Concepts de base

Nicolas Delestre
Nicolas.Delestre@insa-rouen.fr
Michel Mainguenaud
Michel.Mainguenaud@insa-rouen.fr

Plan...

- Le formalisme utilisé
- Qu’est ce qu’une variable ?
- Qu’est ce qu’un type ?
- Qu’est ce qu’une expression ?
- Qu’est ce qu’une affectation
- Les entrées / sorties ?

Formalisme...

- Un algorithme doit être lisible et compréhensible par plusieurs personnes
- Il doit donc suivre des règles
- Il est composé d'une entête et d'un corps :
  - l'entête, qui spécifie :
    - le nom de l'algorithme (**Nom :**)
    - son utilité (**Rôle :**)
    - les données "en entrée", c'est-à-dire les éléments qui sont indispensables à son bon fonctionnement (**Entrée :**)
    - les données "en sortie", c'est-à-dire les éléments calculés, produits, par l'algorithme (**Sortie :**)
    - les données locales à l'algorithmique qui lui sont indispensables (**Déclaration :**)

Formalisme...

- le corps, qui est composé :
  - du mot clef **début**
  - d’une suite d’instructions **indentées**
  - du mot clef **fin**

Formalisme...

- Exemple de code :

Nom : addDeuxEntiers

Rôle : Additionner deux entiers a et b et mettre le résultat dans c

Entrée : a,b : entier

Sortie : c : entier

Déclaration : -

début

c ←a+b

fin

Qu’est ce qu’une variable...

- Une variable est une entité qui contient une information :
  - une variable possède un nom, on parle **d’identifiant**
  - une variable possède une valeur
  - une variable possède un type qui caractérise l’ensemble des valeurs que peut prendre la variable
- L’ensemble des variables sont stockées dans la mémoire de l’ordinateur

Qu’est ce qu’une variable...

- On peut faire l’analogie avec une armoire d’archive qui contiendrait des tiroirs étiquetés :
  - l’armoire serait la mémoire de l’ordinateur
  - les tiroirs seraient les variables (l’étiquette corresponderait à l’identidiant)
  - le contenu d’un tiroir serait la valeur de la variable correspondante
  - la couleur du tiroir serait le type de la variable (bleu pour les factures, rouge pour les bons de comande, etc.)

Qu’est ce qu’un type de données...

- Le type d’une variable caractérise :
  - l’ensemble des valeurs que peut prendre la variable
  - l’ensemble des actions que l’on peut effectuer sur une variable
- Lorsqu’une variable apparaît dans l’entête d’un algortithme on lui associe un type en utilisant la syntaxe suivante
  - `Identifiant de la variable :  Son type`
- Par exemple :
  - `age :  Naturel`
  - `nom :  Chaîne de Caractères`
- Une fois qu’un type de données est associé à une variable, cette variable ne peut plus en changer
- Une fois qu’un type de données est associé à une variable le contenu de cette variable doit **obligatoirement** être du même type

Qu’est ce qu’un type de données...

- Par exemple, dans l’exemple précédent on a déclaré `a` et `b` comme des entiers
  - `a` et `b` dans cet algorithme ne pourront pas stocker des réels
  - `a` et `b` dans cet algorithme ne pourront pas changer de type
- Il y a deux grandes catégories de type :
  - les types simples
  - les types complexes (que nous verrons dans la suite du cours)

Les types simples...

- Il y a deux grandes catagories de type simple :
  - Ceux dont le nombres d’éléments est fini, les **dénombrables**
  - Ceux dont le nombre d’éléments est infini, les **indénombrables**

Les types simples dénombrables...

- **booléen**, les variables ne peuvent prendre que les valeurs VRAI ou FAUX
- **intervalle**, les variables ne peuvent prendre que les valeurs entières défi nies dans cet intervalle, par exemple 1..10
- **énuméré**, les variables ne peuvent prendre que les valeurs explicitées, par exemple les jours de la semaine (du lundi au dimanche)
  - Ce sont les seuls types simples qui peuvent être défi nis par l'informaticien
- **caractères**
- Exemples :

```
masculin :  booléen
mois :  1..12
jour :  JoursDeLaSemaine
```

Cas des énumérés...

- Si l'informaticien veut utiliser des énumérés, il doit défi nir le type dans l'entête de l'algorithme en explicitant toutes les valeurs de ce type de la façon suivante :
  - *nom du type = {valeur1, valeur2, ..., valeurn}*
  - Par exemple :
    - JoursDeLaSemaine = {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche}

Les types simples indénombrables...

- **entier** (positifs et négatifs)
- **naturel** (entiers positifs)
- **réel**
- **chaîne de caractères**, par exemple ’cours’ ou ’algorithmique’
- Exemples :

```
age :  naturel
taille :  réel
nom :  chaîne de caractères
```

Opérateur, opérande et expression...

- Un **opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs"
- Une **opérande** est une entité (variable, constante ou expression) utilisée par un opérateur
- Une **expression** est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type

Opérateur, opérande et expression...

- Par exemple dans `a+b` :
  - `a` est l'opérande gauche
  - `+` est l'opérateur
  - `b` est l'operande droite
  - `a+b` est appelé une expression
  - Si par exemple `a` vaut 2 et `b` 3, l'expression `a+b` vaut 5
  - Si par exemple `a` et `b` sont des entiers, l'expression `a+b` est un entier

Opérateur...

- Un opérateur peut être unaire ou binaire :
  - **Unaire** s'il n'admet qu'une seule opérande, par exemple l'opérateur non
  - **Binaire** s'il admet deux opérandes, par exemple l'opérateur +
- Un opérateur est associé à *un* type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de *ce* type
  - Par exemple l'opérateur + ne peut être utilisé qu'avec les types arithmétiques (naturel, entier et réel) ou (exclusif) le type chaîne de caractères
  - **On ne peut pas additionner un entier et un caractère**
  - Toutefois *exceptionnellement* dans certains cas on accepte d'utiliser un opérateur avec deux opérandes de types différents, c'est par exemple le cas avec les types arithmétiques (2+3.5)

Opérateur...

- La signifi cation d’un opérateur peut changer en fonction du type des opérandes
  - Par exemple l’opérateur `+` avec des entiers aura pour sens l’addition, mais avec des chaînes de caractères aura pour sens la **concaténation**
    - `2+3` vaut `5`
    - `"bonjour" + " tout le monde"` vaut `"bonjour tout le monde"`

Les opérateurs booléens...

- Pour les booléens nous avons les opérateurs `non`, `et`, `ou`, `ouExclusif`
  - `non`

a	non a
Vrai	Faux
Faux	Vrai

  - `et`

a	b	a et b
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Les opérateurs booléens...

- `ou`

a	b	a ou b
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

- `ouExclusif`

a	b	a ouExclusif b
Vrai	Vrai	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Les opérateurs sur les énumérés...

- Pour les énumérés nous avons trois opérateurs `succ`, `pred`, `ord` :
  - `succ` permet d'obtenir le successeur, par exemple avec le type `JourDeLaSemaine` :
    - `succ Lundi` vaut `Mardi`
    - `succ Dimanche` vaut `Lundi`
  - `pred` permet d'obtenir le prédecesseur, par exemple avec le type `JourDeLaSemaine` :
    - `pred Mardi` vaut `Lundi`
    - `pred Lundi` vaut `Dimanche`
  - `ord` permet d'obtenir le naturel de l'énuméré spécifi é dans la bijection du type énuméré vers les naturels, par exemple avec le type `JourDeLaSemaine` :
    - `ord Lundi` vaut `0`
    - `ord Dimanche` vaut `6`

Les opérateurs sur les caractères...

