ALGORITHME ET LANGAGE C

LICENCE 1

Dr MAMBE

Objectif et plan du cours

- **Objectif :**
  - Apprendre les concepts de base de l'algorithmique et de la programmation
  - Être capable de mettre en oeuvre ces concepts pour analyser des problèmes simples et écrire les programmes correspondants

- **Plan :** introduction à l'algorithmique et à la programmation
  - Généralités sur l'algorithmique et les langages de programmation
  - Notion de variable, affectation, lecture et écriture
  - Instructions conditionnels et instructions itératives
  - Les Tableaux, les fonctions et procédures, la récursivité
  - Données structurées
  - Initiation au Langage C (Travaux pratiques)

Programme

- Un programme correspond à la description d'une méthode de résolution pour un problème donné.
- Cette description est effectuée par une suite d'instructions d'un langage de programmation
- Ces instructions permettent de traiter et de transformer les données (entrées) du problème à résoudre pour aboutir à des résultats (sorties).
- Un programme n'est pas une solution en soi mais une méthode à suivre pour trouver les solutions.

Langages informatiques

- Un langage informatique est un code de communication, permettant à un être humain de dialoguer avec une machine en lui soumettant des instructions et en analysant les données matérielles fournies par le système.
- Le langage informatique est l'intermédiaire entre le programmeur et la machine.
- Il permet d'écrire des **programmes** (suite consécutive d'instructions) destinés à effectuer une tache donnée
  - Exemple : un programme de résolution d'une équation du second degré
- Programmation : ensemble des activités orientées vers la conception, la réalisation, le test et la maintenance de programmes.

Langages de programmation

- Deux types de langages:
  - Langages procéduraux : **Fortran, Cobol, Pascal, C, …**
  - Langages orientés objets : **C++, Java, C#,…**

- Le choix d'un langage de programmation n'est pas facile, chacun a ses spécificités et correspond mieux à certains types d'utilisations

Notion d'algorithme

- Un programme informatique permet à l'ordinateur de résoudre un problème
  - Avant de communiquer à l'ordinateur comment résoudre ce problème, il faut en premier lieu pouvoir le résoudre nous même
- Un algorithme peut se comparer à une recette de cuisine
  - Le résultat c'est comme le plat à cuisiner
  - Les données sont l'analogues des ingrédients de la recette
  - Les règles de transformations se comparent aux directives ou instructions de la recette

Algorithme informatique

- Un algorithme est une suite d'instructions ayant pour but de résoudre un problème donné. Ces instructions doivent être exécutées de façon automatique par un ordinateur.

Exemples:

- préparer une recette de cuisine
- montrer le chemin à un touriste
- programmer un magnétoscope
- etc ...

Algorithme et programme

- L’élaboration d’un algorithme précède l’étape de programmation
  - Un programme est un algorithme
  - Un langage de programmation est un langage compris par l'ordinateur
- L’élaboration d’un algorithme est une démarche de résolution de problème exigeante
- La rédaction d’un algorithme est un exercice de réflexion qui se fait sur papier
  - L'algorithme est indépendant du langage de programmation
  - Par exemple, on utilisera le même algorithme pour une implantation en Java, ou bien en C++ ou en Visual Basic
  - L’algorithme est la résolution brute d’un problème informatique

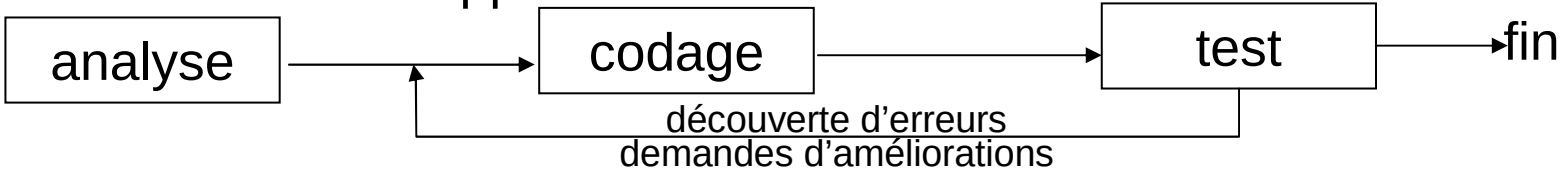
Algorithmique

- algorithme = méthode de résolution
- *algorithme* vient du nom du célèbre mathématicien arabe **Al Khawarizmi** (Abu Ja'far Mohammed Ben Mussa Al-Khwarismi**)**
  - http ://trucsmaths.free.fr/alkhwarizmi.htm
  - http://publimath.irem.univ-mrs.fr/glossaire/AL016.htm

- **L'algorithmique** désigne aussi la discipline qui étudie les algorithmes et leurs applications en Informatique
- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces

algorithmique

- Conception
  - comment développer un algorithme?
  - quelles techniques produisent de bons algorithmes?
- Analyse
  - étant donné un algorithme, quelles sont ses qualités?
  - est-il adapté au problème?
  - est-il efficace?
  - comment mesurer ses performances?
- Étant donné un problème sans solution évidente, comment peut on le résoudre?
  - en considérant les problèmes similaires connus,
  - en considérant les solutions analogues - *algorithmes* - connues,
  - en faisant marcher son imagination !!!
- Processus de développement :



Propriétés d'un algorithme

- Un algorithme doit:
  - avoir un ***nombre fini d'étapes***,
  - avoir un ***nombre fini d'opérations*** par étape,
  - ***se terminer*** après un nombre fini d'opérations,
  - fournir un ***résultat***.
- Chaque opération doit être:
  - ***définie*** rigoureusement et sans ambiguïté
  - ***effective***, c-à-d réalisable par une machine
- Le comportement d'un algorithme est ***déterministe***.

