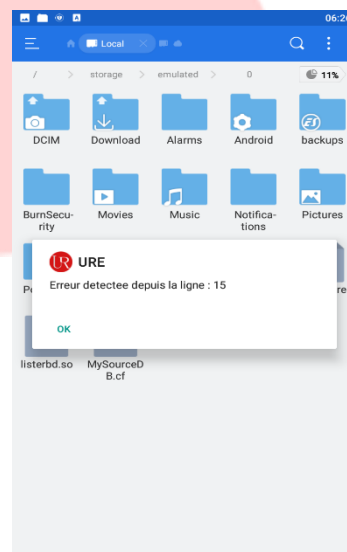
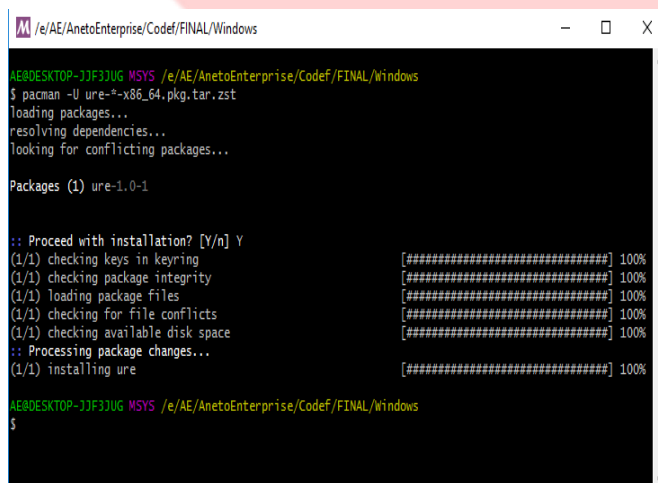
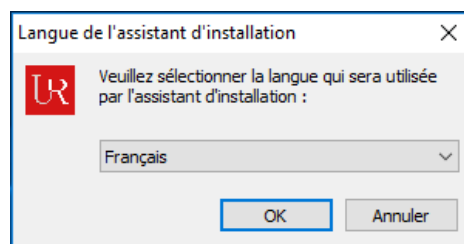


# L'environnement d'exécution universel

LA PROGRAMMATION A LA PORTEE DE TOUS

Universal Runtime Environment (URE)  
ANETOENTERPRISE INC. | KINSHASA RDC



# Introduction

Merci d'avoir choisi l'environnement d'exécution universel (EEU) en tant qu'environnement primordial de votre système d'exploitation pour vous aider à bien créer votre propre langage de programmation universel et d'aider d'autres programmeurs d'utiliser votre nouveau langage à n'importe quelle langue dans le monde. Le problème qui fait en sorte que d'autres personnes dans certains pays n'arrivent pas à maîtriser la programmation informatique, est le manque de compréhension des syntaxes utilisées obligatoirement en anglais. La seule chose à savoir est que, c'est ne pas obligatoire que les gens apprennent à programmer en anglais tandis qu'ils ne comprennent même pas les mots utilisés pour la programmation algorithmique d'une application quelconque. Le taux de l'incompréhension de la programmation informatique dépasse à plus de **75%** dans plusieurs pays à cause d'un langage utilisé et imposé dans le monde entier. Mais la solution est maintenant à la portée de tout le monde afin que vous et moi arrivent à changer non seulement votre pays, mais de changer aussi le monde entier. A la fin de la lecture, vous serez capable de concevoir votre propre langage de programmation universel, programmable et compréhensif dans toutes les langues du monde ou même les langages inconnus extra-terrestres.

## Features

L'environnement d'exécution universel vous facilite les tâches avec :

- ✓ **Le débogueur universel intégré** : Capable de détecter les erreurs dans les syntaxes utilisées par les langages de programmation universel, adaptés et multilingues.
- ✓ **Sécurité des syntaxes appropriée** : Sécurise votre langage de programmation ou application exécutable, pour que personne d'autre n'arrive à le modifier sans votre permission.
- ✓ **Langage formel ou autre** : Vous serez libre de programmer à n'importe quelle langue maternelle ou autre dans le monde.
- ✓ **Cross plates-formes** : Tant que l'environnement d'exécution universel (EEU) est cross plates-formes, programmable et exécutable dans tous les systèmes d'exploitation, votre langage de programmation aussi le sera.

# SOMMAIRE

- 5 | C'est quoi l'URE (Universal Runtime Environment) ?
- 7 | Qu'est-ce qu'un environnement d'exécution ?
- 9 | Exemple d'environnement d'exécution
- 11 | Machine Virtuelle
- 12 | Bibliothèque standard
- 13 | Débogueur
- 15 | Language de programmation
- 17 | Compilateur
- 18 | Assembleur (ASM)
- 19 | Interface de programmation d'application (IPA)
- 20 | Tableau symbolique
- 21 | Installation URE
- 21 | Configuration URE
- 26 | Programmeur (Concepteur langage de programmation)
- 38 | Programmeur (Concepteur des applications)
- 40 | Conclusion

# 1. C'est quoi URE (Universal Runtime Environment) ?

L'environnement d'exécution universel (EEU ou en anglais URE), est un environnement informatique permettant à tous les programmeurs informatiques dans le monde de développer leurs **applications**, **systèmes** et **autres** facilement grâce à sa structure de débogage et d'exécution des programmes écrits dans des différentes langues ou différentes manières pour la meilleure compréhension, sécurité des informations codées ou sécurité des syntaxes utilisées par les programmeurs. Son algorithme à été développé en utilisant le **langage assembleur (ASM)** ou **langage machine**, **C/C++** par le développeur **Guyllit Aneto Minanga** de la nationalité congolaise.

## 1.1. Objectif :

Pour les développeurs experts, l'objectif est de les aider à adapter leurs systèmes, environnement des développements intégré (EDI ou IDE) et langages de programmation à l'environnement d'exécution universel proposé par la société incorporée **AnetoEnterprise Inc** afin que d'autres programmeurs débutants ou même experts arrivent à comprendre les syntaxes et concevoir des applications ou systèmes du future.

En cas des développeurs informatiques débutants ou étudiants en informatique, l'objectif est de rendre plus facilement la programmation informatique de tous les niveaux compréhensifs, afin que dans chaque pays les étudiants comprennent la structure et le comportement adapté de chaque système d'exploitation et langage de programmation.

Dans certains pays, précisément en **Afrique**, les étudiants ont sentences à ne pas être à la page par rapport aux autres étudiants occupants les différents continents restants que nous n'avons pas cités. Prenons un exemple d'un langage de programmation informatique sorti en **2019** ou **2020** par un américain conçus complètement avec des syntaxes Anglo-saxons, les étudiants ou même les nouveaux enseignants dans un pays francophone ou autres, ne seront pas en mesure de s'adapter rapidement au même moment que les étudiants américains ou les étudiants anglophones. Certains étudiants africains anglophones comprendront les syntaxes entières de ce langage proposé par son développeur et d'autres ne comprendront jamais les syntaxes ou peut être la compréhension viendra après quelques années des répétitions. C'est ce qui fait que la compréhension ne pas forcément adaptée dans tous les pays et non plus

au même moment d'apprentissage. C'est à cause de cet inconvénient que les africains sont un peu faible d'inventer ou d'innover quotidiennement.

## 1.2. Idée :

L'idée est de faire en sorte qu'un langage de programmation informatique développé en **anglais, tshiluba, lingala** ou en **swahili** par exemple, ciblant un langage de programmation d'assemblage (**ASM**) qui est le langage proche de la machine, soit manipuler par les programmeurs de tous les pays et en utilisant n'importe quelle langue avec des syntaxes compréhensives adaptées et non seulement les syntaxes anglo-saxons comme les font toujours depuis les décennies.

Avec l'environnement d'exécution universel, un développeur américain peut créer son système ou langage de programmation informatique en anglais, d'autre part, un africain peut développer le sien et les deux langages informatiques développés par deux nationalités différentes, utilisant deux langues différentes des syntaxes ou des instructions, soient manipulés correctement et qu'ils soient traduits dans chaque pays à une langue compréhensive des étudiants. Signifiant que les étudiants américains ou anglophones utilisant la langue anglaise pourraient apprendre un langage de programmation créée en Afrique en **lingala, français** ou en **tshiluba** par exemple, sans aucune difficulté. Mêmement pour les africains pourraient étudier un langage de programmation développé par un américain en anglais, soit compréhensif en **lingala, kikongo, tshiluba**, en **français** ou même en **anglais** la langue par défaut de ce langage de programmation.

## 1.3. Conclusion :

Même si la répétition est la mère de science comme disons-nous, cette complexité d'infériorité ou de supériorité d'utiliser qu'une seule langue pour l'avancer de la technologie devrait changer complètement est que dans chaque continent et dans chaque pays qu'il y ait d'invention, d'innovation et la compréhension assurer pour tout un chacun tant que les **ordinateurs, téléphones mobile** ou d'autres **appareils électronique** ne connaissent aucun langage formel humain. Et c'est grâce à un langage de programmation unique d'assemblage, autrement appelé assembleur conçus pour les processeurs, nos langages formels l'utilisent pour la traduction et l'exécution des programmes informatiques.



Alors pourquoi obliger les gens qu'une seule langue avant la manipulation d'une nouvelle invention ou d'innovation ?

Parce que les inventeurs ou innovateurs sont anglophones ?

## 2. Qu'est-ce qu'un environnement d'exécution ?

Pour mieux comprendre l'environnement d'exécution, un **environnement d'exécution** ou **runtime** en anglais, est un logiciel responsable de l'exécution des programmes informatiques écrits dans un langage de programmation donné. Un runtime offre des services d'exécution de programmes tels que les entrées-sorties, l'arrêt des processus, l'utilisation des services du **système d'exploitation**, le traitement des erreurs de calcul, la génération d'événements, l'utilisation de services offerts dans un autre langage de programmation, et le **débogage** (Débogueur).

