



RODEZ

---

# MANUEL UTILISATEUR INTERPRÉTEUR DU LANGAGE LIR

---

PROJET PROPOSÉ PAR FRÉDÉRIC BARRIOS

Nicolas CAMINADE, Sylvan COURTIOL,  
Pierre DEBAS, Heïa DEXTER,  
Lucàs VABRE

# Sommaire

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Éléments de langage</b>	<b>4</b>
2.1	Mots-clés . . . . .	4
2.1.1	Commandes générales . . . . .	4
2.1.2	Instructions de programmation . . . . .	4
2.2	Étiquettes . . . . .	5
2.3	Constantes littérales . . . . .	5
2.3.1	Entiers . . . . .	5
2.3.2	Chaînes de caractères . . . . .	5
2.4	Identificateurs . . . . .	5
2.4.1	Identificateurs d'entiers . . . . .	5
2.4.2	Identificateurs de Chaînes . . . . .	6
2.5	Expressions . . . . .	6
2.5.1	Expression entière . . . . .	6
2.5.2	Expression sur chaînes . . . . .	6
2.5.3	Conditions . . . . .	6
<b>3</b>	<b>Utilisation</b>	<b>7</b>
3.1	Utilisation des commandes et instructions . . . . .	7
3.1.1	Commandes . . . . .	7
3.1.2	Instructions . . . . .	8
3.2	Programmation en LIR . . . . .	9
<b>4</b>	<b>Exemples d'utilisation</b>	<b>10</b>

## **Chapitre 1**

# **Installation**

## Chapitre 2

# Éléments de langage

### 2.1 Mots-clés

#### 2.1.1 Commandes générales

**Debut** : Vide l'intégralité du contexte d'exécution.

**Fin** : Quitte l'interpréteur.

**Def** : Sert à voir toutes les variables définies dans la session courante (les identificateurs et les valeurs).

**Lance** : Exécute le programme chargé en mémoire avec ou sans étiquette.

**Efface** : Supprime une ou plusieurs lignes de code suivant les étiquettes début et fin mises en argument.

**Liste** : Affiche toutes les lignes de programme mémorisées dans l'ordre croissant entre les numéros de ligne de l'intervalle étiquette début et fin donné en argument. Sans argument Affiche toutes les lignes de programme dans l'ordre croissant des numéros de ligne.

**Sauve** : Sauvegarde un programme LIR dans un fichier.

**Charge** : Charge un programme LIR préalablement enregistré dans un fichier.

#### 2.1.2 Instructions de programmation

**Var** : Stocke une chaîne dans une variable pour pouvoir récupérer/manipuler cet/cette entier/chaîne plus tard dans le programme.

**Entre** : Invite l'utilisateur à saisir la valeur d'une variable au clavier afin de pouvoir communiquer avec le contexte de l'interpréteur et le modifier.

**Affiche** : Affiche le contenu de l'expression que l'on lui donne afin de pouvoir récupérer/vérifier le/les résultats du programme. Sans argument, cela provoque un saut de ligne.

**Vaen** : Effectue un saut vers une ligne spécifique d'un programme afin de créer des branchements ou des itérations dans le programme.

**Si ... vaen** : Effectue un saut vers une ligne spécifique d'un programme si la condition est remplie pour créer des branchements ou des itérations des programmes.

**Procédure** : Transfère l'exécution au numéro d'étiquette spécifié afin de pouvoir exécuter le programme puis reprendre puis reprendre en séquence une fois la procédure terminée (Fait appel à une fonction).

**Retour** : Retourne à la suite de la ligne de code qui a précédé l'appel de la procédure afin de terminer la fonction en cours et reprendre l'exécution du programme.

**Stop** : Arrêter le programme.

## 2.2 Étiquettes

Les étiquettes sont des numéros servant à repérer les lignes d'un programme et à les mettre en ordre. Ces numéros sont des entiers positifs entre 1 et 99999. Il est conseillé de numéroter initialement les lignes de 5 en 5, ou bien de 10 en 10.

## 2.3 Constantes littérales

Elles sont de 2 types : entier signé ou chaînes de caractères.

### 2.3.1 Entiers

Les valeurs entières sont exprimées en base 10 et donc formées des caractères +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Leur valeur est comprise dans l'intervalle ( $-2^{31}=-2147483648$ ,  $2^{31}=2147483648$ ).

### 2.3.2 Chaînes de caractères

Les chaînes seront entourées de guillemets et limitées à 70 caractères.

