Chapitre 2

Introduction à la programmation

Nous avons vu, dans le chapitre précédent, ce qu'est un algorithme et un certain nombre de façons pour exécuter les algorithmes « à la main ». Nous allons voir, dans ce chapitre, comment utiliser la *programmation* pour exécuter les algorithmes, comment on met en place celle-ci sur un ordinateur moderne (*interprétation* et *compilation*). Nous donnerons un premier exemple de *programme source*, en langage C. Nous décrirons quelques compilateurs C et comment mettre en place notre programme sur ceux-ci.

2.1 Notions théoriques

2.1.1 Qu'est-ce que la programmation ?

Un *ordinateur*, et qu'importe pour l'instant une définition précise, peut être utilisé dans des buts divers : pour faire tourner des jeux (c'est souvent de nos jours la première prise de contact, pour les jeunes tout au moins), pour utiliser des logiciels de *bureautique* (traitement de texte, tableur...), pour avoir accès à Internet... Cependant il s'agit d'activités spécialisées. L'activité la plus versatile, celle pour laquelle les ordinateurs ont été conçus à l'origine, est d'effectuer des calculs complexes à décrire. La façon de décrire ces calculs (dans un sens très large) est l'objet de la *programmation*.

> La **programmation** est la façon d'indiquer à l'ordinateur les calculs qu'il doit effectuer.

La programmation permet, entre autres, la conception des *logiciels*. D'une façon plus modeste, elle permet de créer des programmes de calculs pour lesquels il n'existe pas de logiciels.

2.1.2 Façons de programmer

Sur les premiers ordinateurs, la programmation consistait à recâbler à chaque fois entre elles un certain nombre d'unités. Heureusement pour celui qui programme, le **programmeur** en l'occurrence, la façon (physique) de programmer a évolué.

Suivant une idée de JOHN VON NEUMANN de la fin des années 1940, le **programme** est maintenant une donnée comme les autres, entrée dans la mémoire vive de l'ordinateur comme les autres données.

Ce programme peut être écrit en **langage machine**, qui est directement compréhensible par le processeur. L'inconvénient majeur du langage machine est que le programmeur doit faire un très gros effort de codage. On a utilisé ensuite des **langages symboliques**, plus compréhensibles par le programmeur, qu'il faut traduire en langage machine à un certain moment. Ce fut l'idée d'Alan TURING au début des années 1950.

On a pendant longtemps distinguer deux types de langages symboliques (de nos jours, la hiérarchie est devenue beaucoup plus fine) : les **langages d'assemblage**, très proches des langages machine mais utilisant des mnémoniques très utiles pour le programmeur, et les **langages évolués** plus proches de la façon de raisonner des programmeurs, le traducteur de langage faisant le travail pour se rapprocher du langage machine.

Les langages évolués sont **interprétés** ou **compilés**. L'**interprétation** consiste à traduire ligne à ligne le programme au moment où on le lance ; les erreurs sont repérées rapidement, on a une forme d'interactivité intéressante. La **compilation** consiste, dans une première étape, à traduire le programme puis, dans une seconde étape, à l'exécuter. On perd l'interactivité mais l'exécution est plus rapide.

2.1.3 Les grandes étapes de la compilation

De nos jours, on effectue la programmation de la façon suivante. On commence par écrire un **programme**, dit plus exactement **programme source**, grâce à un éditeur de texte, dans un langage particulier, dit **langage de programmation** (aux règles syntaxiques très strictes, contrairement aux **langues naturelles** [ainsi appelés par les informaticiens par opposition aux langages de programmation] tels que le français ou l'anglais). Cependant ce langage est quand même suffisamment proche de l'anglais de base (impérialisme anglo-saxon oblige !).

On utilise ensuite un logiciel particulier, appelé un **compilateur**, pour traduire ce programme

source en quelque chose de plus compréhensible par l'ordinateur, qui est appelé **programme objet**. Le programmeur peut considérer le programme objet comme quelque chose de magique, ce n'est pas son problème. On en apprend plus sur les programmes objets dans le cours de **Système d'exploitation**, dans lequel on voit la conception actuelle (c'est-à-dire celle d'aujourd'hui qui n'est ni celle d'hier, ni certainement celle de demain) des ordinateurs et la structure d'un programme objet.

