IF 121 : Informatique Fondamentale

Organisation

Objectifs :

– Initiation à l'algorithmique et à la conception de programmes
– Apprentissage de la programmation impérative (sauf pointeurs)

Organisation :

Cours/TD : 13 semaines, TD : 1 semaine sur 2, TP : 13 semaines

Vacances :

Du lundi 28 octobre le soir au lundi 4 novembre le matin

Du samedi 21 décembre le soir au lundi 6 janvier le matin

Contrôle des connaissances :

Pa : Partiel le samedi 23 novembre à 13h
Ej : Examen en janvier
Es : Examen de septembre
CC : 4 tests de 30 minutes effectués pendant les TD, TP.

Chaque test est noté sur 5.

La présence aux tests est obligatoire

Ecrit : NE = max (Ej, (Pa+Ej) / 2)
Note finale janvier : (2NE + CC) / 3
Note finale septembre : max(Es, (2Es + CC) / 3)

Bibliographie

Algorithmique

– Concepts fondamentaux de l'informatique,
A. Aho, J. Ullman,
Dunod, 1998. ISBN 2-10-003127-9

Programmation en C++

– Programmation en C++, Théorie et problèmes,
J. Hubbard,
McGraw-Hill, Inc., Paris, Série Schaum, 1997.
– Passeport pour C++ : Concepts et mise en pratique,
G. Hansel,
Vuibert, 1999. ISBN 22-7117-8635-8
– L'essentiel du C++,
S.B. Lippman,
Addison-Wesley, 1992. ISBN 2-87908-002-9
– Le langage C++,
B. Stroustrup,
Addison-Wesley, 1992. ISBN 2-87908-013-4

Introduction

Informatique : traitement automatisé (avec des ordinateurs) de l'information.

Face à un problème à résoudre, un informaticien procède en plusieurs étapes :

- Phase de *modélisation* : il s'agit de décrire l'information à traiter de sorte qu'elle soit compréhensible par une machine.
- Phase de mise au point d'un *algorithme* : une fois que l'on dispose d'un modèle, il faut trouver une méthode, que l'on appellera *algorithme* pour résoudre le problème ainsi formulé.
- Phase de transformation de l'algorithme en *programme*, qui consiste à écrire l'algorithme de façon à ce que l'ordinateur puisse le comprendre ; on utilise pour cela un langage particulier.

Exemple

Organisation des examens de fin d'année.

- Des étudiants doivent passer un ou plusieurs examens.
- Chaque étudiant est inscrit à certains cours, mais pas forcément à tous, et doit passer les examens correspondants.
- Si un même étudiant est inscrit dans deux cours donnés, les examens qu'il devra passer pour ces cours doivent être organisés dans des créneaux horaires différents.

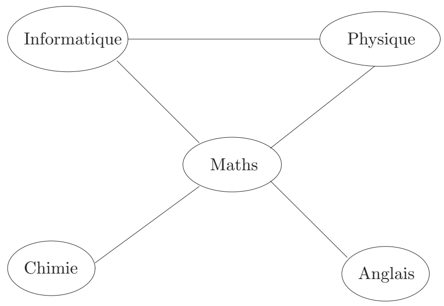
Comment arriver à écrire un programme informatique qui permette de résoudre ce problème ?

Modélisation

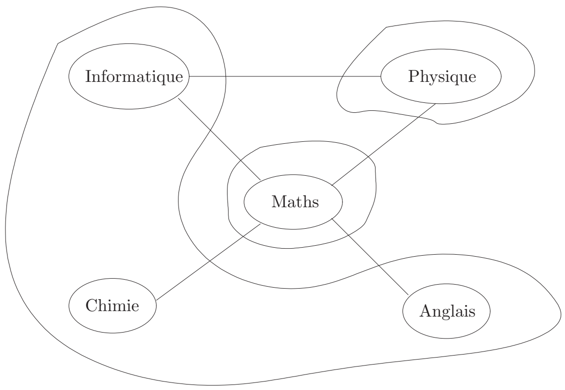
Supposons que l'on doive gérer un ensemble de 13 étudiants que l'on nommera A, B, ..., M, qui suivent les cours suivants :

- Informatique : {A, B, C, D, E, F}
- Mathématiques : {A, B, G, H, M}
- Physique : {B, C, D, F}
- Chimie : {G, H, I, J}
- Anglais : {K, L, M}

On va représenter chaque cours par un rond et on mettra un trait entre deux ronds s'il y a au moins un étudiant inscrit aux cours correspondants.



