Cours 1

M.Adam – JF.Kamp – S.Letellier – F.Pouit

23 août 2016

Table des matières

1 Introduction

1.1 Le contenu de la séance

- Notion de programme
- Structure d'un programme
- Variables
- Affectation
- Séquence
- Alternative
- Entrées/Sorties

2 Notion de programme

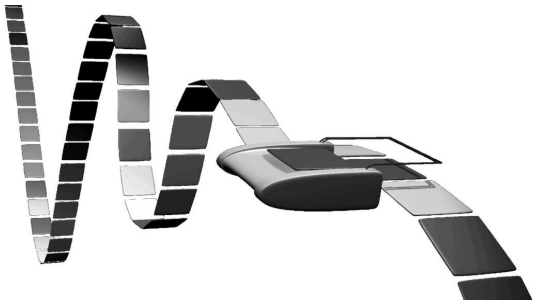
2.1 Définition

Pour wikipedia :

> Un programme est une suite d'instructions qui spécifient étape par étape les opérations à exécuter par un ordinateur.

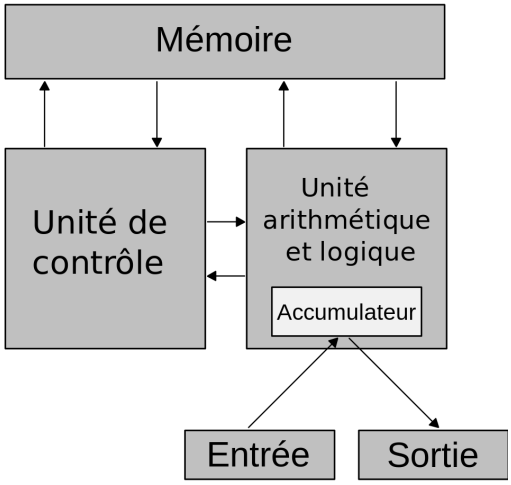
`http://fr.wikipedia.org/wiki/Programme_informatique`

2.2 Machine de Turing



- Un "ruban" divisé en cases consécutives.
- Une "tête de lecture/écriture".
- Un "registre d'état" qui mémorise l'état courant de la machine de Turing.
- Une "table d'actions"

2.3 Architecture Von Neumann



- L'unité arithmétique et logique (UAL ou ALU en anglais) ou unité de traitement.
- L'unité de contrôle, chargée du séquençage des opérations ;
- La mémoire qui contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine).
- Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

2.4 Les bases

Pour concevoir un programme ayant la puissance de calcul d'une machine de Turing, il faut et il suffit :

- des variables,
- la séquence,
- l'affectation,
- l'alternative,
- la boucle.

Tous les langages dit "impératifs" contiennent ces notions. Les autres sont là pour simplifier la vie du programmeur.

2.5 Exécution d'un programme

L'exécution d'un programme passe par :

- un compilateur
- un interpréteur

2.5.1 Compilation

Un programme qui traduit le texte (code source) dans un langage qui permettra son exécution, tel le langage machine, le bytecode ou le langage assembleur.

Première exemple Le fichier source : `calcul.c`

```
#include <stdio.h>
#define TAUX 6.55957

int main () {
   float francs;

   francs=0;
   while (francs<=10) {
      printf("%4.1f francs = %.2f euros\n",francs,francs/TAUX);
      francs=francs+0.5;
   }

   return 0;
}
```

La compilation :

```
> cc calcul.c
> ls -l
total 20
-rwxrwxr-x 1 adam adam 7162 août   8 10:55 a.out
-rw-rw-r-- 1 adam adam  221 août   8 10:54 calcul.c
```

L'exécution :

```
> ./a.out
 0.0 francs = 0.00 euros
 0.5 francs = 0.08 euros
 1.0 francs = 0.15 euros
 ...
 9.5 francs = 1.45 euros
10.0 francs = 1.52 euros
```

