Université Mouloud Mammeri Tizi-ouzou
Faculté d'électronique et d'informatique
Département d'électronique

Cours C++ et programmation orientée objet

Notions de base

Mr. ABAINIA

Master µElectronique et instrumentation

Objectifs du cours ?

- Savoir résoudre les problèmes et concevoir un algorithme.
- Maîtriser les bases d'un langage évolué (c++).
- S'initier à la programmation avancée (POO).
- S'initier au monde des systèmes embarqués.

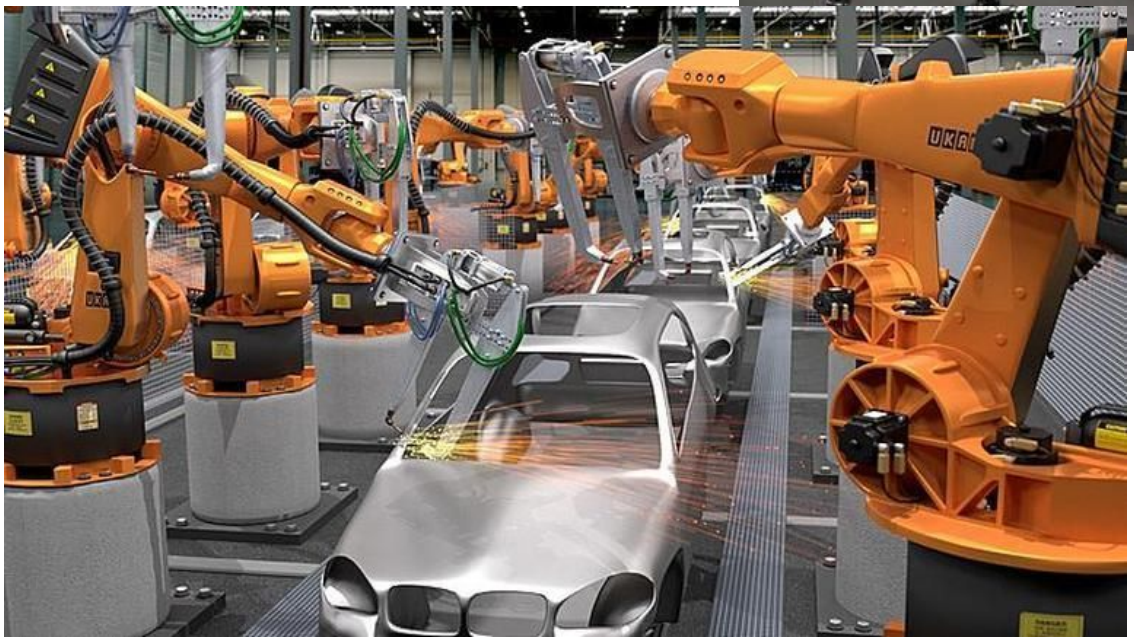
Langage c++ ?

