PROGRAMMEZ!

Le magazine des développeurs

04/2020 N°239 22e année

Interview exclusive

Bjarne Stroustrup

le créateur de C++

Informatique quantique

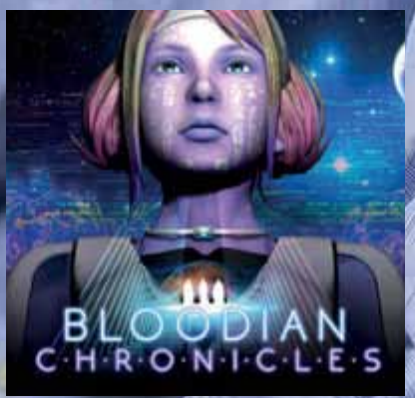
Q# / QISKIT

Écologie

Être un dev responsable

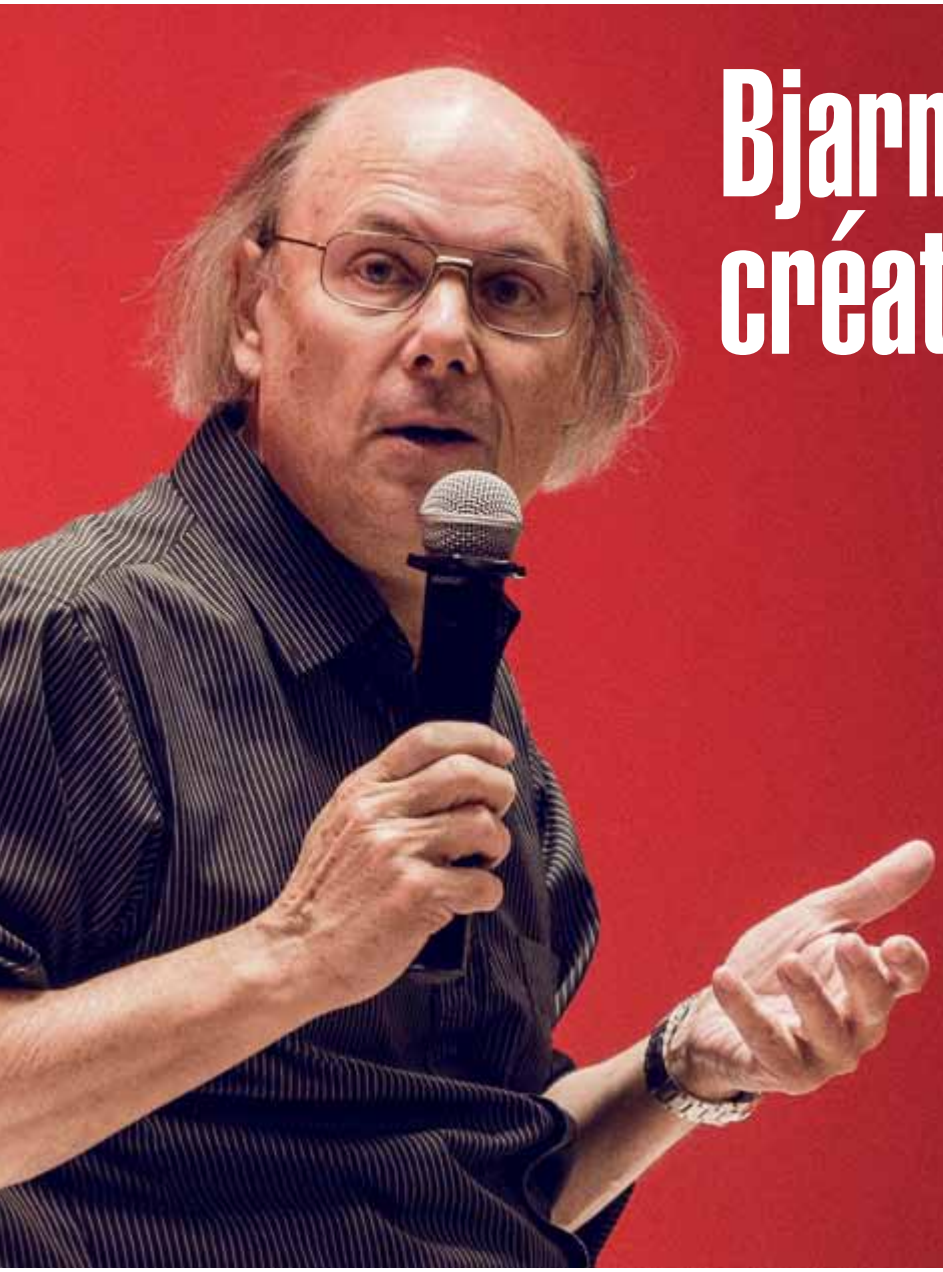
Captain Blood

Renaissance d'un jeu mytique



Le seul magazine écrit par et pour les développeurs

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH



Bjarne Stroustrup créateur de C++

Depuis plusieurs mois, Programmez! souhaite créer une nouvelle série d'articles, et plus spécifiquement, des interviews des créateurs de langages. Pour inaugurer cette rubrique, nous vous proposons une interview exceptionnelle du créateur de C++ : Bjarne Stroustrup. Christophe Pichaud, contributeur C++ bien connu dans le magazine, s'est longuement entretenu avec Bjarne le 24 février dernier, quelques jours après la disponibilité officielle de C++ 20.

Christophe PICHAUD
Lead Software Architect chez Infeeny
christophep@cpixxi.com | www.windowscpp.com

Comment naît un langage de programmation ? Comment avez-vous créé C++ ? Expliquez-nous son histoire.

Bjarne Stroustrup : C++, comme la plupart des langages que j'aime le mieux, a commencé avec un besoin, un problème. Je ne voulais pas particulièrement concevoir un langage de programmation. Je voulais construire un système, mais j'ai décidé que je ne pouvais pas construire ce système - une version d'Unix pour fonctionner comme un système distribué - avec les langages existants. J'avais besoin d'un langage capable de faire des choses de bas niveau comme des gestionnaires de mémoire, des pilotes de périphériques et des planificateurs de processus. J'avais aussi besoin d'un langage qui pouvait faire des choses de haut niveau comme spécifier les composants d'un programme et comment ils communiquaient. Aucun langage ne pouvait faire les deux, j'ai donc construit C++. Je n'ai jamais pu construire mon système distribué, mais d'autres ont construit de tels systèmes en C++.

