



2^e année – Spécialité Informatique

Année 2024 – 2025

Programmation orientée objet en C++

Notes de cours

Sébastien Fourey

Chapitre 8. Épilogue, Bibliographie, Index

Version du 3 septembre 2024



Ce travail est publié sous licence Creative Commons
Paternité – Pas d'utilisation commerciale – Pas de modification

Chapitre 8

Épilogue



Fig. 5. — Haricot qui a germé *en pleine lumière*. Il est vert, son poids a augmenté : il a vécu sur le carbone qu'il a pris à l'acide carbonique de l'air.

Nous terminons ce document par la présentation du moyen d'obtenir, en C++ , des informations propres à une implémentation particulière. Une deuxième courte section est consacrée à des mots-clés qui existent pour des raisons essentiellement historiques et que nous qualifions ici d'exotiques.

8.1 Données propres à l'implémentation

En C++ des données concernant les types de base sont fournies par l'en-tête `<limits>`. Pour ce faire, chaque implémentation définit ses propres *spécialisations* d'un modèle de structure, le modèle `numeric_limits<>`. Celui-ci définit finalement les *traits*¹ des types de base du langage (voir listing 8.1). donne un aperçu du modèle général.

Exemples

Avec le compilateur GNU g++ (GCC) 3.2.1 sur une architecture SPARC™ :

L'instruction

```
|| cout << numeric_limits<unsigned long long>::max();
```

affiche :

```
|| 18446744073709551615
```

D'autre part, l'instruction

```
|| cout << numeric_limits<unsigned long long>::digits;
```

affiche

```
|| 64
```

Exercice 8.1 *Quelle est la relation entre les deux entiers qui viennent d'être « affichés » ?*

1. La notion de *trait* (c.-à-d. caractéristique), basée sur la spécialisation de modèle de classe ou de structure, est normalement abordée par l'exemple en cours, dans le cadre de la programmation générique (§ 3.3).

```

1  template <typename T>
2  struct numeric_limits {
3      static const bool is_specialized;
4      static const int digits = ? ? ? ;
5      static const bool is_signed = ? ? ? ;
6      static const bool is_integer = ? ? ? ;
7      static const bool is_exact = ? ? ? ;
8      static const int radix = ? ? ? ;
9      static const int digits = ? ? ? ;
10     static const int digits10 = ? ? ? ;
11     static const bool is_modulo = ? ? ? ;
12     static const bool has_infinity;
13     inline static T min_exponent = ? ? ? ;
14     inline static T max_exponent = ? ? ? ;
15     inline static T min() throw();
16     inline static T max() throw();
17     inline static T epsilon() throw();
18     inline static T infinity() throw();
19     // ...
20 };

```

Listing 8.1 – Extrait de la définition du modèle `numeric_limits<>`.

Notez que le type « long long » est apparu avec la norme de 2011 du langage, même s’il était disponible auparavant avec certains compilateurs comme g++.

8.2 Exotisme, les synonymes d’opérateurs

Plusieurs opérateurs du langage ont des synonymes sous la forme de mots-clés (qui sont donc des noms réservés).

Opér.	Mot-clé	Opér.	Mot-clé	Opér.	Mot-clé
&&	and	&	bitand	&=	and_eq
	or		bitor	=	or_eq
!	not	~	compl	~=	not_eq
		^	xor	^=	xor_eq

TABLE 8.1 – Synonymes d’opérateurs.

Le listing 8.2 montre que ces mots-clés, qui restent d’un usage marginal, mériteraient peut être d’être utilisés plus souvent puisqu’ils apportent, pour un lecteur peu familier du langage, un certain gain d’« intelligibilité ». Il existe aussi des équivalents pour certaines unités lexicales (cf. tableau 8.2). Il ne s’agit pas d’opérateurs, et à la différence des synonymes précédents leur utilisation a tendance à obscurcir le code pour le programmeur non averti.

