Prototypes "typiques" des surcharges d'opérateurs

Deux règles à garder en mémoire :
— On déclare au maximum les objets comme *constants*, pour éviter des erreurs dues à une modification d'un objet qui aurait dû rester inchangé.
— On utilise au maximum des références, pour éviter des recopies d'objets qui pourraient être coûteuses (par exemple si les objets considérés manipulent des tableaux de grande taille).

Le tableau suivant liste les principaux opérateurs qui sont naturellement surchargés par des méthodes.

Opérateur	Prototype	Appel	Signification
`=`	`ma_classe& operator= (ma_classe const&);`	`x=y`	`x.operator=(y)`
`+,-`	`ma_classe operator- (void) const;`	`-x`	`x.operator-()`
`+=,*=,/=,...`	`ma_classe& operator+=(ma_classe const&);`	`x+=y`	`x.operator+=(y)`
`++,--`	`ma_classe& operator++(void);`	`++x`	`x.operator++()`
`++,--`	`ma_classe operator++(int);`	`x++`	`x.operator++(0)`
`(),[]`	`type_retour operator() (type_args) const;`	`x(a,b)`	`x.operator()(a,b)`

Surcharge par des méthodes de la classe `ma_classe`

Le tableau suivant liste les principaux opérateurs qui sont naturellement surchargés par des fonctions amies.

Opérateur	Prototype	Appel	Signification
`+,*,/,...`	`ma_classe operator+ (ma_classe const&, ma_classe const&);`	`x+y`	`operator+(x,y)`
`==,!=,<,...`	`bool operator==(ma_classe const&, ma_classe const&);`	`x==y`	`operator==(x,y)`
`<<`	`std::ostream& operator<<(std::ostream &, ma_classe const&);`	`o << x`	`operator<<(o,x)`

Surcharge par des fonctions amies de la classe `ma_classe`

Le mot-clé `const` après le prototype d'une méthode signifie que l'objet attaché à la méthode appelée ne peut pas être modifié, sous peine d'une erreur à la compilation.