Une solution au problème initial consistera donc à allouer des créneaux horaires aux cours de sorte que des cours qui sont reliés par un trait ne soient pas dans le même créneau.



Une solution :

- Créneau horaire 1 : Informatique, chimie et anglais.
- Créneau horaire 2 : Mathématiques.
- Créneau horaire 3 : Physique.

Algorithme

Il faut trouver une méthode générale, c'est-à-dire un *algorithme* permettant de résoudre le problème de manière automatique.

Dans l'exemple : très simple... mais on peut imaginer facilement un cas beaucoup plus complexe (Paris 7 : 30 000 étudiants, 400 cours différents...).

Un algorithme pourrait par exemple être :

– Choisir un ensemble de cours, le plus grand possible, tel qu'aucun étudiant ne soit inscrit à deux ou plus de ces cours.
– Placer les examens qui correspondent à ces cours sur le premier créneau horaire disponible.
– Recommencer le même travail avec les cours restants.

D'autres algorithmes seraient bien-sûr possibles.

Programme

Il reste ensuite à transformer l'algorithme choisi en *programme*.

Pour cela, il existe de très nombreux *langages de programmation* qui peuvent être compris par un ordinateur :
– C,
– C++,
– PASCAL,
– ML,
– LISP,
– JAVA, ...

Les mots d'un langage de programmation forment ses *instructions*.

Le langage que nous utiliserons sera le C++ (C-- : version restreinte de C++ utilisée pour ce cours).

Remarques

– Pour un même problème il existe en général plusieurs modélisations.
– Pour un même modèle, il existe en général plusieurs algorithmes.
– Pour un même algorithme, il existe en général plusieurs programmes.

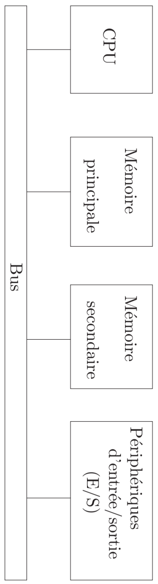
L'objectif de l'informatique est de maîtriser cette chaîne :

Problème ⟶ Modèle ⟶ Algorithme ⟶ Programme

Dans ce cours, nous nous focaliserons sur deux points :

– Comment écrit-on un algorithme pour résoudre un problème ?
– Comment écrit-on un programme à partir d'un algorithme ?

Schéma général d'un ordinateur



– Le *bus* est le lien de communication entre les éléments. Les *données* et les *instructions* transitent sur le bus.

– L'*unité centrale de traitement*, CPU (Central Processing Unit)
  – On l'appelle souvent le *processeur*.
  – Va chercher les instructions en mémoire principale et les exécute dans ses unités de calcul.
  – En une seconde, un processeur d'aujourd'hui exécute 2 milliards d'instructions.

– La *mémoire principale*
  – Stocke les programmes qui s'exécutent ainsi que les données qu'ils utilisent.
  – On la nomme aussi RAM (Random Access Memory).
  – Les données qui y sont inscrites disparaissent quand on éteint l'ordinateur (c'est une mémoire *volatile*).
– La *mémoire secondaire*
  – Formée par des disques ou des bandes magnétiques.
  – Elle est *permanente*, c'est-à-dire non volatile.
  – Elle permet en général de stocker beaucoup plus de données que la mémoire principale (facteur d'échelle ≥ 100).
  – Le temps d'accès à une donnée est beaucoup plus grand (facteur d'échelle 10^6/CPU).
– Les *périphériques d'entrée-sortie*
  – Relient l'ordinateur au monde extérieur (utilisateur, autres ordinateurs, ...).
  – Par exemple : souris, clavier, écran, imprimante,...