Ce programme objet se présente comme un fichier d'une nouvelle sorte (par rapport à nos fichiers texte, maintenant bien connus), dit **fichier binaire** (c'est en fait comme cela que l'on appelle tout fichier qui n'est pas un fichier texte, c'est-à-dire qui ne donne rien de bien compréhensible lorsqu'il est lu avec un éditeur de texte) mais qui est (plus ou moins) **exécutable** : son appel sur la ligne de commande (dans le cas d'un interpréteur de commandes textuel ; par double clic sur son icône dans le cas d'un interpréteur de commande graphique) déclenche l'exécution de ce qui est décrit dans le programme (source).

2.1.4 Diversité des langages de programmation

De même qu'il existe de nombreuses langues naturelles, il existe de nombreux langages de programmation. Les langages se différencient les uns des autres (outre leurs règles syntaxiques, bien sûr) par le fait qu'ils sont plus appropriés pour tel ou tel but.

Le langage PASCAL, par exemple, a été conçu en 1971 comme un langage d'initiation à la programmation et permettant d'exprimer simplement les divers *algorithmes*. Ce cours de programmation sera cependant illustré par un autre langage, le **langage C** pour deux raisons : le langage C est le plus utilisé dans les entreprises, mais ceci est une mauvaise raison ; la raison principale est que si le cours de *programmation structurée* serait bien illustré par PASCAL, il n'en sera pas de même du cours de *programmation orientée objet*, pour lequel le langage C++ est devenue la référence. Le langage C++ utilise le langage C comme noyau, nous avons intérêt à connaître le langage C.

2.1.5 Diversité des compilateurs

Le compilateur doit traduire un programme source, écrit dans un langage de programmation donné, en un langage objet. Il y a donc au moins un compilateur par langage de programmation. Le programme objet dépend du système informatique sur lequel on se trouve, essentiellement du microprocesseur mais aussi du système d'exploitation. Il n'est donc pas suffisant de chercher un compilateur C, il faut un compilateur C adapté à tel ou tel système informatique. De plus, puisque les compilateurs sont souvent des **progiciels** (logiciels conçus par des entreprises commerciales et mis à jour régulièrement), il existe souvent plusieurs compilateurs pour un langage donné sur un système donné qui se font concurrence (tout au moins dans la mesure où le marché est porteur).

2.2 Un premier exemple de programme C

Nous allons écrire un programme permettant d'afficher 'Bonjour' sur l'écran du moniteur.

Première étape : écrire le programme source.- Le programme source s'écrit grâce à un éditeur de texte, n'importe lequel. Écrivons donc le programme suivant (sur votre éditeur de texte préféré) :

```
#include <stdio.h>

void main(void)
  {
  printf("Bonjour");
  }
```

Deuxième étape : sauvegarder le programme.- Une bonne habitude consiste à sauvegarder les programmes (source) avant de passer à la compilation. Enregistrons donc le programme ci-dessus, par exemple sous le nom **essai.c**.

L'extension 'c' est traditionnelle pour les programmes source du langage C.

Troisième étape : la compilation.- La façon de faire dépend du compilateur utilisé. Nous y reviendrons plus loin lors de la description de quelques compilateurs. Si tout se passe bien, un nouveau fichier se trouve dans le répertoire dont le nom dépend du compilateur.

Quatrième étape : exécution du programme.- Ce nouveau fichier est en général un exécutable et, en le lançant, on devrait voir apparaître 'Bonjour' à l'écran.

2.3 Quelques compilateurs

Le compilateur de référence a longtemps est le compilateur GNU `gcc`.

2.3.1 Le compilateur GNU C sous Unix

Tout système d'exploitation Unix est livré avec un compilateur C car c'est le langage de programmation système favori de Unix, d'ailleurs créé pour lui. Plus précisément la première version d'Unix fut écrite en 1969 en langage d'assemblage du mini-ordinateur PDP 7 ; le langage C fut conçu par Dennis RITCHIE à peu près en même temps et en 1973, Dennis RITCHIE et Ken THOMPSON ont réécrit le noyau Unix en langage C, ce qui en fit le premier système d'exploitation à ne pas être écrit en langage d'assemblage.

2.3.1.1 Installation

Nous n'expliquerons pas ici comment installer le compilateur C sous Unix puisque, pour la raison que nous venons d'indiquer, il s'installe en même temps que le système d'exploitation (à de rares exceptions près, à savoir lorsqu'on ne veut vraiment ni programmer, ni compiler des logiciels distribués sous la forme de fichier source, ce qui n'est pas notre cas).

L'installation d'Unix peut être une opération délicate. Nous supposerons ici que Linux est installé sur un compatible PC.

2.3.1.2 Premier exemple

Voyons comment mettre en place notre premier exemple permettant d'afficher 'Bonjour'.

