

RI.01 - INITIATION AU DÉVELOPPEMENT

A. Casali

PLAN

A. Info diverses



B. C'est quoi un algo ?

C. Premiers exemples

D. Types d'action

E. Schéma alternatif

F. Schéma alternatif en cascade

G. Schéma de choix

H. Expression logique complexe

A. Info diverses

A.1 Contact Informaix

<https://discord.gg/chqTEBhzsC>

A.2 Clothes'Inofrmaix



L'ÉQUIPE INFORM'AIX EST FIÈRE DE VOUS PRÉSENTER:

INFORM'AIX
Clothes

POSSIBILITÉ D'ACHAT:
-PAYPAL
-ESPÈCE

VOS T-SHIRTS
14.99€

T-SHIRT IMPRIMÉS AVEC UN DESIGN
SIMPLE ET UNIQUE



PROMO !

POUR LES 15 PREMIÈRES COMMANDES:
LE PULL À **24.99€** !
LE T-SHIRT À **12.99€** !

14.99 + 29.99 = ~~44.98€~~
COMMANDE TON PULL
ET TON T-SHIRT POUR UN
COUP TOTAL DE 40€ !

VOS PULLS
29.99€

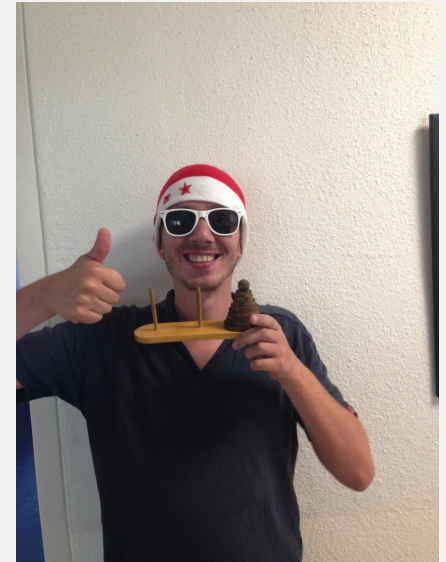
PULL BRODÉS POUR ASSURER
UNE QUALITÉ AU RENDEZ-VOUS !



A. Info diverses

A.3 Ma personne

- Site web : ens.casali.me
- Mail : alain.casali@univ-amu.fr
- Discord : <https://discord.gg/DCNdaHwEeB>
 - Voir section #règles pour vous nommer



A.4 Discord du Dept

<https://discord.gg/WaQj25NrPX>

Utile pour participer en amphi (lien [strawpool](#))

A.5 Qu'est ce qui ne va pas si vous voyez :



Tout va bien, c'est Marc Laporte



Appeler Obi Wan Kenobi

A. Info diverses

A.6 Planning

Disponible : <http://ens.casali.me/category/rl-01-init-dev/rl-01-planning/>

NB: les hyperliens ne sont pas à jour

A.7 Qui va vous faire cours

- Amphi : moi
- Autres : ~10 professionnels (ONU, UBISOFT, SSII, ...)
 - Voir votre emploi du temps

A. Info diverses

A.7 Notation – *temporaire*

- Coef : ~ 20% sur le SI
- Dans la ressource (matière) :
 - Minitests : coef 1;
 - Test#1 : coef 1;
 - Test#2 : coef 1;
 - SAE#1 : coef 0,25;
 - SAE#2 : coef 0,25;

A.8 Minitest & test (à blanc)

- Un mini test c'est quoi ?
- Un mini test comment ça fonctionne ?
- Date du test à *blanc*;
- Date des tests;
- Cheat Code!!

A. Info diverses

A.9 Division de la ressource en 2 parties

1. Partie algorithmique;
2. Partie programmation.

A.10 Tuteurat & ORE

PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

B. C'est quoi un algo ?

(voir fr.wikipedia.org/wiki/Algorithmique)

B.1 Origine des mots

XIII^{ème} siècle, de la traduction en latin d'un mémoire de
Mohammed Ibn Musa Abu Djefar (IX^{ème} siècle) dit

Al-Khuwarismi

(voir fr.wikipedia.org/wiki/Al-Khuwarizmi)

B.2 Définition

Séquence d'opérations visant à la résolution d'un problème en un temps fini
(mentionner la condition d'arrêt)

B. Définition

(voir fr.wikipedia.org/wiki/Algorithmique)

B.3 Mon problème

- Votre capacité de réflexion :



- Exemple(s) :



- ~~ 50% des étudiants n'ont pas la moyenne au test de positionnement de maths :'(

PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

C. Premiers exemples

C.1 Faire un sandwich choco-banane

- Humain : tartiner une tranche de pain avec du Nutella sur une face;
- Ordinateur :
 1. Ai-je du pain ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 2. Ai-je du Nutella ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 3. Ai-je des bananes ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 4. Tant que (1, 2 & 3) ne sont pas satisfait rester ici!
 5. Si l'opercule du Nutella est fermée
 - L'ouvrir avec un couteau
 6. Répartir uniformément le Nutella sur une face d'une tranche de pain;
 7. ...

C. Premiers exemples

C.2 Il se passe quoi si ...

- Vous n'avez ni
 - Pain
 - Nutella
 - Banane



```
Un problème a été détecté et windows a été arrêté afin de prévenir tout dommage
sur votre ordinateur.

Si vous voyez cet écran d'erreur d'arrêt pour la première fois,
redémarrez votre ordinateur. Si cet écran apparaît encore, suivez
ces étapes :

Recherchez tout virus sur votre ordinateur. Supprimez tout disque dur
ou contrôleur de disque dur nouvellement installé. Vérifiez votre disque dur
afin de vous assurer qu'il est correctement configuré et terminé.
Exécutez CHKDSK /F pour vérifier la présence d'un dommage sur votre disque dur
puis redémarrez votre ordinateur.

Informations techniques :

*** STOP: 0x0000007B (0xF78CB528,0xC0000034,0x00000000,0x00000000)
```

Quelque soit votre système d'exploitation (Win XX, Linux XX,
OSX XX)

C. Premiers exemples

C.3 Règle d'or #1

Tout algorithme est faux, mais il fait ce qu'on lui demande – enfin on l'espère !

➡ Nombre de failles 0 day découvertes quotidiennement

C. Premier exemple

C.4 Epic fails



RDV Apollo & Soyouz

*RDV presque manqué parce que les russes parlaient en
mètre, les état-uniuniens en pouce*

C.5 Documentation en anglais

- https://fr.wikipedia.org/wiki/Course_%C3%A0_l%27espace
- https://en.wikipedia.org/wiki/Space_Race

PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

D. Types d'action

D.I Action *simple / imperative*



D. Types d'action

D.2 Action conditionnelle

C. Premiers exemples

C.1 Faire un sandwich choco-banane

- Humain : tartiner une tranche de pain avec du Nutella sur une face;
- Ordinateur :

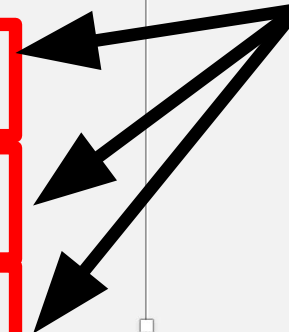
1. Ai-je du pain ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;

2. Ai-je du Nutella ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;

3. Ai-je des bananes ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;

4. Tant que (1, 2 & 3) ne sont pas satisfait rester ici!
5. Si l'opercule du Nutella est fermée
 - L'ouvrir avec un couteau
6. Répartir uniformément le Nutella sur une face d'une tranche de pain;
7. ...

Action
conditionnelle



D. Types d'action

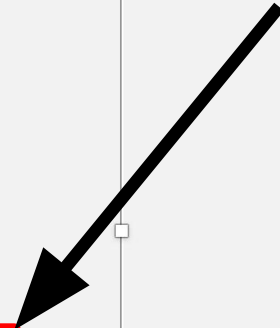
D.3 Action *répétitive*

C. Premiers exemples

C.1 Faire un sandwich choco-banane

- Humain : tartiner une tranche de pain avec du Nutella sur une face;
- Ordinateur :
 1. Ai-je du pain ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 2. Ai-je du Nutella ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 3. Ai-je des bananes ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 4. Tant que (1, 2 & 3) ne sont pas satisfaits rester ici!
 5. Si l'opercule du Nutella est fermée
 - L'ouvrir avec un couteau
 6. Répartir uniformément le Nutella sur une face d'une tranche de pain;
 7. ...

Action *répétitive*



D. Types d'action

D.4 Action *complexe*

C. Premiers exemples

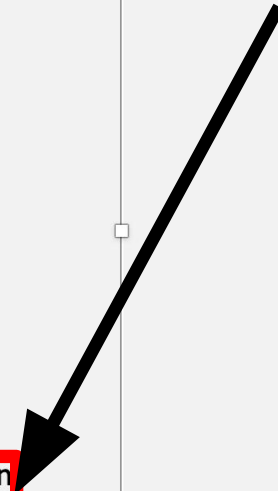
C.1 Faire un sandwich choco-banane

- Humain : tartiner une tranche de pain avec du Nutella sur une face;
- Ordinateur :
 1. Ai-je du pain ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 2. Ai-je du Nutella ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 3. Ai-je des bananes ?
 - Si non en acheter ...
 - Si oui, aller au point suivant;
 4. Tant que (1, 2 & 3) ne sont pas satisfait rester ici!
 5. Si l'opercule du Nutella est fermée
 - L'ouvrir avec un couteau
 6. Répartir uniformément le Nutella sur une face d'une tranche de pain
 7. ...

Combinaison des toutes les autres actions :

- Simple : répartir le Nutella
- Répétitive : répartir le Nutella uniformément
- Conditionnelle : il se passe quoi si on n'a plus de Nutella ?

Action *simple*



PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

E. SCHÉMA ALTERNATIF

Coller du tube PVC

Fonction

(voir [www.mr-bricolage.fr/
?cat=espconseil&page=7&chapitre=Plomberie&choix=386&livre=bricolage](http://www.mr-bricolage.fr/?cat=espconseil&page=7&chapitre=Plomberie&choix=386&livre=bricolage))

Le tube PVC est devenu le matériau courant des évacuations
d'eau
Une canalisation est constituée ...

Commentaires

1. Nettoyer les surfaces à coller avec du papier de verre fin
2. Eventuellement les dégraisser au trichloréthylène
3. Etaler de la colle sur les deux surfaces
4. Emboîter les deux éléments bien à fond et dans la position souhaitée. La colle sèche en quelques minutes

Coller du tube PVC

Objectif

(voir www.mr-bricolage.fr/

?cat=espconseil&page=7&chapitre=Plomberie&choix=386&livre=bricolage)

1. Nettoyer les surfaces à coller avec du papier de verre fin
2. **Eventuellement** les dégraisser au trichloréthylène ;
3. Etaler de la colle sur les deux surfaces ;
4. Emboîter les deux éléments bien à fond et dans la position souhaitée ; // La colle sèche en quelques minutes

Marque de commentaire(s). Tout ce qui suit n'est pas pris en compte. Aide le développeur!

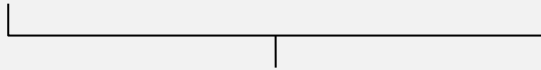
Instruction(s) **impérative(s)**

Instruction **conditionnelle**

Marqueur de fin
d'instruction

Quelle est la condition ?

si les surfaces sont grasses, alors les dégraisser ...



expression logique (vrai ou faux)

et sinon ?

sinon RIEN !

