Notions fondamentales de C++

I. Exemple Introductif

Le meilleur moyen d'introduire le langage C++ est d'étudier un exemple simple.

```
# include <iostream.h>
# include <conio.h>

main( )
{
int x;
cout << "Hello world!";
x = 15;
cout << "la variable x vaut "<< x;
getch( ) ;
}
```

Détaillons ce petit programme (on étudiera les 2 premières lignes plus tard)

- tout programme en C++ commence par **main ( )** (main veut dire principal). main est en fait le nom du programme principal, le seul autorisé.

- le corps du programme est écrit entre **accolades { }**.

- la première instruction déclare une variable entière nommée x. La **déclaration** commence sans mot clé préalable, avec le type (ici int qui signifie entier) suivi de l'identificateur. Comme toute instruction, la déclaration se termine par un **;**

- **cout <<** permet d'effectuer un **affichage** à l'écran. C'est la façon la plus simple d'afficher à l'écran, mais pour pouvoir l'utiliser, il faut avoir préalablement inscrit la directive **#include <iostream.h>** avant le début du programme. cout << "Hello world!"; affiche le message entre guillemets à l'écran.

- L'instruction x = 15 est une instruction d'affectation. Attention, en C++, **=** est l'opérateur d'**affectation**, ce qui peut entrainer des confusions avec l'opérateur d'égalité des mathématiques.

- l'instruction suivante permet d'afficher la valeur de x précédée d'un libellé. Pour afficher des contenus différents à l'écran, il faut les séparer de l'**opérateur <<** (qui remplace la virgule de l'algorithmique)

- l'instruction **getch( )** est l'appel d'une fonction qui permet à l'utilisateur de voir la fenêtre d'exécution tant qu'il n'a pas appuyé sur une touche. Sans cette ligne, la fenêtre d'exécution se fermerait tout de suite sans que l'utilisateur ait le temps de la visualiser (c'est à cause d'un bug de Windows)

- le programme se termine par la fermeture des accolades.

II. Les variables en C++

A. Les types simples du langage C++

Il existe 5 types principaux en C++.
- les caractères **char**
- les entiers **int**
- les réels simples **float**
- les réels doubles **double**
- les booléens **bool**

Ces types sont signés par défaut mais on peut indiquer "unsigned" si l'on veut un type non signé. Il existe plusieurs sortes des différents types, qui varient selon les machines. Il n'y a pas de normalisation en la matière.
Par exemple, Le C++ de Borland annonce dans son aide:

TYPE	LONGUEUR		DOMAINE DE VALEURS
unsigned char	8 bits	1 octet	0 à 255
char	8 bits	1 octet	-128 à 127
unsigned int	16 bits	2 octets	0 à 65 535
short int	16 bits	2 octets	-32 768 à 32 767
int	32 bits	4 octets	-2 147 483 648 à 2 147 483 647
unsigned long	32 bits	4 octets	0 à 4 294 967 295
long int	32 bits	4 octets	-2 147 483 648 à 2 147 483 647
float	32 bits	4 octets	$3.4*10^{-38}$ à $3.4*10^{+38}$
double	64 bits	8 octets	$1.7*10^{-308}$ à $1.7*10^{+308}$
long double	80 bits	8 octets	$3.4*10^{-4932}$ à $1.1*10^{+4932}$

➢ Complément sur le type caractère

En C++, la notion de caractère dépasse celle de caractère imprimable, c'est à dire auquel est associé un graphisme (lettres, chiffres, ponctuation, caractères spéciaux,...). Par exemple, les sons, les retours chariot, les tabulations, les sauts de page sont aussi des caractères.
Quand un caractère est imprimable, il est noté entre apostrophes:
'a', '+', '5', '_'
Les caractères non imprimables possèdent une représentation conventionnelle utilisant le caractère "\" nommé antislash.
Voici la liste de ces caractères:

\a	bip
\b	retour arrière
\f	saut de page
\n	retour chariot avec saut de ligne
\r	retour chariot sans saut de ligne

\t tabulation

Chaque caractère est codé en mémoire suivant un code binaire appelé ASCII. Donc, à chaque caractère est associé un nombre dont la représentation binaire est la même. Il est possible d'effectuer des opérations numériques sur les variables de type caractère à travers leur code ASCII. Il est aussi possible d'afficher le code ASCII d'un caractère en le transformant au préalable en entier.
Exemple:

La séquence suivante :

```
char a;
a = 'Z';
cout << a << "\n";
cout << int (a);
```

donne à l'écran:

```
Z
90
```

En effet, la variable a contient le code ASCII de la lettre Z, qui est aussi la représentation binaire de l'entier 90.

