OPIX : un interpréteur pour entiers de très grande taille

Manuel de l'utilisateur

Plan du manuel :

1°) Résumé
2°) Les opérateurs
3°) Les fonctions
4°) Les fichiers de commandes
5°) Les commandes auxiliaires
6°) L'utilisation
7°) Les mots réservés
8°) Exemples, trucs et astuces

1°) Résumé

OPIX est un logiciel pour effectuer des calculs arithmétiques avec des entiers de très grandes tailles, aussi grandes que nécessaires. En plus des opérations de base : addition, soustraction, multiplication, division, exponentiation et modulo il permet d'effectuer diverses autres calculs parmi lesquels : critère de primalité, plus petit diviseur premier, plus grand commun diviseur, plus petit commun multiple, inverse modulaire, exponentiation modulaire, factorisation ...
OPIX est aussi un langage de programmation avec les notions de variables globales ou locales, les débranchements conditionnels, les boucles d'itérations et quelques autres aspects. Un programme OPIX peut en appeler une autre. Les programmes OPIX sont interprétés.
L'historique de la session peut être enregistré. De plus, les calculs effectués peuvent être sauvegardés et restitués en tout ou en partie.
OPIX repose sur la bibliothèque complémentaire BigInteger, ce qui lui permet de n'avoir aucune dépendance externe.
OPIX fonctionne sous Windows en mode console. Il est compilé avec Visual Studio 2012 Express for Windows Desktop en français. C'est un programme portable : il n'a pas besoin d'un installateur et ne met rien dans le système Windows.

2°) Les opérateurs

Les opérateurs du langage sont : un opérateur à un argument, 11 opérateurs à deux arguments et un opérateur à trois arguments.

2-1 L'opérateur à un seul argument

2-1-1 Calcul de l'opposé

L'opérateur : - permet d'obtenir l'opposé de l'entier désigné, exemples : `x = -n` ou : `-123`. Il convient de noter que chaque fois que l'on référence une variable par son nom on peut aussi référencer son opposé dans les mêmes circonstances.

2-2 Les opérateurs à deux arguments

2-2-1 L'addition : x = a + b

Exemples : `s = t + 7777` ou simplement : `1234567890123 + 9876543210`

2-2-2 Soustraction : x = a - b

Exemples : `d = 1234567 – 567890` ou : `v = -p – q`
Comme indiqué ci-avant, l'exemple : –p – q signifie : soustraire q de l'opposé de p.

2-2-3 Multiplication : x = a * b

Exemples : `m = 2 * -p` ou : `t = n * u`
Le calcul se base en priorité sur la multiplication rapide par la méthode FFT, Fast Fourier Transform, et en cas d'échec sur la méthode traditionnelle.

2-2-4 Division : x = a / b

Exemples : `q = n / 7654321` ou : `quot = num / den`
Le résultat est double : quotient et reste. Comme pour la racine carrée, si le résultat est archivé, le nom de la variable est utilisé pour le quotient et le reste est désigné par le même nom augmenté d'un : .r donc : `x = a / b` produit deux variables : x et x.r ce qu'il convient de savoir. En effet, si la variable x.r existait déjà et contenait une information utile, celle-ci sera perdue sans avertissement. Pour la variable x c'est pareil mais logiquement on s'y attend de manière évidente.

2-2-5 Exponentiation : x = a ^ b

Exemples : `n2 = n ^ 2` ou : `m = 2 ^ 100`
Le résultat est calculé par récursivité.

2-2-6 Modulo : x = a % b

Exemples : `r = u % v` ou simplement : `12345 % 10`
x = a % b signifie : x = a mod(b) c'est le reste de la division de a par b. On dit que x et a sont congrus modulo b.

2-2-7 Plus grand commun diviseur : x = a & b

Exemples : `d = -u & 999` ou : `pgcd = 6510 & 9870`
Le calcul se fait avec la méthode d'Euclide et le temps d'exécution est bon.

2-2-8 Plus petit commun multiple : x = a ° b

Exemples : `m = -u ° 999` ou : `ppcm = 6510 ° 9870`
Quelque soient n et m on a toujours : m * n = pgcd(m,n) * ppcm(m,n). Donc le calcul se fait comme le pgcd.

2-2-9 Comparaison : x = a | b

Exemples : `i = u | v` ou : `m = p | 1000`
Le résultat est -1 si a < b, il est 0 si a = b et il est 1 si a > b

2-2-10 Au hasard : x = a ? b

Exemples : `m = 1000 ? 2000` ou : `in = p ? q`
Le résultat est un nombre obtenu au hasard entre a et b. Le tirage au sort est fait pour fournir une valeur différente quand on relance plusieurs fois la même commande.

