Chapitre 3

Premiers éléments de AsmL

Nous supposons que le lecteur est déjà plus ou moins familiarisé avec un langage de programmation. Nous prendrons le langage C, le plus utilisé de nos jours, à titre de comparaison. Nous allons montrer en quoi le langage ASM se différencie d'un tel langage.

On pourra se référer à [Mic02b] pour d'autres exemples introductifs.

3.1 Héritage de Framework .NET

Nous avons dit que l'implémentation de AsmL est effectuée en utilisant Framework .NET. Une partie de AsmL n'a rien à voir avec les ASM, il s'agit tout simplement d'un héritage minimum de la nouvelle API de Microsoft. Comme celui-ci sera utilisé dans presque tous les programmes, commençons par lui.

3.1.1 Retour sur le premier programme

Programme source.- Rappelons le premier programme que nous avons écrit :

```
Main()
  WriteLine("Bonjour")
```

Début d'un programme.- On marque le début d'un programme par le mot clé :

`Main()`

Cette convention provient évidemment plus ou moins du langage C. Il s'agit en fait d'un mot clé de Framework .NET. On remarquera que ce mot clé commence par une majuscule : cela provient de la programmation Windows ; tout programme écrit pour Windows possède une fonction principale de nom 'Main()', la bibliothèque Windows (écrite en langage C et possédant donc une fonction 'main()') recherchant ce début de programme.

Affichage à l'écran.- L'affichage à l'écran s'effectue grâce à la méthode :

`WriteLine()`

si on veut un passage à la ligne ou :

`Write()`

si on ne veut pas de passage à la ligne. Dans les deux cas le paramètre est une chaîne de caractères. Là encore il s'agit d'une méthode de Framework .NET qui n'est pas propre à AsmL.

Constante chaîne de caractères.- Pour Framework .NET, une chaîne de caractères est une suite de caractères Unicode.

Pour indiquer une constante, on entoure cette suite de guillemets verticaux comme en langage C.

Pour spécifier un caractère Unicode non ASCII strict (de code compris entre 0 et 127), on utilise la convention Java :

`\uXXXX`

avec quatre chiffres hexadécimaux. Par exemple `\u00E9` pour afficher 'é'[1].

Délimitation des blocs.- On aurait pu s'attendre à ce que ce programme s'écrive :

```
Main()
  {
  WriteLine("Bonjour");
  }
```

avec des accolades pour délimiter les blocs et un point-virgule pour terminer l'instruction élémentaire.

Il n'en est pas ainsi en AsmL : les instructions élémentaires ne se terminent pas par un point-virgule ; on joue sur l'indentation pour délimiter les blocs.

[1] En fait, puisqu'un programme AsmL est du texte Unicode, tout caractère qui a été écrit à l'aide du clavier est accepté et sera affiché tel quel.

3.1.2 Les types primitifs

AsmL est un langage orienté objet dont les classes primitives sont :

- La superclasse `Object`, avec les deux constantes `null` et `void`.
- La classe `Boolean` avec les deux constantes `true` et `false`.
- Les quatre classes d'entiers relatifs `Byte`, codés sur un octet, `Short`, codés sur deux octets, `Integer`, codés sur quatre octets, et `Long`, codés sur huit octets.
- Les deux classes de décimaux `Float` et `Double`.
- La classe de caractères `Char`, codés sur deux octets, Unicode oblige.
- La classe de chaînes de caractères `String`.

avec les opérations à lesquelles on peut penser (le symbole de concaténation des chaînes de caractères étant '+' ; il n'y a pas de fonctions mathématiques telles que sinus).

Exemple.- Écrivons un programme AsmL qui affiche '2 + 2 =' ainsi que le résultat de l'opération :

```
Main()
    WriteLine("2 + 2 = " + (2 + 2))
```

Exercice. *Écrire un programme AsmL qui affiche* :

Le garçon se promène en récitant les décimales de π.

[On se servira du code Unicode publié à l'adresse :

`http://www.unicode.org`]

3.1.3 Les variables et les constantes globales

Constantes.- Une constante globale se déclare avant le mot clé '`Main()`' de la façon suivante :

```
identificateur = valeur
```

ou :

```
identificateur as type = valeur
```

Bien souvent AsmL n'a pas besoin du type.