Un runtime peut être vu comme une **machine virtuelle** : de la même manière qu'un code natif est exécuté par le processeur, un code objet est exécuté par le runtime. Le runtime sert alors à exécuter du code objet en mettant le code natif à la disposition du processeur pour l'exécution.

Nous pouvons dire dans ce sens que :

**Le runtime est l'interpréteur** : il interprète le code source, manipule les variables, réserve de la mémoire et prend en charge les erreurs d'exécution et peut accompagner d'une bibliothèque standard. Il met en œuvre les fonctionnalités élémentaires du langage, tandis que la bibliothèque standard est typiquement écrite dans le langage lui-même.

Quant à l'environnement d'exécution universel (EEU), contrôle l'exécution des programmes dans les différents langages de programmation, utilisant les différentes langues dans le monde et fournit des services. Ces services sont exposés de manière exacte même si les langages de programmation concernée utilisent des langues différentes de différents programmeurs dans le monde. Cependant, la palette des fonctions est la même dans toutes les langues et ils sont mis en œuvre par le même environnement d'exécution universel.

### 3. Exemple d'environnement d'exécution :



#### 3.1. Node.js :

Est la plateforme logicielle libre en JavaScript, orientée vers les applications réseau événementielles hautement concurrentes qui doivent pouvoir monter en charge.

Elle utilise la machine virtuelle **V8**, la librairie **libuv** pour sa boucle d'évènements.

Parmi les modules natifs de Node.js, on retrouve **http** qui permet le développement de serveur HTTP. Ce qui autorise, lors du déploiement de sites internet et d'applications web développés avec Node.js, de ne pas installer et utiliser des serveurs web tels que **Nginx** ou **Apache**.

Concrètement, Node.js est un environnement bas niveau permettant l'exécution de **JavaScript** côté serveur.

#### 3.2. XULRunner :

Etait un environnement d'exécution d'applications libre qui servait pour l'exécution des applications **XUL**. Il permettait donc de lancer les applications écrites en XUL sans avoir besoin d'installer le navigateur **Mozilla** ou **Firefox**, les deux logiciels originaux interprétant le XUL. L'intérêt est de disposer d'une plate-forme multi système d'exploitation, tournant aussi bien sous **Windows**, **Mac OS** ou **Linux**, et interprétant des programmes écrits sans précision de système d'exploitation. Si les programmes écrits en XUL n'ont pas besoin d'être adaptés à un système d'exploitation particulier, les versions de **XULRunner** doivent par contre être adaptées à chaque système d'exploitation.



### 3.3. JRE (Java Runtime Environment) :

L'**environnement d'exécution Java**, parfois nommé simplement « Java », est une famille de logiciels qui permet l'exécution des programmes écrits en langage de programmation Java, sur différentes plateformes informatiques.

Il est distribué gratuitement par **Oracle Corporation**, sous forme de différentes versions destinées aux systèmes d'exploitation **Windows, Mac OS X** et **Linux**, toutes conformes aux **Java Specification Requests (JSR)**.

**JRE** est souvent considéré comme une plateforme informatique au même titre qu'un système d'exploitation (OS). En effet, s'il n'est pas un OS, il offre les mêmes fonctionnalités par l'intermédiaire de ses bibliothèques et permet ainsi l'exécution des programmes écrits en langage Java sur de nombreux types d'appareils.

Par exemple : **Ordinateurs personnels, Mainframes, téléphones mobiles** ainsi de suite en faisant abstraction des caractéristiques techniques de la plateforme informatique sous-jacente qui exécute le JRE.

### 3.4. URE (Universal Runtime Environment) :

Pour le fameux environnement d'exécution universel (EEU ou URE en anglais), il est non seulement pour l'exécution des applications cross plateformes, considéré aussi comme un environnement d'exécution au même titre qu'un système d'exploitation (SE ou OS), permet aussi de faciliter l'utilisation ou la programmation des langages informatique afin que chaque programmeur dans les différents pays, utilisant la langue différente que son concepteur, arrive à bien comprendre les syntaxes proposées au langage choisi pour que l'avenir des inventions et d'innovations soient assurées.

Prenons un exemple de deux voisines dont, l'une des voisines est bilingue qui s'exprime très bien en **français** et en **lingala**. Tandis que l'autre s'exprime uniquement en **lingala**. Les deux voisines ont un enfant chacune. Mais, l'une des voisines son enfant comprend aussi les deux langues (**Français et Lingala**) tant que sa mère l'apprend souvent.

Alors, la voisine qui s'exprime uniquement en lingala envoie son enfant chez la voisine bilingue afin de récupérer son argent qu'elle avait prêté à son amie (L'autre voisine).

Pour mieux comprendre cette petite histoire ou procédure effectuée dans un ordinateur, téléphone mobile ou autres, l'une des voisines qui est bilingue est le **système d'exploitation informatique**, l'autre voisine est belle et bien le **logiciel** ou **l'application** créé par un programmeur informaticien.

Les deux enfants, celui de la voisine bilingue, est un **environnement d'exécution** et l'autre enfant, est un **langage de programmation** utilisé à la conception du logiciel ou application par le programmeur informaticien.

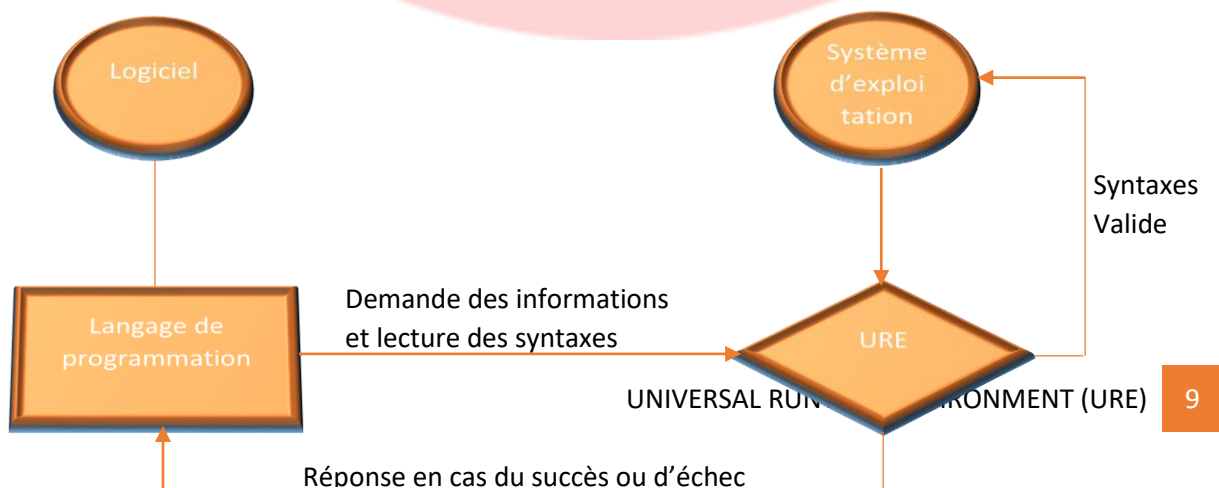
Ce qui donne **4 Personnages** fictifs de l'histoire ou des procédures affectées dans un ordinateur :

- ✓ **Le système d'exploitation** : La voisine bilingue ;
- ✓ **Logiciel** : La maman voisine lingalaphone ;
- ✓ **Environnement d'exécution** : L'enfant de la voisine bilingue ;
- ✓ **Le langage de programmation** : L'enfant de la maman voisine lingalaphone.

Maintenant vous allez comprendre qu'est ce qui se passe dans un **ordinateur, téléphone mobile** ou **autres appareils** qu'utilisent les programmes informatiques. La maman voisine lingalaphone envoie son enfant aisément en lingala afin d'aller récupérer l'argent. L'enfant (Langage de programmation) arrive à la destination et frappe la porte, l'autre enfant (Environnement d'exécution) précipite pour regarder s'il s'agit de qui pour écouter et comprendre ce que cet enfant va lui demander. Une fois la demande est lisible avec des mots exacts (Syntaxes), il transmet la même demande chez sa mère (Système d'exploitation) afin de vérifier si réellement elle a l'argent. En cas du succès ou au cas contraire que sa mère n'a pas l'argent, elle renvoie la réponse ou l'argent à son enfant et l'enfant à son tour transmet la réponse reçue ou l'argent reçu vers l'enfant qui était en attente devant la porte. En outre si les mots ou syntaxes de la demande n'ont pas été correctement défini(e), l'environnement d'exécution universel (**L'enfant bilingue**) détecte et renvoie les erreurs au langage de programmation en attente de la réponse (**L'enfant en attente**) afin de le transmettre au logiciel ou application (**Sa mère**).

**NB** : Avec cet exemple, il nous montre que la maman voisine lingalaphone et son enfant n'ont plus besoin de connaître ou d'apprendre obligatoirement la langue exacte qu'utilisent leurs voisins afin de l'adapter et l'appliquer dans leur maison.

### 3.4.1. Présentation de l'exemple sous forme graphique (Ordinogramme) :



## 4. Machine virtuelle

### 4.1. Introduction :





En informatique, une **machine virtuelle** (VM) est une illusion d'un appareil informatique créée par un logiciel d'émulation ou instanciée sur un hyperviseur. Le logiciel d'émulation simule la présence de ressources matérielles et logicielles telles que la **mémoire**, le **processeur**, le **disque dur**, voire le **système d'exploitation** et les **pilotes**, permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine simulée.