Cas général :

- plusieurs actions à enchaîner si condition remplie
- plusieurs **autres** actions à enchaîner dans le cas contraire
- dans tous les cas reprendre ensuite le traitement normal

En général:

Action1;

Action2;

A faire dans tous les cas

Action3;

Action4;

A faire si **une certaine** condition
est vraie

Action5;

Action6;

A faire si **cette même** condition
est fausse

Action7;

A faire dans tous les cas

Exemple :

```
VerifierVie;  
VerifierArmes;
```

A faire dans tous les cas

```
AttaquerBoss;  
  
...;
```

A faire si **une certaine** condition est vraie

```
PrendrePotion;  
  
...;
```

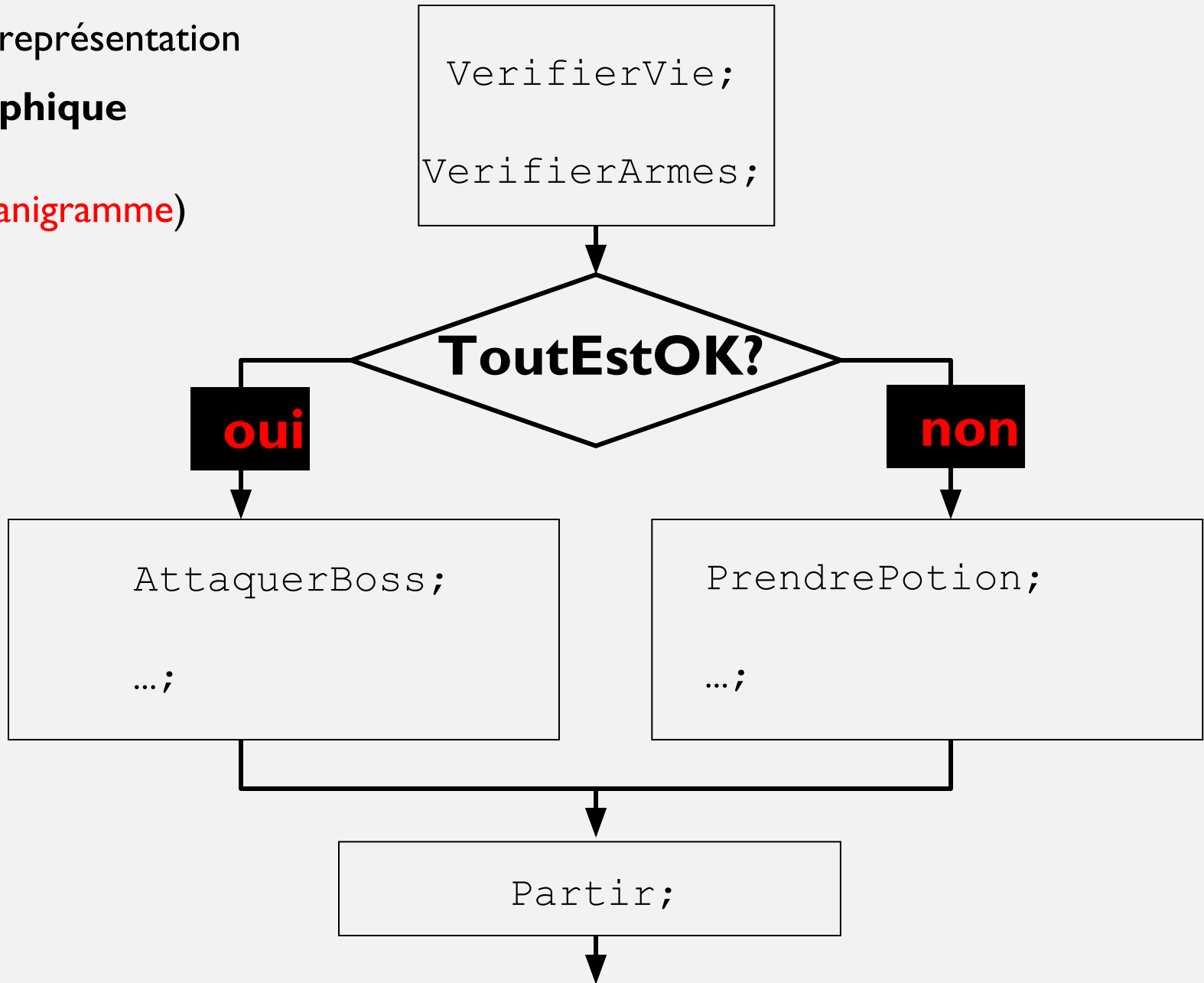
A faire si **cette même** condition est fausse

```
Partir;
```

A faire dans tous les cas

E.I représentation
graphique

(organigramme)



E.2 Représentation **textuelle**

Action1;

Action2;

si (une condition **est** vérifiée)

Action3;

Action4;

si (**cette** condition **n'est pas** vérifiée)

Action5;

Action6;

dans tous les cas

Action7;

```
Action1;  
Action2;
```

```
si (une condition est vérifiée)  
    Action3;  
    Action4;
```

```
si (cette condition n'est pas  
    vérifiée)  
    Action5;  
    Action6;
```

```
dans tous les cas
```

```
Action7;
```

```
VerifierVie;  
VerifierArmes;
```

```
si (ToutEstOK)  
    AttaquerBoss;  
...
```

```
si (Non ToutEstOK)  
    PrendrePotion;  
...
```

```
dans tous les cas
```

```
Partir;
```

E.3 Autre représentation **textuelle**

```
VerfifierVie;  
VerifierArmes;
```

```
si (ToutEstOK)  
    AttaquerBoss;
```

...

```
si (Non ToutEstOK)  
    PrendrePotion;
```

...

dans tous les cas

```
Partir;
```



```
VerfifierVie;  
VerifierArmes;
```

```
si (ToutEstOK)  
    AttaquerBoss;  
...
```

sinon

```
    PrendrePotion;  
...
```

fsi

```
Partir;
```

Décalage = **Indentation**

VerfierVie;
VerfierArmes;

si (ToutEstOK)

AttaquerBoss;
...

sinon

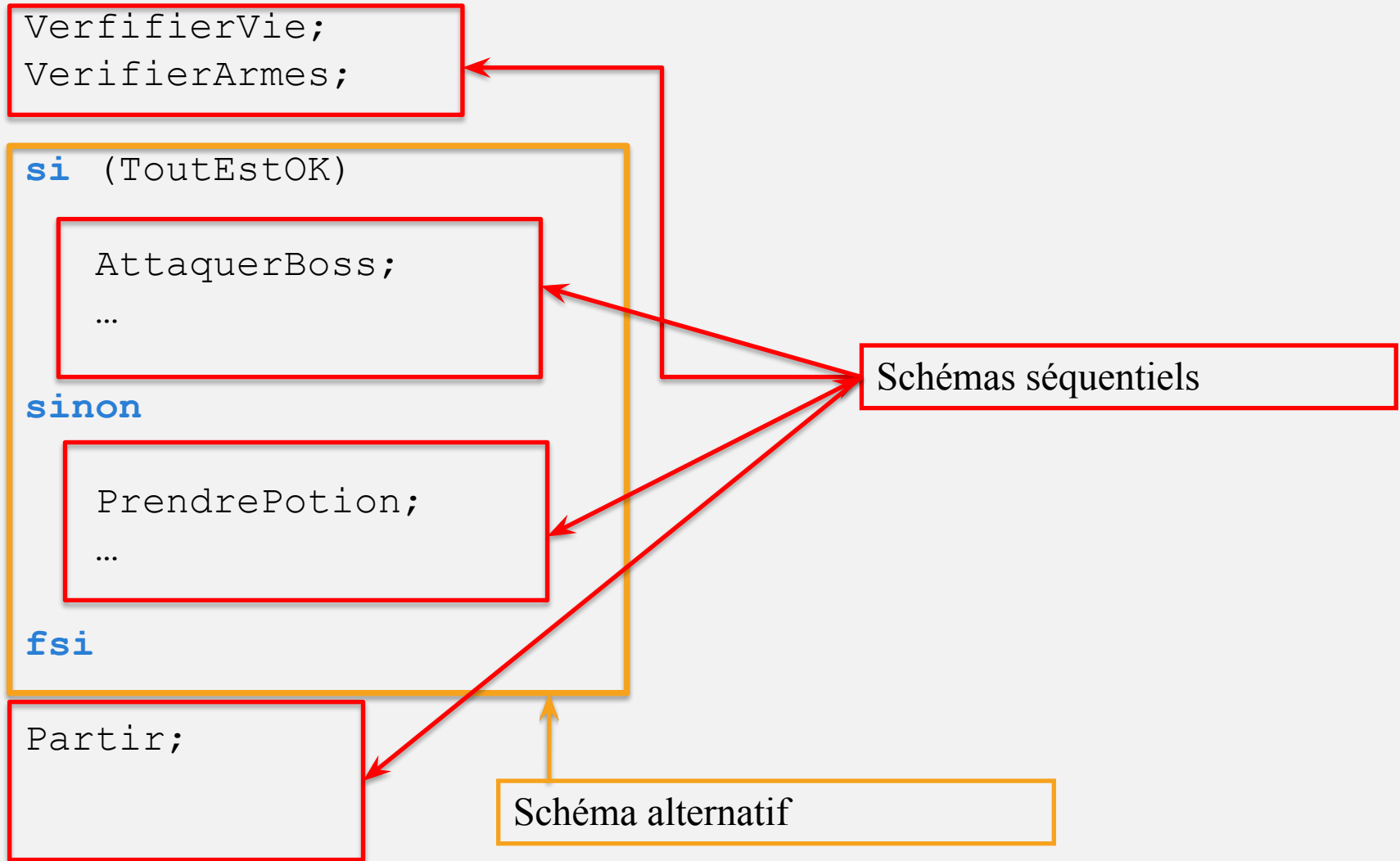
PrendrePotion;
...

fsi

Partir;

Schémas séquentiels

Schéma alternatif



E.4 Cas particulier : absence de la condition **sinon**

```
Action1;
```

```
Action2;
```

A faire dans tous les cas

```
Action3;
```

```
Action4;
```

A faire si **une certaine** condition
est vraie

```
Action7;
```

A faire dans tous les cas

Exemple :

```
VerfierVie;  
VerfierArmes;
```

```
si (ToutEstOK)  
    AttaquerBoss;
```

```
...
```

```
fsi
```

```
Partir;
```

E.5 En résumé

séquence d'instructions alignées verticalement (même indentation)

schéma alternatif :

si (condition)

sinon

fsi

si (condition)

fsi

condition = expression booléenne (vraie ou fausse)

entre

parenthèses

indentation de la séquence d'instructions dans les blocs "**alors**" et "**sinon**"

PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

F. SCHÉMA ALTERNATIF EN CASCADE

```
si (Expr_log_1)  
    Sequ_1;
```

```
sinon
```

```
    si (Expr_log_2)  
        Sequ_2;
```

```
    sinon
```

```
        si (Expr_log_3)  
            Sequ_3;
```

```
        sinon
```

```
            Sequ_4;
```

```
        fsi
```

```
    fsi
```

```
fsi
```

Différentes expressions
logiques



3 niveaux d'imbrication => 3 niveaux d'indentation

=> 3 fois plus pénible à gérer => 3 fois plus illisible que du code « *normal* »

Exemple :

```
si (PeuDeVie)
    PrendreUnePotion
sinon
    si (JeSuisUnGerrier)
        AttaquerAvecEpee;
    sinon
        si (JeSuisUnMage)
            LancerUnSort;
        sinon
            Partir;
    fsi
fsi
fsi
```

expressions logiques
différentes

```
si (Expr_log_1)
    Sequ_1;
sinon
    si (Expr_log_2)
        Sequ_2;
    sinon
        si (Expr_log_3)
            Sequ_3;
        sinon
            Sequ_4;
        fsi
    fsi
fsi
```

3 niveaux d'imbrication =>
3 niveaux d'indentation

```
si (Expr_log_1)
    Sequ_1;
sinon_si (Expr_log_2)
    Sequ_2;
sinon_si (Expr_log_3)
    Sequ_3;
sinon // tous autres cas
    Sequ_4;
fsi
```

3 niveaux d'imbrication =>
1 niveau d'indentation =>
3 fois plus lisible

Exemple :

```
si (PeuDeVie)
    PrendreUnePotion
```

```
sinon
```

```
    si (JeSuisUnGuerrier)
        AttaquerAvecEpee;
```

```
    sinon
```

```
        si (JeSuisUnMage)
            LancerUnSort;
```

```
        sinon
```

```
            Partir;
```

```
        fsi
```

```
    fsi
```

```
fsi
```

```
si (PeuDeVie)
    PrendreUnePotion
```

```
sinon_si (JeSuisUnGuerrier)
    AttaquerAvecEpee;
```

```
sinon_si (JeSuisUnMage)
    LancerUnSort;
```

```
sinon
```

```
    Partir;
```

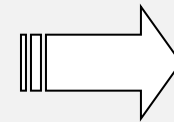
```
fsi
```


PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

G. SCHÉMA DE CHOIX

```
si (Variable_de_choix ∈ Ens_1)
    Sequ_1;
sinon_si (Variable_de_choix ∈ Ens_2)
    Sequ_2;
sinon_si (Variable_de_choix ∈ Ens_3)
    Sequ_3;
sinon // tous autres cas
    Sequ_4;
fsi
```



Ens_*i* : ensemble de valeurs
de même type que
Variable_de_choix

Ens_*i* et Ens_*j* disjoints

choix_sur *Variable_de_choix* **entre**

cas Ens_1 :

Sequ_1;

cas Ens_2 :

Sequ_2;

cas Ens_3 :

Sequ_3;

autre :

Sequ_4;

fchoix

choix_sur *MonPersonnage* **entre**

cas JeSuisUnGuerrier :

AttaquerAvecEpee;

cas JeSuisUnMage :

LancerUnSort;

cas JeSuisUnSuperPlombier :

LancerUneBouleDeFeu;

autre :

Partir;

fchoix

PLAN

- A. Info diverses
- B. C'est quoi un algo ?
- C. Premiers exemples
- D. Types d'action
- E. Schéma alternatif
- F. Schéma alternatif en cascade
- G. Schéma de choix
- H. Expression logique complexe

