Pour apprendre correctement la programmation

Cette discussion suit celle ci : https://openclassrooms.com/forum/sujet/foncteurs-c-pointeur-sur-fonction mais il y a des points a expliquer pour ceux qui veulent apprendre le C++ et la programmation en général (et tout en général).

Dans la discussion cite, on parle des moyens d'utiliser une fonction comme argument d'une autre fonction. Il existe beaucoup d'approches pour réaliser cela en C++. On a cite les foncteurs de la bibliothèque standard, créer ses foncteurs, utiliser un une lambda, la déduction de types, les template, des pointeurs de fonctions, std::bind, std::function, des design patterns command ou strategy, des classes de polices, etc.

Cela fait un gros catalogue de syntaxes, assez difficile a assimiler. Surtout si on prend en compte les cas particuliers (fonctions membres, etc).

Un premier point a comprendre : les devs pros ne connaissent pas toutes les syntaxes par cœur. On s'en fout. On fait comme tout le monde : on regarde la doc. On connait les syntaxes qu'on utilise le plus souvent et on regarde sur google le reste du temps. Les compétences d'un dev pro n'est pas de connaitre pleins de syntaxes !

Le gros probleme quand on a un gros catalogue de syntaxes comme ça, c'est qu'il va être très difficile de tout retenir et surtout de comprendre les différences subtiles entre chaque solutions. Et on n'est alors pas capable de choisir la syntaxe a utiliser pour résoudre un probleme. L'erreur de Yes Man est de penser que l'esprit critique est suffisant pour faire la part des choses entre les bonnes informations et les mauvaises. Mais quand on n'a pas les informations complètes, comment peut-on avoir une vision critique correct du probleme et des solutions ?

La compréhension vient de la pratique. C'est parce que vous pratiquerez des syntaxes que vous les comprendrez, que vous comprendrez leurs

intérêts et défauts.

La bonne approche pour apprendre quelque chose (le C++, la programmation, ou ce que vous voulez), c'est d'apprendre une chose, puis de la mettre en pratique. Et voir les limites d'utiliser de cette chose. Voir que dans un certain contexte, cette solution est limite. Et a ce moment la, vous pouvez apprendre une autre solution, qui répondra a ce nouveau besoin. Et vous saurez quand utiliser la première solution et quand utiliser la seconde solution. Et quand vous atteindrez les limites de solutions, vous pourrez en apprendre une autre, puis une autre.

Il est préférable de connaitre très bien ou une deux solutions très bien que d'essayer d'en connaitre vaguement 10.

On peut alors comparer les approches pédagogiques du C/C++-old-school vs le C++ “moderne”.

En C ou C++ old school, le nombre de syntaxes est limite. Si on peut passer une indirection comme paramètre de fonction, on n'a que le pointeur. si on veut faire de l'allocation dynamique, on a que le pointeur aussi. Si on veut passer une fonction comme paramètre d'une autre fonction, on n'a que le pointeur de fonction.

Beaucoup de “vieux” développeurs pensent que le C (ou le C++ old school) est plus simple a apprendre. Et c'est vrai (en partie). Suivant ce que j'ai dit avant, le fait de n'avoir qu'une seule syntaxe permet de se focaliser dessus, de maîtriser correctement cette syntaxe et de bien comprendre les intérêts et défauts de cette syntaxe.

De ce point de vue la, le C/C++-old-school a un avantage indéniable. (+)

Le C++ “moderne” ajoute de nombreuses syntaxe pour faire la même chose. Par exemple :

- pour passer un paramètre de fonction, on peut utiliser une valeur, les pointeurs nus, les références (const ou non, lvalue ou rvalue), les pointeurs intelligents, un std::reference_wrapper, un std::optional. Et on va ajouter la move semantic.

- pour faire de l'allocation dynamique, on va pouvoir utiliser les pointeurs nus, les pointeurs intelligents, et toutes les classes RAII que propose la bibliothèque standard.

- pour passer un callable a une fonction, on a vu qu'il y a aussi beaucoup de syntaxes possibles.

On voit tout de suite le probleme : si on essaie d'apprendre toutes ces syntaxes, l'apprentissage sera beaucoup plus complexe et on aura du mal a choisir quelle syntaxe utiliser selon le contexte. C'est ce que font par exemple des enseignants ou des cours qui ont été formes au C ou C++ old school et qui doivent enseigner le C++ “moderne” sans eux même avoir de recul sur les syntaxes.