Nous sommes une communauté de plusieurs millions de développeurs C++, en Amérique du Nord, en Australie, en Europe, en Asie. Nous avons traversé les époques et nous ne sommes pas là par hasard. Nous sommes des survivants. Même quand Java est sorti, nous avons survécu et ce sans marketing, sans organe de presse.

C++ doit être bon dans l'abstraction et dans la génération du code, le code bas-niveau, bon dans la gestion du hardware. Il permet de faire aussi bien du C que des templates. C++ n'est pas un full langage OOP au sens strict du terme, cependant il comporte de nombreuses fonctionnalités qui en font un bon candidat.

Les MFC n'utilisaient pas la STL. Dans mon apprentissage, j'ai appris le C++ puis la STL. Dans les MFC, il y a avait des macros car le compilateur ne supportait toutes les fonctionnailités

des templates. Comment avez-vous réagi quand vous avez vu la première version de la STL écrite par Stepanov pour HP ?

Quand j'ai vu la STL pour la première fois, nous avons discuté et j'ai trouvé cela bizarre. Puis, j'ai fait le compte de tout ce qui était utilisé comme fonctionnalités. Tout y était utilisé. Les templates sont complexes mais d'une puissance formidable. Au fur et à mesure que je découvrais son travail, je lui ai demandé de proposer la STL au comité ISO.

Êtes-vous surpris par le succès du C++ ?

Bjarne Stroustrup : Oui définitivement. J'étais - et je suis toujours - assez étonné. Il n'y a jamais eu un moment où C++ n'ait pas eu plusieurs concurrents mieux financés et fortement commercialisés. De nombreux aspects de la conception n'étaient pas à la mode, juste des réponses à des défis réels. Pourtant, la communauté C++ a grandi et continue de croître. C++ répond à des besoins réels.

C++ est une norme ISO. Est-ce important qu'il le soit ? Est-ce une garantie de maintenir un écosystème et d'assurer le fonctionnement sur plusieurs plateformes ?

