



Programmez avec le langage C++

50 hours



Last updated on 3/25/19



Annexes

Ces annexes contiennent quelques informations utiles pour vous aider à comprendre des codes sources que vous aurez trouvés au hasard du web et qui contiendraient des éléments du C++ pas abordés dans le cours.

Liste des opérateurs du C++ ▼

Cette annexe contient la liste des opérateurs du langage C++. Les opérateurs les plus courants ont été vus dans le cours mais il arrive que l'on ait besoin d'autres opérations ou que l'on tombe sur un code source mystérieux au hasard du web. Cette liste complète est là pour vous aider dans ces cas. Il ne sert à rien de l'apprendre par cœur. 😊

Je vous invite à vous en servir comme référence pour avoir le bon prototype lors de la surcharge de vos opérateurs.

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	Ex en ?
Résolution de portée	::	A::x	16	Pas associatif	Non surchargeable	No
Appel de fonction	()	f() f(123)	15	L-R	R C::operator() (...)	Ou

Ex en ?

Nom	Symbole	Syntaxe	Priorité	Associativité	Prototype	Ou
Accès à un élément de tableau	[]	tab[0]	15	L-R	R& C::operator[](A const&)	Ou
					R const& C::operator[](A const&) const	
Accès à membre	.	A.x	15	L-R	Non surchargeable	Ou
Accès à membre via un pointeur	->	A->x	15	L-R	R C::operator->()	Ou
Identification de type	typeid	typeid(A) typeid A	15	L-R	Non surchargeable	No
Incrémantation suffixée	++	x++	15	L-R	R C::operator++(int) R operator++(C&, int)	Ou
Décrémentation suffixée	--	x--	15	L-R	R C::operator--(int) R operator--(C&, int)	Ou
Déréférencement	*	*ptr	14	R-L	R C::operator*() R& operator*(C&)	Ou
Adresse de	&	&x	14	R-L	R C::operator&() R& operator&(C&)	Ou

Ex en ?

Nom	Symbole	Syntaxe	Priorité	Associativité	Prototype	Ou
NON logique	!	!x	14	R-L	bool C::operator!() bool operator! (C const&)	
NON bit à bit	~	~x	14	R-L	bool C::operator~() bool operator~ (C const&)	
Taille en mémoire	sizeof	sizeof(x) sizeof x	14	R-L	Non surchargeable	Ou
Plus unaire	+	+x	14	R-L	R C::operator+ () const R operator+(C const&)	Ou
Moins unaire	-	-x	14	R-L	R C::operator- () const R operator-(C const&)	Ou
Incrémantation préfixée	++	++x	14	R-L	R& C::operator++() R& operator++ (C&)	Ou
Décrémentation préfixée	--	--x	14	R-L	R& C::operator-- () R& operator-- (C&)	Ou
Allocation dynamique	new	new int	14	R-L	void* C::operator new(size_t)	No

Ex
en
?

Nom	Symbole	Syntaxe	Priorité	Associativité	Prototype	
Allocation dynamique de tableau	<code>new[]</code>	<code>new int[4]</code>	14	R-L	<code>void* C::operator new[](size_t)</code>	No
Libération	<code>delete</code>	<code>delete ptr</code>	14	R-L	<code>void C::operator delete(void*)</code>	No
Libération de tableau	<code>delete[]</code>	<code>delete[] ptr</code>	14	R-L	<code>void C::operator delete[](void*)</code>	No
Déréférencement d'un pointeur membre	<code>.*</code>	<code>A.*x</code>	13	L-R	Non surchargeable	No
Déréférencement d'un pointeur membre accédé via un pointeur	<code>->*</code>	<code>A->x</code>	13	L-R	<code>R C::operator ->*(R)</code> <code>R operator ->*(C,R)</code>	No
Multiplication	<code>*</code>	<code>x*y</code>	12	L-R	<code>C C::operator*(C const&)</code> <code>C operator*(C const&, C const&)</code>	Ou
Division	<code>/</code>	<code>x/y</code>	12	L-R	<code>C C::operator/(C const&)</code> <code>C operator/(C const&, C const&)</code>	Ou

Ex
en
?

