Déclaration de variable:
```
type nom_variable = valeur;
```

Déclaration de constante:
```
const type NOM_VARIABLE = valeur;
```

Commentaire simple:
```
// commentaire sur une ligne
```

Commentaire bloc:
```
/* bloc de commentaire */
```

Déclaration pointeur:
```
type* p_nom_pointeur = valeur;
```

Contenue:
```
*(pointeur)
```

Adresse:
```
$variable
```

tableau 1D:
```
type nom_tableau[taille] = {valeurs};
nom_tableau[index] = valeur;
```

tableau 2D:
```
type nom_tableau[N][M] = {valeurs};
nom_tableau[x][y] = valeur;
```

Types de base:

Type	Taille (octets)	Valeur min	Valeur max	Affichage
unsigned short int / unsigned short	2	0	65535	%d
short int / short	2	-32768	32767	%d
unsigned long int /unsigned long	4	0	4294967295	%ld
long int / long	4	-2147483648	2147483647	%ld
unsigned int	2 / 4	0 / 0	65535/ 4294967295	%d
int	2 / 4	-32768 / -2147483648	32767 / 2147483647	%d
unsigned char	1	0	255	%c
signed char	1	-128	127	%c
char	1	-128 / 0	127 / 255	%c
float	4	-3.4e38	3.4e38	%f
double	8	-1.7e308	1.7e308	%f

Structure conditionnelle:
```
if(condition)
{
}
else if(condition)
{
}
else
{
}
```

Structure switch:
```
switch(variable)
{
   case valeur:
   break;
   ...
  default:
  break;
}
```

Opérateur conditionnel:
```
(condition) ? true : false;
```

Boucle while:
```
while(condition)
{
}
```

Boucle do while:
```
do
{
}while(condition);
```

Boucle for:
```
for(init; condition; increment)
{
}
```

break: sortir de la boucle courante
continue: sauter a la fin de la boucle courante

Les conditions:

Symbole	Signification	Exemple
==	est égale à	(password == 1852)
!=	est différent de	(password != 1852)
<	est inférieur à	(age_utilisateur < 18)
>	est supérieur à	(age_utilisateur > 18)
<=	est inférieur ou égale à	(age_utilisateur <= 0)
>=	est supérieur ou égale à	(age_utilisateur >= 0)

Les opérateurs logiques:

Symbole	Signification	Exemple
&&	and (et)	(age>=0 && age<100)
||	or (ou)	(password==1852 || age>=18)
!	not (non)	!(age_utilisateur < 18)

Fonctions:
```
type nomFonction(params)
{
  …
  return valeur;
}
```

Prototype:
```
type nomFonction(params);
```

Préprocesseur:
```
#define NOM_DEFINE valeur
#define NOM_MACRO(params) instructions
```

__LINE__: renvoie le numéro de la ligne ou elle est utilisée (entier).
__FILE__: renvoie le nom du fichier ou elle est utilisée (string).
__DATE__: renvoie la date de la compilation du fichier dans lequel elle est utilisée (string).
__TIME__: renvoie l'heure de la compilation du fichier dans lequel elle est utilisée (string).

module:
fichiers headers (.h): Prototypes des fonctions et enums publiques.
fichiers source (.c): Déclaration des fonctions, enums et variables globales privées.

structures:
```
typedef struct st_nom
{
  type nom;
  …
}Nom;
```

enums:
```
typedef enum NOM_ENUM
{
  ENUM_1: valeur,
  …
}Nom;
```

utilisation:
```
Type ma_structure = {valeurs_init}
ma_structure.element = valeur;

Type* pointeur = NULL;
pointeur->element = valeur;
```

STL (Standard Template Library): https://devdocs.io/c

string.h

int strln(string st): Retourne la taille de la string passé en paramètre en excluant le '\0'

char* strchr(string st, char c): Retourne la première sous chaîne qui commence par le caractère passé en argument. Si le Caractère n'est pas trouvé, elle retourne NULL.

