C

Une approche pédagogique

Guy Migneron
Cégep de Saint-Jérôme

Pourquoi avoir choisi le C

Tout d'abord, et en vrac, voici une série de raisons pour lesquelles on a, il y a quelques années, abandonné le Pascal en faveur du **C**.

- La première raison pour laquelle on a choisi d'enseigner le **C** est sa présence de plus en plus importante sur le marché du travail alors que le Pascal ne fait toujours pas de progrès en ce domaine. Bien sûr, le Pascal est « pédagogique » mais n'y a-t-il pas moyen d'utiliser le **C** d'une manière qui soit tout aussi pédagogique ? Si c'est le cas, les élèves acquièrent non seulement l'habileté de bien programmer mais ils prennent aussi de l'expérience dans un langage qui peut les aider à se trouver un emploi ; il est toujours plus attrayant pour un employeur de se trouver devant quelqu'un qui connaît le langage dont il se servira tous les jours que d'être devant un autre qui ne le connaît pas mais qui dit qu'il a tout ce qu'il faut pour l'apprendre rapidement.

- Le fait qu'on ait cessé d'enseigner l'assembleur dans le programme informatique ouvre aussi une place pour le **C** qui, étant plus près de la machine, nous donne la possibilité de mieux couvrir la matière associée au fonctionnement interne de l'ordinateur.

- Il y a, d'année en année, de plus en plus de volumes disponibles sur tous les aspects du **C** et parmi ceux-ci, un très grand nombre contient du matériel stimulant pour le lecteur en ce sens qu'on y exploite des possibilités particulièrement intéressantes. Les élèves qui veulent aller plus loin ont donc tout ce qu'il leur faut pour avancer.

- L'apprentissage du **C** donne la possibilité de passer au **C++** et, tout comme c'est le cas pour le Pascal d'ailleurs, de travailler en programmation orientée objet.

- Le **C** comporte toutes les structures de programmation requises et il comporte une librairie très étendue de fonctions de toutes sortes.

- Il existe plusieurs compilateurs sur le marché, fonctionnant sous tous les systèmes d'exploitation. Sur micro, on retrouve particulièrement les produits Microsoft et Borland qui ont toutes les qualités requises pour être utilisés dans un contexte d'enseignement.

- Pour le moment (car cela va certainement changer sous peu), l'enseignement du **C** place tous les élèves sur un pied d'égalité car c'est le cas chez nous du moins, personne n'a fait de **C** au secondaire.

Les principes de base que j'applique dans mon enseignement

Avant d'expliquer mon approche de l'enseignement du langage **C**, je dois énoncer deux principes que je respecte dans ma présentation de la matière. Ces deux principes ont des répercussions sur l'ordre des sujets approchés et sur la manière de les enseigner.

- l'exemple avant la règle
- un agresseur à la fois

Je considère que ces deux principes, appliqués particulièrement au cours 101, aident les élèves à s'adapter au nouveau milieu que constitue pour eux le cégep et leur permettent de commencer un apprentissage qui ne soit pas trop douloureux parce qu'approprié à leurs capacités.

En 201, tout en prenant soin de ne pas leur faire affronter de trop nombreux agresseurs à la fois, j'augmente le taux d'agression mentale et je passe progressivement à «la règle avant l'exemple» pour leur apprendre à comprendre des concepts sans que ceux-ci ne soient nécessairement placés dans un contexte précis.

Je m'attacherai plus ici à des détails techniques qui portent sur l'enseignement de la programmation en utilisant le **C** comme langage.

La méthode pédagogique

Toutes les notions sont introduites **par besoin**, pour résoudre un problème précis. C'est cette approche qui nous fera présenter de nouveaux énoncés ou de nouvelles fonctions un ou une à la fois, en respectant le principe énoncé plus tôt d'un agresseur à la fois.

Le principe de l'exemple avant la règle nous fait passer par un programme déjà construit pour lequel on explique les énoncés un à un plutôt que de donner d'abord la syntaxe des énoncés et de les intégrer progressivement à des programmes.

