Noeud (Chaine
           ↪ "", []) ] );
1443         convert_to_abstract_aux (Noeud
           ↪ ((Terminal EOS, s2), []));
1444     ] )
1445 | Noeud
1446     ( (NonTerminal PrintStmt, _),
1447     [
1448         Noeud ((Terminal Print, _), []);

```

5- Annexe

```

1449         Noeud ((NonTerminal FormatIdentifier,
               ↪ _), _);
1450         Noeud ((NonTerminal
               ↪ Comma_OutputItemList_opt, s), l);
1451         Noeud ((Terminal EOS, s2), []);
1452     ] ) ->
1453     Noeud
1454     ( ToFlatten,
1455       [
1456         Noeud
1457         ( Syntax Print,
1458           flatten
1459           (convert_to_abstract_aux
1460            (Noeud ((NonTerminal
1461                  ↪ Comma_OutputItemList_opt,
1462                  ↪ s), l)))
1461           [] );

```

5- Annexe

```

1462         convert_to_abstract_aux (Noeud
           ↪ ((Terminal EOS, s2), []));
1463     ] )
1464 | Noeud
1465     ( (NonTerminal Comma_OutputItemList_opt,
           ↪ _),
1466     [
1467         Noeud ((Terminal Comma, _), []);
1468         Noeud ((NonTerminal OutputItemList, s),
           ↪ 1);
1469     ] ) ->
1470     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal OutputItemList, s), 1))
1471 | Noeud
1472     ( (NonTerminal Comma_OutputItem_star, _),
1473     [
1474         Noeud ((Terminal Comma, _), []);

```

5- Annexe

```

1475         Noeud ((NonTerminal OutputItem, s), 1);
1476     Noeud
1477         ( (NonTerminal Comma_OutputItem_star,
            ↪ _),
            [ Noeud ((Terminal E, _), []) ] );
1478     ] )
1479 | Noeud
1480     ( (NonTerminal OutputItemList, _),
1481       [
1482         Noeud ((NonTerminal OutputItem, s), 1);
1483         Noeud
1484             ( (NonTerminal Comma_OutputItem_star,
                ↪ _),
                [ Noeud ((Terminal E, _), []) ] );
1485         ] ) ->
1486     convert_to_abstract_aux (Noeud
1487         ↪ ((NonTerminal OutputItem, s), 1))

```

5- Annexe

```

1489 | Noeud
1490   ( (NonTerminal Comma_OutputItem_star, _),
1491     [
1492       Noeud ((Terminal Comma, _), []);
1493       Noeud ((NonTerminal OutputItem, s), l);
1494       Noeud ((NonTerminal
           ↪ Comma_OutputItem_star, s1), l1);
1495     ] )
1496 | Noeud
1497   ( (NonTerminal OutputItemList, _),
1498     [
1499       Noeud ((NonTerminal OutputItem, s), l);
1500       Noeud ((NonTerminal
           ↪ Comma_OutputItem_star, s1), l1);
1501     ] ) ->
1502   Noeud
1503     ( ToFlatten,

```

5- Annexe

```

1504         [
1505             convert_to_abstract_aux (Noeud
1506                 ↪ ((NonTerminal OutputItem, s),
1507                 ↪ l));
1508             convert_to_abstract_aux
1509                 (Noeud ((NonTerminal
1510                     ↪ Comma_OutputItem_star, s1),
1511                     ↪ l1));
1512         ] )
1513 | Noeud ((NonTerminal OutputItem, _), [ Noeud
1514     ↪ ((NonTerminal Expr, s), l) ])
1515     ->
1516         convert_to_abstract_aux (Noeud
1517             ↪ ((NonTerminal Expr, s), l))
1518 | Noeud
1519     ( (NonTerminal DoConstruct, _),

```

5- Annexe

```

1514      [ Noeud ((NonTerminal BlockDoConstruct,
           ↪ s), l) ] ) ->
1515      convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal BlockDoConstruct, s), l))
1516 | Noeud
           ( (NonTerminal BlockDoConstruct, _),
1517           [
1518             Noeud ((Terminal Do, _), []);
1519             Noeud ((NonTerminal LoopControl_opt,
1520                 ↪ _), l);
1521             Noeud
1522                 ( (NonTerminal
                     ↪ ExecutionPartConstruct_star, _),
                     [ Noeud ((Terminal E, _), []) ] );
1523             Noeud ((NonTerminal EndDoStmt, _), _);
1524           ] ) -> (
1525
1526      match l with

```

5- Annexe

```

1527 | [ Noeud ((Terminal EOS, s), []) ] ->
1528     Noeud
1529     ( Syntax While,
1530       [
1531         Noeud (Booleen true, []);
1532         convert_to_abstract_aux (Noeud
1533           ↪ ((Terminal EOS, s), []));
1534       ] )
1535 | [
1536   Noeud ((NonTerminal LoopControl, s1), l1);
1537   ↪ Noeud ((Terminal EOS, s), []);
1538 ] -> (
1539     match l1 with
1540     | Noeud ((Terminal While, _), _) :: _
1541       ↪ ->
1542         Noeud
1543         ( Syntax While,

```


5- Annexe

```

1541         flatten
1542         (convert_to_abstract_aux
1543         (Noeud ((NonTerminal
1544                 ↪ LoopControl, s1),
1545                 ↪ 11)))
1546     [
1547         convert_to_abstract_aux
1548         ↪ (Noeud ((Terminal EOS,
1549                 ↪ s), []));
1550     ] )
1551 | Noeud ((NonTerminal VariableName, _),
1552 ↪ _ ) :: _ ->
1553     Noeud
1554     ( Syntax For,
1555       flatten
1556       (convert_to_abstract_aux

```

5- Annexe

```

1552         (Noeud ((NonTerminal
                ↪ LoopControl, s1),
                ↪ 11)))
1553     [
1554         convert_to_abstract_aux
                ↪ (Noeud ((Terminal EOS,
                ↪ s), []));
1555     ] )
1556     | _ -> failwith "BlockDoConstruct 1"
1557     | _ -> failwith "BlockDoConstruct 2")
1558 | Noeud
1559   ( (NonTerminal BlockDoConstruct, _),
1560     [
1561       Noeud ((Terminal Do, _), []);
1562       Noeud ((NonTerminal LoopControl_opt,
                ↪ _), 1);

```

5- Annexe

```

1563         Noeud ((NonTerminal
           ↪ ExecutionPartConstruct_star, s1),
           ↪ 11));
1564         Noeud ((NonTerminal EndDoStmt, _), _);
1565     ] ) -> (
1566 match l with
1567 | [ Noeud ((Terminal EOS, s), []) ] ->
1568     Noeud
1569         ( Syntax While,
1570         [
1571             Noeud (Booleen true, []);
1572             convert_to_abstract_aux (Noeud
1573                 ↪ ((Terminal EOS, s), []));
1573             convert_to_abstract_aux
1574                 (Noeud ((NonTerminal
1575                     ↪ ExecutionPartConstruct_star,
1576                     ↪ s1), 11));

```

5- Annexe

1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587

```

    ] )
  | [
    Noeud ((NonTerminal LoopControl, s2), 12);
    ↪ Noeud ((Terminal EOS, s), []);
  ] -> (
    match 12 with
    | Noeud ((Terminal While, _), _) :: _
    ↪ ->
      Noeud
        ( Syntax While,
          flatten
            (Noeud
              ( ToFlatten,
                [
                  ↪ convert_to_abstract_au

```

5- Annexe

1588

```
(Noeud ((NonTerminal
  ↪ LoopControl,
  ↪ s2), 12));
```

1589

```
↪ convert_to_abstract_aui
```

1590

```
(Noeud ((Terminal
  ↪ EOS, s), []));
```

1591

Noeud

1592

```
( ToFlatten,
```

1593

flatten

1594

```
↪ (convert_to_abstract
```

1595

(Noeud

1596

((

↪ NonTerminal

1597

↪ Exec

5- Annexe

```

1598                                     s1 ),
1599                                     11 )))
1600                                     [] );
1601                                     ] ))
1602                                     [] )
1603 | Noeud ((NonTerminal VariableName, _),
↪   _ ) :: _ ->
1604   Noeud
1605     ( Syntax For,
1606       flatten
1607         (Noeud
1608           ( ToFlatten,
1609             [
1610
↪       convert_to_abstract_au

```

5- Annexe

```

1611      (Noeud ((NonTerminal
           ↪ LoopControl,
           ↪ s2), 12)));

1612
           ↪ convert_to_abstract_aui
1613      (Noeud ((Terminal
           ↪ EOS, s), []));
1614      Noeud
           ( ToFlatten,
1615            flatten
1616
1617
           ↪ (convert_to_abstract
1618            (Noeud
1619              ( (
                 ↪ NonTerminal
1620
                 ↪ Exec

```

5- Annexe

```

1621                                     s1 ),
1622                                     11 )))
1623                                     [] );
1624                                     ] ))
1625                                     [] )
1626         | _ -> failwith "BlockDoConstruct 3"
1627     | _ -> failwith "BlockDoConstruct 4"
1628 | Noeud
1629     ( (NonTerminal LoopControl, _),
1630       [
1631         Noeud ((Terminal While, _), []);
1632         Noeud ((Terminal LParenthesis, _), []);
1633         Noeud ((NonTerminal Expr, s), 1);
1634         Noeud ((Terminal RParenthesis, _), []);
1635       ] ) ->
1636     convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal Expr, s), 1))

```


5- Annexe

```

1637 | Noeud
1638   ( (NonTerminal LoopControl, _),
1639     [
1640       Noeud ((NonTerminal VariableName, s),
1641             ↪ 1);
1642       Noeud ((Terminal Equal, _), []);
1643       Noeud ((NonTerminal
1644             ↪ IntRealDpExpression, s1), 11);
1645       Noeud ((Terminal Comma, _), []);
1646       Noeud ((NonTerminal
1647             ↪ IntRealDpExpression, s2), 12);
1648       Noeud ((NonTerminal
1649             ↪ Comma_IntRealDpExpression_opt, _),
1650             ↪ 13);
1651     ] ) -> (
1652     match 13 with
1653     | [ Noeud ((Terminal E, _), []) ] ->

```

5- Annexe

```
1649         Noeud
1650         ( ToFlatten,
1651         [
1652             Noeud
1653             ( Operateur Assignment,
1654             [
1655                 convert_to_abstract_aux
1656                 (Noeud ((NonTerminal
1657                     ↪ VariableName, s),
1658                     ↪ 1));
1659                 convert_to_abstract_aux
1660                 (Noeud ((NonTerminal
1661                     ↪ IntRealDpExpression,
1662                     ↪ s1), 11));
1663             ] );
1664         convert_to_abstract_aux
```

5- Annexe

```

1661         (Noeud ((NonTerminal
                ↪ IntRealDpExpression, s2),
                ↪ 12));
1662     Noeud (Syntax Step, [ Noeud
                ↪ (Integer "1", []) ]);
1663     ] )
1664 | [
1665     Noeud ((Terminal Comma, _), _);
1666     Noeud ((NonTerminal IntRealDpExpression,
                ↪ s4), 14);
1667 ] ->
1668     Noeud
1669     ( ToFlatten,
1670     [
1671         Noeud
1672         ( Operateur Assigination,
1673         [

```

5- Annexe

```
1674         convert_to_abstract_aux
1675         (Noeud ((NonTerminal
                  ↪ VariableName, s),
                  ↪ 1));
1676         convert_to_abstract_aux
1677         (Noeud ((NonTerminal
                  ↪ IntRealDpExpression,
                  ↪ s1), 11));
1678     ] );
1679     convert_to_abstract_aux
1680     (Noeud ((NonTerminal
              ↪ IntRealDpExpression, s2),
              ↪ 12));
1681     Noeud
1682     ( Syntax Step,
1683     [
1684         convert_to_abstract_aux
```

5- Annexe

```

1685         (Noeud ((NonTerminal
                ↪ IntRealDpExpression,
                ↪ s4), 14));
1686     ] );
1687 ] )
1688 | _ -> failwith "LoopControl")
1689 | Noeud
1690   ( (NonTerminal IntRealDpExpression, _),
1691     [ Noeud ((NonTerminal Expr, s), 1) ] ) ->
1692   convert_to_abstract_aux (Noeud
                ↪ ((NonTerminal Expr, s), 1))
1693 | Noeud
1694   ( (NonTerminal IfConstruct, _),
1695     [
1696       Noeud ((NonTerminal IfThenStmt, _), 1);

```

5- Annexe

1697

```

Noeud ((NonTerminal
    ↪ ExecutionPartConstruct_star, s1),
    ↪ 11);

```

1698

Noeud

1699

```

((NonTerminal
    ↪ ElseIfStmt_ExecutionPartConstruct_star,
    ↪ s2), 12);

```

1700

Noeud

1701

```

((NonTerminal
    ↪ ElseStmt_ExecutionPartConstruct_star,
    ↪ s3), 13);

```

1702

Noeud

1703

```

( (NonTerminal EndIfStmt, _),

```

1704

[

1705

```

    Noeud ((Terminal EndIf, _), []);
    ↪ Noeud ((Terminal EOS, s4),
    ↪ []);

```

5- Annexe

```

1706         ] );
1707     ] ) -> (
1708     let next =
1709         ref [ convert_to_abstract_aux (Noeud
1710             ↪ ((Terminal EOS, s4), [])) ]
1711     in
1712     let inner = ref [] in
1713     (match l3 with
1714     | [ Noeud ((Terminal E, _), []) ] -> ()
1715     | _ ->
1716         next :=
1717             convert_to_abstract_aux
1718                 (Noeud
1719                     ( (NonTerminal
1720                         ↪ ElseStmt_ExecutionPartConstruct_
1721                         ↪ s3),
1722                     13 ))

```

5- Annexe

```

1720         :: !next);
1721     (match 12 with
1722     | [ Noeud ((Terminal E, _), []) ] -> ()
1723     | _ ->
1724         next :=
1725             convert_to_abstract_aux
1726                 (Noeud
1727                     ( ( NonTerminal
1728                         ↪ ElseIfStmt_ExecutionPartConstruct
1729                           s2 ),
1730                       12 ))
1731         :: !next);
1732     (match 11 with
1733     | [ Noeud ((Terminal E, _), []) ] -> ()
1734     | _ ->
1735         inner :=
1736             flatten

```


5- Annexe