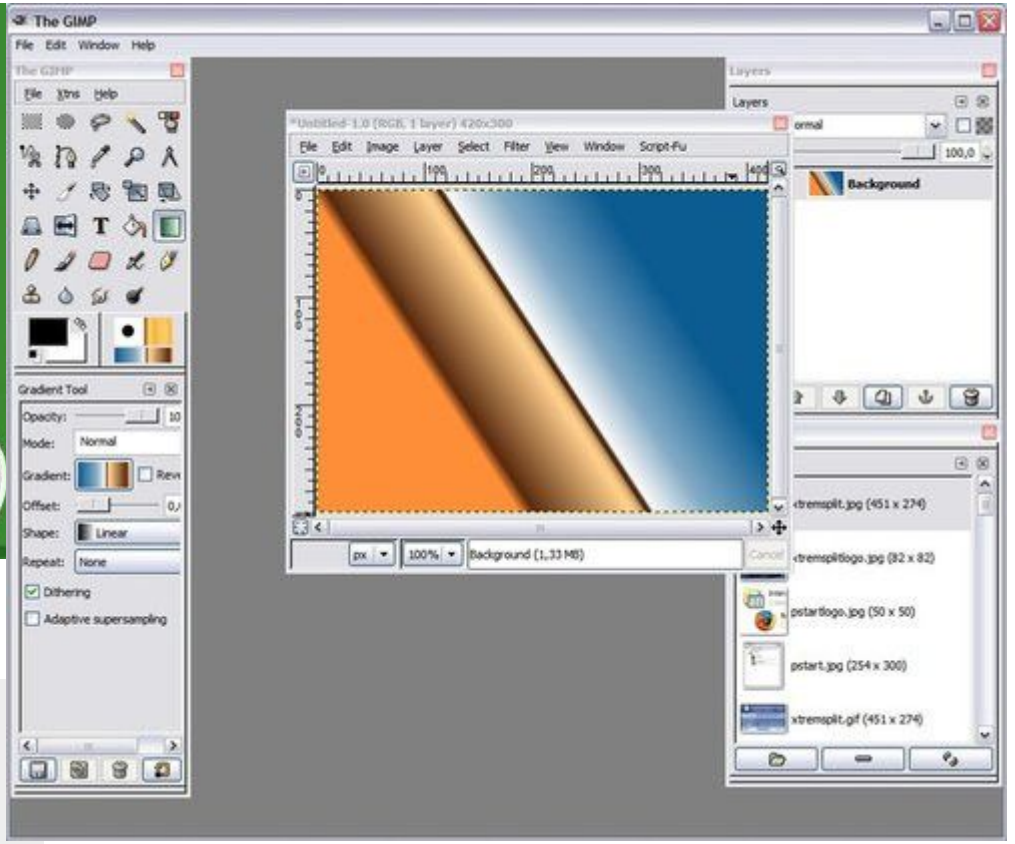
- Descendant du célèbre langage de programmation C.
- Père des langages modernes basés sur POO.
- Langage de bas-niveau et de haut-niveau à la fois.
- Plus performant et plus rapide -> un peu complexé.
- On peut faire tout avec !


131
X9


UNO
ARDUINO


BostonDynamics




The GIMP
File Edit Window Help
Gradient Tool
Opacity:
Mode: Normal
Gradient:
Offset:
Shape: Linear
Repeat: None
Dithering
Adaptive supersampling
File Edit Image Layer Select Filter View Window Script-Fu
Layers
Background
Background (1,33 MB)


A
O
X
P
N
W

- Portable (Qt).
- Nombreuses bibliothèques.
- Communauté active (entraide).
- Multi-paradigmes (différentes façons de programmation).

Création du C++ ?

Proverbe

“*La nécessité est la mère de l’invention*”

- C++ a été créé par Bjarne Stroustrup en 1982.
- Inspiré du langage C, ALGOL 68, Simula, ADA, CLU & ML.
- Abstraction de données, POO, programmation générique
- À l'époque, il y avait deux catégories de langages:
  - ✓Langages de production (mal fichus)
  - ✓Langages pour la recherche (accès limités)

Bjarne Stroustrup

- ***Même Stroustrup ne maîtrise pas le C++ à 100%.***

Différents standards du C++ ? (versions)

- **C++98** (normes ISO/CEI 14882:1998).
- **C++98/3** (normes ISO/CEI 14882:2003).
- **C++11** (normes ISO/CEI 14882:2011).
- **C++14** (normes ISO/CEI 14882:2014).
- **C++17** (normes ISO/CEI 14882:2017).

