

## 2 Introduction au C

C est un langage de programmation impératif généraliste, de bas niveau<sup>1</sup>. Inventé au début des années 1970 pour réécrire Unix, C est devenu un langage très utilisé, encore de nos jours. De nombreux langages plus modernes comme C++, C#, Java et PHP ou JavaScript ont repris une syntaxe proche au C et reprennent en partie sa logique. C offre au développeur une marge de contrôle importante sur la machine (notamment sur la gestion de la mémoire) et est de ce fait utilisé pour réaliser les «fondations» (compilateurs, interpréteurs...) de ces langages plus modernes.

### 2.1 Structure d'un programme en C

Les programmes écrits en C sont généralement sauvegardés avec l'extension ".c" (exemple `tp1.c`). Le fichier `tp1.c` peut être créé à partir de n'importe quel éditeur de texte (Notepad++, emacs, gedit...).

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 int main(){
4     int d;
5     for(d = 1 ; d <= 100 ; d = d +1){
6         if (100 % d == 0)           // <--- on teste si d divise 100
7             printf("%i ", d);      // <--- on affiche d le cas échéant
8     }
9     printf("\n");
10    return EXIT_SUCCESS;
11 }
```

Prog. 2.1 – Un premier programme en C

#### Exercice 1

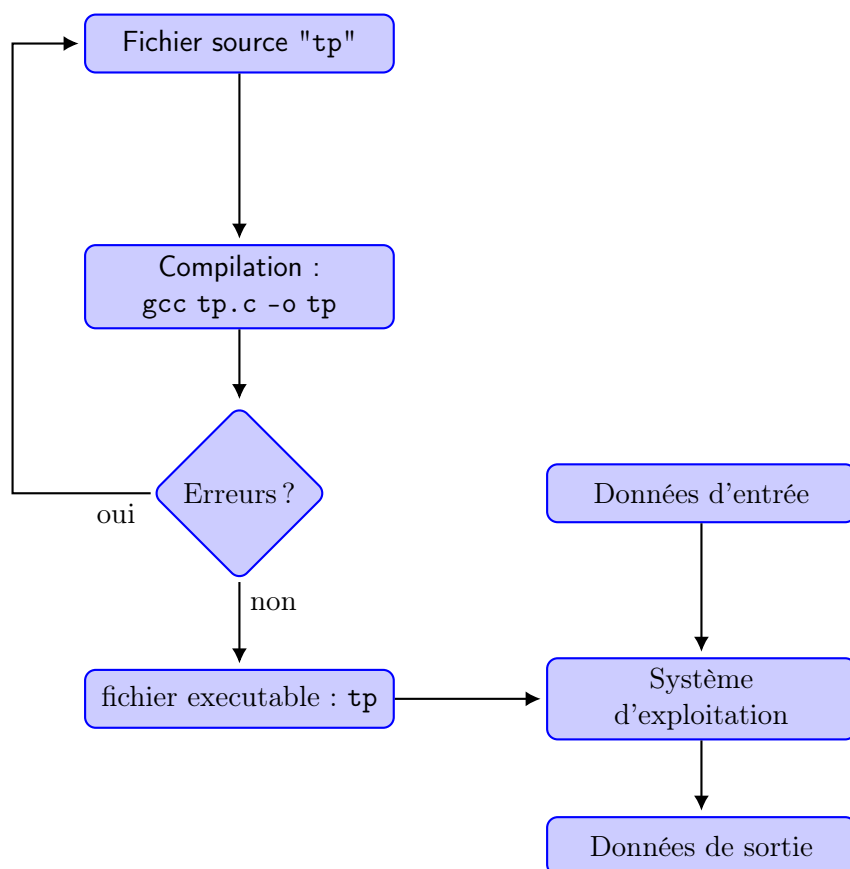
Intuitivement, que produit ce programme ?

Quelques explications sur ce programme :

- `#include<stdio.h>` : pour « Standard Input/Output Header » ou « En-tête Standard d'Entrée/Sortie ». Cette ligne permet d'insérer les fonctions, les constantes et les macros standard d'entrée-sortie.