On doit toujours tenir compte que le **C** a comme grand défaut — ce n'est un défaut que dans l'enseignement — de laisser le programmeur faire à peu près tout ce qu'il veut avec la présomption que le programmeur doit savoir ce qu'il fait. Mais ce n'est pas parce qu'on s'est déjà coupé avec un couteau qu'on va cesser d'en utiliser ! L'important est d'apprendre à être prudent et de respecter des règles élémentaires. Si donc on fait attention de bien encadrer l'apprentissage avec des exemples bien définis et des travaux qui collent aux exemples qu'on a enseignés, ce défaut du **C** devient une qualité qui force les élèves à plus d'attention et de rigueur.

Cela dit, voici maintenant quelques détails qui aident à travailler plus facilement avec le **C.**

- Tout comme c'est la cas en Pascal, les énoncés **if**, **while** et autres portent sur l'énoncé ou le bloc qui les suit. On reconnaît deux dispositions habituelles pour les accolades englobantes : la manière des auteurs du **C**, Kernighan et Ritchie est comme suit :

```
if (condition)  {
        énoncés
   }
```

L'autre méthode reconnue et très utilisée aligne les accolades

```
if (condition)
  {
        énoncés
  }
```

C'est cette dernière que j'ai adoptée parce que je la trouve beaucoup plus facile à évaluer du premier coup d'œil, favorisant à la fois les élèves et la personne qui les aide à mettre leurs programmes au point.

- Les opérateurs logiques **ET** et **OU** en **C**, respectivement exprimés par **&&** et **||**, ne sont peut-être pas évidents comme un **AND** et un **OR** mais on peut dire qu'ils ont le mérite de respecter la loi 101.

- Un des premiers problèmes auxquels on est confronté dans l'utilisation du **C** comme langage d'apprentissage de la programmation est sa très grande variété de fonctions d'entrée-sortie. J'ai opté pour **scanf()** et **printf()** respectivement pour la lecture et l'affichage parce que ces fonctions sont générales, qu'elles se ressemblent au point du vue syntaxe et parce qu'elles donnent l'occasion de faire l'apprentissage de la rigueur dans leur utilisation. On a l'occasion de faire associer une liste de formats avec une liste de variables et/ou de constantes. On peut faire de nombreux exercices avec ces instructions et les élèves peuvent travailler par eux-mêmes à en découvrir plusieurs des possibilités.

  Ces fonctions ont aussi des limites, notamment lorsqu'on utilise **scanf()** pour lire un entier et qu'on tape un caractère non numérique. J'ai tout simplement décidé de passer outre à cette faiblesse et de faire en sorte que les élèves évitent cette situation.

  Également à cause d'une particularité du **scanf()**, on a assez tôt besoin d'utiliser les fonctions **getch()** et **getche()**. Celles-ci sont dont expliquées pour le cas précis où on a besoin de lire un seul caractère et, quand on expliquera les fonctions qui renvoient une valeur, on pourra s'appuyer sur l'expérience acquise avec ces fonctions pour expliquer comment on récupère la valeur de retour.

  scanf() reconnaît l'espace entre deux mots comme un séparateur. Si les élèves veulent lire un nom et un prénom dans une seule variable par exemple, ils doivent lier le nom et le prénom par un tiret ou une barre de soulignement dans leur fichier de données. C'est encore là une limite acceptable avec laquelle on peut vivre un bon bout de temps — toute la première session et le début de la deuxième.

  Dans un **scanf()**, on doit utiliser l'opérateur **&** pour lire les entiers et réels et ne pas mettre cet opérateur pour les chaînes de caractères. Cette particularité qui peut sembler étrange en début d'année est tout simplement donnée comme une des réalités de la vie, une des contraintes à laquelle un élève doit se plier même quand il est trop jeune pour comprendre. Pour agrémenter le tout, il ne faut pas utiliser de **&** dans un **printf()** et cela ne fait qu'une autre règle à mémoriser. Les formateurs se plaignent souvent que les jeunes n'apprennent plus rien par cœur : on a là une façon de travailler à remédier à cette situation.