## 2.4 Identificateurs

Les identificateurs n'ont pas besoin d'être déclarés : leur première utilisation suffit à les créer. Si cette première utilisation n'est pas une affectation ou une entrée leur valeur sera choisie arbitrairement.

Toutes les données utilisées sont variables. Leurs valeurs sont conservées et modifiables durant la session de l'interpréteur. Elles ne sont détruites que par la commande `debut`.

La portée des variables est globale et elles ne peuvent pas être soumises à du transtypage.

### 2.4.1 Identificateurs d'entiers

Les nom identifiants des entiers commenceront obligatoirement par une lettre qui sera suivie d'au plus 24 lettres ou chiffres.

### 2.4.2 Identificateurs de Chaînes

Les noms identifiant des chaînes de caractères commenceront par le symbole \$, suivi d'une lettre, qui sera suivie d'au plus 24 lettres ou chiffres.

## 2.5 Expressions

### 2.5.1 Expression entière

Les expressions concernent toujours deux opérandes séparés par un opérateur (notation infixe). La présence d'espace (séparateur neutre) n'est pas obligatoire (mais conseillée). Un opérande est soit une constante littérale, soit un identificateur. L'opérateur est un caractère symbolisant les opérations arithmétiques courantes : +, -, \*, /, %

- + : addition entière
- : soustraction entière
- \* : multiplication entière
- / : quotient de la division entière
- % : reste de la division entière

### 2.5.2 Expression sur chaînes

La seule opération utilisée sur les chaînes est la concaténation. Composée de 2 opérandes chaînes, c'est-à-dire des constantes chaînes ou identificateurs commençant par \$.

### 2.5.3 Conditions

Les expressions logiques ne sont utilisées qu'avec l'instruction si. Les expressions logiques concernent donc toujours deux opérandes séparés par un opérateur relationnel (notation infixe). Un opérande est soit une constante, soit un identificateur. L'opérateur relationnel oprel est un symbole parmi : =, <>, <, <=, >, >=

- = : représente le test d'égalité
- <> : représente l'inégalité
- < : infériorité stricte
- > : supériorité stricte
- <= : inférieur ou égal
- => : supérieur ou égal

Ces opérateurs ont le sens habituel pour les entiers. Pour les chaînes, c'est l'ordre lexicographique habituel qui sera utilisé.

# Chapitre 3

## Utilisation

### 3.1 Utilisation des commandes et instructions

#### 3.1.1 Commandes

**Debut** : Sur un programme chargé en mémoire centrale j'efface toutes les lignes de code et variables déclarées avec la commande `debut`.

**Fin** : Je suis en train d'utiliser l'interpréteur, je quitte l'interpréteur pour la session courante en exécutant la commande `fin`.

**Defs** : Les variables sont définies dans la session courante de l'interpréteur j'affiche le contexte actuel en exécutant la commande `defs`.

**Lance** : A partir de lignes d'instructions chargées dans la session courante de l'interpréteur LIR lorsque j'entre la commande `lance` sans arguments et la valide, le programme s'exécute à partir de l'étiquette la plus petite. Et avec argument les lignes d'instructions chargées dans la session courante de l'interpréteur LIR et lorsque j'entre la commande `lance` sans arguments et la valide le programme s'exécute à partir de l'étiquette passé en argument.

Exemple : `lance <étiquette début> : <étiquette fin>`

**Efface** : A partir d'une ou plusieurs lignes de programme mémorisées et leur étiquettes

on tape la commande : `efface <étiquette début> : <étiquette fin>`.

L'interpréteur effacera alors les lignes de programme dont le numéro d'étiquette est compris dans la plage.

**Liste** : J'entre la commande `liste <etiquette_debut>:<etiquette_fin>` l'interpréteur va afficher toutes les lignes de programme mémorisées, dans l'ordre croissant de leur étiquette et dont les étiquettes sont situées dans cet intervalle donné. Et sans argument la commande `liste` affiche toutes les lignes de programme mémorisées, dans l'ordre croissant de leur étiquette.

**Sauve** : Quand programme (avec des étiquettes) ai été saisi, lorsque on entre la commande `sauve` avec en argument le chemin du fichier (dans lequel on souhaite sauvegarder le travail), `sauve <cheminFichier>` les lignes de codes tapées dans l'interpréteur s'enregistrent dans le fichier passé en argument de la commande pour pouvoir être rechargées plus tard par l'interpréteur LIR avec la commande `charge <cheminFichier>`.

