POO & C++

Programmation Orientée Objet
&
Langage C++

UVSQ université PARIS-SACLAY
ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Olivier Snoeck

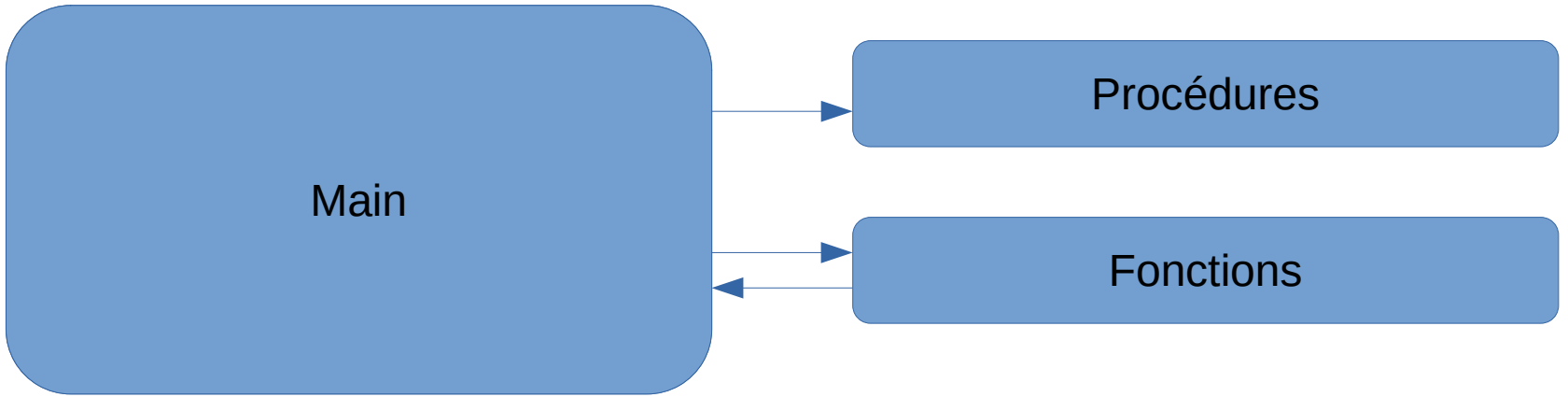
olivier@snoeck.fr

Les programmations

- La programmation procédurale
- La programmation orientée objet

Les programmations

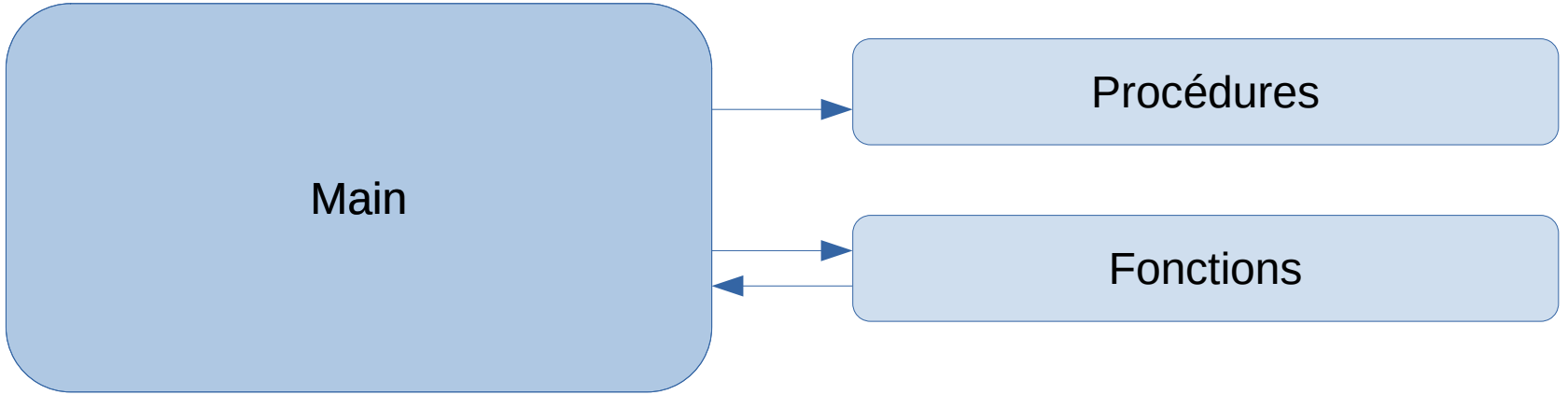
- La programmation procédurale
- La programmation orientée objet



Les programmations

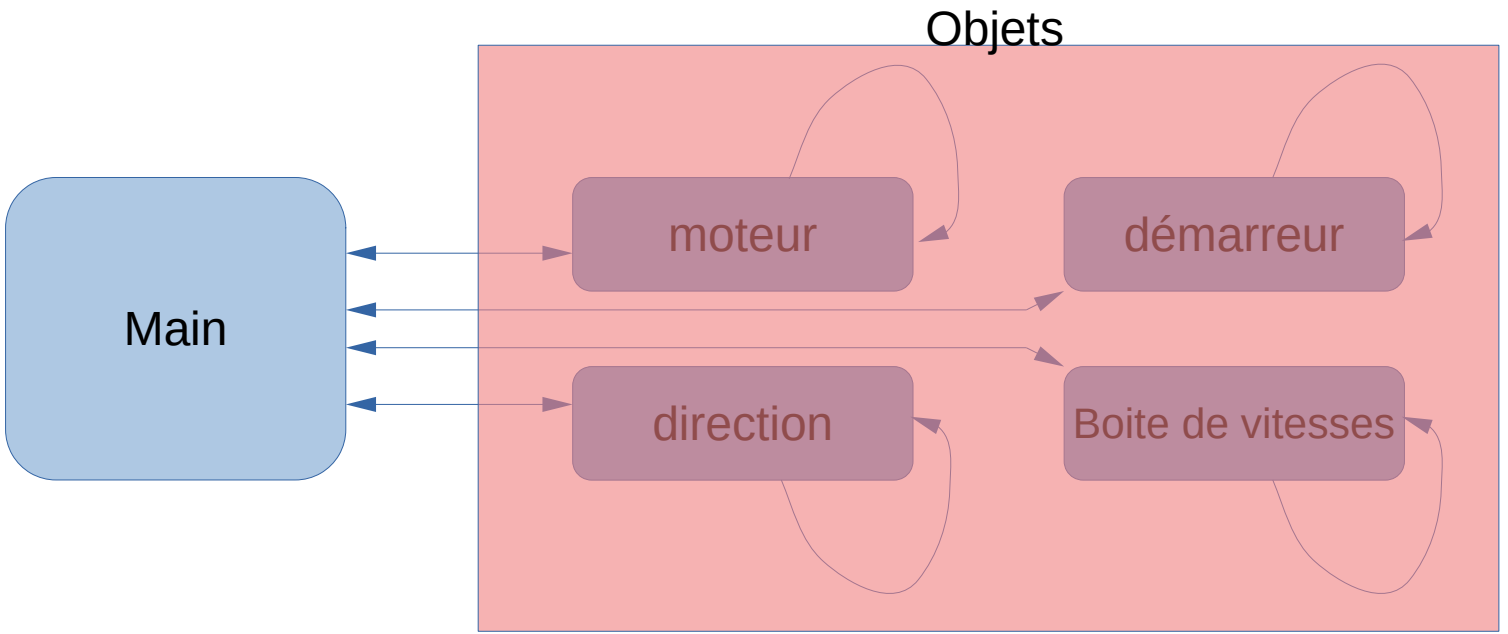
- La programmation procédurale
- La programmation orientée objet

La programmation procédurale est un paradigme de programmation qui gère des actions, de la logique, des événements.



