C/C++

Ch 2 – Caractéristiques de base

I. INTRODUCTION 2

A. STRUCTURE GENERALE 2
B. LES COMMENTAIRES EN C++ 2
C. IDENTIFIER LE PROGRAMME 3
D. LES DIRECTIVES DU PREPROCESSEUR 3
1. Directives d'inclusion : 3
2. Macros (langage C) 3
E. ENTREE DU PROGRAMME : LA FONCTION « MAIN » 4
F. DECLARATION DES DONNEES – CONSTANTES ET VARIABLES 4
G. LES INSTRUCTIONS ET BLOCS D'INSTRUCTIONS 4

II. DECLARATION DES DONNEES : VARIABLES ET CONSTANTES 4

A. TYPES DE DONNEES DE BASE 4
1. Les nombres entiers : char, short, int, long 4
2. Les nombres réels : float, double, long double 5
3. Les caractères : char 5
4. Le type logique : bool 5
5. Complément : la fonction « sizeof » 6
B. DECLARATION DES VARIABLES 6
C. CONSTANTES 6
D. ENUMERATIONS 7

III. INSTRUCTION D'AFFECTATION, OPERATEURS ET EXPRESSIONS 7

A. VALEURS LITTERALES, OU LITTERAUX 7
B. EXPRESSIONS ET OPERATEURS 7
1. Expressions arithmétiques 8
2. Expressions logiques et opérateurs de comparaison (opérateurs relationnels) 8
3. Expressions logiques et opérateurs logiques 9
C. L'AFFECTATION : OPERATEUR = ... 9
D. AFFECTATION ET OPERATEURS ARITHMETIQUES 10
1. Pré- et post- incrémentations 10
2. Affectations composées (ou étendues) 10
E. CONSEIL AU SUJET DES VARIABLES 10
F. CONSEIL AU SUJET DES EXPRESSIONS 10
G. TRANSTYPAGE, CHANGEMENT DE TYPE 11

IV. INSTRUCTIONS D'ENTREE-SORTIE : LES FLUX 12

A. AFFICHER DES INFORMATIONS : FLUX DE SORTIE 12
1. Librairie C++ <iostream> : cout 12
2. Librairie C stdio.h : puts, printf 12
B. RECUPERER DES DONNEES SAISIES : FLUX D'ENTREE 13
1. librairie C++ <iostream> : cin, cin.getline, cin.get 13
2. librairie C <stdio.h> : scanf, fgets, getchar 14
C. CONSEIL POUR LA SAISIE DES NOMBRES ENTIERS 14

V. CHAINES DE CARACTERES C ET C++ 16

A. CHAINES DE CARACTERES C : CHAR[] 16
1. Déclaration 16
2. Les fonctions de la librairie string.h 16
B. CHAINES DE CARACTERES C++ : CLASSE STRING() 16
1. Déclaration d'une instance de 'string' 17
2. Les fonctions membres de la classe 'string' 17
3. Opérateurs pour la classe 'string' 18

I. Introduction

A. Structure générale

La structure générale d'un programme est la suivante :

- Des commentaires d'entête (optionnels, mais fortement conseillés) :

```
/* But du programme : à partir de la saisie de votre âge, ce programme détermine si vous êtes majeur
     Nom : Patrick Dezécache
     Date de création : 04/09/2007
     DONE (REALISE) : date et mise à jour effectuée
     TODO (A FAIRE) : mise à jour à effectuer
*/
```

- Des directives de compilation (librairies à inclure)

```
#include <iostream>
```

- La définition des espaces de noms utilisés (C++)

```
using namespace std ;
```

- Les prototypes de fonctions

```
(absent ici)
```

- Les déclarations de classes

```
(absent ici)
```

- Les déclarations de constantes globales

```
(absent ici)
```

- La définition de la fonction principale (= traduction de l'algorithme principal)

```
int main()
{
```

- L'identification des données qui seront utilisées :

```
const int AGE_MAJORITE = 18; // une constante
int Vage ; // une variable
```

- La suite d'instructions à exécuter pour obtenir le résultat :

```
    cout << "donnez votre âge :" ;
    cin >> Vage ;
    if (Vage >= AGE_MAJORITE)
         {
             cout << "majeur" ;
          }
         else
         {
             cout << "mineur" ;
          }
    return 0 ; // 0 = tout s'est bien passé
} // fin main
```

- La définition des fonctions

```
(absent ici)
```

B. Les commentaires en C++

Les commentaires sont des explications textuelles ajoutées au programme source afin de permettre à celui qui le modifiera par la suite de mieux en comprendre le fonctionnement.

Syntaxe1 : (bloc)

```
/* . . . un commentaire
 sur plusieurs
 lignes (bloc)
 . . . */
```

Ce commentaire délimité par les caractères /* et */ peut s'étendre sur plusieurs lignes.