Quelques unités

– Dans un ordinateur, les informations sont représentées par des suites de "0" et de "1", appelés **bits** (binary digits).
– 1 octet (byte en anglais) = 8 bits
– Exemple : 01010101 et 11100010 sont des octets
– Le code ASCII associe à chaque caractère un octet
– Exemple : 'A'= 01000001, 'C'= 01000011, '9'= 00111001, ' '= 00100000
– 1Ko = 2^{10} octets
– 1Mo = 2^{20} octets
– Go, To

Programmation : les différents niveaux d'abstraction.

1. Matériel : logique numérique
2. Langage machine : ajouter 2 entiers, déplacer une donnée,...
3. Langage assembleur : représentation symbolique des instructions et données
4. Langage de programmation de haut niveau : C, C++, Fortran, Java, Lisp,...
5. Générateurs d'applications

Exemple : Calcul d'un périmètre : `perimetre = 2(L + l)`

– Langage machine :

```
// R1 : longueur, R2 : largeur, R3 : perimetre
R3 = R1 << 1
R4 = R2 << 1
R3 = R3 + R4
```

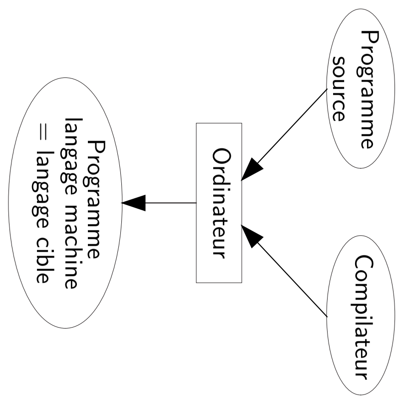
– C++ : `perimetre=2*(L + l)`

Cycle de vie du programme

Édition (dans un fichier, en utilisant un éditeur de texte)

Compilation (un compilateur est un programme spécial qui traduit du langage de programmation en langage machine

Exécution :



Exemples de programmes C++

Hello World ! : fichier hello.cpp

```
#include <iostream.h>

void main()
{//affichage de  Hello World!
    cout << "Hello World!" << endl;
}//fin programme
```

Compilation : g-- hello.cpp -o hello

Exécution : hello

```
Hello World!
```


Compilation : g-- puissance.cpp -o puissance

Exécution : puissance

```
Elévation à une puissance entière.
Donnez un nombre réel : 0.99
Donnez la puissance : 100
0.99 puissance 100 = 0.366033
```


Exemple 2 : Puissance entière

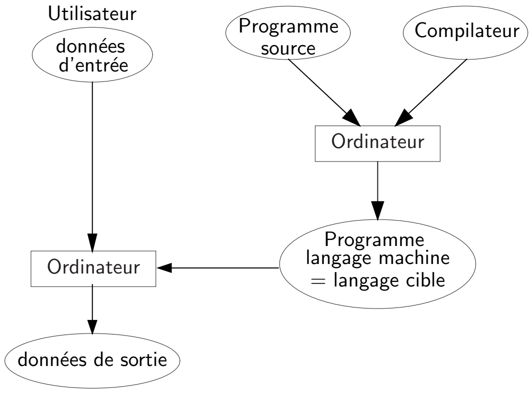
Puissance entière : fichier puissance.cpp

```
#include <iostream.h>

//calcul de la puissance entière d'un réel
float puissance(float r, int n) {//debut puissance
  int i;
  float p;
  for (i=1; i<=n; i++) p = p * r;
  return p;
} // fin puissance

void main()
{
  float a;
  int n;

  cout << "Elévation à une puissance entière.\n";
  cout << "Donnez un nombre réel : "; cin >> a;
  cout << "Donnez la puissance : "; cin >> n;

  if (n<0)
    cout << "La puissance doit être positive.\n";
  else
    cout << a << " puissance " << n << " = "
         << puissance(a,n) << endl;
}
```


