Cours d'Analyse, Algorithmique
Elements de programmation

Florent Hivert

Mél : Florent.Hivert@lri.fr
Adresse universelle : http://www.lri.fr/~hivert

Données et instructions

- Un programme est composé d'**instructions** qui travaillent sur des **données**.
- En programmation impérative, les données sont stockées dans des **variables**.
- Il faut souvent **déclarer** les variables.

```
#include<stdio.h>
double a,b;                    // Déclarations
void main(void) {
  a=1; b=1;
  while ((((a+1)-a)-1)==0) a*=2;
  while ((((a+b)-a)-b)!=0) b++;
  printf ("a=%f, b=%f\n", a, b);
}
```

La notion de variable

- but : stocker des informations en mémoire centrale durant l'exécution d'un programme ;
- on veut éviter d'avoir à manipuler directement les adresses ; on manipule des **variables** ;
- le programmeur donne aux variables des noms de son choix (**identificateur**) ;

Définition (**Notion de Variable**)

Une variable est un espace de stockage où le programme peut mémoriser une donnée.

Les variables désignent une ou plusieurs cases mémoires contenant une suite de 0 et de 1.

La notion de variable (2)

Retenir

Une variable possède quatre propriétés :

- *un nom ;*
- *une adresse ;*
- *un type ;*
- *une valeur.*

Notion de type

Les variables peuvent contenir toutes sortes de données différentes (nombres, texte, relevé de note, etc.)

Définition (Notion de type de données)

Dans de nombreux langage, (C/C++, Java, Pascal) une variable ne peut contenir qu'une seule sorte de données. On appelle cette sorte le ***type*** *de la variable.*

Opération sur les variables

Retenir

Il y a essentiellement deux opérations possibles sur une variable :

- *l'**évaluation** (pas de syntaxe particulière) : la variable est remplacée par sa valeur dans le programme en cours ;*
- *l'**affectation** (*`a=2`, `a:=2`, `a`$\leftarrow 2$ *se lit* `a` *reçoit* 2*) : la valeur de la variable est remplacé par celle du membre de droite. L'ancienne valeur est perdue.*

- l'écriture (`print`, `write`) contient une évaluation ;
- la lecture (`read`, `scan`) contient une affectation ;
- contrainte de type (on ne peut affecter à une variable qu'une valeur ayant le même type).

Opération sur les variables

Retenir

Il y a essentiellement deux opérations possibles sur une variable :

- *l'**évaluation** (pas de syntaxe particulière) : la variable est remplacée par sa valeur dans le programme en cours ;*
- *l'**affectation** (*`a=2`, `a:=2`, `a`$\leftarrow 2$ *se lit* `a` *reçoit* 2*) : la valeur de la variable est remplacé par celle du membre de droite. L'ancienne valeur est perdue.*

- l'écriture (`print, write`) contient une évaluation ;
- la lecture (`read, scan`) contient une affectation ;
- contrainte de type (on ne peut affecter à une variable qu'une valeur ayant le même type).

Opération sur les variables

Retenir

Il y a essentiellement deux opérations possibles sur une variable :

- *l'**évaluation** (pas de syntaxe particulière) : la variable est remplacée par sa valeur dans le programme en cours ;*
- *l'**affectation** (*`a=2`, `a:=2`, `a`$\leftarrow 2$ *se lit* `a` *reçoit* `2`*) : la valeur de la variable est remplacé par celle du membre de droite. L'ancienne valeur est perdue.*

- l'écriture (`print, write`) contient une évaluation ;
- la lecture (`read, scan`) contient une affectation ;
- contrainte de type (on ne peut affecter à une variable qu'une valeur ayant le même type).

Opération sur les variables

Retenir

Il y a essentiellement deux opérations possibles sur une variable :

- *l'**évaluation** (pas de syntaxe particulière) : la variable est remplacée par sa valeur dans le programme en cours ;*
- *l'**affectation** (*`a=2`, `a:=2`, `a`$\leftarrow 2$ *se lit* `a` *reçoit* 2*) : la valeur de la variable est remplacé par celle du membre de droite. L'ancienne valeur est perdue.*

- l'écriture (`print, write`) contient une évaluation ;
- la lecture (`read, scan`) contient une affectation ;
- contrainte de type (on ne peut affecter à une variable qu'une valeur ayant le même type).