Les programmations

- La programmation procédurale
- La programmation orientée objet

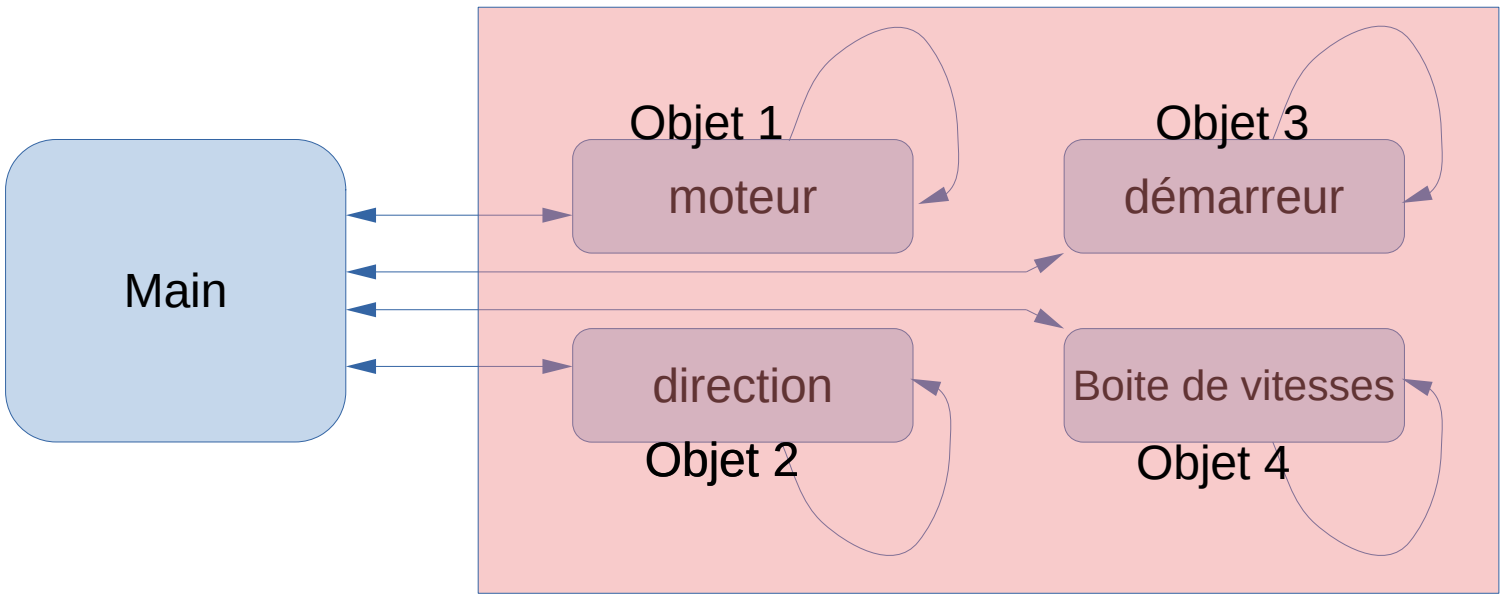


Les programmations

- La programmation procédurale

- La programmation orientée objet

La POO est un paradigme de programmation qui gère des briques logicielles



POO

La POO permet de :

- Développer un projet au sein d’une équipe,
- Pouvoir utiliser des outils collaboratifs,
- Faire évoluer le projet informatique,
- Sécuriser le projet informatique,
- Clarifier le rôle de chaque objet,
- Gérer des objets communs à plusieurs projets informatique,
- ….

Un exemple (main.cpp dans le répertoire Sources)

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout << "Hello world!" << endl;
8       return 0;
9   }
10
```

Directive de pré processeur (travail avant le compilateur)
https://www.cplusplus.com/reference/iostream/

Utilisation de l'espace des noms standard (lié à iostream
https://www.cplusplus.com/doc/oldtutorial/namespaces/

Prog. principal

endl : fin de ligne

cout pour afficher à l'écran
(permis car iostream + using namespace std)

Fin du « int main () » car cette fonction main(), du type int doit retourner un entier

Le même exemple (standard C++ 2011)

```
1  #include <iostream>
2
3
4  int main()
5  {
6      std::cout << "Hello world!" << std::endl;
7      return 0;
8  }
9
```

Directive de pré processeur (travail avant le compilateur)
https://www.cplusplus.com/reference/iostream/

Prog. principal

```
cours1_exemple1
Hello world!

Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

std ::endl : fin de ligne

std ::cout pour afficher à l'écran
(permis car iostream + using namespace std)

Objet : https://www.cplusplus.com/reference/iostream/cout/

Fin du « int main () » car cette fonction main(), du type int doit retourner un entier

La syntaxe du langage

Un langage informatique est composé de :

- Mots-clés :
  👇 Constituent le vocabulaire du langage (exemple : main, int, cout,...)
- Syntaxe : structures et règles
  👇 La « grammaire » du langage
  (= la forme requise des instructions)
- Conventions
  👇 Constituent des règles de notations adoptées par tous les programmeurs

C++

Les variables

Les variables

- Une variable est un endroit de la mémoire auquel on a donné un nom de sorte que l'on puisse y faire facilement référence dans le programme
- Créer une variable implique de lui donner :
  un nom + une valeur + un type
- La valeur d'une variable peut changer (varier) au cours du temps et de l'exécution du programme.
  si elle ne change pas : « const int » par exemple
  const double PI = 3.1415926535897 ;
  #define PI = 3.1415926535897 ;
  int nombre ;
  unsigned int mon_nombre = 1 ;
- Par contre son type ne change pas !
- Convention : 1ère lettre en minuscule.
- Autre convention : 1ère lettre en minuscule qui est le type : int iNombre ;

Les variables

[source]

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$-3.4*10^{-38}$ à $3.4*10^{38}$
double	Flottant double	8	$-1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant double long	10	$-3.4*10^{-4932}$ à $3.4*10^{4932}$
bool	Booléen	Même taille que le type *int*, parfois *1* sur quelques compilateurs	Prend deux valeurs : '*true*' et '*false*' mais une conversion implicite (valant 0 ou 1) est faite par le compilateur lorsque l'on affecte un entier (en réalité toute autre valeur que 0 est considérée comme égale à *True*).

Les variables
Les conversions de types de données

Conversion implicite :
int x ;
X = 8.32 ;
X contiendra 8

Conversion explicite :
int x ;
X = int(8.32) ;
X contiendra 8

int x ;
X = (int)8.32 ;
X contiendra 8

[*source*]

Les variables
La portée d'une variable

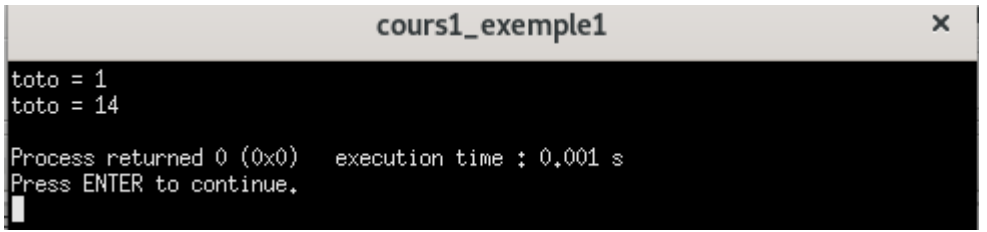
Une variable peut-être accessible de n'importe où (on parle de variable globale) ou seulement dans une fonction (on parle de variable locale).
Il existe l'opérateur de résolution de portée **::** qui permet d'utiliser la variable globale lorsqu'elle a le même nom que celui de la variable locale.