Mais c'est une erreur pédagogique. Si on fait cela, la courbe d'apprentissage est très dure et cela demande beaucoup d'investissement pour apprendre. (++)

L'approche pédagogique correcte en C++ “moderne” est de continuer un apprentissage progressif. Apprendre une syntaxe et la mettre en pratique. Et apprendre les autres syntaxes quand le besoin se présente. (Et un bon cours doit amener progressivement, via les exercices, a se confronter aux limites des syntaxes connus, ce qui permet ensuite d'introduire les nouvelles syntaxes).

Plus généralement, il ne faut pas apprendre pour la syntaxe, mais pour répondre a un probleme que l'on veut résoudre.

Arrivé a ce niveau de la discussion, il y a 2 variantes pédagogiques : soit bottom-top (du bas vers le haut, c'est a dire apprendre en premier les syntaxes bas niveau du C, puis apprendre les nouvelles syntaxes plus haut niveau), soit top-bottom (l'inverse). Personnellement, je pense que la seconde approche est meilleure, mais c'est une autre discussion. (Par contre, contrairement a ce que l'on entend parfois, l'approche top-bottom ne veut pas dire qu'on n'apprend pas les anciennes syntaxes du C++, comme les pointeurs nus ou les pointeurs de fonctions. Cela veut dire qu'on ne les apprend pas dans un premier temps, mais uniquement quand on atteins le niveau pour aborder ce genre de syntaxes).

(+) Mais le C présente aussi d'autres difficultés. En particulier, comme le C propose moins de syntaxes, le code va être plus verbeux et il va falloir écrire vous même pleins de fonctionnalités qui sont proposées par le C++ “moderne” (donc plus de travail et plus de risque de faire des erreurs). A noter, cette remarque est valide aussi pour le C++ face a des langages qui proposent des bibliothèques standards plus complètes, comme le Java, Python ou le C#. Pensez a utiliser les bibliothèques externes en C++, pour éviter ce probleme. (Ne réinventez pas la roue).

(++) Le propos du C++ “moderne” n'est pas réellement d'apprendre a programmer correctement en C++. Même en suivant un cours de C++ old school, vous pouvez apprendre a programmer correctement en C++. La meilleure preuve est que beaucoup de “vieux” experts du C++ qui défendent le C++ “moderne” ont appris via le C et le C++ old school. (C'est mon cas).

Le propos du C++ “moderne” est surtout d'apprendre efficacement ! C'est a dire ne pas refaire les erreurs d'apprentissage, ne pas devoir désapprendre ces erreurs pour pouvoir progresser, gagner plusieurs mois ou années d'apprentissage du C++. Apres plusieurs années a aider les débutants sur les forums, on voit les erreurs qui reviennent encore et encore et on voit ceux qui arrivent a devenir bon (voire très bon) en C++.

Voila mon point de vue sur le probleme de progression pédagogique en C++. Il me semble intéressant que ceux qui débutent comprennent pourquoi on insiste sur certains points, certaines façons d'apprendre. (Et qu'on dit beaucoup de mal de certains cours et auteurs)

Idées pour aller plus loin

- http://www.stroustrup.com/PPP2_Ch0.pdf

Concept interessant : notional machine

- http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002016000200003
- https://computinged.wordpress.com/2012/05/24/defining-what-does-it-mean-to-understand-computing/
- https://www.researchgate.net/publication/259998496_Notional_Machines_and_Introductory_Programming_Education
- https://www.researchgate.net/publication/266657026_The_state_of_play_A_notional_machine_for_learning_programming
- http://trace.dcs.gla.ac.uk/tracing/
- https://kar.kent.ac.uk/37645/1/2013-09-WiPSCE-Notional-Machine.pdf
- https://books.google.co.uk/books?id=ngQH8S7Zx0gC&pg=PA200&lpg=PA200&dq=notional+machine&source=bl&ots=PsjvICc4U_&sig=rKUropUnYOwF2FviJOqND4dDDIY&hl=fr&sa=X&ved=2ahUKEwiDk_Prk_TcAhWwzoUKHapcDU44ChDoATAJegQIABAB#v=onepage&q=notional%20machine&f=false