

Introduction à la programmation structurée, illustrée par le langage C

Patrick Cegielski
cegielski@univ-paris12.fr

Septembre 2006

Pour Irène et Marie

Legal Notice

Copyright © 2006 Patrick Cégielski

Université Paris XII - IUT

Route forestière Hurtaut

F-77300 Fontainebleau

cegielski@univ-paris12.fr

Préface

Faisons le point sur les quelques prérequis nécessaires. Nous supposons ici que vous avez eu un premier contact avec un *ordinateur*, plus exactement avec un micro-ordinateur (et même un micro-ordinateur dit compatible PC), ou plus précisément avec un système informatique. Nous supposons en particulier que vous savez utiliser l'*interpréteur de commande* et un *éditeur de texte*, qu'importe lequel.

Contents

Préface	v
1 Introduction à la programmation	1
1.1 Notions théoriques	2
1.1.1 Qu'est-ce que la programmation ?	2
1.1.2 Façons de programmer	2
1.1.3 Les grandes étapes de la compilation	2
1.1.4 Diversité des langages de programmation	3
1.1.5 Diversité des compilateurs	3
1.2 Un premier exemple de programme C	4
1.3 Quelques compilateurs pour PC	5
1.3.1 Le compilateur GNU C sous Unix	5
1.3.2 Le compilateur Turbo C et ses dérivés	6
1.3.3 Le compilateur DJGPP	7
2 Calculette	11
2.1 Premiers éléments de langage C	12
2.1.1 Forme d'un programme	12
2.1.2 Fichiers inclus	13
2.1.3 Compilation et édition des liens	14
2.1.4 Niveaux d'avertissement du compilateur	14
2.2 Le langage C comme calculette quatre opérations	15
2.2.1 Cas des entiers naturels	15
2.2.2 Cas des entiers relatifs	17
2.2.3 Cas des rationnels	18
2.2.4 Cas des réels	18
2.3 Le langage C comme petite calculette scientifique	21
2.3.1 Utilisation de la bibliothèque mathématique	21
2.3.2 Nouvelles opérations sur les entiers	21
2.3.3 Formatage de l'affichage des réels	22
2.3.4 Fonctions prédéfinies sur les réels	23
2.3.5 Partie entière de \mathbb{R} dans \mathbb{N}	24
2.4 Le langage C comme calculette améliorée	24
2.4.1 Évaluation d'expressions numériques	24
2.4.2 Manipulation de textes	25
2.5 Sémantique réelle	26
2.5.1 Cas des entiers naturels	26
2.5.2 Cas des entiers relatifs	27

2.5.3	Cas des réels	27
3	Variable, affectation et lecture	29
3.1	Un premier programme	30
3.2	L'affectation	31
3.2.1	Variable	31
3.2.2	Identificateur	31
3.2.3	Type	32
3.2.4	Déclaration des variables	34
3.2.5	Affectation	35
3.2.6	Initialisation des variables	36
3.2.7	Partie entière et conversion de type	36
3.3	Ordres de lecture et d'écriture	37
3.3.1	Un exemple	37
3.3.2	Ordre d'écriture formatée	38
3.3.3	Ordre de saisie formatée	38
3.3.4	Cas des chaînes de caractères	39
3.3.5	Saisie d'un caractère	40
3.4	Les constantes	41
3.4.1	Déclaration des constantes	41
3.4.2	Exemples d'utilisation	41
4	Structures de contrôle	43
4.1	Le séquençement	44
4.2	Conditions	44
4.2.1	Expressions booléennes	44
4.2.2	Relations définies en langage C	46
4.3	La répétition	47
4.3.1	Boucle tant que faire	48
4.3.2	Boucle répéter tant que	49
4.3.3	Boucle pour	49
4.3.4	Choix du type de boucle	51
4.4	Le test et l'alternative	51
4.4.1	Le test	51
4.4.2	L'alternative	52
4.4.3	Réduction du nombre d'instructions primitives	53
4.5	Retour sur l'affectation	54

List of Figures

Chapter 1

Introduction à la programmation

Nous allons voir, dans ce chapitre, ce qu'est la *programmation* et comment on met en place la programmation 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.

1.1 Notions théoriques

1.1.1 Qu'est-ce que la programmation ?

Un ordinateur peut être utilisé dans des buts divers : pour faire tourner des jeux (c'est souvent de nos jours la première prise de contact), 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.

1.1.2 Façons de programmer

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

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. Son inconvénient essentiel 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 faudra traduire en langage machine à un certain moment. Ce fut l'idée d'ALAN TURING au début des années 1950.

On a pendant longtemps distingué 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.

1.1.3 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 **langages naturels** [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. Si vous voulez en savoir plus sur les programmes objets il faut aller en cours d'**Architecture** (des ordinateurs) dans lequel vous apprendrez 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 textes, 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 textes) qui est (plus ou moins) **exécutable**, c'est-à-dire que 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) déclenchera l'exécution de ce qui est décrit dans le programme (source).

1.1.4 Diversité des langages de programmation

De même qu'il existe de nombreux langages naturels, 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 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.