Deuxième exemple Le fichier source : `Chaine.java`

```
public class Chaine{

  private String name;

  public Chaine(String name){
      this.name = name;
  }

  public String getName(){
      return(name);
  }

  public void setName (String name){
      this.name = name;
  }

  public static void main(String[] args){
      Chaine ch;
      ch = new Chaine("Bonjour !");
      System.out.println(ch.getName());
      ch.setName("Au revoir !");
      System.out.println(ch.getName());
  }
}
```

La compilation :

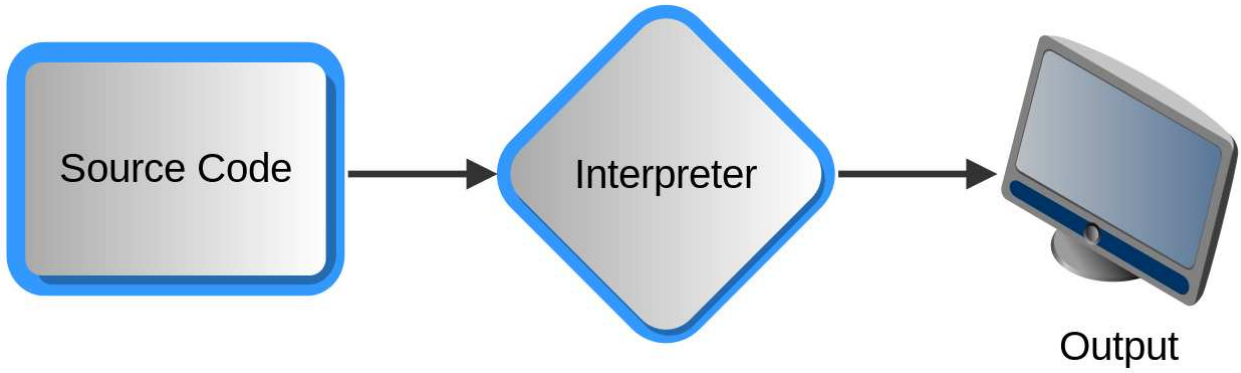
```
> javac Chaine.java
ls -l
total 20
-rw-rw-r-- 1 adam adam  695 août   8 10:39 Chaine.class
-rw-rw-r-- 1 adam adam  423 août   8 10:39 Chaine.java
```

L'exécution :

```
> java Chaine
Bonjour !
Au revoir !
```

2.5.2 Interpréteur

Un programme qui exécute les instructions demandées. Il joue le même rôle qu'une machine qui reconnaîtrait ce langage.




```
> python
Python 2.7.3 (default, Apr 10 2013, 05:46:21)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x=13
>>> resultat=x+2
>>> resultat=resultat*2
>>> print "Quand x vaut",x, ", le résultat vaut",resultat," !"
Quand x vaut 13 , le résultat vaut 30  !
>>>
```

2.5.3 Attention

Tout n'est pas aussi simple :

- un langage compilé peut aussi être interprété
- la compilation est syntaxique ou/et lexicale
- il existe des compilateurs de langages interprétés

3 Structure d'un programme TestAlgo

3.1 TestAlgo

- TestAlgo est un environnement de programmation conçu pour l'apprentissage de la programmation.
- TestAlgo utilise son propre langage impératif.
- TestAlgo pourra servir de langage algorithmique dans les autres matières du DUT informatique.

3.2 Structure du programme TestAlgo

```
##
```

```
# ROLE (Que fait-il ?)
# blabla ...
# @author
##
algo MonAlgo
var
    # Déclaration des variables globales à l'algorithme

    # PRINCIPE (Comment le problème est-il résolu ?)
    # blibli ...
principal
var
    # Déclaration des variables locales au bloc principal
debut
    ...
    instructions ;
    ...
fin
```

