



# Notes abruere - Piscine 07/2022

## Ressources de l'école

 [Pisciniers Onboarding](#)

### ▼ PDF Norminette

Intra Signin New  
 <https://elearning.intra.42.fr/notions/the-norm/subnotions/norm-v3/pdfs/Norm%20V3.pdf>

### ▼ Videos Shell & Git

PEDAGO@42 - S-00  
Share your videos with friends, family, and the world  
 <https://www.youtube.com/playlist?list=PLVQYiy6xNUxxhwI0PGmXb5isUdVwmsg8>

**Documentation**  
**P2P Evaluation**  
Before (as evaluated student)



### Shell

clear : effacer les lignes  
man : manuel commande  
pwd : chemin d'accès  
ls : liste  
ls -p : afficher avec des / pour les dossiers  
ls -l : afficher listing avec details : permissions-nbrerelations-abruere-nbgroupe-taille-date-nomfichier  
ls -la : liste de fichiers cachés  
cd : change directory  
cd .. : remonter d'un niveau dans l'arborescence  
rm / rmdir : effacer fichier / dossier  
chmod  
structure permission (ls -l) : duuugggoooo@ (Directory, Users, Groups, Other)  
r=read, w=write, x=exe, d=directory...  
r=4, w=2, x=1, rien=0  
ex: rwx rwx- r— = 421 420 400 = 764

+=rajouter, -=retirer  
 chmod u/g/o +/- r/z/x/...  
 chmod a-x : retire x de tous les groupes (a pour all)  
 chmod 777 : ajoute toutes les permissions  
 echo truc : afficher truc  
 echo \$truc : affiche la valeur de la variable truc  
 cat : affiche contenu d'un fichier  
 cat -e : affiche également retours à la ligne (?) et caractères non imprimables  
 cat > *fichier*, puis Entrée → taper ce qu'on veut ajouter, puis Ctrl+Z  
 touch : créer fichier  
 touch -t : changer date d'accès fichiers [CC]YY]MMDDhhmm[.SS]  
 attention: si date antérieure à plus de 6 mois, affiche l'année au lieu de l'heure  
 Créer un sous-shell : sh / en sortir : exit  
 env : affiche toutes les variables de mon environnement (sauf celles du shell non-exportées)  
 export : rajouter à l'environnement une variable utilisée dans le shell

## **Git**

Dans l'ordre :

touch *fichier*

Premier commit

**git add *fichier***

**git commit -m "msg du commit" *fichier*** (premier commit)

Commits suivants

**vim *fichier*** (modif directe)

Pour sortir en enregistrant : Echap + :wq!

Pour sortir sans enregistrer :q! (ou :qa!)

