

Programmation Orientée Objet (POO)

Langage C++

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

2018 - 2019

Pourquoi C++

- Répondre à des besoins générés par de **gros** projets.
- Besoin d'un code plus structuré, extensible, réutilisable, portable, etc.
 - ➔ Le langage C est insuffisant !!



Bjarne Stroustrup
(AT &T Bell Laboratories)

➔ C++

Exigences d'un projet informatique

Exigences fonctionnelles:

- Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.

Exigences Techniques :

- **Les performances:** Temps de réponse, Haute disponibilité et tolérance aux pannes, Eviter le problème de montée en charge
- **La maintenance:** Une application doit évoluer dans le temps., Doit être fermée à la modification et ouverte à l'extension
- **Sécurité**
- **Portabilité**
- **Distribution**
- **Capacité de communiquer avec d'autres applications distantes.**
- **Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http...)**
-
- **Coût du logiciel**

C++ extension du langage C

Commentaire
Déclarations libres
Référence
Arguments par défaut
Surdéfinition de fonctions
Opérateurs new et delete
Fonctions inline
Type bool
Espace de nom

Classe
Objet
Encapsulation
Héritage
polymorphisme

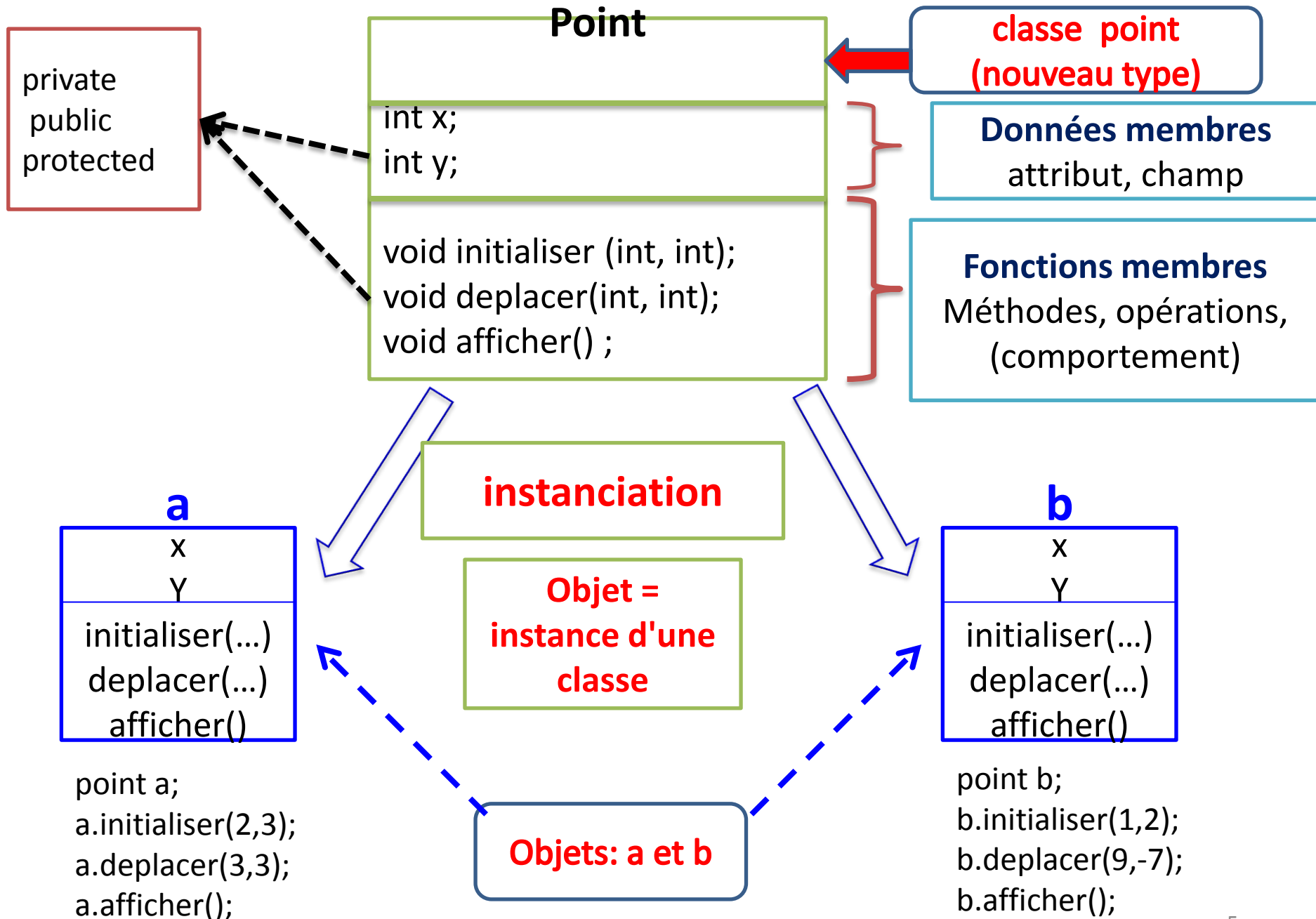
langage
C++

Notions
non O.O.

Notions
O.O.

Langage C

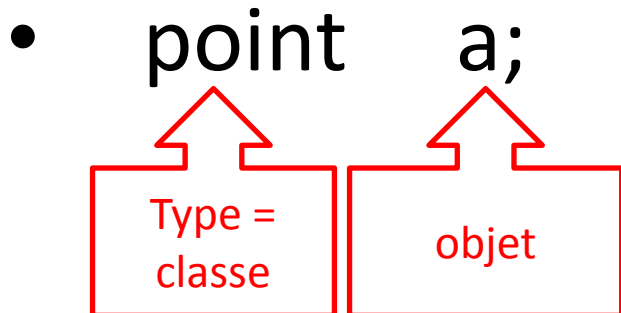
O.O.:
Orientée
Objet



La Programmation Orientée Objet (P.O.O.)

- Exemple de langages OO: C++, java, C#, etc
- **Objet:** ensemble de **données** et de **méthodes** qui agissent sur ces données.

Objet = données + méthodes



Programmation procédurale vs programmation orientée objet

Programmation procédurale

```
struct point
{
    int x;
    int y;
};
typedef struct point POINT;
```

```
POINT saisirPoint();
void saisirPoint2(POINT*);
void afficherPoint(POINT);
```

```
void main()
{
    POINT p;
    p=saisirPoint();
    saisirPoint2(&p);
    afficherPoint(p);
    getch();
}
```

programmation orientée objet

```
class point
{
    int x;
    int y;
public:
    void initialiser(int,int);
    void deplacer(int,int);
    void afficher();
};
```

```
void main()
{
    point a;
    a.initialiser(2,3);
    a.deplacer(4,4);
    a.afficher();
    system("PAUSE");
}
```

Les entrées – sorties du C++


```
#include<iostream>
using namespace std;
void main()
{
    cout<<"premiere annee "<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe

```
premiere annee
Appuyez sur une touche pour continuer...
```

(Global scope)

```
#include<iostream>
using namespace std;
void main()
{
    int x;
    cin>>x;
    cout<<"la valeur lue "<<x<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe

```
22
la valeur lue 22
Appuyez sur une touche pour continuer...
```