- Pour les caractères on retrouve les trois opérateurs des énumérés avec en plus un quatrième opérateur nommé `car` qui est le dual de l'opérateur `ord` avec comme fonction de bijection la table de correspondance de la norme ASCII
  - Cf. http://www.commentcamarche.net/base/ascii.htm
  - Par exemple
    - `ord A` vaut `65`
    - `car 65` vaut `A`
    - `pred A` vaut `@`

Les opérateurs sur les naturels, entiers et réels...

- On retrouve tout naturellemment `+`, `-`, `/`, `*`
- Avec en plus pour les naturels et les entiers `div` et `mod`, qui permettent respectivement de calculer une division entière et le reste de cette division, par exemple :
  - `11 div 2` vaut `5`
  - `11 mod 2` vaut `1`

L’opérateur pour les chaînes de caractères...

- C’est l’opérateur de concaténation vu précédemment qui est +

L’opérateur d’égalité, d’inégalité, etc...

- L’opérateur d’égalité
  - C’est l’opérateur que l’on retrouve chez tous les types simples qui permet de savoir si les deux opérandes sont égales
  - Cet opérateur est représenté par le caractère $=$
  - Une expression contenant cet opérateur est un booléen
- On a aussi l’opérateur d’inégalité $\neq$
- Et pour les types possédant un ordre les opérateurs de comparaison $<, \leq, \geq, >$

Priorité des opérateurs...

- Tout comme en arithmétique les opérateurs ont des priorités
  - Par exemple * et / sont prioritaires sur + et -
- Pour les booléens, la priorité des opérateurs est `non`, `et`, `ouExclusif` et `ou`
- Pour clarifi er les choses (ou pour dans certains cas supprimer toutes ambiguités) on peut utiliser des parenthèses

Actions sur les variables...

- On ne peut faire que deux choses avec une variable :
  1. Obtenir son contenu (*regarder le contenu du tiroir*)
     - Cela s'effectue simplement en nommant la variable
  2. Affecter un (nouveau) contenu (mettre une (nouvelle) information dans le tiroir)
     - Cela s'effectue en utilisant l'opérateur d'affectation représenter par le symbole ←
     - La syntaxe de cet opérateur est :
       - `identifiant de la variable` ← `expression sans operateur d'affectation`

Actions sur les variables...

- Par exemple l'expression `c ← a + b` se comprend de la façon suivante :
  - On prend la valeur contenue dans la variable `a`
  - On prend la valeur contenue dans la variable `b`
  - On additionne ces deux valeurs
  - On met ce résultat dans la variable `c`
  - Si `c` avait auparavant une valeur, cette dernière est perdue !

Les entrées/sorties...

- Un algorithme peut avoir des interactions avec l'utilisateur
- Il peut afficher un résultat (du texte ou le contenu d'une variable) et demander à l'utilisateur de saisir une information afin de la stocker dans une variable
- En tant qu'informaticien on raisonne en se mettant **"à la place de la machine"**, donc :
  - Pour affi cher une information on utilise la commande **écrire** suivie entre parenthèses de la chaîne de caractères entre guillemets et/ou des variables de type simple à affi cher séparées par des virgules, par exemple :
    - **écrire**("Le valeur de la variable a est", a)
  - Pour donner la possibilité à l'utilisateur de saisir une information on utilise la commande **lire** suivie entre parenthèses de la variable de type simple qui va recevoir la valeur saisie par l'utilisateur, par exemple :
    - **lire**(b)

Exemple d’algorithme...

Nom : euroVersFranc1

Rôle : Convertisseur des sommes en euros vers le franc, avec saisie de la somme en euro et affi chage de la somme en franc

Entrée : -

Sortie : -

Déclaration : valeurEnEuro,valeurEnFranc,tauxConversion : Réel

début

```
    tauxConversion ←6.55957
    écrire("Votre valeur en euro :")
    lire(valeurEnEuro)
    valeurEnFranc ←valeurEnEuro * tauxConversion
    écrire(valeurEnEuro," euros = ",valeurEnFranc," Frs")
```

fin

Exemple d’algorithme...

Nom : euroVersFranc2

Rôle : Convertisseur des sommes en euros vers le franc

Entrée : valeurEnEuro : Réel

Sortie : valeurEnFranc : Réel

Déclaration : tauxConversion : Réel

début

tauxConversion ←6.55957

valeurEnFranc ←valeurEnEuro * tauxConversion

fin

Variables (locales et globales), fonctions et procédures

Nicolas Delestre
Nicolas.Delestre@insa-rouen.fr
Michel Mainguenaud
Michel.Mainguenaud@insa-rouen.fr

INSA de Rouen

Plan...

- Rappels
- Les sous-programmes
- Variables locales et variables globales
- Structure d’un programme
- Les fonctions
- Les procédures

Vocabulaire...

- Dans ce cours nous allons parler de “programme” et de “sous-programme”
- Il faut comprendre ces mots comme “programme algorithmique” indépendant de toute implantation

Rappels...

- La méthodologie de base de l’informatique est :
  1. Abstraire
     - Retarder le plus longtemps possible l’instant du codage
  2. **Décomposer**
     - "...diviser chacune des difficultés que j’examinerai en autant de parties qu’il se pourrait et qu’il serait requis pour les mieux résoudre." Descartes
  3. Combiner
     - Résoudre le problème par combinaison d’abstractions

Par exemple...

- Résoudre le problème suivant :
  - Écrire un programme qui affiche le nombres parfaits compris entre 0 et une valeur n saisie par l'utilisateur
- Revient à résoudre les problèmes suivants :

  Demander à l'utilisateur de saisir un entier n

  Affi cher les nombres parfaits compris 0 et n

  Savoir si un nombre donné est parfait

  Calculer la somme des diviseurs d'un nombre

  Savoir si un nombre est diviseur d'un autre nombre
- Chacun de ces sous-problèmes devient un nouveau problème à résoudre
- Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait "quasiment" résoudre le problème initial

Sous-programme...

- Donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial
- En algorithmique il existe deux types de sous-programmes :
  - Les fonctions
  - Les procédures
- Un sous-programme est obligatoirement caractérisé par un nom (un identifi ant) unique
- Lorsqu'un sous programme a été explicité (on a donné l'algorithme), son nom devient une nouvelle instruction, qui peut être utilisé dans d'autres (sous-)programmes
- Le (sous-)programme qui utilise un sous-programme est appelé **(sous-)programme appelant**

Règle de nommage...

- Nous savons maintenant que les variables, les constantes, les types défi nis par l'utilisateur (comme les énumérateurs) et que les sous-programmes possèdent un nom
- Ces noms doivent suivre certaines rêgles :
  - Ils doivent être explicites (à part quelques cas particuliers, comme par exemple les variables *i* et *j* pour les boucles)
  - Ils ne peuvent contenir que des lettres et des chiffres
  - Ils commencent obligatoirement par une lettre
  - Les variables et les sous-programmes commencent toujours par une miniscule
  - Les types commencent toujours par une majuscule
  - Les constantes ne sont composées que de majuscules
  - Lorsqu'ils sont composés de plusieurs mots, on utilise les majuscules (sauf pour les constantes) pour séparer les mots (par exemple JourDeLaSemaine)

Les différents types de variable...

