Introduction au C

CM1 - IG3

Marc Hartley
marc.hartley@umontpellier.fr

Polytech Montpellier

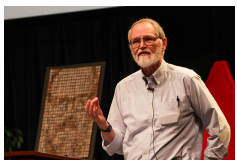
February 7, 2025

Overview

Historique

Dennis Ritchie

- Développé initialement par Dennis Ritchie et Brian Kernighan en 1972.
- Créé pour reprogrammer le système d'exploitation UNIX, qui était initialement écrit en assembleur.



Brian Kernighan

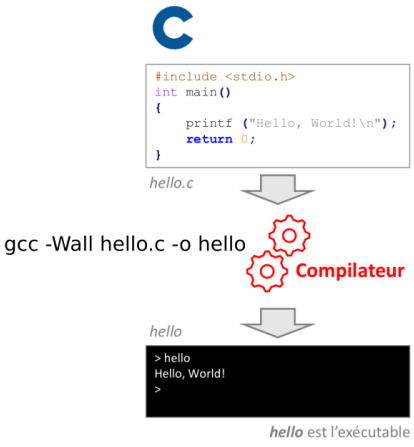
Historique

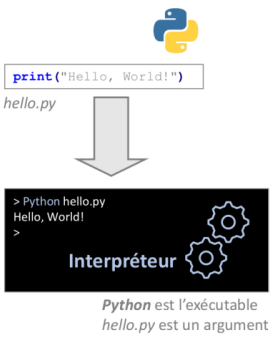
"Si Dennis avait décidé de consacrer cette décennie à des mathématiques ésotériques, Unix aurait été mort-né."

Bjarne Stroustrup, inventeur du C++

Langage compilé v. interprété

Le C est un langage **compilé** (par opposition à **interprété**, e.g. *Python*, *Javasript*)





Langage compilé v. interprété

Le C est un langage **compilé** (par opposition à **interprété**, e.g. *Python*, *Javasript*)

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portu
marco.l.couto@inesctec.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacomes@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portu
jpf@dei.uc.pt

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Structure d'un programme

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello World!");
    return 0;
}
```

Variables et types de données

```c
int main() {
    short court = 32767;
    unsigned short courtNonSigne = 65535;
    int entier = 42;
    unsigned int entierNonSigne = 100u;
    long longEntier = 9876543210L;
    unsigned long longEntierNonSigne = 9876543210UL;
    float reel = 3.14f;
    double grandReel = 2.718281828459045;
    char caractere = 'A';
    char chaine[20] = "Bonjour Polytech!";
    const float CONSTANTE = 2.71828182846f;
    return 0;
}
```

Le type `void` est un type qui représente "rien", ou "indéfini", ou "c'est compliqué".

Opérations de base

- Arithmétiques :
  - +, −, *, /, %, ...
- Binaires :
  - &, |, <<, >>, ...
- Logiques :
  - || (OR), && (AND), ! (NOT), ...
- Comparaisons :
  - <, <=, >, , >=, ==, !=, ...

```
if (x * x + y * y <= r * r) {
    // Dans un cercle de rayon "r"
}
if (input == 'q' && isRunning) {
    // L'utilisateur veut quitter
}
```

Structures de contrôle

```c
// i++ équivaut à i = i + 1 ou i += 1
for(int i = 0; i < 5; i++) {
    printf("Hello world 5 fois!");
}
int maVariable = 0;
while (maVariable < 5) {
    maVariable ++;
}
maVariable = 0;
do {
    maVariable ++;
} while(maVariable < 5);
```

```c
if (maVariable == 3) {
    // ...
} else if (maVariable <3) {
    // ...
} else {
    // ...
}
```

Structures de contrôle

```
for ([initialisation] ; [conditions] ; [incrément] ) {
}
// Usage normal
for (unsigned int i = 0; i < 100; i = i + 1) {
    // ...
}
// Très borderline
for (unsigned short j = 10; j < 0xFFFF; j += 5) {
    // ...
}
```

```
// Prison à vie
for (;;) {
    // Boucle infinie
}
```

```
int i;
char c;
for (i = 1, c = 'z'; i < 10 && c != 'a'; i++, c--) {
    // ...
}
```

Structures de contrôle