Syntaxe2 : (en ligne)

```
// . . . un commentaire sur une ligne . . .
instruction ; // un commentaire en bout de ligne
```

Ce commentaire s'étend jusqu'au bout de la ligne : le retour à la ligne met fin au commentaire. Il peut être placé n'importe où sur la ligne

Les commentaires situés dans l'entête incluent en général :

- le but du programme,
- Le nom de l'auteur,
- La date de création,
- Les modifications à apporter et celles réalisées.

Il est souvent intéressant d'ajouter des commentaires pour apporter des informations supplémentaires au sujet de données utilisées ou de traitements effectués.

LES COMMENTAIRES PARTICIPENT A LA QUALITE D'UN PROGRAMME (MAINTENABILITE). IL FAUT CEPENDANT QUE CES COMMENTAIRES SOIENT UTILES !

C. Identifier le programme

Le C/C++ ne prévoit pas de syntaxe pour identifier un programme dans le code source, seul le nom du fichier peut porter cette indication. A la compilation, il sera possible de préciser le nom du programme exécutable.

D. Les directives du préprocesseur

Les directives du préprocesseur sont des instructions particulières permettant, entre autre, l'inclusion de fichiers dans le programme source juste avant la compilation.

1. Directives d'inclusion :

Syntaxe :

```
#include <nom_fichier_entete>
```

à cet endroit du fichier source, sera inséré (inclus) le fichier dont le nom est balisé par < et >. Ce fichier contient généralement des prototypes[1] de procédures et fonctions de bibliothèque standard du C/C++ ou de bibliothèques d'extension (exemple SDL pour utiliser des fonctions graphiques).

2. Macros (langage C)

Syntaxe :

```
#define PI 3.1415927
#define carre(x) (x*x)
```

[1] Prototype de fonction : description simplifiée d'une fonction, incluant ses paramètres et son type de retour.

Les macros permettent de définir des valeurs constantes qui seront remplacées telles quelles (avec modification des variables) dans le programme source. Par exemple ici, toutes les occurrences de PI seront remplacées par 3.1415927, toutes celles de carre(x) par (x*x).

E. Entrée du programme : la fonction « main »

L'exécution d'un programme doit absolument débuter à un endroit précis : la fonction « main » (principal en anglais) est le point d'entrée d'un programme C/C++.
Cette fonction renvoie un code numérique (nombre entier) qui signale à l'environnement d'exécution (le système d'exploitation, par exemple) que le traitement s'est bien déroulé (cas où la valeur renvoyée est 0) ou non (cas où la valeur renvoyée est différente de 0).

F. Déclaration des données – constantes et variables

La déclaration des données peut se trouver à n'importe quel endroit dans un programme C/C++. Cependant, les données doivent être déclarées avant toute utilisation, c'est-à-dire « au dessus » des instructions qui les utilisent.

G. Les instructions et blocs d'instructions

Les instructions doivent être écrites dans le bloc principal de la fonction « main », entre l'accolade de début et l'accolade de fin du cette fonction.
Chaque phrase C/C++ doit se terminer par un point-virgule.
Des blocs d'instructions peuvent être ouverts n'importe où à l'intérieur du bloc principal, pour isoler des portions de code avec leurs données propres.

II. Déclaration des données : variables et constantes

Le C et le C++ sont des langages typés : cela signifie que chaque donnée doit être associée à un type de données précis afin que le compilateur puisse vérifier la validité des opérations effectuées sur celle-ci.

A. Types de données de base

	Nombres		Textes		Logique
Types de de base	*ENTIER*	*REEL*	*CARACTERE*	*CHAINE*	*LOGIQUE (BOOLEEN)*
En C++	**`char`** **`int`**	**`float`** **`double`**	**`char`**		**`bool`**
Contenu	Nombre sans décimales	Nombre avec décimales	1 seul caractère		2 valeurs possibles

Exemples	`0, -125, 10, 3256, 2654980`	`123.56f`
		`123f`
		`-0.56f`
		`3.1415927f`
			`'a'`
			`'b'`
			`'-'`
			`'\n'`
				`"bonjour"`
				`"bonne annee"`
					`TRUE, 1`
					`FALSE, 0`

1. Les nombres entiers : char, short, int, long

Le langage C/C++ dispose de plusieurs types d'entiers, chacun utilisant une zone mémoire de taille différente (en nombre d'octets) et permettant de stocker des plages de nombres différentes :

signed (défaut)			*Ou*		
		char	*int8*	1 octet	[-128, +127]

	short	int	*int16*	2 octets	[-32768,+32767]
		int	*int32*	4 octets	-2.147.483.648, +…
	long	int	*int32/int64*	4 octets	Idem.
unsigned					
unsigned		**char**	*uint8*	1 octet	[0, +255]
unsigned	**short**	int	*uint16*	2 octets	[0,+65535]
unsigned		**int**	*uint32*	4 octets	[0,4.294.967.295]
unsigned	**long**	int	*uint32/uint64*	4 octets	Idem.