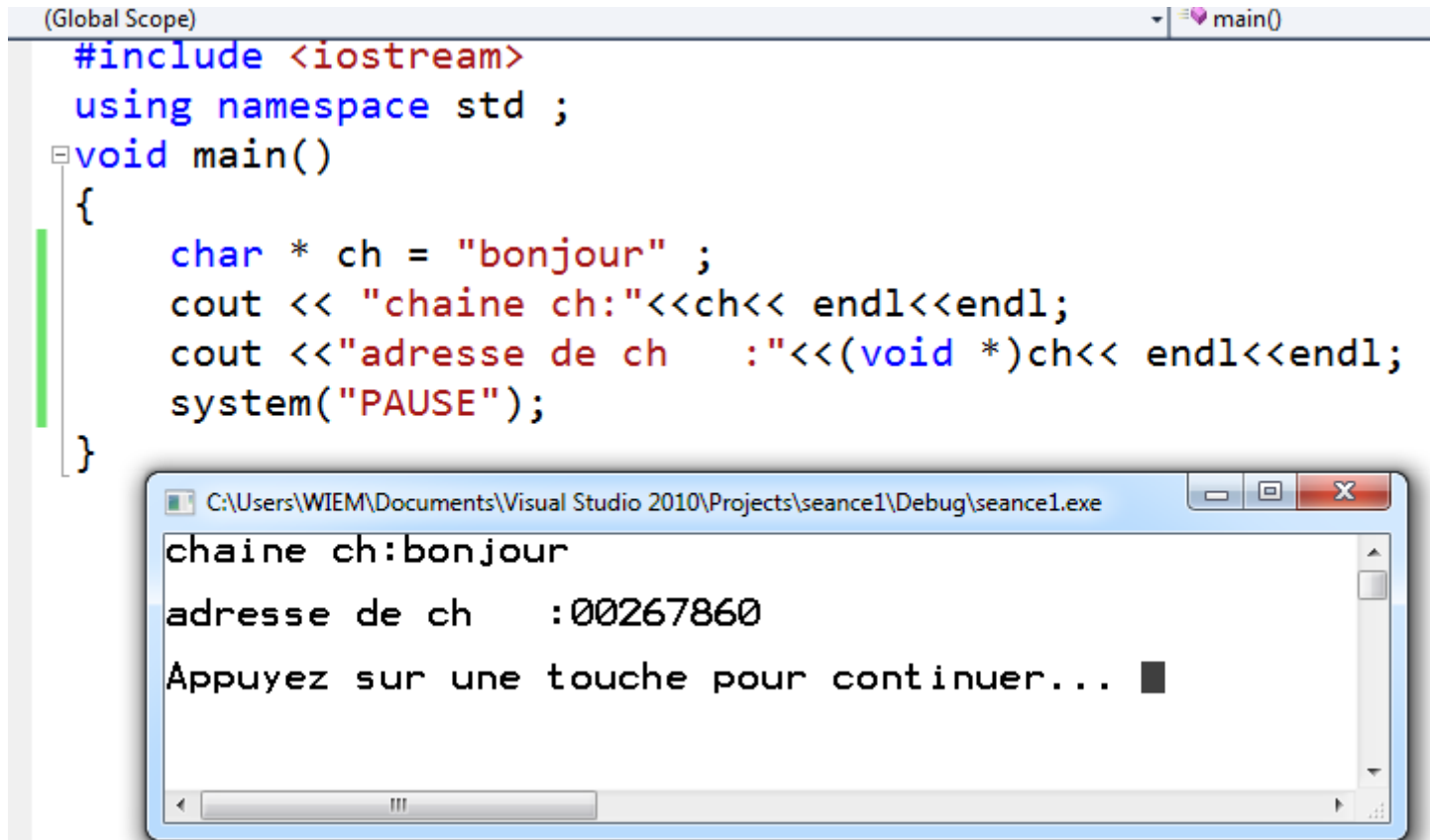
The image shows a Visual Studio 2010 IDE window with a file named `main.cpp`. The code is in the Global Scope and contains the following C++ code:

```
#include<iostream>
using namespace std;
void main()
{
    int x;
    float y;
    char z;
    cin>>x>>y>>z;
    cout<<x<<" "<<y<<" "<<z<<endl;
    system("PAUSE");
}
```

Below the code editor, a console window titled `C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe` displays the output of the program:

```
12
22.33
A
12 22.33 A
Appuyez sur une touche pour continuer...
```

Chaines de caractères



The image shows a C++ program in a Visual Studio editor window and its execution output in a console window.

Code Editor (Global Scope):

```
#include <iostream>
using namespace std ;
void main()
{
    char * ch = "bonjour" ;
    cout << "chaine ch:"<<ch<< endl<<endl;
    cout <<"adresse de ch    :"<<(void *)ch<< endl<<endl;
    system("PAUSE");
}
```

Console Output (C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\seance1.exe):

```
chaine ch:bonjour
adresse de ch    :00267860
Appuyez sur une touche pour continuer...
```

pointeurs

(Global Scope)

main()

```
#include <iostream>
using namespace std ;
void main()
{
    int n=25;
    int * ad = & n ;
    cout << "adresse de n   :" << &n <<endl<<endl;
    cout << "adresse de n   :" << ad <<endl<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\seance1.exe

adresse de n :0043FD28

adresse de n :0043FD28

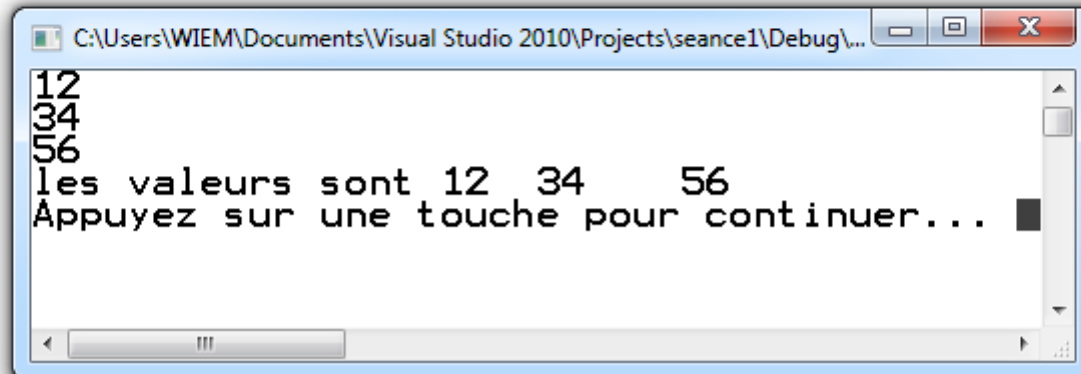
Appuyez sur une touche pour continuer...

Lecture de plusieurs valeurs

(Global Scope)

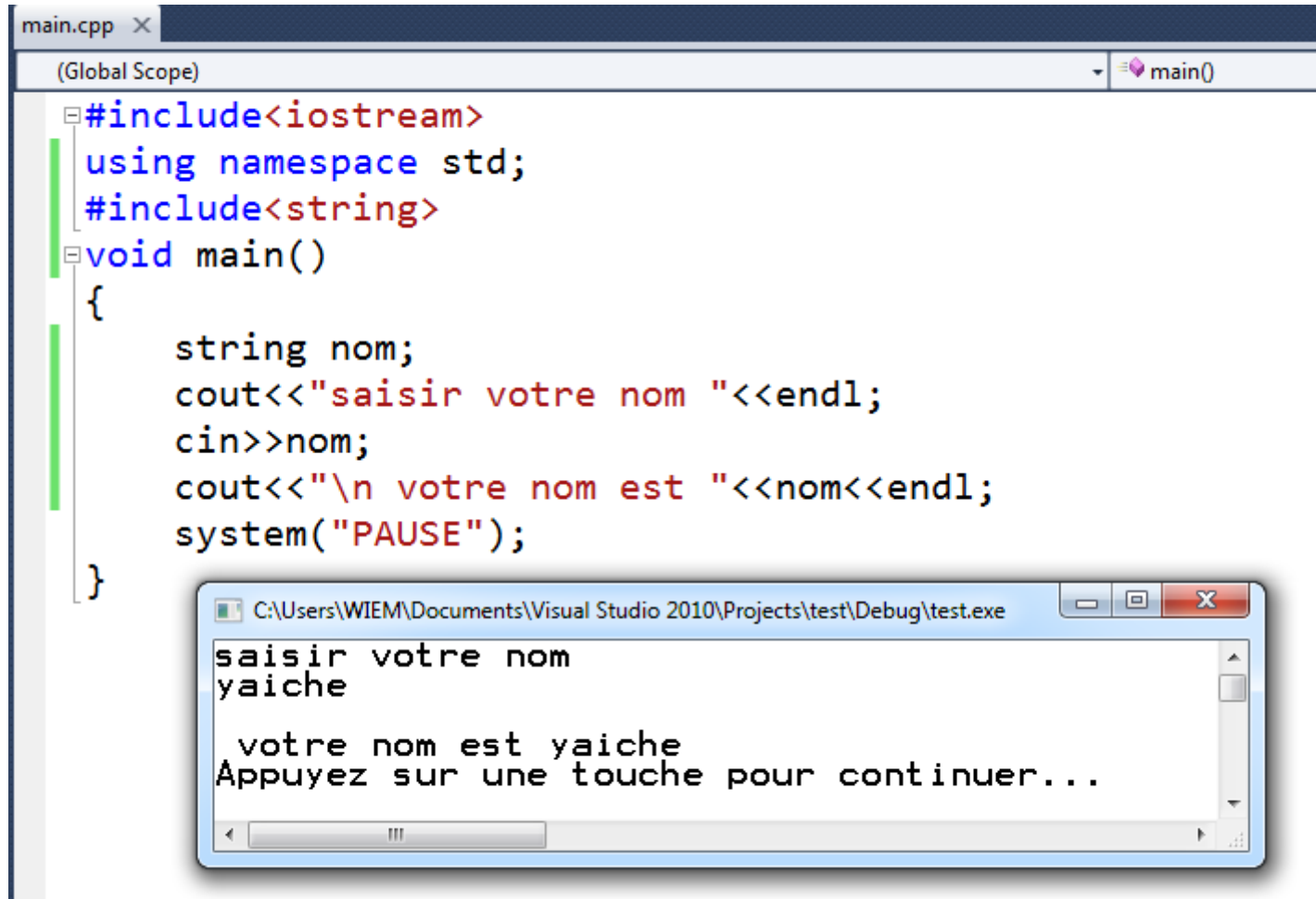
main()