➢ **Les type chaîne**

Le type chaîne n'existe pas dans le langage de base du C++. On peut cependant utiliser un type chaîne (string) en incluant la directive # include <cstring.h> au début du fichier.

B. les déclarations

Les déclarations de variables se font généralement en début de programme, après main( ), sans qu'il soit utile de les précéder d'un mot clé. La déclaration indique tout d'abord le type suivi de l'identificateur et d'un point virgule.

type identificateur ;

ex :

```
int ma_variable ;
```

➢ **Contraintes sur les identificateurs**

- Les identificateurs doivent **commencer par une lettre ou le caractère _** (soulignement) et **ne doivent pas contenir d'espaces, ni de lettres accentuées.**

ATTENTION!!
Le langage C fait la différence entre les majuscules et les minuscules!
Ainsi , les identificateurs nb et Nb désignent deux variables distinctes!

- Les mots du langage ne peuvent pas servir d'identificateurs. Ils sont réservés.

On peut déclarer plusieurs variables de même type dans une même instruction en séparant les identificateurs par des virgules.
exemple :

```
double x, var, toto ;
```

➢ **déclaration des constantes**

La déclaration des constantes peut se faire après le mot clé **const**. On spécifie le type de la constante, son identificateur puis sa valeur après l'opérateur =.
exemple :

const float ttva = 0.196

C. opérations élémentaires

➢ Les opérateurs de comparaison

opérateur	**symbole utilisé en C++**
egal	= =
différent	!=
supérieur	>
supérieur ou égal	>=
inférieur	<
inférieur ou égal	<=

➢ Les opérateurs arithmétiques

opérateur	**symbole utilisé en C++**
addition	+
soustraction	-
multiplication	*
division réelle	/
division entière	/ (même opérateur que pour la division réelle)
modulo	%

Remarque : il n'existe pas d'opérateur « au carré » ou d'opérateur « puissance ». Pour cela, nous verrons plus tard qu'il faut utiliser des fonctions.

➢ Les opérateurs logiques

ET &&
OU || (Alt Gr +6)
NON !

Ces opérateurs sont utilisés pour créer des conditions complexes.

D. Initialisation et déclaration

Il est possible d'initialiser les variables dès leur déclaration.
Ex :

```
int a , b ;
float c ;
a  = 5 ;
b = 0 ;
c = 1000 ;
```

peut s'écrire

```
int a = 5, b = 0 ;
float c = 1000
```

III. Les instructions élémentaires

A. L'affectation

Rappelons que l'affectation est totalement différente de l'égalité mathématique. Cette instruction permet seulement de donner la valeur de l'expression de droite à la variable de gauche.
Mais **le langage C++ utilise le symbole de l'égalité = pour l'affectation.**
Pour affecter un caractère, il faut l'entourer d'apostrophes (sinon le compilateur pourrait le confondre avec un nom de variable). Pour une chaîne, il faut utiliser des guillemets.

Exemples :

```
x = 5;
```

La variable x prend pour valeur 5

```
i = i + 1;
```

La variable i prend pour valeur son ancienne valeur augmentée de 1

```
C = 'a';
```

La variable C prend pour valeur le caractère a.

B. L'affichage

En C++, plusieurs opérateurs sont possibles pour l'écriture. Nous ne présentons ici que le plus simple, l'opérateur **<<** précédé du mot clé **cout**(prononcer c-out).
Pour utiliser cet opérateur, il faut avoir préalablement écrit au tout début du programme :

```
# include <iostream.h>
```

cout << suivi du nom d'une variable, d'une expression ou d'un texte (entre guillemets) inscrit le contenu de la variable, le résultat d'une expression ou bien du texte à l'écran.

Exemple:

```
cout << y – x;
```

Cette instruction affiche 5 si y=10 et x=5

Si l'on veut qu'apparaissent successivement différents affichages (du texte, puis des variables par exemple), on les séparera avec l'opérateur << .

Exemple :

```
cout<< "la valeur de x est " << x << " et celle de y est " << y ;
```

En supposant que x ait pour valeur 5 et y pour valeur 10, on aura à l'écran:

```
La valeur de x est 5 et celle de y est 10
```

Pour inclure des sauts de ligne dans un texte, on utilise le caractère \n .

Exemple :

```
cout << "Bonjour\nCa va?" ;
```

Va donner à l'écran :

```
Bonjour
Ca va?
```

C. La saisie

En C++, la lecture s'effectue le plus souvent grâce à cin (lire c-in) suivie de l'opérateur >>.
Pour pouvoir l'utiliser, il faut aussi que la directive **#include <iostream.h>** apparaisse au début du fichier source.
cin >> lit au clavier les valeurs introduites successivement.