2-2-11 Inverse modulaire : x = a \ b

Exemples : `i = n \ 2000` ou : `d = e \ m`
Le résultat, quand il existe, est le nombre x de l'intervalle [0, b-1] tel que : x * a = 1 + k * b
Si on augmente ou bien si on diminue la valeur de a de une ou plusieurs fois la valeur de b, cela donne le même inverse modulaire x avec seulement une autre valeur pour le nombre k. Le théorème de Bezout dit que pour tout couple de nombres entiers a et b il existe deux entiers u et v tels que a u + b v = r = pgcd(a,b) Donc si a et b n'ont aucun diviseur commun on aura toujours a u = 1 – b v. L'inverse modulaire x = a \ b se calcule avec l'algorithme d'Euclide étendu qui donne u et v, il reste ensuite à translater u autant de fois que nécessaire pour aller dans [0, b-1]. On peut aussi noter que a \ 1 = 0 pour tout a. Par contre si a et b ont un ou plusieurs diviseurs commun a \ b n'existe pas et OPIX fournit un résultat nul.

2-3 L'opérateur à trois arguments

2-3-1 Exponentiation modulaire : x = a ^ b % m

Exemple : `y = x ^ e % n`
L'exponentiation binaire encore appelée exponentiation par carrés, adaptée à ce cas, permet d'utiliser de bien plus grandes valeurs de l'exposant que l'exponentiation directe le permettrait. Cette opération est la base de la méthode RSA pour chiffrer ou déchiffrer un message.

3°) Les fonctions

Les fonctions OPIX ont toutes un, et un seul, argument.

3-1 Nombre de chiffres décimaux

Exemples : `n = nbch a` ou simplement : `nbch x`
Le résultat est le nombre de chiffres décimaux utilisés dans l'écriture usuelle de l'argument.

3-2 Critère de primalité

Exemples : `t = prem p` ou simplement : `prem 538338664565940325587701217593`
Le résultat est 1 pour un nombre premier ou 0 si non. Il est évalué par la méthode probabiliste de Miller et Rabin. Les nombres négatifs sont contrôlés en valeur absolue.

3-3 Factoriel

Exemples : `fn = fact n` ou : `f100 = fact 100`
A noter : 1000! a 2568 chiffres décimaux et peut être calculé rapidement. Mais 10000! a 35660 chiffres décimaux, ce qui est beaucoup ! L'argument doit être positif.

3-4 Primoriel

Exemples : `x = prim n` ou : `p100 = prim 100`
Le primoriel de x est le produit de tous les nombres premiers qui sont plus petits ou égaux à x. L'argument doit être positif.

3-5 Valeur absolue

Exemple : `x = vabs n`

3-6 Un diviseur

Exemple : `x = ndiv n`
Le résultat est un diviseur de n. Si n est premier c'est la valeur de n lui-même, si non c'est un diviseur de n plus petit que n. Le résultat n'est pas toujours premier et n'est pas toujours le plus petit diviseur. C'est seulement le premier diviseur trouvé. On utilise en cascade plusieurs méthodes : les divisions successives par les premiers nombres premiers limitées à un nombre maximum d'essais, la méthode de Fermat limitée à un nombre maximum d'essais, l'algorithme rho de Pollard, la méthode p-1 de Pollard limitée à un nombre maximum d'essais et la méthode ecm (elliptic curve method) de Lenstra. Le temps de calcul est très variable et difficilement prévisible.

3-7 Racine carrée

Exemples : `x = sqrt n` ou : `sqrt 123457`
Il y a là aussi un double résultat. Si le résultat est archivé, le reste est désigné par le même nom que le résultat principal augmenté d'un : .r donc : `x = sqrt n` produit deux variables : x et x.r comme pour les résultats de la division, il convient d'y faire attention.

3-8 Logarithme népérien

Exemples : `logn n` ou : `logn 987654`
Le résultat est un nombre réel. Il ne peut pas être archivé. Il est seulement affiché.

3-9 Factorisation

Exemples : `fctp n` ou : `fctp 987654`
C'est le calcul et l'affichage de la décomposition de l'argument en facteurs premiers. Là aussi
le résultat ne peut pas être archivé. Il est seulement affiché. Pour cette fonction comme pour la fonction ppdp le temps de calcul est très variable et difficilement prévisible. A ce propos, on peut rappeler ici que le chiffrement RSA repose sur l'impossibilité, en pratique, de factoriser un grand nombre entier qui est le produit de deux nombres premiers inconnus. Un utilitaire de démonstration du procédé RSA est disponible plus en avant.