L'inclusion du fichier « limits.h » (#include <limits.h>) donne accès à des constantes systèmes donnant les valeurs limites de chacun des types :

- CHAR_MIN, CHAR_MAX et UCHAR_MAX pour le type char
- SHRT_MIN, SHRT_MAX et USHRT_MAX pour le type short (abbrev. de short int)
- Etc.

2. Les nombres réels : float, double, long double

Le langage C/C++ dispose de 3 types de nombres réels, chacun utilisant une zone mémoire de taille différente (en nombre d'octets) et permettant de stocker des plages de nombres différentes.
Les nombres sont dits en virgule flottante (signe, mantisse et exposant) :

	float	4 octets	32 bits : 1 bit de signe, 8 bits d'exposant (-126 à 127), 23 bits de mantisse
	double	8 octets	64 bits : 1 bit de signe, 11 bits d'exposant (-1022 à 1023), 52 bits de mantisse
long	**double**	16 octets	

LES NOMBRES EN VIRGULE FLOTTANTE PERMETTENT LA REALISATION DE CALCULS SUR DES GRANDS NOMBRES, MAIS AVEC UNE PRECISION LIMITEE.

3. Les caractères : char

Le type « **char** » est stocké sur 1 octet : il permet la représentation de la codification ASCII[2] des caractères et, en cela il peut être également utilisé comme un nombre.

```
char Vcaract = 'A' ;
Vcaract++ ;
cout << Vcaract ; // affiche le caractère B à l'écran
```

Il existe un certain nombre de caractères de contrôles :

\t	tabulation	
\n	Retour à la ligne	
\0	Caractère nul	Termine les tableaux de caractères
\a	sonnerie	

4. Le type logique : bool

Le type logique « **bool** », stocké sur 1 octet, permet de définir les valeurs « **true** » et « **false** », les 2 valeurs résultant de l'évaluation des expressions logiques.

```
bool VestGrand;
VestGrand = (Vtaille > 220) ; // affectation
if (VestGrand == true) cout << "grand personnage !" ; // test
```

[2] norme de codage de caractères en informatique la plus connue, associant un caractère à la valeur de l'octet

```
if (VestGrand) cout << "grand personnage !" ; // test équivalent
```

5. Complément : la fonction « sizeof »

La fonction « **sizeof** » permet de connaître le nombre d'octets réservés par type de données de base :

```
cout << sizeof(char) ; // affiche : 1
cout << sizeof(short) ; // affiche : 2
cout << sizeof(int) ; // affiche : 4
```

L'application de la fonction sizeof à un tableau donne la longueur du tableau : pour connaître le nombre de « cases », il faut donc diviser par la taille d'une case.

B. Déclaration des variables

Syntaxes :

```
type_de_données ident_var1 ;
type_de_données ident_var2, ident_var3, . . . ;
type_de données ident_var4 = valeur_init4;
```

- **type_de_données** est l'un des types défini ci-dessus
- **ident_var1, ident_var2, ident_var3, ident_var4**
  - identifiant unique de la variable
  - composé de lettres et chiffres, des caractères – et _, le premier caractère est une lettre, PAS D'ESPACE
  - préfixé par V (dans notre convention)
- (indicateur de tableau : cf. chapitre suivant.)
- **valeur_init4** : valeur initiale de la variable

Exemples :

```
int Vage ; // entiers
float Vprime, VprimeEnFrancs, VsalaireBase ; // reels
int VvaleurDebut = 100 ; // entier initialisé
char Vlettre ; // caractère - entier
char Vprenom[20] ; // chaine, tableau de caractères
```

IL EST RECOMMANDE DE DECLARER UNE VARIABLE PAR LIGNE DE CODE SOURCE.

C. Constantes

Syntaxe :

```
const type_de_données NOM_CONSTANTE = valeur;
```

- **type_de_données** est l'un des types défini ci-dessus
- **NOM_CONSTANTE**
  - identifiant unique de la constante
  - composé de lettres et chiffres, des caractères – et _, le premier caractère est une lettre, PAS D'ESPACE
  - EN MAJUSCULE
- **valeur:**
  - valeur que prendra cette constante tout au long de l'algorithme (littéral)

Exemples :

```
const double PI = 3.14 ;
const double TAUX_EURO = 6.55957 ;
const char MOT_ACCUEIL[10]  = "Bonjour !" ;
```

```
const int NOMBRE_DEBUT = 1, NOMBRE_FIN = 10 ;
const char DEBUT_ALPHABET  = 'a' ;
```

D. Enumérations

L'énumération permet la constitution d'une liste de noms associés à des valeurs entières

Syntaxe :

```
Enum nom_enumération{ NOM = valeur, ...};
```

- **Nom_énumération**
  - Optionnel, nom donné à l'énumération
- **NOM**
  - identifiant unique de la constante
- **valeur:**
  - valeur que prendra ce nom

Exemples :

```
enum sexe {HOMME = 1, FEMME = 2} ;
```

III. Instruction d'affectation, opérateurs et expressions

A. Valeurs littérales, ou littéraux

Une **VALEUR LITTERALE**, ou un **LITTERAL**, désigne simplement une valeur fixe, nom identifiée (nombre, caractère, texte, etc.).