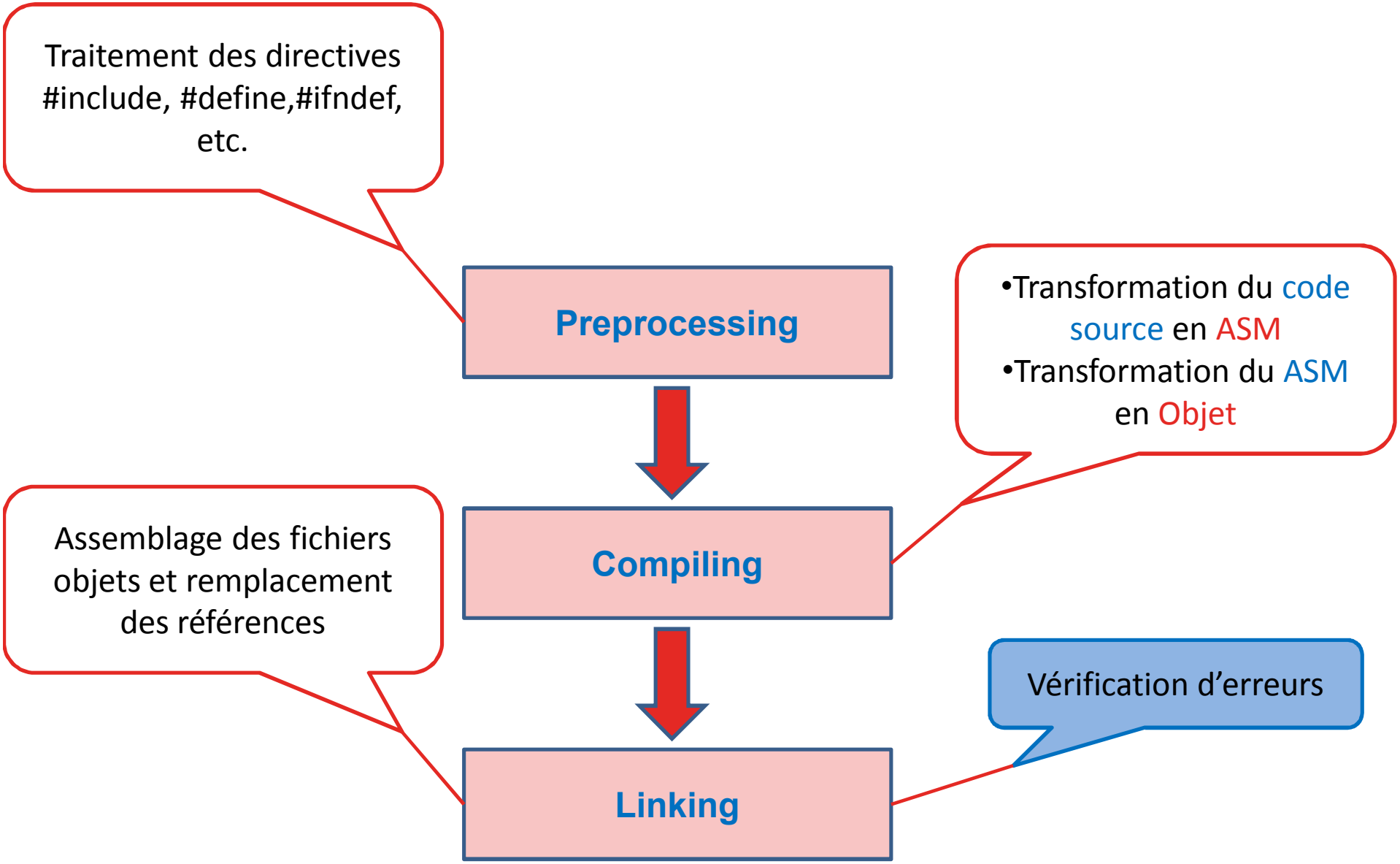
Créer un programme ?

- **Analyser les besoins.**
- **Réfléchir sur une solution.**
- **Ecrire le code source dans un éditeur.**
- **Compiler le programme avec un compilateur.**
- **Exécuter le programme.**
- *Tester et debugger.*

Quelques compilateurs c++ ?
(Gratuits)

- Apple C++, Oracle C++, IBM C++, Intel C++
- MingW, gcc, g++
- CygWin
- Digital Mars C++.
- DJ Delorie's C++

Processus de compilation ?


Traitement des directives #include, #define,#ifndef, etc.
Preprocessing
•Transformation du code source en ASM
•Transformation du ASM en Objet
Assemblage des fichiers objets et remplacement des références
Compiling
Vérification d'erreurs
Linking

Compilation par lignes de commandes ?