```
#include <iostream>
using namespace std ;
void main()
{
    int x, y, z;
    cin >> x >> y >> z;
    cout << "les valeurs sont " << x << " " << y << " " << z << endl;
    system("PAUSE");
}
```



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\...
12
34
56
les valeurs sont 12 34 56
Appuyez sur une touche pour continuer...
```

Chaînes de caractères



The image shows a Visual Studio code editor window with a file named `main.cpp`. The code is in C++ and uses the `std::string` and `std::cout`/`std::cin` for string handling. The program prompts the user to enter a name and then displays it. Below the code editor, a console window shows the program's execution, where the user has entered 'yaiche' and the program has responded with the name and a pause instruction.

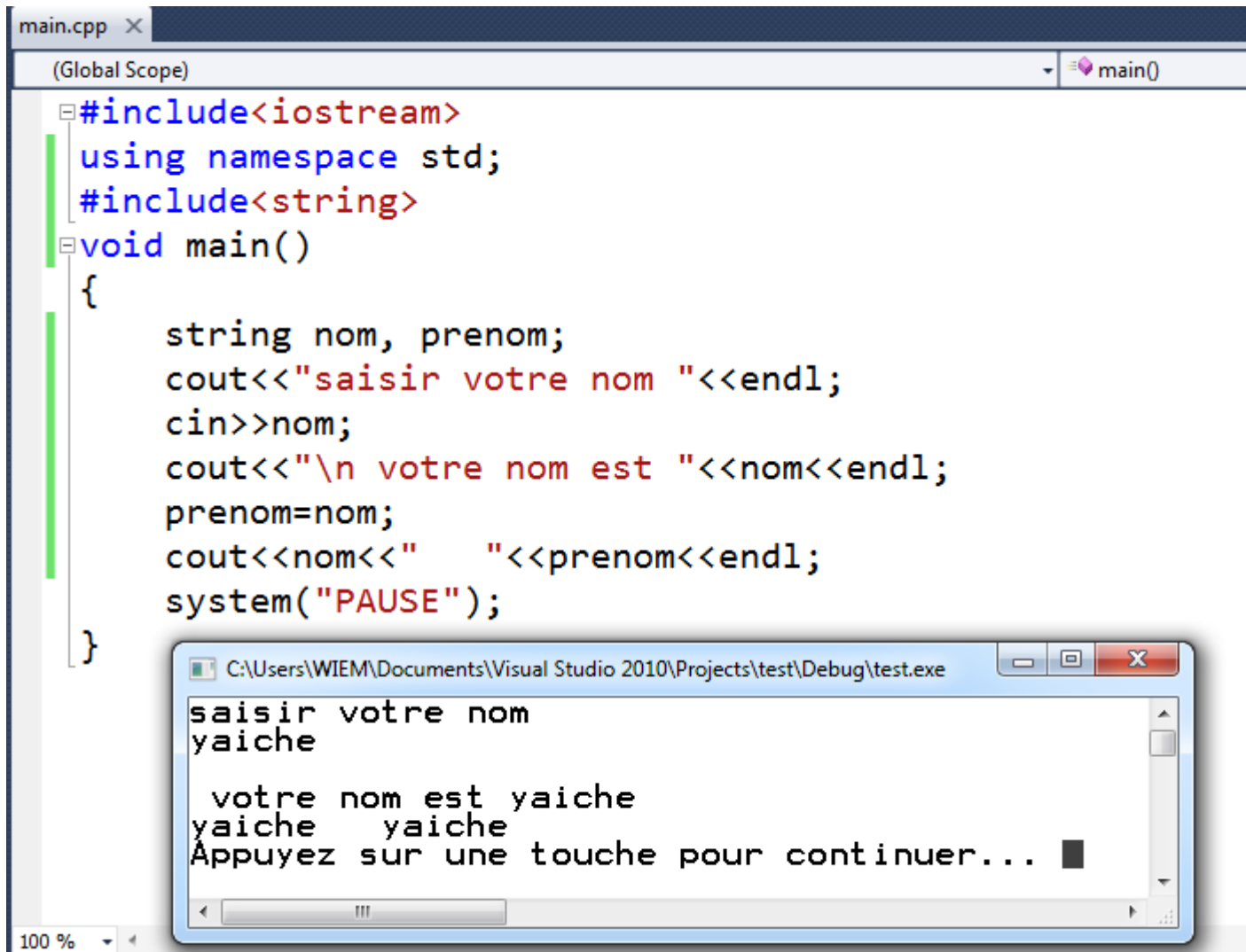
```
main.cpp ×
(Global Scope) main()
#include<iostream>
using namespace std;
#include<string>
void main()
{
    string nom;
    cout<<"saisir votre nom "<<endl;
    cin>>nom;
    cout<<"\n votre nom est "<<nom<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe

```
saisir votre nom
yaiche

votre nom est yaiche
Appuyez sur une touche pour continuer...
```

Affectation de deux chaînes



The image shows a Visual Studio IDE window with a C++ file named `main.cpp`. The code defines a `main` function that prompts the user to enter a name, reads the input, and then prints the name twice. A console window is open in the foreground, showing the program's execution with the input `yaiche` and the corresponding output.