Symbole	Équiv.	Symbole	Équiv.
{	<%	}	%>
[<:]	:>
#	%:	##	%::%

TABLE 8.2 – Synonymes d'unités lexicales.

```

1 struct Knowledge {};
2 struct Teacher {
3     void increases(Knowledge & k);
4 };
5 struct Language {
6     bool isGoodToKnow() {
7         return true;
8     }
9     Language & operator++();
10    Language operator++(int);
11 };
12 struct Reading {
13     bool isGood() {
14         return true;
15     }
16 };
17 Reading theLecture() {
18     return Reading();
19 }
20 bool youKnow(Language language);
21 namespace Your
22 {
23     int Help;
24 }
25
26 int main() {
27     Knowledge yourKnowledge;
28     Teacher sebastienFourey;
29     Language c;
30
31     if ((youKnow(c++) or theLecture().isGood())
32         and (c++).isGoodToKnow()) {
33         using Your::Help;
34         sebastienFourey.increases(yourKnowledge);
35     }
36 }

```

Listing 8.2 – Un code correct et très lisible, mais plutôt vers la fin.

Remerciements

Les personnes citées ci-après ont contribué à l'amélioration de la qualité de ce support par leurs suggestions de corrections ou simples modifications.

- Nicolas Signolle, attaché temporaire à l'enseignement et à la recherche à l'ENSICAEN en 2007-2008 ;
- Christine Porquet, maître de conférences à l'ENSICAEN ;
- Loïc Simon, maître de conférences à l'ENSICAEN ;
- Yoran Le Bagousse, étudiant en spécialité informatique à l'ENSICAEN (promotion 2014) ;
- Charles Chaudet, étudiant en spécialité informatique à l'ENSICAEN (promotion 2019).
- Bastien Hubert, étudiant en spécialité informatique à l'ENSICAEN (promotion 2021).
- Julien Zaïdi, étudiant en spécialité informatique à l'ENSICAEN (promotion 2022).

Tout élève est invité à mettre en défaut le contenu de ce document pour voir apparaître son nom dans les versions futures. En effet, l'auteur ne peut prétendre qu'approcher (modestement) de très loin le niveau d'exactitude du créateur du langage. D'ailleurs, Bjarne Stroustrup lui-même commet des erreurs comme le confirme la page d'errata de la 3^e édition du livre de référence [8] dont l'URL est donnée ci-dessous.

cf. <http://www.informit.com/content/images/0201889544/errata%5C833.pdf>

Conclusion

Beaucoup de sujets n'ont pas été abordés dans ce document. Certains sont traités lors des cours magistraux, d'autres sont rencontrés à l'occasion des travaux pratiques. Les exercices proposés dans les différents chapitres devraient aussi avoir initié quelques recherches personnelles. À titre de document de référence complet sur le langage, et en dehors de la norme elle-même [4], le livre de Bjarne Stroustrup [11] est sans doute le plus complet.

Le support s'achèvera par une citation de M. Clouard, enseignant-chercheur en informatique à l'ENSICAEN depuis 1999.

« Si tu veux savoir programmer en C++, il faut que tu pratiques en permanence ; y compris les week-ends car sinon le lundi, tu ne sais plus rien faire avec. »

Régis Clouard ²

2. Entretien avec l'auteur, Été 2006.

Bibliographie

- [1] Paul BERT. *La première année d'enseignement scientifique (Sciences naturelles et physiques)*. Librairie classique Armand Colin et C^{ie}, Paris, 1882. 550 gravures.
- [2] Michel DESVIGNES. *Programmation objet en C++, notes de cours*. ENSICAEN, M^{me} Mullois Édition, 2002.
- [3] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Programming languages – C++*. International standard ISO/IEC 14882:2003, Octobre 2003. Remplace 14882:1998.
- [4] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Information technology – Programming languages – C++*. International standard ISO/IEC 14882:2011, Mars 2011. Remplace 14882:2003.
- [5] Ray LISCHNER. *STL – Précis & concis*. O'Reilly, 2004.
- [6] Scott MEYERS. *Programmer efficacement en C++*. Dunod, 2016.
- [7] STANDARD C++ FOUNDATION. C++ FAQ, Exceptions and Error Handling [Page Web]. <https://isocpp.org/wiki/faq/exceptions#ctors-can-throw>, 2016. [visité en octobre 2022].
- [8] Bjarne STROUSTRUP. *The C++ programming language, third edition*. Addison Wesley Publishing Company, 1997.
- [9] Bjarne STROUSTRUP. *C++*. CampusPress France, 1999. Troisième édition du livre de Bjarne Stroustrup, traduit de l'américain par Christine Eberhardt.
- [10] Bjarne STROUSTRUP. Sibling Rivalry: C and C++. AT&T Labs – Research Technical Report TD-54MQZY, http://www.stroustrup.com/sibling_rivalry.pdf, Janvier 2002. [visité en octobre 2022].
- [11] Bjarne STROUSTRUP. *The C++ Programming Language, fourth edition*. Pearson Education, 2013.
- [12] Bjarne STROUSTRUP. Bjarne Stroustrup's C++11 FAQ [Page Web]. <http://www.stroustrup.com/C++11FAQ.html>, 2014. [visité en octobre 2022].
- [13] Bjarne STROUSTRUP. Bjarne Stroustrup's FAQ [Page Web]. http://www.stroustrup.com/bs_faq.html#really-say-that, 2014. [visité en octobre 2022].