Exemple :

```
int toto = 13 ;  // création et initialisation de la variable globale

int main()
{
      int toto = 1 ; // création et initialisation de la variable locale

       toto = toto +::toto ; // on ajoute la variable globale à la variable locale...
}
```

Les variables
La portée d'une variable

```cpp
#include <iostream>
int toto = 13;

int main()
{
    int toto = 1;
    std::cout << "toto = " << toto << std::endl;

    toto = toto + ::toto;
    std::cout << "toto = " << toto << std::endl;

    return 0;
}
```



La portée d’une variable :

C++ en ligne

https://www.programiz.com/cpp-programming/online-compiler/

```
#include <iostream>
int toto = 13;

int main() {
    // Write C++ code here
    int toto = 1;
    std::cout << "toto = " << toto << std::endl;
    toto = toto + ::toto;
    std::cout << "toto = " << toto << std::endl;
    return 0;
}
```

```
toto = 1
toto = 14
```

La portée d'une variable :

C++ en ligne

```
#include <iostream>
int toto=1;
/*interdit de faire ceci:
std::cout << "Saisissez la 1ère valeur: ";
std::cin >> toto;
*/

int main() {
    std::cout << "Saisissez la 1ère valeur: ";
    std::cin >> ::toto;
    int toto;
    std::cout << "Saisissez la 2nde valeur: " ;
    std::cin >> toto;
    std::cout << "toto = " << toto << std::endl;
    toto = toto + ::toto;
    std::cout << "toto = " << toto << std::endl;
    return 0;
}
```

```
Saisissez la 1ère valeur: 12
Saisissez la 2nde valeur: 5
toto = 5
toto = 17
```

C++

L’espace des noms

L’espace des noms

Permet d’éviter les conflits de variables

```
namespace toto
{
}
```

Exemple :

```
namespace test
{
    int i = 5 ;
}

int i = 8 ; // c’est une 2nde variable i mais qui est valable hors de l’espace de nom « test »

test::i = 7; //là, je viens de modifier le contenu de la variable de l’espace de nom « test »
```

L’espace des noms

Permet d’éviter les conflits de variables

```
#include <iostream>
using namespace std;
const int UneConstanteGlobale = 1;
int UneVariableGlobale;
namespace MonEspaceDeNom
{
    const int MaConstanteDePorteeNommee = 2;
    int MaVariableDePorteeNommee;
    int UneVariable;

}
int main()
{
    int UneVariable = 3;
    MonEspaceDeNom::MaVariableDePorteeNommee = UneConstanteGlobale;
    UneVariableGlobale = MonEspaceDeNom::MaConstanteDePorteeNommee;
    MonEspaceDeNom::UneVariable = 4;
    cout << "MonEspaceDeNom::MaVariableDePorteeNommee = " << MonEspaceDeNom::MaVariableDePorteeNommee <<endl;
    cout << "UneVariableGlobale = " << UneVariableGlobale << endl;
    cout << "UneVariable = " << UneVariable << endl;
    cout << "MonEspaceDeNom::UneVariable = " << MonEspaceDeNom::UneVariable << endl;
    return 0;
}
```

L’espace des noms

Permet d’éviter les conflits de variables

```cpp
#include <iostream>
using namespace std;
const int UneConstanteGlobale = 1;
int UneVariableGlobale;
namespace MonEspaceDeNom
{
    const int MaConstanteDePorteeNommee = 2;
    int MaVariableDePorteeNommee;
    int UneVariable;

}
int main()
{
    int UneVariable = 3;
    MonEspaceDeNom::MaVariableDePorteeNommee = UneConstanteGlobale;
    UneVariableGlobale = MonEspaceDeNom::MaConstanteDePorteeNommee;
    MonEspaceDeNom::UneVariable = 4;
    cout << "MonEspaceDeNom::MaVariableDePorteeNommee = " << MonEspaceDeNom::MaVariableDePorteeNommee
        <<endl;
    cout << "UneVariableGlobale = " << UneVariableGlobale << endl;
    cout << "UneVariable = " << UneVariable << endl;
    cout << "MonEspaceDeNom::UneVariable = " << MonEspaceDeNom::UneVariable << endl;
    return 0;
}
```

```
/tmp/pnjD8JfuWn.o
MonEspaceDeNom::MaVariableDePorteeNommee = 1
UneVariableGlobale = 2
UneVariable = 3
MonEspaceDeNom::UneVariable = 4
```

C++

Les opérateurs

Les opérateurs (incrémentation et décrémentation)

- int i = 5 ;
  i++ ; la valeur de i est maintenant de 6
- int i = 5 ;
  int j = ++i ; // j vaut 6 et i vaut 6
  int k = i++ ; // k vaut 7 et i vaut 6

i++ incrémente la variable i de 1 → l'expression vaut la valeur de i avant incrémentation.

++i incrémente la variable i de 1 → l'expression vaut la valeur de i après incrémentation.

Même remarque avec la décrémentation...

```
#include <iostream>

int i = 5;
int j = ++i; // j vaut 6 et i vaut 6
int k = i++; // k vaut 6 et i vaut 7

int main()
{
    std::cout << " i = "<<i;
    std::cout << " j = " <<j;
    std::cout << " k = " <<k;
    return 0;
}
```

Les opérateurs de calcul

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	x*3	21
/	opérateur de division	Divise deux valeurs	x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Le modulo : X%3 = 1 si x = 7

Les opérateurs d’assignation

Ces opérateurs permettent de simplifier des opérations.
Exemple : x=x+2 devient : x+=2
Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après..

Opérateur	
+=	additionne deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
*=	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable

Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
== **A ne pas confondre avec le signe d'affectation (=) !**	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne 1 si x est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne 1 si x est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne 1 si x est inférieur ou égal à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne 1 si x est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne 1 si x est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne 1 si x est différent de 3, sinon 0

Les opérateurs logiques

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

Opérateur	Dénomination	Effet	Syntaxe	Résultat
&	ET bit-à-bit	Retourne 1 si les deux bits de même poids sont à 1	9 & 12 (1001 & 1100)	8 (1000)
|	OU bit-à-bit	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)	9 | 12 (1001 | 1100)	13 (1101)
^	OU exclusif bit-à-bit	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)	9 ^ 12 (1001 ^ 1100)	5 (0101)

~ est le complément à 1

Opérateur	Dénomination	Effet	Syntaxe	Résultat
<<	Rotation à gauche	Décale les bits vers la gauche (multiplie par 2 à chaque décalage). Les zéros qui sortent à gauche sont perdus, tandis que des zéros sont insérés à droite	6 << 1 (110 << 1)	12 (1100)
>>	Rotation à droite avec conservation du signe	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que le bit non nul de poids plus fort est recopié à gauche	6 >> 1 (0110 >> 1)	3 (0011)
>>>	Rotation à droite avec remplissage de zéros	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que des zéros sont insérés à gauche	6 >>> 1 (0110 >>> 1)	3 (0011)

Les opérateurs logiques

quelques exemples

```
unsigned short i = 0x0FF0 ;
unsigned short k = ~i ;
```
→ k = 0xF00F

```
bool a = true ;
bool c =!b ;
```
→ c = false

```
int a = 3 ;
int b ;
b=a ;
```
→ b = 3

Les opérateurs logiques

quelques exemples

unsigned short i = 0x0FF0 ;
unsigned short a = i << 4 ; → a = 0xFF00

! attention aux signes !

i = 0b 0000 1111 1111 0000

unsigned short i = 0x0FF0 ;
unsigned short b = i >> 4 ; → b = 0x00FF

! attention aux signes !