**git add *fichier***

**vim *fichier***

**git add *fichier***

... etc jusqu'à avoir fait les modifs qu'on veut

**git commit -m "nom du commit"**

## ▼ Videos C

### ▼ C00 - Variables, Opérateurs, Conditionnelles, Boucles, Fonctions...

PEDAGO@42 - C-00

Share your videos with friends, family, and the world

<https://www.youtube.com/playlist?list=PLVQYiy6xNUxz5wbzZn4tfUhF4djgzscB->

```
//VIDEOS C00
VARIABLES
Pour compiler en affichant tous les msgs d'erreur :
```

```

gcc -Wall -Werror -Wextra main.c

En cas de succes, le fichier a.out est cree, on peut faire les 2 en meme temps :
gcc -Wall -Werror -Wextra main.c && ./a.out

Recuperer la taille d'une variable :
printf("%lu\n", sizeof(var)); //Attention printf necessite <stdio.h> et ne passe pas a la moulinette

Argument %d dans printf : affiche le nombre correspondant dans la table ascii
Argument %c dans printf : affiche le caractere contenu dans

man ascii

Le caractere \n est un retour a la ligne en C, pas besoin de ' ou "

int a[6] //Tableau a une dimension de 6 caracteres
int b[7][29] //Tableau a deux dimensions de 7*29 caracteres

OPERATEURS
*Operateurs mathematiques : +, -, /, *, %
*Incrementation : ++ ; Decrementation : --
*Operateurs binaires
    aff_bin(var) //Afficher code binaire d'une variable
    aff_bin(-128) -> 11111111 ; aff_bin(-1) -> 10000000"
    aff_bin(11 & 6) équivaut à 00001011 & 00000010 -> 00000010 //Operateur "ET"
    aff_bin(11 | 6) équivaut à 00001011 | 00000010 -> 00001111 //Operateur "OU"
    aff_bin(11 ^ 6) équivaut à 00001011 ^ 00000010 -> 00001101 //Operateur "XOR"
    aff_bin(~11) équivaut à ~00001011 -> 11110100 //Operateur de negation
    aff_bin(11 << 2) équivaut à 00001011 << 2 -> 00101100 //Operateur de decalage vers la gauche
    aff_bin(11 >> 2) équivaut à 00001011 >> 1 -> 00000101 //Operateur de decalage vers la droite
*Operateur ! :
    Si var=/=0, !var renvoie 0
    Si var=0, !var renvoie 1
*Operateurs de comparaison :
    a == b : teste si a=b, renvoie 1 si vrai, 0 si faux
    a != b : teste si a est different de b, renvoie 1 si vrai (a different de b), 0 si faux (a=b)
    a >= b : teste si a est superieur ou egal a b, idem 1 si vrai, 0 si faux
    a <= b : teste si a est inferieur ou egal a b, idem
    a && b : teste si a ET b sont differents de 0, 1 si vrai, 0 si faux
        //Note : si a=0, renvoie 0 sans meme regarder b
    a || b : teste si a OU b sont differents de 0, 1 si vrai, 0 si faux
        //Note : si a=/=0, renvoie 1 sans meme regarder b

CONDITIONNELLES
IF
    if (condition)
    {
        trucs a faire 1;
    }
    else if (condition) //Optionnel, on peut en mettre autant qu'on veut
    {
        trucs a faire 2;
    }
    else
        trucs a faire si aucune condition n'est vérifiée;

SWITCH
    switch (var)
    {
        case $var:
            trucs a faire 1;
            break; //Optionnel : permet de sortir du switch des que rencontré
        case $var:
            trucs a faire 2;
            break; //idem
        ...
        default;
            trucs a faire si aucune condition n'est vérifiée;
            break; //idem
    }

TERNAIRE
    b = condition ? valeur_si_vrai : valeur_else ; //if, else
    b = condition ? (condition_else_if ? valeur_si_else_if_vrai : valeur_si_else_if_faux) : valeur_else ;
        //Note1: valeur_si_else_if_faux = valeur_si_condition_vrai
        //Note2: difficilement lisible des qu'il y a des else if

BOUCLES
WHILE
    while (condition) //Attention aux boucles infinies
    {

```

```

        trucs à faire;
    }

DO WHILE
do
{
    trucs à faire;
}
while (condition);

FOR
for (valeur_depart; valeur_arrivee; trucs_a_faire_a_chaque_boucle)
{
    trucs à faire;
}

CONTROLEURS DE BOUCLES
break; //Permet de sortir du switch des que rencontré
continue; //Permet de relancer la boucle à son départ
Difficile à maîtriser mais quand même mentionné : label: ... goto label;

FONCTIONS
Structure canonique
void fonction(void)
{
    trucs à faire;
}
int main (void)
{
    trucs à faire avec fonction definie en dehors du main;
    return(0); //Met fin au main
}

Prototypes
Exemple : void fonction(void);
* Si les fonctions utilisees dans le main sont en-dessous des fonctions definies, il faut les appeler avant le main car le C lit
* Si les fonctions utilisees dans le main sont dans un autre fichier, il faut compiler les deux fichiers ensemble
-> gcc main.c fichier2.c

Return
"Il faut s'assurer d'avoir une homogeneite dans les 'return' a la fin des fonctions"
"Exemple : return(0) ou return(42) si c'est une fonction int qu'on utilise dans une fonction int"

Arguments
"La encore il s'agit d'harmoniser les types de variables utilisees"

Autres mots clés
* "static_fct" -> le static indique que cette fonction ne pourra être utilisée tel quel que dans ce fichier
* "static type var" -> variable globale utilisee dans le cadre d'une seule fonction
* "const type var", ou plutot "type const var" -> definit une constante, on ne peut pas attribuer de nouvelles valeurs a la fonction

