

# Fonctions amies

## 1. Généralités

- Une fonction membre a accès aux membres publics et privés de la classe.
- Les données sont masquées, seules les fonctions membres y ont accès. Si l'on veut que ces données soient accessibles de l'extérieur, on peut utiliser aussi les fonctions amies.
- Une fonction amie d'une classe est une fonction qui, bien que non-membre de la classe, a accès aux données privées de la classe.
- L'amitié est déclarée en utilisant le mot-clé réservé: `friend`.
- Il importe peu de déclarer une fonction amie dans la partie `public` ou `private` d'une classe donnée. Dans les deux cas, la fonction sera vue comme étant une fonction `public`.
- Il existe plusieurs situations d'amitié ...

## 2. Fonction indépendante amie d'une classe

```
#include <iostream>
using namespace std;
class A {
    int i;
    friend void exemple (A&);
public:
    A(int e=100):i(e){}
};
void exemple (A& a) {
    // accès direct à i qui est membre données privées.
    cout << a.i << endl;
}
int main() {
    A y;
    exemple(y);
    return 0;
}
```

- La fonction « exemple » est une fonction amie de la classe « A ».
- Déclarer la fonction « exemple » « private » ou « public » importe peu.

- La définition de la fonction amie est différente de la définition d'une fonction membre d'une classe.

```
// Fausse (déclaration utilisée pour une fonction membre).  
void A::exemple (A& a) { /* etc. */ }  
  
// Correcte (déclaration utilisée pour une fonction amie).  
void exemple (A& a) { /* etc. */ }
```

- La fonction `exemple` est une fonction amie et **non pas une fonction membre de la classe A.**

### 3. Fonction membre d'une classe, amie d'une autre classe

```
#include <iostream>  
  
using namespace std;  
  
// déclaration de la classe A, le compilateur sait  
// que la classe A existe.  
  
class A;
```

```
// définition de la classe B.
class B {
public:
    /* La fonction exemple est uniquement déclarée. Comme
    argument, elle prend une référence à un objet de la classe
    A. On ne peut pas définir la fonction exemple à ce niveau,
    car si elle utilise un membre donné de la classe A,
    sachant qu'a n'a pas été encore définie, on aura des
    problèmes à la compilation.*/

    void exemple (A&);
};

// définition de la classe A
class A {
    int i;
    /* On connaît déjà le contenu de la classe B, exemple est
    une fonction membre de cette classe, elle est déclarée
    aussi, à ce niveau, comme amie de la classe A. Elle aura
    donc accès aux données privées de la classe A. */

    friend void B::exemple (A&);
public:
    // constructeur qui initialise i à la valeur de e,
    // par défaut 100.

    A(int e=100):i(e) {}
};
```

```
// A & B étant définies, on peut définir la fonction exemple.
void B::exemple (A& a) {
    cout << a.i << endl;
}
int main() {
    A x;
    B y;
    y.exemple(x); // en sortie: 100
    return 0;
}
```

- Faire attention à l'ordre des déclarations et des définitions.

## 4. Classe amie d'une autre classe

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

// déclaration de la classe A.
class A;

// définition de la classe B.
class B {
public:
    void change (A&);
    void affiche(A&);
};

class A {
    int i;

    /* La classe B est amie la classe A. De ce fait, les
    fonctions membres de la classe B c.-à-d. (change &
    affiche) sont amies de la classe A. */
    friend class B;

public:
    A(int e=100):i(e) {} // constructeur.
};
```

```
// définition de change, fonction membre de la classe B.

void B::change(A& a) {
    a.i = 200;
}

// définition de la fonction affiche, fonction membre
// de la classe B.

void B::affiche(A& a) {
    cout << a.i << endl;
}

int main() {

    A x;
    B y;
    y.affiche(x); // en sortie: 100
    y.change(x);
    y.affiche(x); // en sortie: 200
    return 0;
}
```

- Là aussi, faites attention à l'ordre des déclarations et des définitions.



## 5. Fonction amie de plusieurs classes

```
#include <iostream>

using namespace std;

// déclaration de A. Nous aurions pu déclarer B en premier.
class A;

// définition de B (dans le cas ou la classe B a été déclarée
// en 1er, nous aurions défini A à ce niveau).
class B {
    int j;
public:

    // fonction amie de deux classes A & B.
    friend void affiche(A&,B&);

    // constructeur: j=w, par défaut j=999.
    B(int w=999):j(w){}
};
```

```
// Définition de A (Si la classe B a été déclarée en 1er,  
// nous aurions défini B à ce niveau).  
class A {  
    int i;  
    // Fonctions amies de deux classes A & B.  
    friend void affiche(A&,B&);  
  
public:  
    A(int e=555):i(e) {} // constructeur: i=e, par défaut i=555.  
};  
  
// Définition de la fonction amie affiche.  
void affiche(A& a,B& b) {  
    cout << "A:i " << a.i << endl;  
    cout << "B:j " << b.j << endl;  
}  
  
int main() {  
  
    A x;  
    B y;  
    affiche(x,y); // en sortie  
    return 0;  
}
```

- Là aussi, faites attention à l'ordre des déclarations et des définitions.

## 6. L'amitié n'est ni symétrique ni transitive

- Si la classe A est amie de la classe B  $\nRightarrow$  la classe B est amie de la classe A (pas de symétrie),
- Si la classe A est amie de la classe B **ET** si la classe B est amie de la classe C  $\nRightarrow$  la classe A est amie de la classe C (pas de transitivité).

$\nRightarrow$  (n'implique pas).

- Pour plus de détails, à ce sujet, voir les exercices de démonstration.

## 7. Quand utiliser l'amitié ?

- Dans le cas de l'amitié décrite dans "fonction indépendante, amie d'une classe", assez souvent l'amitié est utilisée quand la fonction manipule plus de deux objets à la fois.
- Sinon, on utilise une fonction membre d'une classe. Pour plus de détails, voir le chapitre suivant: "Surcharge des opérateurs".
  
- Utiliser une fonction membre quand vous pouvez, et une fonction amie quand vous êtes dans l'obligation de le faire.