Première étape : écrire et sauvegarder le programme.- Le programme s'écrit avec votre éditeur de texte favori sous Unix. Par exemple avec *emacs*, on lance l'éditeur, à partir du répertoire sur lequel on veut sauvegarder le programme, en écrivant (en supposant que le prompteur soit $) :

```
$ emacs bonjour.c &
```

Écrivons alors le texte vu ci-dessus dans la fenêtre qui apparaît, sauvegardons (grâce au menu déroulant) et quittons (grâce au menu déroulant ou en faisant `CTRL-X CTRL-C`, la façon de quitter *emacs*).

Deuxième étape : la compilation.- Pour compiler ce programme, toujours à partir du même répertoire, on écrit :

```
$ cc bonjour.c
```

ou :

```
$ gcc bonjour.c
```

comme on veut.

La commande `cc` correspond à *Compilateur C* (en fait à *C Compiler*), la commande `gcc` à *Gnu cc*.

Il existe alors un nouveau fichier dans le répertoire, appelé **a.out** (pour *Assembler Output*).

Troisième étape : exécution du programme.- Pour exécuter le programme il suffit de faire appel à lui, en écrivant sur la ligne de commande :

```
$ a.out
```

puis en appuyant sur la touche « retour ». Normalement on doit voir apparaître ce que le programme est censé faire, c'est-à-dire qu'apparaît :

```
Bonjour$
```

Remarquons qu'il n'y a pas de passage à la ligne après affichage du résultat.

Quatrième étape : donner un nom au programme.- Cela peut être gênant d'avoir toujours le même nom pour l'exécutable, surtout si on veut en utiliser deux. Si on veut que l'exécutable s'appelle `Bonjour`, par exemple, on compile de la façon suivante :

```
$ cc -o Bonjour bonjour.c
```

en utilisant le *paramètre* `'-o'` (pour *'output'*).

2.3.2 Le compilateur de MinGW sous Windows

MinGW, contraction de l'anglais « *Minimalist Gnu for Windows* », est un environnement de développement minimaliste d'applications (ou logiciels) pour Microsoft Windows.

2.3.2.1 Installation

Récupération sur Internet.- Le compilateur MinGW se trouve sur le site :

`http://www.mingw.org`

Cliquer dans le menu de navigation à gauche de la page d'accueil sur « downloads » pour se retrouver sur la page :

`https://sourceforge.net/projects/mingw/files/`

puis cliquer sur « Download mingw-get-setup.exe (86.5 kB) » qu'il faut enregistrer dans le répertoire que vous voulez.

Exécution.- Faites exécuter le fichier récupéré.

Notez bien l'emplacement dans lequel le compilateur est placé, emplacement proposé ou celui que vous lui indiquez, par exemple dans :

« c :\programmes\mingw\ »

Modification de la variable d'environnement PATH.- L'exécution précédente fait presque tout automatiquement, mais n'indique pas au système d'exploitation l'emplacement du compilateur `gcc`, alors que nous avons besoin.

Vérifions-le ! Ouvons l'invite de commandes : sous Windows 8, par exemple, cliquer sur « Rechercher » sur le bandeau de droite (qui apparaît en effectuant un léger mouvement de droite à gauche sur le bord droit de l'écran tactile) ; écrire « commande » dans la fenêtre de recherche (« application » est par défaut, sinon il faut cliquer dessus) ; « invite de commandes » est proposé ; cliquer dessus ; une nouvelle fenêtre (voir figure ci-dessous) apparaît.