Par exemple:

```
cin >> a >> b ;
```

Si l'on écrit au clavier 2 puis 3, la variable a va prendre pour valeur 2 et b va prendre pour valeur 3.

D. Les commentaires

Il existe deux façons d'inclure des commentaires dans un programme en C++:

- soit on les entoure de /* et */
- soit on utilise // pour un commentaire qui se finit à la fin de la ligne

IV. Exemple complet

Pour récapituler voilà un exemple de programme interactif avec son comportement.

```
# include <iostream.h> // pour pouvoir utiliser cout<< et cin>>
# include <conio.h>    // pour pouvoir utiliser getch()

main ()
{
//déclaration
int ddn ;

// affichage d'un libellé
cout << "Bonjour! \nEntrez votre annee de naissance puis validez, s'il vous plait\n" ;
// on affecte la valeur inscrite au clavier à la variable ddn avec // cin>>
cin >> ddn ;

// On affiche l'âge grâce à un petit calcul
cout <<  "Vous avez, ou allez avoir dans l'annee "<< 1999 - ddn << "ans.\n";

/* grâce à l'instruction getch (),le programme attend la pression
d'une touche pour laisser le temps de lire le résultat*/
cout << "\nTapez sur une touche pour sortir";
getch();

}
```

Déroulement du programme à l'écran:

```
Bonjour!
Entrez votre annee de naissance puis validez
```

On introduit l'année, par exemple 1979

```
Bonjour !
Entrez votre annee de naissance puis validez
```

```
1979
Vous avez, ou allez avoir dans l'annee 20 ans.

Tapez sur une touche pour sortir
```

V. Les structures de contrôle

A. Les structures conditionnelles

La structure if...else

La structure conditionnelle du C++ a le formalisme suivant:

```
if (condition) //pas de then
{
   traitement1 //exécuté si la condition est vraie
}
else
{
   traitement2 //exécuté si la condition est fausse
}
```

Un traitement est composé d'une ou plusieurs instructions terminées par un point-virgule.
Remarque: si le traitement n'est composé que d'une seule instruction, les accolades ne sont pas obligatoires.

Le else est facultatif. On peut avoir tout simplement:

```
if (condition)
{
   traitement
}
```

Si la condition est fausse, le programme "saute" le traitement et passe à la suite.

Rappel sur les opérateurs:
Les opérateurs de comparaison utilisables dans la condition sont:

>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à
==	égal à
!=	différent de

Les opérateurs logiques utilisés pour les conditions complexes sont les suivants:

opérateur	symbole
et	&&
ou (non exclusif)	||
non	!

Lorsque la condition est complexe, chaque condition simple est entourée de parenthèses, ainsi que la condition toute entière.

Remarquez qu'**il n'y a pas de mot qui indique la fin de la structure**. En fait, la structure s'arrête automatiquement après le traitement suivant le else (ou après l'accolade s'il n'y a pas de else). S'il n'y a pas d'accolades, la structure se termine à la fin de l'instruction suivant la condition, sinon, elle se termine à l'accolade de fermeture. Il est donc très important de bien présenter son programme (utiliser les tabulations) pour retrouver où se terminent les différentes structures.

La structure switch...case (en C++)

L'instruction switch offre au programmeur un moyen d'effectuer un choix multiple par valeurs. En d'autres termes, elle permet de faire l'équivalent d'une structure Selon de l'algorithmique, lorsque les cas sont des valeurs discrètes (et non des intervalles).
Cette structure de contrôle ne comporte donc pas de condition. En revanche, une expression va être évaluée et en fonction du résultat de cette évaluation, des instructions vont être réalisées.

Voici le formalisme de cette structure:

```
switch (expression)
{
   case va1eur1:
        traitement1
        break;
   case constante2 :
        traitement2
        break;
   ...
   case constanteN :
        traitementN
        break;
   [default:                    //les crochets ne sont pas à écrire: ils indiquent
        traitement]             //seulement que cette clause est facultative

}
```

La clause default est facultative.

Voilà comment fonctionne cette structure de contrôle:
- elle commence par évaluer l'expression fournie (qui est le plus souvent une variable)
- pour chaque clause "case", elle compare le résultat de l'expression à la valeur constante. En cas d'égalité, les instructions sont exécutées jusqu'à l'instruction "break;"
- les instructions appartenant à la clause "default" sont exécutées dans le cas où si aucune constante n'est égale à l'expression.