**Charge** : On a un fichier contenant un programme LIR dans notre ordinateur lorsque j'entre la commande `charge` avec en argument le chemin de ce fichier les lignes de codes enregistrées dans le fichier sont chargées dans le programme pour pouvoir être exécutées et/ou modifiées par l'interpréteur LIR.

Exemple : `sauve <D:\leCheminDuFichier\leFichier.LextensionDuFichier>`  
OU `sauve <leFichier.LextensionDuFichier>`

### 3.1.2 Instructions

**Var** : L'interpréteur enregistre dans la variable spécifiée l'expression.

Syntaxe : `<etiquette> var <identificateur> = <expression>`

Exemple :

Pour une variable de chaine de caractère : `5 var $varChaine = "chaine"`

Pour une variable d'entier : `5 var varEntier = 42`

**Entre** : Lorsque la valeur est saisie, elle est stockée dans la variable déterminée dans le contexte, sous réserve que les types concordent.

Syntaxe : `<etiquette> entre <identificateur>`

Exemple :

Pour une variable de chaine de caractère : `10 entre $varChaine`

Pour une variable d'entier : `10 entre varEntier`

**Affiche** : L'interpréteur évalue dans l'expression spécifiée la valeur de celle-ci et renvoie cette valeur sur la console. Si cette commande n'a pas d'argument elle effectue un saut de ligne.

Syntaxe : `<etiquette> affiche [identificateur]`

Exemple :

Sans argument : `15 affiche`

Pour une variable de chaine de caractère (`$varChaine = une`) : `15 affiche $varChaine`

Pour une variable d'entier (`varEntier = 12`) : `15 affiche varEntier + 5`

**Vaen** : Lors de l'exécution de l'instruction, le programme ignorera les lignes suivantes et sautera directement à la ligne indiquée.

Syntaxe : `<etiquette> vaen <etiquette>`

Exemple : `20 vaen 30`

**Si ... vaen** : Lors de l'exécution de l'instruction, le programme ignorera les lignes suivantes et sautera directement à la ligne indiquée si et seulement si la condition (booléenne) imposée est valide.

Syntaxe : `<etiquette> si <condition> vaen <etiquette>`

Exemple : `25 si varEntier <> 8 vaen 35`

**Procedure** : L'interpréteur va chercher la ligne qui a pour identificateur celui référencé en étiquette et va l'exécuter jusqu'à la fin de la séquence.

Syntaxe : `<etiquette> procedure <etiquette>`

Exemple : `30 procedure 60`



**Retour :** L'interpréteur va chercher la ligne qui suivait l'instruction "procedure" et va l'exécuter jusqu'à la fin de la séquence.

Syntaxe : `<etiquette> retour`

Exemple : `70 retour`

**Stop :** A son exécution, le programme s'arrête lorsqu'il a atteint l'étiquette spécifiée.

Syntaxe : `<etiquette> stop`

Exemple : `40 stop`

## 3.2 Programmation en LIR

La programmation en LIR peut s'effectuer de deux manières :

1. **Programmation directement dans l'interpréteur.** Celle-ci s'effectue en ajoutant une étiquette, donnant l'ordre d'exécution, avant l'instruction saisie. Les lignes de code peuvent être ajoutés dans le désordre dans le programme chargé. Le remplacement d'une instruction à une certaine étiquette se fait par la saisie d'une ligne de code ayant la même étiquette. Les commandes liste et efface permettent l'édition du programme.
2. **Programmation dans un fichier d'extension .lir** qui sera chargé à posteriori dans l'interpréteur avec la commande charge. Une ligne saisie dans le fichier correspond à une saisie dans l'interpréteur ainsi les mêmes spécificité s'appliquent. Une ligne invalide empêche le chargement de l'entièreté du fichier. Les lignes blanches sont ignorées par l'interpréteur.

Ces deux méthodes se complètent grâce aux commandes sauve et charge permettant de passer d'une méthode à l'autre.