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	Ou
Modulo	%	x%y	12	L-R	C C::operator% (C const&) const C operator%(C const&, C const&)	Ou
Addition	+	x+y	11	L-R	C C::operator+ (C const&) const C operator+(C const&, C const&)	Ou
Soustraction	-	x-y	11	L-R	C C::operator- (C const&) const C operator-(C const&, C const&)	Ou
Décalage de bit vers la gauche	<<	x<<2	10	L-R	C C::operator<<(C const&) const C operator<<(C const&, C const&)	Ou
Décalage de bit vers la droite	>>	x>>2	10	L-R	C C::operator>> (C const&) const C operator>>(C const&, C const&)	Ou

Ex en ?

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	Ou
Strictement inférieur à	<	x<2	9	L-R	bool C::operator<(C const&) const bool operator<(C const&, C const&)	Ou
Inférieur à	<=	x<=2	9	L-R	bool C::operator<=(C const&) const bool operator<=(C const&, C const&)	Ou
Strictement supérieur à	>	x>2	9	L-R	bool C::operator>(C const&) const bool operator>(C const&, C const&)	Ou
Supérieur à	>=	x>=2	9	L-R	bool C::operator>=(C const&) const bool operator>=(C const&, C const&)	Ou
Est égal à	==	x==2	8	L-R	bool C::operator==(C const&) const bool operator==(C const&, C const&)	Ou

Ex
en
?

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	
Est différent de	<code>!=</code>	<code>x!=2</code>	8	L-R	<code>bool C::operator!=(C const&) const</code> <code>bool operator!=(C const&, C const&)</code>	Ou
ET bit à bit	<code>&</code>	<code>x&y</code>	7	L-R	<code>bool C::operator&(C const&) const</code> <code>bool operator&(C const&, C const&)</code>	Ou
OU exclusif (XOR) bit à bit	<code>^</code>	<code>x^y</code>	6	L-R	<code>bool C::operator^(C const&) const</code> <code>bool operator^(C const&, C const&)</code>	Ou
OU bit à bit	<code> </code>	<code>x y</code>	5	L-R	<code>bool C::operator (C const&) const</code> <code>bool operator (C const&, C const&)</code>	Ou
ET logique	<code>&&</code>	<code>x&&y</code>	4	L-R	<code>bool C::operator&&(C const&) const</code> <code>bool operator&&(C const&, C const&)</code>	Ou

Ex en ?

Nom	Symbole	Syntaxe	Priorité	Associativité	Prototype	Ou
OU logique		x y	3	L-R	bool C::operator (C const&) const const bool operator (C const&, C const&)	ou
Opérateur ternaire	? :	x ? y : z	2	R-L	Non surchargeable	ou
Affectation	=	x=2	1	R-L	R& C::operator=(A const&)	ou
Affectation par addition	+=	x+=2	1	R-L	R& C::operator+=(A const&) R operator+=(C const&, A const&)	ou
Affectation par soustraction	-=	x-=2	1	R-L	R& C::operator-=(A const&) R operator-=(C const&, A const&)	ou
Affectation par multiplication	*=	x*=2	1	R-L	R& C::operator*=(A const&) R operator*=(C const&, A const&)	ou

Ex
en
?

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	Ou
Affectation par division	/=	x/=2	1	R-L	R& C::operator/=(A const&) R operator/=(C const&, A const&)	Ou
Affectation par modulo	%=	x%#=2	1	R-L	R& C::operator%=(A const&) R operator%=(C const&, A const&)	Ou
Affectation par ET bit à bit	&=	x&#=2	1	R-L	R& C::operator&=(A const&) R operator&=(C const&, A const&)	Ou
Affectation par OU exclusif (XOR) bit à bit	^=	x^#=2	1	R-L	R& C::operator^=(A const&) R operator^=(C const&, A const&)	Ou
Affectation par OU bit à bit	=	x =2	1	R-L	R& C::operator =(A const&) R operator =(C const&, A const&)	Ou

Ex en ?

