

الجمهورية الجزائرية الديمقراطية الشعبية

ⵜⴰⴳⴷⴰⵢⵜ ⵜⴰⵎⴻⵔⴰⵏⵜ ⵜⴰⵖⴻⵔⴰⵏⵜ ⵜⴰⵣⴰⵢⴻⵔⴰⵏⵜ

République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي

ⵎⴰⵏⴷⴰⵢⵜ ⵜⴰⵎⴻⵔⴰⵏⵜ ⵜⴰⵖⴻⵔⴰⵏⵜ ⵜⴰⵣⴰⵢⴻⵔⴰⵏⵜ

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي

ⵎⴰⵏⴷⴰⵢⵜ ⵜⴰⵎⴻⵔⴰⵏⵜ ⵜⴰⵖⴻⵔⴰⵏⵜ ⵜⴰⵣⴰⵢⴻⵔⴰⵏⵜ

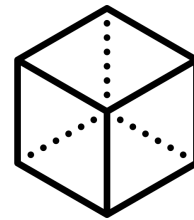
École nationale Supérieure d'Informatique

Projet de compilation Réalisation d'un mini compilateur

Proposition du Langage: *Langage Cube*

Realise par :

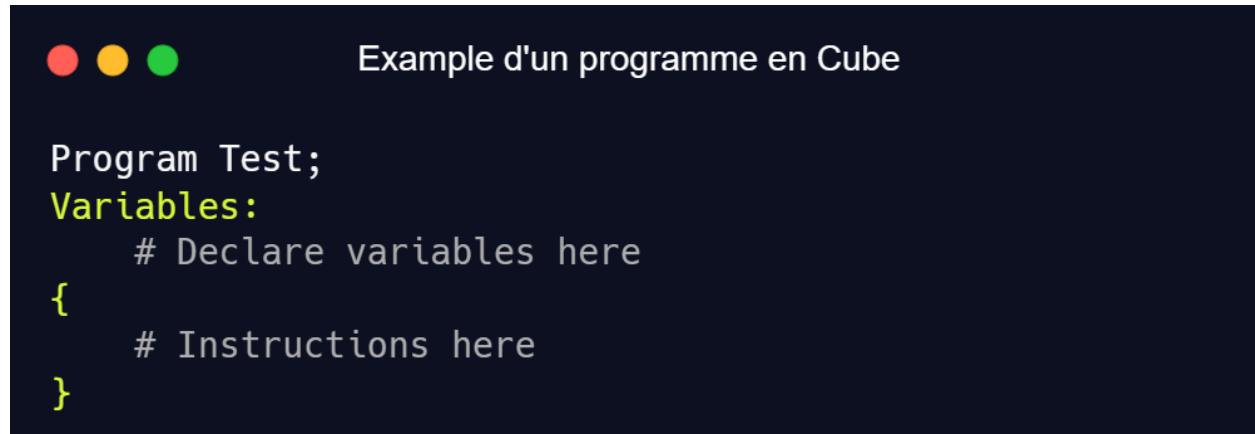
- Medjahdi Islem
- Moussaoui Abdelmouncif
- Habbouche Abderrahmen
- Kenniche Abderrazak



1.Description du Language Cube.....	2
1.1 Commentaires.....	3
1.2 Declarations.....	3
1.2.1 Les types.....	3
1.2.2 Declaration des variables simple :.....	4
1.2.3 Declaration des variables structures :.....	4
1.3 Les instructions de base :.....	5
1.3.1 Affectation :.....	5
1.3.2 Condition :.....	5
1.3.3 Boucles :.....	5
1.3.4 Les entrees / sorties :.....	6
1.4 Les opérateurs :.....	6
1.4.1 Les opérateurs logiques :.....	6
1.4.2 Les opérateurs arithmétiques :.....	6
1.4.3 Les opérateurs de comparaison :.....	7
1.4.4 Règles d'associativité :.....	7

1. Description du Language Cube

Un programme en Langage *Cube* se compose d'une séquence de déclarations et d'instructions. Chaque instruction doit occuper une seule ligne et se conclut par un point virgule ; Les blocs de code sont délimités par les mots-clés { et } .

A screenshot of a code editor window with a dark blue background. The title bar shows three colored circles (red, yellow, green) and the text "Exemple d'un programme en Cube". The code is as follows:

```
Program Test;  
Variables:  
    # Declare variables here  
{  
    # Instructions here  
}
```

1.1 Commentaires

Un commentaire est précédé par un '#'. Il doit être ignoré par le compilateur.

```
# This is a comment in Cube
```

1.2 Declarations

Une déclaration peut prendre la forme de variables simples telles que des *entiers*, des *réels*, des *caractères* ou des *booléens*, ainsi que de variables structurées telles que des *tableaux* ou des *enregistrements*.