Variables.- Une variable globale se déclare avant le mot clé '`Main()`' de la façon suivante :

```
var identificateur = valeur
```

ou :

```
var identificateur as type = valeur
```

ou :

```
var identificateur as type
```

Exemple.- Écrivons un programme AsmL qui utilise une constante et une variable, sans vraiment tenir compte de leurs statuts :

```
a = 2
var x = 3
Main()
    WriteLine("a*x = " + (a*x))
```

Les identificateurs en AsmL.- En gros les identificateurs sont des chaînes de caractères sur l'alphabet Unicode (ne commençant pas par un chiffre) sauf les mots réservés. Nous renvoyons au manuel Asml [Mic02a] pour une définition précise.

3.2 Mise à jour

3.2.1 Syntaxe

Notion.- Comme dans tout langage de programmation, l'instruction la plus élémentaire est l'affectation, renommée *mise à jour* (*update* en anglais) en ASM.

Syntaxe.- AsmL utilise le signe composé du langage Pascal pour indiquer la mise à jour :

```
valeur := expression
```

Exemple.- Écrivons un programme AsmL qui convertit une température de degré Celsius en degré Farenheit :

```
var C = 19
var F = 0

Main()
    F := (9*C)/5 + 32
    WriteLine(F)
```

On reconnaît bien là la forme d'un programme auquel on s'attend mais on a une surprise lorsqu'on le fait exécuter. L'affichage donne souvent 0 et non pas le 66 attendu, cela dépend de la session. Cela est dû à la sémantique des ASM.

3.2.2 La simultanéïté par défaut

Dans tous les langages de programmation, la structure de contrôle par défaut est le *séquencement*, c'est-à-dire que lorsqu'on a une instruction suivie d'une autre instruction, la première instruction est effectuée et immédiatement prise en compte puis la seconde instruction est effectuée. Par conséquent l'action de la seconde instruction dépend de celle de la première.

L'essence des ASM est que la structure de contrôle par défaut n'est pas le séquencement. Au lieu d'être effectuées l'une après l'autre, les instructions sont, dans une première étape, effectuées les unes indépendamment des autres. Si ceci ne donne pas lieu à une incohérence, on prend en compte ce qui a été effectué. On parle de *parallélisme* dans les premiers écrits sur les ASM bien qu'il ne s'agisse pas réellement de cela : les instructions peuvent être exécutées dans n'importe quel ordre et les valeurs des variables ne seront affectées que lorsque toutes les instructions auront été exécutées. Qu'importe comment cette structure de contrôle est implémentée ! Il s'agit visiblement d'une structure de contrôle naturelle lorsqu'on énonce des algorithmes. C'est toute la gloire de ASM de l'avoir dégagée, cinquante ans après la naissance des premiers langages algorithmiques (machines de Turing et autres). On parle maintenant de **simultanéïté**.

3.3 Le séquencement

3.3.1 Indication d'étape

Syntaxe.- Nous venons de voir que la structure de contrôle par défaut est la simultanéïté. Cependant, et notre premier exemple l'a montré, on a souvent besoin de séquencement. En ASM on émule le séquencement avec les structures de contrôle comme nous le verrons plus tard. Mais, puisque cela est très utilisé, AsmL utilise aussi le mot clé :

```
step
```

pour indiquer que le bloc qui suit constitue une étape. On distingue donc les étapes par les préfixes `step`.

Exemple.- Écrivons maintenant l'exemple précédent de façon correcte en AsmL :

```
var C = 19
var F = 0

Main()
  step
    F := (9*C)/5 + 32
  step
    WriteLine(F)
```

Cette fois, l'exécution permet bien d'afficher ce qu'il faut, à savoir 66.