Un des intérêts des machines virtuelles est de pouvoir s'abstraire des caractéristiques de la machine physique utilisée, permettant une forte portabilité des logiciels et la gestion de systèmes hérités étant parfois conçus pour des machines ou des environnements logiciels anciens.

Les machines virtuelles sont également utilisées pour isoler des applications pour des raisons de sécurité, pour augmenter la robustesse d'un serveur en limitant l'impact des erreurs système ou pour émuler plusieurs machines sur une seule machine physique (**La virtualisation**).

### 4.2. Virtualisation :

Au sens large, la virtualisation consiste à simuler l'existence de plusieurs machines informatiques en utilisant une seule machine. Celle-ci permette également de gagner du temps : En parlant de l'environnement d'exécution universel (URE), si un langage de programmation est adapté à cet environnement, les programmeurs pourraient développer leurs applications informatiques utilisant un seul fichier exécutable avec la langue compréhensive de son pays et ce fichier exécutable s'exécutera dans toutes les plateformes que l'environnement d'exécution universel à était installé sans pour autant modifier les codes sources ou comprendre les comportements du système d'exploitation invité à exécuter son application. Pour une meilleure compréhension à ce point, si l'environnement d'exécution universel est installé dans plus de **4 systèmes d'exploitation** qu'on peut citer :

Windows  , Linux  , Mac OS  et Android  ou autres, les développeurs experts de langages des programmations seraient aussi obligés de développer leurs langages de programmation en 4 différentes bibliothèques compatibles pour 4 systèmes d'exploitation ciblés. Et pour les programmeurs informatiques des applications exécutables (Les étudiants) à qui utiliseraient ce nouveau langage de programmation cross plate-forme et multilingues, ils seront censés d'utiliser qu'un seul fichier pour qu'il soit vérifié et interpréter depuis l'environnement d'exécution universel, par la suite s'exécuter via les 4

différentes bibliothèques standards (Dynamiques) de ce nouveau langage approprié.

## 5. Bibliothèque standard

La **bibliothèque standard**, est la bibliothèque logicielle utilisée dans toute implémentation des langages de programmation utilisés dans un environnement d'exécution universel.

Une bibliothèque standard peut inclure :

- ✓ Des fonctions
- ✓ Des macros
- ✓ Variables globales
- ✓ Des classes

La plupart des bibliothèques standard incluent :

- ✓ Des algorithmes (Par exemple pour le **tri**) ;
- ✓ Des structures de données (Telles que les **listes**, les **arbres** et les **tables de hachage**) ;
- ✓ Des routines d'entrées-sorties et d'appel système.

En informatique, une **bibliothèque logicielle** est une collection des routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. Les bibliothèques sont enregistrées dans des fichiers semblables, voire identiques aux fichiers de programmes, sous la forme d'une collection de fichiers de code objet rassemblés accompagnée d'un index permettant de retrouver facilement chaque routine. Le mot « **librairie** » est souvent utilisé à tort pour désigner une bibliothèque logicielle.

Les bibliothèques sont devenues un sujet incontournable de programmation. Elles sont utilisées pour réaliser des interfaces de programmation, des **framework**, des **plugins** ainsi que des **langages de programmation**. Les routines contenues dans les bibliothèques sont typiquement en rapport avec des

opérations fréquentes en programmation : manipulation des interfaces utilisateur, manipulation des **bases de données** ou **calculs mathématiques**.

## 6. Débogueur

Un débogueur (de l'anglais **debugger**) est un logiciel qui aide un développeur à analyser les bugs d'un programme. Pour cela, il permet d'exécuter le programme ligne par ligne. C'est-à-dire, d'afficher la valeur des variables à tout moment et de mettre en place des points d'arrêt sur des conditions ou sur des lignes du programme.

Le programme à déboguer est exécuté à travers le débogueur et s'exécute normalement. Le débogueur offre alors au programmeur la possibilité d'observer et de contrôler l'exécution du programme, en lui permettant par divers moyens de **l'observer**, de le **stopper** (Arrêter l'exécution du programme) et de le **changer**. Par exemple, la **pile** d'exécution et le contenu des variables en mémoire peuvent être observés, et la valeur des variables peut être changée pour altérer le **flot de contrôle** du programme afin, par exemple, de déterminer la cause d'une défaillance.

Quand l'exécution d'un programme est stoppée, le débogueur affiche la position courante d'exécution dans le code source original pour le débogage formel et le débogage au niveau source. Le débogueur de bas niveau ou de niveau machine montre la ligne désassemblée de défaillance.

L'environnement d'exécution universel quant à lui, facilite les développeurs et les fassent gagner du temps en leur offrant un débogueur **orienté, universel, compatible** pour toutes les langues **existantes** ou **inexistantes** dans le monde et **personnalisable** par les programmeurs des applications informatiques. Un débogueur standard et adapté à son utilisateur. Capable d'afficher les erreurs compréhensibles détectées à la langue utilisée par le système d'exploitation ou personnalisée par l'utilisateur de ce système d'exploitation (Windows, Linux, Mac OS et Android).



## 6.1. Outils pour stopper l'exécution du programme :

- ✓ **Arrêt automatique** : le débogueur arrête l'exécution automatiquement du programme en cours en cas d'erreur détectée durant l'interprétation des syntaxes structurées.
- ✓ **Point d'arrêt** : Se place à un endroit spécifique du programme (**généralement dans le code source**), il indique à l'utilisateur ou le programmeur la ligne et le lexique ou la syntaxe qui cause problème.
- ✓ **Point d'arrêt conditionné** : Comme le point d'arrêt, mais avec une condition supplémentaire. Si depuis la ligne des codes d'un fichier exécutable détecté qu'il existe une erreur qui contienne l'abréviation entre parenthèses (**CURE**) signifiant Command Universal Runtime Environment (L'affichage de la commande d'environnement d'exécution universel) et que les codes sources de ce fichier n'ont pas été **compilés** ou **cryptés** par le programmeur de ce fichier exécutable, le débogueur d'environnement affichera le **vrai lexique** ou la **vraie syntaxe** à utiliser par la personne à qui utilise le programme en cours d'arrestation. Par contre si le fichier exécutable qu'il a détecté une erreur n'a pas cette abréviation et qu'il n'a pas été compilé ou chiffré par son programmeur, il affiche directement la ligne de l'erreur et le **lexique** ou **syntaxe** qui cause problème provoquant cette arrestation inattendue.
- **Point d'arrêt sémantique** : Ici le débogueur d'environnement d'exécution universel (DEEU) n'a pas d'effet relatif à cette arrestation. L'environnement laisse à la portée des développeurs le choix de déterminer les syntaxes correctes envoyées depuis l'environnement d'exécution universel, afin d'être vérifié et exécuter directement au système d'exploitation utilisé. C'est-à-dire cette tâche est censée être gérée par le développeur propriétaire du langage de programmation utilisé par le programmeur informaticien.

## 6.2. Outils pour piloter l'exécution du programme :

- ✓ **Vérification de la bibliothèque standard existante pour chaque instruction valide** : L'environnement d'exécution universel valide l'instruction et vérifie avant son exécution si réellement la bibliothèque spécifiée dans la syntaxe demandée existe depuis le système en cours d'utilisation. En outre, si la bibliothèque n'existe pas ou n'a pas été installée depuis le système d'exploitation utilisé, le débogueur d'environnement renvoie la réponse d'avertissement afin que l'utilisateur ou le programmeur trouve la solution sur ce dernier.



## 7. Langage de programmation

Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle composé d'un **alphabet**, d'un **vocabulaire**, de règles de **grammaire**, de **significations**, mais aussi d'un **environnement de traduction** censé de rendre sa syntaxe compréhensible par la machine.

Les langages de programmation permettent de décrire d'une part, les structures de données qui seront manipulées par l'appareil informatique et d'autre part, d'indiquer comment sont effectuées les manipulations selon l'algorithme. Ils servent les moyens de communication par lesquels les programmeurs communiquent avec les ordinateurs et avec des robots. Comme nous l'avion présenté sur le point **3.4 de l'environnement d'exécution universel** (L'enfant de la maman lingalaphone), il est là pour effectuer une demande à un environnement d'exécution afin d'obtenir le résultat en cas du succès ou au cas contraire.

Un langage de programmation est mis en œuvre par un traducteur automatique : **Compilateur** ou **Interpréteur**. Un compilateur est un programme informatique qui transforme dans un premier temps un code source écrit dans un langage de programmation donné en un code cible qui pourra être directement exécuté par un ordinateur, à savoir un programme en **langage machine** ou en **code intermédiaire**. Tandis que l'interpréteur réalise cette traduction « à la volée » tout comme l'environnement d'exécution universel (EEU), est un interpréteur adapté pour toutes les langues dans le monde. Le **EEU** ou **URE** n'a pas un langage de programmation spécifique afin d'interpréter les programmes ou d'autres langages de programmation à sa possession. Les développeurs sont libre d'indiquer à l'environnement d'exécution universel sur quel langage de programmation que son débogueur va s'adapté et corriger les programmeurs consommateurs ou utilisateurs.

Un langage de programmation offre un ensemble de notions qui peuvent être utilisées comme primitives pour développer des algorithmes. Les programmeurs apprécient que le langage soit **clair**, **simple** et **unifié**, qu'il y ait un minimum de notions qui peuvent être combinées selon des règles **simples** et **régulières**.

Les qualités d'un langage de programmation influent sur la facilité avec laquelle les programmes pourront être **écrits**, **testés**, puis plus tard **compris** et **modifié**.