```
main.cpp x
(Global Scope) main()
#include<iostream>
using namespace std;
#include<string>
void main()
{
    string nom, prenom;
    cout<<"saisir votre nom "<<endl;
    cin>>nom;
    cout<<"\n votre nom est "<<nom<<endl;
    prenom=nom;
    cout<<nom<<"    "<<prenom<<endl;
    system("PAUSE");
}
```

Output window (C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe):

```
saisir votre nom
yaiche

votre nom est yaiche
yaiche    yaiche
Appuyez sur une touche pour continuer...
```

The image shows a Visual Studio 2010 window with a C++ file named `main.cpp` open. The code is in the Global Scope and contains the following:

```
#include<iostream>
using namespace std;
#include<string>
void main()
{
    int x;
    float y;
    string nom;
    cout<<"saisir entier, chaine et reel"<<endl;
    cin>>x>>nom>>y;
    cout<<x<<" "<<nom<<" "<<y<<endl;
    system("PAUSE");
}
```

Below the code editor, a console window titled `C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe` displays the program's output:

```
saisir entier, chaine et reel
12
azerty
6.5
12 azerty 6.5
Appuyez sur une touche pour continuer...
```

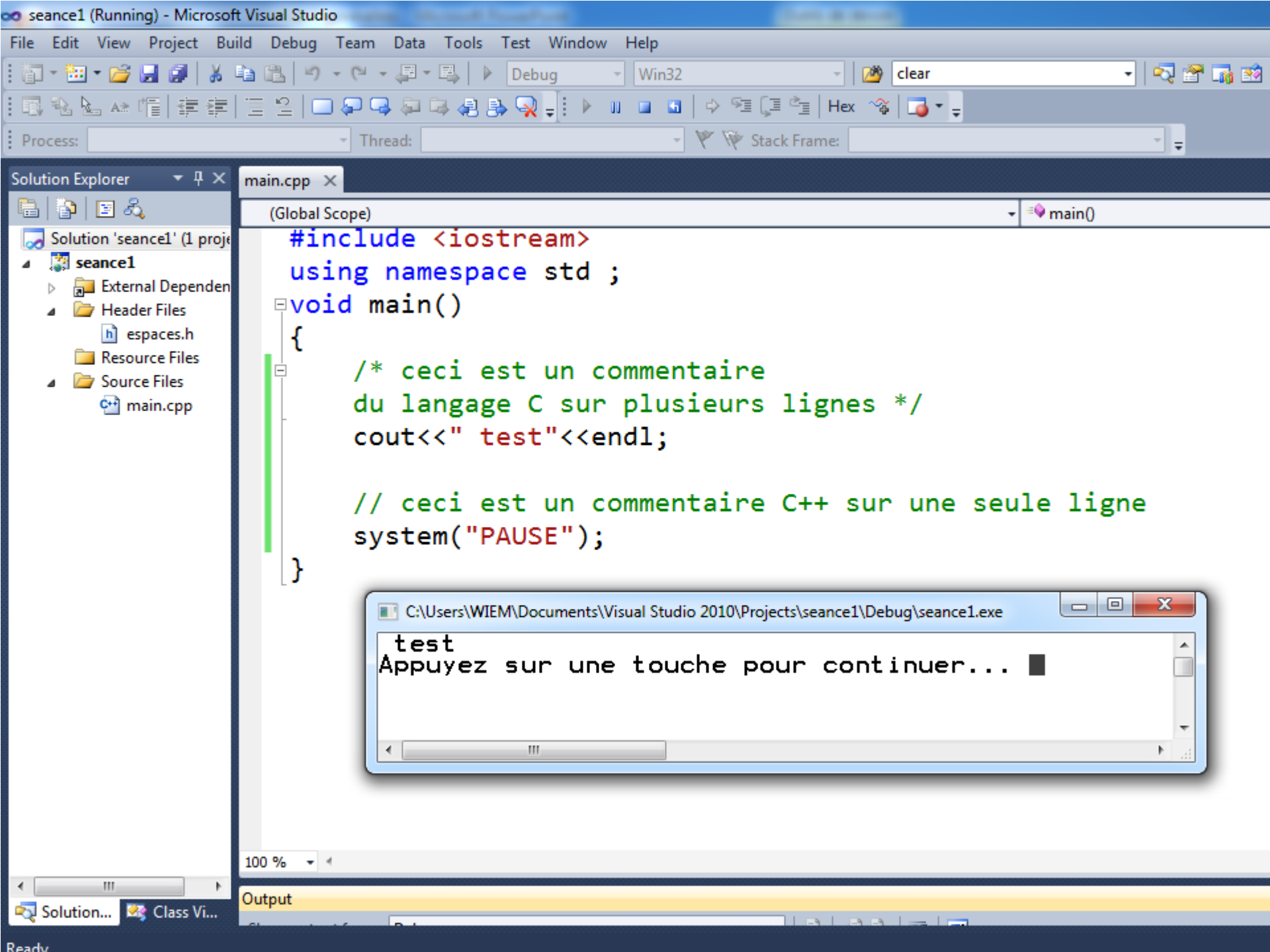
The console window also shows a scrollbar and a status bar at the bottom.

Les spécificités du C++ (non O.O.)

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O.

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.



plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O.

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

Déclarations et initialisations

- C++ est plus souple que le C en matière de déclarations.
- en C++, il n'est plus obligatoire de **regrouper** au début les déclarations effectuées au sein d'une fonction ou au sein d'un bloc.
- Celles-ci peuvent être effectuées **n'importe où**, pourvu qu'elles apparaissent avant que l'on en ait besoin: leur portée reste limitée à la partie du bloc ou de la fonction suivant leur déclaration

Déclaration de
q

The image shows a C++ program in a Visual Studio editor window titled 'main.cpp'. The code is in the 'Global Scope' and defines a 'main()' function. Inside 'main()', it includes the <iostream> header, uses the std namespace, declares an integer 'n', prompts the user to enter a value for 'n', reads the input, calculates 'q' as twice 'n', and prints the value of 'q'. It also includes a 'system("PAUSE");' call to keep the console window open. A green arrow points from a text box on the left to the 'int q = 2*n;' line. Below the code editor is a console window showing the program's execution: it prompts for 'n', the user enters '4', it prints 'la valeur de n est: 4', 'la valeur de q est: 8', and ends with 'Appuyez sur une touche pour continuer...'.

```
main.cpp X
(Global Scope) main()
#include <iostream>
using namespace std ;
void main()
{
    int n;
    cout<<"saisir une valeur de n "<<endl;
    cin>>n;
    cout<<"la valeur de n est: " <<n<<endl;
    int q= 2*n;
    cout<<"la valeur de q est: " <<q<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\sean...
saisir une valeur de n
4
la valeur de n est: 4
la valeur de q est: 8
Appuyez sur une touche pour continuer...

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O.

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. **notion de référence,**
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

Passages d'arguments à une fonction

- En langage C:
 - Passage par valeur
 - Passage par adresse
- En langage C++
 - Passage par valeur
 - Passage par adresse
 - Passage par référence

notion de référence

- A côté des pointeurs, les références sont une autre manière de manipuler les adresses des objets placés dans la mémoire.
- Une référence est un pointeur géré de manière interne par la machine.
- Si **T** est un type donné, le type « **référence sur T** » se note **T&**. (exemple: `int &` ; `float&` ; `point&`)

The screenshot shows a Visual Studio 2010 IDE with three tabs: point.cpp, point.h, and main.cpp. The main.cpp tab is active, showing the following code:

```
#include "point.h"
#include <iostream>
using namespace std;

void main()
{
    int x=5;
    int &ref=x; // ref est une référence sur x

    cout<<x<<"  "<<&x<<endl;
    cout<<"\n-----"<<endl;
    cout<<ref<<"  "<<&ref<<endl;
    system("PAUSE");
}
```