3.3.2 L'échange

On peut se demander pourquoi avoir choisi la simultanéïté comme structure de contrôle par défaut. Nous allons le voir dans l'exemple classique de l'échange. Nous avons déjà dit au chapitre un que sa programmation n'est naturelle dans aucun langage (antérieur aux ASM). Dans le cas des ASM, pour échanger `a` et `b`, il suffit d'une étape naturelle :

```
    a := b
    b := a
```

comme le confirme l'exécution du programme AsmL suivant :

```
var a = 5
var b = 12

Main()
  step
    Write("Dans la premi\u00E8re \u00E9tape, a = " + a)
    WriteLine(" et b = " + b)
    a := b
    b := a
  step
    Write("Dans la seconde \u00E9tape, a = " + a)
    WriteLine(" et b = " + b)
```

3.3.3 Saisie

Arrivé à ce point, le néophyte demandera comment effectuer une saisie, ce qui rendrait les programmes plus réalistes. Les ASM ne s'intéressent ni aux ordres de lecture, ni aux ordres d'écriture. Il s'agit d'interaction avec le monde extérieur, qui n'est pas fondamental pour l'aspect algorithmique.

Syntaxe en AsmL.- Le langage AsmL, par contre, ne peut pas faire l'impasse sur cet ordre. Comme pour l'écriture, il hérite du Framework .NET de la méthode :

```
ReadLine()
```

qui renvoie une chaîne de caractères (comme en Java, pour ceux qui connaissent ce langage).

Exemple.- Réécrivons notre programme sur la conversion de température sous une forme un peu plus interactive :

```
var C as Integer
var F as Integer
var s as String

Main()
  step
    s := ReadLine()
  step
    C := ToInteger(s)
  step
    F := (9*C)/5 + 32
  step
    WriteLine(F)
```

Puisqu'on ne peut lire qu'une chaîne de caractères, il faut effectuer une conversion explicite de la chaîne de caractères obtenu en un entier. Là encore Framework .NET nous fournit des méthodes adéquates, telles que `ToInteger()`.

3.3.4 Cohérence des mises à jour

La simultanéïté peut conduire à des incohérences dans les mises à jour : on peut, dans une étape donnée, proposer une valeur pour une variable dans une première instruction et une autre valeur dans une autre instruction. Ceci n'arrive pas lorsque la structure de contrôle par défaut est le séquencement puisque les instructions sont prises en compte l'une après l'autre. Mais ceci n'est pas le cas avec la simultanéïté puisqu'il n'y a pas d'ordre pour les instructions.

De façon à ne pas rendre le déroulement du programme non déterministe, de telles incohérences donnent lieu à des exceptions.

Exemple.- Le programme suivant :

```
var a = 5

Main()
  step
    WriteLine("Dans la premi\u00E8re \u00E9tape, a = " + a)
    a := 8
    a := a + 1
  step
    Write("Dans la seconde \u00E9tape, a = " + a)
```

ne présenterait pas de problème dans un langage impératif habituel et afficherait '9' pour la seconde étape. Dans le cas des ASM il y a un problème : lors de la première étape, on essaie de donner les valeurs 8 et 9 à la variable.

On pourra vérifier que le programme se compile normalement mais, que lors de l'exécution, une exception est levée.

3.3.5 Les commentaires

Syntaxe.- Comme en langage C++, on peut utiliser les commentaires de fin de ligne :

rien de ce qui suit '`//`' jusqu'à la fin de la ligne n'est pris en compte par le compilateur

ou les commentaires multi-lignes :

rien de ce qui suit '`/*`' n'est pris en compte par le compilateur jusqu'à ce qu'on rencontre la fin de commentaire '`*/`'.

Exemple.- On peut indiquer le nom du fichier et d'autres commentaires à un de nos programmes antérieurs :

```
// celsius.asml

var C as Integer // degre Celsius
var F as Integer // degre Farenheit
var s as String  // pour la lecture

Main()
  step
    s := ReadLine()
  step
    C := ToInteger(s)
  step
    F := (9*C)/5 + 32
  step
    WriteLine(F)
```

3.4 Le test et l'alternative

Syntaxe.- Le test et l'alternative existent en ASM avec la syntaxe suivante pour AsmL :

`if` *condition* `then` *instruction*

et :

`if` *condition* `then` *instruction1* `else` *instruction2*

Exemple.- Écrivons un programme qui demande de saisir un entier et qui affiche si celui-ci est pair ou impair :

```
var n as Integer
var s as String

Main()
  step
    Write("Entrer un entier : ")
  step
    s := ReadLine()
  step
    n := ToInteger(s)
  step
    if (n mod 2 = 0) then WriteLine("entier pair")
                     else WriteLine("entier impair")
```

où a mod b donne le reste dans la division euclidienne de a par b (comme en langage Pascal).