- Défi nitions :
  - La **portée** d'une variable est l'ensemble des sous-programmes où cette variable est connue (les instructions de ces sous-programmes peuvent utiliser cette variable)
  - Une variable définie au niveau du programme principal (celui qui résoud le problème initial, le problème de plus haut niveau) est appelée **variable globale**
    - Sa portée est totale : **tout** sous-programme du programme principal peut utiliser cette variable
  - Une variable définie au sein d'un sous programme est appelée **variable locale**
    - La portée d'un variable locale est uniquement le sous-programme qui la déclare
- Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée
  - Dans ce sous-programme la variable globale devient inaccessible

Structure d’un programme...

- Un programme doit suivre la structure suivante :

  Programme *nom du programme*

  Définition des constantes

  Définition des types

  Déclaration des variables globales

  Définition des sous-programmes

  début

  instructions du programme principal

  fin

Les paramètres formels/effectifs...

- Vocabulaire
  - Un *paramètre formel* est aussi appelé aussi *paramètre*
  - Un *paramètre effectif* est aussi appelé *argument*
- Signification
  - Un paramètre formel d'un sous-programme est une variable locale particulière
    - Il a donc un type
  - Un paramètre effectif est une variable ou constante (numérique ou définie par le programmeur)
  - Le paramètre formel et le paramètre effectif sont associé lors de l'appel du sous-programme
    - Le paramètre formel et effectif doivent donc être de même type

Les paramètres formels/effectifs...

- Par exemple, si le sous-programme *sqr* permet de calculer la racine carrée d'un réel:
  - Ce sous-programme admet un seul paramètre formel de type réel positif
  - Le (sous-)programme qui utilise *sqr* doit donner le réel positif dont il veut calculer la racine carrée, cela peut être :
    - une variable, par exemple *a*
    - une constante, par exemple *5.25*

Les passages de paramètres...

- Il existe trois types d'association (que l'on nomme **passage de paramètre**) entre le paramètre formel et le paramètre effectif du (sous-)programme appelant :
  - Le **passage de paramètre en entrée**
  - Le **passage de paramètre en sortie**
  - Le **passage de paramètre en entrée/sortie**

Le passage de paramètres en entrée...

- Les instructions du sous-programme ne peuvent pas modifi er le paramètre effectif
  - En fait c'est la **valeur** du paramètre effectif qui est copiée dans le paramètre formel
  - C'est le `seul` passage de paramètre qui admet l'utilisation d'une constante
- Par exemple :
  - le sous-programme *sqr* permettant de calculer la racine carrée d'un nombre admet un paramètre en entrée
  - le sous-programme **écrire** qui permet d'afficher des informations admet n paramètres en entrée

Le passage de paramètres en sortie...

- Les instructions du sous-programme affectent obligatoirement une valeur à ce paramètre formel (valeur qui est donc aussi affectée au paramètre effectif)
- Il y a donc une liaison forte entre la paramètre formel et le paramètre effectif
  - C’est pour cela qu’on ne peut pas utiliser de constante pour ce type de paramètre
- La valeur que pouvait posséder la variable utilisée comme paramètre effectif n’est pas utilisée par le sous-programme
- Par exemple :
  - le sous-programme **lire** qui permet de mettre dans des variables des valeurs saisies par l’utilisateur admet n paramètres en sortie

Le passage de paramètres en entrée/sortie...

- Passage de paramètre qui combine les deux précédentes
- À utiliser lorsque le sous-programme doit utiliser et/ou modifi er la valeur de la variable du (sous-)programme appelant
- Comme pour le passage de paramètre en sortie, on ne peut pas utiliser de constante
- Par exemple :
  - le sous-programme **échanger** qui permet d'échanger les valeurs de deux variables

Les fonctions...

- Les fonctions sont des sous-programmes admettant des paramètres et retournant un **seul** résultat (comme les fonctions mathématiques y=f(x,y,. . .))
  - les paramètres sont en nombre fi xe (≥ 0)
  - une fonction possède un seul type, qui est le type de la valeur retournée
  - le passage de paramètre est **uniquement en entrée** : c'est pour cela qu'il n'est pas précisé
    - lors de l'appel, on peut donc utiliser comme paramètre des variables, des constantes mais aussi des résultats de fonction
  - la valeur de retour est spécifi ée par l'instruction **retourner**
- Généralement le nom d'une fonction est soit un nom (par exemple *minimum*), soit une question (par exemple *estVide*)

Les fonctions...

- On déclare une fonction de la façon suivante :

  fonction *nom de la fonction* **(***paramètre(s) de la fonction***) :** *type de la valeur retournée*

  Déclaration *variable locale 1 : type 1; . . .*

  début

  instructions de la fonction avec au moins une fois l'instruction ***retourner***

  fin

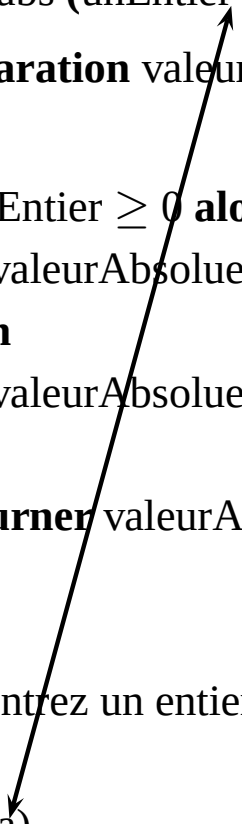
- On utilise une fonction en précisant son nom suivi des paramètres entre parenthèses
  - Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètre

Exemple de déclaration de fonction...

```
fonction abs (unEntier : Entier) : Entier
début
    si unEntier ≥ 0 alors
        retourner unEntier
    sinon
        retourner -unEntier
    finsi
fin
```

Exemple de programme...

```
Programme exemple1
    Déclaration a : Entier, b : Naturel
    fonction abs (unEntier : Entier) : Naturel
        Déclaration valeurAbsolue : Naturel
    début
        si unEntier ≥ 0 alors
            valeurAbsolue ← unEntier
        sinon
            valeurAbsolue ← -unEntier
        fi nsi
        retourner valeurAbsolue
    fi n
début
    écrire("Entrez un entier :")
    lire(a)
    b ← abs(a)
    écrire("la valeur absolue de ",a," est ",b)
fi n
```



Lors de l'exécution de la fonction *abs*, la variable *a* et le paramètre *unEntier* sont associés par un passage de paramètre en entrée : La valeur de *a* est copiée dans *unEntier*

Un autre exemple...

```
fonction minimum2 (a,b : Entier) : Entier
début
    si a ≥ b alors
        retourner b
    sinon
        retourner a
    finsi
fin
fonction minimum3 (a,b,c : Entier) : Entier
début
    retourner minimum2(a,minimum2(b,c))
fin
```

Les procédures...

- Les procédures sont des sous-programmes qui ne retournent **aucun** résultat
- Par contre elles admettent des paramètres avec des passages :
  - en entrée, préfi xés par **Entrée** (ou **E**)
  - en sortie, préfi xés par **Sortie** (ou **S**)
  - en entrée/sortie, préfi xés par **Entrée/Sortie** (ou **E/S**)
- Généralement le nom d’une procédure est un verbe

Les procédures...