- `#include<stdlib.h>` : cette bibliothèque contient des constantes (par ex : `EXIT_SUCCESS`, `EXIT_FAILURE`), des fonctions (`malloc`, `rand`...) pour exécuter diverses opérations dont la conversion, la génération de nombres pseudo-aléatoires, l'allocation de mémoire, le contrôle de processus, la gestion de l'environnement et des signaux, la recherche et le tri.
- `int main(){... }` : La fonction `main` est la fonction principale du programme : elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel à la fonction

---

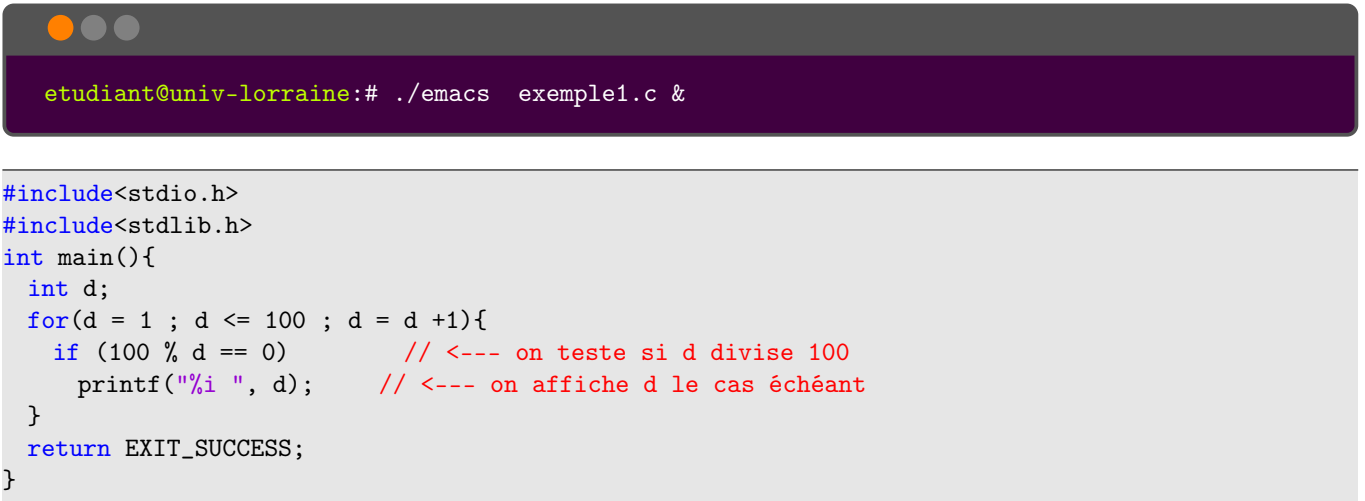
1. Un langage de programmation de bas niveau ne fournit que peu d'abstraction par rapport au jeu d'instructions du processeur de la machine. Les langages de bas niveau sont à opposer aux langages de haut niveau, qui permettent de créer un programme sans tenir compte des caractéristiques particulières (registres, etc) de l'ordinateur censé exécuter le programme.



## 2.3 Un exemple complet

On reprend le Prog. 2.1 page 1.

1. On saisit le **code source** suivant à l'aide d'un éditeur de texte (emacs par exemple). On enregistre le fichier sous le nom `exemple1.c`.



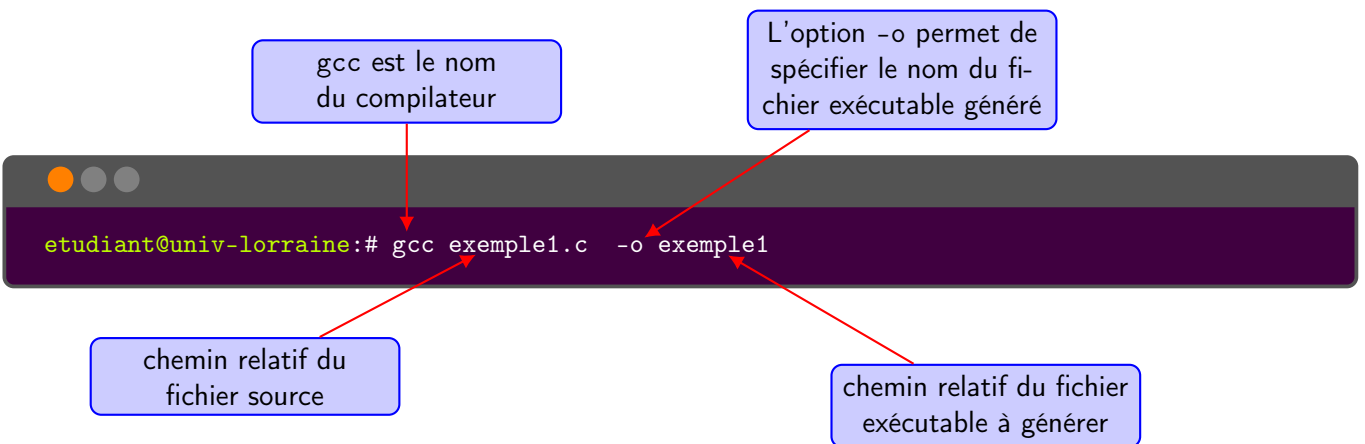
```

etudiant@univ-lorraine:~$ ./emacs exemple1.c &

1 #include<stdio.h>
2 #include<stdlib.h>
3 int main(){
4     int d;
5     for(d = 1 ; d <= 100 ; d = d +1){
6         if (100 % d == 0)           // <--- on teste si d divise 100
7             printf("%i ", d);      // <--- on affiche d le cas échéant
8     }
9     return EXIT_SUCCESS;
10 }
  
```

Prog. 2.2 – Un premier programme en C

2. Dans la même fenêtre de terminal (puisque on a utilisé le caractère `&`) on compile le code source à l'aide de la commande suivante :