```

1736             (convert_to_abstract_aux
1737             (Noeud ((NonTerminal
                     ↪ ExecutionPartConstruct_star,
                     ↪ s1), l1)))
1738             !inner);
1739 match l with
1740 | [
1741   Noeud ((Terminal If, _), []);
1742   Noeud ((Terminal LParenthesis, _), []);
1743   Noeud ((NonTerminal ScalarLogicalExpr, s),
           ↪ l);
1744   Noeud ((Terminal RParenthesis, _), []);
1745   Noeud ((Terminal Then, _), []);
1746   Noeud ((Terminal EOS, s5), []);
1747 ] ->
1748     Noeud
1749     ( ToFlatten,

```

5- Annexe

```

1750         Noeud
1751         ( Syntax If,
1752           convert_to_abstract_aux
1753           (Noeud ((NonTerminal
1754                  ↪ ScalarLogicalExpr, s),
1755                  ↪ 1))
1756           :: flatten
1757           (convert_to_abstract_aux
1758            (Noeud ((Terminal EOS,
1759                    ↪ s5), [])))
1760            !inner )
1761           :: !next )
1762         | _ -> failwith "IfConstruct")
1763 | Noeud
1764   ( (NonTerminal
1765     ↪ ElseIfStmt_ExecutionPartConstruct_star_star
1766     ↪ _),

```

5- Annexe

```

1762     [
1763         Noeud ((NonTerminal ElseIfStmt, s), l);
1764         Noeud ((NonTerminal
            ↪ ExecutionPartConstruct_star, s1),
            ↪ l1);
1765     ] ) -> (
1766     let inner = ref [] in
1767     (match l1 with
1768     | [ Noeud ((Terminal E, _), []) ] -> ()
1769     | _ ->
1770         inner :=
1771             flatten
1772                 (convert_to_abstract_aux
1773                     (Noeud ((NonTerminal
                            ↪ ExecutionPartConstruct_star,
                            ↪ s1), l1)))
1774         !inner);

```

5- Annexe

```

1775     match l with
1776     | [
1777         Noeud ((Terminal Else, _), []);
1778         Noeud ((Terminal If, _), []);
1779         Noeud ((Terminal LParenthesis, _), []);
1780         Noeud ((NonTerminal ScalarLogicalExpr,
1781             ↪ s1), l1);
1782         Noeud ((Terminal RParenthesis, _), []);
1783         Noeud ((Terminal Then, _), []);
1784         Noeud ((Terminal EOS, s2), []);
1785     ] ->
1786         Noeud
1787         ( Syntax Else_if,
1788           convert_to_abstract_aux
1789             (Noeud ((NonTerminal
1790                 ↪ ScalarLogicalExpr, s1), l1))
1791             :: convert_to_abstract_aux (Noeud
1792                 ↪ ((Terminal EOS, s2), []))

```

5- Annexe

```

1790         :: !inner )
1791     | _ -> failwith
1792     ↪ "ElseIfStmt_ExecutionPartConstruct_star_star"
1793 | Noeud
1794 ( (NonTerminal ExecutionPartConstruct_star,
1795   ↪ _),
1796   [
1797     Noeud ((NonTerminal
1798             ↪ ExecutionPartConstruct, s), l);
1799     Noeud
1800       ( (NonTerminal
1801         ↪ ExecutionPartConstruct_star, _),
1802         [ Noeud ((Terminal E, _), []) ] ) );
1803   ] ) ->
1804 convert_to_abstract_aux
1805 (Noeud ((NonTerminal
1806         ↪ ExecutionPartConstruct, s), l))

```

5- Annexe

```
1802 | Noeud
1803   ( (NonTerminal ExecutionPartConstruct_star,
1804     ↪ _),
1805     [
1806       Noeud ((NonTerminal
1807         ↪ ExecutionPartConstruct, s), l);
1808       Noeud ((NonTerminal
1809         ↪ ExecutionPartConstruct_star, s1),
1810         ↪ l1));
1811   ] ) ->
1812   Noeud
1813     ( ToFlatten,
1814       [
1815         convert_to_abstract_aux
1816         (Noeud ((NonTerminal
1817           ↪ ExecutionPartConstruct, s),
1818             ↪ l));
```

5- Annexe

```

1813         convert_to_abstract_aux
1814         (Noeud ((NonTerminal
                  ↪ ExecutionPartConstruct_star,
                  ↪ s1), l1));
1815     ] )
1816 | Noeud
1817   ( (NonTerminal
      ↪ ElseStmt_ExecutionPartConstruct_star_opt,
      ↪ _),
1818     [
1819       Noeud ((NonTerminal ElseStmt, _), l);
1820       Noeud ((NonTerminal
                  ↪ ExecutionPartConstruct_star, s1),
                  ↪ l1);
1821     ] ) -> (
1822     let inner = ref [] in
1823     (match l1 with

```

5- Annexe

```

1824 | [ Noeud ((Terminal E, _), []) ] -> ()
1825 | _ ->
1826     inner :=
1827         convert_to_abstract_aux
1828         (Noeud ((NonTerminal
1829             ↪ ExecutionPartConstruct_star,
1830             ↪ s1), l1))
1831         :: !inner);
1832 match l with
1833 | [ Noeud ((Terminal Else, _), []); Noeud
1834     ↪ ((Terminal EOS, s2), []) ] ->
1835     Noeud
1836         ( Syntax Else,
1837         convert_to_abstract_aux (Noeud
1838             ↪ ((Terminal EOS, s2), []))
1839         :: !inner )

```


5- Annexe

```

1836         | _ -> failwith
           ↪ "ElseStmt_ExecutionPartConstruct_star_opt")
1837     | Noeud
1838         ( (NonTerminal ExecutionPartConstruct, _),
1839           [ Noeud ((NonTerminal
           ↪ ExecutableConstruct, s), l) ] ) ->
1840         Noeud
1841             ( ToFlatten,
1842               [
1843                 convert_to_abstract_aux
1844                     (Noeud ((NonTerminal
1845                               ↪ ExecutableConstruct, s), l));
1846               ] )
1847     | Noeud
1848         ( (NonTerminal ScalarLogicalExpr, _),
           [ Noeud ((NonTerminal Expr, s), l) ] ) ->

```

5- Annexe

```

1849         convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal Expr, s), 1))
1850 | Noeud
1851     ( (NonTerminal ScalarIntLiteralConstant,
           ↪ _),
1852       [ Noeud ((Terminal Icon, s), 1) ] ) ->
1853       Noeud (Integer s, [])
1854 | Noeud ((NonTerminal Expr, _), [ Noeud
           ↪ ((NonTerminal Level5Expr, s), 1) ])
1855     ->
1856     convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal Level5Expr, s), 1))
1857 | Noeud
1858     ( (NonTerminal EquivOp_EquivOperand_star,
           ↪ _),
1859       [
1860         Noeud ((NonTerminal EquivOp, _), _);

```

5- Annexe

```

1861         Noeud ((NonTerminal EquivOperand, s),
               ↪ 1);
1862     Noeud
1863         ( (NonTerminal
               ↪ EquivOp_EquivOperand_star, _),
           [ Noeud ((Terminal E, _), []) ] );
1864     ] )
1865 | Noeud
1866     ( (NonTerminal Level5Expr, _),
       [
1867         Noeud ((NonTerminal EquivOperand, s),
               ↪ 1);
1868         Noeud
1869             ( (NonTerminal
                   ↪ EquivOp_EquivOperand_star, _),
               [ Noeud ((Terminal E, _), []) ] );
1870     ] ) ->

```

5- Annexe

```

1874         convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal EquivOperand, s), 1))
1875 | Noeud
1876   ( (NonTerminal EquivOp_EquivOperand_star,
           ↪ _),
1877     [
1878       Noeud ((NonTerminal EquivOp, _), _);
1879       Noeud ((NonTerminal EquivOperand, s),
           ↪ 1);
1880       Noeud ((NonTerminal
           ↪ EquivOp_EquivOperand_star, s1),
           ↪ 11);
1881     ] )
1882 | Noeud
1883   ( (NonTerminal Level5Expr, _),
1884     [

```

5- Annexe

```

1885         Noeud ((NonTerminal EquivOperand, s),
               ↪ 1);
1886         Noeud ((NonTerminal
               ↪ EquivOp_EquivOperand_star, s1),
               ↪ l1);
1887     ] ) -> (
1888     match l1 with
1889     | Noeud
1890         ((NonTerminal EquivOp, _), [ Noeud
               ↪ ((Terminal Equivalent, _), []) ])
1891     :: _ ->
1892         Noeud
1893             ( OperateurLogique Equivalent,
1894               [
1895                 convert_to_abstract_aux
1896                   (Noeud ((NonTerminal
                           ↪ EquivOperand, s), l));

```

5- Annexe

```

1897         convert_to_abstract_aux
1898         (Noeud ((NonTerminal
                  ↪  EquivOp_EquivOperand_star,
                  ↪  s1), l1));
1899     ] )
1900 | Noeud
1901     ( (NonTerminal EquivOp, _),
1902       [ Noeud ((Terminal NotEquivalent, _),
                ↪  []) ] )
1903 :: _ ->
1904     Noeud
1905     ( OperateurLogique NonEquivalent,
1906       [
1907         convert_to_abstract_aux
1908         (Noeud ((NonTerminal
                  ↪  EquivOperand, s), l));
1909         convert_to_abstract_aux

```

5- Annexe

```

1910         (Noeud ((NonTerminal
                ↪ EquivOp_EquivOperand_star,
                ↪ s1), l1));
1911     ] )
1912     | _ -> failwith "Level5Expr"
1913 | Noeud
1914     ( (NonTerminal OrOp_OrOperand_star, _),
1915     [
1916         Noeud ((Terminal OrOp, _), []);
1917         Noeud ((NonTerminal OrOperand, s), l);
1918         Noeud
1919             ( (NonTerminal OrOp_OrOperand_star,
                ↪ _),
                [ Noeud ((Terminal E, _), []) ] ) );
1920     ] )
1921 | Noeud
1922     ( (NonTerminal EquivOperand, _),

```

5- Annexe

```

1924         [
1925             Noeud ((NonTerminal OrOperand, s), 1);
1926             Noeud
1927                 ( (NonTerminal OrOp_OrOperand_star,
1928                     ↪ _),
1929                     [ Noeud ((Terminal E, _), []) ] );
1930         ] ) ->
1931         convert_to_abstract_aux (Noeud
1932             ↪ ((NonTerminal OrOperand, s), 1))
1933         | Noeud
1934             ( (NonTerminal OrOp_OrOperand_star, _),
1935             [
1936                 Noeud ((Terminal OrOp, _), []);
1937                 Noeud ((NonTerminal OrOperand, s), 1);
1938                 Noeud ((NonTerminal
1939                     ↪ OrOp_OrOperand_star, s2), 12);
1940             ] )

```


5- Annexe

```

1938 | Noeud
1939     ( (NonTerminal EquivOperand, _),
1940     [
1941         Noeud ((NonTerminal OrOperand, s), 1);
1942         Noeud ((NonTerminal
1943             ↪ OrOp_OrOperand_star, s2), 12);
1944     ] ) ->
1945     Noeud
1946     ( OperateurLogique Ou,
1947     [
1948         convert_to_abstract_aux (Noeud
1949             ↪ ((NonTerminal OrOperand, s), 1));
1950         convert_to_abstract_aux
1951             (Noeud ((NonTerminal
1952                 ↪ OrOp_OrOperand_star, s2), 12));
1953     ] )
1954 | Noeud

```

5- Annexe

```

1952      ( (NonTerminal AndOp_AndOperand_star, _),
1953      [
1954          Noeud ((Terminal AndOp, _), []);
1955          Noeud ((NonTerminal AndOperand, s), 1);
1956          Noeud
1957              ( (NonTerminal AndOp_AndOperand_star,
1958                  ↪ _),
1959                  [ Noeud ((Terminal E, _), []) ] ) );
1960      ] )
1961  | Noeud
1962      ( (NonTerminal OrOperand, _),
1963      [
1964          Noeud ((NonTerminal AndOperand, s), 1);
1965          Noeud
1966              ( (NonTerminal AndOp_AndOperand_star,
1967                  ↪ _),
1968                  [ Noeud ((Terminal E, _), []) ] ) );

```

5- Annexe

```

1967         ] ) ->
1968         convert_to_abstract_aux (Noeud
1969         ↪ ((NonTerminal AndOperand, s), 1))
1969 | Noeud
1970   ( (NonTerminal AndOp_AndOperand_star, _),
1971     [
1972       Noeud ((Terminal AndOp, _), []);
1973       Noeud ((NonTerminal AndOperand, s1),
1974         ↪ 11);
1975       Noeud ((NonTerminal
1976         ↪ AndOp_AndOperand_star, s2), 12);
1977     ] )
1978 | Noeud
1979   ( (NonTerminal OrOperand, _),
1980     [
1981       Noeud ((NonTerminal AndOperand, s1),
1982         ↪ 11);

```

5- Annexe

```

1980         Noeud ((NonTerminal
           ↪ AndOp_AndOperand_star, s2), 12);
1981     ] ) ->
1982     Noeud
1983     ( OperateurLogique Et,
1984     [
1985         convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal AndOperand, s1),
           ↪ 11));
1986         convert_to_abstract_aux
1987         (Noeud ((NonTerminal
           ↪ AndOp_AndOperand_star, s2),
           ↪ 12));
1988     ] )
1989 | Noeud
1990     ( (NonTerminal AndOperand, _),
1991     [

```

5- Annexe

```

1992         Noeud ((NonTerminal NotOp_opt, _), [
           ↪   Noeud ((Terminal E, _), []) ]);
1993         Noeud ((NonTerminal Level4Expr, s), 1);
1994     ] ) ->
1995     convert_to_abstract_aux (Noeud
           ↪   ((NonTerminal Level4Expr, s), 1))
1996 | Noeud
1997   ( (NonTerminal AndOperand, _),
1998     [
1999         Noeud
2000           ((NonTerminal NotOp_opt, _), [ Noeud
           ↪   ((Terminal NotOp, _), []) ]);
2001         Noeud ((NonTerminal Level4Expr, s), 1);
2002     ] ) ->
2003     Noeud
2004       ( OperateurLogique Non,

```

5- Annexe