## 7.1. Les règles lexicales :

Définies chaque mot ou commande utilisé(e) dans une syntaxe afin de vérifier les mots structurés dans une syntaxe avant d'être analysé syntaxiquement.

### 7.1.1. L'alphabet

L'alphabet des langages de programmation est basé sur les normes courantes comme **ASCII**, qui comporte les lettres de **A** à **Z** sans accent, des **Chiffres** et des **Symboles**, ou **Unicode** pour la plupart des langages modernes (dans lesquels l'utilisation se limite en général aux chaînes de caractères littérales et aux commentaires, avec quelques exceptions notables).

## 7.2. Les règles de syntaxe :

Définies par une grammaire formelle, elles régissent les différentes manières dont les éléments du langage peuvent être combinés pour obtenir des programmes. La ponctuation (par exemple l'apposition d'un symbole ; en fin de ligne d'instruction d'un programme) relève de la syntaxe.

### 7.2.1. Le vocabulaire

Parmi les éléments du langage, le vocabulaire représente l'ensemble des instructions construites d'après les **symboles**. L'instruction peut être **mnémotechnique** ou uniquement **symbolique** comme quand elle est représentée par des **symboles d'opérations** tels que les opérateurs arithmétiques (« + » et « - ») ou **booléens** (&& pour le et logique par exemple). On parle aussi parfois de *mot clé* pour désigner une instruction.

### 7.2.2. Les commentaires

Les commentaires sont des textes qui ne seront pas traduits. Ils peuvent être ajoutés dans les programmes pour y laisser des explications. Les commentaires sont délimités par des marques qui diffèrent d'un langage de programmation à l'autre. Depuis l'environnement d'exécution universel, un langage de programmation peut être commenté en utilisant les symboles **##**. Grâce à ces symboles, le débogueur de l'environnement d'exécution peut l'utiliser pour sauter les lignes utilisant les deux symboles des commentaires.

### 7.3. La sémantique :

Les règles sémantiques définissent le sens de chacune des phrases qui peuvent être construites dans le langage, en particulier quels seront les effets de la phrase lors de l'exécution du programme.

## 8. Compilateur

En informatique, un **compilateur** est un programme qui transforme un **code source** en un **code objet**. Généralement, le code source est écrit dans un **langage de programmation** (le langage source), il est de haut niveau d'abstraction, et facilement compréhensible par l'humain. Le code objet est généralement écrit en langage de plus bas niveau (appelé langage cible), par exemple un **langage d'assemblage** (ASM) ou **langage machine**, afin de créer un programme exécutable par une machine.

Un compilateur effectue les opérations suivantes : **analyse lexicale**, prétraitement (préprocesseur), **analyse syntaxique** (parsing), **analyse sémantique**, et **génération de code** optimisé. La compilation est souvent suivie d'une étape d'édition des liens, pour générer un fichier exécutable. Quand le programme compilé (code objet) est exécuté sur un ordinateur dont le processeur ou le système d'exploitation est différent de celui du compilateur, on parle de compilation croisée.

Les logiciels des premiers ordinateurs étaient écrits en langage assembleur (ASM). Les langages de programmation de plus haut niveau (dans les couches d'abstraction) n'ont été inventés que lorsque les avantages apportés par la possibilité de réutiliser le logiciel sur différents types de processeurs sont devenus plus importants que le coût de l'écriture d'un compilateur. La capacité de mémoire très limitée des premiers ordinateurs a également posé plusieurs problèmes techniques dans le développement des compilateurs.

## 9. Assembleur

Un **langage d'assemblage** ou **langage assembleur** est, en programmation informatique, le langage de plus bas niveau qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des symboles dits « **mnémoniques** », c'est-à-dire faciles à retenir. Le programme assembleur convertit ces mnémoniques en langage machine, ainsi que les valeurs (écrites en décimal) en binaire et les libellés d'emplacements en adresses, en vue de créer par exemple un fichier objet ou un fichier exécutable.

Dans la pratique courante, le même terme assembleur est utilisé à la fois pour désigner le langage d'assemblage et le programme assembleur qui le traduit. On parle ainsi de « **programmation en assembleur** ».

La traduction une fois pour toutes par beaucoup d'interpréteurs de chaque nom de variable rencontré dans une instruction (évolué) par la position mémoire associée et de chaque constante (écrite par l'utilisateur en décimal) en binaire est typique d'une opération d'assemblage, bien que le nom d'assembleur ne soit pas couramment utilisé dans ce cas précis.

Les langages d'assemblages ont éliminé une grande partie des erreurs commises par les programmeurs de la première génération d'ordinateurs, en les dispensant de mémoriser les codes numériques des instructions et de faire des calculs d'adresses. La programmation en assembleur était alors utilisée pour écrire toutes sortes de programmes.

Dans les années 1970-1980, l'utilisation de l'assembleur pour écrire des applications a été très largement supplantée par l'emploi de langages de programmation de haut niveau : **Fortran**, **COBOL**, **PL/I**, etc...

La puissance des machines le permettait et consacrer quelques minutes de temps machine à une compilation pour économiser quelques heures de temps de programmeur était une opération rentable, même si les compilateurs de l'époque fournissaient un code moins performant (plus volumineux et souvent plus lent). Par ailleurs, ces langages de haut niveau permettaient de s'affranchir de la dépendance à une architecture matérielle unique. De plus, le code est nettement plus facile à lire, même pour son propre auteur. Et réduit le risque d'un bug.

## 10. Interface de programmation d'application (IPA)

Une **interface de programmation d'applications** ou **interface de programmation applicative** (souvent désignée par le terme **API** pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.

De manière plus générale, on parle d'API à partir du moment où une entité informatique cherche à agir avec ou sur un système tiers, et que cette interaction se fait de manière normalisée en respectant les contraintes d'accès définies par le système tiers. On dit que le système tiers « expose une API. » À ce titre, des choses aussi diverses que la signature d'une fonction, une **URL**, un **RPC**... sont parfois considérés comme des API (ou micro-API) à part entière.

Dans l'industrie contemporaine du logiciel, les applications informatiques se servent de nombreuses interfaces de programmation, car la programmation se fait en réutilisant des briques de fonctionnalités fournies par des logiciels tiers. Cette construction par assemblage nécessite pour le programmeur de connaître la manière d'interagir avec les autres logiciels, qui dépend de leur interface de programmation. Le programmeur n'a pas besoin de connaître les détails de la logique interne du logiciel tiers, et celle-ci n'est pas nécessairement documentée par le fournisseur. Seule l'API est réellement nécessaire pour utiliser le système tiers en question.

Des logiciels tels que les **systèmes d'exploitation**, les **systèmes de gestion de base de données**, les **langages de programmation**, ou les **serveurs d'applications** comportent une ou plusieurs interface(s) de programmation. Alors pour que l'environnement d'exécution universel soit adapté dans tous les systèmes y compris les langages de programmation nouveaux ou existants, nous vous proposerons quelques lignes des codes à utiliser à l'étape suivante pour l'adaptation et utilisation d'un langage de programmation à un environnement d'exécution universel de toutes les langues.



## 11. Tableau symbolique

Avant de commencer à comprendre le comportement physique interne de notre environnement d'exécution universel, voici un tableau symbolique à comprendre en premier lieu afin de s'adapter correctement aux étapes qui suivent :

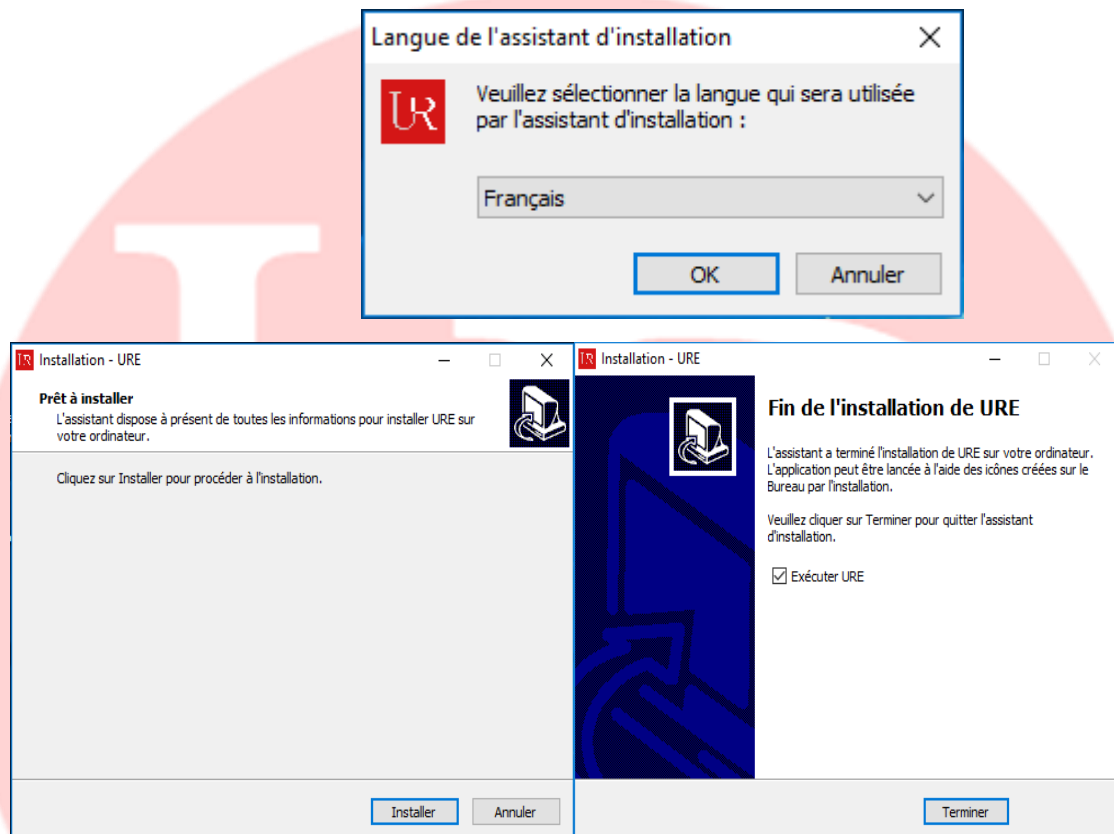
SYMBOLES	ROLES
<code>:s:</code>	Est appelé depuis la console (terminal) afin de déterminer le langage sur lequel vous allez utiliser pour votre application.
<code>##</code>	Utilisé pour commenter un bloc des syntaxes afin de le retrouver facilement.
<code>(CURE)</code>	Permet d'activer les détails des erreurs détectées afin d'afficher les vraies syntaxes cachées. Cette option ou symbole est déconseillée pour ceux qui veulent cacher et sécuriser leurs syntaxes. Par défaut elle est désactivée.
<code>-e</code>	A utiliser depuis le terminal afin de compiler et sécuriser vos syntaxes.
<code>-l</code>	Sert à exécuter URE en mode console suivi d'un fichier des syntaxes cible.
<code>-h</code>	Pour afficher de l'aide concernant URE.
<code>`?'` ou `?'` ou encore `"?`</code>	Effectue la lecture des valeurs qui sont entre guillemet.