3.2.1 Exemple de TestAlgo

```
##
# Calcul de la moyenne en informatique
# à partir de la saisie des moyennes en
# - programmation
# - outils Méthode de Génie Logiciel
# - Architecture, réseau et systême
# @author M.Adam
##
algo CalculMoyenne
principal
var
    reel  moyenneProg;
    reel  moyenneOMGL, moyenneASR;
    reel  moyenneINFO;
debut
    saisir("Moyenne en programmation", @moyenneProg);
    saisir("Moyenne en OMGL", @moyenneOMGL);
    saisir("Moyenne en ASR", @moyenneASR);
    moyenneINFO := (moyenneProg+moyenneOMGL+moyenneASR)/3;
    afficherln("Votre moyenne en informatique est : "+moyenneINFO);
fin
```

3.3 Exécution du programme

En deux étapes :

1. La **compilation** qui vérifie la "syntaxe" du programme.
2. L'**exécution** qui déroule les instructions du programme.

Évidemment un programme qui n'est pas syntaxiquement correct ne peut être exécuté.

3.3.1 Programme correct

- Un programme syntaxiquement correct n'est pas un programme correct.
- Un programme correct est un programme qui produit le résultat attendu.

3.3.2 Un programme non correct

```
##
# Calcul de la moyenne en informatique
# à partir de la saisie des moyennes en
# - programmation
# - outils Méthode de Génie Logiciel
# - Architecture, réseau et systême
# @author M.Adam
##
algo CalculMoyenne
principal
var
    reel  moyenneProg;
    reel  moyenneOMGL, moyenneASR;
    reel  moyenneINFO;
debut
    saisir("Moyenne en programmation", @moyenneProg);
    saisir("Moyenne en OMGL", @moyenneOMGL);
    saisir("Moyenne en ASR", @moyenneASR);
    moyenneINFO := moyenneProg+moyenneOMGL+moyenneASR;
    afficherln("Votre moyenne en informatique est : "+moyenneINFO);
fin
```

Il ne calcule pas correctement la moyenne.

3.3.3 Un programme syntaxiquement faux

```
##
# Calcul de la moyenne en informatique
# à partir de la saisie des moyennes en
# - programmation
# - outils Méthode de Génie Logiciel
# - Architecture, réseau et systême
# @author M.Adam
##
algo CalculMoyenne
principal
var
    reel  moyenneProg;
    reel  moyenneOMGL, moyenneASR;
    reel  moyenneINFO;
debut
    saisir("Moyenne en programmation, @moyenneProg);
    saisir("Moyenne en OMGL, @moyenneOMGL);
```

```
    saisir("Moyenne en ASR, @moyenneASR);
    moyenne := (moyenneProg+moyenneOMGL+moyenneASR)/3;
    afficherln("Votre moyenne en informatique est : "+moyenneINFO);
fin
```

4 Premiers éléments du langage de TestAlgo

4.1 Les variables

Les variables permettent de stocker des valeurs et des résultats de calcul. Une variable est identifiée par un nom et possède un type.

Une type permet d'identifier l'ensemble des valeurs susceptibles d'être stockées dans la variable. A rapprocher de $x \in Z$.

TestAlgo gère 5 types (ensemble de valeurs) de base :

- `entier` : `-1`, `0`, `30`, ...
- `reel` : `-1.12`, `10.5`, `12`,...
- `caractere` : `'a'`, `'b'`, `'='`, `'?'`, ...
- `chaine` : `"chaine"`, `"C"`, ...
- `booleen` : `vrai`, `faux`

4.1.1 Domaines et opérateurs

`entier`	32 bits	-2^{31} à $2^{31}-1$	`+ - * - == > >= < <= <>`
`reel`	64 bits	-10^{45} à $10^{38}-1$	idem
`caractere`	8 bits	`'a'`, `'b'`, `'='`, ...	`== <>`
`chaine`	8	`"chaine"`, ...	`+ == <>`
`booleen`	8	`vrai`, `faux`	`et`, `ou`, `non`

4.1.2 Déclarations

```
var
  "type" nomVar1 [ := valeur initiale ];
  const "type" NOM_CONSTANTE1 := valeur initiale;
```

Une constante `const` est une variable qui ne peut changer de contenu.