```

2005      [ convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal Level4Expr, s), 1)) ]
2006    )
2007  | Noeud
2008    ( (NonTerminal RelOp_Level3Expr_star, _),
2009    [
2010      Noeud ((NonTerminal RelOp, _), _);
2011      Noeud ((NonTerminal Level3Expr, s), 1);
2012      Noeud
2013        ( (NonTerminal RelOp_Level3Expr_star,
      ↪ _),
      [ Noeud ((Terminal E, _), []) ] ) );
2014    ] )
2015  | Noeud
2016    ( (NonTerminal Level4Expr, _),
2017    [
2018      Noeud ((NonTerminal Level3Expr, s), 1);
2019

```

5- Annexe

```

2020         Noeud
2021         ( (NonTerminal RelOp_Level3Expr_star,
2022           ↪ _),
2023           [ Noeud ((Terminal E, _), []) ] );
2024     ] ) ->
2025     convert_to_abstract_aux (Noeud
2026     ↪ ((NonTerminal Level3Expr, s), l))
2027 | Noeud
2028     ( (NonTerminal RelOp_Level3Expr_star, _),
2029     [
2030         Noeud ((NonTerminal RelOp, _), _);
2031         Noeud ((NonTerminal Level3Expr, s), l);
2032         Noeud ((NonTerminal
2033             ↪ RelOp_Level3Expr_star, s1), l1);
2034     ] )
2035 | Noeud
2036     ( (NonTerminal Level4Expr, _),

```

5- Annexe

```

2034         [
2035             Noeud ((NonTerminal Level3Expr, s), l);
2036             Noeud ((NonTerminal
                ↪ RelOp_Level3Expr_star, s1), l1);
2037         ] ) -> (
2038     match l1 with
2039     | [ Noeud ((Terminal E, _), []) ] ->
2040         convert_to_abstract_aux (Noeud
                ↪ ((NonTerminal Level3Expr, s), l))
2041     | Noeud ((NonTerminal RelOp, _), [ Noeud
                ↪ ((Terminal IsEqual, _), []) ])
2042         :: _ ->
2043         Noeud
2044             ( Comparateur Egal,
2045               [
2046                 convert_to_abstract_aux

```


5- Annexe

```

2047         (Noeud ((NonTerminal
                ↪ Level3Expr, s), 1));
2048     convert_to_abstract_aux
2049     (Noeud ((NonTerminal
                ↪ RelOp_Level3Expr_star, s1),
                ↪ 11));
2050     ] )
2051 | Noeud ((NonTerminal RelOp, _), [ Noeud
    ↪ ((Terminal NotEqual, _), []) ])
2052 :: _ ->
2053     Noeud
2054     ( Comparateur NonEgal,
2055     [
2056         convert_to_abstract_aux
2057         (Noeud ((NonTerminal
                ↪ Level3Expr, s), 1));
2058         convert_to_abstract_aux

```

5- Annexe

```

2059         (Noeud ((NonTerminal
                ↪ RelOp_Level3Expr_star, s1),
                ↪ 11));
2060     ] )
2061 | Noeud
2062     ((NonTerminal RelOp, _), [ Noeud
                ↪ ((Terminal StrictLess, _), []) ])
2063 :: _ ->
2064     Noeud
2065     ( Comparateur StrictPlusPetit,
2066       [
2067         convert_to_abstract_aux
2068         (Noeud ((NonTerminal
                ↪ Level3Expr, s), 1));
2069         convert_to_abstract_aux

```

5- Annexe

```

2070         (Noeud ((NonTerminal
                ↪ RelOp_Level3Expr_star, s1),
                ↪ l1));
2071     ] )
2072 | Noeud ((NonTerminal RelOp, _), [ Noeud
    ↪ ((Terminal LessEqual, _), [ ] ) ])
2073 :: _ ->
2074     Noeud
2075     ( Compareteur PlusPetit,
2076     [
2077         convert_to_abstract_aux
2078         (Noeud ((NonTerminal
                ↪ Level3Expr, s), l1));
2079         convert_to_abstract_aux
2080         (Noeud ((NonTerminal
                ↪ RelOp_Level3Expr_star, s1),
                ↪ l1));

```

5- Annexe

```

2081         ] )
2082     | Noeud
2083         ((NonTerminal RelOp, _), [ Noeud
           ↪ ((Terminal StrictGreater, _), [])
           ↪ ])
2084     :: _ ->
2085         Noeud
2086         ( Comparateur StrictPlusGrand,
2087           [
2088             convert_to_abstract_aux
2089               (Noeud ((NonTerminal
2090                 ↪ Level3Expr, s), 1));
2091             convert_to_abstract_aux
2092               (Noeud ((NonTerminal
2093                 ↪ RelOp_Level3Expr_star, s1),
2094                 ↪ 11));
2095           ] )

```

5- Annexe

```

2093 | Noeud
2094     ((NonTerminal RelOp, _), [ Noeud
      ↪ ((Terminal GreaterEqual, _), []) ])
2095 :: _ ->
2096     Noeud
2097     ( Comparateur PlusGrand,
2098       [
2099         convert_to_abstract_aux
2100         (Noeud ((NonTerminal
2101                 ↪ Level3Expr, s), l));
2102         convert_to_abstract_aux
2103         (Noeud ((NonTerminal
2104                 ↪ RelOp_Level3Expr_star, s1),
2105                 ↪ l1));
2106       ] )
2107 | _ -> failwith "Level4Expr"
2108 | Noeud

```

5- Annexe

```

2106      ((NonTerminal Level3Expr, _), [ Noeud
      ↪ ((NonTerminal Level2Expr, s), 1) ])
2107  ->
2108      convert_to_abstract_aux (Noeud
      ↪ ((NonTerminal Level2Expr, s), 1))
2109  | Noeud
2110      ( (NonTerminal
      ↪ AddOp_Sign_opt_AddOperand_star, _),
2111      [
2112          Noeud ((NonTerminal AddOp, _), _);
2113          Noeud ((NonTerminal
      ↪ Sign_opt_AddOperand, s), 1);
2114          Noeud
2115              ( (NonTerminal
      ↪ AddOp_Sign_opt_AddOperand_star,
      ↪ _),
2116              [ Noeud ((Terminal E, _), []) ] ) );

```

5- Annexe

```

2117         ] )
2118     | Noeud
2119         ( (NonTerminal Level2Expr, _),
2120         [
2121             Noeud ((NonTerminal
2122                 ↪ Sign_opt_AddOperand, s), l);
2123             Noeud
2124                 ( (NonTerminal
2125                     ↪ AddOp_Sign_opt_AddOperand_star,
2126                     ↪ _),
2127                 [ Noeud ((Terminal E, _), []) ] ) );
2128         ] ) ->
2129         convert_to_abstract_aux
2130             (Noeud ((NonTerminal Sign_opt_AddOperand,
2131                 ↪ s), l))
2132     | Noeud

```

5- Annexe

```
2129      ( (NonTerminal
        ↪ AddOp_Sign_opt_AddOperand_star, _),
2130      [
2131          Noeud ((NonTerminal AddOp, _), _);
2132          Noeud ((NonTerminal
        ↪ Sign_opt_AddOperand, s), 1);
2133          Noeud ((NonTerminal
        ↪ AddOp_Sign_opt_AddOperand_star,
        ↪ s2), 12);
2134      ] )
2135  | Noeud
2136      ( (NonTerminal Level2Expr, _),
2137      [
2138          Noeud ((NonTerminal
        ↪ Sign_opt_AddOperand, s), 1);
```


5- Annexe

```

2139         Noeud ((NonTerminal
           ↪ AddOp_Sign_opt_AddOperand_star,
           ↪ s2), 12);
2140     ] ) -> (
2141 match 12 with
2142 | Noeud
2143   ( (NonTerminal AddOp, _),
2144     [
2145       Noeud ((NonTerminal Sign, _), [
2146         ↪ Noeud ((Terminal Plus, _), [])
2147         ↪ ]));
2148     ] )
2149 :: _ ->
2150     Noeud
2151       ( Operateur Plus,
2152         [
2153           convert_to_abstract_aux

```

5- Annexe

```

2152         (Noeud ((NonTerminal
                ↪ Sign_opt_AddOperand, s),
                ↪ 1));
2153     convert_to_abstract_aux
2154     (Noeud ((NonTerminal
                ↪ AddOp_Sign_opt_AddOperand_star,
                ↪ s2), 12));
2155     ] )
2156 | Noeud
2157     ( (NonTerminal AddOp, _),
2158       [
2159         Noeud
2160         ((NonTerminal Sign, _), [ Noeud
                ↪ ((Terminal Minus, _), []) ]]);
2161     ] )
2162 :: _ ->
2163     Noeud

```

5- Annexe

```

2164         ( Operateur Moins,
2165         [
2166             convert_to_abstract_aux
2167             (Noeud ((NonTerminal
2168                 ↪ Sign_opt_AddOperand, s),
2169                 ↪ 1));
2170             convert_to_abstract_aux
2171             (Noeud ((NonTerminal
2172                 ↪ AddOp_Sign_opt_AddOperand_star,
2173                 ↪ s2), 12));
2174         ] )
2175     | _ -> failwith "Level2Expr"
2176 | Noeud
2177     ( (NonTerminal Sign_opt_AddOperand, _),
2178     [
2179         Noeud ((NonTerminal Sign_opt, _), [
2180             ↪ Noeud ((Terminal E, _), []) ]));

```

5- Annexe

```

2176         Noeud ((NonTerminal AddOperand, s), 1);
2177     ] ) ->
2178     convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal AddOperand, s), 1))
2179 | Noeud
        ( (NonTerminal Sign_opt_AddOperand, _),
2180     [
2181         Noeud
2182             ( (NonTerminal Sign_opt, _),
2183             [
2184                 Noeud
2185                     ((NonTerminal Sign, _), [ Noeud
2186                         ↪ ((Terminal Plus, _), [])
2187                         ↪ ]));
2188             ] );
2189         Noeud ((NonTerminal AddOperand, s), 1);
2190     ] ) ->

```

5- Annexe

```

2190         Noeud
2191         ( Operateur Plus,
2192         [ convert_to_abstract_aux (Noeud
2193         ↪ ((NonTerminal AddOperand, s), 1)) ]
2194         )
2195 | Noeud
2196 ( (NonTerminal Sign_opt_AddOperand, _),
2197 [
2198     Noeud
2199     ( (NonTerminal Sign_opt, _),
2200     [
2201         Noeud
2202         ((NonTerminal Sign, _), [ Noeud
2203         ↪ ((Terminal Minus, _), [])
2204         ↪ ]));
2205     ] );
2206 Noeud ((NonTerminal AddOperand, s), 1);

```

5- Annexe

```

2204         ] ) ->
2205         Noeud
2206         ( Operateur Moins,
2207         [ convert_to_abstract_aux (Noeud
2208             ↪ ((NonTerminal AddOperand, s), 1)) ]
2209         )
2210     | Noeud
2211     ( (NonTerminal MultOp_MultOperand_star, _),
2212     [
2213         Noeud ((NonTerminal MultOp, _), _);
2214         Noeud ((NonTerminal MultOperand, s),
2215             ↪ 1);
2216         Noeud
2217         ( (NonTerminal
2218             ↪ MultOp_MultOperand_star, _),
2219         [ Noeud ((Terminal E, _), []) ] );
2220     ] )

```

5- Annexe

```

2218 | Noeud
2219   ( (NonTerminal AddOperand, _),
2220     [
2221       Noeud ((NonTerminal MultOperand, s),
2222             ↪ 1);
2223       Noeud
2224         ( (NonTerminal
2225           ↪ MultOp_MultOperand_star, _),
2226           [ Noeud ((Terminal E, _), []) ] );
2227     ] ) ->
2228     convert_to_abstract_aux (Noeud
2229       ↪ ((NonTerminal MultOperand, s), 1))
2230 | Noeud
2231   ( (NonTerminal MultOp_MultOperand_star, _),
2232     [
2233       Noeud ((NonTerminal MultOp, _), _);

```

5- Annexe

```

2231         Noeud ((NonTerminal MultOperand, s),
                ↪ 1);
2232         Noeud ((NonTerminal
                ↪ MultOp_MultOperand_star, s2), 12);
2233     ] )
2234 | Noeud
2235     ( (NonTerminal AddOperand, _),
2236     [
2237         Noeud ((NonTerminal MultOperand, s),
                ↪ 1);
2238         Noeud ((NonTerminal
                ↪ MultOp_MultOperand_star, s2), 12);
2239     ] ) -> (
2240 match 12 with
2241 | Noeud ((NonTerminal MultOp, _), [ Noeud
                ↪ ((Terminal Asterisk, _), []) ])
2242     :: _ ->

```


5- Annexe

```

2243         Noeud
2244         ( Operateur Foix,
2245         [
2246             convert_to_abstract_aux
2247             (Noeud ((NonTerminal
2248                 ↪ MultOperand, s), 1));
2248             convert_to_abstract_aux
2249             (Noeud ((NonTerminal
2250                 ↪ MultOp_MultOperand_star,
2251                 ↪ s2), 12));
2252         ] )
2253 | Noeud ((NonTerminal MultOp, _), [ Noeud
2254     ↪ ((Terminal Divise, _), [ ] ) ] )
2255 :: _ ->
2256         Noeud
2257         ( Operateur Division,
2258         [

```

5- Annexe

```

2256         convert_to_abstract_aux
2257         (Noeud ((NonTerminal
                ↪ MultOperand, s), 1));
2258     convert_to_abstract_aux
2259     (Noeud ((NonTerminal
                ↪ MultOp_MultOperand_star,
                ↪ s2), 12));
2260     ] )
2261     | _ -> failwith "AddOperand"
2262 | Noeud
2263     ( (NonTerminal PowerOp_Level1Expr_star, _),
2264     [
2265         Noeud ((Terminal PowerOp, _), []);
2266         Noeud ((NonTerminal Level1Expr, s), 1);
2267         Noeud
2268             ( (NonTerminal
                ↪ PowerOp_Level1Expr_star, _),

```

5- Annexe