L'affectation

Retenir

Donne une nouvelle valeur à une variable.

- *Syntaxe : nomDeVariable ← expression*
- *Sémantique : calcul (ou évaluation) de la valeur de l'expression et rangement de cette valeur dans la case mémoire associée à cette variable.*

La manipulation des variables (2)

Retenir

On peut modifier la valeur des variables tout au long du programme.

Exemples : x : 1 y : 3

opération	instruction	valeur
affecter la valeur x + 1 à la variable x	x ← x + 1	x : 2
affecter la valeur y + x à la variable y	y ← y + x	y : 5

Exemples d'affectations

Retenir (ATTENTION ! ! !)

Une fois définies, les variables possèdent une valeur même si elles n'ont fait l'objet d'aucune instruction d'affectation.

Il faut initialiser les variables !

```
#include<stdio.h>
void main(void)
{
  int a, b, c, d;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  a = 1;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  b = 3;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  c = a + b;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  d = a - b;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  a = a + 2 * b;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  b = c + b;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
  c = a * b;
  printf("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
}
```

Structures de contrôle

En programmation impérative, les instructions sont exécutées de manière séquentielle (les une après les autres), dans l'ordre du programme. On a souvent besoin de rompre la séquence d'exécution :

- Dans certain cas particulier, on veut sauter certaines instructions : **Instructions conditionnelles** ;
- Dans d'autres cas, on a besoin de répéter certaines instructions : **Instructions itératives**.
- On peut également vouloir faire appel à un **sous-programme** écrit par ailleurs.
- On gère aussi parfois les cas particulier par un mécanisme **d'exception**.

Instructions conditionnelles

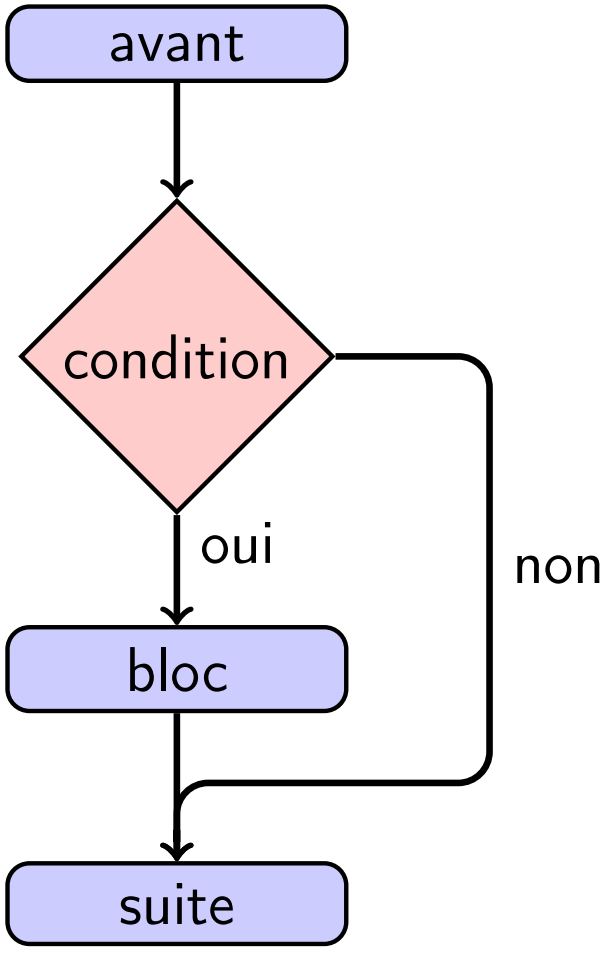
Retenir (Instruction conditionnelle)