Nom	Symbol	Syntaxe	Priorité	Associativité	Prototype	Ou
Affectation par décalage de bits vers la gauche	<code><<=</code>	<code>x<<=2</code>	1	R-L	<code>R&</code> <code>C::operator<<=(A const&)</code> <code>R operator<<=(C const&, A const&)</code>	
Affectation par décalage de bits vers la droite	<code>>>=</code>	<code>x>>=2</code>	1	R-L	<code>R&</code> <code>C::operator>>=(A const&)</code> <code>R operator>>=(C const&, A const&)</code>	
Opérateur virgule	<code>,</code>	<code>x=2, y=3</code>	0	R-L	<code>R C::operator,(A const&)</code> <code>R operator,(C const&, A const&)</code>	

Quelques explications

Il y a deux colonnes un peu mystérieuses dans ce tableau, celles intitulées "Priorité" et "Associativité". La première indique dans quelle ordre les opérations seront effectuées, ce qu'on appelle l'ordre de préséance. Vous avez certainement appris à l'école qu'en arithmétique on effectue les multiplications avant les additions. C'est la même chose en C++. Certains opérations s'effectuent avant d'autres. Les opérations avec une plus grande priorité seront effectuées avant les autres. Ainsi, la ligne

```
1 a %= ++b*8;
```

correspond à la ligne

```
1 a %= ((++b)*8);
```

Un opérateur surchargé ne change pas de priorité, il garde celle qui lui est donnée par le langage. Ainsi, surcharger l'opérateur `^` pour représenter le produit vectoriel n'est pas

toujours une bonne idée car cette opérateur n'a pas la bonne priorité par rapport à la multiplication (mathématiquement, elles devraient être égales) et peut donc conduire à des situations ambiguës.

L'associativité indique dans quel ordre sont effectuées les opérations de même priorité. C'est-à-dire en allant de gauche à droite (LR = left to right) ou de droite à gauche (RL = right to left). Une suite de divisions comme :

cpp

```
1 a/b/c/d
```

sera interprétée comme valant :

cpp

```
1 ((a/b)/c)/d
```

et non pas comme valant :

cpp

```
1 a/(b/(c/d))
```

ce qui serait le cas si l'opérateur division avait une associativité R-L.

Les prototypes

Les prototypes des surcharges sont donnés à titre d'exemple pour les cas d'utilisation les plus courants. C représente la classe dont on cherche à surcharger les opérateurs, R est le type du résultat (qui peut être le même que C) et A est une autre classe (potentiellement la même que C ou R).

Ainsi, pour la classe Duree , l'opérateur += pour un entier s'écrira:

cpp

```
1 Duree& Duree::operator+=(int secondes); //Methode de classe
```

ou

cpp

```
1 Duree& operator+=(Duree& duree, int secondes); //Fonction libre en-dehors de toute classe
```

La première version étant une méthode de la classe et la deuxième une fonction libre. La première version est celle que nous avons dans le chapitre sur la surcharge d'opérateurs et je vous invite vivement à suivre les recettes données dans ce chapitre pour écrire vos opérateurs surchargés de manière optimale.

La différence entre l'opérateur incrémentation préfixée et suffixée est la présence d'un argument de type int dans la version suffixée. Cet argument est inutilisé dans le corps des fonctions et est uniquement là pour différentier les deux opérateurs.

L'opérateur ternaire

Cet opérateur est un brin spécial dans le sens où c'est le seul qui prend trois arguments. Il permet de remplacer l'instruction `if(){ }else{ }` un peu à la manière de la fonction `SI()` que l'on retrouve dans les tableurs comme Microsoft Excel.