❖**Se fait en terminal sous Linux et cmd sous Windows.**

❖**g++ –opt1 –opt2 –opt3 –opt4 …**

Option	Utilité
-I	Chemin de recherche des fichiers entêtes supplémentaires
-L	Chemin de recherche des bibliothèques supplémentaires
-l	Nom d'une bibliothèque supplémentaire (ex. libcrypt.so on écrit -lcrypt)
-o	Nom du fichier exécutable
-E	Arrêter la compilation après le passage du préprocesseur et avant le compilateur
-S	Arrêter la compilation après le passage par l'assembleur
-c	Arrêter la compilation après l'assemblage en laissant les fichiers objets dispos
-Wall	Activer tous les avertissements
-g	Inclure des informations nécessaires pour utiliser le débuggeur

```
g++ -Wall –g programme1.cpp –c
g++ -Wall –g programme2.cpp –c
g++ programme1.o programme2.o –o pgm.out
```

```
g++ -Wall –g programme1.cpp programme2.cpp –o pgm.out
```

Structure générale d'un programme c++

```
#include <bibliothèque>
#include ”bibliothèque”

int main()
{
   instruction1;
   instruction2;
   instruction3;
   …

   return 0;
}
```

```
#include <bibliothèque>
#include ”bibliothèque”

int main() {
   instruction1;
   instruction2;
   instruction3;
   …

   return 0;
}
```

```
int main()
{
}
```

```
int main(int argc, char** argv)
{
}
```

```
int main(int argc, char* argv[])
{
}
```

Exécuter un programme sous linux

./pgm.out arg1 arg2 …

Commentaires ?

Un paradigme primordial en programmation.

Doit être significatif, bien rédigé, ni trop court ni trop long.

Comprend deux types:

- **Multi-lignes /* comment */**
- **Seul ligne // comment**

```
/**
Utilisé généralement
Pour faire une
documentation
*/
```

```
/* peut être utilisé pour une seule ligne */
```

```
// peut être utilisé
// pour plusieurs lignes
```

Entrées et sorties

La célèbre fonction printf du C est remplacée par cout

Syntaxe: `cout << expression;`

```
/**
 Un exemple de programme simple qui affiche
 une phrase (un avertissement).
*/

#include <iostream>

int main()
{
    std::cout<<"La température du système est élevée";

    return 0;
}
```

```
/**
 Un exemple de programme simple qui affiche
 deux phrases (un avertissement).
*/

#include <iostream>

int main()
{
    std::cout<<"La température du système est élevée ! \n
                    L'activation du ventilateur est cours…";

    return 0;
}
```

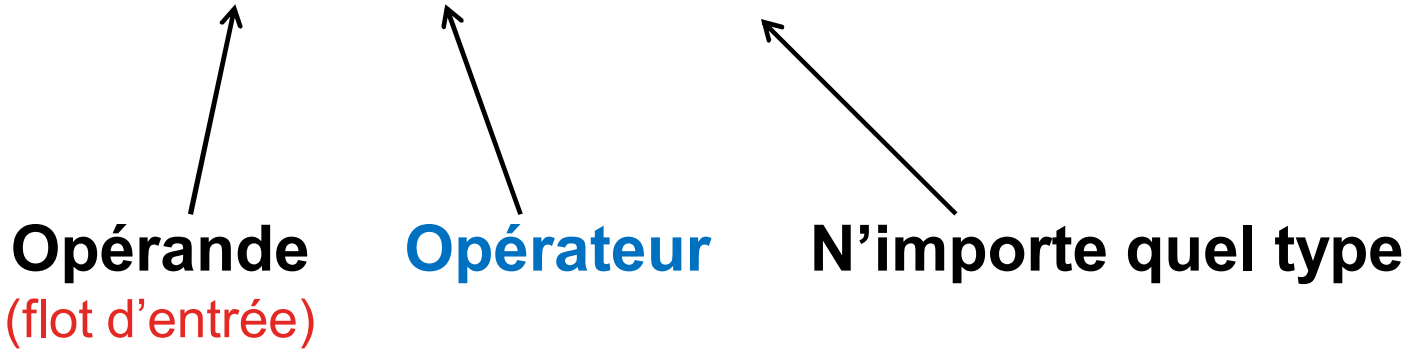
```
/**
 Un exemple de programme simple qui affiche
 deux phrases (un avertissement).
*/

#include <iostream>

int main()
{
    std::cout<<"La température du système est élevée !"<<
                    std::endl<<"L'activation du ventilateur est en
                    cours…"<< std::endl;

    return 0;
}
```

```
/**
 Un exemple de programme simple qui affiche
 deux phrases (un avertissement).
*/

#include <iostream>
using namespace std;

int main()
{
    cout<<"La température du système est élevée !"<<endl<<
          "L'activation du ventilateur est cours…"<<endl;

    return 0;
}
```