```

Type	Taille par defaut
char	1 octet (8 bits)
int	4 octets (32 bits)
float	4 octets (32 bits)
double	8 octets (64 bits)

#### C data types - Wikipedia

In the C programming language, data types constitute the semantics and characteristics of storage of data elements. They are expressed in the language syntax in form of declarations for memory locations or variables. Data types also determine the types of operations or methods of processing of data elements.

W [https://en.wikipedia.org/wiki/C\\_data\\_types](https://en.wikipedia.org/wiki/C_data_types)



#### ▼ C01 - Pointeurs



▶ <https://www.youtube.com/playlist?list=PLVQYiy6xNUxytsXWxZx6odBJMbRktIHTs>

```
//VIDEOS C01
POINTEURS
Gestion de la mémoire
Un programme a potentiellement accès virtuellement à l'ensemble de la mémoire de l'\architecture : 32 bits = 4 Go, 64 bits = bcp
"stack" = part du haut des adresses et descend
"heap" = part du bas des adresses et monte
Mémoire soit sur ram soit sur DD, mais virtuellement uniquement, pas physiquement (role du MMU)

Déréférencement : 🐱🐱🐱
int *adr -> pointeur = renvoie la valeur stockée à l'adresse adr
int **adr2 -> adresse de pointeur = renvoie la valeur stockée à l'adresse *adr2, sachant que adr2 est déjà une adresse
...
int *****adr11 -> adresse d'adresse ... d'\adresse de pointeur
Attention : n'affecte que les adresses, si on modifie la valeur utilisée à l'adresse ptr, toutes les variables à cette adresse

Arithmétique des pointeurs
* + / - : Déplacement dans la mémoire
* int *ptr à l'\adresse 0x7...38 -> ptr+2 à l'\adresse 0x7...40

Tableaux
* *tab : valeur de tab[0] ; *(tab+i) : valeur de tab[i]
* Equivalences d'écriture entre tab[] et *tab (cf ci-dessous)

Chaines de caractères
* Ensemble de caractères finissant par '\0'
* Bus error : impossible de déclarer char *str, puis appeler str[i]

Utilisation générale
* Modifier dans une fonction une variable provenant d'une autre fonction
* Manipuler des adresses

Void
* void * permet de ne pas spécifier le type de pointeur pour l'utiliser avec n'\importe quel type
```

```
#include <stdio.h>

int main(void)
{
    int      a;
    int      *ptr;

    a = 56;
    ptr = &a;
    printf("%d\n", *ptr);
    *ptr = 78;
    printf("%d\n", a);
    return (0);
}
```

```
~42>$gcc main.c && ./a.out
56
~42>$gcc main.c && ./a.out
56
~42>$gcc main.c && ./a.out
56
78
~42>$
```

Déréférencement 🐱🐱🐱

```

#include <stdio.h>

int main(void)
{
    int tab[3];
    int *tab2[2];

    tab[2] = 145;
    tab2[1] = tab;
//    tab2[1][2] = 18;
//    *(tab2 + 1) + 2) = 18;
    printf("%d\n", *(tab + 2));
    return (0);
}

main.c
"main.c" 15L, 211C w

```

and this is the same thing!! It's a bit hard to really understand it

Equivalences d'écriture des tableaux

## ▼ C05 - Récursion

PEDAGO@42 - C-05

Share your videos with friends, family, and the world

<https://www.youtube.com/playlist?list=PLVQYiy6xNUxxZbeH9b0VC-nC6QsJRw5Ah>



//VIDEOS C05

RÉCURSION

- \* Idée générale : la fonction s'appelle elle-même à des indices différents
- \* Importance du cas de sortie de la boucle récursive
- \* Exemple de fonction récursive (vs itérative)