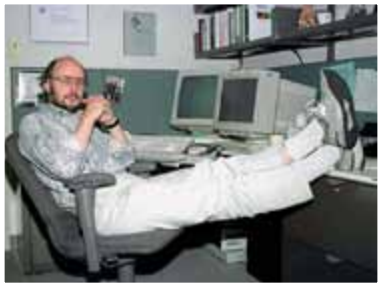
Bjarne Stroustrup : Pour C++, le comité de normalisation ISO (WG21) est essentiel. Nous n'avons pas de riche propriétaire, et, pour collaborer, les organisations concurrentes ont besoin d'un forum ouvert comme le WG21. Où des personnes d'organisations concurrentes peuvent-elles s'asseoir et résoudre ensemble des problèmes si ce n'est pas à la WG21 ? Avoir un comité des normes ne garantit rien. Seule la qualité de la norme donne le succès. Il est crucial que la norme soit basée sur le consensus, afin que tous les fournisseurs de compilateurs et de bibliothèques de normes trouvent utile de s'y conformer. A Prague, le vote pour le C++ 20 était de 79-0 en sa faveur! Cela signifie que C++ 20 deviendra officiel à l'automne. La plupart des parties de C++ 20 sont déjà livrées dans plusieurs compilateurs et une grande partie est déjà utilisée en production. C++ 20 n'est donc pas de la science-fiction.

Cette idée d'associer le C++ à un comité ISO vous est venue comment ?

Je n'ai pas eu l'idée. IBM et HP m'ont dit de faire cela, car ils ne faisaient pas confiance à mon employeur (AT&T). Il fallait donc que je fasse cet effort de standardisation, car ils avaient des milliards en investissement sur C++ ; je n'avais pas le choix. Au début je ne savais pas trop ; le langage n'était pas complètement terminé en termes de design. Cela prend des années pour concevoir un langage. Puis je me suis rendu compte que c'était une bonne idée.

Le C++ a évolué avec de nombreuses fonctionnalités, est-ce une bonne chose ?

On pioche dans différents langages et il y a du fonctionnel, du procédural, de l'orienté objet, etc. Il n'y a pas de langage parfait et unique. On peut toujours programmer comme le C++ à la Papa et aussi avec le style Moderne.



Il est difficile d'apporter des améliorations à un langage de programmation. Nous l'avons vu avec Sun et Oracle pour Java. Le cas Kotlin faisant courir le risque de casser la compatibilité. Quel est votre sentiment avec C++ ?

Bjarne Stroustrup : La stabilité est une caractéristique importante, en particulier pour les personnes qui construisent des systèmes qui doivent durer des décennies. Le C++ est devenu un langage beaucoup plus expressif et performant que ce que j'avais dans les années 80, mais nous avons fait très attention à maintenir un grand degré de compatibilité. Tout le monde veut un langage plus petit et plus simple. Ils veulent également plus de fonctionnalités. De plus, la plupart des développeurs C++ insistent pour que leur ancien code ne soit pas cassé. Il est impossible de livrer les trois. J'essaie de maintenir la compatibilité tout en ajoutant des moyens de simplifier l'utilisation de C++.

C++ a évolué au fil des ans avec une croissance rapide au cours de la dernière décennie. C'est une bonne chose ?

Bjarne Stroustrup : Oui. De nouveaux défis émergent et de nouvelles idées sont nécessaires pour les relever. Une langue change pour s'adapter à un monde en évolution. Cela est vrai pour le langage naturel et artificiel. De plus, j'ai toujours su que le C++ n'était pas encore le mieux adapté à mes idéaux. Par exemple, en 1981, j'ai écrit qu'une programmation générique était nécessaire. J'ai supposé que les macros pouvaient être utilisées pour répondre à ce besoin, mais cela ne s'est pas mis à l'échelle, j'ai donc implémenté des modèles. Cependant, les modèles ne rencontraient qu'une partie de mes idéaux pour la programmation générique. Ce n'est qu'avec des «concepts» que nous nous rapprochons.

Un autre exemple est la concurrence et le parallélisme où nous avons dû normaliser un niveau de support de threads et de verrous fortement typé. En tant que langage de programmation de systèmes, C++ devait fournir un support direct à ce que les systèmes d'exploitation offraient. Ensuite, nous devons offrir un meilleur support standard pour la programmation sans verrouillage, et ce n'est que plus tard que nous pourrons aborder les modèles de plus haut niveau d'algorithmes simultanés et parallèles. C++ 20 fournit des algorithmes parallèles et C++ 23 est susceptible de fournir un modèle général de concurrence.