Exemple de littéraux : -100, 1, 3.14, 'z', "bonjour"

Un signe – (moins) précède une valeur numérique (ENTIER ou REEL) pour indiquer qu'elle est négative.

En C/C++, des valeur littérales peuvent être définies

- **En héxadécimal** : 0x2AF (débute par 0x)
- **En octal** : 0127 (débute par 0)

B. Expressions et opérateurs

Une **EXPRESSION** désigne un calcul associant des valeurs (littéraux, constantes et variables) et des opérateurs.
L'évaluation de cette expression fournit un résultat unique qui dépend des valeurs et des opérateurs utilisés.

Les opérateurs utilisés dans une expression dépendant du type de la valeur à évaluer : on trouve :

- Les opérateurs arithmétiques,
- Les opérateurs de comparaison et les opérateurs logiques.

LES VALEURS RESULTANT DE L'EVALUATION D'UNE AFFECTATION SONT PERDUES, A MOINS D'ETRE CONSERVEES : C'EST L'OBJET DE L'INSTRUCTION D'AFFECTATION.

1. Expressions arithmétiques

Une **EXPRESSION ARITHMETIQUE** est une expression utilisant des valeurs numériques (type ENTIER ou REEL) et des opérateurs arithmétiques

Opérateur arithmétique	Signification	Exemple avec des valeurs littérales	Résultat de l'évaluation	
+	Addition	2 + 3	5	
-	Soustraction	2 – 3	-1	
*****	Multiplication	3 * 2	6	
/	Division	3 / 2	1.5	
/	Division entière	3 / 2	1	
%	Modulo, reste de la division	3 % 2	1	

Dans une expression arithmétique, les calculs intermédiaires sont effectués selon la priorité des opérateurs : multiplication, division et modulo, puis addition et soustraction.
Par exemple : 2 + 3 * 5 ➔ effectuer (3 * 5), soit 15, puis ajouter 2 ➔ 17

L'USAGE DES PARENTHESES EST FORTEMENT CONSEILLE POUR DEFINIR PRECISEMENT LE CALCUL.
Par exemple : (2 + 3) * 5 ➔ effectuer (2 + 3), soit 5, puis 5 * 5 ➔ 25

2. Expressions logiques et opérateurs de comparaison (opérateurs relationnels)

Un **EXPRESSION LOGIQUE** est une expression dont le résultat est une valeur logique, soit VRAI ou FAUX.
Elle résulte de la combinaison de valeurs et d'opérateurs de comparaison

Opérateurs de comparaison	Signification	Exemple	Résultat de l'évaluation	
==	égal	2 == 3	**false**	
!=	différent	2 != 3	**true**	
<	Inférieur	2 < 3	**true**	
<=	Inférieur ou égal	2 <= 3	**true**	
>	Supérieur	2 > 3	**false**	
>=	Supérieur ou égal	2 >= 3	**false**	

Exemple :

```
(Vage >= 18) // VRAI si Vage est >= 18, FAUX sinon
(1 == 1) // toujours VRAI
(1 == 2) // toujours FAUX
```

3. Expressions logiques et opérateurs logiques

Un **EXPRESSION LOGIQUE** peut aussi résulter de la combinaison d'expressions logiques et d'opérateurs logiques.

Les opérateurs logiques **&&** (and) , **||** (or) et **!** (not) permettent la construction d'expressions logiques plus élaborées :

Expression logique A	Expression logique B	A **ET** B **&&** (and)	A **OU** B **||** (or)	**NON** A **!** (not)
F	F	F	F	V
F	V	F	V	V
V	F	F	V	F
V	V	V	V	F

Exemple :

```
((Vage >= 18) && (Vnote > 10)) // VRAI si Vage >= 18 ET Vnote > 10
((Vage >= 18) || (Vnote > 10)) // VRAI si Vage >= 18 OU Vnote > 10
```

Attention : cf. Conseil au sujet des expressions

C. L'affectation : opérateur =

Le **AFFECTATION** est l'opération qui donne une nouvelle valeur à une variable.

La nouvelle valeur doit correspondre au type de données de la variable.
L'ancienne valeur est bien entendue « écrasée » par la nouvellle.