```

2269         [ Noeud ((Terminal E, _), []) ] );
2270     ] )
2271 | Noeud
2272     ( (NonTerminal MultOperand, _),
2273     [
2274         Noeud ((NonTerminal Level1Expr, s), 1);
2275         Noeud
2276             ( (NonTerminal
2277                 ↪ PowerOp_Level1Expr_star, _),
2278                 [ Noeud ((Terminal E, _), []) ] ) );
2279     ] ) ->
2280     convert_to_abstract_aux (Noeud
2281         ↪ ((NonTerminal Level1Expr, s), 1))
2282 | Noeud
2283     ( (NonTerminal PowerOp_Level1Expr_star, _),
2284     [
2285         Noeud ((Terminal PowerOp, _), []);

```

5- Annexe

```

2284         Noeud ((NonTerminal Level1Expr, s), 1);
2285         Noeud ((NonTerminal
           ↪ PowerOp_Level1Expr_star, s2), 12);
2286     ] )
2287 | Noeud
2288     ( (NonTerminal MultOperand, _),
2289     [
2290         Noeud ((NonTerminal Level1Expr, s), 1);
2291         Noeud ((NonTerminal
           ↪ PowerOp_Level1Expr_star, s2), 12);
2292     ] ) ->
2293     Noeud
2294         ( Operateur Puissance,
2295         [
2296             convert_to_abstract_aux (Noeud
           ↪ ((NonTerminal Level1Expr, s),
           ↪ 1));

```

5- Annexe

```

2297         convert_to_abstract_aux
2298         (Noeud ((NonTerminal
                ↪ PowerOp_Level1Expr_star, s2),
                ↪ 12));
2299     ] )
2300 | Noeud
2301     ((NonTerminal Level1Expr, _), [ Noeud
        ↪ ((NonTerminal Primary, s), 1) ])
2302 ->
2303     convert_to_abstract_aux (Noeud
        ↪ ((NonTerminal Primary, s), 1))
2304 | Noeud ((NonTerminal Primary, _), 1) -> (
2305     match 1 with
2306     | [ Noeud ((Terminal Icon, s), []) ] ->
        ↪ Noeud (Integer s, [])
2307     | [ Noeud ((Terminal Rcon, s), []) ] ->
        ↪ Noeud (Floating s, [])

```

5- Annexe

```

2308 | [ Noeud ((Terminal Dcon, s), []) ] ->
      ↪ Noeud (Double s, [])
2309 | [
2310   Noeud ((NonTerminal Name, _), [ Noeud
      ↪ ((Terminal Ident, s), []) ] );
2311   Noeud
2312     ( (NonTerminal FunctionReference_opt,
      ↪ _),
2313     [ Noeud ((Terminal E, _), []) ] );
2314 ] ->
      Noeud (Name s, [])
2315 | [
2316   Noeud ((NonTerminal Name, _), [ Noeud
      ↪ ((Terminal Ident, s), []) ] );
2317   Noeud
2318     ( (NonTerminal FunctionReference_opt,
2319       ↪ _),

```

5- Annexe

```

2320     [
2321         Noeud ((Terminal LParenthesis, _),
2322             ↪  []);
2323         Noeud
2324             ( ( NonTerminal
2325                 ↪  FunctionArg_Comma_FunctionArg
2326                 _ ),
2327             [ Noeud ((Terminal RParenthesis,
2328                 ↪  _), []) ] ) );
2329     ] ->
2330         Noeud (Syntax Call, [ Noeud (Name s,
2331             ↪  []) ])
2332     | [
2333         Noeud ((NonTerminal Name, _), [ Noeud
2334             ↪  ((Terminal Ident, s), []) ] );

```

5- Annexe

```

2332         Noeud ((NonTerminal FunctionReference_opt,
                ↪ s1), l1);
2333     ] ->
2334         Noeud
2335             ( Syntax Call,
2336               Noeud (Name s, [])
2337                   :: flatten
2338                       (convert_to_abstract_aux
2339                         (Noeud ((NonTerminal
2340                               ↪ FunctionReference_opt,
2341                               ↪ s1), l1)))
2342                         [] )
2343     | [
2344         Noeud ((NonTerminal Scon, _), [ Noeud
2345             ↪ ((Terminal SconSingle, s), []) ] );
2346     ] ->
2347         Noeud (Chaine s, [])

```


5- Annexe

```
2345 | [
2346   Noeud ((NonTerminal Scon, _), [ Noeud
      ↪ ((Terminal SconDouble, s), []) ]));
2347 ] ->
      Noeud (Chaine s, [])
2348 | [
2349   Noeud
2350     ((NonTerminal LogicalConstant, _), [
      ↪ Noeud ((Terminal True, _), []) ]));
2351 ] ->
      Noeud (Booleen true, [])
2352 | [
2353   Noeud
2354     ( (NonTerminal LogicalConstant, _),
      [ Noeud ((Terminal False, _), []) ] );
2355 ] ->
      Noeud (Booleen false, [])
```

5- Annexe

```

2360 | [
2361   Noeud ((Terminal LParenthesis, _), []);
2362   Noeud ((NonTerminal Expr, s), l);
2363   Noeud ((Terminal RParenthesis, _), []);
2364 ] ->
2365   Noeud
2366     ( ToFlatten,
2367       [
2368         Noeud (Parentheseouvrante, []);
2369         convert_to_abstract_aux (Noeud
2370           ↪ ((NonTerminal Expr, s), l));
2371         Noeud (Parenthesefermante, []);
2372       ] )
2373 | _ -> failwith "Primary")
2374 | Noeud
2375   ( (NonTerminal FunctionReference_opt, _),
    [

```

5- Annexe

```

2376         Noeud ((Terminal LParenthesis, _), []);
2377     Noeud
2378         ( ( NonTerminal
2379             ↪ FunctionArg_Comma_FunctionArg_star_opt
2380               s ),
2381           l );
2382     ] ) ->
2383     convert_to_abstract_aux
2384     (Noeud
2385         ( ( NonTerminal
2386             ↪ FunctionArg_Comma_FunctionArg_star_opt
2387               s ),
2388           l ))
2389 | Noeud
2390     ( (NonTerminal
2391         ↪ FunctionArg_Comma_FunctionArg_star_opt_RPar
2392         ↪ _),

```

5- Annexe

```

2389         [
2390             Noeud ((NonTerminal FunctionArg, s),
2391                 ↪ 1);
2392             Noeud
2393                 ( (NonTerminal
2394                     ↪ Comma_FunctionArg_star, _),
2395                     [ Noeud ((Terminal E, _), []) ] );
2396             Noeud ((Terminal RParenthesis, _), []);
2397         ] ) ->
2398         convert_to_abstract_aux (Noeud
2399             ↪ ((NonTerminal FunctionArg, s), 1))
2400 | Noeud
2401     ( (NonTerminal
2402         ↪ FunctionArg_Comma_FunctionArg_star_opt_RPar
2403         ↪ _),
2404     [

```

5- Annexe

```

2400         Noeud ((NonTerminal FunctionArg, s),
                ↪ 1);
2401         Noeud ((NonTerminal
                ↪ Comma_FunctionArg_star, s1), l1);
2402         Noeud ((Terminal RParenthesis, _), []);
2403     ] ) ->
2404     Noeud
2405     ( ToFlatten,
2406       [
2407         convert_to_abstract_aux (Noeud
                ↪ ((NonTerminal FunctionArg, s),
                ↪ 1));
2408         convert_to_abstract_aux
2409         (Noeud ((NonTerminal
                ↪ Comma_FunctionArg_star, s1),
                ↪ l1));
2410     ] )

```

5- Annexe

```

2411 | Noeud
2412   ( (NonTerminal Comma_FunctionArg_star, _),
2413     [
2414       Noeud ((Terminal Comma, _), []);
2415       Noeud ((NonTerminal FunctionArg, s),
2416               ↪ 1);
2417       Noeud
2418         ( (NonTerminal
2419             ↪ Comma_FunctionArg_star, _),
2420           [ Noeud ((Terminal E, _), []) ] ) );
2421   ] ) ->
2422   convert_to_abstract_aux (Noeud
2423     ↪ ((NonTerminal FunctionArg, s), 1))
2424 | Noeud
2425   ( (NonTerminal Comma_FunctionArg_star, _),
2426     [
2427       Noeud ((Terminal Comma, _), []);

```

5- Annexe

```

2425         Noeud ((NonTerminal FunctionArg, s),
                ↪ 1);
2426         Noeud ((NonTerminal
                ↪ Comma_FunctionArg_star, s1), l1);
2427     ] ) ->
2428     Noeud
2429     ( ToFlatten,
2430     [
2431         convert_to_abstract_aux (Noeud
                ↪ ((NonTerminal FunctionArg, s),
                ↪ 1));
2432         convert_to_abstract_aux
2433         (Noeud ((NonTerminal
                ↪ Comma_FunctionArg_star, s1),
                ↪ l1));
2434     ] )

```

5- Annexe

```

2435 | Noeud ((NonTerminal FunctionArg, _), [ Noeud
    ↪ ((NonTerminal Expr, s), 1) ])
2436     ->
2437     convert_to_abstract_aux (Noeud
    ↪ ((NonTerminal Expr, s), 1))
2438 | Noeud ((NonTerminal Name, _), [ Noeud
    ↪ ((Terminal Ident, s), []) ])
2439 | Noeud ((NonTerminal ArrayName, _), [ Noeud
    ↪ ((Terminal Ident, s), []) ])
2440 | Noeud ((NonTerminal ComponentName, _), [
    ↪ Noeud ((Terminal Ident, s), []) ])
2441 | Noeud ((NonTerminal EndName, _), [ Noeud
    ↪ ((Terminal Ident, s), []) ])
2442 | Noeud ((NonTerminal DummyArgName, _), [ Noeud
    ↪ ((Terminal Ident, s), []) ])
2443 | Noeud ((NonTerminal FunctionName, _), [ Noeud
    ↪ ((Terminal Ident, s), []) ])

```


5- Annexe

```

2444 | Noeud
2445 |     ((NonTerminal ImpliedDoVariable, _), [
2446 |         ↪ Noeud ((Terminal Ident, s), []) ])
2447 | Noeud ((NonTerminal ProgramName, _), [ Noeud
2448 |     ↪ ((Terminal Ident, s), []) ])
2449 | Noeud
2450 |     ((NonTerminal SubroutineName, _), [ Noeud
2451 |         ↪ ((Terminal Ident, s), []) ])
2452 | Noeud ((NonTerminal SubroutineNameUse, _), [
2453 |     ↪ Noeud ((Terminal Ident, s), []) ])
2454 | Noeud ((NonTerminal VariableName, _), [ Noeud
2455 |     ↪ ((Terminal Ident, s), []) ])
2456 | Noeud ((NonTerminal ObjectName, _), [ Noeud
2457 |     ↪ ((Terminal Ident, s), []) ])
2458 |     ->
2459 |     Noeud (Name s, [])

```

5- Annexe

```

2455 | Noeud ((Terminal EOS, s), []) ->
2456     Noeud (ToFlatten, convert_comments (Noeud
      ↪ (Commentaire s, [])))
2457 | _ ->
2458     (let string_of_symbol (s : symbol) : string
      ↪ =
2459         match s with
2460         | Terminal x -> string_of_terminal x
2461         | NonTerminal x ->
      ↪     string_of_non_terminal x
2462     in
2463     match t with
2464     | Noeud ((s, _), Noeud ((x, _), _) :: _)
      ↪     ->
2465         prerr_string (string_of_symbol s);
2466         prerr_char ' ';
2467         prerr_string (string_of_symbol x);

```

5- Annexe

```
2468         prerr_newline ()
2469     | Noeud ((s, _), _) ->
2470         prerr_string (string_of_symbol s);
2471         prerr_newline ());
2472     failwith "is not implemented yet"
2473 in
2474 link_return_function (convert_to_abstract_aux t)
    ↪ None
```

5- Annexe

```
1  open Automates
2
3  let syntax_automate_det = {
```

5- Annexe

```

4  nodes = [244; 243; 242; 241; 240; 239; 238; 237;
    ↪ 236; 235; 234; 233; 232; 231; 230; 229; 228;
    ↪ 227; 226; 225; 224; 223; 222; 221; 220; 219;
    ↪ 218; 217; 216; 215; 214; 213; 212; 211; 210;
    ↪ 209; 208; 207; 206; 205; 204; 203; 202; 201;
    ↪ 200; 199; 198; 197; 196; 195; 194; 193; 192;
    ↪ 191; 190; 189; 188; 187; 186; 185; 184; 183;
    ↪ 182; 181; 180; 179; 178; 177; 176; 175; 174;
    ↪ 173; 172; 171; 170; 169; 168; 167; 166; 165;
    ↪ 164; 163; 162; 161; 160; 159; 158; 157; 156;
    ↪ 155; 154; 153; 152; 151; 150; 149; 148; 147;
    ↪ 146; 145; 144; 143; 142; 141; 140; 139; 138;
    ↪ 137; 136; 135; 134; 133; 132; 131; 130; 129;
    ↪ 128; 127; 126; 125; 124; 123; 122; 121; 120;
    ↪ 119; 118; 117; 116; 115; 114; 113; 112; 111;
    ↪ 110; 109; 108; 107; 106; 105; 104; 103; 102;
    ↪ 101; 100; 99; 98; 97; 96; 95; 94; 93; 92; 91;
    ↪ 90; 89; 88; 87; 86; 85; 84; 83; 82; 81; 80;
    ↪ 79; 78; 77; 76; 75; 74; 73; 72; 71; 70; 69; 68; 67; 66; 65; 64; 63; 62; 61; 60; 59; 58; 57; 56; 55; 54; 53; 52; 51; 50; 49; 48; 47; 46; 45; 44; 43; 42; 41; 40; 39; 38; 37; 36; 35; 34; 33; 32; 31; 30; 29; 28; 27; 26; 25; 24; 23; 22; 21; 20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 10; 9; 8; 7; 6; 5; 4; 3; 2; 1; 0;

```

5- Annexe