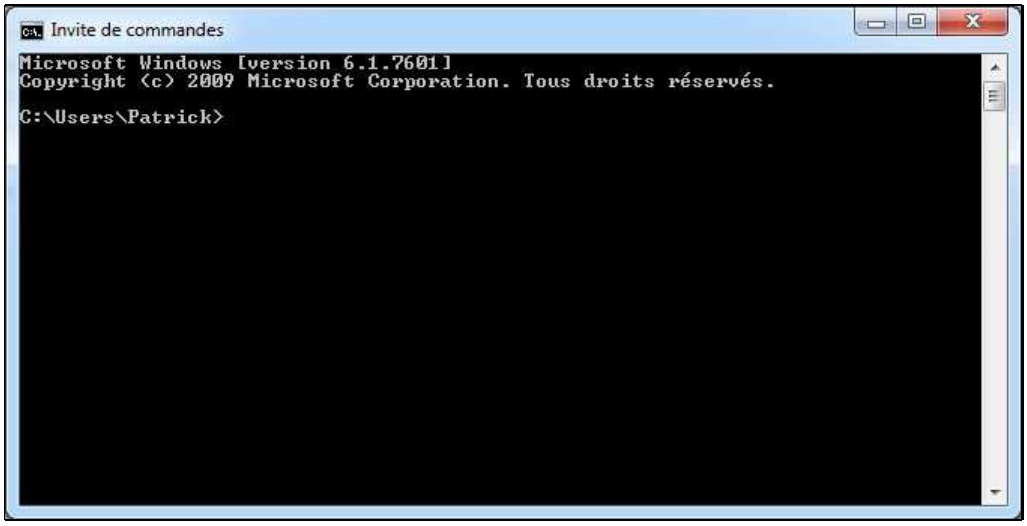

FIGURE 2.1 – L'invite de commande de Windows

Écrire 'gcc' puis appuyer sur la touche « entrée ». La réponse est :

« `'gcc' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commandes.` »

Pour remédier à cela, faisons apparaître l'explorateur de fichiers. Un clic avec le bouton de droite de la souris sur « Ordinateur » fait apparaître un menu déroulant. Double-cliquer sur « Propriétés ». Sur la fenêtre qui surgit, cliquer sur « Paramètres système avancés » (Attention ! il faut détenir les droits de superutilisateur). Sur la nouvelle fenêtre, cliquer sur « Variables d'environnement ». Sur la fenêtre suivante, cliquer sur « Path » de « Variables système » puis sur « Modifier ». En faisant très attention (ce n'est pas très lisible, d'une part, et, d'autre part, si on détruit une partie de ce qui est déjà écrit, d'autres applications ne fonctionneront plus), ajoutons à la fin un point-virgule et le chemin conduisant au sous-répertoire 'bin' de MinGW, soit dans l'exemple vu ci-dessus :

« ;c :\programmes\mingw\bin »

Cliquer sur 'OK', encore sur 'OK' pour la nouvelle fenêtre, puis une troisième fois.

Si tout est correct, sur une nouvelle instance de l'invite de commandes (Attention ! l'ancienne ne prend pas en compte la nouvelle version des variables d'environnement), 'gcc' devrait donner lieu à la réponse suivante '`gcc: fatal error: not input files. Compilation terminated`'. Si ce n'est pas le cas, il faut recommencer en cherchant là où on s'est fourvoyé.

2.3.2.2 Premier exemple

Reprendre ce qui est dit à propos du compilateur `gcc` sous Linux, à part que l'on utilisera éventuellement un autre éditeur de texte (`bloc-notes` de Windows, par exemple). Sous Windows, on se placera dans une fenêtre de ligne de commandes, évidemment.

2.3.3 Le compilateur sous MacOS

2.3.3.1 Le terminal

L'analogue de l'invite de commande de Windows s'appelle « Terminal » sous le système d'exploitation MacOS d'Apple. Beaucoup d'utilisateurs ne l'utilisent jamais.

La première fois qu'on veut l'utiliser, il faut aller chercher cette application dans le dossier « /Applications/Utilitaires ». Mettez son icône dans le « Dock » pour l'avoir toujours à portée de souris dans la suite. Lancez-le en cliquant sur l'icône.

2.3.3.2 Installation du compilateur gcc

Tapez « gcc » ou « cc » dans le terminal puis sur la touche retour. Deux situations peuvent se produire :

Premier cas : le compilateur est déjà installé.- Dans ce cas on voit une phrase analogue au cas de Windows.

Deuxième cas : le compilateur n'est pas encore installé.- Dans ce cas un message s'affiche, indiquant qu'il faut installer l'application (gratuite) « Xcode ».

Il faut donc aller télécharger « Xcode » sur « Mac App Store »

2.3.3.3 Utilisation de l'éditeur de texte `TextEdit`

L'analogue d'à la fois l'éditeur de texte « Bloc-Note » et du traitement de texte « Wordpad » de Windows s'appelle, sous MacOS, « TextEdit », qui vient avec MacOS X.

Par défaut, il va plutôt se comporter comme un traitement de texte, à la « WordPad ».

Pour l'utiliser comme éditeur de texte, choisir dans le menu « `Format` > `Make Plain Text` » (ou utiliser le raccourci clavier 'Command+Shift+T').

On peut aussi décider de ce mode éditeur de texte comme mode par défaut à chaque fois qu'on lance cette application : dans « `Preferences` », sous `Format`, choisir `Plain Text`.