## 12. Installation URE

L'installation de l'environnement d'exécution universel est très simple :

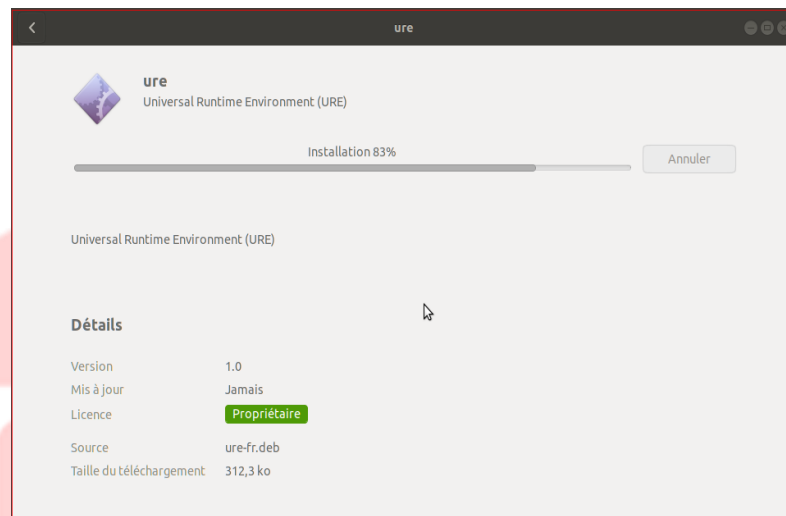
- ✓ **Sous Windows** : Télécharger la version exécutable depuis l'adresse [https://sourceforge.net/projects/ure/files/Windows/URE\\_x64.exe/download](https://sourceforge.net/projects/ure/files/Windows/URE_x64.exe/download) et lancer l'installation comme suite :



- ✓ **Sous Linux** : Depuis les distributions linux, télécharger l'archive à l'adresse <https://sourceforge.net/projects/ure/files/Linux/ure-linuxx64-1.0.tar.xz/download> et procédez comme suite pour installer l'environnement d'exécution universel. Pour procéder, depuis votre terminal exécutez les commandes suivantes :

```
cd /download/dir/  
tar -xvf ure-linuxx64-1.0.tar.xz  
cd ure-1.0/  
./install.sh
```

Vous pouvez aussi télécharger la version du **.deb** depuis <https://sourceforge.net/projects/ure/files/Linux/ure-fr.deb/download> et double cliquez directement afin de lancer l'installation.



- ✓ **PHP** : L'installation de l'environnement d'exécution universel en tant qu'extension PHP est très simple. Choisissez la bibliothèque concernant votre système d'exploitation (**ure.so** ou **ure.dll**) et depuis votre terminal exécuter les commandes ci-après :
  - Premièrement créer un répertoire URE et copier les deux fichiers de configuration en utilisant les codes ci-après depuis votre terminal :

```
mkdir -p /usr/share/ure
cp ./lang.conf /usr/share/ure/
cp ./langGUI.conf /usr/share/ure/
```
  - Maintenant vous devez afficher le chemin relatif aux extensions PHP afin de copier la bibliothèque URE à ce répertoire. Dans notre exemple, le chemin relatif aux extensions PHP est la version 8.0 :

```
php -i | grep extension_dir
cp ./ure.so /usr/lib/php/20200930/
```
  - Afficher le chemin du fichier php.ini et ajoutez-y la ligne de l'extension URE :

```
php -i | grep 'Configuration File'
nano /etc/php/8.0/cli/php.ini
extension=ure.so
;extension=ure.dll
```

- Par la suite, redémarrer votre serveur Apache2 :  
`sudo service apache2 restart`

✓ **Sous Mac** : Cette version est encore en phase des tests, bientôt la version exécutable sera disponible.

**Sous DOS** : Pour d'autres systèmes que nous n'avons pas cités, vous aurez une bibliothèque cross plates-formes au même moment que **Mac** afin de l'adapter à votre système pour faciliter vos utilisateurs.

## 13. Configuration URE

Nous venons de voir comment nous avons procédé à l'installation de l'environnement d'exécution universel. Maintenant nous allons devoir configurer notre environnement afin qu'il exécute les applications que nous allons programmer à notre langage formel et de renvoyer les résultats compréhensifs à notre langue.

Une fois l'environnement d'exécution universel (EEU ou URE) installé :

- ✓ **Sous Windows** : Depuis le répertoire **C:\URE** vous verrez deux fichiers (**lang.conf** et **langGUI.conf**) à configurer selon votre langage compréhensif.
  - **lang.conf** : Les informations qui seront modifiées dans ce fichier de configuration seront aussi exécutées ou appelées depuis votre terminal du système. C'est-à-dire, lorsque l'environnement d'exécution universel détecte une erreur au débogage de vos syntaxes, il va appeler une ligne concernant cette erreur depuis le dit fichier de configuration afin de l'afficher correctement à une meilleure compréhension pour te permettre de corriger les erreurs détectées.
  - **langGUI.conf** : Toutes les informations relatives au débogage qui se trouvent dans le fichier précédent **lang.conf**, se trouvent aussi dans ce fichier. La différence est que ses

informations seront affichées ou appelées depuis l'interface graphique de l'application sur lequel l'environnement d'exécution universel a pu détecter l'erreur. Par contre le premier fichier de configuration est utilisé depuis la console (Terminal).

**NB** : Pour modifier les informations relatives à ces deux fichiers de configurations, vous devez juste double cliquer afin de l'éditer depuis n'importe quel éditeur de texte (Notepad++, Bloc-notes etc...).

✓ **Linux** : Avec les distributions linux, les procédures sont similaires. Mais vous pouvez aussi depuis votre terminal, exécuter les commandes suivantes :

- **sudo nano /usr/share/ure/lang.conf** : Pour la configuration des informations de débogage dans un terminal ;
- **sudo nano /usr/share/ure/langGUI.conf** : Pour configurer les informations de débogage des applications utilisant l'interface graphique utilisateur (IGU ou GUI).

a) **Exemple** :

Modifions une ligne dans le fichier **lang.conf** français pour l'utiliser en **anglais** par exemple :

FRANCAIS	ANGLAIS
:0: ->:B1:ERREUR 1045 (:E1:): Aucune information définie !:A2::EE:	:0: ->:B1:ERROR 1045 (:E1:): No information found !:A2::EE:

Désormais, une fois que l'environnement d'exécution universel aura besoin d'afficher un message en provenance de cette ligne, s'affichera uniquement en anglais parce que vous l'avez modifié en utilisant votre langue compréhensive l'anglais.

**NB** : Nous avons pu laisser le fichier de débogage à la portée de tout le monde afin que chacun arrive à le traduire correctement selon la langue compréhensive de son choix.

b) **Symboles** :

A noter que, les symboles que vous allez trouver dans chaque ligne aux fichiers de configurations sont tous significatifs.

- **:0:** : S'occupe de la première ligne dans le fichier afin de permettre à l'environnement d'exécution universel de bien trouver l'information à afficher. Vous verrez que la ligne suivante est occupée de **:1:** et ainsi de suite.
- **:A1:** : Permet d'afficher le message obtenu en vert.
- **:B1:** : Affiche le message obtenu en rouge.
- **:A2:** : Fermeture de la couleur choisie (Verte ou Rouge).
- **:EE:** : Effectue un saut de ligne à la fin du message affiché pour une meilleure compréhension de la ligne détectée.
- **:E1:** : Place la première information détectée depuis la syntaxe vers un message sélectionné depuis le fichier de configuration.
- **:E2:** : Place la deuxième information détectée depuis la syntaxe d'erreur vers un message sélectionné en provenance du fichier de configuration.
- **:E3:** : Place la troisième information détectée depuis la syntaxe d'erreur vers un message sélectionné du fichier de configuration.

c) **Conclusion** :

Notre environnement d'exécution universel à été installé et configuré correctement afin qu'il soit compréhensif à la programmation et aux débogages des applications en cas des erreurs détectées. Maintenant nous allons procéder à la conception de notre propre langage de programmation.