Index

- (
 surcharge, 61
- , ++
 surcharge, 59
- >
 surcharge, 61
- ::
 espaces de noms, 15
 résolution de portée, 14
- =
 opérateur d'affectation, 44
 opérateur d'affectation par déplacement, 48
- []
 surcharge, 59
- algorithmes, *voir* STL, algorithmes
- allocation dynamique sur le tas, 25
- arguments
 valeurs par défaut, 13
- arité des opérateurs et surcharge, 57
- Bibliothèque standard, 111
 - algorithmes, **125**
 - C++11, 130
 - modification de séquence, 125
 - numériques, 130
 - opérations ensemblistes, 129
 - séquences triées, 128
 - conteneurs ordonnés, 116
 - exemples d'utilisation, 118
 - `iterator_traits`, 123
 - itérateurs, 122
 - d'insertion, 138
 - objets fonctions, 130
 - prédicats, 135
- `binary_function`, 132
- `bind2nd`, éditeur de liaison (bibliothèque standard), 135
- boucle `for`, 14
- C++11
 - =, affectation par déplacement, 48
 - algorithmes, nouveaux \sim , 130
 - `std::bind`, 161
 - boucle `for` et itérateurs, 156
 - constructeur par déplacement, 47
 - conteneurs, 114
 - `default` et `delete`, 50
 - délégation de constructeur, 39
 - `extern template`, 97
 - fonctions anonymes, 156
 - fonctions *lambda*, 156
 - `std::function`, 159
 - héritage explicite des constructeurs, 71
 - inférence de type, 155
 - initialisation des données membres, 39
 - initialisation uniforme, 149
 - liste d'initialiseurs, 149
 - `long long`, 10
 - `new` et initialisation, 40
 - `noexcept`, 144
 - opérateur de conversion explicite, 63
 - `override` et `final`, 85
 - pointeur de fonction généralisé, 159
 - pointeurs intelligents, 166
 - rvalue reference*, 47
 - `shared_ptr`, 173
 - `tuple`, 118
 - type de retour suffixé, 163
 - `unique_ptr`, 170
 - `weak_ptr`, 183
- `catch`, *voir* exceptions
- classe
 - abstraite, 84
 - amie, *voir* `friend`
- commentaires, 10
- `composer1`, `compose1`, 137
- `const`
 - méthodes constantes, 54
 - références constantes, 24
- constructeur, 33
 - \sim et conversion implicite, 37
 - constructeur par déplacement, 47
 - constructeur par recopie, 46