Remarques:

- L'évaluation est toujours basée sur l'**égalité**. Il est interdit d'utiliser des opérateurs logiques ou des valeurs booléennes. Les cas ne peuvent pas être des intervalles.
- L'expression doit retourner un **type entier** et les constantes sont aussi de type entier.
- Les instructions suivant un "case" ne sont **pas obligatoirement regroupées en bloc**. Il ne s'agit pas d'une instruction composée mais bien de plusieurs instructions en séquence.
- Il faut terminer les instructions suivant un "case" par un "**break**", sinon le compilateur enchaîne sur les instructions du "case" suivant. (L'instruction break sert en fait à sortir directement de la structure switch)
- Lorsqu'une même séquence d'instructions est exécutée pour plusieurs cas qui se suivent, il suffit de laisser se suivre les clauses "case" sans leur attribuer d'instructions (même pas "break") sauf à la dernière (voir l'exemple)

Exemple:
Ce programme affiche un commentaire à partir de la saisie d'une note entière sur 10 (pas de demis-points). Certaines notes voisines se verront attribuer le même commentaire.

```
main( )
{
int note;
cout << "Entrez la note sur 10";
cin >> note;
switch (note)
{
   case 0:
   case 1:  cout << "\nC'est nul"
   case 2:
   case 3:  cout << "\nC'est très insuffisant";
            break;
   case 4:  cout << "\nC'est insuffisant";
            break;
   case 5:
   case 6:  cout << "\nC'est moyen";
            break;
   case 7:
   case 8:  cout << "\nC'est bien";
            break;
   case 9:
   case 10: cout << "\nC'est très bien";
            break;
   default: cout << "\nErreur, la note doit être entière et comprise entre 0 et 10";
}
```

Attention: il ne faut pas oublier les "break", sinon le compilateur exécute toutes les instructions suivantes, y compris les instructions qui font partie des "case" suivants.

Dans notre exemple, si on avait oublié les "break", voilà ce qui se serait passé.
Supposons que la note saisie soit 5.
Le commentaire "C'est moyen" s'affiche mais aussi "C'est bien", "C'est très bien" et "Erreur, la note doit être entière et comprise entre 0 et 10" !

Comment faire l'équivalent d'une structure Selon où les cas sont des intervalles?

En C++, utiliser switch est dans ce cas impossible: il faut utiliser l'imbrication des structures if.
Un traitement qui suit un if ou un else peut être une structure de contrôle, par exemple une structure if. On dit dans ce cas qu'on a des structures imbriquées.

L'algorithme suivant se traduit de la façon suivante:

Selon montant **Faire**
  <1000 : taux ← 1
  =1000 et < 3000: taux ← 2
  =3000 et < 10000: taux ← 3
  = 10000: taux ← 4
FinSelon

```
if (montant <1000)
  taux = 1;
else if (montant < 3000)
  taux = 2;
else if (montant < 10000)
  taux = 3;
else
  taux = 4;
```

Remarquez que comme les traitements ne comportent qu'une seule instruction, on ne met pas d'accolades qui alourdiraient le code. Mais attention à ne pas les oublier quand un traitement comporte plusieurs instructions!

La boucle while

La boucle while correspond à la boucle Tant que ... Faire de l'algorithmique. Elle a la syntaxe suivante:

while (*condition***)**
{
traitement
}

La condition est tout d'abord évaluée. Si elle est fausse, le traitement de la boucle (ou corps de la boucle) n'est pas exécuté et on passe à l'instruction suivante. Si elle est vraie, le traitement est exécuté, puis la condition est de nouveau évaluée. Le traitement est recommencé tant que la condition est vraie.
Le traitement peut être une instruction simple (sans accolades), un bloc d'instruction entre accolades ou encore une instruction structurée (structure conditionnelle ou répétitive). Dans ce cas, les accolades ne sont pas nécessaires mais elles peuvent permettre une meilleure lisibilité.
Remarque: il n'y a pas de mot réservé qui indique la fin du corps de la boucle. La fin de la boucle est soit le point-virgule si le traitement n'est formé que d'une instruction simple, soit la fin de l'instruction structurée si le traitement est une structure de contrôle, soit l'accolade de fermeture si le traitement est formé d'un blocs d'instructions. **Attention: lorsque le traitement de la boucle est formé de plusieurs instructions, ne pas oublier de les mettre entre accolades!**

Exemples:

Cas où le traitement est une instruction simple
Voilà un programme qui utilise une boucle while pour afficher les carrés des 10 premiers nombres entiers naturels.

```
main ( )
{
int e = 1;
while (e <= 10)
      cout << e * e << "\t";  // le corps de la boucle est une instruction simple. Il se termine au point-virgule
cout << "Fin";          // cette instruction est en dehors de la boucle
}
```