```
5 | debut = 0;
```

5- Annexe

```

6  fin = [|None; Some EOS; Some Space; None; None;
    ↪ None; Some LParenthesis; Some RParenthesis;
    ↪ Some Asterisk; Some Plus; Some Comma; Some
    ↪ Minus; None; Some Divide; Some Icon; Some
    ↪ Colon; Some StrictLess; Some Equal; Some
    ↪ StrictGreater; Some Ident; Some Ident; Some
    ↪ Ident; Some Ident; Some Ident; Some Ident;
    ↪ Some Ident; Some Ident; Some Ident; Some
    ↪ Ident; Some Ident; Some Ident; Some Ident;
    ↪ Some Ident; Some Ident; Some Ident; Some
    ↪ While; Some Ident; Some Ident; Some Then;
    ↪ Some Ident; Some Ident; Some Ident; Some
    ↪ Ident; Some Ident; Some Ident; Some Ident;
    ↪ Some Ident; Some Subroutine; Some Ident; Some
    ↪ Ident; Some Ident; Some Ident; Some Ident;
    ↪ Some Ident; Some Ident; Some Return; Some
    ↪ Ident; Some Ident; Some Result; Some Ident;
    ↪ Some Ident; Some Ident; Some Ident; Some

```

5- Annexe

```
7  transitions = [|  
8    [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; 1;  
    ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
    ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; 2;  
    ↪ 3; 4; -1; -1; -1; -1; 5; 6; 7; 8; 9; 10;  
    ↪ 11; 12; 13; 14; 14; 14; 14; 14; 14; 14; 14;  
    ↪ 14; 14; 15; -1; 16; 17; 18; -1; -1; 19; 19;  
    ↪ 20; 21; 22; 23; 19; 19; 24; 19; 19; 25; 19;  
    ↪ 19; 26; 27; 19; 28; 29; 30; 19; 19; 31; 19;  
    ↪ 19; 19; -1; -1; -1; -1; -1; -1; 19; 19; 20;  
    ↪ 21; 22; 23; 19; 19; 24; 19; 19; 25; 19; 19;  
    ↪ 26; 27; 19; 28; 29; 30; 19; 19; 31; 19; 19;  
    ↪ 19; -1; -1; -1; -1; -1|];
```


5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

```

11      [|3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 244; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3; 3;
      ↪ 3; 3|];

```

5- Annexe

```

12      [|4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 243; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4; 4;
      ↪ 4|];

```

5- Annexe

```

13      [|5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 242; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
      ↪ 5|];

```

5- Annexe

[illegible]

5- Annexe

```

15      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

16

[illegible]

5- Annexe

[illegible]

5- Annexe

```

18      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

[illegible]

5- Annexe

```

20      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; 202; 202; 202; 202; 202;
      ↪ 202; 202; 202; 202; 202; -1; -1; -1; -1;
      ↪ -1; -1; -1; 203; -1; -1; -1; 204; 205; 206;
      ↪ -1; -1; -1; -1; 207; -1; 208; 209; -1; -1;
      ↪ -1; -1; 210; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; 203; -1; -1; -1; 204;
      ↪ 205; 206; -1; -1; -1; -1; 207; -1; 208;
      ↪ 209; -1; -1; -1; -1; 210; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

21	<pre>[-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; 201; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↳ -1; -1; -1; -1; -1; -1; -1; -1 -1];</pre>
----	---

5- Annexe

```

22      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 186; -1; 187; 187; 187; 187; 187;
      ↪   187; 187; 187; 187; 187; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 188; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

23	<pre>[-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1 -];</pre>
----	---

5- Annexe

```

24      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; 185; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1|-1|];

```


5- Annexe

[illegible]

5- Annexe

26	<pre>[-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; 183; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; ↪ -1; -1; -1; -1; -1; -1; -1; -1 -];</pre>
----	--

5- Annexe

```

27      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

28      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; -1; -1; -1; -1; -1; -1; -1;
      ↪ 160; 19; 19; 19; 19; 19; 19; 19; 161; 19; 19;
      ↪ 19; 19; 19; 19; 162; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; -1; -1; -1; -1; 19;
      ↪ -1; 160; 19; 19; 19; 19; 19; 19; 161; 19;
      ↪ 19; 19; 19; 19; 19; 162; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; 19; -1; -1; -1; -1;
      ↪ -1|];

```

5- Annexe

```

29      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 145; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 145; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

30      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   110; 19; 111; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   110; 19; 111; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

31      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 103; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 103; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

32      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 91; 19; 19; 19; 19; 19; 19;
      ↪   19; 92; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 91; 19; 19; 19; 19; 19; 19; 19;
      ↪   92; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1];

```


5- Annexe

```

33      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 85; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 85; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

34      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 83; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 83; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

35      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 66;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 67; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 66; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 67; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

36      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 48; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 48; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

37      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 39; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 39; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

38      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 36; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 36; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

39      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 32; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 32; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

40      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 33; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 33; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

41      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 34;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 34; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

42      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 35; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪    19; 19; 35; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

43      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

44      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 37; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 37; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

45      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 38; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   38; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

46      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

47      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   40; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 40;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

48      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 41; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 41; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

49      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 42; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 42; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

50      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 43; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 43; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

51      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 44; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 44; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

52      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 45; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 45; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

53      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 46; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   46; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

54      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 47; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 47; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

55      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

56      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 49;
      ↪ 19; 50; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; 51; 52; 19; 19; 19;
      ↪ 19; 19; 19; -1; -1; -1; -1; 19; -1; 49; 19;
      ↪ 50; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 51; 52; 19; 19; 19; 19;
      ↪ 19; 19; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

57      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 65;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 65; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

58      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 59; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 59; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

59      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 56; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 56; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

60      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 53; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 53; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

61      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 54; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 54; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

62      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 55; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   55; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

63      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

64      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 57;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 57; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

65      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 58; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 58; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

66      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

67      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 60; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 60; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

68      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 61; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 61; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

69      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 62; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 62; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

70      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 63; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 63; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

71      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 64; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 64; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

72      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

73      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

74      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 76; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 76; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

75      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 68; 19; 19; 19;
      ↪   19; 19; 69; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 68; 19; 19; 19; 19;
      ↪   19; 69; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

76      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 74; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   74; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

77      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 70; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 70; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

78      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 71; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 71; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

79      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 72;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 72; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

80      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   73; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 73;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```


5- Annexe

```

81      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

82      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 75; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 75; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

83      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

84      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 77;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 77; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

85      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   78; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 78;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

86      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 79; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 79; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

87      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 80; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 80; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

88      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 81; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 81; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```


5- Annexe

```

89      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 82; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 82; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

90      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

91      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 84; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 84; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

92      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

93      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 19; 19; 86; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪    19; 19; 19; 19; 86; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

94      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 87; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 87; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

95      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 88; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   88; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

96      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 89;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 89; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```


5- Annexe

```

97      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 90;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 90; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

98      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

99      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

100      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 93; 19; 19; 19; 19; 94; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 93; 19; 19; 19; 19; 94; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

101      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 101; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 101; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

102      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 95; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 95; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

103      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 96; 19; 19; 19; 19; 19;
      ↪   19; 97; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 96; 19; 19; 19; 19; 19; 19;
      ↪   97; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-]);

```

5- Annexe

```

104      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 99; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 99; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-1|];

```


5- Annexe

```

105      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 98; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 98; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

106      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

107      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 100; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 100; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

108      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

109      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 102; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 102; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

110      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

111      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 104; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 104; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

112      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 105; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 105; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

113      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 106; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 106; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

114      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 107; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 107; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

115      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 108; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 108; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

116      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 109; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 109; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

117      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-)];

```

5- Annexe

```

118      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 143; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 143; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

119      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 112; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 112; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

120      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   113; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1; -1;
      ↪   -1; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

121      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 114; -1; 115; -1; -1; 116; -1; -1;
      ↪   -1; -1; -1; -1; 117; -1; -1; 118; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 114; -1; 115; -1; -1; 116; -1;
      ↪   -1; -1; -1; -1; -1; 117; -1; -1; 118; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

122      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; 142; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; 142; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|-1];

```

5- Annexe

```

123      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; 135; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; 135; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

124      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 134; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 134; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

125      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 128; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 128; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

126      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 119; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 119; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

127      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    120; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    120; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1 |];

```

5- Annexe

```

128      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 121; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 121; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```


5- Annexe

```

129      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 122; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 122; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

130      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; 123; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; 123; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

131      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 124; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 124; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

132      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 125; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 125; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

133      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 126; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 126; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

134      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 127; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 127; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-1];

```

5- Annexe

[illegible]

5- Annexe

```
136      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; 129; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; 129; -1; -1; -1; -1; -1; -1; -1;  
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1|-];
```


5- Annexe

```

137      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 130; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 130; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

138      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 131; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 131; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

139      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   132; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   132; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

140      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   133; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   133; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

141      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

[illegible]

5- Annexe

```
143      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; 136; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; 136; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1|];
```

5- Annexe

```

144      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; 137; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; 137; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```


5- Annexe

```

145      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 138; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 138; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

146      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 139; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 139; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```
147      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; 140; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; 140; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1|];
```

5- Annexe

```

148      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 141; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 141; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

149      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```
150      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];
```

5- Annexe

```

151      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 144; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 144; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

152      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

153      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 146; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 146; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

154      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   147; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   147; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

155      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   148; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   148; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

156      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 149; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 149; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

157      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   150; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1; -1;
      ↪   -1; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

158      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 151; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 151; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-1];

```

5- Annexe

```
159      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; 152; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; 152; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1|];
```

5- Annexe

```

160      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 153; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 153; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

161      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 154; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 154; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-1|];

```

5- Annexe

```

162      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 155; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 155; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

163      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 156; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 156; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

164      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 157; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; 157; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

165      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 158; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 158; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

166      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 159; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 159; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

[illegible]

5- Annexe

```

168      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   181; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   181; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

169      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1;
      ↪   174; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   174; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1]]];

```

5- Annexe

```

170      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   163; 164; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   163; 164; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

171      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 170; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 170; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

172      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 165; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 165; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

173      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1;
      ↪   166; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   166; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1]]];

```

5- Annexe

```

174      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 167; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 167; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

175      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 168; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 168; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

176      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 169; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 169; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

177      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

178      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   171; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   171; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

179      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 172; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 19; 19; 172; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

180      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   173; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   173; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

181      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪ -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪ 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪ 19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

182      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 175; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 175; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

183      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1;
      ↪   176; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   176; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1]]];

```

5- Annexe

```

184      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 177; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 177; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```


5- Annexe

```

185      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 178; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 178; 19; 19;
      ↪   19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

186      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 179; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 19; 19; 179; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

187      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 180; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; 19; -1; 19;
      ↪    19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; 19; 180; 19; 19; 19; 19;
      ↪    19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

188      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

189      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   182; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; 19; -1;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   182; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; -1; -1; -1; -1; -1|];

```

5- Annexe

```

190      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; -1; -1; -1; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; -1; -1; -1; -1; 19; -1; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; 19; 19; 19; 19; 19; 19; 19; 19; 19;
      ↪   19; 19; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

191      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

[illegible]

5- Annexe

193

	[-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;	-1;
	↪	-1;	-1;	-1;	-1;	-1;	-1 -]				

5- Annexe

```

194      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 192; 192; 192; 192; 192;
      ↪   192; 192; 192; 192; 192; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 193; 194; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

195      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 186; -1; 187; 187; 187; 187; 187;
      ↪   187; 187; 187; 187; 187; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; 188; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

196      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   189; -1; 189; -1; -1; 190; 190; 190; 190;
      ↪   190; 190; 190; 190; 190; 190; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1|];

```

5- Annexe

```

197      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; 191; 191; 191; 191; 191;
      ↪    191; 191; 191; 191; 191; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1|-1];

```

5- Annexe

```

198      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 191; 191; 191; 191; 191;
      ↪   191; 191; 191; 191; 191; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

199      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 191; 191; 191; 191; 191;
      ↪   191; 191; 191; 191; 191; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

200      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 192; 192; 192; 192; 192;
      ↪   192; 192; 192; 192; 192; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 193; 194; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```


5- Annexe

```

201      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   198; -1; 198; -1; -1; 199; 199; 199; 199;
      ↪   199; 199; 199; 199; 199; 199; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

202      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   195; -1; 195; -1; -1; 196; 196; 196; 196;
      ↪   196; 196; 196; 196; 196; 196; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

203      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 197; 197; 197; 197; 197;
      ↪   197; 197; 197; 197; 197; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

204      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 197; 197; 197; 197; 197;
      ↪   197; 197; 197; 197; 197; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

205      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 197; 197; 197; 197; 197;
      ↪   197; 197; 197; 197; 197; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

206      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 200; 200; 200; 200; 200;
      ↪   200; 200; 200; 200; 200; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

207      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 200; 200; 200; 200; 200;
      ↪   200; 200; 200; 200; 200; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```

5- Annexe

```

208      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 200; 200; 200; 200; 200;
      ↪   200; 200; 200; 200; 200; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|-1];

```


5- Annexe

[illegible]

5- Annexe

```

210      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 202; 202; 202; 202; 202;
      ↪   202; 202; 202; 202; 202; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 193; 194; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

211      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 238; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; 238; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

212      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 235; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 235; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

213      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    230; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    230; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

214      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 227; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 228; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 227; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; 228;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

215      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 224; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 225; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 224; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; 225;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```

5- Annexe

```

216      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 217; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 218; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 217; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; 218; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1|];

```


5- Annexe

```

217      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 215; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 215; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

218      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 211; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; 211; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

219      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 212; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; 212; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

220      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 213; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; 213; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1 |];

```

5- Annexe

[illegible]

5- Annexe

```

222      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

[illegible]

5- Annexe

```

224      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```


5- Annexe

```

225      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 201; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 221; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; 221; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

226      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 219; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; 219; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

[illegible]

5- Annexe

```

228      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

229      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; 222;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; 222;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

```

234      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

[illegible]

27 / 27

5- Annexe

```

237      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];

```

5- Annexe

```

238      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   231; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   231; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

239      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; 232; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; 232; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

```

240      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; 233; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; 233; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1 |];

```


5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

```

243      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; 184; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   236; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   236; -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

[illegible]

5- Annexe

```
245      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1|-];
```

5- Annexe

```

246      [| -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; 239; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; 239; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪    -1; -1; -1; -1; -1; -1; -1; -1; -1|];

```

5- Annexe

[illegible]

5- Annexe

```

248      [|-1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1; -1; -1; -1; -1;
      ↪   -1; -1; -1; -1; -1; -1; -1|-];

```


5- Annexe

[illegible]

27 / 27

5- Annexe

[illegible]

5- Annexe

[illegible]