Les opérateurs logiques

Exercice 1

```
#include <iostream>

int main()
{
int a=4, b=3, c=2, d=1;
std::cout << a+b << " , " << a-b << " , " << a*b << std::endl;
std::cout << a/b << " , " << b/a << " , " << a%b << std::endl;

a += 10;
b *= 3;
++b;

std::cout << (a+b)*c - d*(a-b);
return 0;
}
```

Qu’obtiens-je à l’écran ?

Les opérateurs logiques

Exercice 1 (corrigé)

main.cpp

Run

Output

```cpp
#include <iostream>

int main()
{
int a=4, b=3, c=2, d=1;
std::cout << a+b << " , " << a-b << " , " << a*b << std::endl;
std::cout << a/b << " , " << b/a << " , " << a%b << std::endl;

a += 10;
b *= 3;
++b;

std::cout << (a+b)*c - d*(a-b);
return 0;
}
```

```
/tmp/mIqcPEi8xO.o
7 , 1 , 12
1 , 0 , 1
44
```

Les opérateurs logiques

Exercice 2

```
#include <iostream>

int main()
{
unsigned short a=10, b=4;
a |=5;
std::cout << "a = " << a << std::endl;

b = (a&3)|(a^4);
std::cout << "b = " << b << std::endl;

unsigned short c = (a>b)?10:2; //opérateur ternaire ;-) si a>b alors c=10 sinon c=2
std::cout << "Ligne 13: c = " << c << std::endl;

c <<= (a-b);
std::cout << "Ligne 16: c = " << c << std::endl;

return 0;}
```

Qu’obtiens-je à l’écran ?

Les opérateurs logiques

Exercice 2 (corrigé)

main.cpp

Run

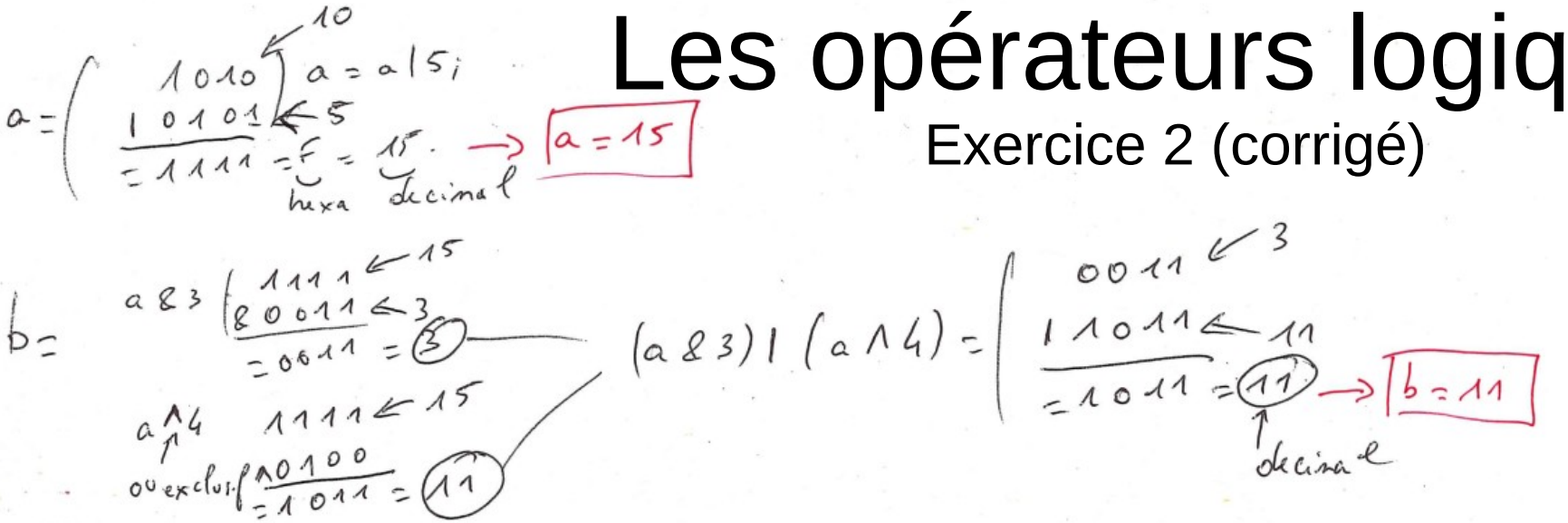
Output

```cpp
#include <iostream>

int main()
{
unsigned short a=10, b=4;
a |=5;
std::cout << "a = " << a << std::endl;

b = (a&3)|(a^4);
std::cout << "b = " << b << std::endl;

unsigned short c = (a>b)?10:2; //opérateur ternaire ;-) si a>b alors c=10 sinon c=2
std::cout << "Ligne 13: c = " << c << std::endl;

c <<= (a-b);
std::cout << "Ligne 16: c = " << c << std::endl;

return 0;
}
```

```
/tmp/mIqcPEi8xO.o
a = 15
b = 11
Ligne 13: c = 10
Ligne 16: c = 160
```

Les opérateurs logiques

Exercice 2 (corrigé)



ligne 13 : a > b : ou : donc c = 10 **c = 10 ligne 13.**

c <<= (a - b) → a - b = 4

↳ c = c << 4

10 = 0000 0000 0000 1010 (unsigned short sur 2 octets).

0000 0000 1010 0000 = $2^5 + 2^7$ = 32 + 128 = 160

c = 160 ligne 16

C++

Les structures de contrôles

Les structures de contrôle

- Alternatives
  if (condition)
  { ;}

Les structures de contrôle

- Alternatives
  if (condition)
  { ;}
  else
  { ;}

Les structures de contrôle

- Alternatives
  if (condition 1)
  { ;}
  else if (condition 2)
  { ;}
  …

Les structures de contrôle

- Alternatives
  if (condition 1)
  { ;}
  else if (condition 2)
  { ;}
  …
  else
  { ;}

Les structures de contrôle

- Structure de sélection

```
switch (a)
{
case 0 :
        …. ;
        break ;
case 1 :
        …. ;
        break ;
default :
        … ;
}
```

Les structures de contrôle

- Itération (boucle)

```
while (condition)
{
        … ;
}
```

Les structures de contrôle

- Itération (boucle)

```
do
{
        … ;
}
while (condition) ;
```

Les structures de contrôle

- Itération (boucle)
  for (initialisation;condition ; post-action
  {
  … ;
  }

- for (int i=1 ; i<=10 ; i++)
  {
  somme += i ;
  }

Les structures de contrôle

main.cpp

Run

Output

```
1
2  // Online C++ compiler to run C++ program online
3  #include <iostream>
4
5  unsigned int somme =0;
6
7  int main() {
8
9      for ( int i=1;i<=10; i++)
10 {
11       somme += i;
12       std::cout << "i = " << i << std::endl;
13       std::cout << "Somme = " << somme << std::endl;
14 }
15
16
17     return 0;
18 }
```

```
/tmp/RpCy02qsU3.o
i = 1
Somme = 1
i = 2
Somme = 3
i = 3
Somme = 6
i = 4
Somme = 10
i = 5
Somme = 15
i = 6
Somme = 21
i = 7
Somme = 28
i = 8
Somme = 36
i = 9
Somme = 45
i = 10
Somme = 55
```

Les structures de contrôle

Il y a donc des choix à faire pour plus de lisibilité, une maintenance plus aisée...

Les structures de contrôle

Sortie d’une boucle :
1ère solution : break ;
2nde solution : goto

Exercices

1) Écrire un programme permettant de calculer la somme des 50 premiers entiers,
2) Écrire un code permettant d'afficher les multiples de 6 inférieurs à 1000.
3) Écrire un code permettant de déterminer si un nombre saisi par l'utilisateur est premier.

Comparons vos solutions….