- On déclare une procédure de la façon suivante :

  procédure *nom de la procédure* **(E** *paramètre(s) en entrée;* **S** *paramètre(s) en sortie;* **E/S** *paramètre(s) en entrée/sortie* **)**

  Déclaration *variable(s) locale(s)*

  début

  instructions de la procédure

  fin

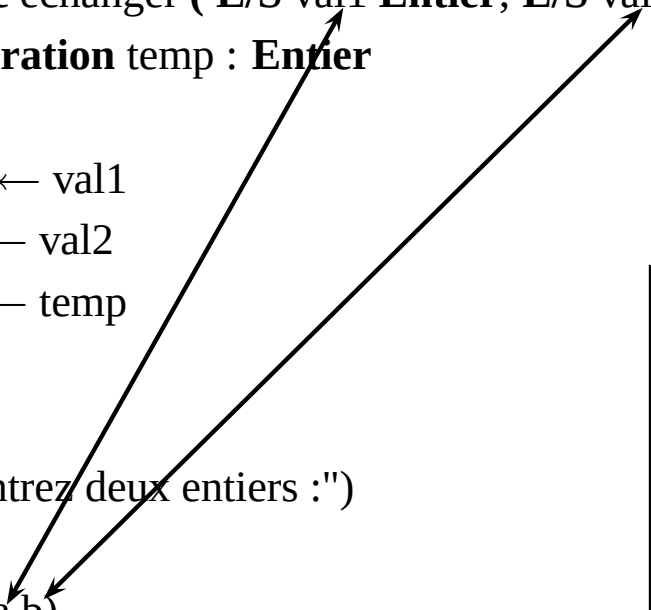
- Et on appelle une procédure comme une fonction, en indiquant son nom suivi des paramètres entre parenthèses

Exemple de déclaration de procédure...

procédure calculerMinMax3 **(E** a,b,c : **Entier** ; **S** m,M : **Entier)**
début
    m ← minimum3(a,b,c)
    M ← maximum3(a,b,c)
fin

Exemple de programme...

```
Programme exemple2
    Déclaration a : Entier, b : Naturel
    procédure echanger ( E/S val1 Entier; E/S val2 Entier;)
        Déclaration temp : Entier
    début
        temp ← val1
        val1 ← val2
        val2 ← temp
    fin
début
    écrire("Entrez deux entiers :")
    lire(a,b)
    echanger(a,b)
    écrire("a=",a," et b = ",b)
fin
```



Lors de l'exécution de la procédure *echanger*, la variable *a* et le paramètre *val1* sont associés par un passage de paramètre en entrée/sortie : Toute modification sur val1 est effectuée sur a (de même pour b et val2)

Autre exemple de programme...

```
Programme exemple3
    Déclaration entier1,entier2,entier3,min,max : Entier
    fonction minimum2 (a,b : Entier) : Entier
    ...
    fonction minimum3 (a,b,c : Entier) : Entier
    ...
    procédure calculerMinMax3 ( E a,b,c : Entier ; S min3,max3 : Entier )
    début
        min3 ← minimum3(a,b,c)
        max3 ← maximum3(a,b,c)
    fin
début
    écrire("Entrez trois entiers :")
    lire(entier1)  ;
    lire(entier2)  ;
    lire(entier3)
    calculerMinMax3(entier1,entier2,entier3,min,max)
    écrire("la valeur la plus petite est ",min," et la plus grande est ",max)
fin
```

Programme affi chant des nombres partfaits...

Programme *affi chage les nombres parfaits compris entre de 1 à nb*

Déclaration nb : **Naturel**

fonction saisirMax **() : Naturel**

. . .

fonction estUnDiviseur **(**a,b : **Naturel) : Booléen**

. . .

fonction calculerSommeDesDiviseurs **(**n : **Naturel) : Naturel**

. . .

fonction estParfait **(**n : **Naturel) : Booléen**

. . .

procédure affi cherNbsParfaits **(E** max : **Naturel)**

. . .

début

nb ← saisirMAX

affi cherNbParfait(nb)

fi n

Programme affi chant des nombres partfaits...

```
fonction saisirMax () : Naturel
    Déclaration resultat : Naturel
début
    écrire("Valeur maximale d'affi chage des nombres parfaits")
    lire(resultat)
    retourner resultat
fi n
fonction estUnDiviseur (a,b : Naturel) : Booléen
début
    retourner a mod b=0
fi n
```

Programme affi chant des nombres partfaits...

```
fonction calculerSommeDesDiviseurs (n : Naturel) : Naturel
    Déclaration i : Naturel; somme : Naturel
début
    somme ← 0
    pour i ←1 à n div 2 faire
        si estUnDiviseur(n,i) alors
            somme ← somme+i
        fi nsi
    fi npour
    retourner somme
fi n
fonction estParfait (n : Naturel) : Booléen
début
    retourner n=calculerSommeDesDiviseurs(n)
fi n
```

Programme affi chant des nombres partfaits...

```
procédure affi cherNbsParfaits ( E max : Naturel )
    Déclaration i : Naturel
début
    pour i ←1 à max faire
        si estParfait(i) alors
            écrire(i)
        fi nsi
    fi npour
fi n
```

Conditionnelles et itérations

Nicolas Delestre
Nicolas.Delestre@insa-rouen.fr
Michel Mainguenaud
Michel.Mainguenaud@insa-rouen.fr

Plan...

- Rappels sur la logique
- Les conditionnelles
- Les itérations

Rappels sur la logique booléenne...

- Valeurs possibles : `Vrai` ou `Faux`
- Opérateurs logiques : `non` `et` `ou`
  - optionellement `ouExclusif` mais ce n'est qu'une combinaison de `non` , `et` et `ou`
  - a `ouExclusif` b = (`non` a `et` b) ou (a `et` `non` b)
- Priorité sur les opérateurs : `non` , `et` , `ou`
- 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`

Rappels sur la logique booléenne...

- 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
  - `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`

Les conditionnelles...

- Jusqu'à présent les instructions d'un algorithme étaient **toutes** interprétées **séquentiellement**
  - **Nom :** euroVersFranc2
    Rôle : Convertisseur des sommes en euros vers le franc
    Entrée : valeurEnEuro : Réel
    Sortie : valeurEnFranc : Réel
    Déclaration : tauxConversion : Réel
    début
        tauxConversion ← 6.55957
        valeurEnFranc ← valeurEnEuro * tauxConversion
    fin
- Mais il se peut que l'on veuille conditionner l'exécution d'un algorithme
  - Par exemple la résolution d'une équation du second degré est conditionnée par le signe de Δ

L'instruction `si alors sinon`…

- L'instruction `si alors sinon` permet de conditionner l'éxecution d'un algorithme à la valeur d'une expression booléenne
- Sa syntaxe est :

  si *expression booléenne* **alors**

  suite d'instructions exécutées si l'expression est vrai

  sinon

  suite d'instructions exécutées si l'expression est fausse

  finsi

- Le deuxième partie de l'instruction est optionnelle, on peut avoir la syntaxe suivante :

  si *expression booléenne* **alors**

  suite d'instructions exécutées si l'expression est vrai

  finsi

Exemple...