1.2.1 Les types

Type	Description	Plages de valeurs
Integer	Représente les nombres entiers.	[-2147483648 , 2147483647]
Real	Représente les nombres réels.	[-3.4*10 ⁻³⁸ , 3.4*10 ³⁸]
Bool	Représente les valeurs de vérité	True ou False
Char	Représente un seul caractère.	Un caractère ASCII valide
Text	Représente une chaîne de caractères	

1.2.2 Declaration des variables simple :

Format	Exemple
TYPE variable_name ;	Integer amount;

Les noms de variables commencent par une lettre ou un trait de soulignement, et peuvent contenir des caractères alphanumériques ou underscores.

Utilisez des guillemets pour entourer les chaînes de caractères.

Évitez les mots réservés et les caractères spéciaux (!, @, #, \$, %).

1.2.3 Declaration des variables structures :

Structure	Format	Exemple
<i>Tableau</i>	TYPE nom[size];	Integer a[4];
<i>Record</i>	Record name : {{ TYPE Key; }}	Record student : {{ Integer age; }}

La taille d'un tableau doit être un entier positif

Pour accéder à un élément d'un tableau on utilise : `table_name[index]` avec index compris entre 0 et size-1

Pour accéder à un attribut d'un Record on utilise : `record_name.attribute_name`

1.3 Les instructions de base :

1.3.1 Affectation :

Format	Example
<code>variable_name = value ;</code>	<code>x = 10;</code> <code>student.age = 21;</code>

1.3.2 Condition :

Format	Example
<code>If (condition) {</code> <code># instructions if true</code> <code>} else {</code> <code># instructions if false</code> <code>}</code>	<code>If (A > 10){</code> <code>A = A + 1;</code> <code>} else {</code> <code>A = A - 1;</code> <code>}</code>

1.3.3 Boucles :

Format	Example
<code>Loop (init, condition , step){</code> <code># instruction to repeat</code> <code>}</code>	<code>Loop (i=0, i<10 , i=i+1) {</code> <code>Output(i);</code> <code>}</code>
<code>While (condition) {</code> <code># instructions to repeat</code> <code>}</code>	<code>While(i < 10){</code> <code>Output(i);</code> <code>}</code>

On peut avoir des boucles imbriquées

1.3.4 Les entrees / sorties :

Instruction	Example
TYPE var_name = Input(prompt);	Real x = Input('Enter a number');
Output(value);	Output("Hello World!");

1.4 Les opérateurs :

1.4.1 Les opérateurs logiques :

Operateur	Format	Example
AND	Expression1 and Expression2	(a>10) and (isOk)
OR	Expression1 or Expression2	(a>10) or (isOk)
NOT	not (Expression)	not (a>10)

L'associativité s'applique aux opérations AND, OR .

1.4.2 Les opérateurs arithmétiques :

Operateur	Format	Example
+	Expression1 + Expression2	10 + A
-	Expression1 - Expression2	A - 2
*	Expression1 * Expression2	(A+2) * 5
//	Expression1 // Expression2 # Expression2 <> zero	A // 3
%	Expression1 % Expression2	A % 2 # modulo

L'associativité s'applique aux opérations + , *

1.4.3 Les opérateurs de comparaison :

$>, >=, <, <=, <>, ==$

1.4.4 Règles d'associativité :

Ce passage explique les priorités et les règles d'associativité dans l'évaluation des expressions mathématiques ou logiques.

1. **Parenthèses** : Les opérations à l'intérieur des parenthèses ont la priorité la plus élevée et sont évaluées en premier.
2. **Not** : L'opérateur logique "not" (négation logique) est appliqué ensuite.
3. **And** : L'opérateur logique "and" (conjonction logique) a une priorité inférieure à "not" et est évalué après.
4. **Or** : L'opérateur logique "or" (disjonction logique) est appliqué après "and".
5. **Multiplication**, Division, Modulo : Ces opérations ont une priorité supérieure à l'addition et à la soustraction.
6. **Addition**, **Soustraction** : Les opérations d'addition et de soustraction ont une priorité inférieure à la multiplication, à la division et au modulo.
7. **Comparaisons** : Les opérations de comparaison (comme $<$, $>$, $<=$, $>=$, $==$, $<>$) sont effectuées en dernier, ayant la priorité la plus basse.

En respectant ces règles, l'ordre d'exécution des opérations est déterminé dans une expression complexe, assurant une évaluation correcte et cohérente.