UNIVERSITÉ PARIS DESCARTES

IAP - Introduction à l'Algorithmique et à la Programmation

T03 – Commentaires
T05 – Entrées/sorties
T11 – Expressions
T13 – Classification des opérateurs
T14 – Les tables de vérité
T15 – Expressions booléennes et arithmétiques
T16 – Règle d'évaluation des expressions
T18 – Opérateurs
T19 – Associativité des opérateurs
T20 – Priorité des opérateurs
T21 – Exemple d'évaluation d'expression
T23 – Structures de contrôle
T25 – Conditionnelle (if… else…)
T32 – Sélection à choix multiple (switch)

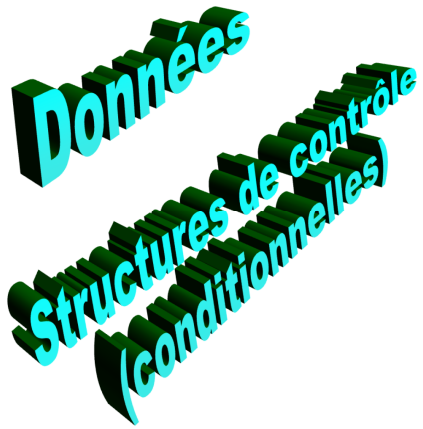
Equipe pédagogique
Marie-José Caraty, Julien Rossit, Camille Kurtz, Jacques Alès-Bianchetti, Eloi Keita

Cours n° 1

Les fondamentaux de la programmation impérative

Les éléments de langage de programmation

Sommaire



1. DOCUMENTATION

Commentaires (1/2)


```c
/* @file Cours1a-Exo01.c */
#include "stdafx.h"
int main() {
   float b=0., c=0., delta=0.;
   printf("Resolution de l'equation x^2+bx+c = 0\n");
   printf("Entrez les valeurs de b et c : ");
   scanf("%f %f", &b, &c);
   delta = (b*b)-(4*c);
   if (delta<0.) // 2 racines imaginaires
      printf("Pas de racine reelle");
   else
      if (delta==0.) // 1 racine double
         printf("Une racine reelle : %f", -b/2);
      else // 2 racines reelles
         printf("Deux racines reelles : %f et %f",
            (-b-sqrt(delta))/2., (-b+sqrt(delta))/2.);
   system("pause"); return 0;
}
```


1. DOCUMENTATION

Commentaires (2/2)

```c
/* Je suis un commentaire sur une ou plusieurs lignes,
   je précède ce que je commente (sans ligne blanche) et suis
   indenté sur la ligne que je commente (ici un main) : même
   retrait de mon premier "/" que la déclaration du main */
int main() {
   …
}


/* COM1 Je peux commenter une portion de code en utilisant
   une étiquette (par exemple COM1), je commente ici
   le problème et l'entrée des données du problème */
printf("Resolution de l'equation x^2+bx+c = 0\n");
printf("Entrez les valeurs de b et c : ");
scanf("%f %f", &b, &c);
/* FIN COM1 */


// Je suis un commentaire a priori de fin de ligne
float t; // temps d'exécution en s
// mais on m'utilise aussi en début de ligne
```

Les entrées-sorties


```
/* @file Cours1a-Exo01.c */
#include "stdafx.h"
int main() {
   float b=0., c=0., delta=0.;
   printf("Resolution de l'equation x^2+bx+c = 0\n");
   printf("Entrez les valeurs de b et c : ");
   scanf("%f %f", &b, &c);
   delta = (b*b)-(4*c);
   if (delta<0.) // 2 racines imaginaires
      printf("Pas de racine reelle");
   else
      if (delta==0.) // 1 racine double
         printf("Une racine reelle : %f", -b/2);
      else // 2 racines reelles
         printf("Deux racines reelles : %f et %f",
            (-b-sqrt(delta))/2., (-b+sqrt(delta))/2.);
   system("pause"); return 0;
}
```


Pseudo-code ecrire(v)

Sortie à partir du flot standard (écran) (1/5)