Nom : abs

Rôle : Calcule la valeur absolue d'un entier

Entrée : unEntier : **Entier**

Sortie : laValeurAbsolue : **Entier**

Déclaration : -

début

    si unEntier $\geq$ 0 **alors**

        laValeurAbsolue $\leftarrow$unEntier

    sinon

        laValeurAbsolue $\leftarrow$-unEntier

    fi nsi

fi n

Exemple...

Nom : max

Rôle : Calcule le maximum de deux entiers

Entrée : lEntier1,lEntier2 : **Entier**

Sortie : leMaximum : **Entier**

Déclaration : -

début

    si lEntier1 < lEntier2 **alors**

        leMaximum ←lEntier2

    sinon

        leMaximum ←lEntier1

    fi nsi

fi n

Exemple...

Nom : max

Rôle : Calcule le maximum de deux entiers

Entrée : lEntier1,lEntier2 : **Entier**

Sortie : leMaximum : **Entier**

Déclaration : -

début

leMaximum ←lEntier1

si lEntier2 > lEntier1 **alors**

leMaximum ←lEntier2

fi nsi

fi n

L'instruction `cas`...

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

```
si a=1 alors
    faire une chose
sinon
    si a=2 alors
        faire une autre chose
    sinon
        si a=4 alors
            faire une autre chose
        sinon
            ...
        finsi
    finsi
finsi
```

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

L'instruction `cas`...

- Sa syntaxe est :
  cas où v **vaut**
  $v1$: $action_1$
  $v2_1$,$v2_2$,...,$v2_m$: $action_2$
  $v3_1$ ... $v3_2$: $action_3$
  ...
  vn : $action_n$
  autre : $action$
  fi ncas
- où :
  - v1,...,vn sont des **constantes** de type **scalaire** (entier, naturel, enuméré, ou caractère)
  - $action_i$ est exécutée si v = vi (on quitte ensuite l'instruction cas)
  - *action* est exécutée si $\forall$ i, v $\neq$ vi

Exemple...

Nom : moisA30Jours

Rôle : Détermine si un mois à 30 jours

Entrée : mois : **Entier**

Sortie : resultat : **Booléen**

Déclaration :

début

    cas où mois **vaut**

        4,6,9,11 : resultat ←Vrai

        autre : resultat ←Faux

    fi ncas

fi n

Les itérations...

- Lorsque l'on veut répéter plusieurs fois un même traitement, plutôt que de copier n fois la ou les instructions, on peut demander à l'ordinateur d'éxecuter n fois un morceau de code
- Il existe deux grandes catégories d'itérations :
  - Les itérations **déterministes** : le nombre de boucle est défini à l'entrée de la boucle
  - les itérations **indéterministes** : l'exécution de la prochaine boucle est conditionnée par une expression booléenne

Les itérations déterministes...

- Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction `pour`
- Sa syntaxe est :

  pour *identifiant d'une variable de type scalaire* ←*valeur de début* **à** *valeur de fin*

  faire

  instructions à exécuter à chaque boucle

  finpour
- dans ce cas la variable utilisée prend successivement les valeurs comprises entre *valeur de début* et *valeur de fi n*

Exemple...

Nom : somme

Rôle : Calculer la somme des n premiers entiers positifs, s=0+1+2+. . . +n

Entrée : n : **Naturel**

Sortie : s : **Naturel**

Déclaration : i : **Naturel**

début

    s ← 0

    pour i ←0 **à** n **faire**

        s ← s+i

    fi npour

fi n

Les itérations indéterministes...

- Il existe deux instructions permettant de faire des boucles indéterministes :
  - L'instruction **tant que** :

    tant que *expression booléenne* **faire**

    instructions

    fintantque

    - qui signifie que tant que l'expression booléenne est vraie on exécute les instructions
  - L'instruction **répéter jusqu'à ce que** :

    répéter

    instructions

    jusqu'à ce que *expression boolèenne*

    - qui signifie que les instructions sont exécutées jusqu'à ce que l'expression booléenne soit vraie

Les itérations indéterministes...

- À la différence de l'instruction *tant que*, dans l'instruction *répeter jusqu'à ce que* les instructions sont exécutées au moins une fois
- Si vous ne voulez pas que votre algorithme "tourne" indéfi niment, l'expression booléenne doit faire intervenir des variables dont le contenu doit être modifi é par au moins une des instructions du corps de la boucle

Un exemple...

Nom : invFact

Rôle : Détermine le plus grand entier e telque e! $\leq$ n

Entrée : n : **Naturel**$\geq$ 1

Sortie : e : **Naturel**

Déclaration : fact : **Naturel**

début

fact $\leftarrow$ 1

e $\leftarrow$ 1

tant que fact $\leq$ n **faire**

e $\leftarrow$ e+1

fact $\leftarrow$ fact*e

fintantque

e $\leftarrow$ e-1

fin

Explication avec n=10...

Nom : invFact
Rôle : Détermine le plus grand entier e telque e! $\leq$ n
Entrée : n : **Naturel**$\geq$ 1
Sortie : e : **Naturel**
Déclaration : fact : **Naturel**
début
    fact $\leftarrow$ 1
    e $\leftarrow$ 1
    tant que fact $\leq$ n **faire**
        e $\leftarrow$ e+1
        fact $\leftarrow$ fact*e
    fintantque
    e $\leftarrow$ e-1
fin

	n	e	fact	fact $\leq$ n
fact $\leftarrow$1	10	?	1	vrai
e $\leftarrow$1	10	1	1	vrai
e $\leftarrow$e+1 fact $\leftarrow$fact*e	10	2	2	vrai
e $\leftarrow$e+1 fact $\leftarrow$fact*e	10	3	6	vrai
e $\leftarrow$e+1 fact $\leftarrow$fact*e	10	4	24	faux
e $\leftarrow$e-1	10	3	24	faux

Un autre exemple...

Nom : calculerPGCD

Rôle : Calculer le pgcd(a,b) à l'aide de l'algorithme d'euclide

Entrée : a,b : **Naturel** non nul

Sortie : pgcd : **Naturel**

Déclaration : reste : **Naturel**

début

    répéter

        reste ← a mod b

        a ← b

        b ← reste

    jusqu'à ce que reste=0

    pgcd ← a

fi n

De l’algorithmique au C

Nicolas Delestre
Nicolas.Delestre@insa-rouen.fr
Michel Mainguenaud
Michel.Mainguenaud@insa-rouen.fr

INSA de Rouen

Plan...

- Notes
- Un langage compilé
- Un langage basé sur des modules
- Traduire les types de bases
- Traduire les constantes
- Traduire les opérateurs usuels
- Traduire les instructions simples et composées
- Traduire les conditionnelles
- Traduire les itératifs
- Les pointeurs
- Traduire les fonctions et les procédures
- Traduire `lire` ou `écrire`
- Traduire un programme
- Un exemple

Notes...

- Ce cours n’est pas un cours présentant tous les concepts du C
- Il présente seulement la manière de traduire proprement un programme algorithmique en programme C
- Vous ne trouverez donc pas dans ce cours toutes les subtilités du C, mais vous pouvez vous référer à des cours sur le langage C aux adresses suivantes :
  - `http://www.enstimac.fr/~gaborit/lang/CoursDeC/`
  - `http://www-rocq.inria.fr/codes/Anne.Canteaut/COURS_C/`
  - `http://www-ipst.u-strasbg.fr/pat/program/tpc.htm`
  - `http://www710.univ-lyon1.fr/~bonnev/CoursC/CoursC.html`
  - …

Un langage compilé...