## ▼ C06 - Arguments du main

PEDAGO@42 - C-06

Share your videos with friends, family, and the world

<https://www.youtube.com/playlist?list=PLVQYiy6xNUxxDlCkkCX262SI90TsIIYUW>



//VIDEOS C06

ARGUMENTS DU MAIN

- 1- int argc = argument counter -> nb d'arguments (inclue le nom du programme)
- 2- char \*\*argv = argument value -> valeur des arguments (argv[0] = nom du programme)
- 3- char \*\*environ = environnement du shell

## ▼ C07 - Malloc, Free

PEDAGO@42 - C-07

Share your videos with friends, family, and the world

<https://www.youtube.com/playlist?list=PLVQYiy6xNUxzNYF00nlmx624twFlamqlt>



```
//VIDEOS C07
MALLOC, FREE
int *tab;
tab = malloc(sizeof(type) * nb_delements_auxquels_allouer_memoire);
free(tab); //dès qu'on n'a plus besoin de la mémoire il faut la vider
```

## ▼ C08 - Préprocesseur, Data structures

PEDAGO@42 - C-08

Share your videos with friends, family, and the world

 [https://www.youtube.com/playlist?list=PLVQYiy6xNUxxMI\\_GiGGb2hxMcd3lwNYRy](https://www.youtube.com/playlist?list=PLVQYiy6xNUxxMI_GiGGb2hxMcd3lwNYRy)



```
//VIDEOS C08
PREPROCESSEUR
Compilation
gcc : 3 phases de compilation
1- Preprocess : cpp file.c
2- Compilation : gcc -c file1.c file2.c : crée des fichiers ".o" (object)
3- Link : ld file1.o file2.o: lie plusieurs fichiers ".o"

#include
* < ... .h > = directives de base du compilateur
* "....h" = répertoire actuel ou répertoire fourni en -I (include)
* .h = "header"

#define TEXT1 text2
* sans variables : en écrivant TEXT1, le compilateur va lire text2
* avec variables : "x" définit une variable
ex: #define LOL(x) "lol %d\n", x -> printf(LOL(42)); affiche 42

#if, #else, #elif et #endif
int main(void)
{
    #if CONDITION (ex: POUIC < 42 ; ex: defined(POUIC))
        faire des trucs
    #elif CONDITION
        faire des trucs
    #else
        faire des trucs
    #endif
        suite de la fct
}

#ifndef : sert à executer/afficher ou non certaines parties du code selon définition
int main(void)
{
    #ifdef POUIC
        executer cette partie du code (à condition d'avoir #define POUIC au dessus)
    #else
        faire plutôt ça
    #endif
        faire ça dans tous les cas
}

#ifndef : sert à protéger les .h en les rendant uniques avec un ID
#ifndef IDENTIFIANT
# define IDENTIFIANT //ne pas oublier l'espace !!
... //contenu du .h
#endif

DATA STRUCTURES
Typedef //sert à renommer un type
typedef type_qu_on_vient_renommer nouveau_nom;
//attention utilisable seulement dans le scope en question et ses sous-scopes

Struct //sert à démultiplier le nb de variables
struct s_structre
{
    type1 var1;
    type2 var2;
    ...
}
```

```

};

//idée pratique : combiner typedef et struct
typedef struct s_structure
{
    ...
} t_structure;
//pour appeler les variables (non-pointeurs) : t_struct.var
//pour appeler les variables (pointeurs) : t_struct->var

Enum //sert à déclarer des constantes
enum e_enumeration
{
    constante1 = int1,
    constante2 = int2,
    constante3 = int3
};
//si on n'attribue pas de valeur, fait +1 par rapport à la précédente

Union //fonctionne comme les struct mais en utilisant le même espace mémoire pour tout (par 1 pour chaque)
typedef union s_union
{
    ...
} t_union;
//s'utilise rarement (jeux vidéos ?)