Exemple 3 : Triangle de Sierpinski

Triangle de Sierpinski : fichier sierpinski.cpp

```
#include <iostream.h>
#include <math.h>
#include "MacLib.h"

void sierpinski(int a, int x, int y)
// Dessine un triangle de Sierpinski de cote "a"
// au point (x,y).
{
    const int coteMin=4;
// en dessous de cette valeur, on dessine un
// triangle plein.
    int i;
    if (a < coteMin)
        for (i=0;i<=a;i++) Drawline(x,y+i,x+i,y+i);
    else {
        sierpinski(a/2,x,y);
        sierpinski(a/2,x,y+a/2);
        sierpinski(a/2,x+a/2,y+a/2);
    }
}

void main()
{
    int cote;

    Rect windowBounds = { 40, 420, 440, 820 };
// top,left, bottom, right
    Rect myDrawingRect = { 0, 0, 400, 400 };
// top,left, bottom, right
    InitQuickDraw(); SetDrawingRect(&windowBounds);

    cout << "Triangle de Sierpinski" << endl;
    cout << "Donner la longueur du coté : ";
    cin >> cote;

    ShowDrawing(); EraseRect(&myDrawingRect);

    sierpinski(cote,10,10);
}
```



Un (pseudo-)langage pour les algorithmes

Programme en C-- : parfois difficilement lisible
⇒ On commence toujours par écrire un algorithme avant d'écrire un programme.
⇒ Il nous faut un pseudo-langage pour cela.

Pseudo-langage : Pourquoi ?

- pas de contraintes syntaxiques
- niveau d'abstraction adapté aux problèmes et aux connaissances
- Exemple :
  - lire les données
  - trier les données
  - afficher les données triées

Exercices

1. Ecrire un algorithme qui calcule le maximum de deux nombres.
2. Ecrire un algorithme qui calcule le maximum de trois nombres.
3. Ecrire un algorithme qui lit une température en degré Celsius et affiche son équivalent en degré Fahrenheit.
   La formule est degré Fahrenheit=degré Celsius*1.8+32.

Identificateurs

Pour désigner ou donner un nom à quelque chose dans un algorithme, on utilise un *identificateur*.

- Identificateur = une **lettre** suivie d'une suite de lettres ou de chiffres (ou d'autres caractères).
- Exemples :
  `x, y, m1, ABC, Nombre_Max, ...`
- Contre-exemples :
  `1m, Nombre-Max, ...`

Variables et constantes

– Une *variable* est désignée par un identificateur. Son contenu est pris dans un domaine de valeur appelé *type*.
Exemple :
si x est une variable de type entier, elle peut contenir la valeur 3 mais pas la valeur 3,14.

– Une variable peut changer de valeur au cours du temps, tout en restant du même type.

– L'opérateur d'affectation noté `<-` permet de donner une valeur à une variable.
On écrira par exemple

```
x <- 5
```

qui se lit "x prend la valeur 5" ou "x reçoit 5".

– Si la valeur d'une variable est modifiée, son ancienne valeur est perdue.

Une *constante* est désignée par un identificateur et typée. Sa valeur ne peut pas varier.

Types de base

Le *type* d'une variable est l'ensemble des valeurs que peut prendre cette variable. Il y a quatre *types de base*.

– Le type *booléen* : {vrai, faux}

– Le type *entier* : c'est un intervalle de l'ensemble des entiers relatifs, en général $[-2^{31}, 2^{31}-1]$.

– Le type *réel* : c'est un sous-ensemble de l'ensemble des réels.

– Le type *caractère* : c'est l'ensemble de tous les caractères du clavier.

Nous utiliserons également

– Le type *chaîne de caractères* : c'est l'ensemble des mots construits à partir de caractères.

Opérations élémentaires

Pour chacun de ces types est définie une liste d'*opérations* que l'on peut effectuer sur les valeurs de ce type.

Cette liste d'opérations de base peut être étendue au gré du programmeur par la notion de *fonction*.

– Type entier
  – lecture, écriture.
  – +, -, *.
  – div : division euclidienne.
  – mod : reste dans la division euclidienne.
  – comparaisons : <, <=, =, ≠, >=, >.
  – Par exemple, si A, B, C et D sont de type entier, on pourra écrire :

```
A <- B + C
A <- B div C
A <- B mod C
A <- B div (C mod D)
```


– Type réel :
  – lecture, écriture.
  – +, -, *, /.
  – racine carrée, carré, sin, cos,...
  – comparaisons : <, <=, =, ≠, >=, >.