```
switch (maVariable) {
    case 'a':
        instructionA();
        break;
    case 'b':
        instructionB();
    case 'c':
        instructionC();
        break;
    case 'd':
    case 'e':
        instructionDorE();
        break;
    default:
        printf("Erreur");
}
```

Contrôle de flux

- Quelques fonctions présentes dans la bibliothèque standard :
  `#include <stdio.h>`
- Afficher du texte :
  `printf()`
- Récupérer une entrée utilisateur :
  `scanf()`
- Récupérer une frappe clavier :
  `getchar()`

Contrôle de flux - printf()

- %d : notation décimale
- %o : notation octale
- %x : notation hexadécimale
- %u : notation décimale non signée
- %f : notation flottante (%2.4f $\rightarrow$ 12,4587)
- %c : caractère ASCII
- %s : chaîne de caractères

Contrôle de flux - printf()

```c
#include <stdio.h>

int main() {
    // Déclaration et initialisation de différents types de données
    int entier = 42;
    float reel = 3.14f;
    double grandReel = 2.718281828459045;
    char caractere = 'A';
    char chaine[20] = "Bonjour Polytech!";

    // Affichage des variables de différents types
    printf("Valeur entière: %d\n", entier);
    printf("Valeur réelle (float): %.2f\n", reel);
    printf("Valeur réelle (double): %.15f\n", grandReel);
    printf("Caractère: %c\n", caractere);
    printf("Chaîne de caractères: %s\n", chaine);

    return 0;
}
```

Contrôle de flux - scanf()

```c
#include <stdio.h>

int main() {
    int entier;
    float reel;
    char chaine[20];

    scanf("%d", &entier);
    scanf("%f", &reel);
    scanf("%s", chaine);

    return 0;
}
```

Contrôle de flux - getchar()

```c
#include <stdio.h>

int main() {
    char touche;

    printf("Appuyez sur Q pour quitter");
    c = getchar();
    switch (c) {
        case 'q':
            printf("On quitte!");
            break;
        default:
            instructionParDefaut();
            break;
    }
    return 0;
}
```

Fonctions

```
[type retour] [nom de fonction] (
                                    [type parametre] [nom parametre] ,
                                    [type parametre] [nom parametre] ,
                                    ... )
{
    // Du code ...
    return [une valeur de type "type retour"] ;
}
```

Les pointeurs

```c
int x = 42;  // Une variable
int* p = &x; // Un pointeur vers l'adresse de "x"
```

Un pointeur est une *variable* qui contient une *adresse mémoire*.
L'opérateur & devant une variable donne son adresse mémoire.
L'opérateur * APRÈS un *type* indique que c'est un pointeur de ce type.

Généralement un pointeur contient l'adresse d'une autre variable.

Les pointeurs

```c
int x = 42;  // Une variable
int* p = &x; // Un pointeur vers l'adresse de "x"
```

L'opérateur * DEVANT un nom de pointeur donne accès au bloc mémoire sur lequel pointe le pointeur. (opérateur d'*indirection* ou de *déréférence*)

```c
printf("%d", x); // Maintenant "x" vaut 42
*p = 10;
printf("%d", x); // Maintenant "x" vaut 10
```

Les pointeurs

```c
int x = 42;  // Une variable
int* p = &x; // Un pointeur vers l'adresse de "x"
int y = *p; // "y" vaut maintenant 42
*p = 10;