Syntaxe :

```
nom_var1 = nouvelle_valeur ;
nom_var2 = nom_var3 = . . . = nouvelle_valeur ;
```

- **nom_var1, nom_var2, nom_var3**
  - identificateurs de variables préalablement déclarées
- **nouvelle_valeur**
  - valeur à affecter à la variable : valeur littérale, nom d'une variable ou d'une constante, expression

Exemples avec des valeurs littérales :

```
Vage = 20 ;
Vprime = 120.5 ;
VestMajeur = true ;
Vtitre = "titre du programme" ;
Vlettre = 'a' ;
i = j = k = l = 0 ; // initialisation de plusieurs variables à 0
```

Exemples avec des expressions :

```
Vage = (20 + 5) ;
Vprime = (120.5 * (1 + 0.50)) ;
VestMajeur = (Vage > 17) ;
```

D. Affectation et opérateurs arithmétiques

Le langage C/C++ propose des opérateurs permettant de simplifier l'écriture de certaines opérations.

1. Pré- et post- incrémentations

Simplifier l'écriture des opérations d'incrémentation et de décrémentation :

Opérateur arithmétique	Signification	Exemple avec des valeurs littérales	Equivalent à	
++	Incrémentation	i++	i = i + 1	Post incrémentation
		++i	i = i + 1	pré incrémentation
--	Décrémentation	i--	I = i - 1	Post décrémentation
		--i	I = i – 1	pré décrémentation

L'utilisation de ces opérateurs est très intéressante :

```
for (i=1 ;i<=10 ;i++) {;} // boucle 10 fois . . . pour rien . . .
```

ATTENTION : L'UTILISATION DANS DES EXPRESSIONS EST DELICATE (ET DECONSEILLEE) !!!!
EN EFFET

- **EN POST-, LA VALEUR EST UTILISEE PUIS L'OPERATION ARITHMETIQUE EST REALISEE**
- **EN PRE-, L'OPERATION ARITHMETIQUE EST REALISEE PUIS LA VALEUR EST UTILISEE**

2. Affectations composées (ou étendues)

Combiner l'affectation avec d'autres opérateurs :

Opérateur arithmétique	Signification	Exemple avec des valeurs littérales	Equivalent à	
+=	Addition et affectation	I+=5	i = i + 5	
-=	soustraction et affectation	i-=5	i = i - 5	
***=**	multiplication et affectation	i*=5	i= i + 5	
/=	Division et affectation	i/=3	i=i/3	

E. Conseil au sujet des variables

L'INITIALISATION D'UNE VARIABLE consiste à lui affecter une valeur initiale en fonction de son type de données.
TOUTE VARIABLE DOIT SUBIR UNE AFFECTATION AVANT D'ÊTRE UTILISEE SINON SON CONTENU EST INDETERMINE

F. Conseil au sujet des expressions

UTILISEZ DES PARENTHESES POUR DEFINIR CLAIREMENT LA PRIORITE DES OPERATIONS A EFFECTUER.

EVITER L'UTILISER LE CARACTERES ASSOCIATIF DES OPERATEURS DE COMPARAISON

Exemple à rejeter :

```
int a = 2;
int b = 2;
int c = 2;
if ( a < b < c ) ... ;// retourne VRAI, FAUX est plutôt attendu
if ( a == b == c ) ... ;// retourne FAUX, VRAI est attendu
```

Exemple à utiliser :

```
int a = 2;
int b = 2;
int c = 2;
if ( a < b && b < c ) ... ;  // retourne FAUX : OK
if ( a == b && b == c ) ... ; // retourne VRAI : OK
```

EVITER LES OPERATEURS POUVANT PRODUIRE DES EFFETS DE BORD DANS LES EXPRESSIONS DE COMPARAISON APRES LE PREMIER AND ou OR (évalué seulement si la première condition est suffisante pour évaluer l'expression)

Exemple à rejeter :

```
int i;
int max;
// ...
if ( (i >= 0 && (++i) <= max) ) {
// ...
}
```

Exemple à utiliser :

```
int i;
int max;
// ...
if ( (i >= 0 && (i + 1) <= max) ) {
  i++;
// ...
}
```

G. Transtypage, changement de type

Le **TRANSTYPAGE** consiste à demander explicitement la conversion de la valeur d'une expression en un certain type donné lors de son évaluation.