C++

Les fonctions

Les fonctions en C++

En programmation, une fonction joue un rôle. Ce rôle sera appelé plusieurs fois, ou ce rôle est très spécifique d'où la nécessité de la mettre en dehors du programme principal.

Une fonction est un sous-programme, une fonction ou une procédure...

Quel que soit le type, sa syntaxe est la suivante :

```
<type de retour><nom de la fonction>( <paramètres> )
{
	…. ;
	return <valeur de retour> ;
}
```

Les fonctions en C++

main.cpp

Run

Output

```
1
2  // Online C++ compiler to run C++ program online
3  #include <iostream>
4
5  unsigned int nombre =0;
6
7  bool EstPair (int n)
8  {
9      return n%2;
10 }
11
12
13
14 int main() {
15     std::cout << "Donnez un chiffre (ou un nombre): ";
16     std::cin >> nombre;
17     if (!EstPair(nombre))
18     {
19         std::cout << "Le nombre saisi est pair" << std::endl;
20     }
21     else
22     {
23         std::cout << "Le nombre saisi est impair" << std::endl;
24     }
25     return 0;
26 }
```

```
/tmp/RpCyO2qsU3.o
Donnez un chiffre (ou un nombre): 201
Le nombre saisi est impair
```

Les fonctions en C++

main.cpp

Run

Output

```
1
2  // Online C++ compiler to run C++ program online
3  #include <iostream>
4
5  unsigned int nombre =0;
6
7  bool EstPair (int n)
8  {
9      return n%2;
10 }
11
12
13
14 int main() {
15     std::cout << "Donnez un chiffre (ou un nombre): ";
16     std::cin >> nombre;
17     if (!EstPair(nombre))
18     {
19         std::cout << "Le nombre saisi est pair" << std::endl;
20     }
21     else
22     {
23         std::cout << "Le nombre saisi est impair" << std::endl;
24     }
25     return 0;
26 }
```

```
/tmp/RpCyO2qsU3.o
Donnez un chiffre (ou un nombre): 201
Le nombre saisi est impair
```

Passage de paramètre d'entrée

Une fonction renvoie un résultat (ici, un booleen)

Les fonctions en C++

Exemple

main.cpp

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>

unsigned int nombre =0;

bool EstPair (int n)
{
    return n%2;
}

void AfficheResultat(bool toto)
{
    if (toto)
    {
        std::cout << "Le nombre saisi est pair" << std::endl;
    }
    else
    {
        std::cout << "Le nombre saisi est impair" << std::endl;
    }
}

int main() {
    std::cout << "Donnez un chiffre (ou un nombre): ";
    std::cin >> nombre;
    AfficheResultat(!EstPair(nombre));

    return 0;
}
```

Run

Output

```
/tmp/RpCyO2qsU3.o
Donnez un chiffre (ou un nombre): 121
Le nombre saisi est impair
```

Passage de paramètre d'entrée
(qui sera perdu une fois la fonction ou la procédure terminée)

Une fonction renvoie un résultat (ici, un booleen)

Une procédure ne renvoie aucune valeur.
→ Le « return » n'est pas obligatoire
→ void signifie vide : pas de retour

Les fonctions en C++

2 types de passage de paramètre d'entrée :

1er mode

```
int Incrementer (int nombre)
{
      return ++nombre ;
}


 int main()
 {
 ….
 Toto = Incrementer(Toto) ;
 }
```

2nd mode

```
int Incrementer (int& nombre)
{
      return ++nombre ;
}


 int main()
 {
 ….
 Incrementer(Toto) ;
 }
```

Passage par référence : la valeur de la variable Toto est modifiée directement par la fonction Incrémenter

Les fonctions en C++

2 types de passage de paramètre d'entrée :

Passage par référence

Ce sera la même ressource mémoire → gain de place

main.cpp

Run

```cpp
#include <iostream>

int toto = 0, tata = 0 ;

int Incrementer (int nombre)
{
    return ++nombre;
}

int IncrementerPR (int& nombre)
{
    return ++nombre;
}

int main()
{
    std::cout << "Saisissez un nombre" << std::endl;
    std::cin >> toto;
    std::cout << "toto1 = " << toto << std::endl;
    tata = Incrementer(toto);
    std::cout << "toto2 = " << toto << std::endl;
    std::cout << "toto3 = " << Incrementer(toto) << std::endl;
    std::cout << "tata = " << tata << std::endl;
    IncrementerPR(toto);
    std::cout << "toto4 = " << IncrementerPR(toto) << std::endl;
    std::cout << "toto5 = " << toto << std::endl;    //la variable initiale a été modifiée

return 0;
}
```

Output

```
/tmp/mIqcPEi8x0.o
Saisissez un nombre
12
toto1 = 12
toto2 = 12
toto3 = 13
tata = 13
toto4 = 14
toto5 = 14
```

Exercice 4

L’algorithme emprunt montre comment utiliser le pseudo-langage pour écrire un programme. Le calcul de la mensualité se décompose en trois calculs plus simples.
Calcul1 = Capital × TauxMensuel
Calcul2 = (1+TauxMensuel) x NbMois
Calcul3 = (1+T) x NbMois –1

L’algorithme est alors :

Exercice 4

```
Programme emprunt
Déclarations
      Variables Mensualité, Capital en Réel
      Variables TauxMensuel, TauxAnnuel en Réel
      Variables Calcul1,Calcul2,Calcul3 en Réel
      Variables NbAn,NbMois en Entier
Début
     //Saisiedesdonnéesinitiales
     Écrire(“Montantducapitalemprunté :”)
     Lire(Capital)
     Écrire(“Nombred’années :”)
     Lire(NbAn)
     Écrire(“TauxAnnuel(exemple5,5%):”)
     Lire(TauxAnnuel)
     //Calculs
     NbMois ← NbAn*12
     TauxMensuel ← (TauxAnnuel/100)/12
     calcul1 ← Capital*TauxMensuel
     calcul2 ← (1+TauxMensuel)**NbMois
     calcul3 ← calcul2-1
     Mensualité ← calcul1*(calcul2/calcul3)
     //Affichagedurésultat
     Écrire(“Mensualité:”,Mensualité)
Fin
```

Réalisez ce programme en utilisant des fonctions, des procédures, des paramètres d’entrée, des passages par référence, par groupe !

Exercice 5

L'ordinateur choisit un nombre entre 1 et 100. L'utilisateur doit deviner ce nombre en un minimum de coups. Voici le début de la solution...

```
//fichier utils.h
#ifndef UTILS_H
#define UTILS_H
#include <string>
namespace ag{
    extern void Afficher (const std::string& message);
    extern void SaisirNombre (int& nombre);
    extern int TireNombre (int mini, int maxi);
    extern void Saisir (char& car);
}
#endif
```

```
//fichier utils.cpp
#include <iostream>
#include <string>
#include "utils.h"

namespace ag{
    void Afficher (const std::string& message)
    {
        std::cout << message << std::endl;
    }
    void SaisirNombre (int& nombre)
    {
        std::cin >> nombre;
    }
    int TireNombre (int mini, int maxi)
    {
        return (std::random(maxi-mini) + mini);
    }
    void Saisir (char& car);
    {
        std::cin >> car;    }
}
```

C++

Les variables abstraites

Les types abstraits

Nous avons vu les types de variables intégrés au langage C++ (int, char, double,...)

Le C++ permet de définir ses propres types de variables pour :

- Les dates par exemple,
- Les jours de semaine par exemple,
- Les couleurs,
- …..

Les types abstraits

Les alias de types

Nous avons vu les types de variables intégrés au langage C++ (int, char, double,...)

Le C++ permet de définir ses propres types de variables pour :

- Les dates par exemple,
- Les jours de semaine par exemple,
- Les couleurs,
- …..

Alias :Adverbe : Autrement appelé (de tel ou tel nom)
Nom masculin : Nom d'emprunt

Les types abstraits

Les alias de types *(typedef)*