Bibliothèque `stdio.h`

Prototype `int printf( const char* format, ... );`

```
Exemple :
// Affichage des variables r, e et c (reel, entier, caractère)
// sous la forme de l'égalité des variables et de leur valeur
printf("r=%f e2=%d c=%c", r, e*e, c);
```

chaîne de format de sortie

Variables/ expression correspondant aux formats %

à chaque format % doit correspondre la variable/expression à afficher

Remarque **Le retour entier** de la fonction `printf` peut être récupéré

```
int n;
n=printf("r=%f e=%d c=%c", r, e+2, c);
```

`n` indique alors le nombre de caractères affichés

BNF de description de format (2/5)

`%[flag][prefixe][width][.precision][length]specifieur`

`flag`	cadrage à droite par défaut, pour un cadrage à gauche (`-`)
`préfixe`	+ 0 _ (_ espace)
`width`	largeur du champ
`precision`	nombre de chiffres après la virgule
`length`	court (`h`), long (`l`)
`specifieur`	caractère (`c`), chaîne (`s`)
	entier décimal (`d` ou `i`), octal (`o`), hexadécimal (`x ou X`)
	court (`hd`), long (`ld`)
	unsigned (`u`)
	réel court (`f`), notation scientifique (`e` ou `E`), général (`g`)
	long (`%lf`), notation scientifique (`le` ou `lE`), général (`lg`)

Exemple `int n=12; printf("*%-+6d*", n); // affiche :*+12__*`

Pseudo-code lire(v)

Entrée à partir du flot standard (clavier) (3/5)

Bibliothèque "`stdio.h`"

Prototype `int scanf( const char* format, ... );`

`format` est la chaine de conversion des entrées

Exemple : Forme simple de la saisie (au clavier) des 3 variables r, e et c de type resp. réel, entier et caractère

```
scanf("%f %d %c", &r, &e, &c);
```

Chaîne de format de conversion

adresses des variables « à lire »

à chaque format % doit correspondre l'adresse de la variable à saisir au clavier

Remarque Vérification des entrées par le retour entier de la fonction `scanf`

```
int n;
n=scanf("r=%f e=%d c=%c", &r, &e, &c);
```

`n` indique le nombre de variables lues (en cohérence avec les formats)

Exemple d'entrées-sorties (4/5)

```
#pragma warning(disable: 4996)
/* Cours1-Exo01.c */
#include "stdafx.h"
int main() {
    int m, n, e;
    char c;
    double r;
    printf("Entrez un caractere : ");
    m=scanf("%c", &c);
    printf("%d variable(s) lue(s), ", m);
    printf("caractere lu : %c \n", c);
    printf("Entrez un entier et un reel : ");
    m=scanf("%d %lf", &e, &r);
    n = printf("%d variables lues : e=%d, r=%lf\n", m, e, r);
    printf("%d caracteres affiches\n", n);
    printf("Test de formats (\"|\" pour encadrer l'affichage)\n");
           // \" pour afficher le caractère "
    printf("Entier cadre a droite sur 5 caracteres |n=%5d|\n", e);
    printf("Entier cadre a gauche sur 5 caracteres avec prefixe + si >0
           |n=%-+5d|\n", e);
    printf("Reel sur 10 chiffres avec 3 de precision |r=%10.3lf|\n", r);
    system("pause"); return 0;
}
```

à remplacer par les inclusions d'entête de bibliothèque

```
#include <stdio.h>
#include <stdlib.h>
```

stdio.h, pour les Entrées-Sorties
stdlib.h, pour la fonction system(…)

`system("pause");`
Commande de pause de la fonction `system()`
(spécifique à une exécution sous Visual Studio, figer la console d'exécution jusqu'à la l'entrée d'un caractère)

Exemple d'entrée-sortie (5/5)

Trace d'exécution

```
Entrez un caractere : W
1 variable(s) lue(s), caractere lu : W
Entrez un entier et un reel : 12 2.713743
2 variables lues : e=12, r=2.713743
36 caracteres affiches        Rem : le caractère "\n" est comptabilisé
Test de formats ("|" pour encadrer l'affichage)
Entier cadre a droite sur 5 caracteres |n=   12|
Entier cadre a gauche sur 5 caracteres avec prefixe + si >0 |n=+12  |
Reel sur 10 chiffres avec 3 de precision |r=     2.714|
```

Remarque A la lecture de plusieurs variables enchaînées dans un `scanf` les données saisies au clavier doivent être séparées par un « white space »