H. Expressions logiques complexes

H.1 Opérateur **ET**

Schémas alternatifs imbriqués

```
si (Expr_log_1)
    si (Expr_log_2)
        Sequ_1;
    fsi
fsi
Sequ_n;
```

opérateur logique

```
si (Expr_log_1 ET Expr_log_2)
    Sequ_1;
fsi
Sequ_n;
```

un seul niveau d'imbrication

expression logique complexe

on n'exécute Sequ_1 **que si**
les deux expressions logiques sont vraies

Exemple de schémas alternatifs imbriqués

```
si (JeSuisUnMage)
    si (BeaucoupDeVie)
        Attaquer;
    fsi
fsi
Avancer;
```

```
si (JeSuisUnMage ET BeaucoupDeVie)
    Attaquer;
fsi
Avancer;
```

Schémas alternatifs imbriqués

```
si (Expr_log_1)
    si (Expr_log_2)
        Sequ_1;
    sinon
        Sequ_2;
fsi
fsi
```

impossible à simplifier

On ne peut pas simplifier
s'il y a 2 séquences différentes

Exemple :

```
si (JeSuisUnMage)
    si (BeaucoupDeVie)
        Attaquer;
    sinon
        PrendreUnePotion;
fsi
fsi
Avancer;
```


H.2 Opérateur **OU**

Schémas alternatifs imbriqués

```
si (Expr_log_1)
    Sequ_1;
sinon
    si (Expr_log_2)
        Sequ_1;
    fsi
fsi
```

on exécute Sequ_1
si Expr_log_1 est vraie
ou, à défaut,
si Expr_log_2 est vraie

opérateur logique

```
si (Expr_log_1 OU Expr_log_2)
    Sequ_1;
fsi
```

un seul niveau d'imbrication

expression logique complexe

Exemple de schémas alternatifs imbriqués :

```
si (BeaucoupDeVie)
    Attaquer;
sinon
    si (JeSuisUnGuerrier)
        Attaquer;
    fsi
fsi
```

```
si (BeaucoupDeVie OU JeSuisUnGuerrier)
    Attaquer;

fsi
```

COURS D'ALGORITHMIQUE (2)

A. Casali

PLAN

A. Expressions logiques complexes

B. Variable (s)

C. Entrées / Sorties

D. Type primitif `booléen`

E. Types primitifs `entier` **et** `entier_naturel`

F. Type primitif `reel`

G. Type primitif `caractere`

H. Type complexe `tableau_de`

I. Cas particulier : les `tableaux_de` caractères

J. Un Premier algorithme

A. Expressions logiques complexes

A.1 Opérateur **ET_ALORS**

Attention !

```
si (Expr_log_1)
    si (Expr_log_2)
        Sequ_1;
    fsi
fsi
Sequ_n;
```

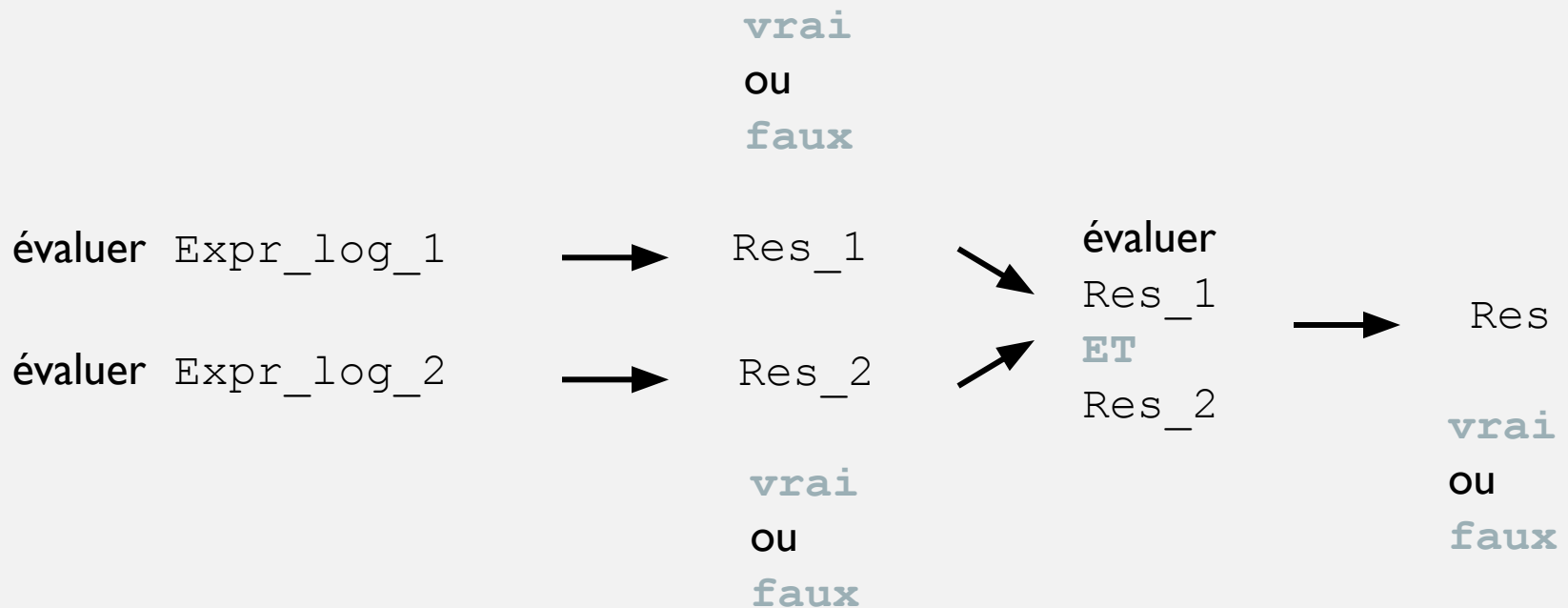
```
si (Expr_log_1 ET Expr_log_2)
    Sequ_1;
fsi
Sequ_n;
```

schémas pas **tout-à-fait** équivalents

Mathématiquement

$\text{Expr_log_1} \text{ ET } \text{Expr_log_2} \quad \equiv \quad \text{Expr_log_2} \text{ ET } \text{Expr_log_1}$

Evaluation mathématique



pas de chronologie (pas de notion de temps en mathématique !)

Evaluation algorithmique de la **condition double**

≡ mathématiques

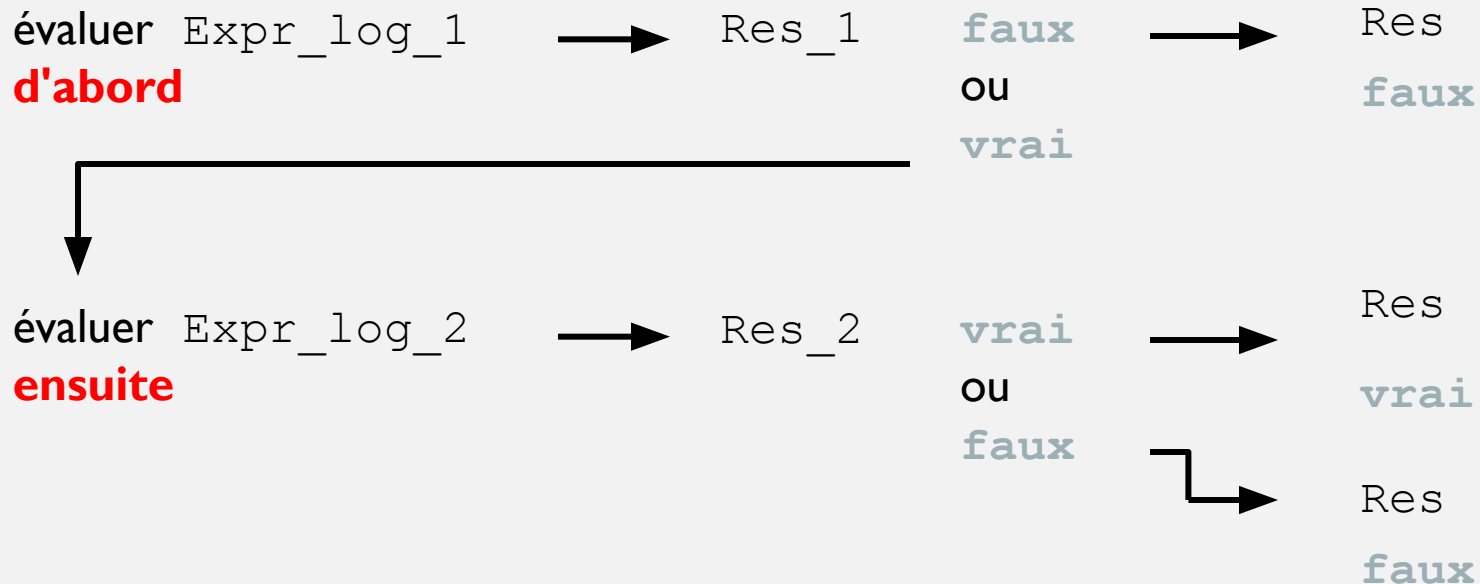
=> les deux expressions sont **toujours** évaluées

Evaluation algorithmique des **deux schémas alternatifs imbriqués**

```
si (Expr_log_1)
    si (Expr_log_2)
        Sequ_1;
    fsi
fsi
Sequ_n;
```

=> les deux expressions sont **toujours** évaluées

Evaluation algorithmique des deux schémas alternatifs imbriqués



=> si `Res_1` est **faux**, `Expr_log_2` n'est **jamais** évaluée

Problème

évaluée seulement si fichier "ADetruire" existe

```
si (fichier "ADetruire" ∃)
```

```
    si (date_de_création_fichier "ADetruire"  
        < 01/01/2012)
```

```
        Effacer fichier "ADetruire";
```

```
    fsi
```

```
fsi
```

toujours évaluée même si fichier "ADetruire" n'existe pas !!!

```
si (fichier "ADetruire" ∃ ET
```

```
    date_de_création_fichier "ADetruire"  
    < 01/01/2012)
```

```
    Effacer fichier "ADetruire" ;
```

```
fsi
```

Solution

"sorte d'opérateur booléen"



```
si (fichier "ADetruire"  $\exists$  ET ALORS  
    date_de_création_fichier "ADetruire"  
                                < 01/01/2012)
```

```
    Effacer fichier "ADetruire" ;
```

```
fsi
```

A.2 Opérateur **OU_SINON**

De même

```
si (Expr_log_1)
    Sequ_1;
sinon
    si (Expr_log_2)
        Sequ_1;
    fsi
fsi
```

≠

```
si (Expr_log_1 OU Expr_log_2)
    Sequ_1;
fsi
```

≡

```
si (Expr_log_1 OU_SINON
    Expr_log_2)
    Sequ_1;
fsi
```

```
si (BeaucoupDeVie)
    Attaquer;
sinon
    si (JeSuisUnGuerrier)
        Attaquer;
    fsi
fsi
```

```
si (BeaucoupDeVie OU_SINON JeSuisUnGuerrier)
    Attaquer;

fsi
```

A.3 Propagation de l'opérateur NON

```
si (NON (Expr_log_1 ET Expr_log_2))  
    ≡  
si ((NON Expr_log_1) OU (NON Expr_log_2))
```

Exemple :

```
si (NON (JeSuisUnGuerrier ET BeaucoupDeVie))  
    Avancer;  
fsi  
Attaquer;
```

Se traduit par :

```
si ((NON JeSuisUnGuerrier) OU (NON BeaucoupDeVie))  
    Avancer;  
fsi  
Attaquer;
```

De la même manière

```
si (NON (Expr_log_1 ET_ALORS Expr_log_2))  
    ≡  
si ((NON Expr_log_1) OU_SINON (NON Expr_log_2))
```

Exemple :

```
si (NON (JeSuisUnGuerrier ET_ALORS BeaucoupDeVie))  
    Avancer;  
fsi  
Attaquer;
```

Se traduit par :

```
si ((NON JeSuisUnGuerrier) OU_SINON (NON BeaucoupDeVie))  
    Avancer;  
fsi  
Attaquer;
```

```
si (NON (Expr_log_1 OU Expr_log_2))
```

≡

```
si ((NON Expr_log_1) ET (NON Expr_log_2))
```

Exemple :

```
si (NON (PeuDeVie OU JeSuisUnMage))
```

```
    Attaquer;
```

```
fsi
```

```
Avancer;
```

Se traduit par :

```
si ((NON PeuDeVie) ET (NON JeSuisUnMage))
```

```
    Attaquer;
```

```
fsi
```

```
Avancer;
```

De la même manière

```
si (NON (Expr_log_1 OU_SINON Expr_log_2))  
    ≡  
si ((NON Expr_log_1) ET_ALORS (NON Expr_log_2))
```

Exemple :

```
si (NON (PeuDeVie OU_SINON JeSuisUnMage))  
    Attaquer;  
fsi  
Avancer;
```

Se traduit par :

```
si ((NON PeuDeVie) ET_ALORS (NON JeSuisUnMage))  
    Attaquer;  
fsi  
Avancer;
```