```

## ▼ C09 - Makefile, Bibliothèques

PEDAGO@42 - C-09

Share your videos with friends, family, and the world

 [https://www.youtube.com/playlist?list=PLVQYiy6xNUxw6n6q\\_i8wek6U7t7CeAXhU](https://www.youtube.com/playlist?list=PLVQYiy6xNUxw6n6q_i8wek6U7t7CeAXhU)



```

//VIDEOS C09
MAKEFILE
SRCS      = file1.c file2.c ... //liste de tous les fichiers .c
OBJS      = ${SRCS:.c=.o}        //règle pour transformer les .c en .o
NAME      = executable
RM        = rm -f
CC        = gcc
CFLAGS    = -Wall -Wextra -Werror -I ${HEADER}
${NAME}: ${OBJS}
    ${CC} ${CFLAGS} -o ${NAME} ${OBJS}
all: ${NAME}
clean:           //supprime les .o
    ${RM} ${OBJS}
fclean: clean   //force la suppression des .o
    ${RM} ${NAME}
re:    fclean all //force la recompilation
.PHONY   all clean fclean re
//optionnel:
.c.o = ${CC} ${CFLAGS} -c $< -o $< //redéfinition de la règle implicite

BIBLIOTHEQUES
Etape 1: créer les .o files
Etape 2: "ar rcs libstr.a file1.o file2.o ..." //la partie "lib" est obligatoire

```

## ▼ C11 - Pointeurs sur fonctions

PEDAGO@42 - C-11

Share your videos with friends, family, and the world

 <https://www.youtube.com/playlist?list=PLVQYiy6xNUxx8sKygTdqtOPytqN7sb0Vz>



```

//VIDEOS C11
POINTEURS SUR FONCTIONS
Fonctionne presque comme des pointeurs, à une notation près : (*ptr)(arg)

```

```

#include <stdio.h>

int fct(char c)
{
    printf("%c\n", c);
    return (0);
}

int main(void)
{
    int (*ptr)(char);

    ptr = &fct;
    (*ptr)('o');
    return (0);
}

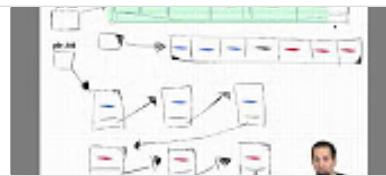
```

## ▼ C12 - Listes chainées

PEDAGO@42 - C-12

Share your videos with friends, family, and the world

 <https://www.youtube.com/playlist?list=PLVQYiy6xNUxwmUOmyYSaI6gD1UyfF9MSj>



```

//VIDEOS C12
LISTES CHAINÉES
Théorie
    *Structure contenant l'adresse du prochain maillon (pointeur)
    *Le dernier élément de la liste vaut 0
    *Similaire au tableau mais utilisé différemment : si on veut une valeur on doit parcourir toutes les précédentes
Pratique
    *Screenshot ci-dessous : utilisation "à la main"
    *Fonction pour afficher les éléments
void aff_list(struct s_list *begin)
{
    while(begin)
    {
        printf("%d\n", begin->i);
        begin = begin->next;
    }
}

```

```

xterm
struct s_list
{
    int     i;
    char    c;
    struct s_list *next;
};

int main(void)
{
    struct s_list elem1;
    struct s_list elem2;
    struct s_list elem3;
    struct s_list *begin;

    begin = &elem1;
    elem1.next = &elem2;
    elem2.next = &elem3;
    elem3.next = 0;

    elem1.i = 98;
    elem2.i = 109;
    elem3.i = 42;

    printf("%d\n", begin->i);
    printf("%d\n", begin->next->i);
    printf("%d\n", begin->next->next->i);

    return (0);
}
main.c

```

```

xterm
~42>$gcc main.c && ./a.out
42
~42>$gcc main.c && ./a.out
98
109
42
~42>$

```

## ▼ C13 - Arbres

PEDAGO@42 - C-13

Share your videos with friends, family, and the world

<https://www.youtube.com/playlist?list=PLVQYiy6xNUxzusAgMiybYwkLvuMFbVat9>

C

Arbres

42

```

//VIDEOS C13
ARBRES
Sortes de listes chainées avec plusieurs "branches"
typedef struct s_binary_tree
{
    int             a;
    struct s_binary_tree *left;
    struct s_binary_tree *right;
} t_binary_tree;