- **scanf()** et **printf()** ont des équivalents qui lisent et écrivent sur fichier. Il est facile, une fois leur syntaxe et leur fonctionnement maîtrisés, de faire utiliser **fscanf()** et **fprintf()** en n'ayant à y

consacrer qu'un minimum de temps. Ceci amène directement aux fonctions **fopen()**, **fclose()**, et **feof()** qui s'imposent d'elles-mêmes.

Dans l'environnement **C++**, on peut choisir d'opter plutôt pour les énoncés **cin** et **cout**. Ceux-ci simplifient les entrées-sorties de façon significative et demandent un moindre effort de compréhension de la part des élèves. Par contre, leur utilisation demandant moins de rigueur, je pense qu'elle conduit à un apprentissage moindre,

- L'utilisation des fichiers demande de respecter plusieurs règles : il ne faut pas oublier l'énoncé **#include <stdio.h>**, il faut taper le mot **FILE** en majuscules, il ne faut pas oublier de mettre une étoile devant le nom de la variable de fichier et, comme pour toutes les autres fonctions et les autres langages, il faut respecter scrupuleusement le type et l'ordre des paramètres. Encore là, c'est une occasion pour les élèves de suivre des règles sans nécessairement tout comprendre. Ils doivent accepter que c'est ainsi et qu'ils ne sauront que plus tard le pourquoi de tous ces détails.

- Toujours pour respecter le principe d'un agresseur à la fois, je n'introduis pas les fonctions écrites par le programmeur avant que les élèves n'aient eu l'occasion de travailler sur un programme de plus de cent lignes qui soit tout d'un bloc — ce qui se produit vers la fin de la première session. À ce moment, quand on leur présente tous les avantages qu'il y a à travailler en fonctions et qu'ils peuvent en constater les qualités, ils adoptent cette manière de travailler sans aucun problème puisque cela solutionne un problème qu'ils commencent à vraiment ressentir.

- Par contre, je me contente de n'utiliser à ce moment que des variables globales pour éviter que l'utilisation des fonctions ne leur complique trop la vie et pour tenir compte des difficultés inhérentes à l'utilisation des paramètres (un agresseur à la fois).

- Les tableaux en **C** commencent à l'indice zéro. Je ne crois pas qu'il soit utile ou correct de faire comme si le premier élément n'existait pas et de déclarer les tableaux avec n+1 éléments pour pouvoir travailler avec les indices de 1 à n. Il faut leur présenter les indices comme le déplacement par rapport au début du tableau et leur donner l'habitude de toujours travailler avec des boucles qui tiennent compte des bornes réelles du tableau. Dans tous les traitements de tableaux, on aura une variable ***nb_elements*** qu'on initialisera à zéro et qui servira à contrôler l'ajout d'une valeur toujours par des énoncés du type

  tableau[*nb_elements*] = *nouvelle valeur*;
  nb_elements++;

  On traitera donc toujours un tableau complet avec les mêmes boucles

  for(i=0;i<nb_elements;i++)...

  Dans une recherche dans un tableau, on ne renverra pas zéro pour indiquer qu'on n'a pas trouvé mais simplement la valeur ***nb_elements*** qui désigne le prochain élément libre. On fera de même pour toutes les techniques de base associées aux tableaux, prenant soin de suivre un méthode qui soit toujours cohérente.

- Les chaînes de caractères demandent qu'on comprenne comment **C** les traite. La notion de marque de fin de chaîne est essentielle à la bonne utilisation des fonctions. J'utilise les chaînes

de caractères en entrée et en sortie dès la première session parce qu'il n'y a pas de différences avec les autres variables mais ce n'est qu'en deuxième session que je leur fais utiliser des comparaisons de chaînes. Au même moment, je présente les fonctions les plus utiles pour les chaînes, omettant celles dont le résultat est un pointeur.

- À mesure qu'ils écrivent des programmes, les élèves doivent utiliser de moins en moins de variables globales et de plus en plus de variables locales et de paramètres. Il ne faut pas se raconter d'histoires, le passage des paramètres par adresse est plus compliqué en **C** qu'en Pascal quand il s'agit de variables simples. Je les fais donc travailler exclusivement sur des paramètres transmis par valeur pour les variables simples pendant la plus grande partie de la deuxième session. Les paramètres par adresse sont ensuite utilisés pour des tableaux et des chaînes puisque ces variables sont automatiquement transmises ainsi.