4.1.3 Les opérateurs et les types

Par exemple :

```
1 + 2
x / 2
```

+ - *	entier	reel
entier	entier	reel
reel	reel	reel

/	entier	reel
entier	reel	reel
reel	reel	reel

4.1.4 Négation

a	non(a)
vrai	faux
faux	vrai

L'opérateur `non` est prioritaire sur le `ou` et le `et`.

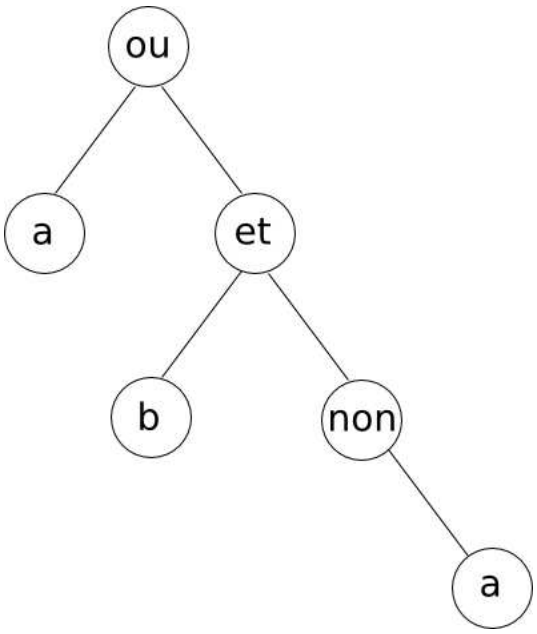
4.1.5 Disjonction et conjonction

a	b	a ou b	a et b
vrai	vrai	vrai	vrai
vrai	faux	vrai	faux
faux	vrai	vrai	faux
faux	faux	faux	faux

- L'opérateur `et` est **prioritaire** sur le `ou`.
- Chaque opérateur est **commutatif**.

4.2 L'évaluation

```
##
# Evaluation d'une expression logique
# @author M.Adam
##
algo OuEtNon
principal
var
    booleen a, b, c;
debut
    a := faux;
    b := vrai;
    c := a ou b et (non a);
    afficherln("c = "+c);
fin
```



Et si `a` et `b` sont `faux` ?

4.2.1 Évaluation totale

Attention : l'évaluation en TestAlgo est totale. Toutes les parties de l'expression sont évaluées et doivent donc être évaluables sans erreur.

Certains langages, ex java, n'évaluent qu'une partie de l'expression en partant de la gauche vers la droite. La commutativité n'est alors plus assurée !

4.3 Les comparaisons

Les opérateurs : `<`, `<=`, `>`, `>=`, `==`, `<>`,

4.3.1 Exemples

```
var
   entier    e1, e2;
   reel      r1, r2;
   caractere c1, c2;
   chaine    ch1, ch2;
...
   e1 > e2;
   r1 == r2;
   e1 == r1;
```

```
c1 <> c2;
ch1 == ch2;
c1 == ch1;
```

4.3.2 Égalité et différence

Les opérateurs == et <> :

	ent	reel	car	chaîne
ent	oui	oui		
reel	oui	oui		
car			oui	oui
chaîne			oui	oui

4.3.3 Infériorité et supériorité

Les opérateurs >, >=, < et <= :

	ent	reel	car	chaîne
ent	oui	oui		
reel	oui	oui		
car				
chaîne				

4.3.4 Les comparaisons de chaînes et de caractères

Il est possible de comparer des chaînes et des caractères.

	==’a’	<>’a’
"a"	VRAI	FAUX
"b"	FAUX	VRAI
"ab"	FAUX	VRAI

4.4 L’affectation

Le signe de l’affectation est :=.

