Programmation Modulaire
Info 3

Florent Hivert

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

Plan

Plan

Informations pratiques

- $9 \times 2 = 18$ heures de cours (en fait plutôt $6 \times 2\text{h} + 4 \times 1\text{h}30$) ;
- $12 \times 2 = 24$ heures de travaux pratiques ;
- Évaluations :
  1. 40% examen final
  2. 30% partiel
  3. 20% projet (rendu + soutenance)
  4. 10% notes TP (présence + rendu).

Plan

Rappels

Notions supposées connues :

- Notion de programme
- Notion de variable, de type, déclaration
- Instruction conditionnelle
  - `if (...) {...} else {...}`
- Instruction itérative :
  - `while (...) {...}`
  - `do {...} while (...);`
  - `for (...; ...; ...) {...}`
- Notion de fonction, appel de fonction
- Notion de tableau (les `vector` du C++)

Généralités sur le traitement de l'information

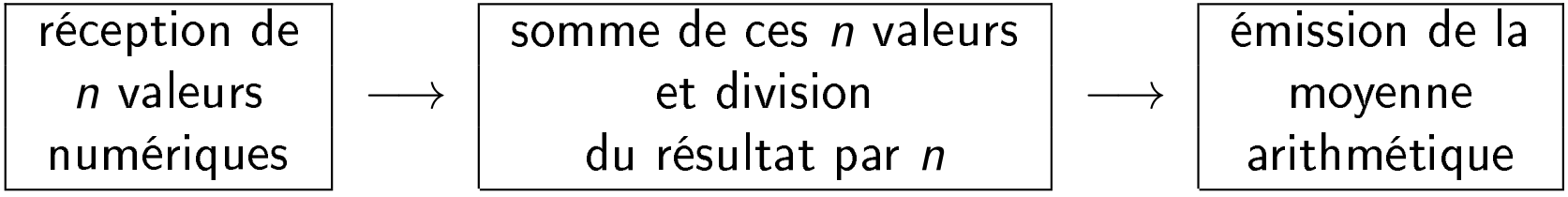
Les ordinateurs sont utilisés pour

- le traitement d'informations ;
- le stockage d'informations.

Généralités sur le traitement de l'information (2)

Le schéma global d'une application informatique est toujours le même :

Exemple :



La notion de programme

Tout traitement demandé à la machine, par l'utilisateur, est effectué par l'exécution séquencée d'opérations appelées **instructions**. Une suite d'instructions est appelée un **programme**.

Retenir

*Un programme est une **suite d'instructions** permettant à un système informatique d'exécuter une tâche donnée*

écrit dans un langage de programmation compréhensible (directement ou indirectement) par un ordinateur.

Programme Impératif

Définition (Rappel)

*un **programme impératif** est une séquence d'**instructions** qui spécifie étape par étape les opérations à effectuer pour obtenir à partir des entrées un résultat (la sortie).*

Différents paradigmes de programmation : impératif/procédural, objet, fonctionnel, déclaratif, concurrent.

Exemples de langages de programmations :
`C C++ Java Python Perl Ruby PHP Javascript Rust Go Scala Ocaml Haskell Lisp Erlang Ada Assembleur Fortran Pascal Delphi Julia Prolog`

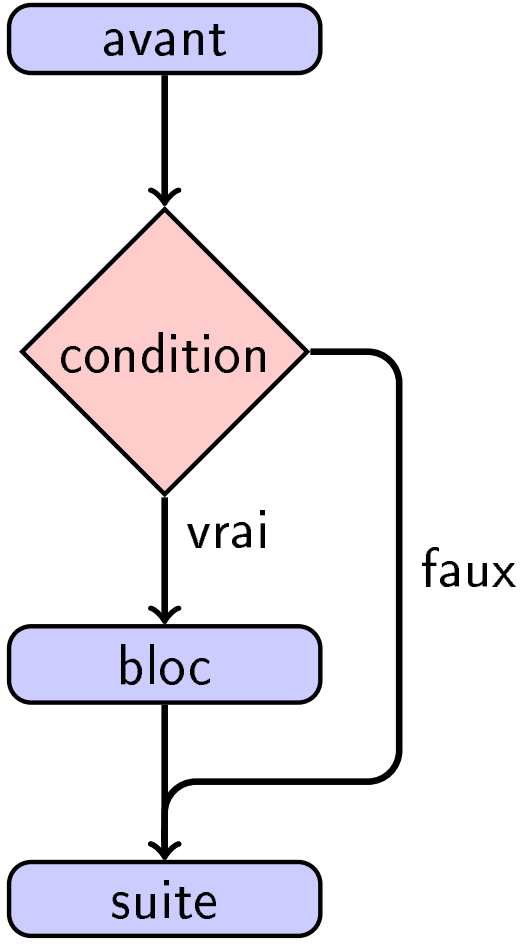
Rappel : instruction conditionnelle

```
instructions avant;

if (condition) {
    bloc d'instructions;
}

instructions suite;
```