« white space »
un espace, un retour chariot ou une tabulation

Les expressions


```
/* @file Cours1a-Exo01.c */
#include "stdafx.h"
int main() {
   float b=0., c=0., delta=0.;
   printf("Resolution de l'equation x^2+bx+c=0\n");
   printf("Entrez les valeurs de b et c : ");
   scanf("%f %f", &b, &c);
   delta = (b*b)-(4*c);
   if (delta<0.) // 2 racines imaginaires
      printf("Pas de racine reelle");
   else
      if (delta==0.) // 1 racine double
         printf("Une racine reelle : %f", -b/2);
      else // 2 racines reelles
         printf("Deux racines reelles : %f et %f",
            (-b-sqrt(delta))/2., (-b+sqrt(delta))/2.);
   system("pause"); return 0;
}
```


Opérateurs et expressions

Opérateur

Symbole indiquant une opération à effectuer entre 1, 2 ou plusieurs opérandes et retournant un résultat typé dépendant de l'opérateur et du type des opérandes

Un opérateur est caractérisé par son arité (nombre d'arguments)
Lorsque l'arité est supérieure à 1, l'opération est en notation infixée (l'opérateur est situé entre les 2 opérandes)

Exemples : moins unaire : `-10`
moins binaire : `10-7`

Expression

Combinaison de littéraux (constantes de type numérique/caractère), de variables, de fonctions et d'opérateurs

L'expression exprime un calcul, une manipulation de caractères ou un test de données

Exemples `2+3*x+1`
`(a<b && a>0)||(a>b && a==0)`

Classification des opérateurs

Opérateur binaire modulo : reste de la division entière
10%2 vaut 0, 10%3 vaut 1

Arithmétiques `+, -, *, /, %` (modulo)
associés à une/des opérandes de type entier/réel/caractère
Résultat entier/réel/caractère

Relationnels `<, >, <=` (≤), `>=` (≥), `==` (=), `!=` (≠)
associés à deux opérandes de même type,
l'expression est booléenne
Résultat booléen (`faux` ou `vrai` en logique)
codé respectivement en langage C (`0` ou `1`)

Logiques `&&` (∧), `||` (∨)
∧ et ∨ logiques associés à des opérandes booléennes,
l'expression est booléenne
Résultat booléen (`faux` ou `vrai` en logique)
entier codé respectivement en langage C (`0` ou `1`)

Les tables de vérité

Opérateur	Opération
!	NON logique
&&	ET logique
||	OU logique
^	OU EXCLUSIF

x	y	! x	x && y	x || y	x ^ y
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Expressions booléennes et arithmétiques

```c
/* @file Cours1-Exo02.c */
#include <stdio.h>
#include <stdlib.h>

int main() {
   int a=7, b=3, x=5, Exp2;
   printf("Pour a=7 et b=3 :\n");
   printf("%d<%d est evalue a %d\n", a, b, a<b);
   printf("%d>%d est evalue a %d\n", a, b, a>b);

   printf("Exp1 2+3*x+1=%d\n", 2+3*x+1);
   Exp2= ((a<b && a>0)||(a>b && a==0));
   printf("Exp2 (a<b && a>0)||(a>b && a==0) vaut %d\n", Exp2);
}
```

```
Pour a=7, b=3, x=5 :
7<3 est evalue a 0
7>3 est evalue a 1
T12. 2+3*x+1=18
T12. (a<b && a>0)||(a>b && a==0) vaut 0
```

expression booléenne fausse
expression booléenne vraie
expression arithmétique évaluée à 18
expression booléenne fausse

Règle d'évaluation des expressions (1/2)

Evaluation d'une expression

`2+3*x` // est évaluée comme `(2+3)*x` ? ou comme `2+(3*x)` ?