Représentation d'un algorithme

Historiquement, deux façons pour représenter un algorithme:

- L'Organigramme: représentation graphique avec des symboles (carrés, losanges, etc.)
  - offre une vue d'ensemble de l'algorithme
  - représentation quasiment abandonnée aujourd'hui
- Le pseudo-code: représentation textuelle avec une série de conventions ressemblant à un langage de programmation
  - plus pratique pour écrire un algorithme
  - représentation largement utilisée

Algorithmique

Notions et Instructions de base

instructions de base

- Un programme informatique est formé de quatre types d'instructions considérées comme des petites briques de base :
  - l'affectation de variables
  - la lecture et/ou l'écriture
  - les tests
  - les boucles

Notion de variable

- Une **variable** sert à stocker la valeur d'une donnée dans un langage de programmation
- Une variable désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom de variable)
- Chaque emplacement mémoire a un numéro qui permet d'y faire référence de façon unique : c'est l'adresse mémoire de cette cellule.
- Règle : La variable doit être **déclarée** avant d'être utilisée, elle doit être caractérisée par :
  - un nom (**Identificateur**)
  - un **type** qui indique l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, caractère, chaîne de caractères, …)
  - Une valeur

Identificateurs : règles

Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique

  exemple : E1 (1E n'est pas valide)

- doit être constitué uniquement de lettres, de chiffres et du soulignement (« _ ») (Éviter les caractères de ponctuation et les espaces)

  Exemples : SMI2008, SMI_2008

  (SMP 2008, SMP-2008, SMP;2008 : sont non valides)

- doit être différent des mots réservés du langage (par exemple en C: **int, float, double, switch, case, for, main, return**, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Identificateurs : conseils

Conseil: pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées

exemples: NoteEtudiant, Prix_TTC, Prix_HT

Remarque: en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

Types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre. Les types offerts par la plus part des langages sont:

- Type numérique (entier ou réel)
  - **Byte** (codé sur 1octet): de $[-2^7,2^7[$ ou $[0, 2^8[$
  - **Entier court** (codé sur 2 octets) : $[-2^{15},2^{15}[$
  - **Entier long** (codé sur 4 octets): $[-2^{31},2^{31}[$
  - **Réel simple précision** (codé sur 4 octets) : précision d'ordre 10^{-7}
  - **Réel double précision** (codé sur 8 octets) : précision d'ordre 10^{-14}
- Type logique ou booléen: deux valeurs VRAI ou FAUX
- Type caractère**:** lettres majuscules, minuscules, chiffres, symboles,..
  Exemples : 'A', 'b', '1', '?', …
- Type chaîne de caractère: toute suite de caractères
  Exemples: " " , " Nom, Prénom", "code postale: 1000", …

Déclaration des variables

- Rappel: toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable
- En pseudo-code, la déclaration de variables est effectuée par la forme suivante :

  Variables liste d'identificateurs : type

- Exemple:

Variables i, j, k : entier
x, y : réel
OK: booléen
Ch1, ch2 : chaîne de caractères

Variables : remarques

- pour le type numérique, on va se limiter aux entiers et réels sans considérer les sous types
- Pour chaque type de variables, il existe un ensemble d'opérations correspondant.
- Une variable est l'association d'un nom avec un type, permettant de mémoriser une valeur de ce type.

Constante

- Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme, elle peut être un nombre, un caractère, ou une chaine de caractères.
- En pseudo-code, Constante identificateur=valeur : type,…
  (par convention, les noms de constantes sont en majuscules)
- Exemple : pour calculer la surface des cercles, la valeur de pi est une constante mais le rayon est une variable.
  Constante PI=3.14 : réel, MAXI=32 : entier
- Une constante doit toujours recevoir une valeur dès sa déclaration.

Affectation

- **L'affectation** consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire)
- En pseudo-code, l'affectation est notée par le signe ←

  Var← e : attribue la valeur de e à la variable Var

  - e peut être une valeur, une autre variable ou une expression
  - Var et e doivent être de même type ou de types compatibles
  - l'affectation ne modifie que ce qui est à gauche de la flèche

  Exemples : i ←1 j ←i k ←i+j

  x ←10.3 OK ←FAUX ch1 ←"SMI"

  ch2 ←ch1 x ←4 x ←j

  (avec i, j, k : entier; x :réel; ok :booléen; ch1,ch2 :chaine de caractères)
- Exemples **non valides: i ←10.3 OK ←"SMI" j ←x**

Affectation

- Les langages de programmation C, C++, Java, … utilisent le signe égal **=** pour l'affectation ←.

  Remarques :

  - Lors d'une affectation, l'expression de droite est évaluée et la valeur trouvée est affectée à la variable de gauche. Ainsi, A←B est différente de B←A
  - l'affectation est différente d'une équation mathématique :
    - Les opérations x ← x+1 et x ← x-1 ont un sens en programmation et se nomment respectivement incrémentation et décrémentation.
    - A+1 ← 3 n'est pas possible en langages de programmation et n'est pas équivalente à A ← 2
  - Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées.