Remarque : s'il n'y a qu'une instruction dans un bloc les accolades ne sont pas obligatoires. On choisira ce qui est le plus **lisible**.

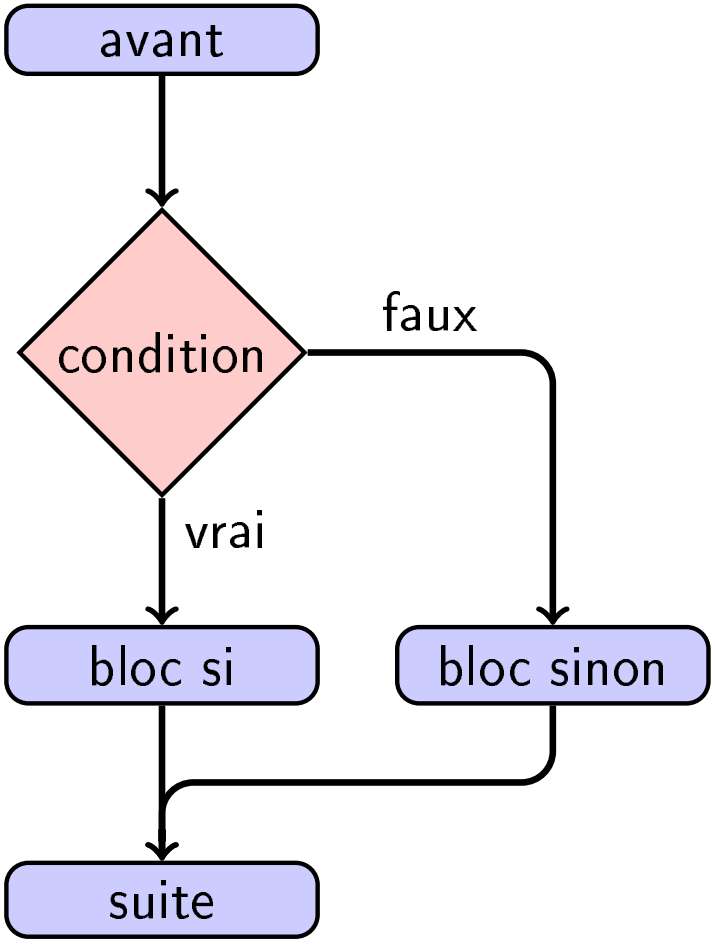


Rappel : instruction conditionnelle alternative

```
avant;

if (condition) {
    bloc si;
} else {
    bloc sinon;
}

suite;
```