- délégation de \sim , 39
- `explicit`, 37
- héritage, 71
- héritage explicite des \sim , 71
- conteneurs, *voir* Bibliothèque standard
- conversion
 - explicite
 - `dynamic_cast`, 79
 - `reinterpret_cast`, 79
 - `static_cast`, 78
 - implicite
 - grâce au constructeur, 37
 - opérateur de conversion, 61
- `copy` (algorithme de la bibliothèque standard), 127
- `default`
 - modificateur, 50
- `delete`
 - désallocation, 27
 - modificateur, 50
- destructeur, 40
 - héritage, 71
- `dynamic_cast`, 79
- en-têtes standard
 - syntaxe, 17
 - `<fstream>`, 107
 - `<ios>`, 104
 - `<istream>`, 107
 - `<ostream>`, 106
 - `<sstream>`, 108
 - `<string>`, 108
- espaces de noms, 15
 - anonymes, 19
- exceptions, 141
 - spécifier des \sim , 144
 - spécifier des \sim en C++11, 144
 - sim* standard, 145
 - `try`, *catch*, 143
- `explicit`
 - constructeur, 37
 - opérateur de conversion \sim , 63
- `extern`, 19
- `final`, 85
- fonctions
 - amies, 54
 - anonymes, 156
 - `std::function`, 159, 161
 - lambda*, 156
 - modèles de \sim , 100
- `for`
 - boucle, 14
 - nouvelle syntaxe C++11, 156
- `friend`
 - classe amie, 56
 - fonction amie, 54
- héritage, 65
 - \sim et conversions, 71
 - explicite des constructeurs, 71
 - multiple, 86
 - et dérivation virtuelle, 90
- inférence de type, 155
- initialisation uniforme, 149
- itérateurs, *voir* Bibliothèque standard, itérateurs
- `less`, prédicat de la bibliothèque standard, 135
- limites, *voir* `numeric_limits`
- liste d'initialiseurs, 149
- `long long`, 10
- `lvalue`, 14
- `mem_fun`, appelant de méthode (bibliothèque standard), 137
- modèles, 93
 - de classes, 94
 - et fonctions amies, 98
 - spécialisation, 98
 - de fonctions, 100
 - surcharge, 102
- méthodes
 - constantes, 54
 - implémentation, 74
 - statiques, 33
 - virtuelles, 82
 - virtuelles pures, 84
- namespace, 15
- `new`, 25
- `numeric_limits`, 11, 191
- objets
 - constructeur par recopie, 46
 - contrôle d'accès, 31
 - et héritage, 68
 - destructeur, 40
 - instances de classe, 29
 - objets temporaires

- création, 38
- références sur les \sim , *voir* réf. constantes
- opérateurs
 - synonymes d' \sim , 192
- opérateur
 - =, affectation, 44
 - =, affectation par déplacement, 48
 - chaînage (exemple avec `operator>>`), 23
 - \sim de conversion, 61
- opérateur de conversion explicite, 63
- override, 85
- pair, modèle des paires d'éléments, 116
- partage de données, 44
- pointeur
 - notion de propriété, 166
- pointeurs intelligents, 166
- private
 - dérivation, 68
 - spécificateur d'accès, 31
- protected
 - dérivation, 68
 - spécificateur d'accès, 31
- prédicats, *voir* Bibliothèque standard, prédicats
- ptr_fun, fabricant d'objets fonctions, 138
- public
 - dérivation, 68
 - spécificateur d'accès, 31
- reinterpret_cast, 79
- rvalue, 14
- rvalue reference
 - affectation par déplacement, 48
 - constructeur par déplacement, 47
 - `std::move`, 49
- règle des trois, 47
- références, **21**
 - argument de fonction, 22
 - retour de fonction, 23
 - \sim constantes et objets temporaires, 24
- shared_ptr, 173
- static
 - donnée membre, 33
 - et unité de compilation, 19
 - méthode, 33
- static_cast, 78
- std::move, 49
- STL
 - adaptateurs, 116
 - conteneurs séquentiels, 115
- surcharge
 - d'un modèle de classe (impossible), 96
 - d'un modèle de fonction, 102
 - de fonctions, 20
 - des opérateurs, 57
 - des opérateurs
 - listes des opérateurs surchargeables, 57
 - synonymes d'opérateurs, 192
- tailles des types de base, 10
- template, *voir* modèles
- this, 74
- throw, 143–145
- traits
 - iterator_traits, 123
- trois
 - règle des \sim , 47
- try, *voir* exceptions
- tuple, 118
- type de retour suffixé, 163
- typeid, 77
- unary_function, 132
- unique_ptr, 170
- using, déclaration, 15
- using namespace, directive, 15
- weak_ptr, 183