int strcmp(string st1, string st2): Compare deux chaîne, retourne 0 si elles sont identiques, sinon elle retourne -1 dans le cas ou la première chaîne est avant dans l'ordre alphabétique et elle retourne 1 dans l'autre cas.

strcpy(string st1, string st2): Copie le contenue de la chaîne 2 dans la chaîne 1. Attention à ce que le tableau de chaine 1 soit suffisamment grand pour contenir chaîne 2.

strcat(string st1, string st2): Insére le contenue de la chaîne 2 devant la chaîne 1. Attention à ce que le tableau de chaine 1 soit suffisamment grand.

long strtol(string st, pt_fin, base): Convertit une chaîne en la valeur numérique entière qu'elle représente. **pt_fin**: la fonction s'en sert pour renvoyer la position du premier caractère qu'elle a lu et qui n'était pas un nombre. On peut mettre NULL si on ne souhaite pas l'utiliser. **base**: base avec laquelle le nombre est écrit dans la chaîne. le plus souvent base 10

double strtod(string st, pt_fin): Convertit une chaîne en la valeur numérique flottante qu'elle représente. **pt_fin**: la fonction s'en sert pour renvoyer la position du premier caractère qu'elle a lu et qui n'était pas un nombre. On peut mettre NULL si on ne souhaite pas l'utiliser.

stdio.h

printf(string template, params)
scanf(string template, &params)

sprintf(string dest, string template, params): Comme la fonction printf, sprintf permet de formater une chaîne de caractère mais au lieu de l'afficher dans la console, on l'écrit dans un tableau de char (string).

fgets(string st, int taille, stdin): lecture sécurisé, fgets retourne NULL si il y a une erreure lors de la lecture. **st**: sting de destination. **taille**: nombre maximum de caractère à lire.

file* fopen(string path, string mode): ouvrir le fichier **path**
fclose(file* fichier): permet de fermer le fichier

int fputc(char c, file* fichier): écrire le char c dans le fichier
int fputs(char* st, file* fichier): écrire la chaîne st dans le fichier
int fprintf(file* fichier, string template, params): Comme la fonction printf, fprintf permet de formater une chaîne de caractère mais au lieu de l'afficher dans la console, on l'écrit dans le fichier.

char fgetc(file* fichier): lit le prochain char dans le fichier
char*fgets(char* st, file* fichier): lit la prochaine chaîne dans le fichier
int fscanf(file* fichier, string template, *params): Comme la fonction scanf, fscanf permet de lire un ensemble de paramètres selon un template. sauf que ici, l'entrée, est un fichier.

long ftell(file* fichier): retourne la position de lecture courante du fichier
int fseek(file* fichier, long delta, int position): déplacer la position de lecture du fichier.
rewind(file* fichier): réinitialise la position de lecture du fichier à son début.

int rename(char* origin, char* dest): déplace le fichier **origin** vers **dest**.
int remove(char* path): supprime le fichier.

stdlib.h

void* malloc(taille_en_octet): permet d'allouer dynamiquement un espace mémoire dans le TAS. la fonction prend en paramètre la taille en octet de la zone à réserver et retourne l'adresse de la zone allouée ou NULL en cas d'échec.

free(pointeur): permet de libérer dynamiquement un espace mémoire alloué dans le TAS. Attention à ne pas passer un pointeur null à la fonction.

void* realloc(pointeur, taille_en_octet): permet de ré-allouer dynamiquement un espace mémoire dans le TAS. la fonction prend en paramètre le pointeur de la zone mémoire à ré-allouer et la taille en octet de la zone à réserver. La fonction retourne l'adresse de la nouvelle zone allouée ou NULL en cas d'échec.

.

void* calloc(nombre_element, taille_en_octet_element): comme la fonction malloc, calloc alloue une zone mémoire dans le TAS. Mais en plus, celle-ci initialise la mémoire allouée à la valeur '0000 0000' pour chaque octet alloué.

Tableau ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]