Au cours de l'année, C++ a également fourni de nombreuses fonctionnalités pour simplifier la programmation. «Simplifiez les choses simples!» est un commentaire courant. Dans le C++ moderne, nous devons écrire beaucoup moins de code pour gérer les ressources, pour exprimer des algorithmes généraux, pour écrire des boucles simples, etc., que dans les anciennes versions de C++. Nous ne pouvons pas simplifier le langage sans casser le code, mais nous pouvons simplifier l'utilisation du langage.

Vous avez reçu de nombreuses récompenses et divers honneurs. Quelle est la récompense dont vous êtes le plus fier ?

Bjarne Stroustrup : Le prix Charles Stark Draper de la US National Academy of Engineering (https://en.wikipedia.org/wiki/Charles_Stark_Draper_Prize). C'est l'une des plus hautes distinctions honorifiques au monde

pour un ingénieur. Il a été fondé pour compenser le fait qu'il n'y a pas de prix Nobel d'ingénierie. Très peu d'informaticiens ont reçu «The Draper». C'est pour tous les domaines de l'ingénierie. Les ingénieurs ont toujours apprécié mon travail et je suis fier d'avoir fait quelque chose d'utile à grande échelle.

Pouvez-vous nous donner votre sentiment lorsque les jeunes développeurs considèrent le C++ comme un vieux matériel hérité par rapport à Rust, Kotlin ?

Bjarne Stroustrup : Tout le monde aime la nouveauté et nous aimerions tous ignorer les complexités désordonnées du monde réel, dont beaucoup sont liées au travail passé. Je suis quelque peu amusé quand quelqu'un m'accuse d'emprunter quelque chose à Rust que j'ai inventé des décennies plus tôt. Chaque langue réussie deviendra «héritée» et devra faire face à des idées plus anciennes et à un code plus ancien. L'alternative est l'échec.

Quelles sont les fonctionnalités les plus importantes pour vous ?

Bjarne Stroustrup : La fonctionnalité unique la plus importante en C++ est la classe avec des constructeurs et des destructeurs pour gérer la gestion des ressources. C'est une fonctionnalité qui distingue C++ de la plupart des autres langages et elle est fondamentale pour la programmation C++. Après cela, je vais pointer vers des modèles avec des concepts pour prendre en charge la programmation générique. C'est généralement une erreur de se concentrer sur les fonctionnalités de chaque langue. Je caractérise souvent le C++ comme ceci :

- Un système de type statique avec un support égal pour les types intégrés et les types définis par l'utilisateur ;
- Sémantique de valeur et de référence ;
- Gestion systématique et générale des ressources (RAII) ;
- Prise en charge d'une programmation orientée objet efficace ;
- Prise en charge d'une programmation générique flexible et efficace ;
- Prise en charge de la programmation au moment de la compilation ;
- Utilisation directe des ressources de la machine et du système d'exploitation ;
- Prise en charge de la concurrence par le biais de bibliothèques (si nécessaire implémentée en utilisant intrinsèques).

C'est pour C++, bien sûr. D'autres langages sont conçus pour différents domaines d'applications et différentes populations de développeurs, ils ont donc des critères de conception différents.

SECONDE PARTIE DE L'INTERVIEW

Au-delà des questions génériques, nous avons posé des questions plus spécifiques sur le langage et les fondations de C++.

La sémantique de déplacement est-elle une bonne chose (et qu'en est-il du flou &&) ?

Bjarne Stroustrup : Déplacer la sémantique est une très bonne chose dans la mesure où elle complète le modèle de gestion des ressources de C++ en permettant aux ressources d'être déplacées d'une étendue à l'autre, avec peu ou pas de surcharge, et sans tripoter les pointeurs ou la gestion explicite des ressources. Cela simplifie considérablement le code et complète les développements commencés par l'optimisation de la valeur de retour (que j'ai implémentée en 1984). J'utilise presque exclusivement la sémantique de déplacement implicitement dans les instructions de retour où elle est sûre et implicite une fois que vous avez défini les constructeurs de déplacement et les affectations de déplacement. Je suis très méfiant à propos de l'utilisation explicite de && et std :: move () car une telle utilisation est sujette aux erreurs et généralement inutile. Toujours au cœur de C++; exprimez ce que vous pensez!

Avons-nous besoin d'améliorations des classes ?

Bjarne Stroustrup : Les classes avec constructeurs et destructeurs (si nécessaire) sont au cœur de C++. Je pense qu'ils sont assez bons et n'ont pas besoin d'améliorations significatives.

Pouvons-nous nous présenter les concepts de C++ 20 ?