## 14. Programmation (Concepteur langage de programmation)

Il est maintenant grand temps d'apprendre comment adapter un langage de programmation existant ou créer un, pour qu'il soit exécutable ou utilisable depuis l'environnement d'exécution universel. Passons maintenant aux choses sérieuses :

Premièrement, n'oubliez pas de créer un répertoire de votre langage de programmation via `/usr/share/test`. Le **test** est le nom que nous avons pu donner à notre nouveau langage de programmation d'exemple. Vous pouvez changer comme bon vous semble.

Pour procéder, depuis votre terminal tapez la commande :

- ✓ Sous Windows :  
`mkdir -p C:/test`
- ✓ Sous Linux :  
`mkdir -p /usr/share/test`

C'est avec ce répertoire **test** créé que l'environnement d'exécution universel utilisera afin de trouver et récupérer une ligne dans le fichier **lang.conf** de votre nouveau langage de programmation.

Pourquoi l'environnement d'exécution universel utilise déjà un fichier **lang.conf** et notre nouveau langage aussi à besoin de son propre fichier **lang.conf** ?

La différence est que : Le fichier **lang.conf** de l'environnement d'exécution universel est appelé afin de sélectionner une ligne de réponse en cas d'erreur depuis la syntaxe analysée par l'analyse **lexicale** et **syntactique** tant que l'URE effectue que deux analyses. Tandis que le fichier **lang.conf** de notre langage de programmation sera utile pour être appelé et afficher un message sélectionné d'analyse **sémantique**.

Dans notre exemple de langage de programmation que nous allons créer, utilisera deux bibliothèques **test\_exit** et **test\_print** parce que l'environnement d'exécution universel requiert des bibliothèques pour l'exécution des applications multilingues.



- **test\_exit** : Cette bibliothèque va permettre aux concepteurs des applications de définir un mot ou une syntaxe qui sera capable de quitter ou arrêter l'exécution de l'application.
- **test\_print** : C'est la deuxième bibliothèque que nous allons créer qui permettra aux programmeurs des applications de l'utiliser pour afficher un message dans l'application.

Comme vous le savez très bien, la conception d'un langage de programmation requise une expérience d'un langage de programmation cible. Vous devez avoir la maîtrise d'un langage de programmation parmi ces trois langages **ASM**, **C** ou **C++**. Dans cet exemple, nous allons utiliser le langage de programmation **C++** comme langage cible de notre nouveau langage de programmation **test**.

a) Exercice :

Créer un fichier header appelé **test.h** et ajoutez-y les codes ci-dessous :

```
#ifndef TEST_H
#define TEST_H

#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>

using namespace std;

class TEST{
public:virtual string CFExec(string request, string goweb, int
goresponse)=0;
};

#endif
```

**NB** : C'est grâce à ce fichier d'entête que l'environnement d'exécution universel utilisera afin de se communiquer avec votre nouveau langage de programmation.

Créer par la suite deux fichiers **test\_exit.cpp** et **test\_print.cpp** dans le répertoire de votre projet. Ouvrez le premier fichier crée **test\_exit.cpp**, et ajoutez les codes ci-après :

```
#include "test.h"
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>

using namespace std;

class TESTLib : public TEST {
public:string CFExec(string request, string goWeb, int goresponse){
string resultat="quitter";

return resultat;
}
};

extern "C" TESTLib* create(){
return new TESTLib;
}

extern "C" void destroy(TESTLib* TI){
delete TI;
}
```

Et dans le deuxième fichier `test_print.cpp`, ajoutez les codes ci-après :

```
#include "test.h"
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>
#include <fstream>

#include <boost/algorithm/string.hpp>
#include <boost/lexical_cast.hpp>

using namespace std;

string lineLang="";
string file="";

//Début Replace
string goReplace(string data, string from, string to){
    size_t ReplaceDetect=data.find(from);
    if(ReplaceDetect!=string::npos){
        size_t start_pos = 0;
        while((start_pos = data.find(from, start_pos)) != std::string::npos) {
            data.replace(start_pos, from.length(), to);
            start_pos += to.length(); // Handles case where 'to' is a substring of 'from'
        }
    }
    return data;
}
//Fin Replace

//Début Response LANG
string goResponseLANG(string myGetPosition){
    string myLang="";
    ifstream goLoadUserLang("/usr/share/test/lang.conf", ios::binary);
    if (goLoadUserLang.is_open()){
        while(getline(goLoadUserLang, lineLang)){
            if(lineLang==" | | lineLang=" "-;")}{else{
                size_t DetectLANG=lineLang.find(": " + myGetPosition + ":");
                if(DetectLANG!=string::npos){
                    lineLang=goReplace(lineLang, "EE:", "\n");
                    lineLang=goReplace(lineLang, ":" + myGetPosition + ":", "");
                }
            }
        }
    }
}
```

```

lineLang=goReplace(lineLang, ":A1:", "\x1B[32m");
lineLang=goReplace(lineLang, ":A2:", "\033[0m");
lineLang=goReplace(lineLang, ":B1:", "\x1B[31m");
myLang=lineLang;
goLoadUserLang.close();
}}}
goLoadUserLang.close();
}else{
myLang="    ->Lecture des réponses impossible (LANG).\n    ->";
}

return myLang;
}
//Fin response LANG

class TESTLib : public TEST {
public:string CFExec(string request, string goWeb, int goresponse){
string resultat="";
string GetedScriptFile="";
string GetedSyntaxes="";
string GetedData="";

size_t DetectPV=request.find(";");
if(DetectPV!=string::npos){
vector<string> dataToPV;
boost::split(dataToPV, request, boost::is_any_of(";"));
GetedScriptFile=dataToPV[0];
GetedSyntaxes=dataToPV[1];

//Begin to verify syntaxes
size_t DetectEQ=request.find("=");
if(DetectEQ!=string::npos){
vector<string> dataToEQ;
boost::split(dataToEQ, GetedSyntaxes, boost::is_any_of("="));
GetedData=dataToEQ[1];
GetedData=goReplace(GetedData, "`", "");
GetedData=goReplace(GetedData, "\"", "");
resultat=GetedData;
}else{
resultat=goResponseLANG("1");
resultat=goReplace(resultat, ":E1:", "" + GetedSyntaxes + "");
}
}
}

```

```

resultat=goReplace(resultat, ":E2:", "" + GetedScriptFile + "");
}
//Ended to verify syntaxes
}else{
resultat=goResponseLANG("0");
}
return resultat;
}
};

```

```

extern "C" TESTLib* create(){
return new TESTLib;
}
extern "C" void destroy(TESTLib* TI){
delete TI;
}

```

Créons maintenant notre fichier **lang.conf** pour permettre à notre langage de programmation de l'utiliser en cas d'analyse **sémantique** dans le répertoire que nous avons créé (/usr/share/test). Une fois le fichier crée, ajoutez-y les codes suivants :

```

:0:  ->:B1:Désolé, erreur fatale de la résolution.:A2::EE:
:1:  ->Désolé, syntaxes ::EE::B1::E1::A2::EE:Non valide depuis ::A1::E2::A2::EE:

```

**NB** : Vous allez voir à l'exécution de nos bibliothèques, une fois l'erreur **sémantique** détectée, les **:E1:** et **:E2:** vont être remplacé par le nom des objets détectés en provenance de **URE**.

#### b) Compilation :

Nous venons de finaliser à créer nos **4 fichiers** important qui nous permettrons d'adapter notre nouveau langage de programmation à l'environnement d'exécution universel pour qu'il soit utilisé à n'importe quel langage dans le monde. Maintenant, compilons les deux fichiers **tes\_exit.cpp** et **test\_print.cpp** à deux bibliothèques dynamiques selon l'environnement du système d'exploitation que vous utilisez.

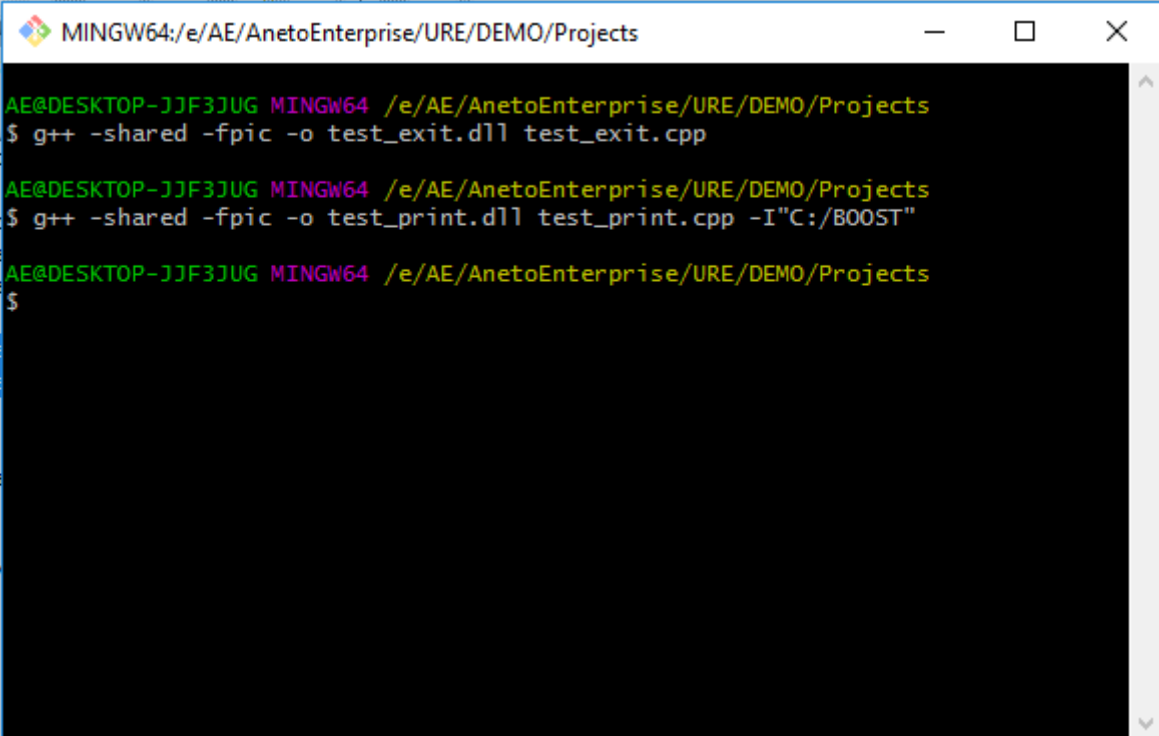
Placez-vous dans le répertoire de votre projet et compiler vos deux bibliothèques comme suite :