Règle d'évaluation

(i) Fondée sur l'ordre de priorité/précédence des opérateurs
(nombre arbitraire fixant pour deux priorités distinctes l'ordre d'évaluation)
l'expression correspondant à l'opérateur de plus forte priorité
est évaluée en premier

(ii) Fondée sur l'associativité de l'opérateur : de gauche (G) à droite (D) ou de D à G
Associativité gauche : a ♥ b ♥ c = (a ♥ b) ♥ c
Associativité droite : a ♥ b ♥ c = a ♥ (b ♥ c)

(iii) Règle d'évaluation en cas d'égalité de priorité : de gauche à droite
a ♥ b ♣ c // si ♥ et ♣ sont deux opérateurs de même priorité
(a ♥ b) ♣ c est évalué // ordre d'évaluation de gauche à droite : 1) ♥ et 2) ♣

(iv) Pour un opérateur (binaire) donné, l'ordre d'évaluation est (généralement)
celui du premier opérande puis du deuxième opérande

Règle d'évaluation des expressions (2/2)

Remarque la règle d'évaluation des expressions n'est pas standard dans les langages de programmation

i) les priorités, ii) les règles d'associativité et iii) la règle d'évaluation des opérateurs binaires doivent être connues et apprises pour chaque langage

Parenthésage explicite

Le parenthésage dans une expression force l'ordre d'évaluation : les expressions entre parenthèses sont évaluées en premier

Bonne Pratique

N'utiliser que les priorités d'opérateurs les « plus usuelles »
Définir explicitement la priorité des opérations par le parenthésage approprié, même si elles sont inutiles

Opérateurs

≡ est équivalent à

Rem 1. Pré/post-incrémentation (`++i` et `i++`)
Incrémentation préfixe : `++i`
`++i ≡ i+=1 ≡ i=i+1`
si `i` vaut `5`, `++i` vaut `6` et `i` vaut `6`

Incrémentation postfixe : `i++`
`i++ ≡ tmp=i, i+=1, tmp`
, est l'opérateur d'évaluation séquentielle
`i++` vaut `tmp` (dernière expression évaluée)
si `i` vaut `5`, `i++` vaut `5` et i vaut `6`

Opérateur	Utilisation	Signification
()	f(x1, x2, ...)	appel de fonction
[]	t[i]	indexation
->	p->champ	sélection de champ de structure
.	s.champ	sélection de champ de structure
!	!a	négation logique
~	~a	complément à 1
-	-a	moins unaire
+	+a	plus unaire
*	*p	indirection
&	&x	adresse de
++	x++ ou ++x	post ou pré-incrémentation
--	x-- ou --x	post ou pré-décrémentation
(type)	(type)a	conversion explicite (*cast*)
sizeof	**sizeof**(x)	taille mémoire d'un objet
*	a*b	multiplication
/	a/b	division
%	a%b	modulo (reste de la division)
+	a+b	addition
-	a-b	soustraction
<<	a<<b	décalage gauche
>>	a>>b	décalage droit
>	a<b	inférieur
<=	a<=b	inférieur ou égal
>	a>b	supérieur
>=	a>=b	supérieur ou égal
==	a==b	égal
!=	a!=b	différent de
&	i&j	« et » bit à bit
^	i^j	« ou exclusif » bit à bit
|	i|j	« ou » bit à bit
&&	a&&b	« et » logique séquentiel
||	a||b	« ou » logique séquentiel
? :	a?b:c	expression conditionelle
= += *= ...	a=b	affectations
,	x1,x2,...,xn	évaluation séquentielle

http://users.polytech.unice.fr/~jpr/ANSI_C99/Documents/ANSI_C99-poly.pdf

Associativité des opérateurs

	Associativité
() [] -> .	gauche à droite
Opérateurs unaires ! ~ - + * & ++ -- (type) sizeof	droite à gauche
* / \%	gauche à droite
+ -	gauche à droite
<< >>	gauche à droite
< <= > >=	gauche à droite
== !=	gauche à droite
&	gauche à droite
^	gauche à droite
|	gauche à droite
&&	gauche à droite
||	gauche à droite
? : (conditionnelle)	droite à gauche
Affectations simples et composées = += *= ...	droite à gauche
, (virgule)	gauche à droite

Exemples de la règle d'associativité

L'opérateur **<** est associatif de G à D
`a < b < c` **eq.** `(a < b) < c`

L'opérateur d'affectation **=** est associatif de D à G
`a = b = c` **eq.** **`a = (b = c)`**

La valeur de c est ainsi affectée à b puis à a

Rem 3. L'affectation composée (%)
Définition de l'assignation composée
`a ♥= b;` est une contraction de `a = a ♥ b;`