Below the code editor, a console window titled "C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe" displays the output of the program:

```
5  0043F9CC
-----
5  0043F9CC
Appuyez sur une touche pour continuer...
```

x ref
5

notion de référence

- Une valeur de type référence est une adresse
- toute opération effectuée sur la référence **agit sur la variable référencée**, non sur l'adresse.
- Il en découle qu'il est obligatoire **d'initialiser** une référence lors de sa création ;
- Une fois déclarée et initialisée, il **est interdit de modifier une référence**. → Contrairement à un pointeur, **une référence est toujours constante**.

```
point.cpp point.h main.cpp x
(Global Scope)
#include "point.h"
#include <iostream>
using namespace std;
void main()
{
    int x=5;
    int &ref=x; // ref est une référence sur x

    cout<<x<<" " <<&x<<endl;
    cout<<ref<<" " <<&ref<<endl;
    cout<<"\n-----" <<endl;
    x++;
    cout<<x<<" " <<&x<<endl;
    cout<<ref<<" " <<&ref<<endl;
    system("PAUSE");
}
```

Output Window: C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test...

```
5 002FFEC4
5 002FFEC4

-----
6 002FFEC4
6 002FFEC4
Appuyez sur une touche pour continuer...
```

toute opération effectuée sur la référence agit sur l'objet référencé

The screenshot shows a C++ program in Visual Studio. The code defines a reference variable `ref` that points to `x`. The line `ref++;` is circled in red. The output window shows the memory addresses for both `x` and `ref` at two different points in the program, both being `003AF90C`, which demonstrates that they refer to the same memory location.

```
point.cpp point.h main.cpp X
(Global Scope)
#include "point.h"
#include <iostream>
using namespace std;
void main()
{
    int x=5;
    int &ref=x; // ref est une référence sur x

    cout<<x<<" "<<&x<<endl;
    cout<<ref<<" "<<&ref<<endl;
    cout<<"\n-----"<<endl;
    ref++;
    cout<<x<<" "<<&x<<endl;
    cout<<ref<<" "<<&ref<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\te...
5 003AF90C
5 003AF90C

6 003AF90C
6 003AF90C
Appuyez sur une touche pour continuer...

main.cpp

(Global Scope)

```
#include <iostream>
using namespace std ;
void main()
{
    int i=5;
    int &r;
    r=i;
    cout<<"\n la valeur de i est: "<<i<<endl;
    cout<<"\n la valeur de r est: "<<r<<endl<<endl;

    system("PAUSE");
}
```

Référence non initialisée

il est obligatoire d'initialiser une référence lors de sa déclaration

100 %

Output

Show output from: Build

InitializeBuildStatus:
Touching "Debug\seance1.unsuccessfulbuild".
Compile:
main.cpp
\\users\wiem\documents\visual studio 2010\projects\seance1\seance1\main.cpp(6): error C2530: 'r' : references must be initialized
ild FAILED.

Code Definition Window Output

Ln13 Col1

Remarque

l'opérateur **&** (à un argument) a une signification très différente selon le contexte dans lequel il apparaît :

- employé dans une **déclaration**, comme dans **int &r = i;** il sert à indiquer un type référence : « r est une référence sur un int »
- employé ailleurs que dans une déclaration, il indique l'opération « **obtention de l'adresse** », comme dans l'expression suivante qui signifie « affecter l'adresse de x à p » : **p = &x;**

Remarque 2

- Il n'est pas possible de définir des **pointeurs sur des références**, ni **des tableaux de références**.
- On ne peut pas initialiser une référence avec une constante.

```
int &p=3; // incorrect
```


Passage par adresse

```
#include<iostream>
using namespace std;
void permuter (int *a, int *b)
{
    int aide;
    aide=*a;
    *a=*b;
    *b=aide;
}
void main()
{
    int x,y;
    cout<<"\n saisir deux valeurs " <<endl;
    cin>>x>>y;
    cout<<"\n avant la permutation " <<x<<" " <<y<<endl;
    permuter(&x,&y);
    cout<<"\n apres la permutation " <<x<<" " <<y<<endl;
    system("PAUSE");
}
```

Output:

```
saisir deux valeurs
3 5
avant la permutation 3 5
apres la permutation 5 3
Appuyez sur une touche pour continuer.
```

Diagram illustrating memory addresses and variable values:

Variable	Memory Address	Value
a	F800	3
b	F200	5

Passage par référence

Solution Explorer

- Solution 'seance1' (1 project)
- seance1
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

main.cpp

```
#include <iostream>
using namespace std;
void permuter(int &a, int &b)
{
    int aide;
    aide=a;
    a=b;
    b=aide;
}
void main()
{
    int x,y;
    cout<<"saisir deux valeurs "<<endl;
    cin>>x>>y;
    cout<<"\n avant : "<<x<<" "<<y<<endl;
    permuter(x,y);
    cout<<"\n apres : "<<x<<" "<<y<<endl;
    system("PAUSE");
}
```

Console Output:

```
saisir deux valeurs
2
3
avant : 2 3
apres : 3 2
Appuyez sur une touche pour continuer...
```

Memory Diagram:

Variable	Value	Address
x a	3	F800
y b	5	F200

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O.

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

Arguments par défaut

- En C, il est indispensable que l'appel d'une fonction contienne **autant** d'arguments que la fonction en attend effectivement.
- C++ permet de s'affranchir en partie de cette règle, grâce à un mécanisme **d'attribution de valeurs par défaut à des arguments**.

main.cpp* X

(Global Scope) fct(int x, int y, int z)

```
#include<iostream>
using namespace std;
void fct(int x=99, int y=88, int z=77)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl<<endl;
}
void main()
{
    fct();
    fct(4);
    fct(4,5);
    fct(4,5,6);
    system("PAUSE");
}
```

c:\users\wiem\documents\visual studio 2010\Projects\test_surdef\Debug\test_surdef.exe

99	88	77
4	88	77
4	5	77
4	5	6

Appuyez sur une touche pour continuer...

The image shows a Visual Studio 2010 IDE window with a file named `main.cpp`. The code is in C++ and defines a function `fct` and a `main` function. The `main` function calls `fct` with various arguments, including a commented-out call `// fct(); ==> erreur`. The output window shows the results of these calls, which are formatted with tabs. The output is as follows:

```
#include<iostream>
using namespace std;
void fct(int x, int y=88, int z=77)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl<<endl;
}
void main()
{
    // fct(); ==> erreur
    fct(4);
    fct(4,5);
    fct(4,5,6);
    system("PAUSE");
}
```

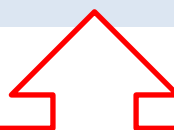
Output:

```
c:\users\wiem\documents\visual studio 2010\Projects\test\Debug\test.exe
4      88      77
4      5       77
4      5       6
Appuyez sur une touche pour continuer
```

Les propriétés des arguments par défaut

- Lorsqu'une déclaration prévoit des valeurs par défaut, les arguments concernés doivent obligatoirement être **les derniers** de la liste.