1.1.5 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, un peu, 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).

1.2 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 textes, 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.

1.3 Quelques compilateurs pour PC

Nous allons utiliser des micro-ordinateurs compatible PC pour les séances de travaux pratiques. La gamme de compilateurs de langage C est très importante pour ces ordinateurs. Les systèmes d'exploitation utilisés sont MS-DOS (en fait abandonné depuis 2000, mais qui a encore un intérêt pédagogique), Windows et Linux.

Le compilateur de référence a longtemps été Turbo C (et ses dérivés), à la fois pour MS-DOS et Windows. Le compilateur GNU gcc est la référence dans le monde Unix (donc Linux) ; il existe également pour MS-DOS. Le compilateur Visual C++ pour Windows a un intérêt pour la conception des logiciels pour Windows.

1.3.1 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 C, ce qui en fit le premier système d'exploitation à ne pas être écrit en langage d'assemblage.

1.3.1.1 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 supposons ici que Linux est installé sur un compatible PC.

Le compilateur C peut dépendre de la machine. Le meilleur compilateur, en particulier suivant la norme ANSI, est le compilateur GNU qui a l'avantage supplémentaire d'être gratuit.

1.3.1.2 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 *xemacs*, on lance l'éditeur, à partir du répertoire sur lequel on veut sauvegarder le programme, en écrivant (en supposant que le prompteur soit \$) :

```
$ xemacs 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 :

```
$ a.out
```

et 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*').

1.3.2 Le compilateur Turbo C et ses dérivés

Devant le succès de son progiciel **Turbo-Pascal**, compilateur Pascal pour les micro-ordinateurs compatibles PC, la société BORLAND a conçu un compilateur pour le langage C pour ces mêmes ordinateurs, tout naturellement appelé **Turbo C**. Au moment de l'engouement pour les langages objets, elle a conçu un compilateur pour le langage C++, appelé **Turbo C++**, permettant également la compilation des programmes en langage C. Le premier compilateur n'est plus commercialisé, mais ceci n'est pas gênant grâce à la compatibilité que nous venons d'indiquer. Devant le développement de l'interface graphique Windows de Microsoft, une version pour cette interface a été conçue, à la fois pour fonctionner à partir de cette interface mais aussi pour la programmation système de celle-ci. La version pour MS-DOS seule n'est plus commercialisée. Nous nous référerons ici essentiellement à la version Turbo C++ pour Windows ou Borland C++ (avec le compilateur Borland C++, on choisit si le programme objet peut être lancé à partir de MS-DOS ou de telle version de *Windows*, ce qui n'est pas le cas avec Turbo C++).

1.3.2.1 Installation

L'installation des huit disquettes (pour Windows 3.1) ou du CD-ROM (pour Windows 95 ou 98, et donc pour les versions ultérieures de Windows, le progiciel n'ayant plus suivi) se fait de façon automatique en exécutant le programme `install` (situé sur la première disquette dans le cas d'une distribution par disquettes) à partir de Windows. Notons qu'il faut au moins 27 Mo de libre sur le disque dur pour l'installation complète (de Turbo-C++ 3.1).

1.3.2.2 Démarrage

Après s'être placé sous Windows 3.11, il suffit de cliquer deux fois sur l'icône de **Turbo C++** du groupe **Turbo C++**. On se trouve alors dans ce qu'on appelle un *environnement intégré* ou **IDE** (pour l'anglais *Integrated Development Environment*).

Pour Windows 95 ou 98, on cherche dans le menu déroulant des programmes.

Exercice 1.- Manipuler et décrire cet environnement intégré.

1.3.2.3 Premier exemple

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

Première étape : écrire le programme.- Le programme source vu plus haut peut être écrit avec n'importe quel éditeur de texte mais, en général, on utilisera l'éditeur de texte intégré de l'IDE Borland.

Deuxième étape : sauvegarder le programme.- Considérons le cas où on veut le sauvegarder sur une disquette 3"5. Dans le menu déroulant *Options/Répertoires*, à la ligne *Répertoire destination* écrire **a:** si le lecteur de disquettes 3"5 est A. Cette étape sera faite une fois pour toutes.

En utilisant le menu déroulant *Fichier/Enregistrer sous* enregistrer le programme ci-dessus sous le nom **essai.c**.

Troisième étape : la compilation.- Pour compiler ce programme il suffit d'aller dans le menu déroulant intitulé *Compiler* et de choisir *Compiler*.

Exercice 2.- Décrire ce qui se passe.

On remarquera que la compilation est assez longue et qu'elle se fait, si on se fie à ce qui est affiché, en deux passes. Si vous n'avez pas commis d'erreurs un nouveau fichier se trouve sur votre disquette, le fichier **essai.obj**.

Quatrième étape : exécution du programme.- Ceci se fait en choisissant *Exécuter* dans le menu déroulant *Exécuter*.

Exercice 3.- Décrire ce qui se passe.

Apparemment une nouvelle étape de compilation s'effectue puis on accède à une fenêtre intitulée (**inactive A:\ESSAI.EXE**) dans laquelle est écrit ce que nous voulions, à savoir 'Bonjour'. On peut vérifier qu'il existe encore un nouveau fichier sur la disquette, intitulé **essai.exe**. Comme son nom l'indique, il est directement exécutable comme on peut le tester (à partir de Windows).

1.3.3 Le compilateur DJGPP

Le compilateur DJGPP (pour *Delorie J Gnu Orogramming Platform*) est le portage du compilateur **gcc** pour le système d'exploitation MS-DOS, la difficulté étant que MS-DOS est un système 16 bits alors que **gcc** a été prévu pour les systèmes 32 bits.

1.3.3.1 Installation

Le compilateur DJGPP se trouve sur le site :

<http://www.delorie.com/djgpp/>

On a intérêt à commencer par la page :

<http://www.delorie.com/djgpp/zip-picker.cgi>

pour savoir quels sont les fichiers `.zip` à télécharger (et à les télécharger, tous dans un *même* répertoire, par exemple dans `c:\>logiciels\djgpp`).

On chargera ensuite la page :

<http://www.delorie.com/djgpp/doc/ug/intro/installing-djgpp.html>

pour savoir comment installer `djgpp` sur MS-DOS ou Windows.

1.3.3.2 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 (`edit` de MS-DOS ou `wordpad` de Windows, par exemple). Sous Windows, on se placera dans une fenêtre de ligne de commandes, évidemment.

Chapter 2

Le langage C comme calculatrice... peu pratique

À l'origine le problème de la programmation consiste d'abord à faire effectuer les opérations élémentaires de façon automatique (c'est-à-dire sans qu'on ait à le faire comme à l'école élémentaire, car cela prend du temps). Il existe d'ailleurs des *machines d'aide aux calculs* avant l'apparition des ordinateurs, par exemple les bouliers et la machine à additionner de Pascal. Vous savez que ces dernières machines sont de nos jours avantageusement remplacées par les **calculatrices** (quatre opérations ou **scientifiques** ou même **alphanumériques** avec plus de possibilités ; nous oublierons les **calculatrices programmables** qui sont de vrais ordinateurs, en plus petits). Du point de vue de la technologie, ces calculatrices sont en fait une retombée de la conception des ordinateurs.

Un ordinateur peut faire la même chose qu'une calculatrice, et ceci quel que soit le langage de programmation, par exemple en langage C. Mais, comme nous nous en sommes aperçus à propos de notre premier exemple, il faut bien avouer que ceci est plus pénible à mettre en oeuvre. Ce n'est pas grave ; comme nous le verrons dans les chapitres suivants, l'intérêt des ordinateurs ne vient pas de l'émulation des calculatrices mais de la possibilité de combinaison d'opérations élémentaires déjà présentes sur les calculatrices.

Nous allons étudier de façon critique dans ce chapitre l'analogie de la calculatrice, c'est-à-dire les opérations possibles. Vous n'êtes pas obligés de lire ce chapitre en entier avant d'aborder le suivant.

2.1 Premiers éléments de langage C

Reprenons le programme `essai.c` que nous avons vu lors de la mise en route du langage C :