Syntaxe générale de l'algorithme

Algorithme exemple
    / Lapartiedéclaration del'algorithme *\/*
Constantes *// lesconstantes nécessitent unevaleurdèsleurdéclaration*
    const1←20 : entier
        const2←"bonjour!" : chaîne
Variables *// lesvariables proprement dites*
    var1, var3 : réels
    var2 : chaîne
Début *// corpsdel'algorithme*

    / instructions *\/*

Fin

la séquence des instructions

- Les opérations d'un algorithme sont habituellement exécutées une à la suite de l'autre, en séquence (de haut en bas et de gauche à droite).
- L'ordre est important.
- On ne peut pas changer cette séquence de façon arbitraire.
- Par exemple, *enfiler ses bas puis enfiler ses bottes* n'est pas équivalent à *enfiler ses bottes puis enfiler ses bas*.

Affectation : exercices

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C: **Entier**

Début

A ← 7

B ← 17

A ← B

B ← A+5

C ← A + B

C ← B – A

Fin

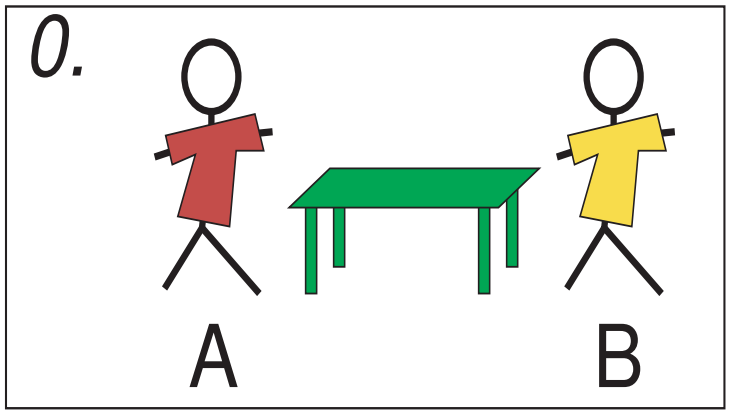
Affection : exercices

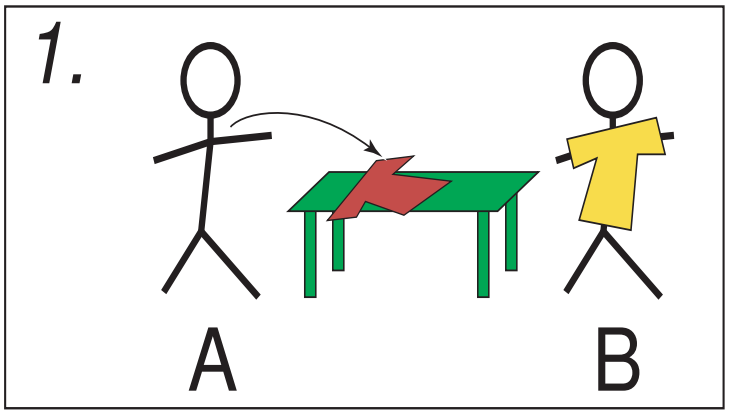
Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

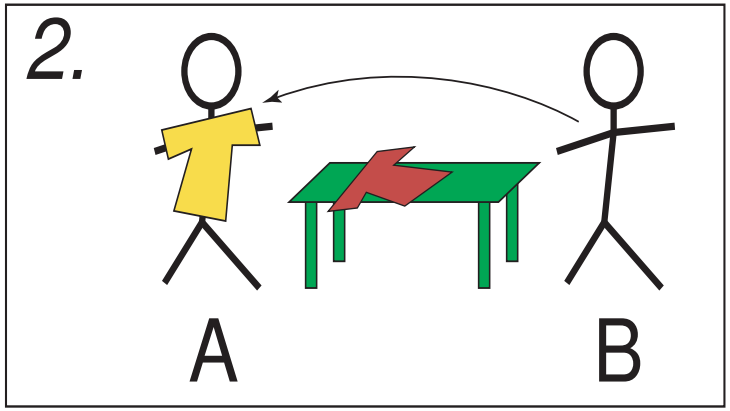
Variables A, B : **Entier**
Début
A ← 6
B ← 2
A ← B
B ← A
Fin

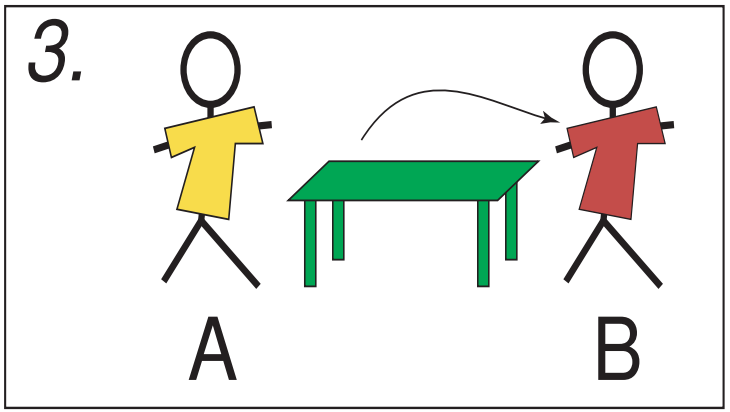
Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Affectation : l'échange des tricots









Affectation : échanges

Écrire un algorithme permettant d'échanger les valeurs de deux variables A et B ?