♥ : opérateur arithmétique générique {+, -, *, /, %}

http://users.polytech.unice.fr/~jpr/ANSI_C99/Documents/ANSI_C99-poly.pdf

Priorité/précédence des opérateurs

		Associativité
15	() [] -> .	gauche à droite
14	*Opérateurs unaires* ! ~ - + * & ++ -- (type) sizeof	droite à gauche
13	* / \%	gauche à droite
12	+ -	gauche à droite
11	<< >>	gauche à droite
10	< <= > >=	gauche à droite
09	== !=	gauche à droite
08	&	gauche à droite
07	^	gauche à droite
06	|	gauche à droite
05	&&	gauche à droite
04	||	gauche à droite
03	? : (conditionnelle)	droite à gauche
02	*Affectations simples et composées* = += *= ...	droite à gauche
01	, (virgule)	gauche à droite

Par ordre croissant de priorité
Le sens de la flèche indique une croissance (15 est plus prioritaire que 12)

Principe d'évaluation

Dans une expression (sans parenthésage), l'opérateur de plus forte priorité est évalué en premier
En cas d'égalité de priorité, l'évaluation de l'expression se fait de gauche à droite

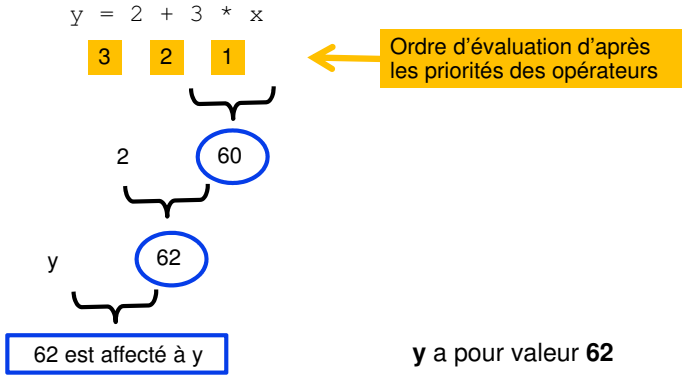
http://users.polytech.unice.fr/~jpr/ANSI_C99/Documents/ANSI_C99-poly.pdf

Exemple d'évaluation des expressions

Opérateur	Priorité
*	13
+	12
=	2

Quel est l'ordre d'évaluation de l'expression ? `y = 2 + 3 * x`

Quelle est la valeur de y pour x=20



y a pour valeur **62**

Vous savez tout, maintenant à vous !!! Préparation du TD1

Suivant les mêmes principes (cf. T. 21),
exercez-vous à évaluer les expressions suivantes :

1) (a==b && a<0) || (a<b && a>b) pour a=2 et b=10
2) (a==0) && (a<b && a>0) || (a>b && a==0) pour a=2 et b=16
3) 2 * 5 + 20 % 7 / 3 – 12
4) i+=a+=b pour i=4, a=1 et b=2

Les structures de contrôle – Conditionnelle (if… else…)

```
/* @file Cours1a-Exo01.c */
#include "stdafx.h"
int main() {
   float b=0., c=0., delta=
   printf("Resolution de l'
   printf("Entrez les valeurs de b e
   scanf("%f %f", &b, &c);
   delta = (b*b)-(4*c);
   if (delta<0.) // 2 racines imaginaires
      printf("Pas de racine reelle");
   else
      if (delta==0.) // 1 racine double
         printf("Une racine reelle : %f", -b/2);
      else // 2 racines reelles
         printf("Deux racines reelles : %f et %f",
            (-b-sqrt(delta))/2., (-b+sqrt(delta))/2.);
   system("pause"); return 0;
}
```

Rôle des structures de contrôle

Une structure de contrôle permet de modifier l'ordre séquentiel d'exécution des instructions d'un programme (flux d'exécution)

Faire exécuter des instructions
- en fonction de certaines conditions (les branchements)
- de façon répétitive (cf. les boucles, Cours 2)
- par appel de fonction (cf. les fonctions, Cours 3)

Parmi les structures de contrôle de type branchement (exprimée en pseudo-code)
- Branchement conditionnel avec alternative ou non
  si … alors … sinon … finSi
- Sélection à choix multiples
  suivant … cas … faire … finFaire