```
typedef unsigned char byte ;

byte Mot1 = 0xF2 ;
if (Mot1 == 10)….
```

Dans l'exemple ci-contre, nous avons créé un alias de type « unsigned char ». L'alias de type « unsigned char » est donc « byte » (octet :-)) que je peux utiliser pour créer une nouvelle variable de ce même alias de type.
Avec cet alias, on peut faire des opérations binaires, des compléments à 1,….

Version c++ 2003

Alias :Adverbe : Autrement appelé (de tel ou tel nom)
Nom masculin : Nom d'emprunt

Les types abstraits

Les alias de types *(using)*

```
using byte = unsigned char;

byte Mot1 = 0xF2 ;
if (Mot1 == 10)….
```

Dans l'exemple ci-contre, nous avons créé un alias de type « unsigned char ». L'alias de type « unsigned char » est donc « byte » (octet :-)) que je peux utiliser pour créer une nouvelle variable de ce même alias de type.
Avec cet alias, on peut faire des opérations binaires, des compléments à 1,….

Version c++ 2011

Alias :Adverbe : Autrement appelé (de tel ou tel nom)
Nom masculin : Nom d'emprunt

Les types abstraits

Les alias de types *(using)*

```
#include <iostream>
#include <string>

int main() {
using byte = unsigned char;

byte Mot1 = 0xF2;
for (int i=0;i<=0xF2; i++)
{
    std::cout << "Mot1 en décimal= " << std::dec << i ;
    //int toto = i & 0xF0
    std::cout << " qui donne en héxadécimal = " << std::hex << i << std::endl;
    if (i == Mot1)
    {
        std::cout << " Opération réussie " << std::endl;
    }
}

    return 0;
}
```

Version c++ 2011

main.cpp

Run

```cpp

// Online C++ compiler to run C++ program online
#include <iostream>
#include <string>

int main() {
using byte = unsigned char;

byte Mot1 = 0xF2;
for (int i=0;i<=0xF2; i++)
{
    std::cout << "Mot1 en décimal= " << std::dec << i ;
    //int toto = i & 0xF0
    std::cout << " qui donne en héxadécimal = " << std::hex << i << std::endl;
    if (i == Mot1)
    {
        std::cout << " Opération réussie " << std::endl;
    }
}

    return 0;
}

```

Output

```
Mot1 en décimal= 211 qui donne en héxadécimal = d3
Mot1 en décimal= 212 qui donne en héxadécimal = d4
Mot1 en décimal= 213 qui donne en héxadécimal = d5
Mot1 en décimal= 214 qui donne en héxadécimal = d6
Mot1 en décimal= 215 qui donne en héxadécimal = d7
Mot1 en décimal= 216 qui donne en héxadécimal = d8
Mot1 en décimal= 217 qui donne en héxadécimal = d9
Mot1 en décimal= 218 qui donne en héxadécimal = da
Mot1 en décimal= 219 qui donne en héxadécimal = db
Mot1 en décimal= 220 qui donne en héxadécimal = dc
Mot1 en décimal= 221 qui donne en héxadécimal = dd
Mot1 en décimal= 222 qui donne en héxadécimal = de
Mot1 en décimal= 223 qui donne en héxadécimal = df
Mot1 en décimal= 224 qui donne en héxadécimal = e0
Mot1 en décimal= 225 qui donne en héxadécimal = e1
Mot1 en décimal= 226 qui donne en héxadécimal = e2
Mot1 en décimal= 227 qui donne en héxadécimal = e3
Mot1 en décimal= 228 qui donne en héxadécimal = e4
Mot1 en décimal= 229 qui donne en héxadécimal = e5
Mot1 en décimal= 230 qui donne en héxadécimal = e6
Mot1 en décimal= 231 qui donne en héxadécimal = e7
Mot1 en décimal= 232 qui donne en héxadécimal = e8
Mot1 en décimal= 233 qui donne en héxadécimal = e9
Mot1 en décimal= 234 qui donne en héxadécimal = ea
Mot1 en décimal= 235 qui donne en héxadécimal = eb
Mot1 en décimal= 236 qui donne en héxadécimal = ec
Mot1 en décimal= 237 qui donne en héxadécimal = ed
Mot1 en décimal= 238 qui donne en héxadécimal = ee
Mot1 en décimal= 239 qui donne en héxadécimal = ef
Mot1 en décimal= 240 qui donne en héxadécimal = f0
Mot1 en décimal= 241 qui donne en héxadécimal = f1
Mot1 en décimal= 242 qui donne en héxadécimal = f2
 Opération réussie
```

Les types abstraits

Les types énumérés

```
enum jour {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche} ;
// lundi vaut 0, mardi vaut 1, mercredi vaut 2,….

jour today = lundi ;

if (today == dimanche)
….
```

Cela revient à :

const int dimanche = 0;
const int lundi = 1;
const int mardi = 2;
etc...

```
#include <iostream>

int main() {

    enum jour {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche};

    jour today = lundi;
```

Les types abstraits

Les types énumérés

```
do
   {
      std::cout << "Nous sommes " << today << " il faut travailler! "<< std::endl;
      switch (today)
      {   case lundi:
        {
        today = mardi;
        break;
        }
        case mardi:
        {
        today = mercredi;
        break;
        }
        case mercredi:
        {
        today = jeudi;
        break;
        }
        case jeudi:
        {
        today = vendredi;
        break;
        }
        case vendredi:
        {
        today = samedi;
        break;
        }
        case samedi:
        {
        today = dimanche;
        break;
        }
        case dimanche:
        {
        today = lundi;
        break;
        }
      }
   }
   while (today != dimanche);
   std::cout << "Repos :-) " << std::endl;
   return 0;
}
```

C++

Les tableaux

Les tableaux

Un tableau est une collection indicée de variables de même type.

Forme de programmation : <type><nom du tableau>[<taille>] ;

Exemples :

```
int fraction[2] ;              // tableau de 2 entiers
char mot[10], s[256];          // tableaux de respectivement 10 et 256 caractères
double tab[3];                 // tableau de trois nombres réels
vache troupeau[1000];          // tableau de mille vache si le type est créé ...
```

Les tableaux

Un tableau est une collection indicée de variables de même type.

Forme de programmation : <type><nom du tableau>[<taille>] ;

```
int T[3] ={5, 10, 15};
char voyelle[6] ={’a’, ’e’, ’i’, ’o’, ’u’, ’y’};
int M[2][3] ={{1, 2, 3},{3, 4, 5}};   // ou  int M[2][3] ={1, 2, 3, 3, 4, 5};
```

Exemple : T[0] = 1 ;
→ T[3] ={1, 10, 15};

Les tableaux

Un tableau est une collection indicée de variables de même type.

Forme de programmation : <type><nom du tableau>[<taille>] ;

Autre exemple :

```
 int tabl[3][3] = {{4, 3, 6},{10, 0, 0},{-1, 5, 3}};

for(int y = 0; y < 3; y++)
{
    for(int x = 0; x < 3; x++)
    cout << tabl[x][y] << '\t';//’t’ permet une tabulation
    cout << endl;
}
```

Les tableaux

Un tableau est une collection indicée de variables de même type.

Forme de programmation : <type><nom du tableau>[<taille>] ;

main.cpp

Run

Output

```cpp
#include <iostream>

int main()
{
int tabl[3][3] = {{4, 3, 6},{10, 0, 0},{-1, 5, 3}};

for(int y = 0; y < 3; y++)
{
    for(int x = 0; x < 3; x++)
    std::cout << tabl[x][y] << '\t'; //'t' permet une tabulation
    std::cout << std::endl;
}
return 0;
}
```

```
/tmp/miLHsbmjgb.o
4   10  -1
3   0   5
6   0   3
```

C++

Les chaînes de caractères

Les chaînes de caractères

Une chaîne de caractères est un tableau, avec un caractère nul (\0) à la fin de la chaîne.