Syntaxe (en C):

```
int dividende ;
int diviseur ;
double resultat ;
resultat = ((double)dividende) / diviseur ;
```

Syntaxe (en C++) : A PRIVILEGIER

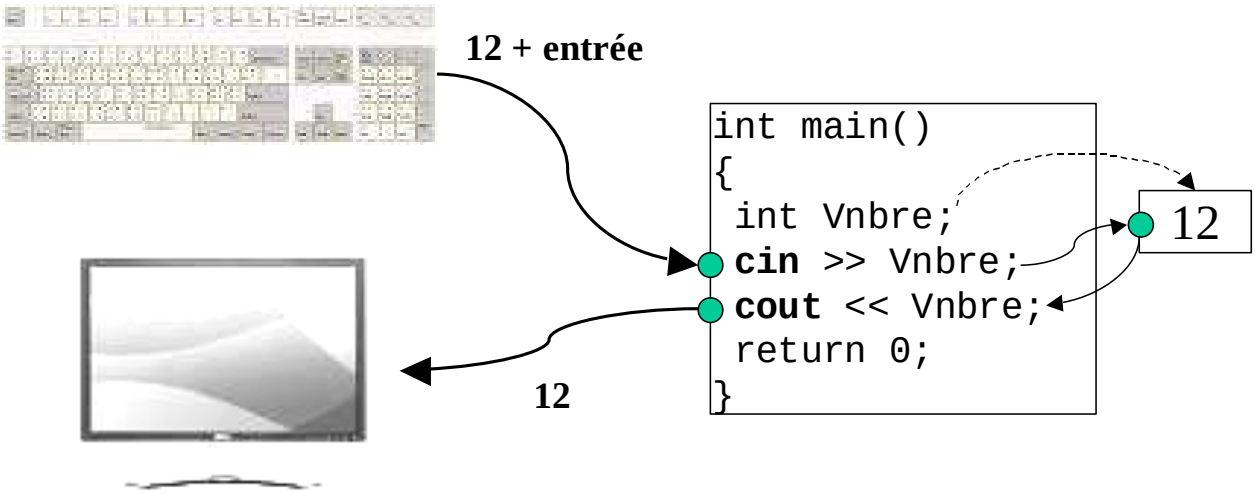
```
int dividende ;
int diviseur ;
```

```
double resultat ;
resultat = static cast<double>(dividende) / diviseur ;
```

IV. Instructions d'Entrée-sortie : les flux

Un flux représente un flot de données

- en provenance d'une source de données (clavier, par exemple),
- ou à destination d'une autre partie du système (écran, par exemple) .



A. Afficher des informations : flux de sortie

1. Librairie C++ <iostream> : cout

Syntaxe : avec l'opérateur d'insertion <<

cout << valeur1 << valeur2 << valeur3 << . . . ;

- **valeur1, valeur2, valeur3** représentent
  - les valeurs à envoyer sur le flux de sortie : littéraux (textes et nombres), variables, constantes
  - **endl** : saut de ligne
  - contrôle de formatage des sorties (iomanip.h) : spécifier une largeur : setw(col), un nombre de décimales : setprecision(n), etc.

```
cout << "bonjour !" ;
cout << "bonjour " << VnomEtudiant << endl ;
```

2. Librairie C stdio.h : puts, printf

puts / fputs : écrire une chaine de caractères (sans \0) et un CRLF sur stdout

puts(chaine) ; // écrit la chaine (sans \0) et un CRLF sur stdout
fputs(chaine, **stdout**) ; // écrit la chaine (sans \0) et un CRLF sur stdout

- **valeur1, valeur2, valeur3** représentent
  - les valeurs à envoyer sur le flux de sortie : littéraux (textes et nombres), variables, constantes

printf / fprintf: écrire une chaine de format avec substitution de valeurs de variables sur stdout

printf(format, variable1, variable2, variable3, . . .) ;

fprintf(**stdout**, format, variable1, variable2, variable3, . . .) ;

- **valeur1, valeur2, valeur3** représentent
  - les valeurs à envoyer sur le flux de sortie : littéraux (textes et nombres), variables, constantes
- **format** représente une chaîne de formatage de sortie
  - valeurs littérales à afficher : textes
  - caractères de formatage des valeurs en fonction de leur position

	Type de données à afficher	**Caractère de formatage % suivi de :**
Nombres	Entier décimal signé	d
	Entier décimal non signé	u ou i
	Etc.	
	Flottants de type double	F, e, g, E, G
Caractères	Isolé	c
	Chaînes de caractères (tableau)	s

```
printf("la valeur du nombre entier est %d", Vnombre) ;

printf("\nVous vous appelez %s", nom) ;
```

B. Récupérer des données saisies : flux d'entrée

1. librairie C++ <iostream> : cin, cin.getline, cin.get

Syntaxe 1 : avec l'opérateur d'extraction >>

cin >> variable1 >> variable2 >> variable3 >> . . . ;

- **variable1 variable 2, variable 3** représentent
  - des noms de variables déclarées qui vont stocker les valeurs saisies

```
cin >> Vnombre ;
cin >> Vnombre >> VprixUnitaire ;

char Vtexte[80] ;
cin >> Vtexte ; // récupère la saisie MAIS s'arrête au premier ' '
```

Syntaxe 2 :

cin.getline(variable_chaine**,** taille_maxi**)** ;