On l'utilise comme suit:

cpp

```
1 int a(2), b(4), c(5);
2
3 int resultat = a > 0 ? b : c;
```

qui se traduit par

cpp

```
1 int a(2), b(4), c(5);
2
3 int resultat;
4 if(a > 0)
5     resultat = b;
6 else
7     resultat = c;
```

C'est une notation plus concise que certains programmeurs adorent mais qui laisse souvent les débutants sur leur faim.

Liste des mots-clés du C++



Cette annexe contient la liste des mots-clés du langage C++. Nous avons vu une grande partie de ces mots-clés dans le cours mais certains plus rares ont été omis. L'important est de savoir qu'il est interdit d'utiliser un de ces mots pour nommer une variable, une fonction, méthode ou classe.

Nom	Courte description
and	Permet de remplacer le symbole <code>&&</code> .
and_eq	Permet de remplacer le symbole <code>&=</code> .
asm	Permet de déclarer une portion de code écrite directement en assembleur.
auto	Permet de déclarer une variable comme locale. La mémoire allouée sera automatiquement désallouée à la fin du bloc. Ce mot-clé est devenu inutile car toutes les variables sont <code>auto</code> par défaut.
bitand	Permet de remplacer le symbole <code>&</code> .
bitor	Permet de remplacer le symbole <code> </code> .

Nom	Courte description
bool	Type de variable contenant une valeur booléenne (valant vrai ou faux uniquement).
break	Instruction permettant de sortir d'une boucle (<code>for</code> ou <code>while</code>) ou d'un <code>switch</code> .
case	Instruction indiquant un des cas d'un <code>switch</code> .
catch	Instruction introduisant un bloc permettant d'attraper des exceptions.
char	Type de variable contenant une lettre.
class	Mot-clé introduisant la définition d'une classe.
compl	Permet de remplacer le symbole <code>\~</code> .
const	Indique qu'une variable ou objet ne peut pas être modifié. Indique qu'une méthode ne changera pas l'état d'un objet.
const_cast	Permet de changer le type d'un objet déclaré <code>const</code> ou <code>volatile</code> en un objet de même type mais pas <code>const</code> .
continue	Instruction permettant de passer à l'itération suivante d'une boucle (<code>for</code> ou <code>while</code>).
default	Instruction indiquant le cas par défaut d'un <code>switch</code> .
delete	Opérateur libérant la mémoire allouée précédemment via <code>new</code> .
do	Déclare une boucle constituée d'une condition finale unique dont le corps est exécuté au moins une fois.
double	Type de variable contenant un nombre à virgule (généralement sur 64 bits).
dynamic_cast	Permet de changer le type d'un objet déclaré d'une classe parent vers enfant ou d'une classe parent à une autre.
else	Introduit une portion de code exécutée si un test <code>if</code> ne renvoie pas <code>true</code> .
enum	Mot-clé introduisant la définition d'un type énuméré.

Nom	Courte description
explicit	Indique que le constructeur doit être appelé explicitement par l'utilisateur.
export	Mot-clé (non utilisé) permettant de séparer le prototype de la définition d'un template .
extern	Indique qu'une variable ou un objet est déclaré dans un autre fichier source.
false	La valeur logique "faux".
float	Type de variable contenant un nombre à virgule (généralement sur 32 bits).
for	Déclare une boucle constituée d'une initialisation, d'une condition finale et d'un incrément.
friend	Mot-clé permettant de déclarer une fonction ou une classe amie d'une autre classe.
goto	Instruction permettant de sauter directement à un autre point du code source.
if	Introduit une structure conditionnelle.
inline	Indique que le corps d'une fonction ou méthode doit être (si possible) copié à l'endroit de l'appel.
int	Type de variable contenant un nombre entier (généralement sur 32 bits).
long	Type de variable contenant un nombre entier (généralement sur 64 bits).
mutable	Indique qu'un attribut peut changer même si l'instance de la classe est déclarée constante.
namespace	Mot-clé introduisant la définition d'un espace de nom.
new	Opérateur allouant de la mémoire et appelant le constructeur d'une variable ou d'un objet.
not	Permet de remplacer le symbole ! .
not_eq	Permet de remplacer le symbole != .

