Référence rapide C

Syntaxe, pointeurs, gestion mémoire, bibliothèque standard

Bases

Hello World

```c
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Compiler et exécuter

```c
gcc -o app main.c          # compiler
gcc -Wall -Wextra -std=c17 main.c  # strict
./app                       # exécuter
```

Commentaires

```c
// commentaire sur une ligne (C99+)
/* commentaire
   multi-ligne */
```

Types de données

Types primitifs

`char`	1 octet, caractère ou petit entier
`short`	Au moins 16 bits
`int`	Au moins 16 bits (typiquement 32)
`long`	Au moins 32 bits
`long long`	Au moins 64 bits (C99+)
`float`	IEEE-754 32 bits
`double`	IEEE-754 64 bits
`_Bool / bool`	**`0`** ou **`1`** (utiliser **`<stdbool.h>`** pour **`bool`**)

Types à largeur fixe (stdint.h)

`int8_t, uint8_t`	Signé/non signé exact 8 bits
`int16_t, uint16_t`	Exact 16 bits
`int32_t, uint32_t`	Exact 32 bits
`int64_t, uint64_t`	Exact 64 bits
`size_t`	Non signé, résultat de **`sizeof`**

Conversion de types

```c
int i = (int)3.14;          // cast explicite
double d = (double)5 / 2;   // 2.5, pas 2
char c = (char)65;          // 'A'
```

Flux de contrôle

If / Else

```c
if (x > 0) { printf("positive\n"); }
else if (x == 0) { printf("zero\n"); }
else { printf("negative\n"); }
```

Switch

```c
switch (choice) {
    case 1:  printf("one\n"); break;
    case 2:  printf("two\n"); break;
    default: printf("other\n");
}
```

Boucles

```c
for (int i = 0; i < 10; i++) { }
while (condition) { }
do { } while (condition);
```

Instructions de saut

`break`	Quitter la boucle ou le switch le plus proche
`continue`	Passer à l'itération suivante
`return`	Quitter la fonction avec une valeur optionnelle
`goto label`	Sauter à un label (utiliser avec parcimonie)

Fonctions

Déclaration et définition

```c
int add(int a, int b);         // prototype
int add(int a, int b) {
    return a + b;
}
```

Pointeurs de fonctions

```c
int (*op)(int, int) = add;
int result = op(3, 4);       // appelle add(3, 4)
typedef int (*MathFn)(int, int);
MathFn fn = add;
```

Fonctions statiques

```c
// visible uniquement dans cette unité de traduction
static int helper(int x) {
    return x * 2;
}
```

Pointeurs

Bases des pointeurs

```c
int x = 42;
int *p = &x;        // p pointe vers x
printf("%d\n", *p); // déréférencement : 42
*p = 100;           // x vaut maintenant 100
```

Arithmétique des pointeurs

```c
int arr[] = {10, 20, 30};
int *p = arr;
printf("%d\n", *(p + 1));  // 20
printf("%d\n", p[2]);      // 30 (idem *(p+2))
```

Patterns courants de pointeurs

`int *p = NULL`	Pointeur null (toujours initialiser)
`void *`	Pointeur générique (doit être casté pour utilisation)
`const int *p`	Pointeur vers constante (ne peut modifier la valeur)
`int *const p`	Pointeur constant (ne peut réassigner le pointeur)
`int **pp`	Pointeur vers pointeur (double indirection)

Tableaux et chaînes

Tableaux

```c
int nums[5] = {1, 2, 3, 4, 5};
int matrix[2][3] = {{1,2,3}, {4,5,6}};
int len = sizeof(nums) / sizeof(nums[0]);
```

Fonctions de chaînes (string.h)

`strlen(s)`	Longueur (sans le terminateur null)
`strcpy(dst, src)`	Copier une chaîne (non sécurisé, préférer **`strncpy`**)
`strncpy(dst, src, n)`	Copier au plus n caractères
`strcat(dst, src)`	Concaténer des chaînes
`strcmp(a, b)`	Comparer : 0 si égales, <0 ou >0 sinon
`strchr(s, c)`	Trouver la première occurrence d'un caractère
`strstr(haystack, needle)`	Trouver une sous-chaîne

Littéraux de chaînes

```c
char greeting[] = "hello";     // tableau mutable
const char *msg = "world";     // pointeur vers littéral
char buf[64];
snprintf(buf, sizeof(buf), "%s %s", greeting, msg);
```

Structures

Définition et utilisation

```c
struct Point { double x; double y; };
struct Point p = {1.0, 2.0};
printf("(%g, %g)\n", p.x, p.y);
```

Typedef

```c
typedef struct {
    char name[50];
    int age;
} Person;
Person p = {"Alice", 30};
```

Pointeurs de structures

```c
void set_age(Person *p, int age) {
    p->age = age;    // opérateur flèche
}
```

Énumérations et unions

```c
enum Color { RED, GREEN, BLUE };
union Data { int i; float f; char c; };
// les membres d'une union partagent la même mémoire
```

Gestion de la mémoire

Allocation dynamique (stdlib.h)

```c
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) { /* gérer l'erreur */ }
arr = realloc(arr, 20 * sizeof(int));
free(arr);
arr = NULL;   // éviter les pointeurs pendants
```

Fonctions d'allocation

`malloc(size)`	Allouer de la mémoire non initialisée
`calloc(count, size)`	Allouer et initialiser à zéro
`realloc(ptr, size)`	Redimensionner un bloc alloué
`free(ptr)`	Libérer la mémoire allouée

Pièges courants

`Fuite mémoire`	Oublier de **`free()`** la mémoire allouée
`Double free`	Appeler **`free()`** deux fois sur le même pointeur
`Pointeur pendant`	Utiliser un pointeur après **`free()`** — mettre à NULL
`Dépassement de tampon`	Écrire au-delà des limites allouées

E/S fichier

Lire des fichiers

```c
FILE *f = fopen("data.txt", "r");
if (!f) { perror("open"); return 1; }
char line[256];
while (fgets(line, sizeof(line), f)) printf("%s", line);
fclose(f);
```

Référence rapide C

Écrire dans des fichiers

```
FILE *f = fopen("out.txt", "w");
fprintf(f, "value: %d\n", 42);
fputs("hello\n", f);
fclose(f);
```

Modes d'ouverture de fichier

`"r"`	Lecture (le fichier doit exister)
`"w"`	Écriture (tronque ou crée)
`"a"`	Ajout (crée si nécessaire)
`"rb", "wb"`	Lecture/écriture binaire
`"r+"`	Lecture et écriture (le fichier doit exister)

Préprocesseur

Directives

```
#include <stdio.h>       // en-tête système
#include "myheader.h"    // en-tête local
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Compilation conditionnelle

```
#ifdef DEBUG
    printf("debug: x = %d\n", x);
#endif
#ifndef HEADER_H   /* garde d'inclusion */
#define HEADER_H   /* ... */ #endif
```

Macros courantes

`__FILE__`	Nom du fichier source courant
`__LINE__`	Numéro de ligne courant
`__func__`	Nom de la fonction courante (C99+)
`__DATE__`	Chaîne de date de compilation
`NULL`	Constante pointeur null
`sizeof(x)`	Taille d'un type ou d'une variable en octets