– Type caractère : lecture, écriture, comparaisons
Rappel : les valeurs de type caractère sont codées (code ASCII).
Exemples : '0' : 48 + 0, ..., '9' : 48 + 9
'A' : 65 + 0, ..., 'Z' : 65 + 25, 'a' : 65 + 32 + 0, ..., 'z' : 65 + 32 + 25

– Type chaîne de caractères : lecture, écriture, comparaisons (=,≠), concaténation (+), longueur
Exemples : "Oui" ≠ "Non", "A" + " bientôt"

– Type booléen :

- – et, ou, non.
- – Par exemple, si A, B et C sont de type booléen, on pourra écrire :

```
A <- B et C
A <- non(A) ou (B et C)
```

Utiliser les *tables de vérité* pour évaluer les expressions booléennes.

A	B	A et B	A ou B	non(A)	non(B)	non(A et B)	non(A ou B)	non(A) ou non(B)	non(A) et non(B)
Vrai	Vrai	Vrai	Vrai	Faux	Faux	Faux	Faux	Faux	Faux
Vrai	Faux	Faux	Vrai	Faux	Vrai	Vrai	Faux	Vrai	Faux
Faux	Vrai	Faux	Vrai	Vrai	Faux	Vrai	Faux	Vrai	Faux
Faux	Faux	Faux	Faux	Vrai	Vrai	Vrai	Vrai	Vrai	Vrai

Exercice

Ecrire la table de vérité des expressions booléennes suivantes :

A ou (non(B et non(A)) et (A ou non(B)))

(A ou B) ou (non(B et non(A)) et (A ou B))

Structure d'un algorithme

1- Déclarations de variables

Pseudo :

```
variables i, j, k : entiers
          q, r : réels
          somme : entier initialisé à 0
          c : caractère
          b, encore_b : booléens initialisés à vrai
          s : chaîne de caractères
```

C++ :

```
int i, j, k;
float q, r;
int somme = 0;
char c;
bool b = true, encore_b = true;
string s;
```

2- Déclarations de constantes

Pseudo :

```
constante Max=100
```

C++ :

```
const int Max=100;
```


3- Déclaration de fonctions

Dans le pseudo-langage, un algorithme est représenté par une fonction.

Pseudo :

```
// Calcule le périmètre d'un rectangle de longueur L et largeur l
fonction périmètre(valeur L, l : entiers) résultat entier
  // déclaration de variables locales à la fonction
  variable p : entier
début périmètre
  p <- 2*(L + l)
  retourner p
fin périmètre
```

L et l sont des variables appelées *paramètres formels* de la fonction.

Toute fonction doit retourner une valeur.

Déclaration de fonction en C++ :

```
// Calcule le perimetre d'un rectangle de longueur L et largeur l
int perimetre(int L, int l)
{ // debut perimetre
    // declaration de variables locales a la fonction
    int p;

    p = 2*(L + l);

    return p;
} // fin perimetre
```


4- Mise en oeuvre d'un programme en C- -

Dans un fichier `nomProgramme.C` :

```
<déclarations de constantes>

<déclarations de fonctions>

void main ()
{// nom Programme
    <déclarations de variables>
    <lire les données>
    <liste d'instructions>
    <afficher les résultats>
} // nom Programme
```


Instructions simples

1- Affectation

Pseudo : nomVariable <- expression

C- - : <nomVariable> = <expression> ;

expression = variables et/ou constantes combinées avec des opérateurs et/ou appels de fonctions :
- expression arithmétique : x+5*y+racine(x)
- expression booléenne : a et non(b ou c)

L'expression doit avoir le même type que la variable.

L'expression est calculée (évaluée) et sa valeur est rangée dans la variable.

Exemple :

x <- 1.25	x = 1.25 ;
i <- 5	i = 5 ;
j <- 3*i*i+2*i+4	j = 3*i*i+2*i+4 ;
x <- sin(Pi/5)	x = sin(Pi/5) ;
x <- (1+racine(5))/2	x = (1+sqrt(5))/2 ;
trouve <- (i<j) et (i>0)	trouve = (i<j) && (i>0) ;
P <- perimetre(i, i+5)	P = perimetre(i, i+5) ;

Attention à la priorité des opérateurs.
On utilise des parenthèses pour
- modifier la priorité,
- améliorer la lisibilité.