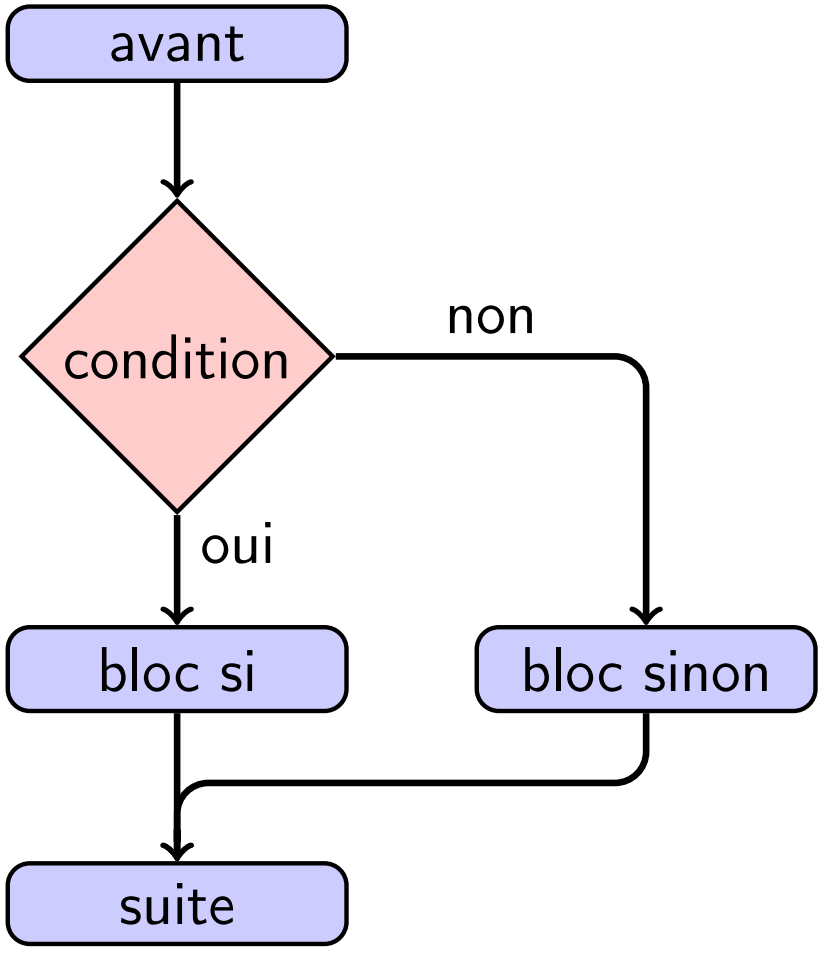
...
si condition alors
    bloc d'instructions
...

...
si condition alors
    bloc d'instructions
sinon
    bloc d'instructions
...

Selon les langages, les blocs d'instructions sont délimités différemment : `{ ... }`, `begin ... end`, indentation, *etc*

Exécution des instructions conditionnelles





Instructions conditionnelles imbriquées, arbre de choix

Pour s'assurer que différentes conditions

- sont mutuellement exclusives ;
- couvre la totalité des cas ;

il est souvent plus facile d'imbriquer les conditions :

```
si n == 0 alors afficher "n est nul"
si n > 0  alors afficher "n est positif"
si n <= 0 alors afficher "n est négatif" # Erreur

si n == 0 alors      afficher "n est nul"
sinon si n > 0 alors afficher "n est positif"
      sinon          afficher "n est négatif"
```

Instructions itératives

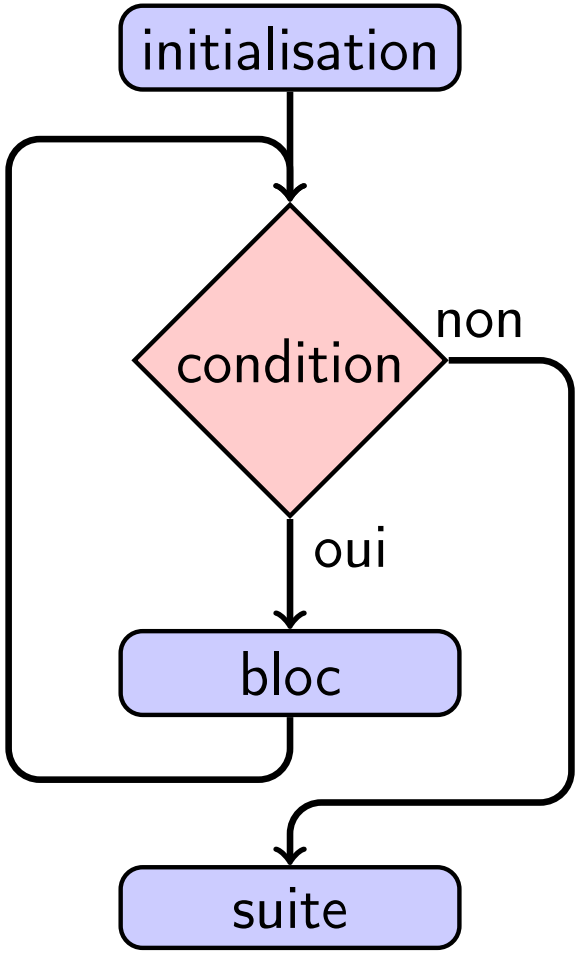
L'instruction itérative permet de répéter un certain nombre de fois l'exécution d'une suite d'instructions sous une certaine condition. De façon imagée, on appelle **boucle** cette méthode permettant de répéter l'exécution d'un groupe d'instructions.