Bjarne Stroustrup : Quand j'ai conçu les modèles, je voulais trois choses :

- Généralités - pour permettre aux gens de faire plus que ce que j'imaginais ;
- Frais généraux zéro ;
- Interfaces précisément spécifiées.

Je ne savais pas comment faire les trois en même temps, donc nous n'avons eu que les deux premiers. Cependant, les deux premiers ont suffi pour que les modèles deviennent un succès majeur. Ils ont en revanche également causé beaucoup de problèmes et de confusion. «Concepts» nous donne ce dernier point : des interfaces spécifiées avec

précision sans surcharges d'exécution ni restrictions sur ce que vous pouvez exprimer. C'est simplement une logique de prédicat au moment de la compilation sur les propriétés des types et des valeurs.
Enfin, nous pouvons dire juste : trier (v); et vous obtenez un message d'erreur décent si v n'est pas triable. Nous pourrions déclarer sort () comme ceci :
void sort (sortable_range auto &);
Cette auto est redondante, mais de nombreux membres du comité des normes ont estimé que le laisser de côté serait source de confusion, car ils ne seraient pas en mesure de voir que sort () était un modèle.

Les coroutines sont au cœur de C++ 20, que faut-il en retenir ?
Bjarne Stroustrup : Les coroutines faisaient partie de la conception originale de C++. De plus, la toute première bibliothèque C++ (lorsque C++ était encore appelée «C avec classes») fournissait des coroutines. Je suis très content de les revoir. Les coroutines simplifient de nombreuses formes de code autrement compliquées en permettant la sauvegarde automatique de l'état d'un calcul entre les calculs. La représentation d'exécution des coroutines C++ 20 est très petite et efficace de sorte qu'elles peuvent être utilisées pour gérer des centaines de milliers de calculs asynchrones dans un seul programme. Ils sont déjà largement utilisés sur Facebook et Microsoft.

Avons-nous besoin d'améliorations sur les classes dérivées?
Bjarne Stroustrup : Non, je ne pense pas que nous devons ajouter aux mécanismes de classes dérivées.

Quand le constructeur fait appel à Close() alors que vous l'aviez oublié…. Qu'en pensez-vous ?
Bjarne Stroustrup : Probablement ma fonctionnalité C++ préférée. C'est la clé de la gestion générale des ressources. La gestion de la mémoire ne suffit pas. Nous devons gérer les ressources autres que la mémoire telles que les descripteurs de fichiers, les verrous et les connexions à la base de données.

Je ne suis pas un gars d'exception mais j'aime tout attraper… (…) Et les exceptions ?
Bjarne Stroustrup : Eh bien, je suis "un gars d'exception" et ça ne me dérange pas "d'attraper (...)". L'exception utilisée avec RAII simplifie énormément le code et élimine des classes entières d'erreurs à des coûts très mineurs. Parfois, il fournit des accélérations par rapport au code jonché de tests de codes d'erreur. Il empêche les flux de contrôle exceptionnels de compliquer le code source.
La plupart des problèmes avec les exceptions proviennent de personnes qui utilisent try-catch simplement comme une forme si-alors-sinon ou dans une base de code qui est un tel gâchis de pointeurs et de gestion manuelle des ressources que les exceptions deviennent une source d'erreurs et de coûts d'exécution. Une autre classe de problèmes survient dans les systèmes qui n'essaient pas de détecter les erreurs et dans les petits systèmes où toute forme de prise en charge à l'exécution peut évincer les fonctionnalités nécessaires.

for-range est assez cool. Dites-en nous plus.
Bjarne Stroustrup : Que dire de plus? Traverser une plage est l'une des actions les plus courantes dans notre code, nous devons donc avoir un moyen simple et relativement sûr de l'exprimer. La plupart des erreurs courantes avec les boucles C traditionnelles ne peuvent pas être faites avec range-for et il est plus facile à optimiser. Avec std :: span (), il élimine la plupart des erreurs de style de dépassement de tampon.

Les fonctions lambda, je les apprécie beaucoup pour les parties STL.
Bjarne Stroustrup : Oui. Ils sont devenus la clé des rappels de toutes sortes, y compris dans la spécification d'arguments pour des algorithmes paramétrés avec des prédicats et d'autres opérations. Les lambdas sont souvent plus rapides et plus pratiques que les alternatives. Notez qu'en C++, un lambda est simplement une notation pratique pour définir et instancier un objet fonction.