- Le C est un langage compilé
- Les compilateurs C transforment un programme C (fichier suffixé par .c) en programme objet (fichier suffixé par .o) en deux phases :
  1. Le préprocesseur agit sur les macros (commandes précédées d'un #) et transforme le code source C en un autre code source C (ne contenant plus aucune macro) en remplaçant le code macro par son évaluation
     - Par exemple si le code C contient l'instruction `#define PI 3.14159`, le préprocesseur remplacera dans le code source la chaîne de caractères `PI` par la chaîne de caractères `3.14159` à partir de la position du `#define`
  2. Le compilateur transforme ce deuxième code source C en programme machine (nommé code objet)
- Sous Linux (et généralement sous unix), on utilise le compilateur gcc (GNU C Compiler) avec l'option -c pour compiler

Un langage compilé...

- Par exemple pour compiler le programme *helloworld.c* suivant (dont on expliquera le contenu plus tard), il faut taper *gcc -c helloworld.c*, on obtient alors le fichier *helloworld.o*

```
#include <stdio.h>

int main(){
   printf("Hello world");
}
```

Un langage basé sur des modules...

- Le langage C utilise le concept de module (ou de librairie) qui permet de créer et d'utiliser des bibliothèques de fonctions qui peuvent être utilisées dans plusieurs programmes
- De ce fait, le code objet produit par le compilateur n'est pas un programme exécutable car il utilise certainement des fonctions définies dans des modules
- Pour le rendre exécutable il faut le lier aux modules adéquates : c'est ce que l'on nomme l'édition des liens (ou *linkage*)
- Cette édition des liens s'effectue toujours avec gcc (sans option) suivi des codes objets du programme et des modules (non standards) utilisés

Un langage basé sur des modules...

- Par exemple pour linker *helloworld.o* on tape *gcc helloworld.o*
  - Il n'y a que *helloworld.o* car le seul module utilisé est le module *stdio.h* qui est un module standard
- Le programme exécutable *a.out* est alors créé
- Si à la place de l'exécutable *a.out*, on veut obtenir l'excutable *helloworld* il faut utiliser l'option *-o* suivi du nom du fichier exécutable désiré (ici *-o helloworld*)
- Remarque :
  - Si le programme n'utilise aucune librairie non standard, on peut compiler et linker directement en utilisant gcc suivi du fichier C (avec optionnellement l'option -o), par exemple :
    - *gcc -o helloworld helloworld.c*

Traduire les types simples...

- Le C propose les types simples suivants :
  - int, long, short, float, double, char
- On peut donc suivre les règles de traduction suivantes :

Algorithmique	C
Entier,Naturel	int, long, short
Réel	float, double
Caractère	char
Booléen	int (1=Vrai, 0=Faux)
Chaîne de caractères	*Voir cours sur les tableaux*

Traduire les constantes...

- Pour traduire les constantes entières, le C propose trois notations :
  1. La notation décimale, en base dix, par exemple 379
  2. La notation octale, en base huit, qui doit commencer par un zéro par exemple 0573
  3. Le notation hexadécimale, en base seize, qui doit commencer par un zéro suivi d'un x (ou X), par exemple 0X17B (ou 0x17B, 0X17b, 0x17b)
- Les constantes caractères sont entourées de quote, par exemple 'a'
  - Certains caractères (les caractères non imprimables, avec un code ASCII inférieur à 32, le \, le ') ne sont utilisables qu'en préfixant par un \:
    - le code ASCII du caractère (exprimé en octal), par exemple '\001'
    - un caractère tel que '\n', '\t', '\\', '\"

Traduire les constantes...

- Pour les constantes réelles, on utilise le “.” pour marquer la virgule et le caractère “e” suivi d’un nombre entier, ici a, pour représenter 10^a, par exemple :
  - 2. , .3, 2e4, 2.3e4
- les constantes chaîne de caractères doivent être entourées de guillemet, par exemple "une chaine"
  - Par défaut le compilateur ajoute à la fin de la chaîne de caractères ’\0’, qui est le marqueur de fin de chaîne
- Il n’y a pas de constante nommée en C, il faut utiliser la commande `#define` du préprocesseur

Traduire les opérateurs...

- On traduit les opérateurs en respectant les règles suivantes :

Algorithmique	C
	=
$=, \neq$	==, !=
$<, \leq, >, \geq$	<, <=, >, >=
et, ou, non	&&, ‖, !
+, -, *, /	+, -, *, /
div, mod	/, %

Traduire les opérateurs...

- Les opérateurs unaires `++` et `--` sont des opérateurs particuliers qui peuvent avoir jusqu'à deux effets de bord:
  - En dehors de toute affectation, elle incrémente l'opérande associée, par exemple
    - `i++` et `++i` sont équivalents à `i=i+1`
  - Lorsqu'ils sont utilisés dans une affectation, tout dépend de la position de l'opérateur par rapport à l'opérande, par exemple :
    - `j=i++` est équivalent à `j=i;i=i+1;`
    - `j=++i` est équivalent à `i=i+1;j=i;`

Les instructions...

- Il existe deux types d'instructions en C :
  - Les instructions simples (expression, appel de fonction, etc.)
    - Elles finissent toujours par un ';'
    - Par exemple :

```
a = a + 1;
```

  - Les instructions composées qui permettent de considérer une succession d'instructions comme étant une seule instruction
    - Elles commencent par "{" et finissent par "}"
    - Par exemple :

```
{ a = a + 1; b = b + 2; }
```

Traduire les variables...

- On déclare en C une variable en la précédant de son type, par exemple :

```
int a;
float x;
char c;
```

- Une variable globale est défi nie en dehors de toute fonction
  - Sa portée est le fi chier où elle est défi nie à partir de sa défi nition (pas au dessus)
- Une variable locale est défi nie à l'intérieur d'une fonction
  - Sa portée est uniquement la fonction ou elle a été défi nie

Les conditionnelles...

- L'instruction `si...alors...sinon` est traduit par l'instruction `if`, qui a la syntaxe suivante :

```
if (condition)
   // instruction(s) du if
[else
   // instruction(s) du else
]
```

  - Par exemple :

```
if (a<10)
   a=a+1;
else
   a=a+2;
```

Les conditionnelles...

- Remarque :
  - Lorsqu'il y ambiguité sur la porté du `else` d'une instruction `if`, le `else` dépend toujours du `if` le plus proche
  - Par exemple :

```
if (a>b) if (c<d) u=v; else i=j;
```

  - Le mieux étant toutefois de clarifi er cette ambiguité :

```
if (a>b)
   if (c<d)
      u=v;
   else
      i=j;
```

```
if (a>b) {
   if (c<d)
      u=v;
} else {
   i=j;
}
```

Les conditionnelles...

- L'instruction `cas..où` est traduite par l'instruction `switch`, qui a la syntaxe suivante :

```
switch (leSelecteur) {
   case cas1 :
      //instruction(s) du cas 1
      break;
   case cas2 :
      //instruction(s) du cas 2
      break;
   ...
   default :
      //instruction(s) du default
}
```

Les conditionnelles...