La fonction printf du C peut être utilisée en C++ en incluant la bibliothèque dédiée (stdio.h)

La fonction d'entrée scanf du C est remplacée par cin

Syntaxe: cin >> variable;



Pas de référence et pas de spécification du format

```
/**
 Un exemple de programme simple qui lit une valeur au clavier.
*/

#include <iostream>
using namespace std;

int main()
{
    cout<<"Tapez le numéro du capteur à utiliser : ";
    int no_capteur;
    cin>>no_capteur;
    cout<<"Le capteur #"<<no_capteur<<" est choisi !"<<endl;

    return 0;
}
```

```
/**
 Un exemple de programme simple qui lit une valeur au clavier.
*/

#include <iostream>
using namespace std;

int main()
{
    cout<<"Tapez le numéro du capteur à utiliser : ";
    int no_capteur;
    cin>>no_capteur;
    cout<<"Tapez 'M' pour le mode Manuel ou 'A' pour le mode auto : ";
    char mode;
    cin>>mode;
    return 0;
}
```

Quiz 1 (erreur ?)

```
#include <iostream>

int main()
{
    cout<<”Tapez le numéro du capteur à utiliser : ”;
    int no_capteur;
    cin>>no_capteur;

    return 0;
}
```

Quiz 2 (erreur ?)

```
#include <iostream>
using namespace std;

int main()
{
    cout<<”Tapez le numéro du capteur à utiliser : ”;
    int no_capteur
    cin>>no_capteur;

    return 0;
}
```

Quiz 3 (erreur ?)

```
#include <iostream>
using namespace std;

int main()
{
    cout<”Tapez le numéro du capteur à utiliser : ”;
    int no_capteur;
    cin>>no_capteur;

    return 0;
}
```

Quiz 4 (erreur ?)

```
#include <iostrem>
using namespace std;

int main()
{
    cout<<”Tapez le numéro du capteur à utiliser : ”;
    int no_capteur;
    cin>>no_capteur;

    return 0;
}
```

Variables

Une variable est un espace mémoire reconnu par une adresse physique dans le système et un nom défini par l'utilisateur dans le code source.

Syntaxe : type nom;

Nom ≠ Mot clé réservé, contenant des caractères spéciaux, commençant par un numéro.

Exemple: int variable;

Les noms composés doivent être séparés par _ (tiret bas) ou par des majuscules.

NB: on privilégie la première méthode, car la deuxième est réservée aux fonctions.

Exemple: int nom_compose;
int nomCompose;

La déclaration de plusieurs variables se fait sur la même ligne (séparées par une virgule) ou indépendamment.

Exemple: `int variable_1, variable_2, variable_3;`

```
int variable_1;
int variable_2;
int variable_3;
```

NB: on privilégie la seconde méthode pour une bonne lisibilité du code source.

- On peut rendre le contenu de la variable inchangeable (constant) en ajoutant le mot const.
- On lui donne une valeur fixe dès l'initialisation.
- Si on veut changer la valeur, on obtient une erreur.