```
float fct (int x=5, long y, int z=3) ; //interdit
```



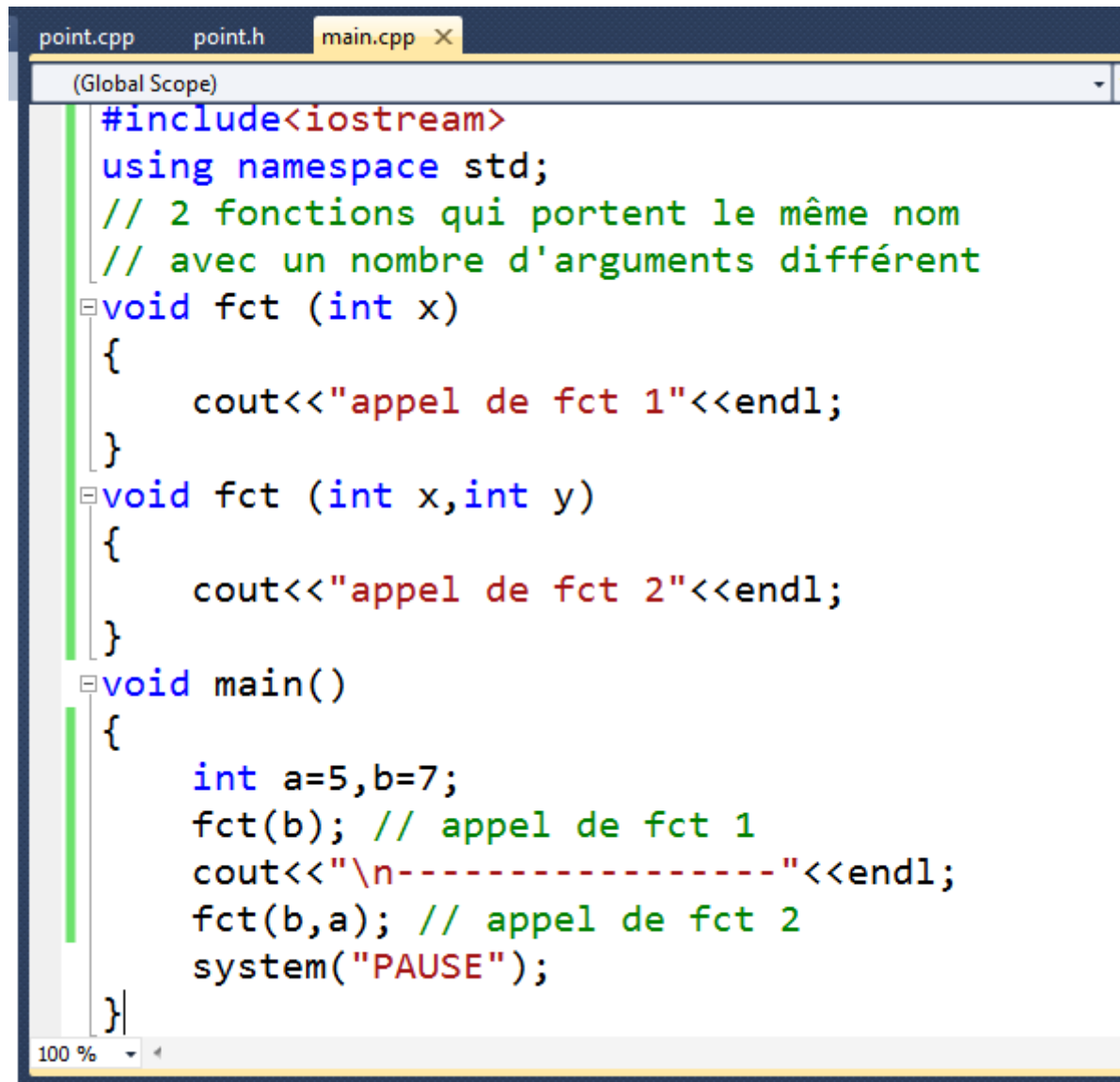
L'argument y doit être soit
initialisé, soit placé au début

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. **surdéfinition de fonctions,**
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

Surdéfinition/surcharge des definitions

- D'une manière générale, on parle de "**surdéfinition**" ou "**surcharge**" ou "**overloading**" lorsqu'un **même** symbole possède plusieurs significations différentes
- le choix de l'une des significations se fait en fonction du contexte.
- Pour pouvoir employer plusieurs fonctions de même nom, il faut bien sûr un critère (autre que le nom) permettant de choisir la bonne fonction.



The image shows a screenshot of a C++ IDE with three tabs: point.cpp, point.h, and main.cpp. The main.cpp tab is active, showing the following code:

```
#include<iostream>
using namespace std;
// 2 fonctions qui portent le même nom
// avec un nombre d'arguments différent
void fct (int x)
{
    cout<<"appel de fct 1"<<endl;
}
void fct (int x,int y)
{
    cout<<"appel de fct 2"<<endl;
}
void main()
{
    int a=5,b=7;
    fct(b); // appel de fct 1
    cout<<"\n-----"<<endl;
    fct(b,a); // appel de fct 2
    system("PAUSE");
}
```

The code demonstrates function overloading with two functions named `fct`. The first function takes a single integer argument and prints "appel de fct 1". The second function takes two integer arguments and prints "appel de fct 2". The `main` function calls `fct(b)` and `fct(b,a)` to demonstrate the compiler's ability to distinguish between the two functions based on the number of arguments.

```
point.cpp  point.h  main.cpp X
(Global Scope)  main()
#include<iostream>
using namespace std;
// 2 fonctions qui portent le même nom
// avec le même nombre d'arguments
// => le type des arguments doit être différent
void fct (int x)
{
    cout<<"appel de fct 1"<<endl;
}
void fct (float y)
{
    cout<<"appel de fct 2"<<endl;
}
void main()
{
    int a=5; float y=2.7;
    fct(a); // appel de fct 1
    cout<<"\n-----"<<endl;
    fct(y); // appel de fct 2
    system("PAUSE");
}
```

Cas de surdefinition de fonctions comportant plusieurs arguments

Conversions implicites:

char → short → int → long

float → double

```
void fct (int) ;           // fct1  
void fct (double) ;       //fct 2
```

```
char c ; float y ;
```

```
fct(c); // appelle fct1, après conversion de c en int  
fct (y); // appelle fct2, après conversion de y en double
```

remarque

- Les fonctions comportant un ou plusieurs arguments par défaut sont traitées comme si **plusieurs** fonctions différentes avaient été définies avec un nombre croissant d'arguments.

Règles de recherche d'une fonction surdéfinie

- Le compilateur recherche la "**meilleure correspondance**" possible:
 - Correspondance exacte
 - Correspondance avec promotions numériques:
char -> short -> int -> long
float -> double
 - Conversions dites standard: , il peut s'agir de conversions dégradantes (float=> int).

Règles de recherche d'une fonction surdéfinie

- La recherche s'arrête au premier niveau ayant permis de trouver une correspondance, qui doit alors être **unique**.
- Si **plusieurs** fonctions conviennent au même niveau de correspondance, il y a erreur de compilation due à l'ambiguïté rencontrée.
- si **aucune** fonction ne convient à aucun niveau, il y a aussi erreur de compilation.

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O.

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. Surdéfinition/surcharge de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

New et delete

- En langage C, la gestion dynamique de mémoire fait appel à des fonctions de la bibliothèque standard telles que *malloc* et *free*.
- comme toutes les fonctions standard, celles-ci restent utilisables en C++.
- Mais dans le contexte de la Programmation Orientée Objet, C++ a introduit deux nouveaux opérateurs, **new** et **delete**, particulièrement adaptés à la gestion dynamique d'**objets**.

new et delete

- Ces opérateurs peuvent également être utilisés pour des "variables classiques".
- Dans ces conditions, il est plus raisonnable, en C++, d'utiliser systématiquement ces opérateurs (new et delete) pour les variables classiques ou les objets.

The image shows a Visual Studio IDE window with a file named `main.cpp`. The code is in C++ and demonstrates the use of `new` to allocate memory. The code is as follows:

```
#include <iostream>
using namespace std ;
void main()
{
    int* adi;
    adi=new int;
    cin>>*adi;
    cout<<"\n l'entier lu est: "<<*adi<<endl;

    system("PAUSE");
}
```

To the right of the code, there is a green-bordered box containing the text **Utilisation de new** in red.

Below the code editor, a console window titled `C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\seance1.exe` shows the program's output. The user has entered the number 5, and the program has printed "l'entier lu est: 5" followed by a pause message "Appuyez sur une touche pour continuer...".