Réponse :

on utilise une variable auxiliaire C et on écrit les instructions suivantes :

C ← A ; A ← B ; B ← C ;

Expressions et opérateurs

- Une **expression** peut être une valeur, une variable ou une opération constituée de variables reliées par des **opérateurs**

  exemples: 1, b, a*2, a+ 3*b-c, …

- L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération
- Les **opérateurs** dépendent du type de l'opération, ils peuvent être :
  - des opérateurs arithmétiques: +, -, *, /, % (modulo), ^(puissance)
  - des opérateurs logiques: NON(!), OU(| |), ET (&&)
  - des opérateurs relationnels: =, <, >, <=, >=
  - des opérateurs sur les chaînes: & (concaténation)
- Une expression est évaluée de gauche à droite mais en tenant compte des **priorités** des opérateurs**.**

Expression : remarques

- On ne peut pas additionner un entier et un caractère
- Toutefois dans certains langages on peut utiliser un opérateur avec deux opérandes de types différents, c'est par exemple le cas avec les types arithmétiques (4 + 5.5)

- La signification d'un opérateur peut changer en fonction du type des opérandes
  - l'opérateur + avec des entiers effectue l'addition, 3+6 vaut 9
  - avec des chaînes de caractères il effectue la **concaténation** "bonjour" + " tout le monde" vaut "bonjour tout le monde"

Expression : remarques

- Pour le langage C, si x et y sont entiers, x/y est une division entière alors que si l'un des deux ne l'est pas la division est réelle
- x+y/z : est une expression arithmétique dont le type dépend des types de x, y et z
- (x>y) | | !(x=y+1) : est une expression booléenne (| | dénote l'opérateur logique ou et ! Dénote la négation)
- Avant d'utiliser une variable dans une expression, il est nécessaire qu'une valeur lui ait été affectée.
- La valeur de l'expression est évaluée au moment de l'affectation
  - **x ←4**
  - **y ←6**
  - **z ←x+y**
  - **Ecrire(z)** ➔ **10**
  - **y ←20**
  - **Ecrire(z)** ➔ **10 la modification de y après affectation n'a aucun effet sur la valeur de z**

Priorité des opérateurs

- Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :
  - () : les parenthèses
  - ^ : (élévation à la puissance)
  - * , / (multiplication, division)
  - % (modulo)
  - + , - (addition, soustraction)

  exemple: 9 + 3 * 4 vaut 21
- En cas de besoin, on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

  exemple: (9 + 3) * 4 vaut 48
- À priorité égale, l'évaluation de l'expression se fait de gauche à droite

Les opérateurs booléens

- Associativité des opérateurs **et** et **ou**
  a et (b et c) = (a et b) et c
- Commutativité des opérateurs et et ou
  a et b = b et a
  a ou b = b ou a
- Distributivité des opérateurs et et ou
  a ou (b et c) = (a ou b) et (a ou c)
  a et (b ou c) = (a et b) ou (a et c)
- Involution (homographie réciproque) : non non a = a
- Loi de Morgan : non (a ou b) = non a et non b
  non (a et b) = non a ou non b
- Exemple : soient a, b, c et d quatre entiers quelconques :
  (a<b)| |((a>=b)&&(c==d)) ⇔ (a<b)| |(c==d)
  car (a<b)| |(!(a<b)) est toujours vraie

Tables de vérité

C1	C2	C1 et C2	C1 ou C2	C1 XOR C2
Vrai	Vrai	Vrai	Vrai	Faux
Vrai	Faux	Faux	Vrai	Vrai
Faux	Vrai	Faux	Vrai	Vrai
Faux	Faux	Faux	Faux	Faux

C1	Non C1
Vrai	Faux
Faux	Vrai

Les instructions d'entrées et sorties : lecture et écriture

- Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur
- La **lecture** permet d'entrer des donnés à partir du clavier
  - En pseudo-code, on note: **lire (var)**

  la machine met la valeur entrée au clavier dans la zone mémoire nommée var
- Remarque: Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la saisie de l'entrée attendue par le clavier et de la touche Entrée (cette touche signale la fin de l'entrée)
- Conseil: Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

Les instructions d'entrées et sorties : lecture et écriture

- **L'écriture** permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)
  - En pseudo-code, on note: **écrire (liste d'expressions)**

  la machine affiche les valeurs des expressions décrite dans la liste.

  Ces instructions peuvent être des variables ayant des valeurs, des nombres ou des commentaires sous forme de chaines de caractères.
- Exemple : écrire(a, b+2, "Message")

Exemple : lecture et écriture

Écrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le carré de ce nombre

Algorithme Calcul_du_Carre
Rôle : calcul du carre
Données : un entier
Résultats : le carre du nombre
variables A, B : **entier**
Début
    écrire("entrer la valeur de A ")
    lire(A)
    B ← A*A
    écrire("le carre de ", A, "est :", B)
Fin

Exercice : lecture et écriture

Écrire un algorithme qui permet d'effectuer la saisie d'un nom, d'un prénom et affiche ensuite le nom complet

Algorithme AffichageNomComplet

…

variables Nom, Prenom, Nom_Complet : **chaîne de caractères**

Début

écrire("entrez le nom")

lire(Nom)

écrire("entrez le prénom")

lire(Prenom)

Nom_Complet ← Nom & " " & Prenom

écrire("Votre nom complet est : ", Nom_Complet)

Fin

Tests: instructions conditionnelles

- Définition : une condition est une expression écrite entre parenthèse à valeur booléenne.

- Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.