```

# Trouvailles

## ▼ General

- Verrouiller session : [cmd] + [ctrl] + [Q]
- Fichiers caches : [cmd] + [shift] + [.]
- Screenshot : [shift] + [cmd] + [3]
- Task manager : [cmd] + [alt] + [esc]
- cd /sgoinfre/Perso/johnson → open Goose.app
- Nouveau bureau : [ctrl] + [up]

## ▼ Shell

- Trouvaille d'Anthony pour se mettre sur git :

Pour identifiant git et cle ssh :

**ssh-keygen**

**faire entree deux fois**

**cat/Users/abruere/.ssh/id\_rsa.pub**

**Ensuite il faut copier la ligne de code qui va apparaitre et coller dans intra42 → setting → SSH KEYS**

Pour creer dossier de travail pour chaque days :

**cd Documents**

**git clone lien.git.vogsphere.projet.sur.l.intra Nom.dossier**

**yes**

**cd Nom.dossier**

- Ecrire dans un fichier : **echo contenu >> fichier** OU **cat >> fichier**, puis ecrire **contenu**
- Remplacer contenu d'un fichier : **echo contenu > fichier** OU **cat > fichier**, puis ecrire **contenu**
- Deplacer un fichier : **mv fichier dossier**
- Creer un fichier d'une certaine taille (ex:40) : **dd if=/dev/zero of=MonFichier bs=1 count=40**
  - Mieux : faire 40 espaces dans le fichier (1 caractere ASCII = 1 caractere)
- Liens entre fichiers : **In** ; Lien fichier symbolique : **In -s**
- Pour eviter que les commandes (chmod, touch) ne soient repercutées sur le fichier du lien symbolique (symlink), faire -h
- Saut de ligne : **|** ; Afficher saut de ligne (\$) : **printf '\n'**
- Soumettre exos :

Avant toute chose, bien checker d'ajouter les fichiers caches au .gitignore (.DS\_Store, .gitignore, etc...)

**git add dossier** (pour tous les dossiers exos)

**git commit -m "commit XXX"**

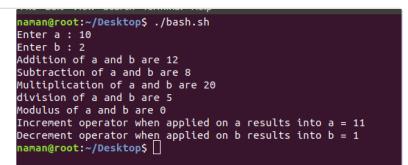
**git push origin master**

- Operateurs arithmétiques, relationnels, booléens, au niveau du bit, de test de fichiers :

#### Opérateurs de base dans les scripts Shell

Il existe 5 opérateurs de base dans les scripts bash/shell : Opérateurs arithmétiques Opérateurs relationnels Opérateurs booléens Opérateurs au niveau du bit Opérateurs de test de fichiers 1. Opérateurs arithmétiques : Ces opérateurs sont utilisés pour effectuer des opérations

 <https://fr.acervolima.com/opérateurs-de-base-dans-les-scripts-shell/>



```
nanan@root:~/Desktop$ ./bash.sh
Enter a : 10
Enter b : 2
Addition of a and b are 12
Subtraction of a and b are 8
Multiplication of a and b are 20
Division of a and b are 5
Modulus of a and b are 0
Increment operator when applied on a results into a = 11
Decrement operator when applied on b results into b = 1
nanan@root:~/Desktop$
```

- Alias

- cat ~/.bashrc
  - alias l='clear&&pwd&&ls -la'
  - alias g='clear&&gcc -Wall -Wextra -Werror \*.c &&./a.out'

## ▼ C language

### ▼ Fonction **write** :

**write**

The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition Copyright © 2001-2004 The IEEE and The Open Group, All Rights reserved.