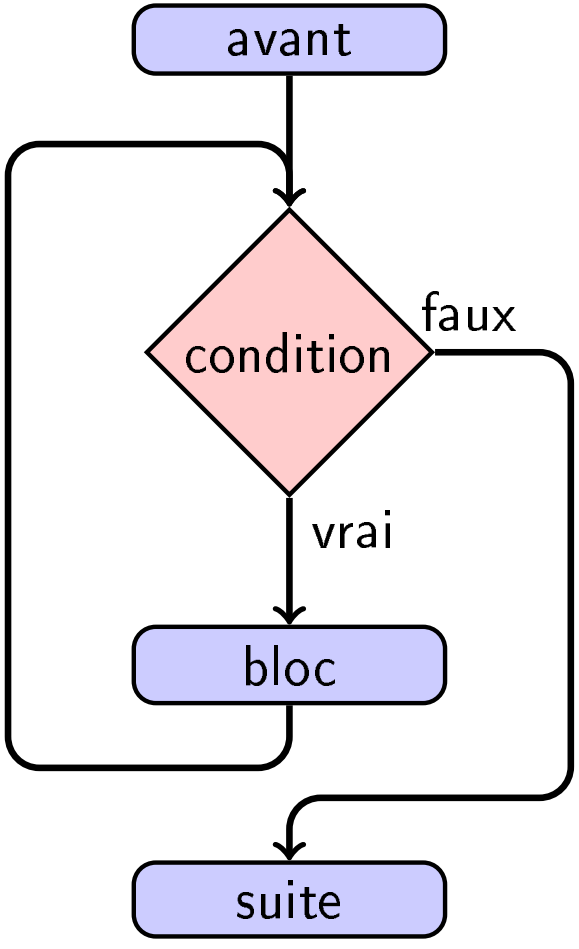


Rappel : instruction itérative (boucle `while`)

```
instructions avant;

while (condition) {
    bloc d'instructions;
}

instructions suite;
```



Remarque : Si la condition est fausse dès le début, le bloc n'est **jamais** exécuté.

Exemple de boucle while

while.cpp

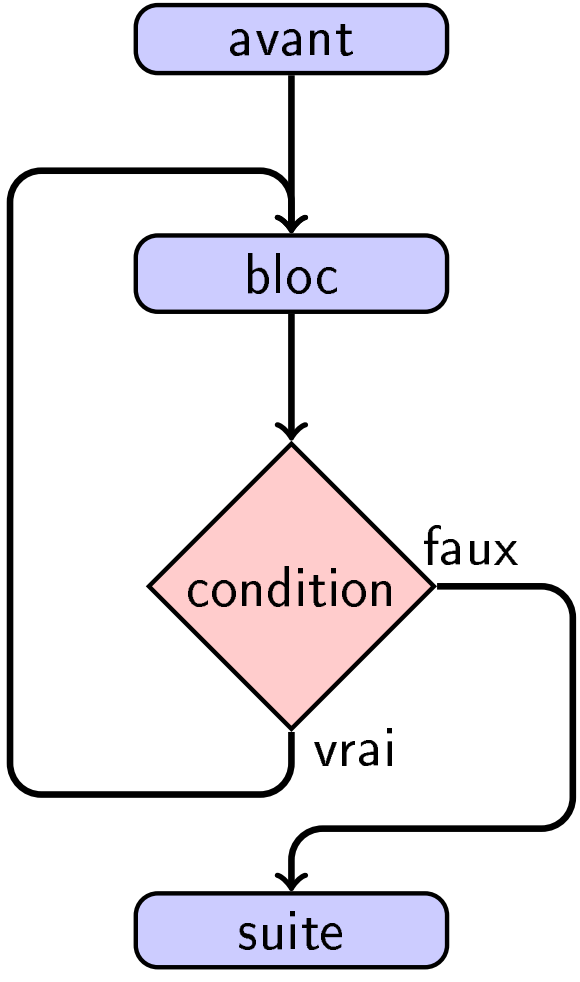
```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    cout << "Saisir un nombre positif : ";
8    cin >> n;
9    while (n < 0) {
10       cout << "Ce nombre est négatif !" << endl;
11       cout << "Saisir un nombre positif : ";
12       cin >> n;
13   }
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;  // = 0, mais plus clair.
16 }
```

Complément : instruction itérative (boucle do while)

```
instructions avant;

do  {
    bloc d'instructions;
} while (condition);

instructions suite;
```



Remarque : le bloc est **toujours** exécuté au moins une fois.

Exemple de boucle do while

dowhile.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    do {
8      cout << "Saisir un nombre positif : ";
9      cin >> n;
10     if (n < 0) {
11         cout << "Ce nombre est négatif !" << endl;
12     }
13   } while (n < 0);
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;  // = 0, mais plus clair.
16 }
```

Rappel : boucle `for`

Syntaxe

```
for (initialisation; condition; incrémentation) {
     instructions;
}
```

C'est un **raccourci plus lisible** de la boucle `while` ci-dessous :

```
initialisation;
while (condition) {
    instructions;
    incrémentation;
}
```

Rappel : exemple de fonction

Syntaxe

```
// Renvoie la somme
// de deux entiers
// @param: deux entiers
// @return: la somme
int somme(int, int);
```