✓ Sous Windows :

`cd /Project/Test/`

`g++ -shared -fpic -o test_exit.dll test_exit.cpp`

`g++ -shared -fpic -o test_print.dll test_print.cpp -I"C:/BOOST"`

A screenshot of a MINGW64 terminal window. The title bar shows the path "MINGW64:/e/AE/AnetoEnterprise/URE/DEMO/Projects". The terminal content shows three lines of commands and their prompts: 1. Prompt: "AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects", Command: "\$ g++ -shared -fpic -o test\_exit.dll test\_exit.cpp". 2. Prompt: "AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects", Command: "\$ g++ -shared -fpic -o test\_print.dll test\_print.cpp -I\"C:/BOOST\"". 3. Prompt: "AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects", Command: "\$".

```
MINGW64:/e/AE/AnetoEnterprise/URE/DEMO/Projects
AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects
$ g++ -shared -fpic -o test_exit.dll test_exit.cpp
AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects
$ g++ -shared -fpic -o test_print.dll test_print.cpp -I"C:/BOOST"
AE@DESKTOP-JJF3JUG MINGW64 /e/AE/AnetoEnterprise/URE/DEMO/Projects
$
```

✓ Sous Linux :

`cd /Project/Test/`

`g++ -shared -fpic -o test_exit.so test_exit.cpp`



`g++ -shared -fpic -o test_print.so test_print.cpp`

### c) Intégration :


Maintenant que nous avons nos deux bibliothèques compilées correctement, nous allons procéder à l'intégration de nos 3 fichiers **lang.conf**, **test\_exit.so** et **test\_print.so** via l'environnement d'exécution universel (URE) pour qu'ils soient utilisés par d'autres programmeurs des applications.



- ✓ **Sous Windows** : Copier vos deux bibliothèques que vous venez de créés et allez dans le répertoire **C:/URE/lib** faites coller.

 test_exit.dll	07/01/2022 12:04	Extension de l'app...	33 Ko
 test_print.dll	07/01/2022 12:04	Extension de l'app...	294 Ko

Maintenant, allez dans le répertoire de votre langage de programmation que vous ayez créé depuis le disque **C:/test** sous **Windows** ou **/usr/share/test** sous **Linux** puis coller le fichier **lang.conf** de votre langage de programmation.

PC > Disque local (C:) > test			
Nom	Modifié le	Type	Taille
 lang	06/01/2022 11:52	Fichier CONF	1 Ko

- ✓ **Sous Linux** : Copiez les deux bibliothèques et le fichier **lang.conf** depuis votre terminal vers les répertoires spécifiques en utilisant les commandes suivantes :

```
cd /Project/Test/  
cp ./*.so /usr/lib/  
cp -r ./lang.conf /usr/share/test/
```

#### d) Syntaxes :

Toute est ok, maintenant nous allons passer au développement ou définition des syntaxes pour notre nouveau langage de programmation afin que d'autres programmeurs arrivent à bien l'utiliser grâce à leurs langages formels.

Créer un fichier **test\_syntaxes.txt** et dans ce fichier, ajoutez les syntaxes suivantes :

```
quitter (test_exit);  
afficher= `?` (test_print);
```

**NB** : Nous pouvons voir que nos syntaxes sont belles et bien en français afin de permettre à n'importe quel programmeur de les traduire correctement à la langue de son choix. Les deux commandes entre parenthèses sur les deux lignes sont nos deux bibliothèques que nous venons de compiler et placées dans le répertoire **C:/URE/lib**. C'est-à-dire, la première syntaxe appelle la bibliothèque **test\_exit** afin de permettre à une application d'arrêter son exécution et la deuxième ligne de syntaxe appelle la bibliothèque **test\_print** pour afficher le message obtenu en provenance de l'utilisateur ou programmeur de l'application. Vous serez libre de le traduire à n'importe quelle langue dans le monde et l'environnement d'exécution universel (URE) le comprendra et l'exécutera correctement pour vous.

Table représentatif des traductions par pays utilisateur :

FRANÇAIS (FRANCE)	ANGLAIS (USA)	LINGALA (RDC)
quitter (test_exit);	exit (test_exit);	bima (test_exit);
afficher = `?` (test_print);	show = `?` (test_print);	lakisa = `?` (test_print);

**NB** : Chaque programmeur dans chaque pays sera libre de traduire les syntaxes de votre langage de programmation afin qu'il soit compréhensif à sa manière. Il ne sera quasiment pas obligé d'apprendre à comprendre les syntaxes, sémantique ou lexiques utilisées. C'est la meilleure manière de faire avancer la technologie rapidement partout dans le monde.

Par la suite, copier le fichier **test\_syntaxes.txt** ou **test\_syntaxes.tt** si vous l'avez compilé et collez-le vers le répertoire **C:/URE/int** sous **Windows** ou vers le répertoire **/usr/share/ure/int** sous **Linux** afin de tester votre nouveau langage de programmation.

Depuis votre terminal, exécuter uniquement :

**ure;**

Par la suite, faites appel au langage de programmation que vous venez de créer afin de tester vos syntaxes :

**:s:test\_syntaxes.tt;**

```

AE@DESKTOP-JJF3JUG MSYS ~
$ ure

Welcome to the URE (Universal Runtime Environment) monitor. Commands end with ;
Copyright (c) 2017, 2022, By AnetoEnterprise Inc. All rights reserved.
URE is a registered trademark of AnetoEnterprise Inc.

URE >:s:test_syntaxes.tt;
URE >afficher='Voici mon message de teste à afficher';
Voici mon message de teste à afficher
quitter;
->Au revoir !

```

Si vous ne voulez pas placer vos syntaxes dans le répertoire **C:/URE/int**, vous pouvez exécuter depuis votre terminal la commande suivante afin que URE appelle directement le langage de programmation que vous voulez utiliser depuis le terminal.

**NB** : Cette procédure n'est pas fiable si une application exécutable à été programmée utilisant vos syntaxes (votre langage de programmation) ne sera pas prise en charge à l'environnement d'exécution universel. La meilleure chose à faire est de placer votre fichier des syntaxes au répertoire **int**.

Tapez la commande suivante depuis votre terminal :

```
ure -l "/test_syntaxes.txt"
```

Par la suite, tester vos syntaxes une par une :  
**afficher='Voici mon message de teste à afficher';**  
**quitter;**

**NB** : Si vous avez un message d'**Erreur fatal de la lecture** en retour, l'environnement d'exécution universel n'a pas trouvé votre fichier de syntaxes (Le langage de programmation cible). Vous devez vous assurer que vous avez bien spécifié le chemin du répertoire sur lequel votre fichier **test\_syntaxes.txt** se trouve. Si tout fonctionne correctement, essayer de traduire vos syntaxes à n'importe quelle langue de votre choix et tester à nouveau. Vous verrez que toutes les syntaxes fonctionneront correctement à condition de respecter les bibliothèques appelées entre parenthèses.

```

URE >afficher='Voici mon message de teste à afficher';
->Erreur fatal de la lecture !

```

**NB** : Nous avons pu ajouter le point-virgule (;) à la fin de chaque commande parce que depuis un terminal, l'environnement d'exécution universel exécute la commande s'il détecte le ; après l'action de la touche **ENTREE** (ENTER). En outre si le ; n'existe pas après avoir appuyé sur la touche entrée (enter), l'environnement d'exécution universel utilisera cette action pour placer le curseur dans une nouvelle ligne depuis le terminal.

- e) **Sécurité** : La sécurité des syntaxes dépend des programmeurs ou sociétés qui veulent cacher leurs syntaxes pour la sécurité de leurs nouveaux langages de programmation. Si vous voulez que votre langage de programmation soit à la portée de tous, laissez les syntaxes lisible pour que tout le monde arrive à bien traduire et de l'utiliser correctement.

Par contre, exécuter la commande suivante afin de sécuriser vos syntaxes :

**ure -e "./test\_syntaxes.txt" ".tt"**



```
AE@DESKTOP-JJF3JUG MSYS ~
$ ure -e "E:/AE/AnetoEnterprise/URE/DEMO/Examples/test_syntaxes.txt" ".tt"

Welcome to the URE (Universal Runtime Environment) monitor. Commands end with ;
Copyright (c) 2017, 2022, By AnetoEnterprise Inc. All rights reserved.
URE is a registered trademark of AnetoEnterprise Inc.

->L'exécutable crée correctement.
```

**NB** : Le **.tt** est l'extension de sortie de notre fichier des syntaxes qui sera compilé et sécurisé. Vous pouvez changer l'extension de sortie de votre choix afin de sécuriser votre fichier des syntaxes.