A.4 En résumé, tables de vérités des expressions logiques complexes

Opérateurs **ET** / **ET_ALORS**

On cherche à évaluer la valeur l'expression logique :

$\text{Expr_log_3} = \text{Expr_log_1} \text{ **ET** } \text{Expr_log_2}$

ET / ET_ALORS	Expr_log_1 (V)	Expr_log_1 (F)
Expr_log_2 (V)	V	F
Expr_log_2 (F)	F	F

Opérateurs OU / OU_SINON

On cherche à évaluer la valeur l'expression logique :

$\text{Expr_log_3} = \text{Expr_log_1} \text{ OU } \text{Expr_log_2}$

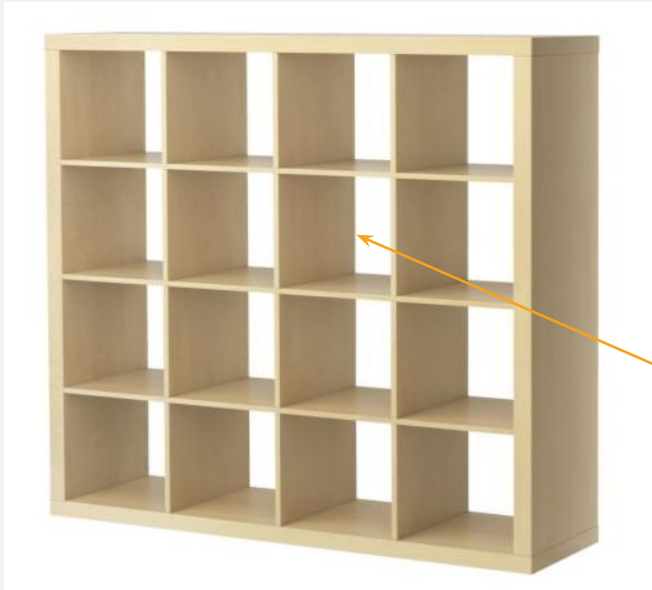
OU / OU_SINON	Expr_log_1 (V)	Expr_log_1 (F)
Expr_log_2 (V)	V	V
Expr_log_2 (F)	V	F

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

B. VARIABLE(S)

B.1 Organisation de la mémoire



1 case de la mémoire ne peut contenir qu'une **unique** variable (du même type)

Dans cette case mémoire (qui s'appelle Jeu pour notre algorithme), on peut ranger un unique jeu de société.

B.2 Déclaration d'une variable

Modèle général :

declarer nomDeVariable : **Type**;

Caractère de terminaison
d'une instruction

Exemple :

declarer jeu : jeuDeSociete;

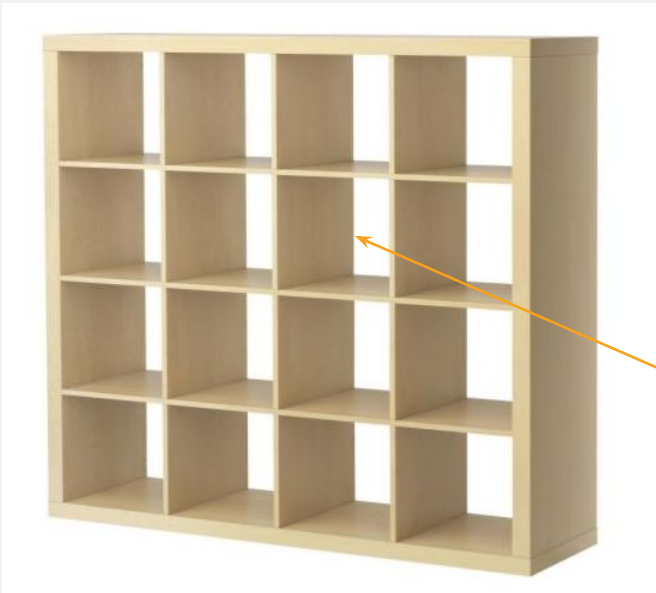
B.3 Affectation du contenu d'une variable

Modèle général :

`nomDeVariable <- Valeur;`

Exemple :

`jeu <- ColonDeCatane;`



La case nommée `jeu` contient le jeu de société `ColonDeCatane`.

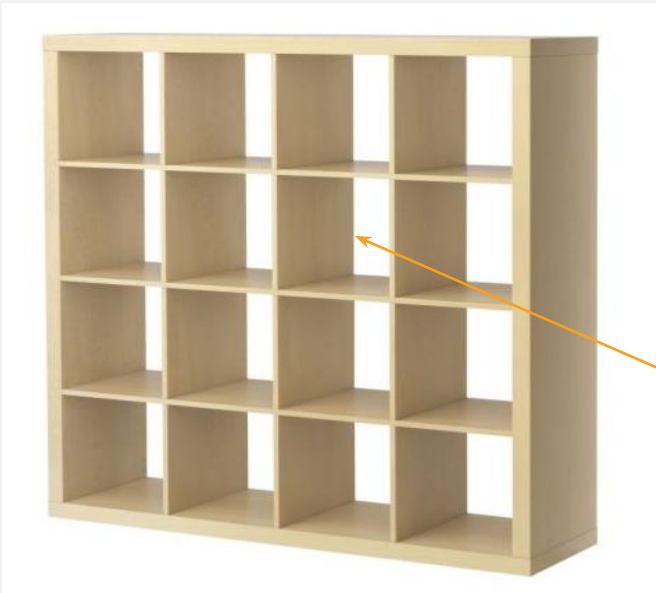
B.4 Réaffectation du contenu d'une variable

Modèle général :

`nomDeVariable <- nouvelleValeur;`

Exemple :

`jeu <- Carcassonne;`



La case nommée `jeu` contient maintenant le jeu de société `Carcassonne`.

B.5 Cas des variables constantes

Une constante est une variable qui : possède une unique valeur tout au long de son existence. En conséquence :

1. On doit l'initialiser lors de sa création;
2. On ne peut modifier son contenu.

Modèle général :

declarer KnomVariable : **constante** Type <- Valeur;;

Exemple :

declarer Kjeu : **constante** jeuDeSociete <- ColonDeCatane; **Juste**

~~Kjeu <- Carcassonne;~~

Faux

B.6 Sémantique des nom de variables

Chaque nom de variable contient la sémantique liée à son type

Exemple :

```
declarer superPlombier : delivreurDePrincesse;
```

```
superPlombier <- Mario;           Juste
```

```
superPlombier <- Luigi;           Juste
```

~~superPlombier <- Link;~~ Juste

Mais sémantiquement faux

B.7 nomenclature des nom de variables

1. 1^{er} mot : en minuscule
2. 1ere lettre du mot suivant en majuscule
3. Reste du mot en minuscule
4. Revenir en 2 si présence d'un autre mot

Exemple :

```
declarer superPlombier : delivreurDePrincesse;
```

Exception : les constantes commence par la lettre 'K'

Exemple :

```
declarer Kjeu : constante jeuDeSociete <- ColonDeCatane;
```

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

C. ENTRÉES / SORTIES

C.1 But

Favoriser les interactions avec l'utilisateur soit :

- En demandant une saisie clavier;
- En provoquant un affichage écran.

C.2 Saisie clavier

Prérequis : on ne peut demander la saisie au clavier d'une variable à l'unique condition qu'elle ait été déclarée auparavant!

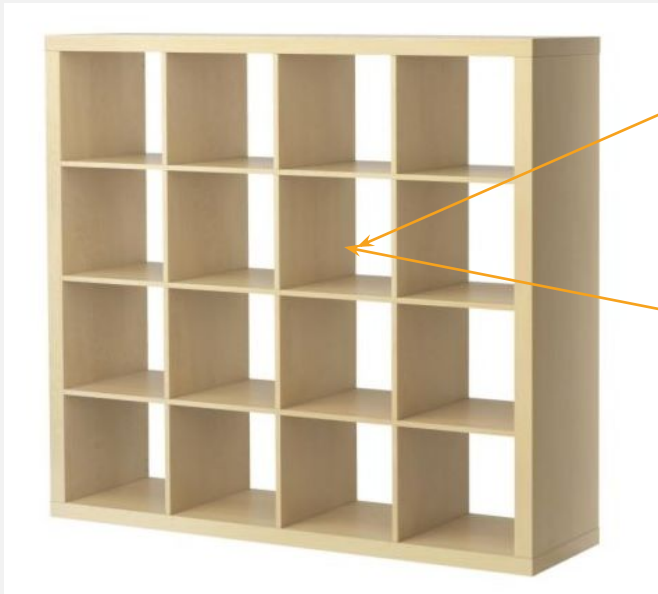
Modèle général :

`saisir` (nomDeVariable);

Exemple :

```
declarer superPlombier : delivreurDePrincesse;
```

```
saisir (superPlombier); //Mario
```



Allocation d'une case nommée
superPlombier ne contenant rien.

Le contenu de la case
superPlombier **contient** Mario



Règle d'or de l'informatique #2 : NTUI (Never Trust
User Input)

C.3 Affichage écran

Prérequis : on ne peut afficher à l'écran le contenu d'une variable uniquement si elle a été initialisée, saisie ou affectée.

Modèle général :

```
afficher (nomDeVariable) ;
```

Exemple :

```
afficher (superPlombier) ;           Mario
```

C.4 Passage à la ligne

L'instruction **ligne_suivante** provoque le passage à la ligne lors d'un affichage écran.

Exemple :

	Affichage
afficher (superPlombier) ;	Mario
superPlombier <- Luigi;	
ligne_suivante ;	↵
afficher (superPlombier) ;	Luigi

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` et `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

D. TYPE PRIMITIF BOOLEEN

D.1 identificateur

`booléen`

D.2 valeurs

`vrai` `faux`

D.3 opérations (opérateurs booléens) produisent un `booléen`

`ET` `OU` `NON` `ET_ALORS` `OU_SINON`

```
declarer toBe        : booléen;
```

```
declarer notToBe    : booléen;
```

```
declarer question : booléen;
```

`...`

```
toBe        <- faux;
```

```
notToBe    <- NON toBe;            // vrai
```

```
question <- toBe OU notToBe; // toujours vrai!
```

D.1 identificateur

`booléen`

D.2 valeurs

`vrai` `faux`

D.3 opérations (opérateurs booléens) produisent un `booléen`

`ET` `OU` `NON`

D.4 opérations (d'identité) produisent un `booléen`

`vaut` `ne_vaut_pas`

```
si (toBe vaut notToBe)
    afficher ("c'est n'importe quoi !");
fsi
```


D.5 Simplification des opérations d'identité



Dans un test de comparaison, on ne compare jamais un booléen avec les valeurs `vrai` ou `faux`.


```
declarer jeSuisUnMage : boolean;  
jeSuisUnMage <- vrai;
```

```
si (jeSuisUnMage vaut vrai)
```

...

```
fsi
```

Sémantiquement juste mais non conforme aux normes de programmation



```
si (jeSuisUnMage)
```

...

```
fsi
```

← Comparaison automatique du booléen à la valeur `vrai`

```
si (NON jeSuisUnMage)
```

...

```
fsi
```

← Comparaison automatique du booléen à la valeur `faux`

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. ~~Types primitifs `entier` et `entier_naturel`~~
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

E. TYPES PRIMITIFS ENTIER ET ENTIER_NATUREL

E.1 identificateur

`entier` `entier_naturel`

E. 2 valeurs

sous-ensemble des entiers mathématiques

E.3 opérations (opérateurs arithmétiques) produisent un *entier*

`+` `-` `*` `/`

Exemple :

`declarer x : entier;`

`afficher (x);`

`x <- 1;`

`afficher (x);`

`x <- (x + 2) / 4 * 3;`

`afficher (x);`

`x <- 1;`

`x <- (x + 2) * 3 / 4;`

`afficher (x);`



division euclidienne (troncature)

affiche n'importe quoi

1

0

2

E.4 opérations (de comparaison) produisent un **booléen**

< <= > >= ∃ relation
d'ordre

E.5 opérations (d'identité) produisent un **booléen**

vaut ne_vaut_pas

Exemple :

```
declarer nb1 : entier_naturel;  
nb1 <- 12;  
declarer nb2 : entier_naturel;  
nb2 <- 5;  
declarer nb3 : entier_naturel;  
nb3 <- nb1 / nb2;      // nb3 vaut 2
```

```
si (nb3 vaut 4)  
    afficher ("nb3 vaut 4");  
sinon  
    si (nb3 >= 2)  
        afficher ("nb3 >= 2");  
    fsi  
fsi
```

E.6 Fonctions mathématiques applicables

produisent un *entier*

modulo (Ent_1, Ent_2)

Retourne le reste de la division entière

Exemple :

```
declarer x : entier_naturel;
```

```
x <- 10;
```

```
declarer y : entier_naturel;
```

```
y <- modulo (2, 10); // y <- modulo (2, x)
```

```
afficher (y) ; //2
```

De manière plus compacte :

```
afficher (modulo (2, x));
```

rand (entier1, entier2)

Retourne un entier aléatoire dans l'intervalle [entier1, entier2]

Exemple :

```
si (rand (1,2) vaut 1)
```


```
    afficher ("j ai gagné");
```

```
sinon
```

```
    afficher ("tu as perdu");
```

```
fsi
```

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`

- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

F. Type primitif `reel`

F.1 identificateur

`reel`

littéral :

`3.5`

`-21.2e12`

F.2 valeurs

sous-ensemble des réels mathématiques

F.3 opérations (opérateurs arithmétiques) produisent un `reel`

`+` `-` `*` `/`



division réelle

F.4 opérations (de comparaison) produisent un `booléen`

`<` `<=` `>` `>=`

\exists relation
d'ordre

F.5 opérations (d'identité) produisent un `booléen`

`vaut` `ne_vaut_pas`

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier algorithme

G. Type primitif caractere

G.1 identificateur

caractere

littéral : symbole entre simples quotes

'a' '9'

G.2 valeurs

G.3 opérations (de comparaison)

produisent un **booléen**

< <= > >= ∃ relation

d'ordre

chiffres < ... < majuscules < ...