3-10 Plus petit diviseur premier

Exemples : `ppdp n` ou : `ppdp 987654`
Ce calcul utilise la décomposition en facteurs premiers pour en sélectionner le plus petit diviseur premier. Le résultat peut être archivé. Le temps de calcul est très variable et difficilement prévisible. Il faut aussi ajouter qu'il est facile d'utiliser cette commande ppdp pour obtenir et archiver tous les diviseurs du nombre proposé et non pas seulement le plus petit. Pour cela, un fichier de commandes est présenté ci-après.

3-11 Premier nombre premier suivant

Exemple : `pnps n` On calcule ici le nombre premier suivant algébriquement par l'intermédiaire d'une recherche systématique à l'aide du critère de Miller & Rabin. De `-2 à 1` on obtient `2` et pour : `pnps 9` on obtient : `11` et pour : `pnps -9` on obtient : `-7.` On rappelle ici que la commande pour savoir si un entier donné est premier ou composé est : `prem n.`

3-12 Suppression d'une variable

Exemple : `zvar n`

4°) Les fichiers de commandes

Sauf exceptions, OPIX utilise seulement les lettres en minuscules. Les commandes sont entrées au clavier ou lues dans des fichiers de commandes. Les variables n'ont plus d'existence à la fin de la session mais selon les besoins on peut les sauvegarder et les restituer. Les variables sont toutes de même type et sont principalement globales mais on peut spécifier des variables externes et des variables internes nécessaires au programme concerné.

4-1 Fichier de commandes ou programme OPIX : exec fichier

La commande : `exec fichier` lance l'exécution d'un fichier de commandes préparé à l'avance. Un fichier de commandes peut en appeler un autre. Ils contiennent en séquence des instructions comme celles déjà vues et d'autres qui leurs sont spécifiques. Il y a des fichiers de commandes qui servent d'utilitaires pour apporter des services et d'autres qui sont plus orientés vers une application particulière

4-2 La commande : echo et les commentaires

La commande : `echo texte` est un message destiné à venir s'afficher pendant l'exécution. Cette instruction accepte tous les caractères de Windows. De plus, les conversions entre les deux codes : ASCII étendu OEM et ASCII étendu ANSI sont effectuées pour cette instruction comme pour toutes les autres chaque fois que nécessaire.
On peut inclure dans les fichiers de commandes deux types de commentaires. Les commentaires habituels sont simplement mis entre parenthèses sur une ligne entière ou sur une fin de ligne seulement. Ils servent à expliquer l'organisation du fichier de commandes. Ils ne sont jamais affichés pendant l'exécution. Les commentaires entre crochets sont vus pendant l'exécution. Si la commande echo est principalement orientée vers des messages à l'utilisateur, les commentaires entre crochets sont utilisés pour afficher un petit renseignement très simple en complément d'une instruction.

4-3 La variable : voir

Le programme dispose d'une variable d'environnement nommée : voir qui ne peut avoir que les valeurs 0 ou 1. Si `voir = 1` les instructions du fichier de commandes et les réponses obtenues sont affichées à l'écran et si `voir = 0` elles ne le sont pas. Le principal avantage de cette variable est qu'il est très facile d'en archiver l'état et de le restituer pendant l'exécution d'un fichier de commandes. La variable voir ne concerne pas les écritures dans le fichier memo s'il est ouvert.

4-4 La commande : stop

Dans un fichier de commandes la commande stop provoque un arrêt momentané de l'exécution en cours. Cela peut servir pour aider la mise au point du fichier de commandes ou bien pour voir les résultats intermédiaires sans que le "scrolling" les fasse disparaître trop vite.

4-5 Les commandes evar et ivar

Les variables sont principalement globales aussi bien en conversationnel qu'à l'intérieur des fichiers de commandes. Ceci peut créer quelques difficultés en cas de conflits dans les noms utilisés dans et hors un fichier de commandes. Pour y faire face deux commandes spécifiques sont utilisables.
La commande : `evar x` a pour effet d'abandonner l'exécution du fichier de commandes en cours si la variable x n'existe pas. Cette instruction sera donc utile pour s'assurer de la disponibilité des variables externes servant de données au fichier de commandes en cours.
La commande : `ivar y` a pour effet d'abandonner l'exécution du fichier de commandes en cours si la variable y existe déjà. Cette instruction sera donc utile pour s'assurer que les variables internes au fichier de commandes ne risquent pas de mettre à jour, et donc de modifier, des variables externes du même nom mais complètement différentes. Si on a contrôlé avec ivar une ou plusieurs variables internes il ne sera plus possible d'exécuter une autre fois le même fichier de commandes dans la même session. En effet, l'existence résiduelle de ces variables internes sera contrôlée à nouveau et provoquera l'abandon du fichier de commandes. Pour y remédier on prendra la précaution de supprimer en fin de fichier les variables internes concernées via la commande zvar v.
La mise en œuvre de evar et ivar reste facultative et n'est pas toujours indispensable.