## Chapitre 4

# Exemples d'utilisation

Exemples d'utilisation de l'interpréteur LIR :

Interpréteur Langage IUT de Rodez, bienvenue !  
Entrez vos commandes et instructions après l'invite ?

```
? defs
aucune variable n'est définie
? liste
aucune ligne à afficher
? 10 var $message = "Hello World !"
ok
? 20 afficher $message
nok : mot clé inconnu
? 20 affiche message
ok
? 20 affiche $message
ok
? liste
10 var $message = "Hello World !"
20 affiche $message
? liste 20:20
20 affiche $message
? 15 stop
ok
? liste
10 var $message = "Hello World !"
15 stop
20 affiche $message
? efface 11:19
ok
? liste
10 var $message = "Hello World !"
20 affiche $message
? 30 stop
ok
? lance
Hello World !
? sauve helloWorld.lir
```

```

ok
? defs
$message = "Hello World !"
? liste
10 var $message = "Hello World !"
20 affiche $message
30 stop
? debut
ok
? defs
aucune variable n'est définie
? liste
aucune ligne à afficher
? charge helloWorld.lir
ok
? liste
10 var $message = "Hello World !"
20 affiche $message
30 stop
? var $message = "Aurevoir !"
ok
? lance 20
Aurevoir !
? fin
Au revoir, à bientôt !

```

Exemples de programmes en LIR :

### 1. Exemple d'un programme demandant un état civil simple :

```

10 affiche "Entre ton nom : "
20 entre $nom
30 affiche "Bienvenue "+$nom
35 affiche
40 var an=2021
50 affiche "Quelle est ton année de naissance ? "
60 entre naissance
65 si naissance > an vaen 50
70 affiche "Tu as autour de "
80 affiche an-naissance
90 affiche "ans "
100 affiche
200 stop

```

Exemple d'exécution :

```

? charge etatCivilSimple.lir
ok
? lance
Entre ton nom : Emmanuel MACRON
Bienvenue Emmanuel MACRON
Quelle est ton année de naissance ? 1977
Tu as autour de 44ans

```

**2. Exemple d'un programme calculant la factorielle à partir d'un entier saisi par l'utilisateur :**

```
10 affiche "Bienvenue dans le programme factorielle.lir !"
20 affiche
30 affiche "Entrez un entier : "
40 entre entier
45 procedure 500
50 procedure 1000
60 affiche entier
70 affiche "! = "
80 affiche factorielle
90 affiche
200 stop

500 si entier >= 0 vaen 600
510 affiche "n! est définie sur l'ensemble des entiers naturels"
520 stop
600 retour
```

```
1000 var factorielle = 1
1010 var entierCourant = 2
1011 var ancienFactorielle = factorielle
1012 var test = factorielle
1015 si entierCourant > entier vaen 1100
1016 si ancienFactorielle <> test vaen 1060
1017 var ancienFactorielle = factorielle
1020 var factorielle = factorielle * entierCourant
1025 var test = factorielle / entierCourant
1030 var entierCourant = entierCourant + 1
1040 vaen 1015
1050 vaen 1100
1060 affiche "dépassement de la capacité des entiers pour "
1070 affiche entier
1080 affiche "!"
1090 affiche
1095 stop
1100 retour
```

Exemple d'exécution :

```
? charge factorielle.lir
ok
? lance
Bienvenue dans le programme factorielle.lir !
Entrez un entier : 10
10! = 3628800
```

**3. Exemple d'un programme déterminant l'entier médian de 3 entiers saisis :**

```
10 affiche "Bienvenue dans le programme Median3Entiers.lir"
20 affiche
30 affiche "Entrez le premier entier : "
```

```
40 entre premier
50 affiche "Entrez le deuxième entier : "
60 entre deuxieme
70 affiche "Entrez le troisième entier : "
80 entre troisieme
90 procedure 1000
100 affiche "Median( "
110 affiche premier
120 affiche ", "
130 affiche deuxieme
140 affiche ", "
150 affiche troisieme
160 affiche ") = "
170 affiche median
180 affiche
250 stop

1000 si premier <= deuxieme vaen 1100
1010 si deuxieme <= troisieme vaen 1200
1020 vaen 1520

1100 si deuxieme <= troisieme vaen 1520
1110 si premier <= troisieme vaen 1540
1120 vaen 1500

1200 si premier <= troisieme vaen 1500
1220 vaen 1540

1500 var median = premier
1510 vaen 1550
1520 var median = deuxieme
1530 vaen 1550
1540 var median = troisieme
1550 retour
```

#### Exemple d'exécution :

```
? charge Median3Entiers.lir
ok
? lance
Bienvenue dans le programme Median3Entiers.lir
Entrez le premier entier : 55
Entrez le deuxième entier : 27
Entrez le troisième entier : 96
Median( 55, 27, 96) = 55
```