Syntaxe

initialisation
tant que condition *faire*
    bloc d'instructions
suite

Exécution des instructions itératives

initialisation
`tant que` condition `faire`
    bloc d'instructions
suite

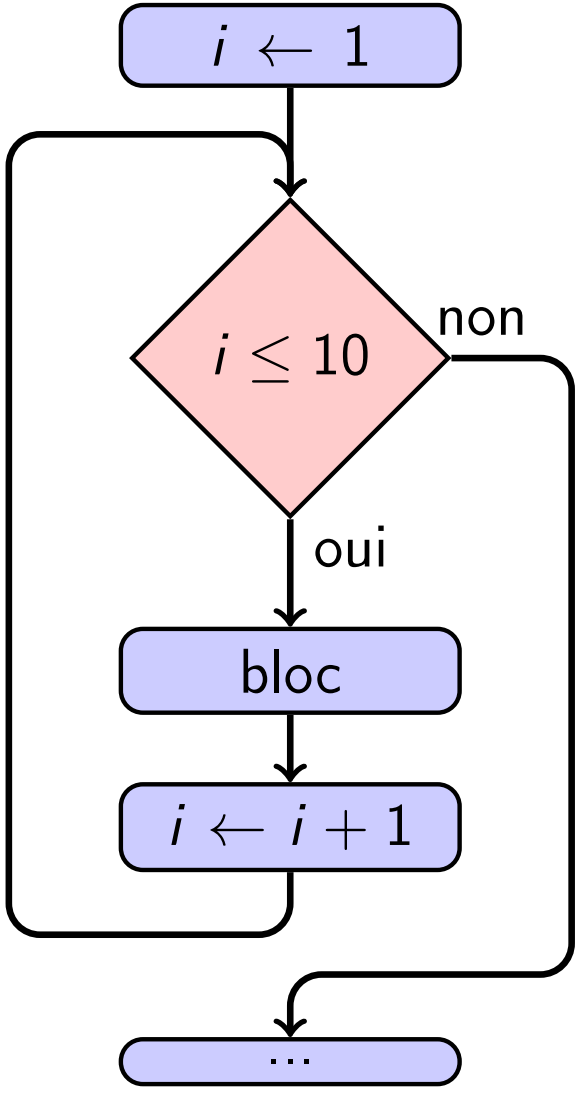


Comptage (boucle pour)

```
i ← 1
tant que i ≤ 10 faire
    bloc d'instructions
    i ← i + 1
...
```

équivalent à

```
pour i de 1 à 10 faire
    bloc d'instructions
...
```



Fin de boucle

Retenir

Attention ! À la fin de la boucle la condition est fausse !

```
i ← 1
tant que i ≤ 10 faire
    bloc d'instructions
    i ← i + 1
** Ici on a i = 11 **
```

Notion de sous programme

Quand on programme on a souvent besoin d'écrire plusieurs fois la même séquence d'instructions. . .

Le Copier-Coller est une très mauvaise méthode : on recopie les erreurs.

Les fonctions

Retenir

- *Notion de sous-programme qui peut être appelé de n'importe quel point du programme.*
- *But 1 : décomposer un problème complexe en sous problème plus simple.*
- *But 2 : partager (factoriser) du code, pour le réutiliser.*

Fonctions prédéfinies

Il existe des bibliothèques de fonctions prédéfinies appelables dans n'importe quel programme (`stdlib.h`, `math.h`).