- En pseudo-code :

  Si condition alors

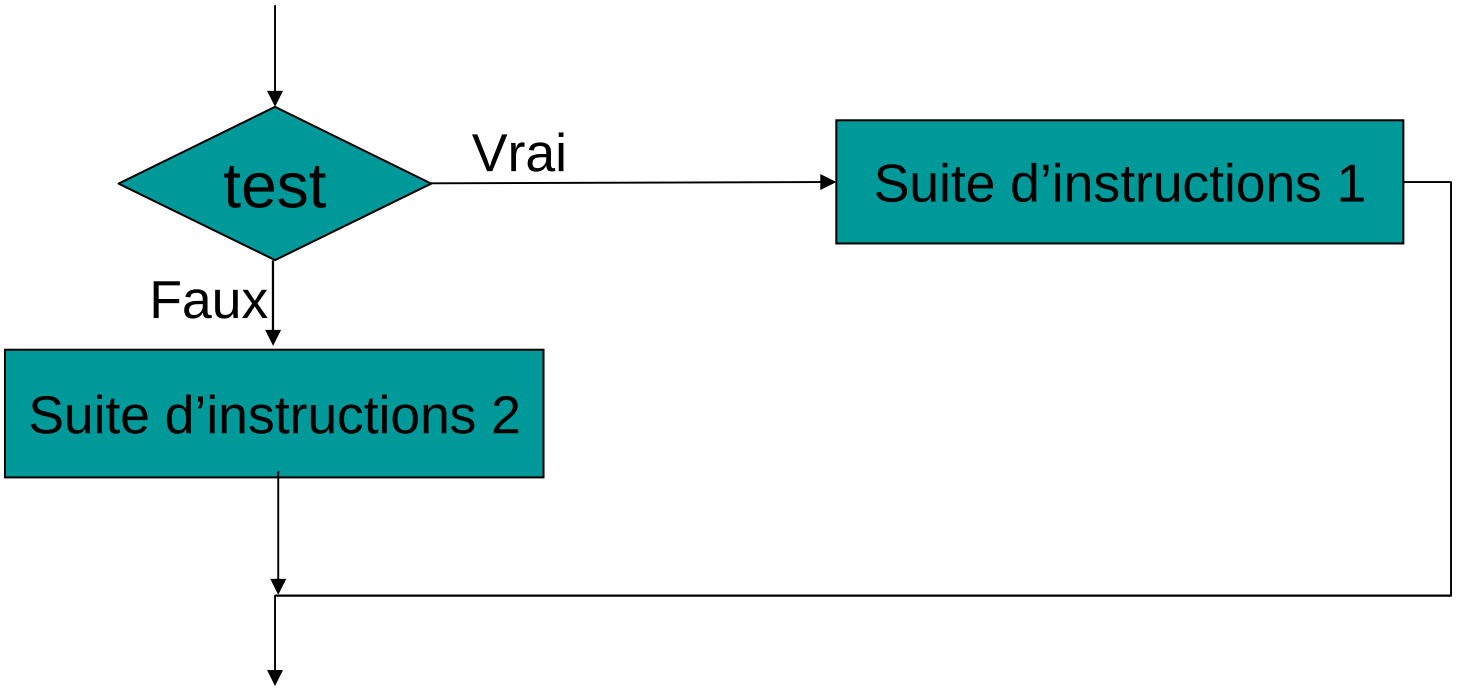
  instruction ou suite d'instructions1

  Sinon

  instruction ou suite d'instructions2

  Finsi

instructions conditionnelles



Instructions conditionnelles

- Remarques :
  - la condition ne peut être que vraie ou fausse
  - si la condition est vraie alors seules les instructions1 sont exécutées
  - si la condition est fausse seules les instructions2 sont exécutées
  - la condition peut être une expression booléenne simple ou une suite composée d'expressions booléennes
- La partie Sinon est optionnelle, on peut avoir la forme simplifiée suivante:

Si condition **alors**

instruction ou suite d'instructions1

Finsi

Si…Alors…Sinon : exemple

```
Algorithme ValeurAbsolue1
Variable x : réel
Début
    Ecrire (" Entrez un réel : ")
    Lire (x)
    Si x < 0 alors
        Ecrire ("la valeur absolue de ", x, "est:",-x)
    Sinon
        Ecrire ("la valeur absolue de ", x, "est:",x)
    Finsi
Fin
```

Si…Alors : exemple

```
Algorithme ValeurAbsolue2
Variable x, y : réel
Début
    Ecrire (" Entrez un réel : " )
    Lire (x)
    y← x
    Si x < 0 alors
            y ← -x
    Finsi
    Ecrire ("la valeur absolue de ", x, "est:",y)
Fin
```

Exercice (tests)

Écrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 7 ou non

```
Algorithme Divsible_par7
Variable n : entier
Début
    Ecrire (" Entrez un entier : ")
    Lire (n)
    Si (n%7=0) alors
        Ecrire (n," est divisible par 7")
    Sinon
        Ecrire (n," n'est pas divisible par 7")
    Finsi
Fin
```

Conditions composées

- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques:
ET, OU, OU exclusif (XOR) et NON
- Exemples :
  - x compris entre 2 et 6 : (x >= 2) ET (x < =6)
  - n divisible par 3 ou par 2 : (n%3=0) OU (n%2=0)
  - deux valeurs et deux seulement sont identiques parmi a, b et c : (a=b) XOR (a=c) XOR (b=c)
- L'évaluation d'une condition composée se fait selon des règles présentées généralement dans ce qu'on appelle tables de vérité

Tests imbriqués

- Les tests peuvent avoir un degré quelconque d'imbrications

```
Si condition1 alors
    Si condition2 alors
        instructionsA
    Sinon
        instructionsB
    Finsi
Sinon
    Si condition3 alors
        instructionsC
    Finsi
Finsi
```