Signe d'égalité.- On remarquera au passage que le signe d'égalité de AsmL est celui utilisé en mathématiques ou en langage Pascal et non le '==' du langage C.

Le signe (composé) de diséquation est '<>', là encore inspiré du langage Pascal, et non le signe ' !=' du langage C.

3.5 L'itération

Vous avez vu avec les langages de programmation impératifs que vous avez étudiés que l'itération (ou répétition, ou boucle) est une structure de contrôle importante. L'itération prend une forme légèrement différente des autres langages de programmation impératifs dans les ASM.

3.5.1 Forme fondamentale de l'itération

Syntaxe.- L'instruction composée :

`step until fixpoint` *instruction*

effectue au moins une fois l'instruction *instruction* et recommence jusqu'à ce que plus aucune mise à jour ne change la valeur d'au moins une variable.

Exemple 1.- (**Exponentiation**)

L'exponentiation est la première opération, souvent utilisée en pratique, qui n'est pas implémentée dans les langages de programmation. Il est très facile de l'implémenter en utilisant une boucle (contrôlée) POUR. Un tel type de boucle n'existe pas dans les ASM mais il est très facile de l'émuler.

Écrivons un programme AsmL qui calcule 5^3 :

```
var a = 5
var n = 3
var p = 1
var i = 1

Main()
  step until fixpoint
    if i <= n then
                p := p*a
                i := i+1
  step
    WriteLine("p = " + p)
```

En fait on écrira plutôt ce programme AsmL en utilisant des variables locales à la première étape de la façon suivante :

```
var p = 1

Main()
  initially a as Integer = 5
  initially n as Integer = 3
  initially i as Integer = 1
  step until fixpoint
    if i <= n then
                p := p*a
                i := i+1
  step
    WriteLine("p = " + p)
```

On remarquera que nous avons considéré que la variable p ne peut pas être locale car on l'utilise également dans la deuxième étape. En fait, en AsmL, une variable déclarée dans une étape est également connue dans les étapes suivantes.

Exercice.- *Écrire un programme AsmL qui demande un entier naturel a, puis un entier naturel n et qui affiche a^n.*

Exemple 2.- (**Calcul du pgcd**)

Lorsque vous rencontrez Yuri Gurevich pour la première fois et qu'il veut vous convaincre de la puissance des ASM, il commence souvent par vous dire : "Donne-moi un algorithme, n'importe lequel. Je vais te l'écrire en ASM pour te montrer que c'est facile." Comme en général, vous hésitez, il continue par : "Prenons l'exemple de l'algorithme d'Euclide."

Rappelons que l'*algorithme d'Euclide* permet de calculer le pgcd de deux entiers naturels a et b en remarquant que :

$$pgcd(a, 0) = a$$

car tout entier divise zéro et que :

$$pgcd(a, b) = pgcd(b, r),$$

où r est le reste dans la division euclidienne de a par b :

$$a = b.q + r,$$

avec : $r < b$. On itère l'application de la seconde règle jusqu'à ce qu'on puisse appliquer la première.

On peut traduire cet algorithme de deux façons dans un langage impératif tel que le langage C : soit *itérativement* en utilisant une boucle pour traduire l'itération, soit *récursivement*. Cette distinction n'a pas de sens en ASM.