Cas où le traitement est une instruction structurée
Voilà un programme qui utilise une boucle while pour afficher le signe des nombres saisis par l'utilisateur. Pour arrêter, l'utilisateur saisi 0.

```
main ( )
{
int n;
while (n != 0)
{
  if (n < 0)
     cout << "Nombre positif";
  else
    cout << "Nombre négatif";
}
cout << "Au revoir! ";        // cette instruction ne fait pas partie de la boucle
}
```

Cas où le traitement est un bloc d'instructions

Voilà un programme qui utilise une boucle while pour afficher le cube des entiers positifs saisis.

```
main ( )
{
int n;
cout << "Entrez un entier positif ou terminez par –1";
cin >> n;
while (n <= 0)
    {
     cout << n << "au cube vaut "<< n * n * n << "\n";
     cout << "Entrez un entier positif ou terminez par –1";
     cin >> n;
     } / / fin de la boucle
cout << "Aurevoir!";
} / / fin du programme
```

corps de la boucle

La boucle for

La boucle for permet, comme la boucle pour de l'algorithmique, d'exécuter un traitement un nombre connu de fois, plus simplement qu'avec la boucle while.
La syntaxe de la boucle for est la suivante:

for (*instruction1* **;** *condition_de_continuation* **;** *instruction3***)**
{
    traitement
}

Si le traitement n'est composé que d'une seule instruction, les accolades peuvent être omises.

- L'instruction1 est exécutée avant l'entrée dans la boucle. Le plus souvent, on l'utilise pour initialiser le compteur.
- La condition est vérifiée au début de chaque tour de boucle et si elle est vraie, le corps de la boucle est exécuté.
- L'instruction2 est exécutée à la fin de chaque tour de boucle. Le plus souvent, on l'utilise pour incrémenter le compteur. En effet l'instruction for, contrairement à l'instruction Pour de l'algorithmique, n'effectue pas automatiquement l'incrémentation du compteur à chaque tour.

Le plus souvent, l'instruction for sera utilisée pour réaliser l'équivalent d'une boucle Pour, ce cette façon:

for (*initialisation_compteur; condition_de_continuation; incrémentation_compteur*)
{
    traitement
}

Exemple:

```
for (i=1; i <= 10; i = i + 1)        / /le traitement va s'effectuer 10 fois
{
  ...
}
```

💣 Attention à la condition! Si vous écrivez for (i = 1; i < 10; i = i + 1) le traitement ne s'exécute que 9 fois.

La boucle for n'est en fait qu'une variante de la boucle while qui correspond à:

initialisation-compteur;
while (*condition_de _continuation*)
{
    traitement
    incrémentation_compteur
}

Exemple:
Voici un programme qui affiche la table de la somme et de multiplication du 5.

avec for

```
main ( )
{
int i;
for (i = 1; i <=10 ; i = i+1)
{
    cout << "5 + "<< i << " = "<< 5 + i;
    cout << "5 * " << i << " = " << 5 * i ;
}
}
```

avec while

```
main ( )
{
int i;
i = 1;
while (i <= 10)
{
   cout << "5 + "<< i << " = "<< 5 + i;
   cout << "5 * " << i << " = " << 5 * i ;
   i = i + 1 ;
}
}
```

La boucle do...while

La boucle do ... while a la syntaxe suivante:

do
{
traitement
}
while (*condition de continuation***) ;** *// ne pas oublier les parenthèses et le point-virgule!!*

Les accolades peuvent être omises si le traitement ne comporte qu'une seule instruction.
Cette boucle se rapproche de la boucle Répéter...jusqu'à de l'algorithmique, sauf au niveau de la condition.

DIFFERENCE AVEC L'ALGO

Dans une boucle répéter, on sort lorsque la condition est vraie, alors que dans une boucle do...while, on sort lorsque la condition est fausse.

La condition est vérifiée après l'exécution du traitement donc ce dernier sera toujours **réalisé au moins une fois.**
Toute instruction réalisée avec une boucle do...while peut être aussi écrite avec une boucle while, alors que la réciproque n'est pas vraie; c'est pourquoi, la boucle do...while n'est utilisée que très rarement.

Exemple:
Ce programme utilise une boucle do...while pour faire saisir à l'utilisateur un nombre inférieur à 100

```
main ( )
{
int nb;
do
{
   cout << "Entrez un nombre inférieur à 100, svp";
   cin >> nb;
}
while (nb >= 100);
cout<< "merci!";
}
```