Tests imbriqués : exemple 1

```
....
Variable n : entier
Début
    Ecrire ("entrez un nombre : ")
    Lire (n)
    Si n < 0 alors
        Ecrire ("Ce nombre est négatif")
    Sinon
        Si n = 0 alors   Ecrire ("Ce nombre est nul")
        Sinon   Ecrire ("Ce nombre est positif")
        Finsi
    Finsi
Fin
```

Tests imbriqués : exemple 2

```
Variable n : entier
Début
    Ecrire ("entrez un nombre : ")
    Lire (n)
    Si n < 0 alors Ecrire ("Ce nombre est négatif")
    Finsi
    Si n = 0 alors Ecrire ("Ce nombre est nul")
    Finsi
    Si n > 0 alors Ecrire ("Ce nombre est positif")
    Finsi
Fin
```

Remarque : dans l'exemple 2 on fait trois tests systématiquement alors que dans l'exemple 1, si le nombre est négatif on ne fait qu'un seul test

Conseil : utiliser les tests imbriqués pour limiter le nombre de tests et placer d'abord les conditions les plus probables

Tests imbriqués : exercice

Le prix de disques compacts (CD) dans espace de vente varie selon le nombre à acheter:

200 Francs CFA l'unité si le nombre de CD à acheter est inférieur à 10,

180 Francs CFA l'unité si le nombre de CD à acheter est compris entre

10 et 20 et 160 Francs CFA l'unité si le nombre de CD à acheter est au-delà de 20.

Écrivez un algorithme qui demande à l'utilisateur le nombre de CD à acheter, qui calcule et affiche le prix à payer

Tests imbriqués : corrigé

```
Variables unites : entier
                   prix : réel
Début
    Ecrire ("Nombre d'unités : ")
    Lire (unites)
    Si unites < 10 Alors
           prix ← unites*200
    Sinon
           Si unites < 20  alors
               prix ← unites*180
            Sinon
               prix ← unites*160
            Finsi
     Finsi
     Ecrire ("Le prix à payer est : ", prix)
 Fin
```

L'instruction cas

- Lorsque l'on doit comparer une **même** variable avec plusieurs valeurs, comme par exemple :

```
si a=1 alors instruction1
sinon  si a=2 alors instruction2
          sinon  si a=4 alors instruction4
                    sinon  . . .
                    finsi
          finsi
finsi
```

- On peut remplacer cette suite de si par l'instruction cas

L'instruction cas

- Sa syntaxe en pseudo-code est :

cas où v **vaut**
    v1 : action1
    v2 : action2
    . . .
    vn : actionn
  autre : action autre
fincas

v1,. . . ,vn sont des **constantes** de type **scalaire** (entier, naturel, énuméré, ou caractère)

action i est exécutée si v = vi (on quitte ensuite l'instruction cas)

action autre est exécutée si quelque soit i, v ≠ vi

L'instruction cas : exemple

```
…
Variables c : caractère
Début
Ecrire(«entrer un caractère»)
Lire (c)
Si((c>=’A’) et (c<=‘Z’)) alors
    cas où c vaut
          ‘A’,’E’, ‘I’, ‘O’, ‘U’, ‘Y’ : ecrire(c,"est une voyelle majuscule")
    autre : ecrire(c," est une consonne majuscule ")
    fincas
 sinon ecrire(c,"n’est pas une lettre majuscule")
Finsi
Fin
```

Instructions itératives : les boucles

- Les boucles servent à répéter l'exécution d'un groupe d'instructions un certain nombre de fois

- On distingue trois sortes de boucles en langages de programmation :

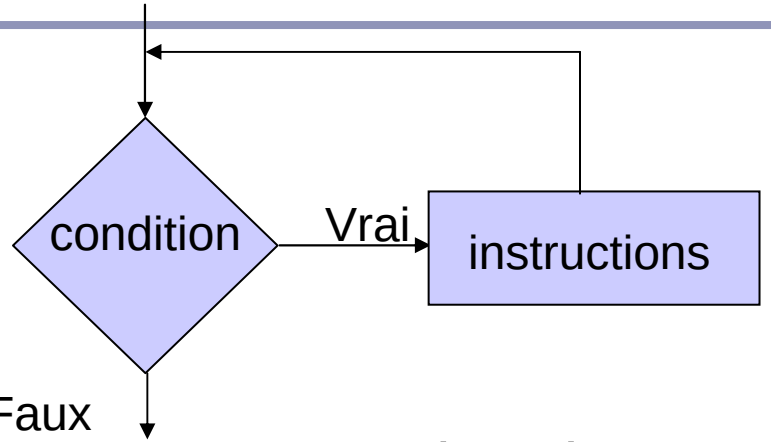
  • Les **boucles tant que** : on y répète des instructions tant qu'une certaine condition est réalisée

  • Les **boucles jusqu'à** : on y répète des instructions jusqu'à ce qu'une certaine condition soit réalisée

  • Les **boucles pour** ou avec compteur : on y répète des instructions en faisant évoluer un compteur (variable particulière) entre une valeur initiale et une valeur finale

Les boucles Tant que

TantQue (condition)
    instructions
FinTantQue

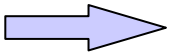


la condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération

- si la condition est vraie, on exécute les instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, …
- si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue
- Il est possible que les instructions à répéter ne soient jamais exécutées.