L’affectation permet de donner un contenu à une variable.

```
nbNotes := 3;
somme := note1 + note2;
somme := somme + 10;
```

Remarques

- Il est possible d'affecter un `entier` à une variable de type `réel`,
- Il est possible d'affecter un `caractere` à une variable de type `chaine`.

4.5 La séquence

Une séquence est une suite d'instructions. Les instructions d'une séquence sont séparées par un point-virgule (`;`).

```
saisir("Moyenne en programmation", @moyenneProg);
saisir("Moyenne en OMGL", @moyenneOMGL);
saisir("Moyenne en ASR", @moyenneASR);
```

4.6 Les entrées-sorties

Les entrées-sorties permettent de communiquer avec l'utilisateur du programme.

- `afficher()` et `afficherln()`
- `alaligne()`
- `saisir()`

4.6.1 Affichage

```
afficher("Bonjour, ");
afficher("La valeur de i est "+i);
afficherln(" et celle de j est "+j);
alaligne();
```

4.6.2 Saisie

```
saisir("Donner la valeur de i", @i);
```

Le signe `@` permet d'indiquer que la valeur saisie doit être stockée dans la variable `i`.

5 L'alternative

5.1 Pourquoi une alternative ?

- Avoir des valeurs dans un bon domaine (**inportant pour la sécurité**),
- éviter des erreurs d'exécution,
- résoudre un problème.

5.1.1 Domaines

```
saisir("Note entre 0 et 20, @note);
```

Il faut vérifier que la note est bien dans le domaine.

5.1.2 Erreurs d'exécution

```
my := somme/nbNotes;
```

Il faut vérifier que le nombre de notes est supérieur à zéro.

5.1.3 Résoudre un problème

Dans le calcul d'un quotient familliale :
- un adulte compte pour une part,
- les deux premiers enfants comptent pour une demi-part,
- les enfants suivants comptent pour une part.

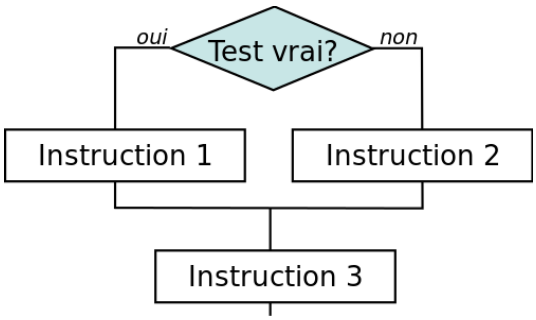
Il faut effectuer un calcul différentiant les enfants.

5.2 Alternative

```
si (booleen condition) alors
    instructions;
[sinon
    instructions;]
finsi
```

Alternative ou conditionnelle.

```
si (Test) alors
    instruction 1;
sinon
    instruction 2;
finsi
instruction 3;
```



5.2.1 Alternative simple

```
si (nbNotes > 0) alors
    my := somme/nbNotes;
finsi
```

5.2.2 Exemple complet

```
##
# Essai d'alternative simple
# @author M.Adam
##
algo AlterSimple
principal
var
    entier nbNotes;
    reel   somme, my;
debut
    saisir("Total des notes",@somme);
    saisir("Nombre de notes",@nbNotes);
    si (nbNotes > 0) alors
        my := somme/nbNotes;
        afficherln("Moyenne = "+my);
    finsi
fin
```

5.2.3 Alternative double

```
si (nbEnfants <= 2) alors
    qf := revenu/(nbAdultes+nbEnfants*0.5);
sinon
    qf := revenu/(nbAdultes+1+nbEnfants-2);
finsi
```

5.2.4 Exemple complet

```
##
# Calcul du Quotient Familiale à partir du revenu, du nombre d'adultes
# et du nombre d'enfants
# @author M.Adam
##
algo QF
principal
var
    entier revenu, nbAdultes, nbEnfants;
    reel    qf;
debut
    saisir("Revenu en Euro",@revenu);
    saisir("Nombre d'adultes",@nbAdultes);
    saisir("Nombre d'enfants",@nbEnfants);

    si (nbEnfants <= 2) alors
        qf := revenu/(nbAdultes+nbEnfants*0.5);
    sinon
        qf := revenu/(nbAdultes+1+nbEnfants-2);
    finsi

    afficherln("Quotient Familliale = "+ qf);
fin
```