- **variable_chaine** : variable déclarée en tableau de caractères
- **taille maxi** : nombre maximal de caractères à récupérer

```
const int TAILLE_MAX = 40;
char Vmsg[TAILLE_MAX];
cout << "Entrez un texte :" << endl;
cin.getline(Vmsg,sizeof(Vmsg)); // récupération jusqu'à \n (EOL)
cout << "Vous avez saisi le texte suivant " << endl;
cout << Vmsg << endl;
```

Syntaxe cin.get : extrait un caractère du buffer et renvoie sa valeur:

cin.get();

2. librairie C <stdio.h> : scanf, fgets, getchar

Syntaxe scanf :

scanf(format, ref_variable1, ref_variable2, ref_variable3, . . .) ;

- **ref_variable1, ref_variable2, ref_variable3** représentent
  - o des références vers les variables variable1, variable2, variable3
- **format** représente la chaîne de format, c'est-à-dire les caractères de formatage des variables à saisir

```
scanf("%d", &Vnombre)
```

```
char nom[12] ;
scanf("%s", nom)
```

Attention :

- la lecture s'arrête au premier caractère espace
- La lecture d'une valeur supérieure à la taille du tableau cause une erreur de débordement (buffer overflow)

Syntaxe fgets :

fgets(ref_variable1, longueur, flux) ;

- **ref_variable1 :** référence vers la variable réceptrice
- **longueur :** longueur de la chaine (récupération d'un caractère en moins, pour la réservation du caractère de fin de chaine \0)
- **flux : stdin :** représente le flux d'entrée clavier

```
char nom[12] ;
fgets(nom, 12, stdin) ;
```

Attention :

- la lecture conserve le caractère « entrée » (\n) qu'il faudra éliminer du texte (localiser par strchr et remplacer par \0)

Syntaxe getchar :renvoie le caractère lu au clavier

getchar() ;

```
int ch ;
do
{
  ch = getchar();
} while (ch>=);
```

C. Conseil pour la saisie des nombres entiers

La récupération d'un flux par **cin** dans le but de récupérer une valeur entière présente un risque : si la valeur saisie n'est pas entière ? Cela produit une erreur à l'exécution.

```
int vAge ;
cin >> Vage ; // erreur si la saisie n'est pas un entier
```

La solution consiste en la récupération d'une chaîne de caractère et sa conversion en un entier après avoir effectué quelques vérifications (cela peut faire l'objet d'une fonction cf. chapitre 5):

```
char buff [25];
char *end_ptr;
long sl;
int Vage;
     Vage = 0 ;
cin.width(24);
cin >> buff; // saisie du tableau de caractères

errno = 0;

sl = strtol(buff, &end_ptr, 0);

if (ERANGE == errno) {
  cout << "nombre hors des limites" << endl;
}
else if (sl > INT_MAX) {
  cout << sl << " valeur trop grande!" << endl;
}
else if (sl < INT_MIN) {
  cout << sl << " valeur trop petite!" << endl;
}
else if (end_ptr == buff) {
  cout << "la saisie ne correspond pas à un nombre" << endl;
}
else if ('\0' != *end_ptr) {
  cout << "caractères non attendus dans la saisie" << endl;
}
else {
  Vage = static_cast<int>(sl);
}
```

TOUTE DONNEE PROVENANT D'UNE SOURCE EXTERIEURE (SAISIE, FICHIER, ETC.) DOIT ETRE CONTROLEE AFIN DE LIMITER LES POSSIBLES EFFETS DE BORD

1. **IDENTIFIER TOUTES LES SOURCES POSSIBLES D'ACQUISITION DE DONNEES ;**
2. **IDENTIFIER LES POINTS D'ACQUISITION DE DONNEES DANS LE CODE SOURCE ;**
3. **DEFINIR LES CRITERES POUR LES DONNEES SOIENT VALIDES (NOMBRES, CHAINES DE CARACTERES, DATES, ETC.) ;**
4. **DEFINIR LE COMPORTEMENT DU PROGRAMME SI UNE DONNEE ERRONEE EST REÇUE ;**
5. **METTRE EN ŒUVRE LE CODE SOURCE POUR CONTROLER LES DONNEES.**

V. Chaînes de caractères C et C++

Deux formes de chaînes de caractères sont disponibles en C/C++ :

- La forme C standard, un tableau de caractères :
- La forme C++ standard, une classe d'objets contenant une chaine de caractères

Chacune des 2 formes pourra être traitée avec des fonctions différentes.