4-5 La commande arsi : arsi a < b

Cette commande permet d'abandonner l'exécution du fichier de commandes en cours si la condition est vraie au moment de son utilisation. C'est la forme la plus simple pour effectuer des débranchements conditionnels. Les arguments sont soit deux variables soit une variable et un nombre.

4-6 Les commandes pour et retour : pour a < b ... retour

Elles permettent de faire des boucles itératives. Les instructions qui sont disposées entre la commande : `pour a < b` et la commande : `retour` seront exécutées autant de fois que la condition est vraie. Si la condition est fausse dès la première fois elles ne seront pas exécutées. De plus on peut imbriquer une boucle `pour . . . retour` dans une autre. Il y a un exemple qui en montre l'imbrication sur 3 niveaux. Enfin, on explique plus en avant qu'il est aussi possible

d’utiliser ce dispositif pour exécuter une séquence d’instructions uniquement quand une condition d’égalité a = b est vraie.

5°) Les commandes auxiliaires

5-1 Création d’une variable

Les noms de variable sont en minuscules, commencent par une lettre et peuvent comporter ensuite des chiffres et le caractère : . (sauf à la fin). Exemple : `habitants.paris.2015` est un nom de variable valide.

5-2 Consultation d’une variable : var

Pour connaître la valeur de l’entier qui est rangé dans une variable il suffit d’entrer son nom.

5-3 Suppression d’une variable : zvar v

Rappel : cette instruction provoque la suppression d’une variable et de son contenu.

5-4 Aide : aide

Cette instruction provoque l'affichage d'un très court mode d'emploi. Mais ce manuel est beaucoup plus complet.

5-5 Exit : exit

Cette instruction provoque la fin de la session. Aucune sauvegarde n'est faite à cette occasion.

5-6 Sauvegarde : sauv fichier

Cette instruction provoque la sauvegarde de toutes les variables actives au moment de son utilisation. Le fichier obtenu est un fichier de commandes qui permet d'en faire la restitution dans une session ultérieure.

5-7 Fichier memo : memo fichier et memo stop

La commande : `memo fichier` crée un fichier qui enregistre toutes les commandes reçues par le système et les réponses qu’elles ont obtenues. Et la commande : `memo stop` ferme le fichier memo. Il faut noter que ce fichier est libéré entre deux écritures. Ce qui permet de le consulter à tout moment de la session. A chaque départ de OPIX le fichier memo nommé `memo.txt` est activé. Si l'ancien fichier memo.txt existe encore, il est perdu. On peut soit désactiver le nouveau fichier memo.txt : `memo stop` soit conserver l'historique de la session en changeant son nom.

6°) L'utilisation

On peut lancer OPIX en double cliquant sur son nom dans Windows ou bien avec un fichier opix.bat comme celui-ci :

```
@echo off
opix fichier-initial.txt
```

dans ce cas, le fichier de commandes : `fichier-initial.txt` sera lancé automatiquement au départ. Le mode conversationnel commence ensuite.
Quand un incident ou une erreur d'exécution est détectée le système émet soit un avertissement soit un message avec arrêt du programme.

Il est préférable d'avoir un fichier memo ouvert. En cas de problème ou d'imprévu on peut le relire.

Il est commode de nommer les fichiers de commandes servant d'utilitaires avec des noms sans type et de réserver des noms comme : application-exe.txt ou monpb-fic.txt pour les fichiers de niveau supérieur.

Si on doit saisir un nombre de très grande taille dans un fichier de commandes il est commode d'utiliser WordPad qui permet facilement de gérer des lignes très longues.

OPIX est portable. Il n'installe rien dans le système Windows ni dans la base de registre. Il n'a pas besoin d'installeur. Il est avantageux de lui réserver un dossier de travail et un autre d'archive ou de référence. On peut l'utiliser à partir d'une clef USB.