```
main.cpp x
(Global Scope) main()

#include <iostream>
using namespace std ;
void main()
{
    int*adi;
    adi = new int;
    cin>>*adi;
    cout<<"\n l'entier lu est: "<<*adi<<endl;
    delete adi;
    cout<<"\n l'entier lu est: "<<*adi<<endl;
    system("PAUSE");
}
```

Utilisation de new et delete

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\se...
5
l'entier lu est: 5
l'entier lu est: -17891602
Appuyez sur une touche pour continuer...

The image shows a Visual Studio 2010 IDE window with a file named `main.cpp`. The code is in C++ and demonstrates dynamic memory allocation for an array of integers. The code is as follows:

```
#include <iostream>
using namespace std ;
void main()
{
    int* adi;
    adi=new int[3];
    for(int i=0; i<3;i++)
        cin>>*(adi+i);
    cout<<endl;
    for(int i=0; i<3;i++)
        cout<<*(adi+i)<<"\t";
    system("PAUSE");
}
```

To the right of the code editor, there is a green-bordered box containing the text:

**Tableau
dynamique
d'entiers
avec new**

Below the code editor, a console window titled `C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\seance1.exe` displays the program's output. The first three lines show the input values: `12`, `45`, and `78`. The fourth line shows the output of the program, which prints the values `12`, `45`, and `78` separated by tabs, followed by the text `Appuyez sur une touche pou`.

main.cpp

(Global Scope)

```
#include <iostream>
using namespace std ;
void main()
{
    int*adi;
    adi = new int[3];
    for(int i=0; i<3 ; i++)
        cin>>*(adi+i);

    for(int i=0; i<3 ; i++)
        cout<<*(adi+i)<<"\t";
    cout<<endl;
    delete []adi;
    for(int i=0; i<3 ; i++)
        cout<<*(adi+i)<<"\t";
    cout<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\seance...

```
12
45
78
12      45      78
-17891602 -17891602 -17891602
Appuyez sur une touche pour continuer...
```

Tableau dynamique
d'entiers avec new et
delete []

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

Utilisation de fonctions en ligne

- Les fonctions inline sont généralement **fréquemment appelées** et de **petites tailles**.
- Une fonction en ligne se définit et s'utilise comme une fonction ordinaire, à la seule différence qu'on fait précéder son en-tête de la spécification ***inline***
- La présence du mot *inline* demande au compilateur de traiter la fonction *fct* différemment d'une fonction ordinaire.
- A chaque appel de *fct*, **le compilateur** devra incorporer au sein du programme les instructions correspondantes en langage machine

Utilisation de fonctions en ligne

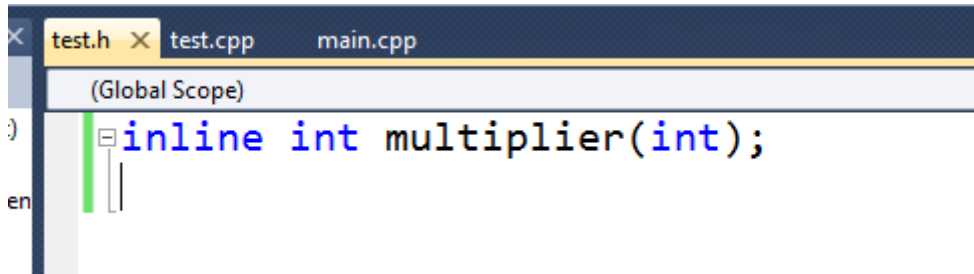
- Le mécanisme habituel de **gestion de l'appel** et du **retour** n'existera plus, ce qui permet une **économie de temps**.
- En revanche, les instructions correspondantes seront générées à chaque appel, ce qui consommera une quantité de **mémoire croissante** avec le nombre d'appels.

```
test.h  test.cpp  main.cpp X
(Global Scope)
#include<iostream>
#include"test.h"
using namespace std;
void main()
{
    int x,y;
    for(int i=1; i<4; i++) // faire 3 itérations
    {
        cout<<"\n saisir une valeur: "<<endl;
        cin>>x;
        y=multiplier(x);
        cout<<y;
    }
    cout<<endl;
    system("PAUSE");
}
```

```
int multiplier(int x)
{
    int y;
    y=x*x*x;
    return y;
}
```

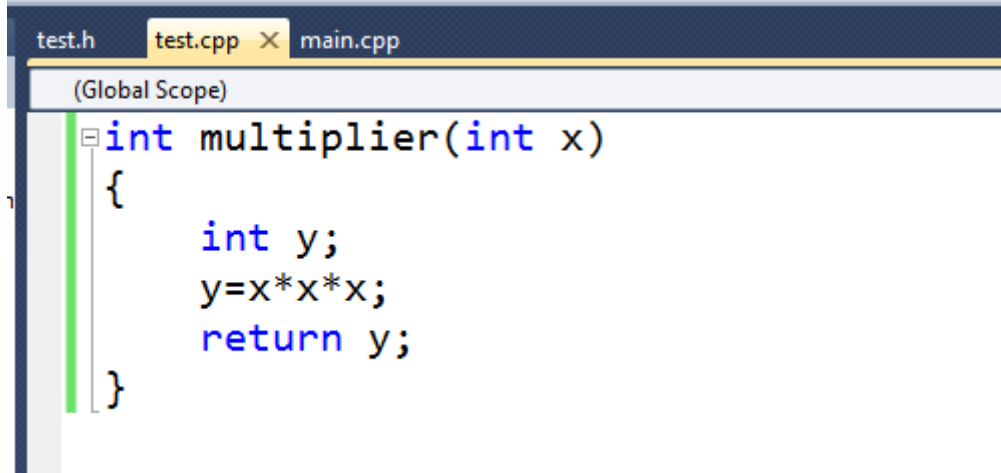
```
c:\users\wiem\documents\visual studio 2010\Projects\test\Debug\te...
saisir une valeur:
2
8
saisir une valeur:
3
27
saisir une valeur:
4
64
Appuyez sur une touche pour continuer
```

Schéma 1



A screenshot of a code editor with three tabs: test.h, test.cpp, and main.cpp. The test.h tab is active, showing the following code in the Global Scope:

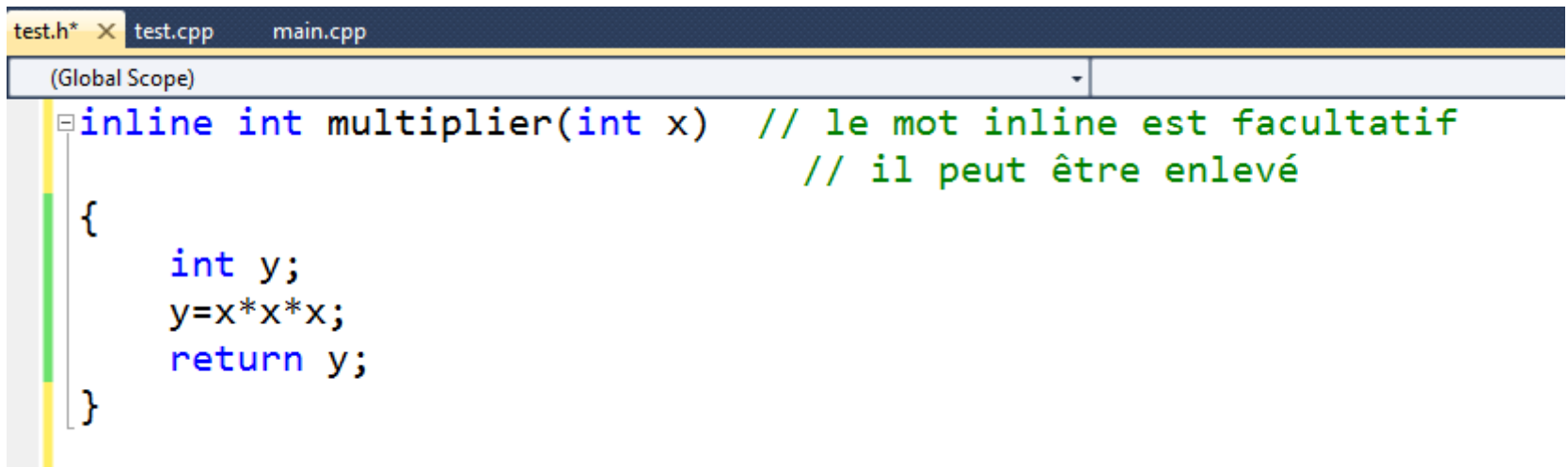
```
inline int multiplier(int);
```



A screenshot of a code editor with three tabs: test.h, test.cpp, and main.cpp. The test.cpp tab is active, showing the following code in the Global Scope:

```
int multiplier(int x)
{
    int y;
    y=x*x*x;
    return y;
}
```