5- Annexe

```

1  open Environnement
2  open Abstract_tokens
3  open Bibliothèques
4  open Convert_to_abstract
5  open LL1
6  open Traduction
7
8  (** Renvoie les imports nécessaires contenus dans
   ↪ [l] *)
9  let rec generate_library_imports (l : libs) :
   ↪ string =
10     match l with
11     | [] -> ""
12     | name :: q -> "#include <" ^ name ^ ".h>\n" ^
   ↪ generate_library_imports q
13
14 (** Renvoie le format de string dans les printf
   ↪ pour afficher la variable ou la

```

5- Annexe

```

15   fonction [n] à l'aide de [env] *)
16   let generate_format_string_of_name (n : string)
    ↪   (env : environnement) : string =
17     match Hashtbl.find_opt env n with
18     | Some (Syntax Real) -> "%f "
19     | Some (Syntax Integer) -> "%d "
20     | Some (Syntax Logical) -> "%d "
21     | _ -> failwith "La variable n'est pas dans
    ↪   l'environnement"
22
23   (** Renvoie le format de string dans les printf
    ↪   pour afficher les éléments de
24     [l] en s'aidant des types dans [env] *)
25   let rec generate_format_string (l : ast list) (env
    ↪   : environnement) : string =
26     match l with
27     | [] -> ""

```

5- Annexe

```

28 | Noeud (Chaine _, []) :: q -> "%s " ^
    ↳ generate_format_string q env
29 | Noeud (Integer _, []) :: q -> "%d " ^
    ↳ generate_format_string q env
30 | Noeud (Floating _, []) :: q -> "%f " ^
    ↳ generate_format_string q env
31 | Noeud (Booleen _, []) :: q -> "%d " ^
    ↳ generate_format_string q env
32 | Noeud (Name n, []) :: q ->
33     generate_format_string_of_name n env ^
        ↳ generate_format_string q env
34 | Noeud (Syntax Call, Noeud (Name n, [])) :: _ ::
    ↳ _ ->
35     generate_format_string_of_name n env
36 | Noeud (s, _) :: _ ->
37     print_token s;
38     failwith " type non def"

```

5- Annexe

```
39
40  (** renvoie la chaîne associée au type de [var]
    ↪   dans [env] *)
41  let str_of_env_type (env : environnement) (var :
    ↪   string) : string =
42    match Hashtbl.find_opt env var with
43    | Some (Syntax Integer) -> "int"
44    | Some (Syntax Real) -> "float"
45    | Some t ->
46        print_token t;
47        failwith "type non supporté"
48    | None -> failwith "var not found in
    ↪   environnement"
49
50  (** Applique la fonction [f] au dernier élément de
    ↪   la liste [l] *)
```


5- Annexe

```

51 let rec map_to_last (f : 'a -> 'a) (l : 'a list) :
    ↪ 'a list =
52     match l with
53     | [] -> failwith "impossible to map to last"
54     | e :: [] -> if e = L [] then failwith "e= []"
    ↪     else [ f e ]
55     | e :: q -> e :: map_to_last f q
56
57     (** ajoute un point-virgule à la fin de [s] si elle
    ↪ n'en a pas déjà *)
58 let rec add_semi_colon (s : string_or_string_list)
    ↪ : string_or_string_list =
59     match s with
60     | S s -> if String.ends_with s ~suffix:";" then S
    ↪     s else S (s ^ ";")
61     | L l -> L (map_to_last add_semi_colon l)
62

```

5- Annexe

```
63  (** Renvoie le type de la fonction d'ast [a] à
    ↪  l'aide de l'environnement [env] *)
64  let get_function_return_type (a : ast) (env :
    ↪  environnement) :
65      string_or_string_list =
66      match a with
67      | Noeud (Syntax Function, l) -> (
68          let ret = last_of_list l in
69          match ret with
70          | Noeud (Syntax Return, [ Noeud (Name
    ↪  var_name, []) ]) ->
71              S (str_of_env_type env var_name)
72          | _ -> failwith "type de retour non pris en
    ↪  charge")
73      | _ -> failwith "Cette fonction prend en
    ↪  paramètre un Noeud(Function)"
```

74

5- Annexe

```

75  (** parcourt la liste des fils pour trouver les
    ↪  définitions des paramètres *)
76  let rec get_function_param_list (l : ast list) (env
    ↪  : environnement) :
77      string_or_string_list * ast list =
78      match l with
79
80      | Noeud (Name nom, []) :: Noeud(Name n, []) :: q
    ↪  ->
81          let sosl, q2 = get_function_param_list
    ↪  (Noeud(Name n, []) :: q) env in
82          (L [ S (str_of_env_type env nom); S " "; S
    ↪  nom; S ", "; sosl ], q)
83
84      | Noeud (Name nom, []) :: q -> L [ S
    ↪  (str_of_env_type env nom); S " "; S nom ], q
85      | l1 -> (S "", l)

```

5- Annexe

```

86
87  (** convertit l'arbre de syntaxe abstrait [ast] à
    ↪  l'aide de l'environnement
88      [env] et l'indente de [n_tab] en un
    ↪  string_or_string_list*)
89  let rec convert_ast_to_C_sosl (ast : ast list) (env
    ↪  : environnement)
90      (nb_tab : int) : string_or_string_list =
91      match ast with
92      | [] -> S ""
93      | [ Noeud (ProgramRoot, l) ] ->
    ↪  convert_ast_to_C_sosl l env nb_tab
94      | Noeud (Syntax Program, Noeud (Name nom, []) ::
    ↪  l1) :: q ->
95          L
96          [
97              Traduction.tabs_to_string nb_tab;

```

5- Annexe

```

98         S "// ";
99         S nom;
100        S "\n";
101        Traduction.tabs_to_string nb_tab;
102        S "void main(void){";
103        convert_ast_to_C_sosl l1 env (nb_tab +
        ↪ 1);
104        S "}";
105    ]
106 | Noeud (Commentaire c, []) :: q ->
107     L
108     [
109         Traduction.tabs_to_string nb_tab;
110         S "//";
111         S c;
112         convert_ast_to_C_sosl q env nb_tab;
113     ]

```

5- Annexe

```

114 | Noeud (Syntax Print, l2) :: q ->
115     L
116     [
117         Traduction.tabs_to_string nb_tab;
118         S "printf(\"";
119         S (generate_format_string l2 env);
120         S "\"";
121         L
122             (List.rev
123                 (List.fold_left
124                     (fun acc x ->
125                         convert_ast_to_C_sosl [ x ] env
126                             ↪ nb_tab :: S " , " :: acc)
127                     [] l2));
128         S ");";
129         convert_ast_to_C_sosl q env nb_tab;
130     ]

```

5- Annexe

```
130      (* définit le type des variables *)
131      | Noeud (Syntax Real, l) :: q ->
132          let s = convert_ast_to_C_sosl l env 0 in
133          let s = add_semi_colon s in
134          L
135          [
136              Traduction.tabs_to_string nb_tab;
137              S "float ";
138              s;
139              convert_ast_to_C_sosl q env nb_tab;
140          ]
141      | Noeud (Syntax Integer, l) :: q ->
142          let s = convert_ast_to_C_sosl l env 0 in
143          let s = add_semi_colon s in
144          L
145          [
146              Traduction.tabs_to_string nb_tab;
```

5- Annexe

```
147         S "int ";
148         S;
149         convert_ast_to_C_sosl q env nb_tab;
150     ]
151 | Noeud (Syntax Logical, l) :: q ->
152     let s = convert_ast_to_C_sosl l env 0 in
153     let s = add_semi_colon s in
154     L
155     [
156         Traduction.tabs_to_string nb_tab;
157         S "bool ";
158         S;
159         convert_ast_to_C_sosl q env nb_tab;
160     ]
161 | Noeud (Syntax Double_precision, l) :: q ->
162     let s = convert_ast_to_C_sosl l env 0 in
163     let s = add_semi_colon s in
```


5- Annexe

```

164         L
165         [
166             Traduction.tabs_to_string nb_tab;
167             S "long ";
168             s;
169             convert_ast_to_C_sosl q env nb_tab;
170         ]
171 | Noeud (Syntax Character, Noeud (Syntax
  ↪ Constant, []) :: l) :: q ->
172     let s = convert_ast_to_C_sosl l env 0 in
173     let s = add_semi_colon s in
174     L
175     [
176         Traduction.tabs_to_string nb_tab;
177         S "const char ";
178         s;
179         convert_ast_to_C_sosl q env nb_tab;

```

5- Annexe

```

180     ]
181 | Noeud (Syntax Character, l) :: q ->
182     let s = convert_ast_to_C_sosl l env 0 in
183     let s = add_semi_colon s in
184     L
185     [
186         Traduction.tabs_to_string nb_tab;
187         S "char ";
188         s;
189         convert_ast_to_C_sosl q env nb_tab;
190     ]
191 | Noeud (Operateur Assignment, Noeud (Name s,
192     ↪ []) :: l) :: q ->
193     L
194     [
195         Traduction.tabs_to_string nb_tab;
196         S s;

```

5- Annexe

```

196         S " = ";
197         convert_ast_to_C_sosl l env 0;
198         S ";";
199         convert_ast_to_C_sosl q env nb_tab;
200     ]
201 | Noeud (Syntax Size, l) :: q ->
202     L
203     [
204         Traduction.tabs_to_string nb_tab;
205         S "[";
206         convert_ast_to_C_sosl l env 0;
207         S "]" ";
208         convert_ast_to_C_sosl q env nb_tab;
209     ]
210 | Noeud (Operateur Assignment, Noeud (Syntax
  ↪ Size, l1) :: l) :: q ->
211     L

```

5- Annexe

```

212     [
213         Traduction.tabs_to_string nb_tab;
214         convert_ast_to_C_sosl [ Noeud (Syntax
215             ↪ Size, l1) ] env 0;
216         convert_ast_to_C_sosl
217             (Noeud (Operateur Assignment, l) :: q)
218             env nb_tab;
219     ]
220 | Noeud (Syntax Any, []) :: q ->
221     ↪ convert_ast_to_C_sosl q env 0
222 | Noeud (Name s, []) :: q -> L [ S s;
223     ↪ convert_ast_to_C_sosl q env nb_tab ]
224 | Noeud (Operateur Plus, elem :: l) :: q ->
225     L
226     [
227         convert_ast_to_C_sosl [ elem ] env
228         ↪ nb_tab;

```

5- Annexe

```

225         S " + ";
226         convert_ast_to_C_sosl l env nb_tab;
227         convert_ast_to_C_sosl q env nb_tab;
228     ]
229
230     (* opérateur unaire *)
231 | Noeud (Opérateur Moins, elem :: []) :: q ->
232     L
233     [
234         S " -";
235         convert_ast_to_C_sosl [ elem ] env
236         ↪ nb_tab;
237         convert_ast_to_C_sosl q env nb_tab;
238     ]
239
240 | Noeud (Opérateur Moins, elem :: l) :: q ->
241     L

```

5- Annexe

```

241      [
242          convert_ast_to_C_sosl [ elem ] env
                ↪ nb_tab;
243      S " - ";
244          convert_ast_to_C_sosl l env nb_tab;
245          convert_ast_to_C_sosl q env nb_tab;
246      ]
247  | Noeud (Opérateur Foix, elem :: l) :: q ->
248      L
249      [
250          convert_ast_to_C_sosl [ elem ] env
                ↪ nb_tab;
251      S " * ";
252          convert_ast_to_C_sosl l env nb_tab;
253          convert_ast_to_C_sosl q env nb_tab;
254      ]
255  | Noeud (Opérateur Division, elem :: l) :: q ->

```

5- Annexe

```
256         L
257         [
258             convert_ast_to_C_sosl [ elem ] env
                ↪ nb_tab;
259         S " / ";
260         convert_ast_to_C_sosl l env nb_tab;
261         convert_ast_to_C_sosl q env nb_tab;
262     ]
263 | Noeud (Parentheseouvrante, []) :: q ->
264     L [ S "("; convert_ast_to_C_sosl q env nb_tab
        ↪ ]
265 | Noeud (Parenthesefermante, []) :: q ->
266     L [ S ")""; convert_ast_to_C_sosl q env nb_tab
        ↪ ]
267 | Noeud (OperateurLogique NonEquivalent, [ p1; p2
        ↪ ]) :: q
268 | Noeud (Compareur NonEgal, [ p1; p2 ]) :: q ->
```

5- Annexe

```

269         L
270         [
271             convert_ast_to_C_sosl [ p1 ] env 0;
272             S " != ";
273             convert_ast_to_C_sosl [ p2 ] env 0;
274             convert_ast_to_C_sosl q env nb_tab;
275         ]
276 | Noeud (OpérateurLogique Equivalent, [ p1; p2 ])
    ↪ :: q
277 | Noeud (Comparateur Egal, [ p1; p2 ]) :: q ->
278     L
279     [
280         convert_ast_to_C_sosl [ p1 ] env 0;
281         S " == ";
282         convert_ast_to_C_sosl [ p2 ] env 0;
283         convert_ast_to_C_sosl q env nb_tab;
284     ]

```


5- Annexe

```
285 | Noeud (Compareur StrictPlusPetit, [ p1; p2 ])
    ↪ :: q ->
    L
    [
288     convert_ast_to_C_sosl [ p1 ] env 0;
289     S " < ";
290     convert_ast_to_C_sosl [ p2 ] env 0;
291     convert_ast_to_C_sosl q env nb_tab;
292   ]
293 | Noeud (Compareur PlusPetit, [ p1; p2 ]) :: q
    ↪ ->
    L
    [
296     convert_ast_to_C_sosl [ p1 ] env 0;
297     S " <=" ;
298     convert_ast_to_C_sosl [ p2 ] env 0;
299     convert_ast_to_C_sosl q env nb_tab;
```

5- Annexe

```

300     ]
301 | Noeud (Compareteur StrictPlusGrand, [ p1; p2 ])
    ↪ :: q ->
302     L
303     [
304         convert_ast_to_C_sosl [ p1 ] env 0;
305         S " > ";
306         convert_ast_to_C_sosl [ p2 ] env 0;
307         convert_ast_to_C_sosl q env nb_tab;
308     ]
309 | Noeud (Compareteur PlusGrand, [ p1; p2 ]) :: q
    ↪ ->
310     L
311     [
312         convert_ast_to_C_sosl [ p1 ] env 0;
313         S " >=";
314         convert_ast_to_C_sosl [ p2 ] env 0;