Pour calculer le pgcd de 144 et de 60, on peut utiliser le programme AsmL suivant :

```
var d = 0

Main()
  initially a as Integer = 144
  initially b as Integer = 60
  step until fixpoint
    if b = 0 then
                d := a
              else
                a := b
                b := a mod b
  step
    WriteLine("d = " + d)
```

Exercice.- *Écrire un programme AsmL qui demande un entier naturel a, puis un entier naturel b et qui affiche $pgcd(a, b)$.*

3.5.2 Exemple d'itération infinie

Bien entendu rien ne dit qu'à un certain moment il n'y a aura pas de mise à jour effective et donc que le programme s'arrêtera. C'est d'ailleurs le cas du programme AsmL suivant :

```
Main()
  initially n as Integer = 1
  step until fixpoint
    step
      n := 2*n
    step
      WriteLine(n)
```

qui lève assez rapidement une exception.

Ceci est déjà le cas avec l'utilisation de boucles non contrôlées dans les autres langages impératifs. Rappelons, de plus, que dans tout langage de programmation capable de calculer toutes les fonctions calculables totales, on peut également écrire un programme qui boucle (il s'agit d'un théorème de la théorie de la calculabilité).

3.5.3 Autres formes de l'itération

Syntaxe.- Nous avons déjà vu deux formes de l'itération :

```
step
```

et :

`step until fixpoint` *instruction*

Les ASM n'ont besoin que de la seconde forme. Le langage AsmL implémente cependant deux autres formes :

`step until` *condition instruction*

et :

`step while` *condition instruction*

dont la sémantique est évidente dès que l'on comprend un peu d'anglais.

Réduction du nombre de formes d'itération.- Nous avons laissé sous-entendre que seule la forme fondamentale d'itération est indispensable et que le marquage du séquencement n'est pas nécessaire. On peut en effet réduire les autres formes d'itération à la forme fondamentale. Montrons-le par exemple pour le séquencement

```
Main()
  initially C as Integer = 19
  initially F as Integer = 0
  initially mode = "calcul"
  step until fixpoint
    if mode = "calcul" then
      F := (9*C)/5 + 32
      mode := "affichage"
    if mode = "affichage"
      WriteLine(F)
```

Exercice.- *Montrer que les deux autres formes d'itération se réduisent à la forme fondamentale.*

3.5.4 Forme normale

Un programme est dit sous **forme normale** s'il y a au plus une occurrence d'une des nombreuses formes des instructions `step`, qui est nécessairement un `step until fixpoint` en début de programme.

Exercice.- *Réécrire tous les programmes déjà vus sous forme normale.*

3.6 La modularité

Pour des raisons de présentation des programmes, liées à notre psychologie, il est important d'utiliser des sous-programmes. Ceci n'est pas important pour les ASM elles-mêmes, comme nous le verrons. Cependant AsmL implémente la notion de méthode, comme en langage C (langage dans lequel on parle plutôt de *fonction*), en C++ ou en Java.

Syntaxe.- Une méthode est définie par un en-tête suivi d'un bloc. L'en-tête est de la forme suivante :

nom`(`*parametre1* `as` *Type1*`, ...,` *parametren* `as` *Typen*`)` `as` *Type*

avec le type de retour apparaissant en dernier (contrairement aux langages C, C++ ou Java).

Le bloc contiendra quelques instructions `return` comme dans les langages C, C++ ou Java.

Exemple.- Réécrivons un de nos programmes antérieurs en utilisant une méthode :

```
var n as Integer
var s as String

IsPair(a as Integer) as Boolean
  if (a mod 2 = 0) then return true
                   else return false

Main()
  step
    Write("Entrer un entier : ")
  step
    s := ReadLine()
  step
    n := ToInteger(s)
  step
    if IsPair(n) then WriteLine("entier pair")
                 else WriteLine("entier impair")
```

Exercices

Exercice 1.- (**Moyenne**)

Écrire un programme AsmL qui permet d'entrer un certain nombre de réels positifs, en terminant par la valeur -1. Le programme doit alors afficher le nombre de réels entrés et la moyenne.

[On utilisera la méthode `ToFloat()`.]

Exercice 2.- (**Manipulations sur les chiffres d'un entier**)

Écrire un programme AsmL qui demande un entier, affiche cet entier, chaque chiffre étant séparé par deux espaces, ainsi que le produit des chiffres de cet entier (en base dix).