```
char nom[20], prenom[20];      // 19 caractères utiles
char adresse[3][40];           // trois lignes de 39 caractères utiles
char turlu[10] ={’t’, ’u’, ’t’, ’u’, ’\0’}; // ou :  char turlu[10] = "tutu";
```

```
prenom = "Antoine";            // illégal mais fonctionne
strcpy(prenom, "Antoine");  // correct
```

Les chaînes de caractères

Dans la bibliothèque standard du C++, il y a de nombreuses fonctions utilitaires sur les chaînes de caractères.
Parmi elles :
strlen(s)(dans cstring) : donne la longueur de la chaînes

strcpy(dest, source)(dans string.h) : recopie la chaîne **source** dans **dest**

strcmp(s1,s2)(dans string.h) : compare les chaînes **s1** et **s2** :
renvoie une valeur < 0, nulle ou > 0 selon que **s1** est inférieure, égale ou supérieure à **s2** pour l'ordre alphabétique.

gets(s)(dans stdio.h): lit une chaîne de caractères tapée au clavier, jusqu'au premier retour à la ligne (inclus) ; les caractères lus sont rangés dans la chaîne **s**, sauf le retour à la ligne qui y est remplacé par '\0'.

Les chaînes de caractères

Exercice de recherche de palindrome

```
//radar, rotor, kayak, été,
//ici, tôt, rêver, réifier, ressasser
//sont des palindromes

#include <iostream>
#include <cstring>



using std::cout;
using std::endl;
using std::cin;

int testPalindrome(char[],int,int,int);
```

```
int main()
{
        int e=0, r, rentre;
    const int elements=100;
        char tab[elements];



        cout<<"Veuillez saisir un mot et je vous dirai s'il s'agit d'un palindrome !"<<endl;
        cin>>tab;
    //2 façons pour connaitre la taille d'une chaîne de caracère
    //1ère façon:
          do     // Pour calculer la taille de la chaine entrée par l'utilisateur !
          {
                e++;
          }while(tab[e]!=0);
          cout << "table de " << e << " caractères" << endl;
    //2nde façon:
          e = strlen(tab);
          cout << "table de " << e << " caractères" << endl;


        r=testPalindrome(tab,elements,0,e); // Appel de la fonction ...


        if(r==0)
                cout<<"Ce n'est pas un palindrome !"<<endl;
        else if(r==1)
                cout<<"Il s'agit d'un palindrome !"<<endl;
        return 0;
}
```

Les chaînes de caractères

Exercice de recherche de palindrome

```
int testPalindrome(char mot[],int indices,int debut,int fin)
{
    int e=0;
    for (debut; debut<=fin/2; debut++, fin--)
        {
            cout << endl;
        cout<<mot[debut]<< " ?=? " <<mot[fin-1]<<endl;
        if (mot[debut]!=mot[fin-1])
            { // Toute la recursivité est là !
             cout<<"KO " <<endl;
             return 0;
         }
        else
            cout<<"OK"<<endl;
        }
    return 1;
}
```

```
int main()
{
        int e=0, r, rentre;
    const int elements=100;
        char tab[elements];



        cout<<"Veuillez saisir un mot et je vous dirai s'il s'agit d'un palindrome !"<<endl;
        cin>>tab;
    //2 façons pour connaitre la taille d'une chaîne de caracère
    //1ère façon:
        do     // Pour calculer la taille de la chaine entrée par l'utilisateur !
        {
                e++;
         }while(tab[e]!=0);
         cout << "table de " << e << " caractères" << endl;
    //2nde façon:
         e = strlen(tab);
         cout << "table de " << e << " caractères" << endl;


        r=testPalindrome(tab,elements,0,e); // Appel de la fonction ...


        if(r==0)
                cout<<"Ce n'est pas un palindrome !"<<endl;
        else if(r==1)
                cout<<"Il s'agit d'un palindrome !"<<endl;
        return 0;
}
```

Exercice 6

```
#include <iostream>

const double prixdulitre = 1.10;
int reserve = 10000;
double prix(int nb)
{
    return nb * prixdulitre;
}
int delivre(int nb)
{
    if (nb > reserve)
        return 0;
    reserve -= nb;
    return 1;
}
int main()
{
    int possible;
    do
    {
        int quantite;
        std::cout << "Bonjour.  Combien voulez-vous de litres de gazole ?  ";
        std::cin >> quantite;
        possible = delivre(quantite);
        if (possible)
            std::cout << "Cela fait " << prix(quantite) << " euros.\n";
    }
    while (possible);
    std::cout << "Plus assez de carburant.\n";
    return 0;}
```

Testez ce programme
Modifiez ce programme afin d'avoir :
1 fichier pompe.h avec toutes les déclarations :

```
const double prixdulitre = 1.10;
int reserve = 10000;
double prix(int nb) ;
int delivre(int nb) ;
```

1 fichier pompe.cpp avec les fonctions

Puis 1 fichier clients.cpp avec le main

```
// ------------------- fichier pompe.h ----------------------

const double prix_du_litre =            // déclaration du prix du litre de gazole (en euros)

double prix(int nb);                    // déclaration de la fonction prix
int delivre(int nb);                    // déclaration de la fonction delivre


// ------------------- fichier pompe.cpp --------------------

#include "pompe.h"                      // pour inclure les déclarations précédentes

int reserve = 10000;                    // déclaration de la réserve de la pompe, en litres

double prix(int nb)                     // définition de la fonction prix
{
        .....
}

int delivre(int nb)                     // définition de la fonction delivre
{
        .......
}


// ------------------- fichier clients.cpp ------------------

#include <iostream.h>                   // pour les entrées-sorties
#include "pompe.h"                      // pour les déclarations communes

int main()                              // définition de la fonction principale
{
        ......
}
```

Exercice 6

Testez ce programme
Modifiez ce programme afin d'avoir :
1 fichier pompe.h avec toutes les déclarations :
const double prixdulitre = 1.10;
int reserve = 10000;
double prix(int nb) ;
int delivre(int nb) ;

1 fichier pompe.cpp avec les fonctions

Puis 1 fichier clients.cpp avec le main

C++

Les directives de préprocesseur

Les directives de préprocesseur
#include & #define & #if….

#include <fichier> permet d'insérer un fichier stocké dans le dossier spécifique du compilateur

#include ‘’fichier’’ permet d'insérer un fichier stocké dans le même dossier que le fichier source
ou #include ‘’chemin/fichier’’ ou #include ‘’chemin\fichier’’

#define permet de définir une macro qui sera étendue avant le traitement par le compilateur.
Exemple : #define PI 3.14 //ceci n'est pas conseillé.
On préfère :
const double PI 3.14

On ne se sert pas du #define comme en C… mais plutôt comme :

Les directives de préprocesseur
#include & #define & #if….

#define permet de définir une macro qui sera étendue avant le traitement par le compilateur.
Exemple : #define PI 3.14 //ceci n'est pas conseillé.
On préfère :
const double PI 3.14

On ne se sert pas du #define comme en C… mais plutôt comme :