Nom	Courte description
operator	Permet de déclarer une surcharge d'opérateur.
or	Permet de remplacer le symbole .
or_eq	Permet de remplacer le symbole = .
private	Droit d'accès aux éléments d'une classe interdisant l'accès depuis l'extérieur de la classe.
protected	Droit d'accès aux éléments d'une classe interdisant l'accès depuis l'extérieur de la classe sauf pour les classes héritées.
public	Droit d'accès aux éléments d'une classe autorisant l'accès depuis l'extérieur de la classe.
register	Indique qu'une variable ou un objet doit être stockée dans les registres du processeurs si possible.
reinterpret_cast	Permet de changer le type d'un objet déclaré en réinterprétant les données brutes bit par bit.
return	Indique la valeur qu'une fonction doit renvoyer.
short	Type de variable contenant un nombre entier (généralement sur 16 bits).
signed	Indique qu'un type peut contenir des valeurs positives et négatives (comportement par défaut).
sizeof	Opérateur renvoyant la taille en octets d'un type.
static	Permet de déclarer un attribut ou une méthode comme étant statique.
static_cast	Permet de changer le type d'un objet déclaré vers un autre objet si les types sont compatibles.
struct	Mot-clé introduisant la définition d'une structure (une classe où tout est public par défaut).
switch	Introduit une structure conditionnelle avec plusieurs sorties possibles selon la valeur de l'expression testée.
template	Mot-clé introduisant la définition d'un patron de classe ou de fonction.

Nom	Courte description
this	Pointeur vers l'objet courant.
throw	Instruction lançant une exception.
true	La valeur logique "vrai".
try	Instruction introduisant un bloc sensible aux exceptions.
typedef	Mot-clé introduisant la définition d'un alias de type.
typeid	Opérateur renvoyant des informations (nom, ...) sur un objet.
typename	Permet de décrire un type indéterminé dans le cadre d'une fonction ou d'une classe template.
union	Mot-clé introduisant la définition d'une union (structure où les attributs utilisent la même adresse mémoire).
unsigned	Indique qu'un type peut contenir des valeurs positives seulement.
using	Permet d'indiquer que l'on utilise les objets et variables d'un espace de nom sans avoir à le spécifier à chaque utilisation. Permet de démasquer une méthode masquée dans une classe fille.
virtual	Permet de déclarer une méthode comme étant virtuelle. Permet de définir un héritage virtuel dans le cas de l'héritage multiple.
void	Type de retour d'une fonction ne renvoyant rien.
volatile	Indique qu'une variable peut changer à tout moment pour des raisons externes au programme.
wchar_t	Type de variable contenant une lettre généralisée (UTF-8 ou autre).
while	Déclare une boucle constituée d'une condition finale uniquement.
xor	Permet de remplacer le symbole <code>^</code> .
xor_eq	Permet de remplacer le symbole <code>^=</code> .

Que pensez-vous de ce cours ?



I FINISHED THIS CHAPTER



CE QUE VOUS POUVEZ ENCORE
APPRENDRE

Teachers

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Matthieu Schaller

Chercheur en astrophysique et cosmologie. Spécialiste en simulations numériques de galaxies sur superordinateurs.

Check out this course via



Book



PDF

OpenClassrooms

What we do

Apprenticeship

Our blog

Work at OpenClassrooms

Business solutions

Employers

Learn more

Become a mentor

Help and FAQ

Terms of Use

Privacy Policy

Contact us

 English