```

etudiant@univ-lorraine:~$ gcc exemple1.c -o exemple1
  
```

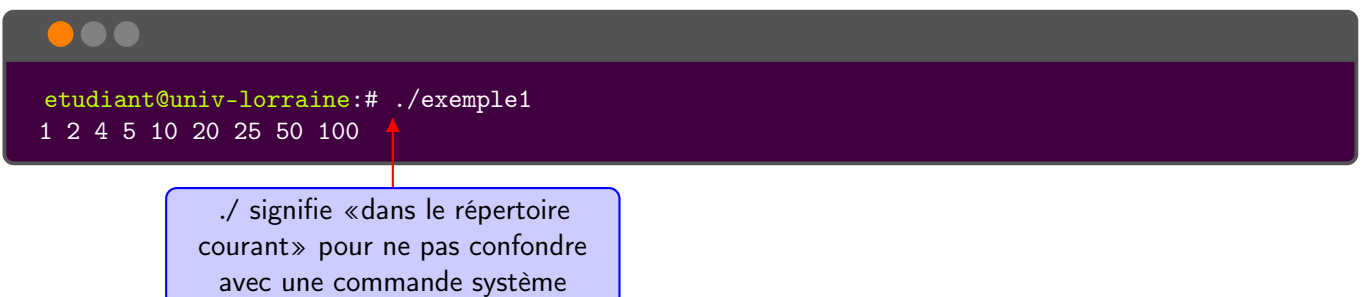
gcc est le nom du compilateur

L'option `-o` permet de spécifier le nom du fichier exécutable généré

chemin relatif du fichier source

chemin relatif du fichier exécutable à générer

3. On exécute le code avec la commande suivante :



```

etudiant@univ-lorraine:~$ ./exemple1
1 2 4 5 10 20 25 50 100
  
```

`./` signifie « dans le répertoire courant » pour ne pas confondre avec une commande système

## 2.4 Quelques règles basiques sur la syntaxe

- [1] Chaque instruction se termine par un ";".
- [2] Les instructions sont comprises entre une accolade ouvrante "{" et une accolade fermante "}".
- [3] Certains mots sont réservés (voir ci-dessous) et ne peuvent être utilisés comme noms de variables ou de fonctions.

Il est important de respecter la syntaxe (le compilateur ne vous pardonnera rien !). Il est tout aussi important de respecter des règles d'écritures même si elles ne seront pas toutes vérifiées par le compilateur. Elles permettent une meilleure lisibilité de votre programme et facilitent les phases de débogage de votre code C.

- [1] Pensez à bien indenter votre code.
- [2] Le nom d'une variable ou d'une fonction ne peut pas commencer par un chiffre.
- [3] Le nom des variables ou fonctions ne peut contenir d'espace ni de caractères accentués.
- [4] Utilisez des noms explicites pour chacune de vos variables.
- [5] Si vous souhaitez donner un nom complexe à une variable ou une fonction utilisez le tiret du bas "\_" ou les majuscules. Exemple `note_informatique` ou `noteInformatique`. N'utilisez pas le tiret "-" qui correspond à l'opération soustraction.
- [6] Aérez votre code, laissez des espaces entre, par exemple, la partie déclaration des variables, instructions, affichages...

## 2.5 Mots réservés

Ce sont les mots prédéfinis du langage C. Ils ne peuvent être réutilisés pour des identifiants. Ils sont relatifs aux différents concepts du langage :

- [1] type des données : `char const double float int long short signed unsigned void volatile`
- [2] instructions de boucle : `do for while`
- [3] sélections : `case default else if switch`
- [4] ruptures de séquence : `break continue goto return`
- [5] divers : `sizeof enum struct typedef union`

## 2.6 Aide-mémoire

### 2.6.1 Les commentaires

```
1 // un commentaire sur une seule ligne
2
3 /* un commentaire
4 sur plusieurs
5 lignes
6 */
```

## 2.6.2 Les variables

### Les types simples

Notation algorithmique	C
booléen	bool (nécessite un <code>#include &lt;stdbool.h&gt;</code> )
caractère	char
entier naturel	unsigned int size_t (pour les indices d'un tableau, p. ex.)
entier (entier relatif)	int
réel	double

### La déclaration des variables :