La surcharge de l'opérateur est assez bonne ; c'est merveilleux même. Est-ce indispensable ?
Bjarne Stroustrup : En effet. Je ne pourrais pas me passer de surcharge (pour la fonction et les opérateurs). Elles sont l'une des clés de la programmation générique et de nombreuses formes de code élégant.

Public, privé et protégé : la séparation entre les choses peut être merveilleuse, et parfois friend (un ami) vient à la rescousse !
Bjarne Stroustrup : En effet, même si je ne trouve pas beaucoup d'utilisations pour «protégé» de nos jours.

Avec l'expérience, je vois les concepts comme une pure beauté de l'abstraction, avec la fonction de spécialisation. Avons-nous besoin d'améliorations des concepts?
Bjarne Stroustrup : Nous devons polir quelques coins difficiles sur l'utilisation des contrats, mais sinon je pense que nous sommes bien pour l'instant. Par exemple, j'aimerais pouvoir contraindre les arguments de modèles dans les définitions de variables :
paire <intégrale, dérivée_de <Forme> *>
p = {27, nouveau cercle {p, 30}};
Avec des concepts, beaucoup de nos conceptions deviendront plus simples et plus faciles à utiliser. Si nous n'avons pas besoin de contraindre, nous pouvons simplifier :
paire p = {27, nouveau cercle {p, 30}}; //
p devient une paire <int, Circle *>
Nous avons la déduction d'argument modèle depuis C++ 17.

Même chose pour la notion de virtuel. Je ne pourrais plus m'en passer en programmation. Vous avez la même approche ?
Bjarne Stroustrup : Et bien souvent, nous le pouvons. Les hiérarchies de classe et les fonctions virtuelles sont excellentes, mais quelque peu surutilisées.

A New-York, vous étiez à l'affiche de 66 Minutes avec le titre « The engine of everything ». Qu'est ce que cela vous fait de savoir que C++ est si populaire depuis 40 ans et ce sans publicité, sans organe de presse ni marketing ?
Bjarne Stroustrup : Évidemment, je me sens heureux quand je vois que le C++ est si largement utilisé pour bien faire. J'ai aussi un peu peur parce que c'est une grande responsabilité. C++ est vraiment un peu partout. La plupart du temps, il est invisible dans le cadre d'un système dont nous

dépendons, par exemple notre téléphone, notre téléviseur, nos films et nos jeux, notre voiture, notre banque, notre appareil photo, la ferme qui produit notre nourriture. Bien que ce soit un honneur et non une publicité pour C++, me voir sur un écran vidéo de trois étages à Times Square était un peu effrayant.

Quand les techniciens pensent au C++, ils pensent au C et aux pointeurs. Quand ils se rendent compte que les pointeurs intelligents peuvent faire le travail, avec RAII, ils sont sans voix. Tout ce qu'ils ont appris avec Java et NET est inutile.

Bjarne Stroustrup : Eh bien, ces autres langages ont leur place, mais oui, il est triste que beaucoup aient une vision complètement déformée du C++. J'ai écrit «A Tour of C++» pour aider les gens à se faire une idée du C++ moderne. Il a l'avantage évident d'être mince. Si vous avez déjà maîtrisé la programmation, vous pouvez la lire sur un week-end.

Quel est votre sentiment avec ces langages soi-disant productifs comme Java et .NET / C #?

Bjarne Stroustrup : Je ne fais pas de comparaisons. Ils sont difficiles à bien utiliser. Je pense que beaucoup gagneraient à apprendre le C++ moderne. Malheureusement, une grande partie de l'enseignement du C++ est bloquée dans les années 1980. Le comité des normes C++ (WG21) dispose désormais d'un groupe d'étude sur l'éducation, qui vise à fournir des directives d'enseignement facilement accessibles.

Comment avez-vous prévu d'écrire les lignes directrices Core C++ avec Herb Sutter sur isocpp github ? Pouvez-vous nous donner l'aspect sous-scène de cela ? Ce pourrait être un livre d'Addison Wesley C++ Series mais c'est gratuit. Était-ce voulu ?