  Pour modifier le contenu d'une variable simple, on utilise la variable de retour d'une fonction plutôt qu'un paramètre passé par adresse. On a donc des énoncés du type

  valeur = fonction1(valeur);

- Dans mes exemples, le prototype des fonctions qui reçoivent une chaîne de caractères est indiqué selon la syntaxe ***fonction1*(char[])** plutôt que ***fonction1*(char *)** parce que cette première façon colle plus à la déclaration de la variable qu'on transmet.

- En fin de deuxième session, et avec plusieurs explications et exemples, j'introduis le passage des paramètres par adresse pour les variables simples. Les élèves ont donc eu l'occasion d'utiliser l'énoncé **return**, et ils ont pu expérimenter la notion de passage par adresse avec les tableaux et les chaînes. Ils passent peut-être moins de temps que les utilisateurs de Pascal à approfondir cette dernière notion mais ils ont par contre une première approche des pointeurs qui leur servira pour le cours de structures de données.

- En **C**, l'énoncé **do...while** a comme inconvénient d'utiliser le même mot clé **while** alors que Pascal a le **repeat...until** qui porte moins à confusion. Encore là, la rigueur et l'attention conduisent les élèves à adopter des habitudes qui leur seront utiles pour toute leur programmation.

- Lors des cours sur les structures d'enregistrements, j'insiste pour qu'on passe les structures en paramètre par adresse plutôt que par valeur même si cela est plus compliqué et implique un nouvel opérateur. Il est important que les élèves comprennent de plus en plus les mécanismes impliqués par le passage des paramètres et qu'ils soient conscients du gain d'efficacité donné par le passage par adresse.

- Les messages d'erreur donnés par le compilateur sont parfois sibyllins à qui ne comprend pas la notion de pointeur. Par exemple, en Turbo **C**, l'oubli du nom de fichier dans un énoncé **fprintf()** donne le message « Suspicious pointer in ... » et l'utilisation d'un caractère simple plutôt que d'une chaîne dans un énoncé comme **strcmp()** donne le message « Non-portable pointer conversion... ». Pour rendre les élèves autonomes sur ce point sans devoir leur donner trop d'explications qu'ils ne pourraient comprendre, je leur fais constituer un fichier dans lequel ils doivent consigner ces messages avec ce qu'il faut faire pour corriger l'erreur correspondante. Ils commencent à mettre un contenu dans ce fichier dès le début de la première session et ils l'enrichissent à mesure qu'ils font des erreurs. Comme le fichier est tapé avec l'éditeur du **C**, ils peuvent le consulter sans devoir quitter l'environnement intégré Turbo **C**.

On peut mal programmer dans n'importe quel langage. Le **C** se prête particulièrement bien aux programmes hermétiques où se côtoient doubles astérisques et perluètes multiples, passage en paramètres de tableaux d'adresses de fonctions, etc. Mais on peut aussi programmer en **C** de façon tout aussi lisible que dans tous les autres langages et c'est là le premier défi à affronter pour qui n'a vu du **C** que des programmes complexes tels que donnés dans des manuels avancés.

Une fois qu'on choisit de ne programmer que de façon claire quitte à ne pas profiter de toutes les « passes » disponibles, le **C** est un langage qui se prête tout autant aux applications pédagogiques que n'importe quel autre...il suffit de cesser de penser Pascal et de commencer à penser **C**.

En somme, pour utiliser le **C** plutôt que le Pascal, il ne s'agit pas de changer les objectifs des cours 420-x01 mais simplement d'adapter aux exigences du langage l'ordre de présentation de certains sujets. On y gagne au niveau du degré de compréhension que les élèves ont besoin d'avoir du fonctionnement de l'ordinateur. On y gagne aussi dans la rigueur que les élèves sont obligés de garder et on leur donne un outil qui peut leur servir de façon directe lors de leur arrivée sur le marché du travail.