On peut conserver l'historique d'une session et restituer ensuite son exécution. Pour cela, il suffit de garder le fichier memo.txt obtenu, par précaution de le renommer en session.txt, par exemple, et de lancer l'utilitaire histo.exe avec histo.bat, ce qui fournit un fichier de commandes session.fic.txt qu'il suffit d'exécuter pour restituer tout l'environnement obtenu au moment où la session avait été mémorisée.

7°) Mots réservés

Les mots du langage qu'il convient de réserver sont :

- la variable d'environnement : voir
- les opérateurs : = + - * / ^ % & ° | ? \
- les mots : sqrt nbch prem vabs fact prim logn ndiv fctp ppdp pnps zvar sauv memo aide exec echo voir stop ivar evar arsi pour retour
- les noms de fichiers : opix-pour-n.txt (avec n : 1, 2, …)

8°) Exemples, trucs et astuces

8-1 Les nombres de Mersenne

Les nombres p = 2 ^ n – 1 tantôt sont premiers et tantôt ne le sont pas. Le 8-ième nombre premier de Mersenne est : 2 147 483 647 = 2 ^ 31 – 1 Il fut découvert par Euler en 1750. Il est bien connu des informaticiens. Pour n plus grand, ils sont rapidement de très grande taille et peuvent servir d'exemples. Deux exemples sont disponibles, les 14-ième et 17-ième : m14.txt et m17.txt.

8-2 Les nombres de Fermat

Les nombres p = 2 ^ (2 ^ n) + 1 sont premiers pour n de 1 à 4 mais ensuite aucun n'est premier. Ils deviennent très vite très grands et ils sont très difficiles à factoriser.

8-3 Exécuter une séquence d'instruction si la condition a = b est vraie

Dans un fichier de commandes, pour exécuter une séquence d'instructions quand une condition égalité est vraie et sauter si la condition est fausse.

Programme principal	Fichier de commandes : xse
xsa = a xsb = b exec xse pour xse < 1 [saut si xsa ≠ xsb] xse = 1 [effectué une fois si xsa = xsb] retour	xsv = voir evar xsa evar xsb voir = 0 xse = 2 xsx = xsa - 1 pour xsx < xsb xse = xse - 1 xsx = xsb + 1 retour xsx = xsb - 1 pour xsx < xsa xse = xse - 1 xsx = xsa + 1 retour voir = xsv

Nota : pour exécuter une fois une séquence d'instructions quand une condition : a < b est vraie il suffit de faire :

```
xsa = a
xsb = b
pour xsa < xsb [ saut si a = b ou a > b ]
xsa = xsb + 1


[ effectué une fois si a < b ]


retour
```

8-4 Obtenir tous les diviseurs

Pour obtenir tous les diviseurs du nombre a, on peut utiliser l'utilitaire ci-joint. Mais attention, les diviseurs, tous nombres premiers, de grande taille sont difficiles voire impossibles à obtenir.

Programme principal	Fichier de commandes : dvs
dva = a exec dvs	dve = voir evar dva voir = 0 echo : calcul de tous les diviseurs de dvb dvb = dva dvn = 0 pour 1 < dvb dvn = dvn + 1 dvv = ndiv dvb voir = 1 echo : le diviseur numéro dvn vaut dvv dvn dvv voir = 0 dvb = dvb / dvv retour echo : fin du script dvs voir = dve

Nota : Si on souhaite exécuter une fois une séquence d'instructions quand un nombre est premier il suffit de faire :

```
nb = ... [ c'est le nombre à contrôler ]
xb = prem nb
pour 0 < xb [ saut si nb n'est pas premier ]
xb = 0


[ effectué une fois si nb est premier ]


retour
```

8-5 Chiffrement et déchiffrement RSA

Voici un exemple de chiffrement et de déchiffrement selon la méthode RSA.

Programme principal	Explications
a = 1000 b = 2000 p = a ? b p = pnps p q = a ? b q = pnps q n = p * q p1 = p - 1 q1 = q - 1 m = p1 * q1 e = 200 ? 300 i = e & m pour 1 < i e = 200 ? 300 i = e & m retour d = e \ m x = 12345 y = x ^ e % n z = y ^ d % n	p : nombre premier au hasard entre a et b q : nombre premier au hasard entre a et b n : 1-er nombre de la clé publique et de la clé secrète n = p * q m = (p – 1)(q – 1) = phi(n) e : 2-nd nombre de la clé publique il faut pgcd(e,m) = 1 d : 2-nd nombre de la clé secrète il faut : e * d = 1 + k * m x : message à chiffrer y : message chiffré z : message déchiffré (=> z = x)