```
#ifndef CEFICHIER_H
#define CEFICHIER_H
…..
#endif
```

Ici, on fait une compilation conditionnelle qui va inclure un fichier d'en-tête 1 seule fois.

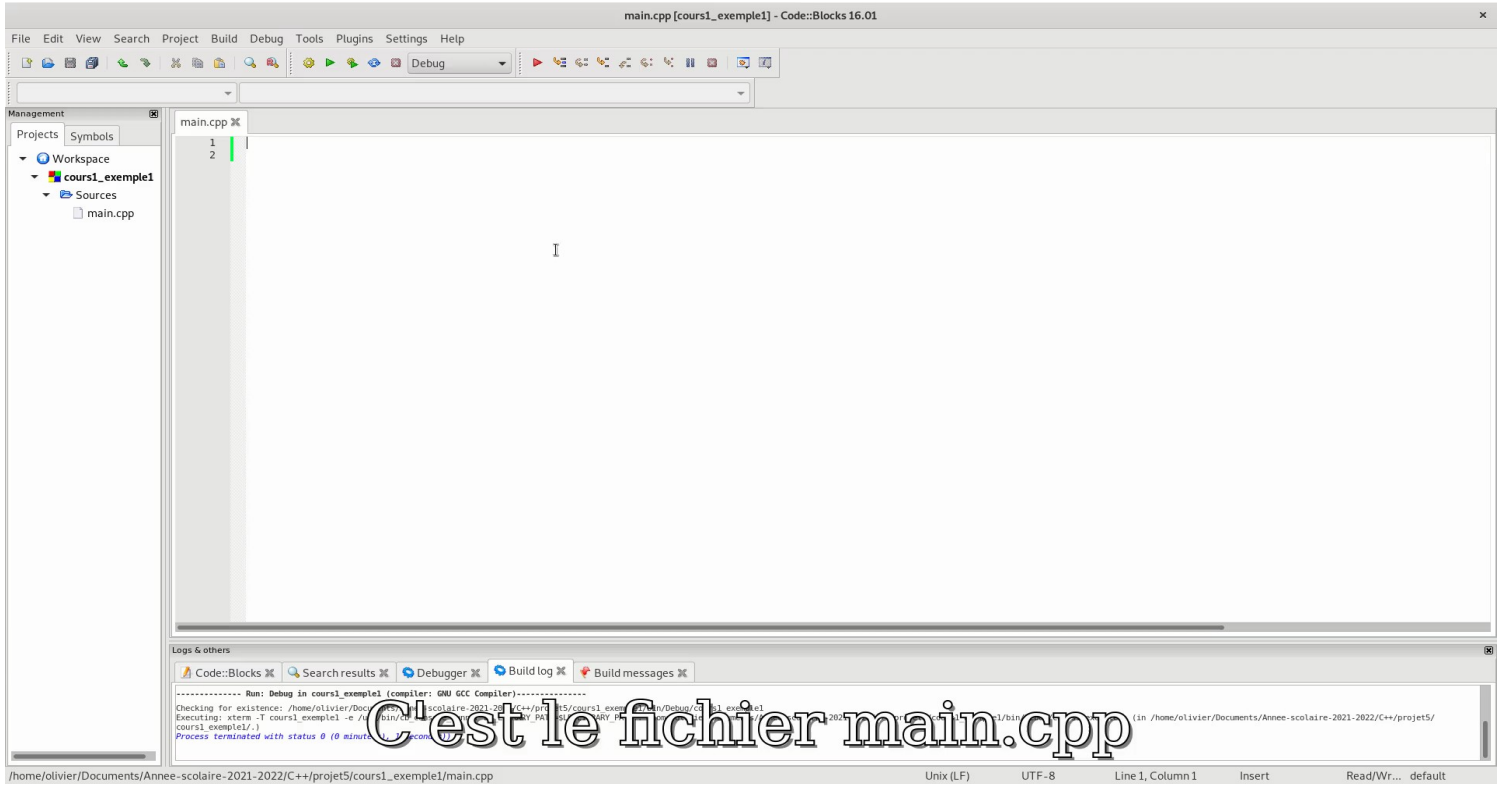
Un fichier en-tête contient tout ce qu'on veut rendre "public" : les constantes symboliques, les prototypes de fonctions, les déclarations de variables globales.
→ compilation séparée (multi-fichiers)

Les directives de préprocesseur
#define & #if…. Exemple !

```
// 1er fichier : test.cpp
#include "test.h"
#include <iostream>
#include "test.h"

int main()
{
    std::cout<<"Bonjour les #include! "<<std::endl;
#ifdef TEST
    std::cout<<"Est-ce que cette ligne apparait??
"<<std::endl;
#endif
return 0;
}
```

```
//2nd fichier : test.h
//même répertoire
#ifndef TEST_H
#define TEST_H

#define TEST

#endif
```

Les directives de préprocesseur

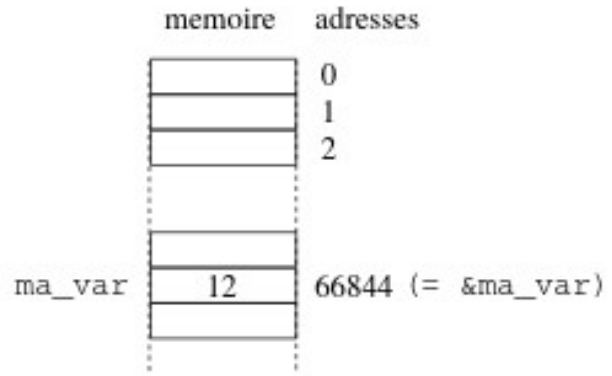


C++

Les pointeurs
Les références

Les pointeurs et les références

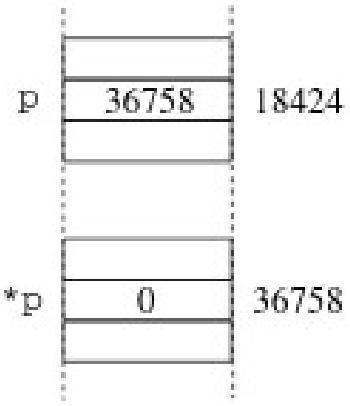
La mémoire d'un ordinateur est un empilement de cases mémoire.
Chaque case mémoire est repérée par un numéro : son adresse physique : un entier long. Dans la case mémoire, il y a un emplacement où l'on peut stocker des données.



Si **mavar** est une variable d'un type quelconque, l'adresse où est stockée la valeur de **mavar** s'obtient grâce à l'opérateur d'adresse &: cette adresse se note **&mavar**

Les pointeurs et les références

Un pointeur est une variable qui contient l'adresse d'une autre variable.
La déclaration d'un pointeur est de la forme suivante :
<type>*<nom>;
Par exemple :int *p;
La variable p est alors un pointeur sur un int; la variable entière dont p contient l'adresse est dite pointée par p et se note *p.
Schématiquement :



Les pointeurs et les références

Une variable pointée s’utilise comme une variable ordinaire. Exemple :

```
int *p, *q, n;
n = -16;
*p = 101;
*q=n+*p;  //donne `a *q la valeur 85
```

Les pointeurs et les références

Une référence est une variable qui coïncide avec une autre variable.
La déclaration d'une référence est de la forme suivante :
<type> &<nom> = <nomvar>;

Par exemple :
int n = 10;
int &r = n;   // r est une référence sur n

Cela signifie que r et n représentent la même variable (en particulier, elles ont la même adresse).
Si on écrit par la suite : r=20, n prend également la valeur 20.

TD (à la maison)

Tri des valeurs dans un tableau :
Votre programme demande de saisir des valeurs. Pour terminer de saisir, il devra appuyer sur la touche P
Les valeurs sont toutes enregistrées dans un tableau
Une fois que l'utilisateur a appuyé sur la touche P, votre programme affiche le tableau et une question sera posée à l'utilisateur : tri sens croissant ou tri sens décroissant ?
Réponses C pour croissant et D pour décroissant
Le tableau est à nouveau affiché avec le tri

→ Fonctions obligatoires
→ Affichage un peu « funky »