 <https://pubs.opengroup.org/onlinepubs/007904975/functions/write.html>

#### Required Include Files

**#include <unistd.h>**

#### Function Definition

```
ssize_t write(int fildes, const void *buf, size_t nbytes);
```

int fildes	The file descriptor of where to write the output. You can either use a file descriptor obtained from the open system call, or you can use 0, 1, or 2, to refer to standard input, standard output, or standard error, respectively.
const void *buf	A pointer to a buffer of at least nbytes bytes, which will be written to the file.
size_t nbytes	The number of bytes to write. If smaller than the provided buffer, the output is truncated.
return value	Returns the number of bytes that were written. If value is negative, then the system call returned an error.

En des termes comprehensibles :

int fildes	"filedes" pour "file descriptor" stdin (0), stdout (1) et stderr (2) - stdin (0) c'est l'entrée standard, celle utilisée dans les programmes, à ne pas confondre avec les arguments de la ligne de commande - stdout (1), la sortie standard, ce que vous voyez généralement dans le terminal (ex: ls écrit la sortie dans l'stdout) - stderr (2), la sortie d'erreur standard (ex: gcc écrit les erreurs dans l'stderr) Après si tu ouvres un fichier avec, par exemple la fonction "open", tu peux write dans ce fichier en passant son numéro (son "file descriptor") en premier paramètre de write 0, 1 et 2 sont des file descriptors un peu spéciaux Pour le moment toujours 1 car on veut afficher
const void *buf	&var Pour "adresse de var"
size_t nbytes	Nombre d'octets de la variable en question
return value	Retourne la valeur

- Fonction **int** :

<https://www.c-programming-simple-steps.com/int-in-c.html>

- Fonction **void** :

*What is **void** in C programming? It means “no value”, “no parameters”, or “no type”, depending on the context.*

- A void function does not return a value
- A void function does not accept parameters
- A pointer does not have a specific type and could point to different types

What is void in C programming?

What is void in C programming? It means "no type", "no value" or "no parameters", depending on the context. We use it to indicate that: This is probably the most used context of the void keyword. Here we use it as a return type of a function.

<https://www.c-programming-simple-steps.com/what-is-void.html#:~:text=void%20in%20C-,What%20is%20void%20in%20C%20programming%3F,function%20does%20not%20accept%20parameters>



- **NO BOOL** → int
- while(str[i]) ne fonctionne que pour des chaînes de caractères, pas des chaînes d'entiers (car ne se terminent pas forcément par \0)
- **int max : 2147483647**  
**int min : -2147483648**  
**char max : 2048 octets**
- ▼ Automatiser les tests

1. Faire un *main* solide avec des printf de tous les cas possibles (fonction écrite en MAJUSCULES dans le main)
  2. Créer un fichier contenant les valeurs renvoyées par ma fonction
    - a. Dans le vim : **#define MAJUSCULE ft\_fonction**
    - b. En dehors du vim :
      - i. **gcc -Wall -Werror -Wextra ft\_fonction.c**
      - ii. **./a.out > ft\_fonction.txt**
  3. Créer un second fichier contenant les valeurs renvoyées par la fonction à copier
    - a. Dans le vim :
      - i. Ajouter la bibliothèque correspondante : **#include <lib.h>**
      - ii. **#define MAJUSCULE fonction**
    - b. En dehors du vim :
      - i. **gcc (-Wall -Werror -Wextra) ft\_fonction.c**
      - ii. **./a.out > fonction.txt**
  4. **diff ft\_fonction.txt fonction.txt**
  5. Selon les erreurs renvoyées : ouvrir les .txt avec vim, faire :set nu pour trouver les lignes
- **gcc -Wall -Werror -Wextra -fsanitize=address file.c**

```
//CONSEILS POUR UTILISER LE DEBUGGER - by alegra
lldb:
gcc -g *.c //compiler avec le flag qui permet de debugger
lldb a.out // lance le debugger
breakpoint set -f mon_fichier.c -l20 //si l'on souhaite mettre un bp ligne 20
settings set target.run-args 1 2 3 // pour rajouter des arguments (optionnel)
run // lance le programme
n // passe a l'instruction suivante
thread step-in // comme n mais permet de rentrer a l'interieur des sous fonction
exit
```