Les boucles Tant que : remarques

- Le nombre d'itérations dans une boucle TantQue n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de la condition
- Une des instructions du corps de la boucle doit absolument changer la valeur de la condition de vrai à faux (après un certain nombre d'itérations), sinon le programme va tourner indéfiniment



Attention aux boucles infinies

- Exemple de boucle infinie :

```
i ← 1
TantQue i > 0
        i ← i+1
FinTantQue
```

correction

```
i ← 1
TantQue i <100
  i ← i+1
 FinTantQue
```

Boucle Tant que : exemple1

Contrôle de saisie d'une lettre alphabétique jusqu'à ce que le caractère entré soit valable

…

Variable C : caractère

Debut

Écrire (" Entrez une lettre majuscule ")

Lire (C)

TantQue (C < 'A' ou C > 'Z')

Ecrire ("Saisie erronée. Recommencez")

Lire (C)

FinTantQue

Ecrire ("Saisie valable"**)**

Fin

Tant que : exemple2

- En investissant chaque année 1000 F CFA à intérêts composés de 7%, après combien d'années serons nous millionnaire ?

```
Variables capital :réel
          nbAnnees : entier
Debut capital ←0.0      nbAnnes ←0
Tantque (Capital < 1000000)
      capital ← capital+1000;
      nbAnnees++;
      capital ←(1+0.07)*capital;
FinTantque
Fin
```

Boucle Tant que : exemple3

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100

version 1

```
Variables som, i : entier
Debut
        i ← 0
        som← 0
TantQue (som <=100)
        i ← i+1
        som ← som+i
FinTantQue
Ecrire (" La valeur cherchée est N= ", i)
Fin
```

Boucle Tant que : exemple3

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100

forme 2: attention à l'ordre des instructions et aux valeurs initiales

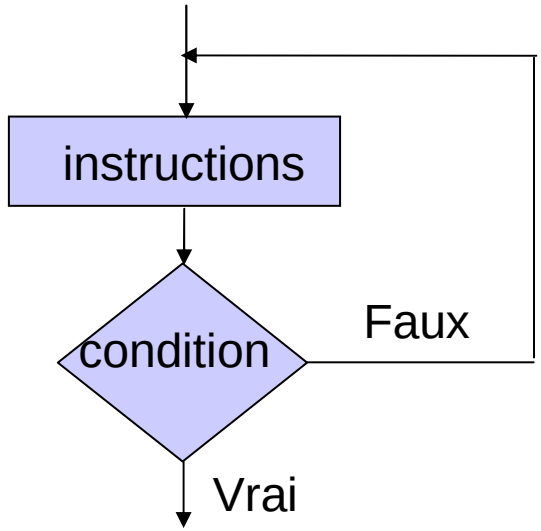
```
Variables som, i : entier
Debut
        som ← 0
        i ← 1
        TantQue (som <=100)
                som ← som + i
                i ← i+1
        FinTantQue
        Écrire (" La valeur cherchée est N= ", i-1)
Fin
```

Les boucles Répéter … jusqu'à …

Répéter

instructions

Jusqu'à condition



- Condition est évaluée après chaque itération
- les instructions entre ***Répéter*** et ***jusqu'à*** sont exécutées au moins une fois et leur exécution est répétée jusqu'à ce que la condition soit vraie (tant qu'elle est fausse)

Boucle Répéter jusqu'à : exemple 1

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100 **(version avec répéter jusqu'à)**

```
Variables som, i : entier
Debut
        som ← 0
        i ← 0
        Répéter
                i ← i+1
                som ← som+i
        Jusqu'à ( som > 100)
        Ecrire (" La valeur cherchée est N= ", i)
Fin
```

Boucle Répéter jusqu'à : exemple 2

Ecrire un algorithme qui compte le nombre de bits nécessaires pour coder en binaire un entier n.

Solution :

```
Variables i, n, nb : entiers
Debut
        Ecrire(" Entrer la valeur de n :")
        lire(n)
        i ← n
        nb ←  0
        Répéter
                i ← i/2
                nb ← nb + 1
        jusqu'à (i=0)
        Ecrire("Pour coder ",n," en binaire il faut ",nb, "bits")
Fin
```

Les boucles Tant que et Répéter jusqu'à

- Différences entre les boucles Tant que et Répéter jusqu'à :

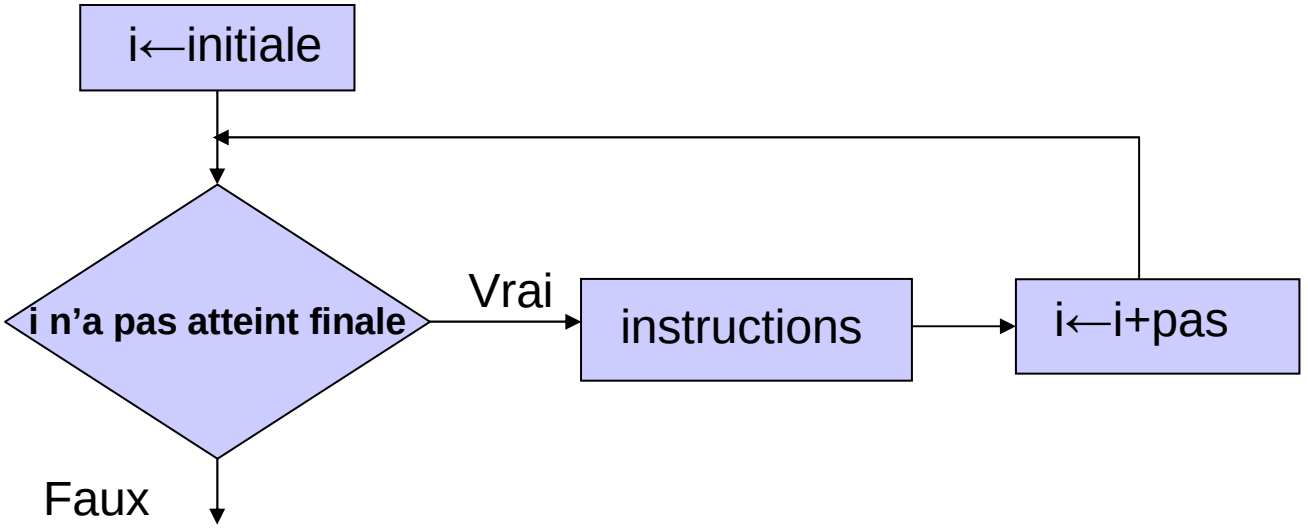
  - la séquence d'instructions est exécutée au moins une fois dans la boucle Répéter jusqu'à, alors qu'elle peut ne pas être exécutée dans le cas du Tant que.