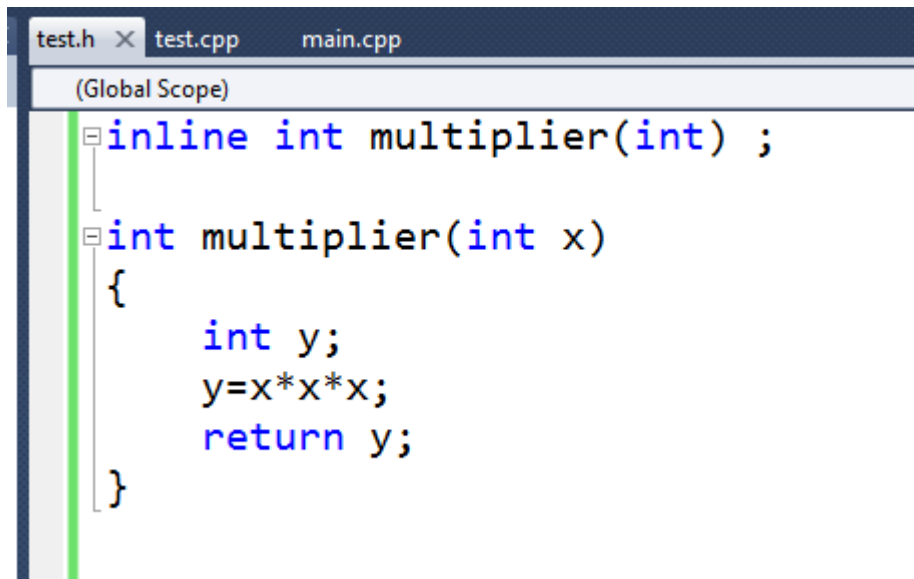
Schéma 2: mettre la définition de la fonction dans le fichier header (.h)



The screenshot shows a code editor with three tabs: test.h*, test.cpp, and main.cpp. The active tab is test.h*, which contains the following C++ code:

```
(Global Scope)
inline int multiplier(int x) // le mot inline est facultatif
                             // il peut être enlevé
{
    int y;
    y=x*x*x;
    return y;
}
```

Schéma 3: mettre la
définition de la fonction dans
le fichier header (.h), après le
prototype



The screenshot shows a code editor with three tabs: test.h, test.cpp, and main.cpp. The active tab is test.h, which contains the following code:

```
(Global Scope)
inline int multiplier(int) ;
{
    int multiplier(int x)
    {
        int y;
        y=x*x*x;
        return y;
    }
}
```

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.

The image shows a Visual Studio 2010 IDE window with a file named `main.cpp`. The code is in the Global Scope and defines a function `est_pair` and a `main` function. Annotations on the left side of the code point to specific parts:

- A box labeled **bool** points to the `bool` keyword in the function signature `bool est_pair (int a)`.
- A box labeled **1** points to the value `1` in the condition `if(est_pair(x)==1)`.
- A box labeled **true** points to the value `true` in the condition `if(est_pair(x)==true)`.

The code in the IDE is as follows:

```
#include <iostream>
using namespace std ;
bool est_pair (int a)
{
    return (a%2==0);
}
void main()
{
    int x;
    cout<<"\n saisir un entier: " << endl;
    cin>>x;
    if( est_pair(x)==1) cout<<"pair"<<endl;
    else cout<<"impair"<<endl;
    if( est_pair(x)==true) cout<<"pair"<<endl;
    else cout<<"impair"<<endl;
    system("PAUSE");
}
```

Overlaid on the IDE is a console window titled `C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\...` showing the program's output:

```
saisir un entier:
5
impair
impair
Appuyez sur une touche pour continuer
```

plan

Le langage C++ dispose d'un certain nombre de spécificités qui ne sont pas véritablement axées sur la P.O.O

1. nouvelle forme de commentaire (en fin de ligne),
2. emplacement libre des déclarations,
3. notion de référence,
4. arguments par défaut dans les déclarations des fonctions,
5. surdéfinition de fonctions,
6. opérateurs *new* et *delete*,
7. fonctions "en ligne" (*inline*),
8. existence d'un type booléen *bool*,
9. notion d'espace de noms.


```
point.cpp  point.h  main.cpp X
{ } anglais  fct()

#include "point.h"
#include <iostream>
using namespace std;
namespace français
{
    int x=5;
    void fct()
    {
        cout<<"\n espace français "<<endl;
    }
}
namespace anglais
{
    int x=3;
    void fct()
    {
        cout<<"\n espace anglais "<<endl;
    }
}
void main()
{
    cout<<français::x<<endl;
    anglais::fct();
    system("PAUSE");
}
```

:: opérateur de résolution de portée

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe

5

espace anglais

Appuyez sur une touche pour continuer...

The image shows a Visual Studio 2010 IDE window with three tabs: `point.cpp`, `point.h`, and `main.cpp`. The `main.cpp` tab is active, showing the following code:

```
#include<iostream>
using namespace std;
namespace francais
{
    int x=5;
    void fct()
    {
        cout<<"\n espace francais "<<endl;
    }
}
namespace anglais
{
    int x=3;
    void fct()
    {
        cout<<"\n espace anglais "<<endl;
    }
}
using namespace anglais;
void main()
{
    cout<<x<<endl;
    fct();
    system("PAUSE");
}
```

The code defines two namespaces: `francais` and `anglais`. The `anglais` namespace contains a variable `x` with the value 3 and a function `fct()` that prints `"\n espace anglais "`. The `main()` function uses the `anglais` namespace and prints the value of `x` (3), calls `fct()`, and then pauses the program with `system("PAUSE")`.

A console window is open in the foreground, showing the output of the program:

```
3
espace anglais
Appuyez sur une touche pour continuer...
```

Espace de nom ou namespace

- Lorsque l'on doit utiliser **plusieurs bibliothèques** dans un programme, on peut être confronté au problème d'avoir un **même identificateur** utilisé par plusieurs bibliothèques.
- la norme ANSI du C++ a introduit le concept d'"**espace de noms**". Il s'agit de **donner un nom à un "espace" de déclarations**:

```
namespace une_bibli  
{  
// déclarations usuelles  
}
```

```
namespace francais  
{  
    int x=5;  
    void fct()  
    {  
        cout<<"\n espace francais "<<endl;    }  
}
```

Espace de nom ou namespace

- Pour se référer à des identificateurs définis dans cet espace de noms, on utilisera une instruction *using* :

```
using namespace une_bibli;  
// ici, les identificateurs de  
une_bibli sont connus
```

```
using namespace francais;  
void main()  
{  
    cout<<x<<endl;  
    fct();  
    system("PAUSE");  
}
```

- Pour lever l'ambiguïté risquant d'apparaître lors de l'utilisation de plusieurs espaces de noms comportant des identificateurs identiques; il suffit de faire appel à l'opérateur de résolution de portée ::

```
void main()  
{  
    cout<<francais::x<<endl;  
    anglais::fct();  
    system("PAUSE");  
}
```

Espace de nom ou namespace

- Tous les identificateurs des fichiers en-tête standard sont définis dans l'espace de noms *std*;

```
using namespace std ;  
➔ utilisation des fichiers en-tête standard
```

- il est nécessaire de recourir systématiquement à l'instruction:

```
using namespace std ; ➔ cin, cout  
Sinon  
std::cin ; std::cout
```