- Par exemple :

```
switch(choix) {
    case 't' : printf("vous voulez un triangle"); break;
    case 'c' : printf("vous voulez un carre"); break;
    case 'r' : printf("vous voulez un rectangle"); break;
    default : printf("erreur. recommencez !");
}
```

Traduire les itérations...

- L'instruction `Pour` est traduite par l'instruction `for`, qui a la syntaxe suivante :

```
for(initialisation ;
    condition_d_arret ;
    operation_effectuée_à_chaque_itération)
  instruction;
```

  - Par exemple :

```
for(i=0;i<10;i++)
   printf("%d\n", i);
```

  - Remarque :
    - Contrairement à l'algortihmique le `for` est une itération indéterministe

Traduire les itérations...

- L'instruction `Tant...que` est traduite par l'instruction `while`, qui a la syntaxe suivante :

```
while(condition)
    instruction;
```

  - Par exemple :

```
i=0;
while(i<10){
    printf("%d\n", i);
    i++;
}
```

Traduire les itérations...

- L'instruction `répéter` est traduite par l'instruction `do..while`, qui a la syntaxe suivante :

```
do
    instruction;
while(condition);
```

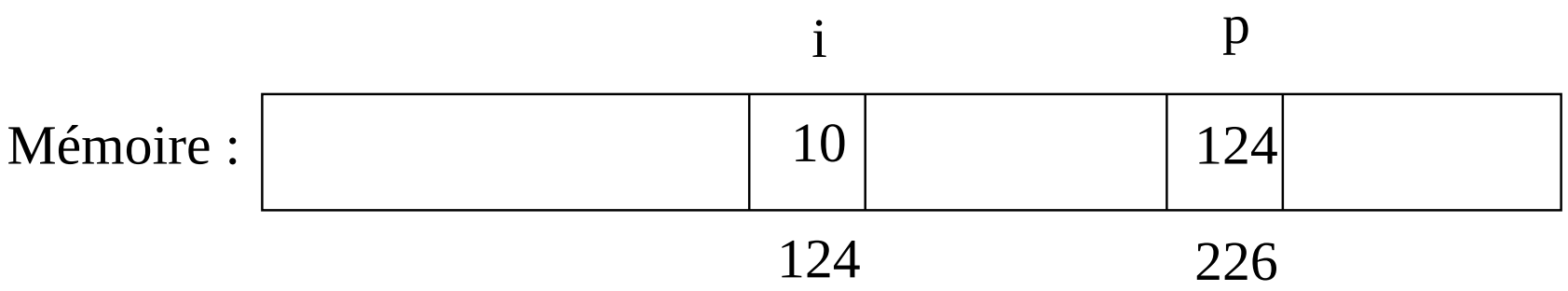
  - Par exemple :

```
i=0;
do {
   printf("%d\n", i);
   i++;
} while (i<=10);
```

  - Attention à la condition qui est la négagtion de la condition d'arrêt de l'instruction algorithmique `répéter...jusqu'à ce que`

Les pointeurs...

- Lorsque l'on déclare une variable, par exemple un entier `i`, l'ordinateur réserve un espace mémoire pour y stocker les valeurs de `i`
- L'emplacement de cet espace dans la mémoire est nommé adresse
  - Dans l'analogie de l'armoire que nous avons vu dans un cours précédent, l'adresse correspond au numéro du tiroir
- Un pointeur est une variable qui permet de stocker une adresse
- Par exemple si on declare une variable entière `i` (initialisée à 10) et que l'on déclare un pointeur `p` dans lequel on range l'adresse de i (on dit que `p` pointe sur `i`), on a par exemple le schéma suivant :



Les pointeurs...

- On déclare une variable de type pointeur en suffi xant le type de la variable pointée par le caractère *, par exemple

```
int i, j;
int* p;
```

- Il existe deux opérateurs permettant d'utiliser les pointeurs :
  - & : permet d'obtenir l'adresse d'une variable, permettant donc à un pointeur de pointer sur une variable, par exemple

```
p=&i;   // p pointe sur i
```

  - * : permet de déréférencer un pointeur, c'est-à-dire d'accéder à la valeur de la variable pointée, par exemple

```
*p=*p+2; // ajoute 2 a i
j=*p; // met la valeur de i dans j (donc 12)
```

Les pointeurs...

- Par défaut lorsque l'on déclare un pointeur, on ne sait pas sur quoi il pointe
- Comme toute variable, il faut l'initialiser
  - On peut dire qu'un pointeur ne pointe sur rien en lui affectant la valeur NULL
  - Par exemple :

```
int i;
int* p1,p2;
p1=&i;
p2=NULL;
```

Traduire les fonctions...

- On déclare une fonction en respectant la syntaxe suivante :

```
typeRetour nomFonction(type1 param1, type2 param2, ...) {
  // variables locales

  // instructions avec au moins
  // une fois l'instruction return
}
```

- Par exemple :

```
int plusUn(int a){
  return a+1;
}
```

Traduire les procédures...

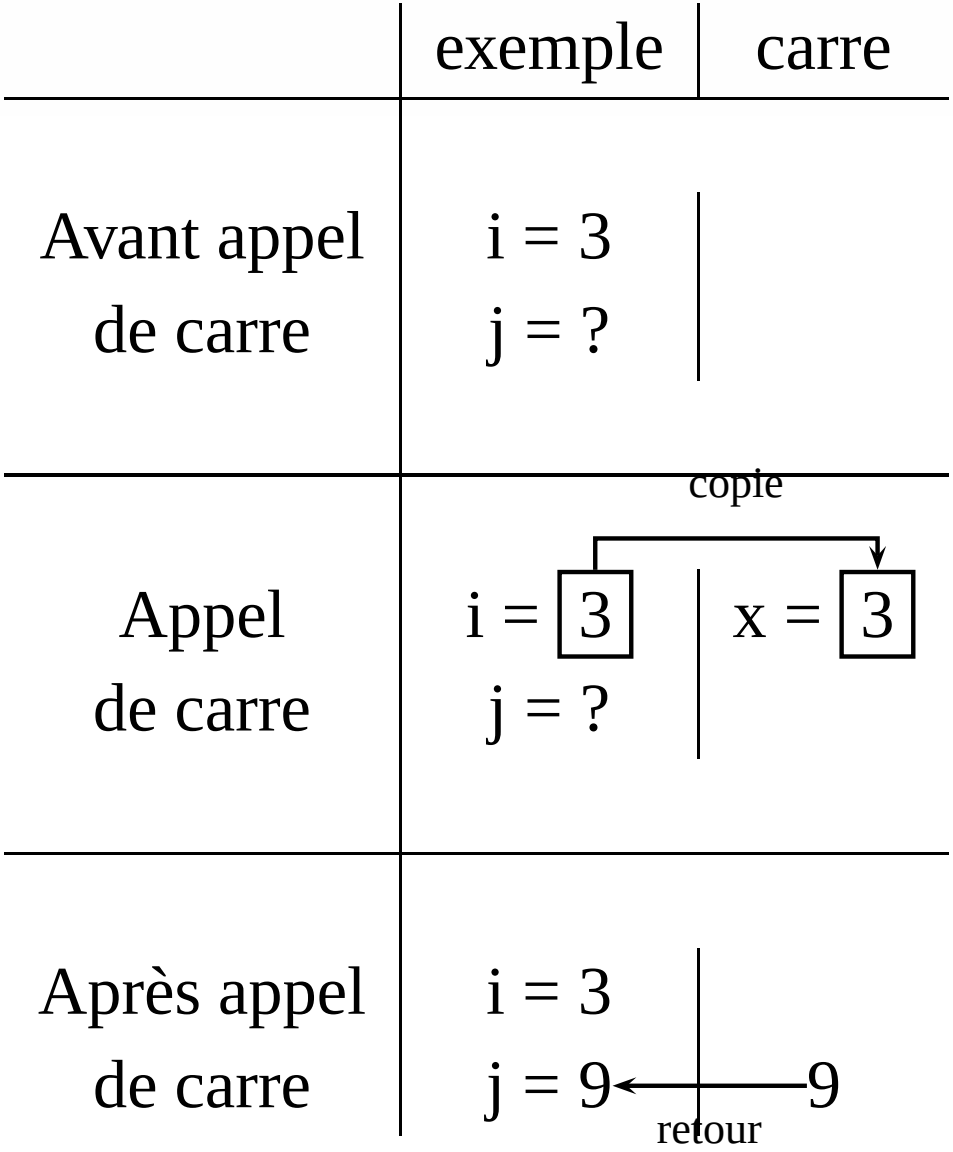
- Il n’y a pas de procédure en C
  - Pour traduire une procédure, on crée une fonction qui ne retourne pas de valeur, on utilise alors le type `void` comme type de retour
- Tous les passages de paramètres en C sont des passages de paramètres en entrée (on parle de passage par valeur)
  - Lorsque l’on veut traduire des passages de paramètres en sortie ou en entrée/sortie on utilise les pointeurs (on parle de passage de paramètre par adresse)
    - On passe le pointeur sur la variable en lieu et place de la variable