6 Les expressions logiques

6.1 Le type booléen

```
booleen trouve;
...
vrai
faux
```

6.2 Les expressions logiques

Un opérateur logique est un opérateur qui, à un ou deux booléens, associe un booléen :

- l'opérateur unaire : `non`
- Les opérateurs binaires : `ou`, `et`

Une expression logique est une suite d'opérateurs logiques et de booléens.

6.3 Alternative et expression logique

Il est possible de réécrire certains codes en utilisant les propriétés de la logique.

Deux programmes sont identiques s'ils produisent toujours la même exécution.

Les codes suivants :

```
si (e1 et e2) alors  | si (e1) alors
   a;                |    si (e2) alors
finsi                |       a;
                     |    finsi
                     | finsi
```

Exécutions identiques ou pas ?

Les codes suivants :

```
si (e1 ou e2) alors  | si (e1) alors
   a;                |    a;
finsi                | finsi
                     | si (e2) alors
                     |    a;
                     | finsi
```

Exécutions identiques ou pas ?

Les codes suivants :

```
si (e1) alors        | si (non e1) alors
   a;                |   b;
sinon                | sinon
   b;                |   a;
finsi                | finsi
```

Exécutions identiques ou pas ?

7 Les tests d'exécution

7.1 Un programme correct

Un code est correct s'il respecte les deux conditions suivantes :
- donne le résultat attendu,
- se termine normalement sans erreur.

7.2 Connaitre le résultat attendu

Il faut connaitre le résultat attendu par l'exécution du programme.

7.2.1 Exemple

Calcul de l'aire d'un cercle :

$R \mapsto R$
$r \to r^2\pi$
$10 \to 314,15$

Que j'aime à faire connaitre ce nombre utile aux sages...

7.3 Tester tous les cas possibles

Il faut dans **tous les cas** que le programme donne un résultat correct. Le programme ne doit jamais rendre un résultat incorrect.

7.3.1 Exemple

```
##
# Calcul du Quotient Familiale à partir du revenu, du nombre d'adultes
# et du nombre d'enfants
# @author M.Adam
##
algo QF
principal
var
    entier revenu, nbAdultes, nbEnfants;
    reel    qf;
debut
    saisir("Revenu en Euro",@revenu);
    saisir("Nombre d'adultes",@nbAdultes);
    saisir("Nombre d'enfants",@nbEnfants);

    si (nbEnfants <= 2) alors
        qf := revenu/(nbAdultes+nbEnfants*0.5);
    sinon
        qf := revenu/(nbAdultes+1+nbEnfants-2);
    finsi

    afficherln("Quotient Familliale = "+ qf);
fin
```

Déterminer tous les cas à tester :

-
-
-
-
-

7.3.2 Attention danger

"Le test de programmes permet de prouver la présence de bugs, non leur absence.

Edsger Dijkstra

8 Conclusion

8.1 Les points à retenir

- Une programme est une suite d'instructions séparées par `;`.
- Une variable permet de stocker des valeurs de types : entier, réel, caractère, chaîne, booléen.
- L'affectation notée `:=` permet de stocker une valeur dans une variable.
- l'alternative exprimée par `si` permet de modifier la séquentialité d'exécution des instructions.
- Les interactions avec l'utilisateur se font par `afficher()`, `afficherln()`, `alaligne()`, `saisir()`.
- La compilation permet de vérifier, entre autre, la syntaxe du programme.
- L'exécution permet de vérifier les erreurs du programme. Un programme qui s'exécute en donnant le bon résultat n'est pas obligatoirement correct.

8.2 A suivre

- Les boucles ou comment répéter plusieurs fois les mêmes instructions