[Un exemple de session est :

```
Entrer un entier : 954
Cet entier est : 9 5 4
Le produit des chiffres est : 180.
```
]

Exercice 3.- (**Inversion d'entier**)

Écrire un programme AsmL qui demande un entier naturel et affiche ce nombre dans l'ordre inverse.

[Un exemple de session est :

```
Entrer un entier naturel : 1234
Le nombre inversé est : 4321.
```
]

Exercice 4.- (**Suite bitone**)

Une **suite monotone** *d'entiers naturels est une suite croissante ou décroissante. Une* **suite bitone** *est une suite monotone, suivie d'une autre suite monotone (éventuellement vide). Par exemple les trois suites suivantes (1, 3, 5, 2, 4, 8), (2, 2, 3, 5, 3, 2) et (4, 3, 1, 2, 7) sont bitones.*

Écrire un programme AsmL qui demande un certain nombre d'entiers naturels, la valeur sentinelle étant -1 pour terminer la saisie, puis qui affiche si la suite ainsi saisie est bitone ou non.

[*On remarquera qu'il y a changement du sens de croissance si on a au moins trois éléments dans la suite et si la différence entre l'élément en cours et le précédent est de signe différent de la différence du précédent et de l'antépénultien.*]

Exercice 5.- (**Heures supplémentaires**)

Dans une certaine entreprise, les 35 premières heures hebdomadaires des employés sont payées à un certain taux horaire et les heures suivantes avec une minoration de 20%.

Écrire un programme AsmL qui aide à déterminer le salaire brut hebdomadaire de chacun des salariés, en se fiant à la session suivante :

```
Entrez le nombre d'heures travaillées (-1 pour terminer) : 43
Entrez le taux horaire de l'employé (00.00 F) : 83.50
Le salaire est de 3 724.10 F

Entrez le nombre d'heures travaillées (-1 pour terminer) : 34
Entrez le taux horaire de l'employé (00.00 F) : 72.25
Le salaire est de 2 465.50 F

Entrez le nombre d'heures travaillées (-1 pour terminer) : 55
```

```
Entrez le taux horaire de l'employé (00.00 F) : 100.04
Le salaire est de 5 102.04 F

Entrez le nombre d'heures travaillées (-1 pour terminer) : -1
```

Exercice 6.- (**Affichage**)

On veut afficher un losange comme celui ci-dessous :

```
        *
       ***
      *****
     *******
    *********
     *******
      *****
       ***
        *
```

centré à l'écran (de 80 colonnes), ici de hauteur cinq (c'est-à-dire comportant cinq lignes de largeurs différentes).

Écrire un programme AsmL demandant la hauteur (comprise entre 1 et 25) et affichant le losange correspondant à l'écran.

Exercice 7.- (**Monnaie**)

Écrire un programme AsmL d'aide à une caissière pour rendre la monnaie. On entre la somme due puis la somme remise par le client. Le programme calcule la différence et affiche la monnaie que doit rendre la caissière : le nombre d'Euros, le nombre de pièces de 50 centimes, de pièces de 20 centimes, de pièces de 10 centimes, de pièces de 5 centimes, de pièces de 2 centimes et enfin de pièces de 1 centime.

[*Un exemple de session est la suivante* :
Montant : 25.31
Somme remise : 30
Rendre 4 Euros
1 pièces de 50 centimes
0 pièces de 20 centimes
1 pièces de 10 centimes
1 pièces de 5 centimes
2 pièces de 2 centimes
0 pièces de 1 centime.

On utilisera une alternative pour marquer le pluriel ou le singulier de 'Euro'. Par contre 'pièces' sera invariant. On utilisera la division euclidienne pour déterminer le nombre de pièces de chaque sorte.]

Exercice 8.- (**Somme de puissances**)

- 1°) *Le mathématicien français Lagrange a démontré que tout entier naturel s'écrit comme la somme de quatre carrés, par exemple* :

$$34 = 1^2 + 2^2 + 2^2 + 5^2.$$

Écrire un programme AsmL qui demande un entier naturel et qui affiche **toutes** les décompositions de cet entier comme somme de quatre carrés.