Passage de paramètre par valeur...

```
int carre (int x){
   return (x*x);
}

void exemple (){
   int i = 3;
   int j;

   j = carre (i);
   printf("%d",i2);
}
```



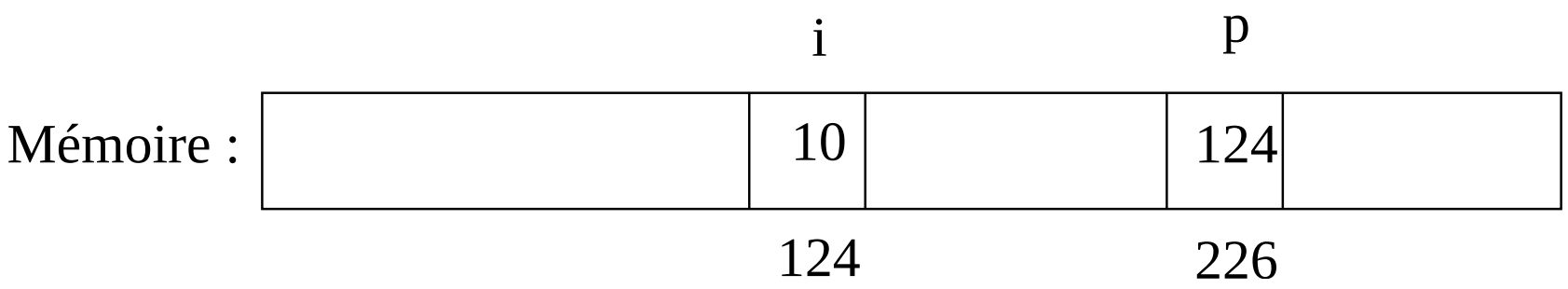
Passage de paramètre par adresse...

```
void remiseAZero(int* p) {
   *p=0;
}

void exemple2() {
   int i=10;

   remiseAZero(&i);
   printf("%d",i);
}
```

	exemple	carre
Avant appel	i = 3	
Appel de remiseAZero	i = 3 &i	p = 124
	copie	
Après appel de remiseAZero	i = 0	



Traduire `écrire`...

- L'instruction `printf` (du module `stdio.h`) permet d'afficher des informations à l'écran
- Syntaxe :

```
printf("chaîne de caractères"[,variables])
```

  - Si des variables suivent la chaîne de caractères, cette dernière doit spécifier comment présenter ces variables :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (float, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères
  - La chaîne de caractères peut contenir des caractères spéciaux :
    - \n pour le retour chariot
    - \t pour les tabulations

Traduire `écrire`...

- Par exemple :

```
int i=1;
float x=2.0;
printf("Bonjour\n");
printf("i = %d\n",i);
printf("i = %d, x = %f\n",i,x);
```

- ...affiche :

```
Bonjour
i = 1
i = 1, x=2.0
```

Traduire `lire`...

- L'instruction `scanf` (du module `stdio.h`) permet à l'utilisateur de saisir des informations au clavier
- Syntaxe :

```
scanf("chaîne de formatage",pointeur var1, ...)
```

  - La chaîne de formatage spécifie le type des données attendues, par exemple :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (fbat, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères

Traduire `lire`...

- Par exemple :

```
int i;
float x;
scanf("%d%f",&i,&x);
```

Traduire les programmes...

- Le programme principal en C est une fonction (comme une autre) dont le nom est `main` et son type de retour est `int`

Programme *nom du programme*
Définition des constantes
Définition des types
Déclaration des variables principales
Définition des sous-programmes
début
instructions du programme principal
fin

⇒

```
// inclure les librairies (#include)
// definir les constantes (#define)
// definir les types
// definir les fonctions
int main () {
  // declarer les variables du
  //   programme principal
  // instructions
}
```

Un exemple...

- Nous allons traduire le programme suivant :

Programme NombrePremier
    Déclaration n : **Entier**; c : **Caractère**
    fonction estPremier **(**a : **Entier) : Booléen**
début
    répéter
        écrire(Entrez un entier :)
        lire(n)
        si estPremier(n) **alors**
            écrire(n," est premier")
        sinon
            écrire(n," n'est pas premier")
        finsi
        répéter
            écrire("Voulez vous tester un autre nombre (O/N) :")
            lire(() c);
        jusqu'à ce que c='O' ou c='o' ou c='n' ou c='N'
    jusqu'à ce que c='n' ou c='N'
fin

Un exemple...

- avec :

fonction estPremier **(**a : **Entier) : Booléen**

Déclaration i : **Entier**

début

i ← 2

tant que i<a div 2 et a mod i≠0 **faire**

i ← i+1

fintantque

si a mod i=0 **alors**

retourner Faux

sinon

retourner Vrai

finsi

fin

Un exemple...

- Le programme C correspondant :

```
#include<stdio.h>

int estPremier(int a) {
  int i=2;
  while ((i<(a/2)) && ((a%i)!=0))
    i++;
  if ((a%i)==0)
    return 0;
  else
    return 1;
}

int main() {
  int n;
  char c;
  do {
    printf("Entrez un nombre entier");
    scanf("%d",&n);
    if (estPremier(n))
      printf("%d est premier\n",n);
    else
```

Un exemple...

```
      printf("%d n'est pas premier\n",n);
    do {
      printf("Voulez vous tester un autre nombre (O/N) :");
      scanf("%c",&c);
    }while (c!='o' && c!='O' && c!='n' && c!='N');
  }while(c=='o' || c=='O');
}
```