En pseudo-code: faire … finFaire
correspond à la notion de blocs d'instructions { … }

<if-cond> ::= **if (** expr **)** B_instr [**else** B_instr]

Branchement conditionnel (1/7)

Permet d'exécuter des traitements selon certaines conditions
(alternative de traitements)

if (<condition>) <Bloc_inst1> [else <Bloc_inst2>]

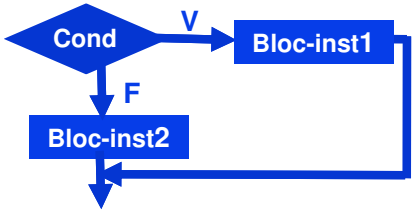
condition : expression
si condition est évaluée à vrai,
    le bloc d'instructions (Bloc_inst1) est exécuté
sinon (condition est évaluée à faux)
    le bloc d'instructions (Bloc_inst2) est exécuté

Pseudo-code
si (condition)
  alors
    Bloc_inst1
  sinon
    Bloc_inst2
fin-si

Remarque :
L'alternative `else` est optionnelle

Rappel :
Un bloc est une séquence d'instructions délimitée par { et }
Le bloc peut être vide ou restreint à une seule instruction (simple ou composée) et dans ce cas les délimiteurs peuvent être omis

Branchement conditionnel



Contrôle d'exécution

```
   inst1;
   if (condition) {
      instV1;
      instV2;
   }
   else {
      instF1;
      instF2;
      instF3;
   }
α  inst3;
```

Flux d'exécution

inst1;
si condition est évaluée vraie
    instV1; instV2;
    // continuer en séquence en α
    inst3;
sinon // condition est évaluée fausse
    instF1; instF2; instF3;
    // continuer en séquence en α
    inst3;

Bonne Pratique

Respect de la règle d'indentation
Les instructions constitutives d'un bloc doivent être **indentées** (mises en retrait par tabulation) relativement à la structure de bloc ({ … })
Objectif : mettre en valeur le flux d'exécution
(séquencement des instructions à l'exécution)

Branchement conditionnel – Exemple sans alternative (3/7)

```
/* Cours1-Exo4.c */
#pragma warning(disable: 4996)
#include <stdio.h>
#include <stdlib.h>

// Calcul de la valeur absolue d'un entier
int main() {
   int x, valAbsolue;
   printf("Entier ? ");
   scanf("%d", &x);

   valAbsolue=x;
   if (x < 0)
      valAbsolue=-x;
   printf("Valeur absolue de %d : %d\n", x, valAbsolue);

   system("pause"); return 0;
}
```

```
Entier ? -7
Valeur absolue de -7 : 7

Entier ? 3
Valeur absolue de 3 : 3
```


Branchement conditionnel – Exemple avec alternative (4/7)

```
/* Cours1-Exo5.c */
#pragma warning(disable: 4996)
#include <stdio.h>
#include <stdlib.h>

// Calcul du minimum de deux entiers
int main() {
   int min, x, y;
   printf("Valeur des deux entiers ? ");
   scanf("%d %d", &x, &y);
   if (x<y)
      min=x;
   else
      min=y;
   printf("Le minimum de %d et %d est : %d\n", x, y, min);

   system("pause"); return 0;
}
```

```
Valeur des deux entiers ? 5 -3
Le minimum de 5 et -3 est : -3

Valeur des deux entiers ? 7 9
Le minimum de 7 et 9 est : 7
```

Branchement conditionnel (5/7)

Imbrication de branchements conditionnels

```
inst1;
if (condition1) {
   inst11;
   …;
}
else if (condition2) {
        inst21;
        …;
     }
     else  if (condition3) {
              inst31;
              …;
           }
           else {
              instParDefaut;
           }
instSuivante;
```