- 2^o) *Certains entiers naturels s'écrivent comme somme de cinq puissances quatre, par exemple :*

$$99 = 0^4 + 1^4 + 1^4 + 2^4 + 3^4,$$

et d'autres non, par exemple 6.

Écrire un programme AsmL qui demande un entier naturel et qui affiche **une** décomposition en somme de cinq puissances quatre, s'il en existe au moins une, et 'impossible' sinon.

Exercice 9.- *Écrire un programme qui demande une date, qui vérifie sa cohérence et qui l'affiche avec le nom du mois, suivant les trois exemples de session ci-dessous* :

```
Jour : 3
Mois : 5
3 avril.

Jour : 31
Mois : 2
Quantième incorrect.

Jour : 25
Mois : 19
Mois incorrect.
```

Exercice 10.- (**Nombres parfaits**)

Un entier naturel non nul est dit **parfait** *s'il est égal à la somme de ses diviseurs propres. Par exemple les diviseurs propres de 6 sont 1, 2 et 3, dont la somme est 6, donc 6 est un nombre parfait.*

- 1^o) Écrire une méthode AsmL ayant un argument entier et qui renvoie un entier, la valeur de retour étant la somme des diviseurs propres de la valeur d'entrée.

- 2^o) Écrire une méthode AsmL ayant un argument entier et qui renvoie un booléen, celui-ci étant vrai ou faux selon que la valeur d'entrée est un nombre parfait ou non.

- 3^o) Écrire un programme AsmL qui demande un entier et qui affiche tous les nombres parfaits inférieurs à la valeur entrée.

[*Chaque question doit évidemment utiliser la fonction définie dans la question précédente.*]

Exercice 11.- (**Rotation deux fois à droite**)

- 1^o) *Écrire une méthode AsmL à cinq arguments entiers, disons a, b, c, d, e, qui effectue une rotation à droite deux fois sur ces entiers, c'est-à-dire, qu'après l'appel de la fonction, les valeurs devraient être les anciennes valeurs de d, e, a, b, c.*

- 2^o) *Écrire un programme AsmL qui demande cinq entiers, qui effectue une rotation à droite deux fois sur ces entiers et qui affiche les entiers ainsi obtenus.*

Exercice 12.- (**Jour en lettre**)

- 1^o) *Écrire une méthode d'initialisation (sans argument) qui demande une chaîne de moins de quinze caractères (représentant la langue) et, pour chaque entier compris entre un et sept, une chaîne de moins de dix caractères (représentant le nom du jour en une certaine langue) déclarée comme variable globale.*

- 2^o) *Écrire un programme AsmL de test de cette méthode.*

Exercice 13.- (**Un jeu**)

- 1^o) Écrire une méthode à deux arguments réels et à valeur réelle. Les deux arguments sont un angle (en degré) et une vitesse (en mètre par seconde). La valeur de la méthode est la distance à laquelle retombe un projectile lancé avec cet angle et cette vitesse initiale.

[*On considèrera que la distance parcourue par le projectile est celle lorsqu'on ignore la résistance de l'air, c'est-à-dire :*

$$d = \frac{v^2 * sin(2 * \theta)}{9.81}$$

où v est la vitesse initiale (en mètre par seconde) et θ l'angle initial (en **radian***, et non en degré).*]

- 2^o) Écrire un programme AsmL qui implémente un jeu dans lequel l'utilisateur commence par entrer la distance d'une cible. Il entre ensuite l'angle (en degré) et la vitesse initiale de son canon. Si le projectile parvient à moins de un pour cent de la cible, l'utilisateur gagne. Sinon on lui indique de combien il a manqué la cible et il peut lancer un autre projectile. S'il n'arrive pas à toucher la cible en cinq projectiles, il a perdu.

[*Un exemple de session est* :

```
Distance : 5
Angle : 10
Vitesse : 4
Vous avez manqué la cible de 4.442440 m
Angle : 45
Vitesse : 5
Vous avez manqué la cible de 2.451581 m
Angle : 45
Vitesse : 7
touché. ]
```

Exercice 14.- (**Disparitions**)

- 1^o) *Écrire une méthode AsmL à un argument entier naturel et à valeur entier naturel, qui renvoie l'entier avec un chiffre sur deux de son développement décimal, le chiffre des unités étant présent.*

[On a : f(12345) = 135 et f(123456) = 246.]