Au final, nous avons notre fichier des syntaxes sécurisé sur le disque. N'oubliez pas de le copier et coller vers le répertoire spécifique de lecture des syntaxes comme nous l'avons précisé ci-haut.

 test_syntaxes.tt	07/01/2022 23:03	Text Template File	1 Ko
 test_syntaxes	07/01/2022 22:48	Fichier TXT	1 Ko

Prenons un autre exemple, nous voulons que notre nouveau langage de programmation soit utiliser qu'avec deux langues le **Français** et le **Lingala** et qu'aucun autre programmeur aura le droit de le modifier. Dans ce cas, modifions notre fichier des syntaxes **test\_syntaxes.txt** comme suite :

```
quitter (test_exit);  
afficher= `?` (test_print);  
bima (test_exit);  
lakisa= `?` (test_print);
```

Enregistrer les modifications apportées et compiler afin de sécuriser le tout :

```
ure -e "./test_syntaxes.txt" ".tt"
```

Vous pouvez maintenant tester votre langage de programmation qui exécutera les syntaxes rien qu'en deux langues.

f) **Conclusion :**

Comme vous venez de le voir, la conception d'un langage de programmation est si simple. Ne requise non seulement la maîtrise de langage de programmation cible **C++**, mais vous avez aussi le choix d'utiliser le **C** ou **ASM** afin de concevoir votre nouveau langage de programmation. Passons maintenant au point final afin de concevoir quelques applications exécutables d'exemple en utilisant les syntaxes de notre nouveau langage de programmation.



## 15. Programmation (Concepteur des applications)

Tant que toutes choses ont étaient placées à leurs places, il est grand temps maintenant de concevoir quelques applications qui seront exécuter et utiliser par les utilisateurs.

### Exemple 1 :

Créer un nouveau dossier pour vos applications des tests. Dans ce nouveau dossier crée, ajouter un fichier **Exemples.txt** et placez dedans les syntaxes suivantes :

**test\_syntaxes.tt**

```
##Début afficher un message##  
afficher=`Voici mon message de teste à afficher`;  
##Fin afficher un message##
```

**NB** : Nous avons placé le fichier des syntaxes **test\_syntaxes.tt** à l'entête de notre fichier **Exemples.txt** pour que l'environnement d'exécution universel reconnaisse que nous voulons utiliser les syntaxes appropriées de notre nouveau langage de programmation et d'ignorer tous les restent des langages de programmation.



Enregistrez les modifications et compilez pour avoir l'application exécutable URE comme suite :

```
ure -e "./Exemples.txt" ".ure"
```

**NB** : L'extension **.ure** définie que nous voulons faire à notre application une application exécutable de l'environnement d'exécution universel. Si vous changez l'extension de sortie autre que celle définie de **.ure**, cette application ne sera pas exécuter à l'environnement universel tant qu'il reconnait uniquement l'extension **.ure**. Excepté que si vous l'avez exécutée en faisant un clic droit et ouvrir avec l'environnement d'exécution universel.

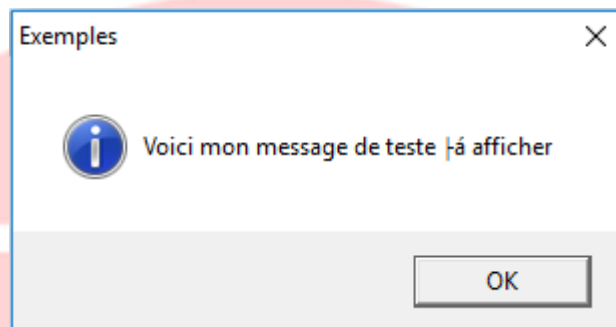
Vous pouvez voir depuis votre répertoire de projet un fichier exécutable **Exemples.ure** porte l'icône d'URE en rouge. Faites un double clic afin de

l'exécuter comme n'importe quelle application exécutable dans votre système d'exploitation.

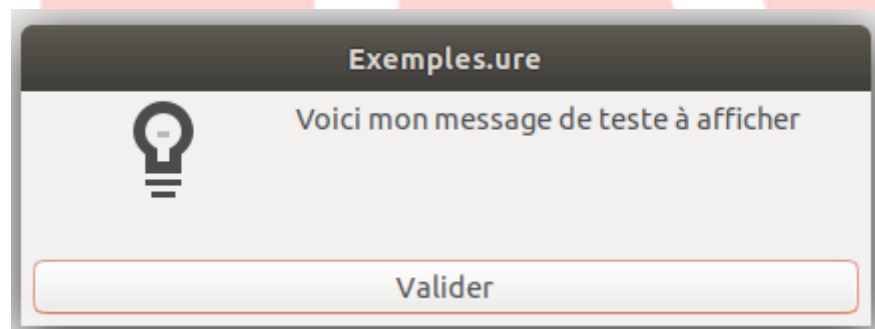
 Exemples	07/01/2022 22:46	Fichier TXT	1 Ko
 Exemples	08/01/2022 05:16	Fichier URE	1 Ko

Après l'exécution par double clic, vous aurez le resultat :

✓ Sous Windows :



✓ Sous Linux :



**Exemple 2 :**

Ajouter un nouveau fichier **Exemple2.txt** et placez dedans les syntaxes suivantes :

**test\_syntaxes.tt**

**##Début afficher un message##**

**afficher= `Message 1`;**

**afficher= `Message 2`;**

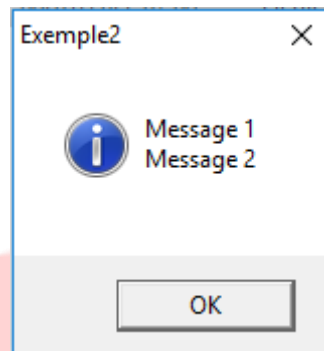
**##Fin afficher un message##**

Enregistrez les modifications et compilez :

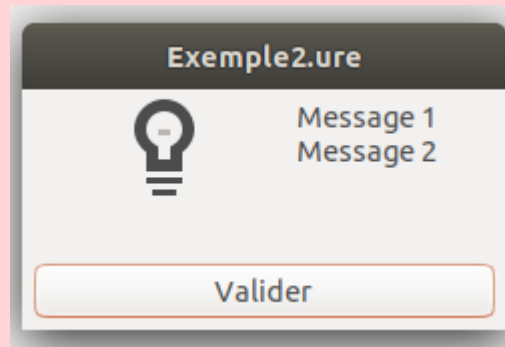
**ure -e "./Exemples.txt" ".ure"**

Exécuter en faisant un double clic afin de voir le resultat final :

✓ Sous Windows :



✓ Sous Linux :



## 16. Conclusion

Nous venons de voir comment installer et configurer l'environnement d'exécution universel, crée, adapté et utilisé un langage de programmation simple ou sécurisé (LP) et sans oublié multilingues. Alors l'avenir de demain, la compréhension, l'invention et de l'innovation de nos enfants, frères, sœurs et petits fils sont à notre portée. Le seul problème d'aujourd'hui, certaines communautés pensent qu'obliger leurs enfants à maîtriser les langues étrangères, peuvent apporter les sourires, non seulement dans des vies quotidiennes de leurs enfants, elles pensent qu'elles changeraient aussi l'avancer de leurs pays. Par contre, l'effet de bien éduquer nos enfants et de les faire comprendre l'existence et les fonctionnements dans certains métiers (domaines), est la base de toute évolution mondiale. L'invention de cet environnement d'exécution universel n'interdit pas le monde entier à apprendre les langues étrangères. Chaque personne est libre d'apprendre n'importe quelle langue de son choix et s'assurer que cette langue à l'avenir apportera un plus, non seulement dans sa vie, mais aussi dans son pays natal ou le pays actuel habité. Si vous voulez être spécialiste en littérature d'une certaine

langue, spécialisez-vous aisément. Vous voulez être expert en programmation informatique ou autres domaines scientifiques, garder l'utilisation de votre langue compréhensive au lieu d'être obligé ou perdre votre temps d'apprendre à comprendre les langues étrangères, à la fin revenir comprendre les bons fonctionnements de ton métier tandis que d'autres communautés comprennent et avance rapidement avec pleines d'inventions et d'innovations. Ton avenir n'a pas était écrit en utilisant une langue étrangère, non plus la connaissance le requise comme tu le pense. Seul Dieu le sait !

- **Etes-vous intellectuel ou pas ?**
- **Comment les autres arrivent à traduire et adapté la mathématique, physique, la programmation et autres en utilisant leurs propres langues compréhensives, tandis que vous vous n'arrivez pas à le faire pour la meilleure compréhension de vos enfants et l'avancer de votre pays ?**

L'environnement d'exécution universel est à la portée de qui conque veut l'utiliser ou l'adapter à sa nouvelle invention et nouvelle innovation pour qu'elle soit multilingues. Alors, grâce à sa liberté de programmation et d'exécution, **AnetoEnterprise Inc.** promet à toutes les communautés dans le monde entier l'invention de nouveaux langages de programmation informatique adaptés, programmables et exécutables dans tous les système d'exploitation existants et dans des cartes programmables telles qu'**Arduino, Rasberry, Pic** etc... Afin de permettre aux gens de votre communauté de comprendre grâce à leurs propres langues et de créer des **systèmes informatiques, robotiques** ou d'autres systèmes de **l'intelligence artificielle (IA)**. Maintenant il est grand temps que vous et moi arrivent à perfectionner nos pensées et de changer ce monde.

Pour les communautés ou les bonnes personnes qui veulent financer **URE** afin de perfectionner son utilisation ou l'adaptation de ses fonctionnements à d'autres systèmes qui ne sont pas disponible pour l'instant, merci de nous contacter aux adresses suivantes.

**Contacts :**

**E-mail :** [Kindlysertir@gmail.com](mailto:Kindlysertir@gmail.com)

**Skype :** AnetoEnterprise



Copyright 2021 by AnetoEnterprise Inc. All rights reserved