Opérateurs en Pseudo/C++

– Opérations sur les entiers :

Pseudo	C++
+, -, *	+, -, *
div	/
mod	%
<,<=, >,>=	<,<=, >,>=
=	==
≠	!=

– Opérations sur les réels :

Pseudo	C++
+, -, *, /	+, -, *, /
racine (carrée)	sqrt
carré	sqr
sin, cos	sin, cos
...	...
<,<=, >,>=	<,<=, >,>=
=	==
≠	!=

– Valeurs booléennes :

Pseudo	C++
vrai	true
faux	false

– Opérations sur les booléens :

Pseudo	C++
et	&&
ou	||
non	!

– Opérations sur les chaînes de caractères

Pseudo	C++
+	+
longueur(s)	s.length()

Appel de fonction

Pseudo/C++ :

nomFonction(<liste paramètres effectifs>)

– L'appel de fonction est une expression.

– La valeur de l'expression est celle retournée par la fonction.

– Les paramètres effectifs sont des expressions dont les valeurs sont affectées aux paramètres formels au moment de l'appel.

– Ils doivent être en même nombre et du même type que les paramètres formels correspondants.

Exemple :

– L, l sont les paramètres formels dans
```
fonction périmètre(valeur L, l : entiers)
                    résultat entier
```
– i, i+5 sont les paramètres effectifs dans
```
périmètre(i, i+5)
```


2- Entrées / sorties en C++

```
cin >> <nomVariable>;
cin >> <nomVariable1> >> <nomVariable2>

cout << <expression>;
cout << <expression1> << <expression2>;
```

Exemple :

```
cin >> i >> x >> k;
cout << "P(" << i << ") = " << 3*i*i+2*i+4 << endl;
```


Exercices

1. Ecrire un algorithme qui somme quatre entiers et le programme qui met en oeuvre cet algorithme.
2. Ecrire un algorithme qui calcule la moyenne des trois entiers en utilisant la fonction écrite à l'exercice précédent. Ecrire ensuite le programme C-- correspondant.
3. Ecrire un algorithme qui calcule la note de janvier à partir des notes de l'écrit et du CC.
4. Ecrire un programme qui lit une température en degré Celsius et affiche son équivalent en degré Fahrenheit.

Instructions conditionnelles

Si . . . alors . . . finsi (Pseudo)

si condition alors

    liste d'instructions

finsi

condition est une expression booléenne.

Exemple 1

```
fonction valeur_absolue(valeur y :réel) résultat réel
début valeur_absolue
   si y<0 alors y <- (-y) finsi
   retourner y
fin valeur_absolue
```


Si . . . alors . . . sinon . . . finsi (Pseudo)

si condition alors
    liste d'instructions 1
sinon
    liste d'instructions 2
finsi

Exemple 2

```
fonction maximum (valeur x,y :réels) résultat réel
   variable M : réel initialisé à 0.0
début maximum
   si x<y alors
      M<- y
   sinon
      M<- x
   finsi
   retourner M
fin maximum
```


Généralisation (Pseudo)

si condition 1 alors
    liste d'instructions 1
sinonsi condition 2 alors
    liste d'instructions 2
. . .
sinonsi condition k alors
    liste d'instructions k
sinon
    liste d'instructions k+1
finsi

Exemple 3

```
// Calcule le nombre de racines d'un trinôme
// dont les coefficients sont a, b, c
fonction nb_racines_trinôme(valeur a,b,c : réels)
                           résultat entier
  variables d : réel
            n : entier
début nb_racines_trinôme
  si a=0 alors // pas un trinôme
    n <- -1
  sinon
    d <- b*b - 4*a*c
    si d < 0 alors // pas de racines réelles
      n <- -1
    sinonsi d = 0 alors // racine double
      n <- 1
    sinon // deux racines réelles
      n <- 2
    finsi
  finsi
  retourner n
fin nb_racines_trinôme
```

Syntaxe en C++

if (<condition>) <instruction>

ou

if (<condition>) <instruction 1> else <instruction 2>

Pour commander plusieurs instructions, il faut écrire

```
if (<condition>) {
   <liste d'instructions>
} // fin if
```

ou

```
if (<condition>) {
   <liste d'instructions 1>
} else {
   <liste d'instructions 2>
} // fin if
```

Pour éviter les ambiguïtés, il est préférable d'utiliser cette forme même lorsqu'il n'y a qu'une instruction par groupe.