- 2^o) *Écrire un programme AsmL qui permet de saisir des entiers naturels et d'afficher à l'écran un chiffre sur deux, le dernier chiffre étant celui des unités. On termine lorsque l'entier saisi est zéro.*

[Un exemple de session est :

```
Entrer un entier : 12345
L'entier modifié est : 135
Entrer un entier : 123456
L'entier modifié est : 246
Entrer un entier : 0
Au revoir. ]
```

Exercice 15.- (**Conversions**)

- 1^o) *Écrire une méthode AsmL* `in2cm()` *dont l'argument et le type de retour sont des réels, qui renvoie la longueur en centimètre alors qu'elle est fournie en pouce.*

[Rappelons que 1 pouce = 2,54 cm.]

- 2^o) *Écrire une méthode AsmL* `cm2in()` *dont l'argument et le type de retour de retour sont des réels, qui renvoie la longueur en pouce alors qu'elle est fournie en centimètre.*

- 3^o) *Écrire un programme AsmL qui demande une longueur, qui demande l'unité ('i' ou 'c') et qui affiche la longueur dans l'autre unité.*

Exercice 16.- (**Conjecture de Goldbach**)

Une célèbre conjecture de théorie des nombres, la **conjecture de Goldbach**, *dit que tout entier naturel pair plus grand que 4 est la somme de deux nombres premiers. Par exemple 16 = 3 + 13. On sait la vérifier sur un grand nombre d'entiers ; nous allons nous aider de l'ordinateur pour la vérifier sur beaucoup d'entiers.*

Rappelons qu'un nombre premier est un entier naturel, différent de 1, qui n'est divisible que par 1 et par lui-même.

- 1^o) Soient $\mathbb{N}$ l'ensemble des entiers naturels et $\mathbb{B}$ l'ensemble des booléens. On note `divide` l'application de $\mathbb{N} \times \mathbb{N}$ dans $\mathbb{B}$ qui, au couple (d, n) d'entiers naturels, associe vrai si d divise n, et faux sinon. Implémenter cette application comme méthode AsmL.

- 2) On note `prime` l'application de $\mathbb{N}$ dans $\mathbb{B}$ qui, à l'entier naturel p, associe vrai si p est un nombre premier et faux sinon. Implémenter cette application comme méthode AsmL.

[*On pourra utiliser l'algorithme naïf consistant à vérifier que les entiers de 2 à p-1 ne divisent pas p.*]

- 3^o) Écrire un programme AsmL qui demande deux entiers naturels a et b, avec $a < b$, et qui liste les entiers pairs compris entre a et b avec une de ses décompositions en la somme de deux nombres premiers.

[*Par exemple pour a = 700 et b = 705, on obtient :*

700 = 17 + 683
702 = 11 + 691
704 = 3 + 701

On n'oubliera pas le cas, certes improbable, où l'on trouve un contre-exemple à la conjecture de Goldbach.

On pourra utiliser l'algorithme naïf qui, pour chaque entier pair n, passe en revue les nombres impairs k compris entre 3 et n et vérifie si k et n − k sont premiers.]

Exercice 17.- (**Altération**)

1^o) *Écrire une méthode* :

```
altere(x,y)
```

qui change la valeur de x en x + y et celle de y en x × y, où x et y sont des réels.

- 2^o) *Tester cette méthode dans un programme AsmL complet.*

Exercice 18.- (**Choix**)

Écrire un programme AsmL qui permet de saisir au clavier un mot d'au plus dix caractères et qui, suivant la valeur du dernier caractère entré :

'A' : trie les caractères dans l'ordre croissant et affiche le mot obtenu ;
'E' : inverse les lettres du mot, puis affiche le mot obtenu ;
'S' : somme les chiffres apparaissant dans le mot et affiche le résultat ;
'X' : lit une autre chaîne de caractères ;
sinon affiche la longueur de la chaîne de caractères.