Flux d'exécution ?
lorsqu'aucune condition n'est vérifiée

Remarque
Code illisible, sans une indentation correcte des bocs `if` et `else`

Branchement conditionnel - Exemple avec imbrication (6/7)

```
/* Cours1-Exo6.c */
#pragma warning(disable: 4996)
#include <stdio.h>
#include <stdlib.h>

// Calcul du minimum de trois entiers
int main() {
    int min, x, y, z;
    printf("Valeur des trois entiers ? ");
    scanf("%d %d %d", &x, &y, &z);

    if (x<y && x<z)
        min=x;
    else
        if (y<z)
            min=y;
        else
            min=z;
    printf("Le minimum de %d, %d et %d est : %d\n",
           x, y, z, min);

    system("pause"); return 0;
}
```

```
Valeur des trois entiers ? 57 75 92
Le minimum de 57, 75 et 92 est : 57

Valeur des trois entiers ? 34 65 20
Le minimum de 34, 65 et 20 est : 20

Valeur des trois entiers ? 5 3 7
Le minimum de 5, 3 et 7 est : 3
```


Branchement conditionnel (7/7)

Style de codage

```
if (condition)
{
  inst1;
  instr2;
}
else
{
  inst3;
}
inst4;
```

```
if (condition) {
  inst1;
  inst2;
}
else {
  inst3;
}
inst4;
```

```
if (condition) {
  inst1;
  inst2;
}
else
   inst3;
inst4;
```

Bonne Pratique

- Coder de façon homogène et respecter l'indentation (retrait des instructions)
- Une règle de codage pour la maintenance peut être de systématiquement délimiter un bloc par { et } (même dans le cas d'une instruction unique) : la structure de bloc est mise en place pour d'éventuels ajouts d'instructions

Sélection à choix multiples (1/3)

Traitements à effectuer pour certaines valeurs (discrètes) d'une expression (de type entier/caractère)

```
switch (<expression>) {
        case <valeur> : [<instructions>]
         …
        [default : [<instructions>]
}
```

expression : le discriminant de type entier ou caractère
valeur : un littéral du type de expression

Bonne Pratique

Lorsque le type de l'expression le permet,
préférer le `switch` à l'enchaînement de branchements conditionnels
dès que le niveau d'imbrication dépasse 2

Sélection à choix multiples (2/3)

Contrôle d'exécution

```
inst1;
switch (expression) {
   case V1 : instV1a;
             instV1b;
             break;
   case V2 : instV2;
   case V3:  instV3;
   case V4:
   case V5:  instV5;
             break;
   default:  instDefault;
}
inst3;
```

Flux d'exécution

`inst1;`
si `expression` vaut `V1`
  `instV1a; instV1b; inst3;`
si `expression` vaut `V2`
  `instV2; instV3; instV5; inst3;`
si `expression` vaut à `V3`
  `instV3;instV5; inst3;`
si `expression` vaut à `V4`
  `instV5; inst3;`
si `expression` vaut `V5`
  `instV5; inst3;`
si `expression` est différente de `V1`, `V2`, `V3`, `V4` et `V5`
  `instDefault; inst3;`

Attention :

Ne pas oublier d'invoquer l'nstruction `break` pour interrompre l'exécution en séquence et provoquer la sortie du bloc `switch`

Sélection à choix multiples - Exemple (3/3)

```
/* Cours1-Exo7.c */
#pragma warning(disable: 4996)
#include <stdio.h>
#include <stdlib.h>

// Oui ou non ?
int main() {
  char c;
  printf("Entrez un caractere, oui ou non (o/n) ? ");
  scanf("%c", &c);

  switch (c) {
    case 'o':
    case 'O': printf("Vous m'avez dit Oui!");
              break;
    case 'n':
    case 'N': printf("Vous m'avez dit Non!");
              break;
    default : printf("Ni oui, ni non. Peut-être ?");
  }

  system("pause"); return 0;
}
```

```
Entrez un caractere, oui ou non (o/n) ? O
Vous m'avez dit Oui!
Entrez un caractere, oui ou non (o/n) ? n
Vous m'avez dit Non!
Entrez un caractere, oui ou non (o/n) ? Y
Ni oui, ni non. Peut-etre ?
```


Bilan de ce que vous avez appris en cours

- Les commentaires
- Le principe des entrées-sorties standard
- Les expressions
  - Leur règle d'évaluation
  - La priorité des opérateurs
- Les structures de contrôle
  - Leur rôle
  - Instruction conditionnelle
  - Instruction à choix multiples

Au prochain cours…

La suite des structures de contrôle : les boucles