A. Chaînes de caractères C : char[]

Une chaîne de caractères est une suite de caractères contigus en mémoire et dont le dernier caractère marque la fin : c'est le caractère NUL, '\0'
L'utilisation de fonctions de manipulation des tableaux de caractères nécessite l'inclusion d'un fichier d'entêtes de fonctions pour gérer ces chaînes :

```
#include <string.h>
```

1. Déclaration

Plusieurs formes de déclaration sont possibles :

```
char Vnom[24] ; // Vnom : longueur de 24 caractères (23 + \0)
char Vnom[] = "Jean Paul" ; // Vnom :  10 (9 + \0)
char Vnom[] = {'J','e','a','n',' ','P','a','u','l'} ; // idem.
```

Un tableau de chaînes de caractères :

```
char Tnom[10][24] ; // Tableau de 10 noms de 24 caractères chacun
```

2. Les fonctions de la librairie string.h

- la fonction **strcat**(Vstr1, Vstr2) : ajoute une chaine Vstr1 au bout de la chaine Vstr1

```
cout << "avant = " << Vstr1;
strcat(Vstr1, Vstr2) ; // ajoute Vstr2 à la fin de Vstr1
cout << "apres = " << Vstr1;
```

- la fonction **strcmp**() : permet la comparaison de 2 chaînes

```
char Vch1[] = "Pierre", Vch2[] = "Paul" ;
if (strcmp(Vch1, Vch2) != 0) cout << "différent" ;
```

```
if (strcmp(Vstr1, Vstr2)==0) cout << "egal" << endl;
              else cout << "différent" << endl;
```

- la fonction **strcpy**() : AFFECTATION : permet la copie d'une chaîne vers une autre

```
cout << "avant = " << Vstr1;
strcpy(Vstr1, Vstr2) ; // copie Vstr2 dans Vstr1
cout << "apres = " << Vstr1;
strcpy(Vstr1, "bonjour") ; // affecte (copie) 'bonjour' dans Vstr1
```

B. Chaînes de caractères C++ : classe string()

L'utilisation nécessite l'inclusion d'un fichier d'entêtes de fonctions pour gérer ces chaînes :

```
#include <string>
```

1. Déclaration d'une instance de 'string'

Plusieurs formes de déclarations sont possibles, en voici quelques syntaxes appuyées par des exemples :

- string nom_chaine() ; // créer une chaînes vide

Exemples :

```
string Vstr1;   //   Vstr1 est vide
string Vstr1(""); //   Vstr1 est vide
string Vstr2( "bonjour" ); //   Vstr2 contient "bonjour"
```

- string nom_chaine1(nom_chaine2) ; // créer une chaine identique à une seconde

Exemple :

```
string Vstr3( Vstr2 );  //   Vstr3 contient "bonjour"
```

- string nom_chaine1(nom_chaine2, position, nombre_caractères) ; // créer une chaîne à partir du contenu d'une autre chaîne, à partir d'une position de caractère, sur une certaine longueur

Exemple :

```
string Vstr4( Vstr2, 1 );       //   Vstr4 contient "onjour"
string Vstr5( Vstr2, 3, 2 );    //   Vstr5 contient "jo"
```

- string (nombre, caractère) ; // créer une chaîne de caractères contenant plusieurs fois un caractère

Exemple :

```
string Vstr6( 10, '*' );     //   Vstr6 vaut "**********"
```

Un tableau de chaînes de caractères :

```
string Tnom[10] ; // Tableau de 10 noms
```

2. Les fonctions membres de la classe 'string'

- la fonction **length**() (la fonction size() aussi); : fonction de type entier non signé qui retourne la longueur d'une chaîne

```
cout << Vstr1.length(); // affiche la longueur de la chaine
```

- la fonction **empty**() : fonction de type logique qui indique si la chaîne est vide (bool empty ()

```
if (Vstr4.empty()) cout << "Vstr4 vide" << endl;
              else cout << "Vstr4 non vide" << endl;
```

- la fonction **find**() : retourne la position d'une sous-chaine dans une chaine

```
cout <<  "position de 'on' " << Vstr2.find("on") ; // affiche 1
```

- la fonction **substr**() retourne une sous-chaine à partir d'une position sur une certaine longueur

```
cout << Vstr2.substr(1,2); // affiche "on"
```

3. Opérateurs pour la classe ‘string’

- Affectation (assignation d’une valeur) : =

```
string Vstr1();
string Vstr2("bonjour");
...
Vstr1 = Vstr2; // le contenu de Vstr2 est copié dans Vstr1
```

- Ajouter à la fin : +=

```
string Vstr1( "abc" );
string Vstr2( "def" );
...
Vstr1 += Vstr2; // Vstr1 contient "abcdef" après l’affectation
```

- Concaténer : +

```
string Vstr1( "abc" );
string Vstr2( "def" );
string Vstr3;
...
Vstr3 = Vstr1 + Vstr2; // Vstr3 vaut "abcdef" après l’affectation
```

- comparaison : opérateurs classiques

```
string Vstr1( "abc" );
string Vstr2( "def" );
bool Vtest = ( Vstr1 == Vstr2 ); // Vtest vaux ‘false’
```