```
#include <stdio.h>

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

et commentons-le.

2.1.1 Forme d'un programme

Syntaxe.- Conformément à l'exemple que nous venons de rappeler, tout programme C doit au minimum être de la forme suivante :

```
#include <stdio.h>

void main(void)
{
    INSTRUCTION
}
```

Nous détaillerons au fur et à mesure ce qu'est une INSTRUCTION, le programme ci-dessus en donnant un exemple.

Premiers commentaires.- 1^o) Comme pour l'étude des langues naturelles, la **syntaxe** nous dit qu'elle est la forme d'un programme valide pour le langage de programmation donné. Un **programme valide** sera traduit en un programme objet par le compilateur ; sinon on aura une **erreur de syntaxe**.

2^o) Il n'y a pas de nom de programme. Un programme C est une suite de fonctions avec une fonction principale dont le nom est le mot réservé **main**.

3^o) Il faut indiquer la nature des arguments et du résultat de cette fonction. Dans notre exemple nous avons **void** deux fois pour indiquer qu'il n'y a ni argument ni résultat.

4^o) Les accolades ouvrante '{' et fermante '}' indiquent le début et la fin d'un **bloc**.

5^o) L'instruction **printf** est une instruction d'écriture, plus précisément d'écriture formatée (d'où le **f**).

6^o) Les constantes mots (ici *Bonjour*) se placent entre guillemets verticaux " (le même symbole pour le guillemet ouvrant et le guillemet fermant).

7^o) Les constantes mots n'ont pas besoin d'être formatées, et qu'importe ce que cela veut dire pour l'instant, le langage C étant orienté manipulation des mots. C'est pourquoi notre premier exemple en C est l'affichage d'un texte et non pas un calcul numérique.

8^o) Le langage C étant lié au système d'exploitation UNIX, la différence entre les majuscules et les minuscules est importante. Par exemple **main** ne doit pas être écrit **Main**.