```
int somme(int a, int b) {
  int res;
  res = a + b;
  return res;
}
```

```
int main() {
  int x, y, s;
  cin >> x >> y;
  s = somme (x, y);
  cout << s << endl;
  return EXIT_SUCCESS; // = 0
}
```

Rappel : notion de fonction

Retenir (Déclaration / Définition / Utilisation)

- ***Déclaration*** *: mode d'emploi, entête – nécessaire pour utiliser la fonction dans le programme ;*
- ***Définition*** *: instructions du code de la fonction – nécessaire pour pouvoir l'exécuter ;*
- ***Utilisation*** *: appel de la fonction dans une autre fonction.*

Rappel : syntaxe des fonctions

Syntaxe

Déclaration (mode d'emploi) :

```
type_retour nom(type1 <param1>, type2 <param2>, ...);
```

où `<blabla>` *signifie que le nom du paramètre est ici optionnel.*

Définition :

```
type_retour nom(type1 param1, type2 param2, ...) {
  déclaration des variables locales;
  instructions;
  ...
  return valeur_de_retour;
}
```

Appel (utilisation dans une expression) :

```
... nom(valeur_param1, valeur_param2, ...) ...
```

Instruction `return`

Retenir

Deux rôles :

- ***quitter** la fonction en cours ;*
- ***renvoyer** une valeur au programme appelant.*

- Dans une procédure (= fonction retournant `void`) seul le premier rôle est utilisé.
- De plus, si l'on ne l'a pas mis, le compilateur ajoute automatiquement une instruction «`return;`» implicite à la fin d'une fonction ou procédure.

```
1  void bonjour() {
2    cout << "Bonjour" << endl;
3    return;    // ajouté automatiquement
4  }
```

Instruction `return`

Retenir

Deux rôles :

- ***quitter** la fonction en cours ;*
- ***renvoyer** une valeur au programme appelant.*

- Dans une procédure (= fonction retournant `void`) seul le premier rôle est utilisé.
- De plus, si l'on ne l'a pas mis, le compilateur ajoute automatiquement une instruction «`return;`» implicite à la fin d'une fonction ou procédure.

```
1   void bonjour() {
2     cout << "Bonjour" << endl;
3     return;    // ajouté automatiquement
4   }
```

Données et instructions

- Un programme est composé d'**instructions** qui travaillent sur des **données**.
- En programmation impérative, les données sont stockées dans des **variables**.
- Dans de nombreux langages de programmation (C/C++/Java...), il faut **déclarer** les variables.