Exemple: const int variable = 10;
const int variable(10);

Une autre alternative de const consiste à utiliser le préprocesseur #define (sans point virgule et sans égale).

- ***C'est plus pratique quand il s'agit d'un paramètre***
- ***Il est recommandé de l'écrire en majuscule pour le distinguer***

Syntaxe: #define nom_majuscule valeur

Exemple: #define MA_CONSTANTE 100

Il y a trois types de variables :

- ✓**Variable locale (interne à un bloc)**
- ✓**Variable globale (externe d'un bloc)**
- ✓**Variable externe (externe du programme)**

Parfois il est difficile de gérer les variables globales dans les grands projets.

Types

Il y a deux catégories de types :

- ✓ **primitifs**
- ✓ **objets (types composés)**

Quelques types primitifs couramment utilisés:

int, long, long int (4 bytes)
Short, short int (2 bytes)
char (1 byte)
bool (1 bit)
long long (8 bytes)

double (8 bytes)
float (4 bytes)

Pour définir un nouveau type on utilise le mot clé typedef

On peut même créer des types travaillants à l'ordre de bits

(on en parlera plus tard)

- Il existe également le type énumération pour donner des noms aux valeurs numériques (ou caractères).

- Par défaut, les valeurs commence de 0 (ordre croissant).

- Les noms sont séparés par des espaces.

- Les variables prennent une des valeurs définies.

Syntaxe: enum nom_type {nom1, nom2, nom3};

Exemple: enum protocole_interfaçage {
SPI,
I2C,
UART,
CAN,
USB
};

Par défaut: SPI=0, I2C=1, UART=2, CAN=3 et USB=4

On peut donner des valeurs quelconques.

Exemple: `enum protocole_interfaçage {`

```
    SPI    = 3,
    I2C    = 2,
    UART = 10,
    CAN   = 7,
    USB   = 1
};
```

Un exemple de déclaration d'une variable de type enum:

```
enum protocole_interfaçage {
                                SPI    = 3,
                                I2C    = 2,
                                UART = 10,
                                CAN    = 7,
                                USB    = 1
                            };
protocle_interfaçage protocole = SPI;

                    ou

protocle_interfaçage protocole = 3;
```

Opérateurs arithmétiques

Opérateurs arithmétiques standards (5 opérateurs):

+ - * / =

Opérateur Modulo (reste de division): %

```
Exemple: int x = 10 + 15; // la valeur de x sera 25
         int y = x  + 3;  // la valeur de y sera 28
         int z = x  + y;  // la valeur de x sera 53

         int r = z  % x;  // la valeur de r sera 3
```

Opérateurs d'incrémentation et décrémentation: ++ et --

i++ et i-- (on incrémente et décrémente la valeur de i mais sera changée dans l'instruction suivante)

++i et --i (on incrémente et décrémente la valeur de i immédiatement)

Exemple:

```
int x = 1; // x = 1
x++;       // x = 1
int y = 8; // y = 8 et x = 2
```

Exemple:

```
int x = 1; // x = 1
++x;       // x = 2
int y = 8; // y = 8 et x = 2
```

Opérateur arithmétique spécial : ( )

- ✓Regrouper des opérations
- ✓Prioritiser des opérations

Exemple 1:

```
int x = 1; // x = 1
x++;       // x = 1
int y = 8; // y = 8 et x = 2
int z = (x + y) * x; // z = 20
```

Exemple 2:

```
int x = 1; // x = 1
x++;       // x = 1
int y = 8; // y = 8 et x = 2
int z = x + y * x; // z = 18
```

Priorité des opérateurs arithmétiques :

1 ( )
2 / * %
3 + -
4 =

Astuces:

a = b = c = d = e = f = … = 10; (affectation multiple)

x = x + 5; --> x += 5;
x = x * 5; --> x *= 5;
x = x / 5; --> x /= 5;
x = x - 5; --> x -= 5;
x = x % 5; --> x %= 5;

Conversion de types (*type casting*)

La conversion implicite des types primitifs (variable scalaire) se fait automatiquement lors de l'affectation.