Exemple 3 en C++

```
// Calcule le nombre de racines d'un trinôme
// dont les coefficients sont a, b, c
float nb_racines_trinôme(float a, float b, float c)
{ // début nb_racines_trinôme
  float d;
  int n;
  if(a == 0.0) { // pas un trinôme
    n = -1;
  } else {
    d = b*b - 4*a*c;
    if(d < 0.0) { // pas de racines réelles
      n = -1;
    } else {
      if(d == 0.0) { // racine double
        n = 1;
      } else { // deux racines réelles
        n = 2;
      } // fin if(d == 0.0)
    } // fin if(d < 0.0)
  } // fin if(a == 0.0)
  return n;
} // fin nb_racines_trinôme
```


Attention : Un else se rapporte au dernier if qui n'a pas encore de else.

```
cin >> a;
x=1;
if (a>=0)
   if (a==0) x=2;
else
    x=3;
cout << x;
```

a	-1	0	1
affichage	1	2	3

La présentation améliore la lisibilité mais ne change pas le comportement du compilateur. Pour avoir un programme qui correspond à la présentation ci-dessus, il faut écrire :

```
cin >> a;
x=1;
if (a>=0)
   {if (a==0) x=2;}
else
   { x=3;}
cout << x;
```

a	-1	0	1
affichage	3	2	1

Instruction sélection (switch)

Pseudo :

selon que expression vaut
cte$_{1,1}$,cte$_{1,2}$,... : liste d'instructions 1
cte$_{2,1}$,cte$_{2,2}$,... : liste d'instructions 2
...
cte$_{k,1}$,cte$_{k,2}$,... : liste d'instructions k
défaut : liste d'instructions k+1
fin selon

Exemple

// Jeu de dé : on gagne si la valeur obtenue est paire.

fonction dé(valeur a : entier) résultat entier
  variable p : entier
début dé
  selon que a vaut
  1, 3, 5 : p <- 0
  2, 4, 6 : p <- 1
  défaut : p <- -1
  fin selon
  retourner p
fin dé

C++

switch (<expression>) {
case <cte> : ... case <cte> : <liste d'instructions 1> break ;
case <cte> : ... case <cte> : <liste d'instructions 2> break ;
...
case <cte> : ... case <cte> : <liste d'instructions k> break ;
default :< liste d'instructions k+1>
}

Exemple

```
#include <iostream>

int de(int a) { // debut de
  int p;
  switch(a) {
  case 1: case 3: case 5: p <- 0; break;
  case 2: case 4: case 6: p <- 1; break;
  default: p <- -1;
  }
  return p;
} // fin de

void main() {
  int d;
  cout << "Entrez la valeur obtenue : ";
  cin >> d;

  d = de(d);

  if (d < 0) cout << "Erreur\n";
  else
    if (d == 0) cout << "Perdu !\n";
    else cout << "Gagne !\n";

}
```


Exercices

1. Ecrire un algorithme qui calcule le maximum de 4 nombres.
2. Ecrire un algorithme qui calcule la note de l'examen de janvier en partant des notes du partiel, de l'examen et du CC.
3. Ecrire un algorithme qui calcule le milieu de 3 entiers. Par exemple, le milieu de 3, 9, 5 est 5.
4. Ecrire un algorithme qui calcule le milieu de 5 entiers.
5. Ecrire un algorithme qui calcule le signe (c'est-à-dire, -1, 0, ou 1) d'un produit de deux nombres entiers sans calculer le produit lui-même.
   Exemple : Le signe du produit de -4 par 7 est -1.

Ecrire pour tous les algorithmes ci-dessus leur mise en oeuvre en C--.