```

5- Annexe

```
315         convert_ast_to_C_sosl q env nb_tab;
316     ]
317 | Noeud (OpérateurLogique Et, [ p1; p2 ]) :: q ->
318     L
319     [
320         convert_ast_to_C_sosl [ p1 ] env 0;
321         S " && ";
322         convert_ast_to_C_sosl [ p2 ] env 0;
323         convert_ast_to_C_sosl q env nb_tab;
324     ]
325 | Noeud (OpérateurLogique Ou, [ p1; p2 ]) :: q ->
326     L
327     [
328         convert_ast_to_C_sosl [ p1 ] env 0;
329         S " || ";
330         convert_ast_to_C_sosl [ p2 ] env 0;
331         convert_ast_to_C_sosl q env nb_tab;
```

5- Annexe

```

332     ]
333 | Noeud (OpérateurLogique Non, [ p1; p2 ]) :: q
    ↪ ->
334     L
335     [
336         convert_ast_to_C_sosl [ p1 ] env 0;
337         S " ! ";
338         convert_ast_to_C_sosl [ p2 ] env 0;
339         convert_ast_to_C_sosl q env nb_tab;
340     ]
341 | Noeud (Opérateur Puissance, [ p1; p2 ]) :: q ->
342     L
343     [
344         S "pow((long)";
345         convert_ast_to_C_sosl [ p1 ] env 0;
346         S ", ";
347         convert_ast_to_C_sosl [ p2 ] env 0;

```

5- Annexe

```

348         S "(long))";
349         convert_ast_to_C_sosl q env nb_tab;
350     ]
351 | Noeud (Syntax If, condition :: instructions) ::
    ↪ q ->
352     L
353     [
354         Traduction.tabs_to_string nb_tab;
355         S "if (";
356         convert_ast_to_C_sosl [ condition ] env
    ↪ nb_tab;
357         S "){";
358         convert_ast_to_C_sosl instructions env
    ↪ (nb_tab + 1);
359         Traduction.tabs_to_string nb_tab;
360         S "}";
361         convert_ast_to_C_sosl q env nb_tab;

```

5- Annexe

```

362     ]
363 | Noeud (Syntax Else_if, condition ::
    ↳ instructions) :: q ->
364     L
365     [
366         Traduction.tabs_to_string nb_tab;
367         S "else if (";
368         convert_ast_to_C_sosl [ condition ] env
    ↳ nb_tab;
369         S "){";
370         convert_ast_to_C_sosl instructions env
    ↳ (nb_tab + 1);
371         Traduction.tabs_to_string nb_tab;
372         S "}";
373         convert_ast_to_C_sosl q env nb_tab;
374     ]
375 | Noeud (Syntax Else, instructions) :: q ->

```

5- Annexe

```

376         L
377         [
378             Traduction.tabs_to_string nb_tab;
379             S "else {";
380             convert_ast_to_C_sosl instructions env
381                 ↪ (nb_tab + 1);
382             Traduction.tabs_to_string nb_tab;
383             S "}";
384             convert_ast_to_C_sosl q env nb_tab;
385         ]
386     | Noeud
387     ( Syntax For,
388       Noeud (Opérateur Assignment, [ variable;
389           ↪ valeur ])
390       :: fin
391       :: Noeud (Syntax Step, [ pas ])
392       :: instructions )

```

5- Annexe

```
391      :: q ->
392      L
393      [
394          Traduction.tabs_to_string nb_tab;
395          S "for (";
396          convert_ast_to_C_sosl
397              [ Noeud (Operateur Assignment, [
398                  ↪ variable; valeur ]) ]
399              env 0;
400          S " ";
401          convert_ast_to_C_sosl
402              [ Noeud (Compareur StrictPlusPetit, [
403                  ↪ variable; fin ]) ]
404              env 0;
405          S "; ";
406          convert_ast_to_C_sosl [ variable ] env 0;
407          S "=";
```


5- Annexe

```

406         convert_ast_to_C_sosl
407         [ Noeud (Operateur Plus, [ variable;
          ↪ pas ]) ]
408         env 0;
409         S ") {";
410         convert_ast_to_C_sosl instructions env
          ↪ (nb_tab + 1);
411         Traduction.tabs_to_string nb_tab;
412         S "}";
413         convert_ast_to_C_sosl q env nb_tab;
414     ]
415 | Noeud (Syntax While, condition :: instructions)
    ↪ :: q ->
416     L
417     [
418         Traduction.tabs_to_string nb_tab;
419         S "while (";
```

5- Annexe

```

420         convert_ast_to_C_sosl [ condition ] env
           ↪ 0;
421     S "){";
422     convert_ast_to_C_sosl instructions env
           ↪ (nb_tab + 1);
423     Traduction.tabs_to_string nb_tab;
424     S "}";
425     convert_ast_to_C_sosl q env nb_tab;
426 ]
427 | Noeud (Integer s, []) :: q -> L [ S s;
   ↪ convert_ast_to_C_sosl q env nb_tab ]
428 | Noeud (Floating s, []) :: q -> L [ S s;
   ↪ convert_ast_to_C_sosl q env nb_tab ]
429 | Noeud (Double s, []) :: q ->
430     let d =
431         String.fold_left

```

5- Annexe

```

432         (fun acc x -> if x = 'd' then acc ^ "e"
           ↪ else acc ^ String.make 1 x)
433         "" s
434     in
435     L [ S d; convert_ast_to_C_sosl q env nb_tab ]
436     (* convertit les d en e de fortran *)
437 | Noeud (Booleen b, []) :: q ->
438     L
439     [
440         S (if b then "true" else "false");
441         ↪ convert_ast_to_C_sosl q env nb_tab;
442     ]
443 | Noeud (Chaine s, []) :: q -> L [ S s;
   ↪ convert_ast_to_C_sosl q env nb_tab ]
444 | Noeud (Syntax Function, Noeud (Name n, []) ::
   ↪ l) :: q ->

```

5- Annexe

```

444     let sosl, l2 = get_function_param_list l env
      ↪ in
445   L
446     [
447       tabs_to_string nb_tab;
448       get_function_return_type
449       (Noeud (Syntax Function, Noeud (Name n,
      ↪   [])) :: 1))
450       env;
451       S " ";
452       S n;
453       S "(";
454       sosl;
455       S "){"n";
456       convert_ast_to_C_sosl l2 env (nb_tab +
      ↪   1);
457       tabs_to_string nb_tab;

```

5- Annexe

```

458         S "} \n";
459     ]
460 | Noeud (Syntax Call, Noeud (Name n, []) :: l) ::
    ↪ q ->
461     L
462     [
463         S n;
464         S "(";
465         convert_ast_to_C_sosl l env 0;
466         S ")";
467         convert_ast_to_C_sosl q env nb_tab;
468     ]
469 | Noeud (Syntax Return, l) :: q ->
470     L
471     [
472         tabs_to_string nb_tab;
473         S "return ";

```

5- Annexe

```

474         convert_ast_to_C_sosl l env 0;
475         S ";\n";
476         convert_ast_to_C_sosl q env nb_tab;
477     ]
478
479 | Noeud (NewLine, []) :: q -> L [ S "\n";
    ↪ convert_ast_to_C_sosl q env nb_tab ]
480 | Noeud (t, _) :: q ->
481     print_token t;
482     failwith "La syntaxe donnée n'est pas encore
    ↪ prise en charge\n"
483
484 (** convertit l'arbre de syntaxe abstrait [ast] et
    ↪ ajoute les librairies
485     nécessaires depuis [biblios] grace à [env] *)
486 let convert_ast_to_C (ast : ast list) (env :
    ↪ environnement)

```

5- Annexe

```
487      (biblios : Bibliothèques.libs) : string =  
488      generate_library_imports biblios  
489      ^ Traduction.string_of_string_or_string_list  
      ↪ (convert_ast_to_C_sosl ast env 0)
```

490

491

5- Annexe

```

1  open Environnement
2  open Abstract_tokens
3  open Bibliothèques
4  open Convert_to_abstract
5  open LL1
6  open Traduction
7
8  let rec convert_ast_to_Fortran_sosl (ast : ast
   ↪ list) (env : environnement)
9      (nb_tab : int) : string_or_string_list =
10     match ast with
11     | [] -> S ""
12     | [ Noeud (ProgramRoot, l) ] ->
   ↪ convert_ast_to_Fortran_sosl l env nb_tab
13     | Noeud (Syntax Program, Noeud (Name nom, []) ::
   ↪ l1) :: q ->
14         L

```


5- Annexe

```

15         [
16             S "program ";
17             S nom;
18             convert_ast_to_Fortran_sosl l1 env
19                 ↪ (nb_tab + 1);
20             S "end program\n";
21             convert_ast_to_Fortran_sosl q env nb_tab;
22         ]
23 | Noeud (Commentaire c, []) :: q ->
24     L
25     [
26         Traduction.tabs_to_string nb_tab;
27         S "!";
28         S c;
29         convert_ast_to_Fortran_sosl q env nb_tab;
30     ]
31 | Noeud (Syntax Print, l2) :: q ->

```

5- Annexe

```

31      L
32      [
33          Traduction.tabs_to_string nb_tab;
34          S "print *";
35          L
36              (List.rev
37                  (List.fold_left
38                      (fun acc x ->
39                          convert_ast_to_Fortran_sosl [ x
40                              ↪ ] env nb_tab
41                              :: S ", " :: acc)
42                      [] 12));
43          convert_ast_to_Fortran_sosl q env nb_tab;
44      ]
45      (* définit le type des variables *)
46      | Noeud (Syntax Real, l) :: q ->
47          let s = convert_ast_to_Fortran_sosl l env 0
48              ↪ in

```

5- Annexe

```

47         L
48         [
49             Traduction.tabs_to_string nb_tab;
50             S "real ::";
51             S;
52             convert_ast_to_Fortran_sosl q env nb_tab;
53         ]
54 | Noeud (Syntax Integer, l) :: q ->
55     let s = convert_ast_to_Fortran_sosl l env 0
56     ↪ in
57     L
58     [
59         Traduction.tabs_to_string nb_tab;
60         S "integer ::";
61         S;
62         convert_ast_to_Fortran_sosl q env nb_tab;
63     ]

```

5- Annexe

```
63 | Noeud (Syntax Logical, l) :: q ->
64 |   let s = convert_ast_to_Fortran_sosl l env 0
65 |     ↪ in
66 |     L
67 |       [
68 |         Traduction.tabs_to_string nb_tab;
69 |         S "logical ::";
70 |         s;
71 |         convert_ast_to_Fortran_sosl q env nb_tab;
72 |       ]
73 | Noeud (Syntax Double_precision, l) :: q ->
74 |   let s = convert_ast_to_Fortran_sosl l env 0
75 |     ↪ in
76 |     L
77 |       [
78 |         Traduction.tabs_to_string nb_tab;
79 |         S "double precision ::";
```

5- Annexe

```

78         S;
79         convert_ast_to_Fortran_sosl q env nb_tab;
80     ]
81 | Noeud (Opérateur Assignment, Noeud (Name s,
↪   [ ] ) :: l) :: q ->
82     L
83     [
84         Traduction.tabs_to_string nb_tab;
85         S s;
86         S " = ";
87         convert_ast_to_Fortran_sosl l env 0;
88         convert_ast_to_Fortran_sosl q env nb_tab;
89     ]
90 | Noeud (Name s, [ ] ) :: q ->
91     L [ S s; convert_ast_to_Fortran_sosl q env
↪     nb_tab ]
92 | Noeud (Opérateur Plus, elem :: l) :: q ->

```

5- Annexe

```
93         L
94         [
95             convert_ast_to_Fortran_sosl [ elem ] env
96             ↪ nb_tab;
97         S " + ";
98         convert_ast_to_Fortran_sosl l env nb_tab;
99         convert_ast_to_Fortran_sosl q env nb_tab;
100     ]
101 | Noeud (Opérateur Moins, elem :: l) :: q ->
102     L
103     [
104         convert_ast_to_Fortran_sosl [ elem ] env
105         ↪ nb_tab;
106     S " - ";
107     convert_ast_to_Fortran_sosl l env nb_tab;
108     convert_ast_to_Fortran_sosl q env nb_tab;
109 ]
```

5- Annexe

```
108 | Noeud (Opérateur Foix, elem :: l) :: q ->
109     L
110     [
111         convert_ast_to_Fortran_sosl [ elem ] env
112         ↪ nb_tab;
113         S " * ";
114         convert_ast_to_Fortran_sosl l env nb_tab;
115         convert_ast_to_Fortran_sosl q env nb_tab;
116     ]
117 | Noeud (Opérateur Division, elem :: l) :: q ->
118     L
119     [
120         convert_ast_to_Fortran_sosl [ elem ] env
121         ↪ nb_tab;
122         S " / ";
123         convert_ast_to_Fortran_sosl l env nb_tab;
124         convert_ast_to_Fortran_sosl q env nb_tab;
```

5- Annexe

```

123     ]
124 | Noeud (Parentheseouvrante, []) :: q ->
125     L [ S "("; convert_ast_to_Fortran_sosl q env
126         ↪ nb_tab ]
127 | Noeud (Parenthesefermante, []) :: q ->
128     L [ S ")"; convert_ast_to_Fortran_sosl q env
129         ↪ nb_tab ]
130 | Noeud (OpérateurLogique NonEquivalent, [ p1; p2
131     ↪ ]) :: q ->
132     L
133     [
134         convert_ast_to_Fortran_sosl [ p1 ] env 0;
135         S " .neqv. ";
136         convert_ast_to_Fortran_sosl [ p2 ] env 0;
137         convert_ast_to_Fortran_sosl q env nb_tab;
138     ]
139 | Noeud (Compérateur NonEgal, [ p1; p2 ]) :: q ->

```


5- Annexe

```

137         L
138         [
139             convert_ast_to_Fortran_sosl [ p1 ] env 0;
140             S " /= ";
141             convert_ast_to_Fortran_sosl [ p2 ] env 0;
142             convert_ast_to_Fortran_sosl q env nb_tab;
143         ]
144 | Noeud (OpérateurLogique Equivalent, [ p1; p2 ])
145 ↪ :: q ->
146         L
147         [
148             convert_ast_to_Fortran_sosl [ p1 ] env 0;
149             S " .eqv. ";
150             convert_ast_to_Fortran_sosl [ p2 ] env 0;
151             convert_ast_to_Fortran_sosl q env nb_tab;
152         ]
153 | Noeud (Comparateur Egal, [ p1; p2 ]) :: q ->