Bjarne Stroustrup : Je travaillais sur un ensemble de lignes directrices pour Morgan Stanley basé sur les sections de conseils dans mes livres (par exemple, «A Tour of C++») dans le but de les rendre largement disponibles. Il m'est venu à l'esprit que je ne pouvais pas être le seul à faire ça. Après tout, C++ 11 et C++ 14 amenaient beaucoup de nouvelles personnes au C++ moderne et ils avaient besoin de lignes directrices. J'ai donc demandé autour de moi et Herb avait commencé un travail similaire chez Microsoft. Évidemment, pour pouvoir collaborer librement et partager notre travail en temps opportun, nous en avons fait un projet open source. Les directives et la petite bibliothèque de support (GSL) sont sur Github:

- https://github.com/isocpp/CppCoreGuidelines
- https://github.com/microsoft/gsl

Le support d'analyse statique (essentiel) est principalement dans Visual Studio avec un peu de support dans Clang Tidy. C'est un projet ambitieux. Pour commencer, nous voulons garantir un style de C++ qui soit sûr pour les types et pour les ressources. Au-delà de cela, nous aimerions éliminer l'utilisation complexe inutile et les problèmes de performances courants. Les Core Guidelines sont une autre bonne façon de voir le C++ moderne. Mais ne lisez pas tout d'un coup. Il est destiné à être utilisé avec un analyseur statique, mais l'introduction peut être très utile et les directives individuelles peuvent être utilisées comme une FAQ. De toute évidence, les gens sont encouragés à contribuer. Ce n'est pas seulement Herb et moi. Nous avons de nombreux contributeurs et plusieurs éditeurs réguliers.

Certaines personnes en marketing ont dit il y a quelques années que le C++ était «dangereux et non sécurisé». Je leur explique que les pointeurs intelligents, auto, nullptr, vector, string font le travail et ce n'est pas logique d'émettre une telle idée. Quelle est votre opinion sur ce point ?

Bjarne Stroustrup : La sûreté et la sécurité sont des propriétés du système. Certaines personnes semblent obsédées par la possibilité d'insécurités linguistiques, mais chaque langue capable de manipuler directement le matériel en aura. Tout langage pouvant utiliser directement l'API des systèmes d'exploitation en possède. Si je voulais pénétrer dans un ordinateur, je ne commencerais pas à jouer avec C++, il y a des moyens plus faciles. Cela dit, la plupart des choses dont les gens se plaignent peuvent être évitées dans le C++ moderne ; il suffit de regarder les directives de base C++ (et de les appliquer). La fonctionnalité que vous mentionnez fait partie des fonctionnalités qui vous aident.

C++ est un diamant. Pour les personnes qui ont besoin d'allouer de la mémoire dynamique, il convient, pour les personnes qui ont besoin d'une mémoire constante, il convient, pour créer toutes sortes de logiciels. Avec le succès de l'outillage natif et spécialement du backend LLVM ou GCC, une série de langages adoptent le style natif car c'est celui qui fonctionne bien et qui fonctionne le mieux. Que ressentez-vous avec l'écosystème des langages de programmation natifs ?

Bjarne Stroustrup : Difficile à dire car il n'y a pas qu'un seul écosystème de ce type. C'est vraiment agréable de voir tous ces nouveaux langages construits sur une infrastructure C++.

Certaines entreprises investissent dans Java, Kotlin, d'autres Rust, d'autres Go, d'autres Swift. Que pensez-vous de la cette diversité?

Bjarne Stroustrup : Il est bon de voir les développements dans les langages de programmation et les techniques de programmation. Je souhaite juste que plus de gens se rendent compte qu'il n'y a pas de langage qui soit le meilleure pour tout et pour tout le monde. Le mieux est qu'ils essaient d'apprendre le mieux possible et qu'ils profitent du meilleur de ce qu'ils connaissent dans les langages pour développer au mieux. Le C++ est souvent «celui à battre». Il y a une raison à cela : C++ est suffisamment bon, efficace et stable pour beaucoup de choses. Tout langage qui survit pendant quelques décennies aura certains des problèmes de C++.

Tout sur mon PC portable est fait en C / C++ : Windows, Sysinternals Suite, Office (Word, Excel, PowerPoint, Outlook), Chrome, VLC, Winamp, Gimp, Acrobat Reader, SQL Server, mes jeux. TOUT. Y a-t-il de meilleures réalisations? C++ domine le monde, non ?

Bjarne Stroustrup : Personnellement, ce dont je suis le plus fier, c'est du rôle du C++ en science et en ingénierie… Pensez au CERN, le projet du génome humain, les Mars Rovers. De plus, C++ se propage dans de nouveaux domaines d'application. Par exemple, grâce à TensorFlow, c'est le fondement de la plupart des IA / ML.