```
void main() {
  double a,b;                    // Déclarations
  a=1; b=1;
  while ((((a+1)-a)-1)==0) { a*=2; }
  while ((((a+b)-a)-b)!=0) { b++; }
  cout << "a=" << a << ", b=", b << endl;
}
```

Rappel : Organisation générale de l'ordinateur

Retenir

Un ordinateur est composée essentiellement de 4 éléments :

- *de la **mémoire** pour stocker les données*
- *des unités de **calcul** (logique, arithmétique, ...)*
- *une unité de **commande** qui organise le travail*
- *des unités de **communications** (clavier, écran, réseau, ...)*

Plan

Exemples de simulations

Les grandes batailles du Seigneur des Anneaux :

MASSIVE (Multiple Agent Simulation System in Virtual Environment)

Exemples de simulations : le logiciel MASSIVE

MASSIVE (Multiple Agent Simulation System in Virtual Environment)

`https://en.wikipedia.org/wiki/MASSIVE_(software)`

Quelques extraits de films créés avec Massive :
`https://www.youtube.com/watch?v=cr5Cwz-5Wsw`

Une simulation élémentaire :
`https://vimeo.com/150884144`

Exemples de simulations (2)

Plus simpliste (Battle for Wesnoth) :

Exemples de simulations (3)

Des exemples plus sérieux : simulation de circulation routière :

- `http://volkhin.com/RoadTrafficSimulator/`
- `https://www.traffic-simulation.de/`

Comment est conçu un tel logiciel ?

Modélisation

Il faut représenter, dans la mémoire de l'ordinateur

- Le monde
- Les personnages (agents) qui évoluent

Retenir

la représentation du monde et des agents est appelée

Modèle

Modélisation

Il faut représenter, dans la mémoire de l'ordinateur

- Le monde
- Les personnages (agents) qui évoluent

Retenir

la représentation du monde et des agents est appelée

Modèle

Modélisation : le monde

Pour modéliser le monde :

- repérage (coordonnées 2D / 3D), directions
- topographie, cartes
- détection des collisions, superposition (système de cases, quadtree, octree)
- objets fixes du monde

Modélisation : les agents

Pour modéliser les agents qui évoluent dans le monde :

- position, direction
- état physique (fatigue/fuel, blessure/dégâts ...)
- état mental (peur, colère ...)
- équipement, matériel
- possibilités de communication inter-agents

Organisation des données du modèle

Retenir

Pour stocker le modèle, il faut

- *une* ***représentation en machine*** *de tous les éléments (monde, objet, agents) du modèle*
- *une* ***organisation structurée, hiérarchique*** *des données*

Exemple : le nombre (entier) de munition de l'arme A, du personnage P, du joueur J...

Problème

On va avoir besoin d'**organiser les données** !

Organisation des données du modèle

Retenir

Pour stocker le modèle, il faut

- *une* ***représentation en machine*** *de tous les éléments (monde, objet, agents) du modèle*
- *une* ***organisation structurée, hiérarchique*** *des données*

Exemple : le nombre (entier) de munition de l'arme A, du personnage P, du joueur J...

Problème

On va avoir besoin d'**organiser les données** !

Évolution du modèle

Les agents

- se déplacent
- interagissent
- apparaissent, disparaissent

Retenir

*Il faut **contrôler l'évolution** du modèle :*

- *interaction avec un humain*
- *algorithmes simples*
- *Intelligence artificielle*

Évolution du modèle

Les agents

- se déplacent
- interagissent
- apparaissent, disparaissent

Retenir

*Il faut **contrôler l'évolution** du modèle :*

- *interaction avec un humain*
- *algorithmes simples*
- *Intelligence artificielle*

Visualisation

On a donc besoin de modèles et de contrôleurs pour faire évoluer le modèle.

Retenir

- *La **visualisation** n'intervient que dans un troisième temps !*
- *Dans un jeu, les joueurs n'ont pas forcement une vue totale de la situation (brouillard de guerre, information partielle).*
- *On ne visualise qu'une partie du modèle.*

Modèle / contrôle / visualisation

Retenir

*On vient de faire un début d'**analyse descendante** d'un jeu :*

- *On part du problème initial, on en extrait les composants, les sous-problèmes, les éléments, jusqu'à obtenir des composants élémentaires (nombres, tableaux)...*

*On a obtenu une ébauche d'**architecture du projet***

C'est une architecture (on parle aussi de patron de conception) d'application très classique

- Architecture Modèle/Vue/Contrôleur (MVC)
- Architecture trois tiers

Voir cours de deuxième et troisième année de **génie logiciel**...

Modèle / contrôle / visualisation

Retenir

*On vient de faire un début d'**analyse descendante** d'un jeu :*

- *On part du problème initial, on en extrait les composants, les sous-problèmes, les éléments, jusqu'à obtenir des composants élémentaires (nombres, tableaux)...*

*On a obtenu une ébauche d'**architecture du projet***

C'est une architecture (on parle aussi de patron de conception) d'application très classique

- Architecture Modèle/Vue/Contrôleur (MVC)
- Architecture trois tiers

Voir cours de deuxième et troisième année de **génie logiciel**...

Objectifs

Retenir

Dans ce cours, on va

- *se concentrer particulièrement sur le **modèle** et les **fonctionnalités du langage** qui permettent de le développer (structures, classes)*
- *voir quelques **éléments de méthodologie** pour permettre de maîtriser le développement d'un logiciel avancé*
- *apprendre quelques **outils** adaptés (infrastructure de tests, système de construction de logiciel et de gestion de version).*

Objectifs du cours :

Retenir

On va apprendre à

- ***organiser ses données*** *(tableaux, structures)*
- ***organiser ses programmes*** *(modularité, tests, encapsulation).*

- Initiation à la programmation objet
- Initiation au génie logiciel

Objectifs du cours :

Retenir

On va apprendre à

- ***organiser ses données*** *(tableaux, structures)*
- ***organiser ses programmes*** *(modularité, tests, encapsulation).*

- Initiation à la programmation objet
- Initiation au génie logiciel

Plan du cours

Alternance de chapitres sur la programmation et d'intermèdes de génie logiciel.

1. **Structures de données**
   ⇒ énumération, structures et tableaux
   - Intermède : Les tests avec `doctests`

2. **La notion d'objet**
   ⇒ attributs, classes, méthodes
   - Intermède : La fabrication avec `make`

3. **Méthodologie de la programmation modulaire**
   ⇒ Décomposition du travail, principe d'encapsulation, interface et implémentation.
   - Intermède : gestion de version avec `git`