```

5- Annexe

```

153         L
154         [
155             convert_ast_to_Fortran_sosl [ p1 ] env 0;
156             S " == ";
157             convert_ast_to_Fortran_sosl [ p2 ] env 0;
158             convert_ast_to_Fortran_sosl q env nb_tab;
159         ]
160 | Noeud (Compareur StrictPlusPetit, [ p1; p2 ])
    ↪ :: q ->
161     L
162     [
163         convert_ast_to_Fortran_sosl [ p1 ] env 0;
164         S " < ";
165         convert_ast_to_Fortran_sosl [ p2 ] env 0;
166         convert_ast_to_Fortran_sosl q env nb_tab;
167     ]
168 | Noeud (Compareur PlusPetit, [ p1; p2 ]) :: q
    ↪ ->

```

5- Annexe

```

169         L
170         [
171             convert_ast_to_Fortran_sosl [ p1 ] env 0;
172             S " <= ";
173             convert_ast_to_Fortran_sosl [ p2 ] env 0;
174             convert_ast_to_Fortran_sosl q env nb_tab;
175         ]
176 | Noeud (Compareur StrictPlusGrand, [ p1; p2 ])
↪ :: q ->
177         L
178         [
179             convert_ast_to_Fortran_sosl [ p1 ] env 0;
180             S " > ";
181             convert_ast_to_Fortran_sosl [ p2 ] env 0;
182             convert_ast_to_Fortran_sosl q env nb_tab;
183         ]
184 | Noeud (Compareur PlusGrand, [ p1; p2 ]) :: q
↪ ->

```

5- Annexe

```
185         L
186     [
187         convert_ast_to_Fortran_sosl [ p1 ] env 0;
188         S " >= ";
189         convert_ast_to_Fortran_sosl [ p2 ] env 0;
190         convert_ast_to_Fortran_sosl q env nb_tab;
191     ]
192 | Noeud (OpérateurLogique Et, [ p1; p2 ]) :: q ->
193     L
194     [
195         convert_ast_to_Fortran_sosl [ p1 ] env 0;
196         S " && ";
197         convert_ast_to_Fortran_sosl [ p2 ] env 0;
198         convert_ast_to_Fortran_sosl q env nb_tab;
199     ]
200 | Noeud (OpérateurLogique Ou, [ p1; p2 ]) :: q ->
201     L
```

5- Annexe

```

202      [
203          convert_ast_to_Fortran_sosl [ p1 ] env 0;
204          S " || ";
205          convert_ast_to_Fortran_sosl [ p2 ] env 0;
206          convert_ast_to_Fortran_sosl q env nb_tab;
207      ]
208 | Noeud (OpérateurLogique Non, [ p1; p2 ]) :: q
    ↪ ->
209     L
210     [
211         convert_ast_to_Fortran_sosl [ p1 ] env 0;
212         S " ! ";
213         convert_ast_to_Fortran_sosl [ p2 ] env 0;
214         convert_ast_to_Fortran_sosl q env nb_tab;
215     ]
216 | Noeud (Opérateur Puissance, [ p1; p2 ]) :: q ->
217     L

```

5- Annexe

```

218         [
219             convert_ast_to_Fortran_sosl [ p1 ] env 0;
220             S "**";
221             convert_ast_to_Fortran_sosl [ p2 ] env 0;
222             convert_ast_to_Fortran_sosl q env nb_tab;
223         ]
224 | Noeud (Syntax If, condition :: instructions)
225   :: Noeud (Syntax Else_if, 1)
226   :: q ->
227       L
228       [
229           Traduction.tabs_to_string nb_tab;
230           S "if (";
231           convert_ast_to_Fortran_sosl [ condition ]
232             ↪ env nb_tab;
233           S ") do";
234           convert_ast_to_Fortran_sosl instructions
235             ↪ env (nb_tab + 1);

```

5- Annexe

```

234         Traduction.tabs_to_string nb_tab;
235         convert_ast_to_Fortran_sosl
236             (Noeud (Syntax Else_if, 1) :: q)
237             env nb_tab;
238     ]
239 | Noeud (Syntax If, condition :: instructions) ::
    ↪ Noeud (Syntax Else, 1) :: q
240     ->
241     L
242     [
243         Traduction.tabs_to_string nb_tab;
244         S "if (";
245         convert_ast_to_Fortran_sosl [ condition ]
    ↪     env nb_tab;
246         S ") do";
247         convert_ast_to_Fortran_sosl instructions
    ↪     env (nb_tab + 1);

```

5- Annexe

```

248         Traduction.tabs_to_string nb_tab;
249         convert_ast_to_Fortran_sosl (Noeud
        ↪ (Syntax Else, l) :: q) env nb_tab;
250     ]
251 | Noeud (Syntax If, condition :: instructions) ::
    ↪ q ->
252     L
253     [
254         Traduction.tabs_to_string nb_tab;
255         S "if (";
256         convert_ast_to_Fortran_sosl [ condition ]
        ↪ env nb_tab;
257         S ") do";
258         convert_ast_to_Fortran_sosl instructions
        ↪ env (nb_tab + 1);
259         Traduction.tabs_to_string nb_tab;
260         S "end if";

```


5- Annexe

```

261         convert_ast_to_Fortran_sosl q env nb_tab;
262     ]
263 | Noeud (Syntax Else_if, condition ::
    ↪ instructions)
264     :: Noeud (Syntax Else, 1)
265     :: q ->
266         L
267         [
268             Traduction.tabs_to_string nb_tab;
269             S "else if (";
270             convert_ast_to_Fortran_sosl [ condition ]
    ↪ env nb_tab;
271             S ")";
272             convert_ast_to_Fortran_sosl instructions
    ↪ env (nb_tab + 1);
273             Traduction.tabs_to_string nb_tab;

```

5- Annexe

```

274         convert_ast_to_Fortran_sosl (Noeud
           ↪ (Syntax Else, l) :: q) env nb_tab;
275     ]
276 | Noeud (Syntax Else_if, condition ::
   ↪ instructions)
277     :: Noeud (Syntax Else_if, l)
278     :: q ->
279     L
280     [
281         Traduction.tabs_to_string nb_tab;
282         S "else if (";
283         convert_ast_to_Fortran_sosl [ condition ]
           ↪ env nb_tab;
284         S ")";
285         convert_ast_to_Fortran_sosl instructions
           ↪ env (nb_tab + 1);
286         Traduction.tabs_to_string nb_tab;

```

5- Annexe

```

287         convert_ast_to_Fortran_sosl
288         (Noeud (Syntax Else_if, 1) :: q)
289         env nb_tab;
290     ]
291 | Noeud (Syntax Else_if, condition ::
  ↪ instructions) :: q ->
292     L
293     [
294         Traduction.tabs_to_string nb_tab;
295         S "else if (";
296         convert_ast_to_Fortran_sosl [ condition ]
  ↪ env nb_tab;
297         S ")";
298         convert_ast_to_Fortran_sosl instructions
  ↪ env (nb_tab + 1);
299         Traduction.tabs_to_string nb_tab;
300         S "end if";

```

5- Annexe

```

301         convert_ast_to_Fortran_sosl q env nb_tab;
302     ]
303 | Noeud (Syntax Else, instructions) :: q ->
304     L
305     [
306         Traduction.tabs_to_string nb_tab;
307         S "else ";
308         convert_ast_to_Fortran_sosl instructions
309         ↪ env (nb_tab + 1);
310         Traduction.tabs_to_string nb_tab;
311         S "end if";
312         convert_ast_to_Fortran_sosl q env nb_tab;
313     ]
314 | Noeud
315     ( Syntax For,
316       Noeud (Operateur Assignment, [ variable;
317           ↪ valeur ])

```

5- Annexe

```
316         :: fin
317         :: Noeud (Syntax Step, [ pas ])
318         :: instructions )
319     :: q ->
320     L
321     [
322         Traduction.tabs_to_string nb_tab;
323         S "do ";
324         convert_ast_to_Fortran_sosl
325             [ Noeud (Operateur Assignment, [
326                 ↪ variable; valeur ]) ]
327             env 0;
328         S ", ";
329         convert_ast_to_Fortran_sosl [ fin ] env
330             ↪ 0;
331         S ", ";
332         convert_ast_to_Fortran_sosl [ pas ] env
333             ↪ 0;
```

5- Annexe

```

331         S "\n";
332         convert_ast_to_Fortran_sosl instructions
           ↪ env (nb_tab + 1);
333         Traduction.tabs_to_string nb_tab;
334         S "end do";
335         convert_ast_to_Fortran_sosl q env nb_tab;
336     ]
337 | Noeud (Syntax While, condition :: instructions)
   ↪ :: q ->
338     L
339     [
340         Traduction.tabs_to_string nb_tab;
341         S "do while (";
342         convert_ast_to_Fortran_sosl [ condition ]
           ↪ env 0;
343         S ")";

```

5- Annexe

```

344         convert_ast_to_Fortran_sosl instructions
           ↪ env (nb_tab + 1);
345     Traduction.tabs_to_string nb_tab;
346     S "end do";
347     convert_ast_to_Fortran_sosl q env nb_tab;
348 ]
349 | Noeud (Integer s, []) :: q ->
350     L [ S s; convert_ast_to_Fortran_sosl q env
           ↪ nb_tab ]
351 | Noeud (Floating s, []) :: q ->
352     L [ S s; convert_ast_to_Fortran_sosl q env
           ↪ nb_tab ]
353 | Noeud (Double s, []) :: q ->
354     L [ S s; convert_ast_to_Fortran_sosl q env
           ↪ nb_tab ]
355     (* convertit les d en e de fortran *)
356 | Noeud (Booleen b, []) :: q ->

```

5- Annexe

```

357         L
358         [
359             S (if b then "true" else "false");
360             convert_ast_to_Fortran_sosl q env nb_tab;
361         ]
362 | Noeud (Chaine s, []) :: q ->
363     L [ S s; convert_ast_to_Fortran_sosl q env
364         ↪ nb_tab ]
365 | Noeud (NewLine, []) :: q ->
366     L [ S "\n"; convert_ast_to_Fortran_sosl q env
367         ↪ nb_tab ]
368 | _ -> failwith "La syntaxe donnée n'est pas
369     ↪ encore prise en charge\n"
370
371 let convert_ast_to_Fortran (ast : ast list) (env :
372     ↪ environnement) : string =
373     Traduction.string_of_string_or_string_list

```


5- Annexe

```
370 | (convert_ast_to_Fortran_sosl ast env 0)
```

5- Annexe

```
1  open TraductionFortran
2  open TraductionC
3  open Det_automaton
4  open LL1
5  open Convert_to_abstract
6  open Environnement
7  open Automates
8  open Grammar
9
10 let transpile_Fortran_to_C (fortran_file_name :
    ↪ string) (c_file_name : string) :
11     unit =
12     let lexemes = exec_of_file syntax_automate_det
    ↪ fortran_file_name in
13     let arbre_syntaxique = analyse_LL1 grammar
    ↪ lexemes in
14     let arbre_syntaxique_abstrait =
    ↪ convert_to_abstract arbre_syntaxique in
```

5- Annexe

```
15   let env = create_env_from_ast
    ↪   arbre_syntaxique_abstrait in
16   (* print_env env; *)
17   let code_C =
18       convert_ast_to_C
19       [ arbre_syntaxique_abstrait ]
20       (env)
21       []
22   in
23   let out_file = open_out c_file_name in
24   output_string out_file code_C;
25   close_out out_file
26
27   let transpile_Fortran_to_Fortran (fortran_file_name
    ↪   : string)
28       (c_file_name : string) : unit =
29   let lexemes = exec_of_file syntax_automate_det
    ↪   fortran_file_name in
```

5- Annexe

```
30  let arbre_syntaxique = analyse_LL1 grammar
    ↪ lexemes in
31  let arbre_syntaxique_abstrait =
    ↪ convert_to_abstract arbre_syntaxique in
32  let env = create_env_from_ast
    ↪ arbre_syntaxique_abstrait in
33
34  let code_Fortran =
35      convert_ast_to_Fortran
36      [ arbre_syntaxique_abstrait ]
37      (env)
38  in
39  let out_file = open_out c_file_name in
40  output_string out_file code_Fortran;
41  close_out out_file
```

5- Annexe

```
1  open Transpileurs
2
3  let print_usage () =
4    print_string "Usage : ";
5    print_newline ();
6    print_string Sys.argv.(0);
7    print_string " -C <inupt_fortran_file> [-o
   ↪ <output_c_file>]";
8    print_newline ();
9    print_string Sys.argv.(0);
10   print_string " -Fortran <inupt_fortran_file> [-o
   ↪ <output_fortran_file>]";
11   print_newline ()
12
13  let output_file (len : int) : string =
14    if len == 5 then
15      if
```

5- Annexe

```
16      (* un nom de fichier de sortie est donné *)
17      Sys.argv.(3) = "-o"
18      then Sys.argv.(4)
19      else (
20          print_usage ();
21          raise (Invalid_argument "")
22      else
23          (* un nom de fichier de sortie n'est pas donné,
24             ↪ on en donne un par défaut *)
25          match Sys.argv.(1) with
26          | "-Fortran" -> "out.f90"
27          | "-C" -> "out.c"
28          | _ ->
29              print_usage ();
30              raise (Invalid_argument "")
31  let main () =
```

5- Annexe

```
32  let len = Array.length Sys.argv in
33  if len <> 3 && len <> 5 then print_usage ()
34  else
35    let input_file_name = Sys.argv.(2) in
36    let output_file_name = output_file len in
37    print_string ("Generating " ^ output_file_name
38    ↪ ^ "...");
39    print_newline ();
40    if Sys.argv.(1) = "-C" then
41      transpile_Fortran_to_C input_file_name
42      ↪ output_file_name
43    else transpile_Fortran_to_Fortran
44      ↪ input_file_name output_file_name;
45    print_string ("Finished generating " ^
46    ↪ output_file_name);
47    print_newline ();
```

5- Annexe

```
45 | let _ = main ()
```