  - la séquence d'instructions est exécutée si la condition est vraie pour Tant que et si la condition est fausse pour Répéter jusqu'à.

  - Dans les deux cas, la séquence d'instructions doit nécessairement faire évoluer la condition, faute de quoi on obtient une boucle infinie.

Les boucles Pour

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas
    instructions
FinPour



Les boucles Pour

- Remarque : le nombre d'itérations dans une boucle Pour est connu avant le début de la boucle
- **Compteur** est une variable de type entier (ou caractère). Elle doit être déclarée
- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à finale - initiale+ 1
- **Initiale** et **finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

Déroulement des boucles Pour

1) La valeur initiale est affectée à la variable compteur

2) On compare la valeur du compteur et la valeur de finale :

a) Si la valeur du compteur est > à la valeur finale dans le cas d'un pas positif (ou si compteur est < à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour

b) Si compteur est <= à finale dans le cas d'un pas positif (ou si compteur est >= à finale pour un pas négatif), instructions seront exécutées

i. Ensuite, la valeur du compteur est incrémentée de la valeur du pas si pas est positif (ou décrémenté si pas est négatif)

ii. On recommence l'étape 2 : La comparaison entre compteur et finale est de nouveau effectuée, et ainsi de suite …

Boucle Pour : exemple 1 (forme 1)

Calcul de x à la puissance n où x est un réel non nul et n un entier positif ou nul

```
…
Variables x, puiss : réel
                     n, i : entier
Debut
        Ecrire (" Entrez respectivement les valeurs de x et n ")
        Lire (x, n)
        puiss ← 1
        Pour i allant de 1 à n
                    puiss← puiss*x
        FinPour
        Ecrire (x, " à la puissance ", n, " est égal à ", puiss)
Fin
```

Boucle Pour : exemple1 (forme 2)

Calcul de x à la puissance n où x est un réel non nul et n un entier positif ou nul (forme 2 **avec un pas négatif**)

```
Variables x, puiss : réel
                     n, i : entier
Debut
    Ecrire (" Entrez respectivement les valeurs de x et n ")
    Lire (x, n)
    puiss ← 1
    Pour i allant de n à 1 par pas -1
            puiss← puiss*x
    FinPour
    Ecrire (x, " à la puissance ", n, " est égal à ", puiss)
Fin
```

Boucle Pour : remarques

- Il faut éviter de modifier la valeur du compteur (et de finale) à l'intérieur de la boucle. En effet, une telle action :

  • perturbe le nombre d'itérations prévu par la boucle Pour
  • rend difficile la lecture de l'algorithme
  • présente le risque d'aboutir à une boucle infinie

  Exemple : **Pour** i **allant de** 1 **à** 5
  i ←i -1
  écrire(" i = ", i)
  Finpour

Lien entre Pour et TantQue

La boucle Pour est un cas particulier de Tant Que (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être remplacé avec TantQue (la réciproque est fausse)

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas
    instructions
FinPour

peut être remplacé par :
(cas d'un pas positif)

compteur ← initiale
TantQue compteur <= finale
        instructions
        compteur ← compteur+pas
FinTantQue

Lien entre Pour et TantQue: exemple 1

Calcul de x à la puissance n où x est un réel non nul et n un entier positif ou nul (forme avec TantQue)

```
Variables x, puiss : réel
                  n, i : entier
Debut
    Ecrire (" Entrez respectivement les valeurs de x et n ")
    Lire (x, n)
    puiss ← 1, i ← 1
    TantQue (i<=n)
              puiss← puiss*x
              i ← i+1
    FinTantQue
    Ecrire (x, " à la puissance ", n, " est égal à ", puiss)
```

Boucles : exercice

Ecrire un algorithme qui vous affiche le nombre de 0 dans un nombre entier n, entré au clavier et constitué de 7 chiffres.

Boucles imbriquées

- Les instructions d'une boucle peuvent être des instructions itératives. Dans ce cas, on aboutit à des **boucles imbriquées**

- **Exemple:**

```
Pour i allant de 1 à 5
        Pour j allant de 1 à i
                écrire("O")
        FinPour
        écrire("K")
FinPour
```

Exécution

OK
OOK
OOOK
OOOOK
OOOOOK

Choix d'un type de boucle

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser *la boucle Pour*

- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des *boucles TantQue ou répéter jusqu'à*

- Pour le choix entre *TantQue* et *jusqu'à :*
  - Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera *TantQue*

  - Si la valeur de la condition de contrôle dépend d'une première exécution des instructions de la boucle, on utilisera *répéter jusqu'à*