< minuscules < ... < caractères

accentués

G.4 opérations (d'identité)

produisent un **booléen**

vaut ne_vaut_pas

F.6 fonctions applicables

produisent un **caractere**

tolower (unCaract)

renvoie le caractère en minuscule (uniquement si c'est une majuscule)

toupper (unCaract)

renvoie le caractère en majuscule (uniquement si c'est une minuscule)

succ (unCaract)

renvoie le caractère suivant dans l'ordre prédéfini

pred (unCaract)

renvoie le caractère précédant dans l'ordre prédéfini

Exemple:

```
declarer caract : caractere;
```

```
caract <- nImporteQuelCaractere;
```

```
...
```

```
afficher (tolower (caract)); // caract en minuscule
```

```
afficher (toupper (caract)); // caract en majuscule
```

```
afficher (succ (caract)); // suivant de caract
```

```
afficher (pred (caract)); // précédent de caract
```

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. ~~Type complexe `tableau_de`~~
 - I. Cas particulier : les `tableaux_de` caractères
 - J. Un Premier algorithme

H. Type primitif `tableau_de`

H.1 identificateur

`tableau_de`

permet de regrouper plusieurs valeurs sous un même identificateur

désignées par leur rang

H.2 modèle général de déclaration

declarer nomTab : `tableau_de` [Taille] `typePrimitif`;

Exemple :

`declarer` tabReel : `tableau_de` 10 `reel`;

TabReel désigne l'ensemble des 10 valeurs réelles



espace mémoire réservé mais **pas initialisé**

H.3 Accès à un élément du tableau

On accède au $(i+1)^{\text{ème}}$ élément du tableau `tab` avec la notation entre crochet `tab[i]`



1. La première case du tableau a pour indice 0;
2. La dernière case du tableau a pour indice sa taille – 1;
3. Aucun contrôle quant à la validité de l'indice n'est effectué lors d'un accès à une case, que ce soit en lecture ou en écriture.

Exemple :

```
tabReel[0] <- 10; //accès en écriture;
```

```
tabReel[1] <- 11;
```

...

```
afficher (tabReel[0]); //10 - accès en lecture
```

```
afficher (tabReel[10]); //oups boulette
```

Un problème a été détecté et Windows a été arrêté afin de prévenir tout dommage sur votre ordinateur.

Si vous voyez cet écran d'erreur d'arrêt pour la première fois, redémarrez votre ordinateur. Si cet écran apparaît encore, suivez ces étapes :

Recherchez tout virus sur votre ordinateur. Supprimez tout disque dur ou contrôleur de disque dur nouvellement installé. Vérifiez votre disque dur afin de vous assurer qu'il est correctement configuré et terminé. Exécutez CHKDSK /F pour vérifier la présence d'un dommage sur votre disque dur puis redémarrez votre ordinateur.

Informations techniques :

*** STOP: 0x0000007B (0xF78C8528,0xC0000034,0x00000000,0x00000000)

H.4 Connaître la taille d'un tableau

`taille` (unTableau)

Renvoie le nombre d'éléments (nombre de cases) du tableau

unTableau

Exemple:

```
afficher (taille (tabReel)); //10
```

Conséquence :

**Tous les indices valides d'un tableau Tab sont dans les bornes
[0, taille (Tab) [**

H.5 Initialisation d'un tableau

On est obligé d'initialiser le tableau case par case.

Il est impossible d'initialiser toutes les cases d'un tableau avec une unique instruction.

=> Pas de tableaux constants

H.6 les tableaux non dimensionnés

La plupart du temps, on ne connaît pas la dimension (taille) d'un tableau à l'avance. On veut pouvoir:

- Ajouter de nouveaux éléments;
 - Supprimer des éléments obsolètes;
- tout en gardant une taille convenable.

```
declarer nomTab : tableau_de typePrimitif;
```

Exemple :

```
declarer tabInt : tableau_de entier;
```

Allongement du tableau :

```
allonger (unTab, nbElem);
```

Ajoute au tableau unTab, nbElem **nouvelles** cases à la fin du tableau.

=> **les nouvelles cases ne sont pas initialisées!**

Exemple :

```
allonger (tabInt, 5);
```

```
afficher (taille (tabInt));
```

```
afficher (tabInt[0]);
```

5

n'importe quoi

Fixer une nouvelle taille à un tableau :

redimensionner (unTab, nbElem) ;

Fixe la taille du tableau unTab à nbElem.

On peut soit :

- Allonger le tableau, auquel cas les nouvelles cases ne sont pas initialisées;
- Tronquer (raccourcir) le tableau.

Exemple :

redimensionner (tabInt, 10);

afficher (**taille** (tabInt));

10

tabInt[3] <- 13;

afficher (tabInt[3]);

13

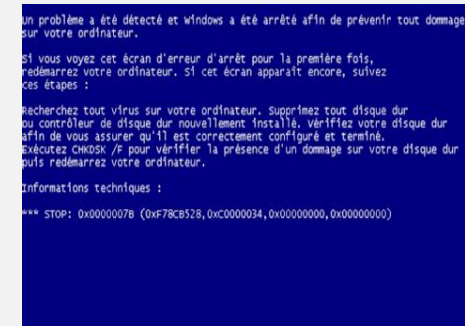
redimensionner (tabInt, 2);

afficher (**taille** (tabInt));

2

afficher (tabInt[3]);

la case n'existe plus



PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. ~~Cas particulier : les `tableaux_de` caractères~~
- J. Un Premier algorithme

I. Type complexe `string`

I.1 identificateur `string` littéral : suite de caractères entre guillemets (doubles quotes)

```
declarer Chaîne : string;
```

```
Chaîne <- "AEIOUY\"";
```

`AEIOUY`

chaîne vide

7 caractères

I.2 opérations et fonctions applicables

```
taille (Chaîne)
```

```
< <= > >=
```

∃ relation d'ordre (lexicographique)

```
vaut ne_vaut_pas
```

17 caractères

I.3 concaténation

```
+
```

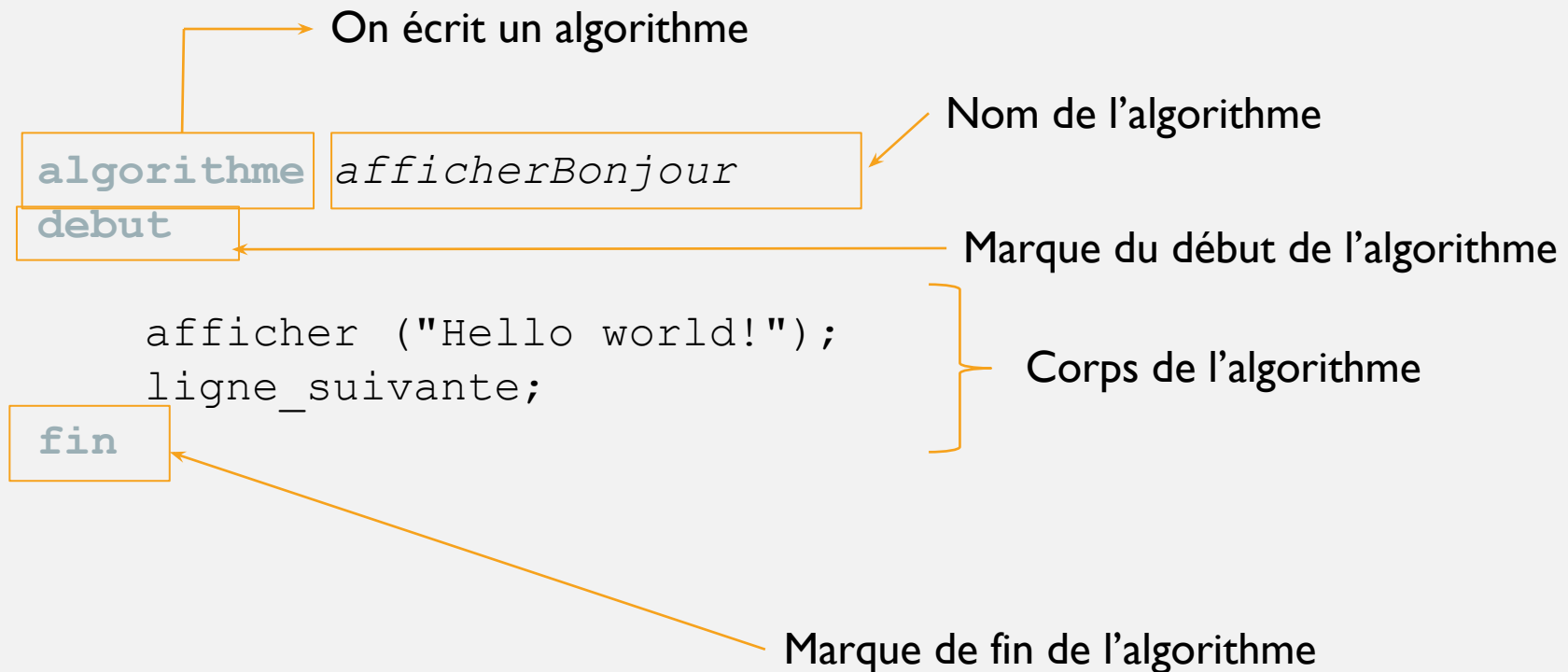
```
Chaîne <- Chaîne + "0123456789";
```

PLAN

- A. Expressions logiques complexes
- B. Variable (s)
- C. Entrées / Sorties
- D. Type primitif `booléen`
- E. Types primitifs `entier` **et** `entier_naturel`
- F. Type primitif `reel`
- G. Type primitif `caractere`
- H. Type complexe `tableau_de`
- I. Cas particulier : les `tableaux_de` caractères
- J. Un Premier `algorithme`

J. Premier algorithme

Exemple : Afficher "Hello world!"



COURS D'ALGORITHMIQUE (3)

A. Casali

PLAN

A. Type Complexe `string`

B. Premier Algorithme

C. Second algorithme

D. Boucle `boucle`

E. Boucle `tant_que` **et** `jusqua`

F. Quelques algorithmes utilisant les boucles

G. Faire ses TDs à la maison

A. Type complexe `string`

A.1 identificateur `string` littéral : suite de caractères entre guillemets (doubles quotes)

```
declarer Chaîne : string;
```

```
Chaîne <- "AEIOUY\"";
```

AEIOUY"

chaîne vide

7 caractères

A.2 opérations et fonctions applicables

```
taille (Chaîne)
```

```
< <= > >=
```

∃ relation d'ordre (lexicographique)

```
vaut ne_vaut_pas
```

17 caractères

A.3 concaténation

```
+
```

```
Chaîne <- Chaîne + "0123456789";
```

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` **et** `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

B. Premier algorithme

Exemple : Afficher "Hello world!"