```
1 int x ; // déclaration d'un entier relatif
2 double y ; // déclaration d'un réel
```

### Affectation de variables

```
1 x = 2 ; // la variable x reçoit la valeur 2
2 x = e + 1 ; // où e est une variable préalablement déclarée et initialisée
```

### Les opérations de base sur les booléens

Notation algorithmique	C
vrai, faux	true, false (nécessite <code>#include &lt;stdbool.h&gt;</code> )
et, ou, non	&&,   , !
=, ≠, <, ≤, >, ≥	==, !=, <, <=, >, >=

### Les opérations de base sur les entiers

Notation algorithmique	C
+, -, ×, //, mod	+, -, *, /, %

### Les opérations de base sur les réels

Notation algorithmique	C
+, -, ×, /	+, -, *, /

## 2.6.3 La génération pseudo-aléatoire de nombres

```
1 #include <stdlib.h> // pour la fonction rand()
2 int main(){
3     double x; // déclaration d'un double x
4     x = ((double)rand() / (RAND_MAX)) ; // x reçoit un réel choisi dans [0.; 1.[
5     x = ((double)rand() / (RAND_MAX)) * 90. + 10. ; // x reçoit un réel choisi dans [10.; 100.[
6
7     int a ; // déclaration d'un entier a
8     a = rand () % 101 ; // a reçoit un entier choisi dans {0,...,100}
9     a = rand () ;% 41 - 20 ; // a reçoit un entier choisi dans {-20, ... , 20}
10 }
```

### 2.6.4 Les entrées/sorties

On suppose que `n` est une variable de type `int` et que `f(x)` est une expression de type `double`. On affiche `n = <valeur de n>` et `f(x) = <valeur de f(x)>` de la manière suivante :

```
1 printf ("n = %d et f(x) = %f", n, f(x))
```

Si on souhaite ajouter un retour à la ligne après l’affichage :

```
1 printf ("n = %d et f(x) = %f\n", n, f(x))
```

En C, les `%<lettre>` indiquent l’endroit où doit être inséré une valeur. La lettre dépend du type, par exemple :

- `%c` correspond à un caractère (un `char`),
- `%u` correspond à un entier naturel (p. ex., un `unsigned int` ou un `size_t`),
- `%d` correspond à un entier relatif (p. ex., un `int`),
- `%f` correspond à un flottant (p. ex., un `double`),
- `%s` correspond à une chaîne de caractères.

### 2.6.5 Les conditionnelles

```
1 if (<condition>)  
2 {  
3   <instructions>  
4 }
```

```
1 if (<condition>)  
2 {  
3   <instructions1>  
4 }  
5 else  
6 {  
7   <instructions2>  
8 }
```

### 2.6.6 Les boucles

```
1 while (<condition>)  
2 {  
3   <instructions>  
4 }
```

```
1 // on suppose que debut <= fin  
2 // i <- debut  
3 // tant que i <= fin, faire <instructions> puis incrémenter i  
4 for (i = debut ; i <= fin ; i = i + 1){  
5   <instructions>  
6 }
```

```
1 // on suppose que debut >= fin  
2 // i <- debut  
3 // tant que i <= start, faire <instructions> puis décrémenter i  
4 for (i = debut ; i >= fin ; i = i - 1){
```

```

5 <instructions>
6 }

```

### 2.6.7 Fonctions et procédures

Notation algorithmique	C
Fonction <code>nomFonction(a1 : type1, a2 : type2) : type</code>	<code>type nomFonction(type1 a1, type2 a2)</code>
Procédure <code>nomProcEDURE(a1 : type1, a2 : type2)</code>	<code>void nomProcEDURE(type1 a1, type2 a2)</code>

où `nomFonction` (resp. `nomProcEDURE`) est le nom de la fonction (resp. procédure), `a1` et `a2` les noms des paramètres, `type1` et `type2` sont les types des paramètres et `type` est le type du résultat de la fonction.

### 2.6.8 Les tableaux à une dimension

```

1 double T[10] ; // T est un tableau de double de taille 10
2 int i; // L'accès en lecture et en écriture au ième élément de T se fait par : T[i]
3 for (i = 0 ; i < 10 ; i = i + 1)
4 {
5     T[i] = 0. ; // initialisation de chaque cas à 0 (en flottant, d'où le point)
6 }
7 for (i = 0 ; i < 10 ; i = i + 1)
8 {
9     printf("T[%d] = %f\n", i, T[i]);
10 }

```

## 2.7 Exercices

Code C des exercices sur les entiers du chapitre 1.