Exemple:

```
int x = 1; // x = 1
float y = x; // y = 1.000000

float a = 5.893625; // a = 5.893625
int b = a; // b = 5

char c = 'a'; // c = 'a'
int d = c; // d = 97 (code ASCII)
```

Quiz
(résultat affiché ?)

```
#include <iostream>
using namespace std;

int main()
{
    int x = 1;
    Int y = 2;

    float z = x / y;
    cout<<z<<endl;

    return 0;
}
```

Attention ! (un bug)

```
int x = 1; // x = 1
Int y = 2; // y = 2

float z = x / y; // z = 0

// car x et y sont des entiers et la
// division d'un entier sur un entier
// donne un entier
```

```
int x = 1; // x = 1
Int y = 2; // y = 2

float z = (float) x / y; // z = 0.500000

// il suffit de convertir x en réel
```

Quelques méthodes de conversion avancées:

- static_cast
- dynamic_cast
- const_cast
- reinterpret_cast

Syntaxe:

```
type_2 variable_1 = static_cast<type_2> variable_2;
```

Opérateurs relationnels *(comparaison)*

== (égale)

!= (différent)

> (supérieur)

< (inférieur)

>= (supérieur ou égale)

<= (inférieur ou égale)

Résultat d'une opération relationnelle est vrai ou faux (0 ou 1)

Exemple 1:

```
int x = 1; // x = 1
float y = x; // y = 1.000000

(x == y) // true
(x != y) // false
(x > y) // false
(x < y) // false
(x >= y) // true
(x <= y) // true
```

Exemple 2:

```
int x = 1; // x = 1
float y = x; // y = 1.000000

(x*2 == y) // false
(x != y+1) // true
(x > y%2) // false
(x < y%2) // false
(x >= y%2) // true
(x <= y%2) // true
```

Opérateurs logiques

&& (and minuscule) ---> et logique

|| (ou minuscule) ---> ou logique

! (not minuscule) ---> non logique

Exemple :

```
int x = 1;
float y = x;

(x == y && x > 3)
(x != y && y != 3)
```

Rappel

&&	vrai	faux
vrai	vrai	faux
faux	faux	faux

||	vrai	faux
vrai	vrai	vrai
faux	vrai	faux

Opérateurs binaires

& ---> et binaire

| ---> ou binaire (combinaison des params)

^ ---> XOR

~ ---> complément (inverseur)

>> ---> décalage vers la droite

<< ---> décalage vers la gauche

Exemple 1:

```
char x = 2;       // 00000010
char y = 5;       // 00000101
char z = x & y; // 00000000
```

Exemple 2:

```
char x = 2;       // 00000010
char y = 5;       // 00000101
char z = x | y;  // 00000111
```

Exemple 3:

```
char x = 3;       // 00000011
char y = 5;       // 00000101
char z = x ^ y; // 00000110
```

Exemple 4:

```
char x = 5;       // 00000101
char z = ~x;      // 11111 010
```

Exemple 5:

```
char x = 5;       // 00000101
char z = x >> 2; // 00000001
z = x >> 1;       // 00000010
```

Exemple 6:

```
char x = 5;       // 00000101
char z = x << 4; // 01010000
```

Quiz

Soit la variable PORTB contenant les états des pins (broches) d'un microcontrôleur. On veut connaitre l'état de la troisième pin.

Déroulement

- ✓Décalage de deux bits vers la droite.
- ✓Mettre à 0 tous les bits sauf le premier.
- ✓Effectuer une opération de AND avec 0x01 (00000001).
- ✓Afficher le résultat (soit 0 soit 1).

Code c++

```
#include <iostream>
using namespace std;

int main()
{
    unsigned char temp;
    temp  = PORTB >> 2;
    temp = temp & 0x01;
    // temp &= 0x01;

    cout<<temp<<endl;

    return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
    cout<<((PORTB >> 2) & 0x01)<<endl;

    return 0;
}
```

Prochain cours

Les instructions de contrôle, les tableaux et les fonctions.