On écrit un algorithme

`algorithme`

`debut`

afficherBonjour

Nom de l'algorithme

Marque du début de l'algorithme

```
afficher ("Hello world!");  
ligne_suivante;
```

Corps de l'algorithme

`fin`

Marque de fin de l'algorithme

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` **et** `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

C. Second algorithme

Exemple : Calculer $y = ax + b$

```
algorithme calculEquationDroite  
debut
```

```
    afficher ("Entrez les valeurs de a, x, puis b ");  
    declarer a, b, x : reel;  
    saisir (a);  
    saisir (x);  
    saisir (b);  
    afficher ("La valeur de la fonction affine \"y = \", a,  
    \"x +\", b, \"\" en x = \", x, \" est de \", a*x + b);  
fin
```

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` et `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

D. BOUCLE BOUCLE

Exemple : tuer un monstre

boucle

attaquer;

fboucle

L'action « *attaquer* » est répétée indéfiniment. On doit sortir de la boucle à un moment!

boucle

attaquer;

si (victoire) sortie;

fboucle

D.I Forme générale (I)

boucle

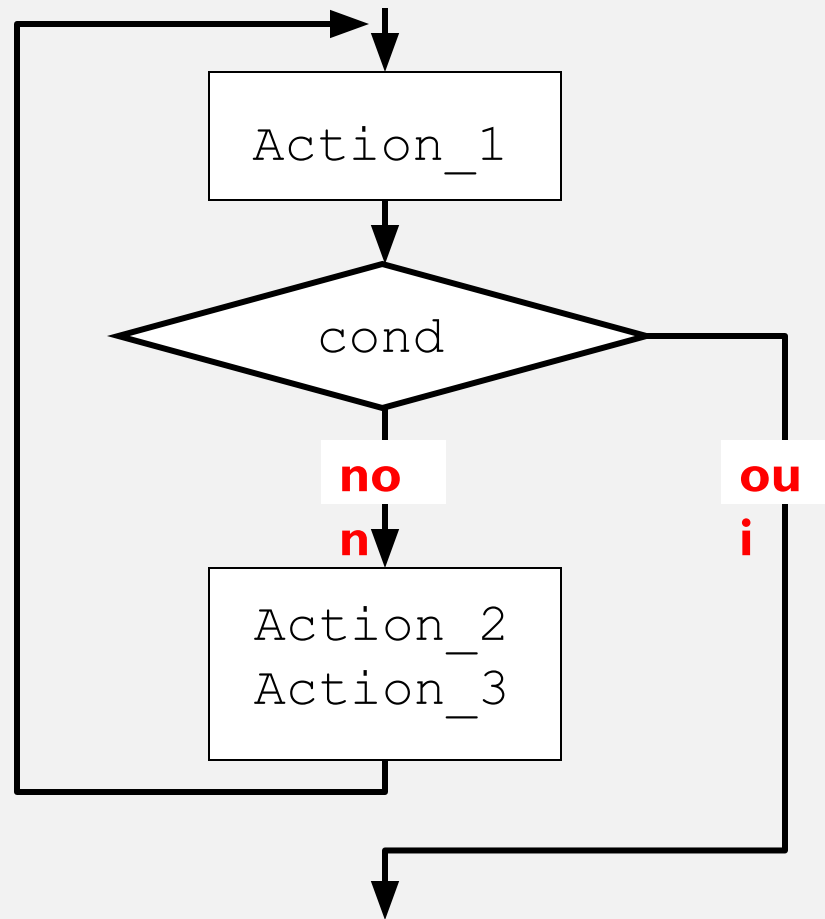
Action_1;

si (cond) sortie;

Action_2;

Action_3;

fboucle



D.2 Forme générale (2)

boucle

```
si (cond_1) sortie;
```

```
Inst_1;
```

```
si (cond_2) sortie;
```

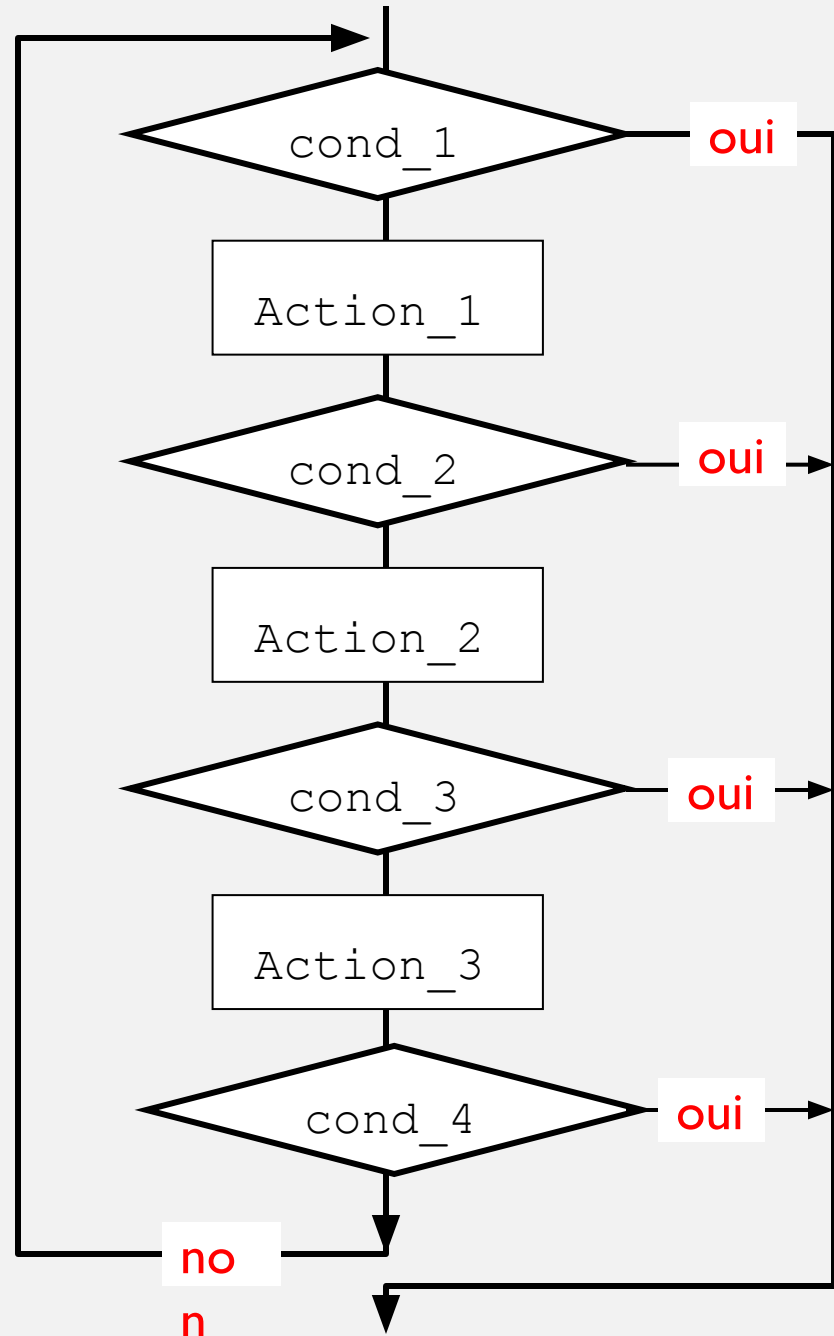
```
Inst_2;
```

```
si (cond_3) sortie;
```

```
Inst_3;
```

```
si (cond_4) sortie;
```

fboucle



D.3 Attention à l'ordre des instructions

Exemple : avancer jusqu'au bord de la falaise

boucle

```
    avancerDUnPas;
```

```
    si (bordDeFalaise) sortie;
```

fboucle



Equivalent?

Oui ssi vous avez confiance en vous 😊

boucle

```
    si (bordDeFalaise) sortie;
```

```
    avancerDUnPas;
```

fboucle

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` et `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

E. Boucle tant_que / jusqu'a

E.1 Couper du tube de cuivre

(voir perso.orange.fr/stippylesite/plomberie.htm)

// On utilise un coupe-tube à molette

1. Placer le tube dans le coupe-tube, à l'endroit de la coupe;
2. Serrer la poignée du coupe-tube pour que la molette soit en appui sur le tube;
3. Serrer la poignée d'un demi-tour;
4. Faire tourner le coupe-tube autour du tube;
5. **Répéter** les opérations 3 et 4 **jusqu'à** coupure complète;
6. Ebavurer l'intérieur du tube à la lime ou au couteau;

Formalisation du problème

commentaires

// On utilise ...

Placer le tube ... ;

Serrer la poignée ... pour que la molette ... ;

repeter

Serrer la poignée d'un demi-tour ;

Faire tourner ... ;

jusqua (coupure complète)

schéma répétitif

"repeter ... jusqu"

condition d'arrêt

= expression logique

Ebavurer l'intérieur ... ;

E.2 Scier du bois

1. Prendre la bûche;

2. **si** (sa longueur > 30 cm)

Lui enlever 30 cm;
Stocker le morceau coupé;

sinon

aller en 4.

fsi

3. Remonter à 2.

4. Stocker le reste de la bûche

ou **mieux** :

1. Prendre la bûche;

2. ~~tant que~~ (sa longueur > 30 cm)

ou, différent :

jusqua (sa longueur <= 30 cm)

faire

Lui enlever 30 cm;

Stocker le morceau coupé;

ffaire

mais pas :

3. Stocker le reste de la bûche

1. Prendre la bûche;

2. **repeter**

Lui enlever 30 cm;

Stocker le morceau coupé;

jusqua (sa longueur ≤ 30 cm)

faux : impossible si
la longueur initiale < 30 cm

3. Stocker le reste de la bûche

E.3 Modèle général des boucles `tant_que` **et** `jusqua`

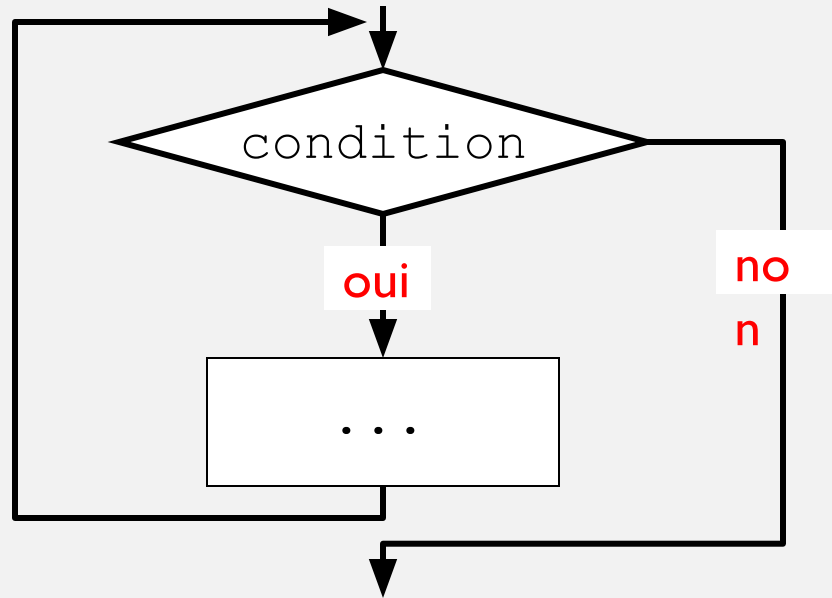
<code>tant_que</code> (condition_A)	<code>jusqua</code> (condition_B)
<code>faire</code>	<code>faire</code>
...	...
<code>ffaire</code>	<code>ffaire</code>

<code>repeter</code>	<code>repeter</code>
...	...
<code>tant_que</code> (condition_C)	<code>jusqua</code> (condition_D)

`tant_que (condition)`
`faire`

...

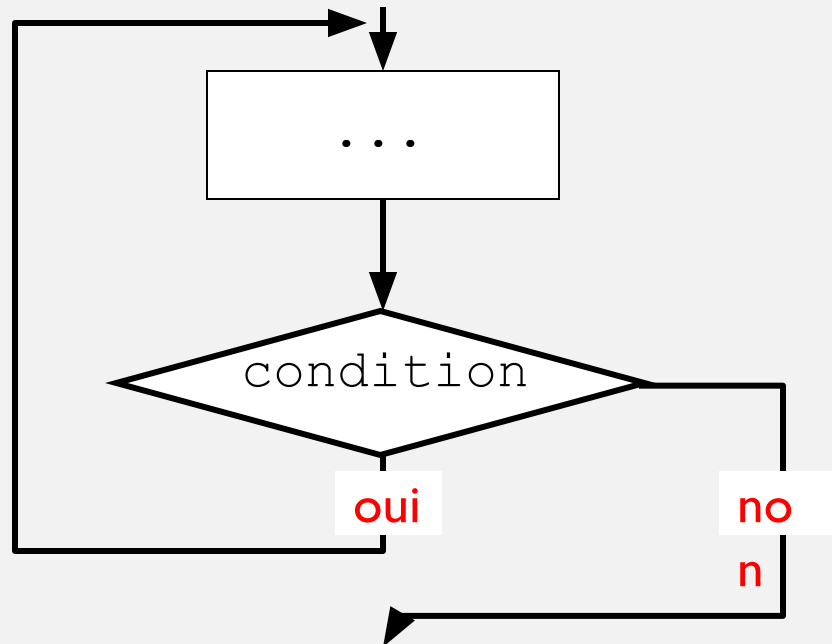
`ffaire`



`repeter`

...

`tant_que (condition)`



E.4 boucle jusquà vue comme un cas particulier de la boucle boucle (1)

boucle

```
si (cond_1) sortie;
```

```
Action_1;
```

```
Action_2;
```

```
Action_3;
```

fboucle



jusqua (cond_1)

faire

```
Action_1;
```

```
Action_2;
```

```
Action_3;
```

ffaire

E.5 boucle jusquà vue comme un cas particulier de la boucle boucle (2)

boucle

Action_1;

Action_2;

Action_3;

si (cond_1) sortie;

fboucle

repeter

Action_1;

Action_2;

Action_3;

jusqua (cond_1)



E.6 Equivalence entre boucle

```
tant_que (condition_A)  
faire
```

...

```
ffaire
```

```
jusqua (Non condition_A)  
faire
```

...

```
ffaire
```



```
jusqua (condition_B)  
faire
```

...

```
ffaire
```

```
tant_que (Non condition_B)  
faire
```

...

```
ffaire
```



repeter

...