Exemples de fonctions prédéfinies :

- `int abs(int j)` retourne la valeur absolue de l'entier `j`
- `double fabs(double x)` retourne la valeur absolue du flottant `x`
- `long int lrint(double x)` arrondis `x`
- `double sqrt(double x)` retourne la racine carrée de `x`

Écriture de nouvelles fonctions

Retenir

- *une fonction doit être déclarée.*
- *déclaration dans la partie déclarative du programme, en général après la déclaration des variables ;*
- *contient tous les renseignements nécessaires à la connaissance de la fonction.*

Écriture de nouvelles fonctions (2)

syntaxe proche de celle du programme principal :

Syntaxe

```
type_du_résultat nomDeLaFonction(
        liste_des_paramètres_formels)
{
    déclaration des variables locales
    instructions
}
```

Liste de paramètres formels

- Pour contrôler les incohérences, le compilateur doit connaître le type des valeurs données en entrée d'une fonction.
- On associe ces valeurs à des identificateurs locaux à la fonction.

Définition

- ***paramètres formel*** *: identificateur local qui reçoit une valeur donnée à la fonction ;*
- ***paramètres réel*** *: valeur donnée à la fonction.*
- *Dans la déclaration, les paramètres formels, munis de leur type, sont séparés par des « , ».*

Appel d'une fonction

Retenir

- *Calcul du résultat d'une fonction pour certaines valeurs,*
- *le **programme appelant** doit appeler la fonction en lui transmettant ces valeurs.*
- *La syntaxe est*

  nomDeLaFonction (liste des paramètres réels)

  qui doit apparaître dans une expression.
- *on peut appeler une fonction autant de fois que l'on veut.*

Passage des paramètres

Retenir

- *La liste des paramètres réels est constituée d'une liste d'expressions séparées par des virgules.*
- *Les paramètres réels doivent correspondre en type et en nombre aux paramètres formels, sinon une erreur est détectée lors de la compilation.*
- *Il est important de respecter l'ordre des paramètres formels.*
- *La valeur de chaque expression est calculée et devient ainsi la valeur du paramètre formel correspondant.*

Le résultat

Retenir

- *La fonction doit contenir une instruction de la forme* `return` *expression* ;
- *Cette instruction permet de fournir le résultat de la fonction au programme appelant.*

En général c'est la dernière instruction de la fonction.

La partie déclarations

- Si le calcul du résultat de la fonction est complexe, il peut être utile de définir des objets locaux comme des variables intermédiaires, des constantes, ou même d'autres fonctions plus simples.

- La définition de la fonction comprend donc une partie déclarative, exactement comme le programme principal.

Variables locale à la fonction

Retenir

- *Une telle variable a une existence qui est locale au sous-programme qui l'a déclarée.*
- *La variable n'existe que le temps d'exécution de la fonction.*
- *La valeur qu'elle peut avoir lors d'un appel au sous-programme est perdue lors du retour au programme appelant et ne peut être récupérée lors d'un appel ultérieur.*

Variables locales/globales

Retenir

- *les variables du programme principale (dite globale) restent accessibles à l'intérieur de la fonction.*
- *On peut modifier la valeur d'une variable globale, mais ceci est très fortement déconseillée (effet de bord).*
- *Une variable locale masque une variable globale du même nom.*

Valeur de Retour

On a parfois besoin de sous-programme qui ne retourne aucune valeur :

- on veut produire un effet (affichage, musique, *etc*)
- on veut modifier (en place) l'état interne d'une structure de donnée.

La valeur de retour est alors du type `void`.

Passage des paramètres par copie

Les paramètres formels d'une fonction sont des variables comme les autres, on peut les modifier. Mais...

Retenir

*Lors d'un appel de fonction ou de procédure, la valeur du paramètre réel est **copiée** dans le paramètre formel.*

En conséquence

- Une modification du paramètre formel, n'affecte pas le paramètre réel ;
- Si la variable est compliquée (tableaux, chaîne de charactères *etc*), cette recopie peut être coûteuse.

Notion de référence et d'adresse

Retenir

*Une **référence** à une variable existante est un moyen d'accéder au contenu de la variable aussi bien en écriture qu'en lecture.*

En pratique, on utilise l'adresse de l'emplacement où est stocké la variable en mémoire.