// Maintenant "x" vaut 10, et "y" vaut toujours 42
printf("%d \n %d", x, y);
```

Les pointeurs

En C les pointeurs et les tableaux sont fortement liés, en effet

Une variable tableau n'est rien d'autre qu'un pointeur *constant* du type du tableau

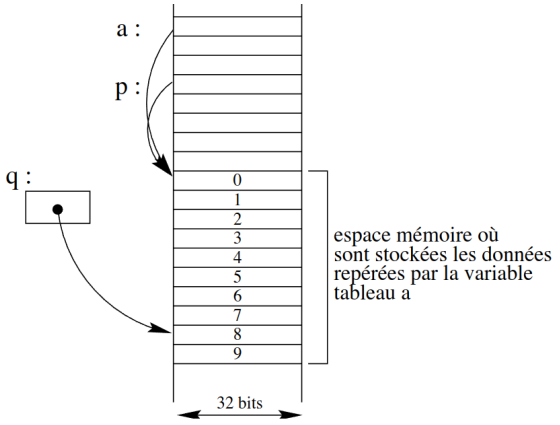
```c
#include <stdio.h>
void main (void)
{
    int a[10]={0,1,2,3,4,5,6,7,8,9};
    int* p;
    p = a;
    printf("%d\n",*p); // Affiche 0
    p++;
    printf("%d\n",*p); // Affiche 1

    printf("%d \n %d", a[5], p[5]); // Affiche "5" et "6"
}
```

Un tableau n'est pas un type, ce n'est qu'un espace mémoire où sont stockées des données d'un certain type repérées par une variable tableau qui n'est rien d'autre qu'un pointeur constant.

Les pointeurs

```
int a[10] = {0,1,2,3,4,5,6,7,8,9};
int *p;
int *q;
p=a;
q=a+8;
```



Allouer et libérer la mémoire

Puisqu'un tableau n'est rien d'autre qu'un espace mémoire où l'on stocke des données repérées par un pointeur, on peut grâce à des fonctions d'allocation mémoire créer dynamiquement des tableaux. Le prototype de ces fonctions est :

```
#include <stdlib.h>
void* malloc(size_t nombre_d_octets_a_allouer);
void* calloc(size_t nombre_d_octets_a_allouer);
```

Ainsi la syntaxe générale pour allouer un tableau de *n* éléments de type type est :

```
T=(type *) malloc(n*sizeof(type));
```

Allouer et libérer la mémoire

Tout espace mémoire alloué dynamiquement doit être libéré par l'utilisateur.

On utilise pour cela la fonction

```
void* free(void* p);
```

Allouer et libérer la mémoire

Exemple de création d'un tableau de 10 entiers :

```
void main (void)
{
    int* p;
    int n = 10;
    p = (int *) malloc(n * sizeof(int));
    for(int i = 0; i < 10; i++){
        p[i] = i;
    }
    printf("p[0]=%d, p[1]=%d, ...\n", p[0], p[1]);

    free(p);
}
```

n aurait pu être calculé, ou bien être saisi au clavier. C'est là l'intérêt de l'allocation dynamique de mémoire.

Allouer et libérer la mémoire

Un tableau à deux dimensions se déclare de la façon suivante :

`type tab[n1][n2];`

où type est le type des éléments du tableau, et n1 et n2 des constantes entières indiquant le nombre de lignes et de colonnes. On accède à un élément par : `tab[i][j]` (PAS de tab[i,j] !)

```
int tab[2][3] = {
{ 0, 1, 2},
{10, 11, 12}
};
```

En général on préfère utiliser les tableaux de pointeurs.

Chaines de caractères

Une chaine de caractères est en réalité un tableau de `char` !

```
char chaine[20] = "Bonjour Polytech!";
```

Le dernier caractère étant le caractère de *fin de chaîne*, ou *caractère nul* : '\0'.

Chaines de caractères

Les chaînes de caractères sont à considérer comme des pointeurs, donc ch1=ch2; copie les pointeurs et pas les caractères !

```c
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
    /* p = adresse où est stockée
    la chaine constante "chaine p" */
    char p[] = "chaine p";
    char* q = p;

    printf("%s, %s\n", p, q);
    p[3] = 'I';
    printf("%s, %s\n", p, q);
}
```

Affiche :

```
chaine p, chaine p
chaIne p, chaIne p
```

Chaines de caractères

Quelques fonctions sur les chaînes de caractères :

```
char * strcpy(char *dst, const char *src)
char * strncpy(char *dst, const char *src, size_t len)
size_t strlen(const char *s)
char * strcat(char *s, const char * append)
int strcmp(const char *s1, const char *s2)
int strncmp(const char *s1, const char *s2, size_t len)
```

et pour toutes les connaître : `man 3 string`