`tant_que` (condition_C)

repeter

...

`jusqua` (Non condition_C)



repeter

...

`jusqua` (condition_D)

repeter

...

`tant_que` (Non condition_C)



E.7 instruction `continue`

Dans une boucle, l'instruction `continue` a pour conséquence de remonter en début de boucle.

Sans instruction `continue`

```
tant_que (NON victoire)
faire
```

```
    prendreUnePotion;
    si (beaucoupDeVie)
        attaquer;
    fsi
ffaire
```

2 blocs => 2 indentations =>
2 fois moins lisible

Avec instruction `continue`

```
tant_que (NON victoire)
faire
```

```
    prendreUnePotion;
    si (NON beaucoupDeVie) continue;
    attaquer;
ffaire
```

1 bloc => 1 indentation =>
plus lisible

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` **et** `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

F. Quelques algo. utilisant les boucles

F.1 Afficher 100 fois "Bonjour" (VI)

boucle

parcourir 100 fois la boucle =>

on a besoin d'un compteur =>

Prendre un objet qu'on désignera par `compteur` ;

Initialiser le `Compteur` à `???` ;

dans quel ordre ?

Ajouter 1 au `compteur` ;

Afficher "Bonjour";

`si` (condition sur `compteur`) `sortie` ;

"Prendre un objet qu'on désignera par `compteur`;"

`declarer compteur : entier_naturel;`

Initialiser le `compteur` à `???`;

signification de `compteur` :

nombre de fois que la boucle a été parcourue =>

Mettre `compteur` à 0 ;

`compteur <- 0;`


```
declarer compteur : entier_naturel;
```

```
compteur <- 0;
```

boucle

Afficher "Bonjour";

Ajouter 1 au compteur ;

si (compteur vaut 100) sortie ;

fboucle

autres constantes

de type entier

compteur est une **variable**

"Bonjour" est une **constante** (un littéral)

de type chaine_de_caracteres

string

```
declarer compteur : entier_naturel;
```

```
compteur <- 0;
```

```
boucle
```

```
    afficher ("Bonjour") ;
```

```
    Ajouter 1 au compteur ;
```

```
    si (compteur vaut 100) sortie ;
```

```
fboucle
```

"nouvelle" valeur de compteur c'est
("ancienne" valeur de compteur + 1);

```
compteur <- (compteur + 1) ;
```

facultatives

version finale

algorithme *Afficher_100_Bonjour*

debut

declarer *compteur* : entier_naturel;

compteur <- 0;

boucle

afficher ("Bonjour") ;

compteur <- *compteur* + 1;

si (*compteur* vaut 100) **sortie** ;

fboucle

fin

F.2 Afficher 100 fois "Bonjour" (V2)

```
tant_que (compteur ne_vaut_pas 100)
faire
    afficher ("Bonjour") ;
    compteur <- compteur + 1;
ffaire
```

```
jusqua (compteur vaut 100)
faire
    afficher ("Bonjour") ;
    compteur <- compteur + 1;
ffaire
```

```
repete
    afficher ("Bonjour") ;
    compteur <- compteur + 1;
jusqua (compteur vaut 100)
```

F.2 Afficher le contenu d'un tableau de réels

```
declarer tabReel : tableau_de reel;  
//génération de tabReel
```

Pour pouvoir afficher le contenu de `tabReel`, il faut :

1. Parcourir `tabReel` case par case;
2. Afficher le contenu de la case courante.

On a besoin d'un indice (un compteur) qui prend toutes les valeurs possibles.

Autrement dit, l'indice doit prendre **toutes les valeurs** entre 0 et `taille (tabReel) - 1` (bornes incluses)

```
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (tabReel))  
faire  
    //afficher la case courante  
    i <- i + 1; //passage à la case suivante  
ffaire
```

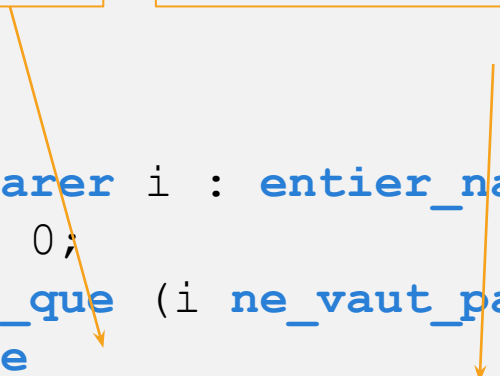
F.2 Afficher le contenu d'un tableau de réels

```
declarer tabReel : tableau_de reel;  
//génération de tabReel
```

Pour pouvoir afficher le contenu de `tabReel`, il faut :

1. Parcourir `tabReel` case par case;
2. Afficher le contenu de la case courante.

```
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (tabReel))  
faire  
    afficher ( TabReel[i] );  
    i <- i + 1; //passage à la case suivante  
ffaire
```



F.3 Afficher le contenu d'un tableau d'entiers

Quels sont les changements par rapport à l'algorithme précédent?

Uniquement le type des éléments à afficher, le parcours ne change pas!

```
declarer tabInt : tableau_de entier;  
//génération de tabInt  
  
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (tabInt))  
faire  
    afficher (tabInt[i]);  
    i <- i + 1; //passage à la case suivante  
ffaire
```

F.4 Afficher le contenu d'un tableau de chaine de caractères

Quels sont les changements par rapport à l'algorithme précédent?

Uniquement le type des éléments à afficher, le parcours ne change pas!

```
declarer tabStr : tableau_de_string;  
//génération de tabStr  
  
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (tabStr))  
faire  
    afficher (tabStr[i]);  
    i <- i + 1; //passage à la case suivante  
ffaire
```


F.5 Afficher le contenu d'une chaîne de caractères

Quels sont les changements par rapport à l'algorithme précédent?

Uniquement le type des éléments à afficher, le parcours ne change pas!

```
declarer str : string;  
//génération de str
```

```
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (str))  
faire  
    afficher (str[i]);  
    i <- i + 1; //passage à la case suivante  
ffaire
```

Ou plus simplement

```
afficher (str);
```

F.6 Afficher le contenu d'une chaîne de caractères un caractère sur deux

Quels sont les changements par rapport à l'algorithme précédent?

Un caractère sur deux

```
declarer str : string;  
//génération de str
```

```
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (str))  
faire  
    si (modulo (i, 2) vaut 0)  
        afficher (str[i]);  
    fsi  
    i <- i + 1; //passage à la case suivante  
ffaire
```

F.6 Afficher le contenu d'une chaîne de caractères un caractère sur deux

Quels sont les changements par rapport à l'algorithme précédent?

Un caractère sur deux

```
declarer str : string;  
//génération de str
```

```
declarer i : entier_naturel;  
i <- 0;  
tant_que (i ne_vaut_pas taille (str))  
faire  
    si (modulo (i, 2) vaut 1) continue;  
    afficher (str[i]);  
    i <- i + 1; //passage à la case suivante  
ffaire
```

Faux : on n'affiche que la première case du tableau

PLAN

- A. Type Complexe `string`
- B. Premier Algorithme
- C. Second algorithme
- D. Boucle `boucle`
- E. Boucle `tant_que` **et** `jusqua`
- F. Quelques algorithmes utilisant les boucles
- G. Faire ses TDs à la maison

Faire ses TDs d'algo (à la maison ou dans les salles machines du département)



Fichier décrivant
un algo



Exécutable (binaire)
GNU / Linux

Normalement impossible, sauf si :

- Utilisation de règles strictes décrivant l'algorithmie;
- Utilisation d'un (ou plusieurs) compilateurs.

Ce qui est fait :

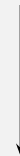
Fichier décrivant
un algo

Traducteur
maison



Fichier c++

Compilateur g++



Exécutable (binaire)
GNU / Linux



Exécution depuis un
terminal uniquement

Plus d'informations :

http://ens.casali.me/casali/Cours_algo/index_algo.html

COURS D'ALGORITHMIQUE (4)

A. Casali

PLAN

- A. Boucle à compteur
- B. Portée des variables
- C. Sous-programme
- D. Tracer la courbe $y = ax^2 + bx + c$

A. Boucle à compteur

A.1 afficher 100 fois "Bonjour"

pour les valeurs successives d'un compteur variant de 1 à 100

```
faire
```

```
    afficher ("Bonjour");
```

```
ffaire
```

```
pour (i variant_de 1 a 100)
```

```
faire
```

```
    afficher ("Bonjour");
```

```
ffaire
```

avantages

pas de déclaration

pas d'initialisation

pas d'incrémentation

pas de test d'arrêt

visibles !

A.2 Afficher le contenu d'un tableau de entier

```
declarer tabInt : tableau_de entier;  
//génération de tabInt
```

```
pour (i variant_de 0 a taille (tabInt) - 1)  
faire  
    afficher (tabInt[i]);  
ffaire
```



On ne doit jamais atteindre la case ayant
pour indice `taille (tabInt)`

A.3 Afficher le contenu d'un tableau de entier à l'envers

Solution 1 :

```
pour (i variant_de 0 a taille (tabInt) - 1)
faire
    afficher (tabInt[taille(tabInt) - 1 - i]);
ffaire
```

Solution 2 :

```
pour (i variant_de taille (tabInt) - 1 a 0 descendant)
faire
    afficher (tabInt[i]);
ffaire
```

A.4 Rupture de séquence et instruction **pour**



Règles : Dans une boucle **pour**, la variable de parcours doit prendre toutes les valeurs entre les bornes minimale et maximale (incluses).



1. Aucune rupture de séquence utilisant une instruction sortie n'est permise.
2. Il faut changer de schéma de boucle.

Exemple : attaquer (au plus) 10 fois un monstre

```
pour (i variant_de 1 a 10)
faire
    si (Victoire) sortie;
    Attaquer;
ffaire
```

```
pour (i variant_de 1 a 10)
faire
    si (victoire) sortie;
    attaquer;
ffaire
```

Se traduit par :

```
declarer i : entier_naturel;
i <- 1;
tant_que (i < 11)
faire
    si (victoire) sortie;
    attaquer;
    i <- i + 1;
ffaire
```

Ou mieux :

```
declarer i : entier_naturel;
i <- 1;
tant_que (i < 11 ET_ALORS NON victoire)
faire
    attaquer;
    i <- i + 1;
ffaire
```

En revanche, l'utilisation de **continue** dans une boucle **pour** est « *légal* ».

Exemple : afficher une lettre sur deux d'un mot saisi au clavier

Sans instruction **continue**

```
declarer chaine : string;
saisir (chaine);
pour (i variant_de 0 a taille (chaine) - 1)
faire
    si (0 vaut modulo (i,2))
        afficher (chaine[i]);
    fsi
ffaire
```

2 blocs => 2 indentations => 2 fois moins lisible

Avec instruction **continue**

```
declarer chaine : string;
saisir (chaine);
pour (i variant_de 0 a taille (Chaine) - 1)
faire
    si (0 ne_vaut_pas modulo (i,2)) continue;
    afficher (chaine[i]);
ffaire
```

1 bloc => 1 indentation => plus lisible

PLAN

- A. Boucle à compteur
- B. Portée des variables
- C. Sous-programme
- D. Tracer la courbe $y = ax^2 + bx + c$

B. Portée des variables

Exemple : saisir un entier (n) tel que $n > 10$

Bloc répétitif

Durée de vie
(portée) de n

```
declarer n : entier_naturel;
```

```
boucle
```

```
    afficher ("Saisir n");
```

```
    saisir (n);
```

```
    si (n > 10) sortie;
```

```
    afficher ("Plus grand que 10 svp");
```

```
    ligne_suivante;
```

```
fboucle
```

```
//n existe et a une valeur > 10
```

```
boucle
```

```
    declarer n : entier_naturel;
```

```
    afficher ("Saisir n");
```

```
    saisir (n);
```

```
    si (n > 10) sortie;
```

```
    afficher ("Plus grand que 10 svp");
```

```
    ligne_suivante;
```

```
fboucle
```

```
//n n'existe pas
```



PLAN

- A. Boucle à compteur
- B. Portée des variables
- C. Sous-programme
- D. Tracer la courbe $y = ax^2 + bx + c$

C. Sous-programme

C.I Procédure (exemple)

rappel de l'algorithme afficher 100 fois "Bonjour"

```
algorithme afficher100Bonjour
debut

    pour (i variant_de 1 a 100)
    faire

        afficher ("Bonjour") ;

    ffais

fin
```

et si on veut
l'afficher 50 fois ?

tout réécrire ?
=> sous-programme

procedure *afficherNFoisBonjour*

(**n** : **in** entier_naturel)

identificateur du paramètre

son type

debut

Marqueur : paramètre "donnée"

pour (i **variant_de** 1 **a** **n**)
faire

afficher ("Bonjour");

ffaire

fin

c'est une **procédure**

C.2 qu'est-ce que c'est ?

c'est un algorithme qui

- ☐ a toujours un identificateur (nommé)
- ☐ a des specs (spécifications = conditions d'utilisation)
- ☐ a **éventuellement** des paramètres

entre autres ses "ingrédients" :
les données sur lesquelles il travaille

la liste des paramètres est un "guichet" de communication
entre le monde extérieur et l'intérieur du sous-programme

- ☐ "produit" **éventuellement** un résultat

C.3 à quoi ça sert ?

□ à "mettre en facteur" du code

séquence d'instructions souvent utilisée

écrite une seule fois

mise sous forme d'un sous-programme

toutes les occurrences de la séquence sont remplacées par
l'appel du sous-programme

□ à "mettre en facteur" du code

exemple : saisir un texte dans une zone de saisie , c'est gérer

- le déplacement du curseur dans la zone
(passage au caractère précédent ou suivant,
au début ou à la fin de la zone)
- l'affichage ou la suppression des caractères
- l'insertion ou le remplacement
- l'effacement du caractère précédent

□ à "mettre en facteur" du code

exemple : saisir un texte dans une zone de saisie

intervient très souvent :

les différents éditeurs de texte

les cellules d'Excel

nombreuses boîtes de dialogues

de fichiers : open, save, saveAs ...

de légendes : graphiques Excel,

nombreuses zones de texte : explorer

nommer un nouveau dossier

renommer un dossier

□ à "mettre en facteur" du code

avantages

réduire la taille du code

efficience

= économie des moyens

ici ressources matérielles : mémoire/disque

être réutilisé dans du code à venir

réutilisabilité

mise au point **une seule fois**

maintenance

maintenabilité

(modifs, améliorations, corrections)

éviter le "Copier/Coller" source d'erreurs

améliorer la

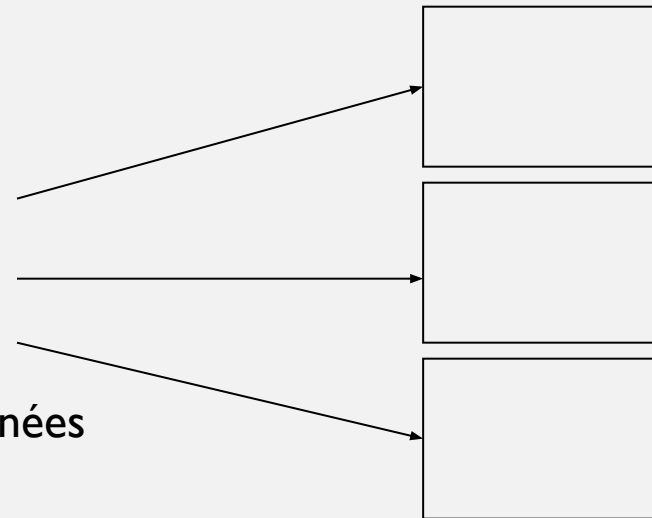
lisibilité

- ❑ à "mettre en facteur" du code
- ❑ à écrire des algorithmes "de haut niveau"

remplacer une suite d'actions élémentaires
par une action "de haut niveau"

exemple :

afficher l'écran d'accueil
établir une connexion
saisir login/password
charger les images
interroger une base de données
...



C.4 comment on s'en sert ?

```
procedure afficherNFoisBonjour  
                (n : in entier_naturel) ;
```

paramètre **formel**

C'est une **déclaration** (ou **interface**, ou **profil** ou **signature**)

précise son mode d'emploi

utilisation

compatibles

```
declarer nbFois : entier_naturel ;
```

```
nbFois <- 12 ;
```

← indispensable car nécessaire au sous-progr.

```
afficherNFoisBonjour (nbFois) ;
```

paramètre **effectif**

C.5 les paramètres

Ils servent à communiquer avec l'extérieur

□ "données" nécessaires au fonctionnement du sous-programme

```
afficherNFoisBonjour (n : in entier_naturel);
```

Marqueur : `in`

Pas de modification du paramètre dans le sous programme

Ils servent à communiquer avec l'extérieur

- ❑ "données" nécessaires au fonctionnement du sous-programme
- ❑ "résultats" produits par le sous-programme

Marqueur : `out`

modification obligatoire du paramètre dans le sous programme

exemple : un sous-programme qui

reçoit une chaîne de caractères,
crée une nouvelle `string` dans laquelle il supprime tous les espaces,
fournit le nombre de suppressions effectuées

```
procedure supprSpaces  
  (strInitiale : in string,  
   strFinale   : out string,  
   nbSuppr     : out entier_naturel) ;
```

```
procedure supprSpaces  
    (strInitiale : in string,
```

on ne sait pas quelle sera la taille de la chaîne à transformer

=> sera de la taille du paramètre effectif

```
    strFinale    : out string,
```

on ne sait pas quelle sera la taille de la chaîne transformée

=> supposée chaîne vide (taille nulle)

```
    sbSuppr      : out entier_naturel) ;
```



Tout paramètre ayant un marqueur out doit être initialisé dès que possible dans un sous programme afin d'éviter les effets de bord



```
procedure supprSpaces
    (strInitiale : in string,
     strFinale   : out string,
     nbSuppr     : out entier_naturel)
debut
    nbSuppr <- 0;
    strFinale <- ""; // au cas où il y ait déjà quelque
chose dedans
    pour (i variant_de 0 a taille (strInitiale) - 1)
    faire
        si (strInitiale[i] vaut ' ') continue;
        strFinale <- strFinale + strInitiale[i];
        nbSuppr <- nbSuppr + 1;
    ffais
fin
```

Ils servent à communiquer avec l'extérieur

❑ "données" nécessaires au fonctionnement du sous-programme

❑ "résultats" produits par le sous-programme

```
procedure supprSpaces
  (strInitiale : in string,
   strFinale   : out string,
   nbSuppr     : out entier_naturel) ;

declarer ligneLue : string;           // par défaut chaîne vide
declarer nouvelleLigne : string;     // par défaut chaîne vide
declarer nbSuppr : entier_naturel;    // non initialisé
saisir lLigneLue);                   // indispensable
                                     donne la véritable taille à LigneLue
nbSuppr <- 10;                         // totalement inutile
supprSpaces ("coucou c'est moi", nouvelleLigne, nbSuppr);
supprSpaces (ligneLue, nouvelleLigne, nbSuppr);
nouvelleLigne et nbSuppr utilisables en sortie du sous-progr.
```

Ils servent à communiquer avec l'extérieur

- ☐ "données" nécessaires au fonctionnement du sous-programme
- ☐ "résultats" produits par le sous-programme
- ☐ "données-résultats" les deux

Marqueur : `in_out`

Modification possible du paramètre dans le sous programme

exemple : un sous-programme qui

reçoit une chaîne de caractères,

dans laquelle il transforme toutes les minuscules en majuscules,
fournit le nombre de transformations effectuées

```
procedure minToMaj  
  (str          : in_out string,  
   nbTransf    : out     entier_naturel) ;
```

```
procedure minToMaj
    (str          : in_out string,
     nbTransf     : out      entier_naturel)
debut
    nbTransf <- 0;
    pour (i variant_de 0 a taille (str) - 1)
        faire
            //si on n'a pas une minuscule, on ne fait rien
            si (NON is_lower(strInitiale[i])) continue;
            //on sait qu'on a une minuscule
            str[i] <- to_upper (str[i]);
            nbTransf <- nbTransf + 1;
        ffaire
    fin
```


C.6 Fonction C'est un sous-programme qui renvoie une instance d'un type.

Soit une procédure qui a au moins un paramètre `out`,
la fonction correspondante renvoie alors ce paramètre `out`

exemple : un sous-programme qui

reçoit un nombre réel,
calcule sa racine carrée

en procédure :

```
procedure sqrt (valInit : in reel,  
               result   : out reel);
```

en fonction :

```
fonction sqrt (valInit : in reel)  
    renvoie reel;
```

Exemple d'utilisation

```
declarer Rac_12 : reel;  
Rac_12 <- sqrt (12.0);
```

```
// fonction qui renvoie la racine carré d'un nombre
// Approximation de  $\sqrt{a}$  à l'aide de suites adjacentes :
http://fr.wikipedia.org/wiki/Racine\_carr%C3%A9e
```

```
fonction sqrt (X : in reel) renvoie reel
debut
```

```
  //la précision
```

```
  declarer eps : reel;
```

```
  eps <- 0.01;
```

```
  declarer u : reel;
```

```
  u <- 1;
```

```
  declarer v: reel;
```

```
  v <- X;
```

```
  //sert à mémoriser l'ancienne valeur de u
```

```
  declarer tmp : reel;
```

```
  tant_que (abs(u-v) > eps)
```

```
  faire
```

```
    tmp <- u;
```

```
    u <- 2 / ((1/u) + (1/v));
```

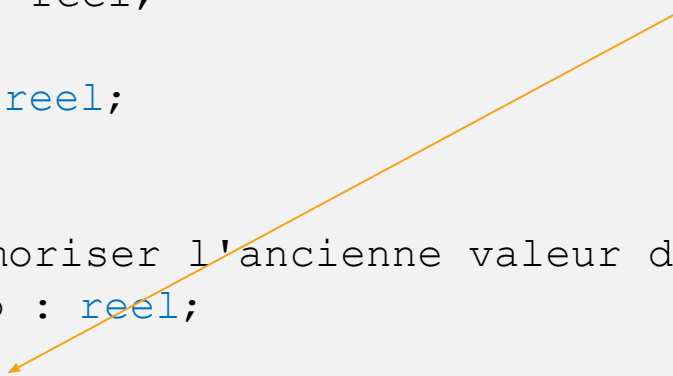
```
    v <- (tmp + v) / 2;
```

```
  ffais
```

```
  renvoie u;
```

```
fin
```

Fonction censée être déjà
connue auparavant



```
// fonction qui renvoie la racine carré d'un nombre
// Approximation de  $\sqrt{a}$  à l'aide de suites adjacentes :
http://fr.wikipedia.org/wiki/Racine\_carr%C3%A9e
```

```
fonction sqrt (X : in reel) renvoie reel
debut
```

```
  //la précision
  declarer eps : reel;
  eps <- 0.01;
```

```
  declarer u : reel;
  u <- 1;
  declarer v: reel;
  v <- X;
```



On a le droit d'appeler un sous-programme dans un sous-programme, à condition que ce dernier soit défini auparavant

```
  //sert à mémoriser l'ancienne valeur de u
  declarer tmp : reel;
```

```
  tant_que (abs(u-v) > eps)
  faire
    tmp <- u;
    u <- 2 / ((1/u) + (1/v));
    v <- (tmp + v) / 2;
```

```
  ffaire
  renvoie u;
```

```
fin
```

C.7 prédicat c'est une fonction qui renvoie un `booléen`

Choisir un identificateur qui commence par `is`

exemple : un sous-programme qui

renvoie vrai si un entier est un multiple d'un autre entier,
tous deux passés en paramètre

profil (= déclaration)

```
fonction isMultiple (entier1 : in entier,  
                    entier2 : in entier)  
renvoie booléen;
```

c'est une fonction qui renvoie un `booléen`

Choisir un identificateur qui commence par `Is`

exemple : un sous-programme qui

renvoie vrai si un entier est un multiple d'un autre entier,
tous deux passés en paramètre

corps (= définition)

```
fonction isMultiple (entier1 : in entier,  
                     entier2 : in entier)  
    renvoie booléen  
debut  
  
    si (modulo (entier1, entier2) vaut 0)  
        renvoie vrai;  
    sinon  
        renvoie faux;  
    fsi  
fin
```

c'est une fonction qui renvoie un `booléen`

Choisir un identificateur qui commence par `Is`

exemple : un sous-programme qui

renvoie vrai si un entier est un multiple d'un autre entier,
tous deux passés en paramètre

corps (= définition)

```
fonction isMultiple (entier1 : in entier,  
                    entier2 : in entier)  
    renvoie booléen  
debut  
  
    renvoie (modulo (entier1, entier2) vaut 0);  
fin
```

c'est une fonction qui renvoie un **boolean**

Choisir un identificateur qui commence par **Is, Has, Are**

exemple : un sous-programme qui
renvoie vrai si un entier est un multiple d'un autre entier,
tous deux passés en paramètre

utilisation (= appel)

```
si (isMultiple (Nombre, 10))  
...
```

Parfois un identificateur qui commence par **has** est plus parlant

```
si (hasValue (Objet))  
...
```

PLAN

- A. Boucle à compteur
- B. Portée des variables
- C. Sous-programme
- D. Tracer la courbe $y = ax^2 + bx + c$

D. Tracer la courbe $y = ax^2 + bx +$

C

Supposition : on dispose de la procédure `traceLigne ()` dont les spécifications sont ??