Bibliothèque ANSI-C

LISTE DES FONCTIONS 5

Liste des fonctions

<assert.h>

assert(x)

Macro
Si NDEBUG est définie, la macro est vide.
Si NDEBUG n'est pas définie, provoque un message si la condition x est fausse.

<ctype.h>

int **isalnum**(int);

Fonction.
Rend 'vrai' si l'argument ε (A-Z,a-z,0-9)

int **isalpha**(int);

Fonction.
Rend 'vrai' si l'argument ε (A-Z,a-z)

int **iscntrl**(int);

Fonction.
Rend 'vrai' si l'argument ε (caractères de contrôle - habituellement < 32)

int **isdigit**(int);

Fonction.
Rend 'vrai' si l'argument ε (0-9)

int **isgraph**(int);

Fonction.
Rend 'vrai' si l'argument ε (caractère imprimable sauf blanc - habituellement > 32)

int **islower**(int);

Fonction.
Rend 'vrai' si l'argument ε (a-z)

int **isprint**(int);

Fonction.
Rend 'vrai' si l'argument ε (caractère imprimable y compris blanc)

int **ispunct**(int);

Fonction.
Rend 'vrai' si l'argument ε (caractères imprimables) et ∉ (A-Z,a-z,0-9,blanc)

```
int isspace(int);
```

Fonction.
Rend ‘vrai’ si l’argument ε (blanc, form-feed, \n, \r, \t, V-tab)

```
int isupper(int);
```

Fonction.
Rend ‘vrai’ si l’argument ε (A-Z)

```
int isxdigit(int);
```

Fonction.
Rend ‘vrai’ si l’argument ε (0-9,A-F)

```
int tolower(int);
```

Fonction.
Rend l’argument converti en minuscule.

```
int toupper(int);
```

Fonction.
Rend l’argument converti en majuscule.

<math.h>

double **acos***(double);*

Fonction.
Rend l'arc cosinus.

double **cos***(double);*

Fonction.
Rend le cosinus.

double **asin***(double);*

Fonction.
Rend l'arc sinus.

double **sin***(double);*

Fonction.
Rend le sinus.

double **atan***(double);*

Fonction.
Rend l'arc tangente entre $-\pi/2$ et $\pi/2$.

double **tan***(double);*

Fonction.
Rend la tangente.

double **atan2***(double y, double x);*

Fonction.
Rend l'arc tangente de y/x, détermine le quadrant d'après les deux signes de x et y.

double **cosh***(double);*

Fonction.
Rend le cosinus hyperbolique.

double **sinh***(double);*

Fonction.
Rend le sinus hyperbolique.

double **tanh***(double);*

Fonction.
Rend la tangente hyperbolique.

double **exp***(double);*

Fonction.
Rend l'exponentielle.

```
double frexp(double value, int * exp);
```

Fonction.
Modifie value, tels que: value = x * 2 puissance exp.
Rend x compris entre [1/2 et 1[, ou 0; et Si value = 0, x vaudra 0.

```
double ldexp(double x, int exp);
```

Fonction.
Rend x fois 2 à la puissance exp.

```
double log(double);
```

Fonction.
Rend le logarithme base e.

```
double log10(double);
```

Fonction.
Rend le logarithme base 10.

```
double modf(double value, double *iptr);
```

Fonction, coupe value en partie entière et fractionnaire du signe de value.
La partie entière est pointée par iptr.
Rend la partie fractionnaire.

```
double pow(double x, double y);
```

Fonction puissance.
Rend x puissance y.

```
double sqrt(double x);
```

Fonction.
Rend la racine carrée.

```
double ceil(double x);
```

Fonction.
Rend le plus petit entier >= x.

```
double floor(double x);
```

Fonction.
Rend le plus grand entier <= x.

```
double fabs(double);
```

Fonction.
Rend la valeur absolue.

```
double fmod(double x, double y);
```

Fonction.
Rend le reste de x/y.

<setjmp.h>

typedef **··· jmp_buf***;*

Type, utilisé par setjmp et longjmp.

int **setjmp***(jmp_buf);*

Fonction, qui dépose dans l'argument de type jmp_buf l'information nécessaire à un goto non local. Voir longjmp.
Rend 0 si on sort de setjmp après y être entré, sinon
Rend la valeur de type int qui a été passée à longjmp si on sort de setjmp après un appel à longjmp.

void **longjmp***(jmp_buf, int);*

Fonction, qui restaure le process dans l'état où il était lors du setjmp qui a initialisé l'argument de type jmp_buf
Rend: rien (on ne sort jamais de longjmp). L'argument de type entier sera rendu par setjmp (voir setjmp). Si on passe 0 à l'argument de type entier, il est changé en 1.

<signal.h>

#define **SIG_DFL** ...

Macro, définissant une valeur indiquant un traitement par défaut un signal. Voir signal et raise.

#define **SIG_ERR** ...

Macro, définissant une valeur retournée par signal. Voir signal et raise.

#define **SIG_IGN** ...

Macro, définissant une valeur retournée par signal, indiquant l'absence de traitement du signal. Voir signal et raise.

#define **SIGABRT** ...

Macro, définissant le code du signal pour une terminaison anormale (ex., appel d'abort). Voir signal et raise.

#define **SIGFPE** ...

Macro, définissant le code du signal pour une opération arithmétique illégale (ex., /0). Voir signal et raise

#define **SIGILL** ...

Macro, définissant le code du signal pour une instruction illégale. Voir signal et raise.

#define **SIGINT** ...

Macro, définissant le code du signal pour une demande d'attention (par exemple, ^C). Voir signal et raise

#define **SIGSEGV** ...

Macro, définissant le code du signal pour un pointeur invalide. Voir signal et raise.

#define **SIGTERM** ...

Macro, définissant le code du signal qui demande la terminaison d'un process. Voir signal et raise.

*void (**__signal__*(int sig, void (*func)(int)))(int);*

Fonction, qui inscrit à l'exécution le fait que, en cas de signal sig, il faudra exécuter la fonction func.
Rend la dernière fonction inscrite pour le signal sig, ou SIG_ERR s'il n'y en a pas.

int **raise***(int);*

Fonction, qui envoie au processus courant le signal spécifié.
Rend 0 s'il n'y a pas d'erreur, autre chose s'il y a un problème.

<stdarg.h>

typedef ••• va_list;
Type dépendant de l'implémentation qui permet de parcourir les arguments variables.

`#define` **`va_start`**`(p, arg)` `•••`

Macro dépendant de l'implémentation qui initialise p du type va_list.

`#define` **`va_arg`**`(p, type)` `•••`

Macro dépendant de l'implémentation qui avance p (du type va_list) sur l'argument suivant.

`#define` **`va_end`**`(p)` `•••`

Macro dépendant de l'implémentation libère p du type va_list.

<stdio.h>

typedef **··· fpos_t**;

Type, permettant de repérer une position dans un fichier.

typedef **··· FILE**;

Type, défini par l'implémentation. Aucune fonction ne travaille sur ce type (toutes les entrées/sorties se font sur des FILE*)

#define **BUFSIZ ···**

Macro, définissant la taille des buffers utilisés par setbuf.

#define **EOF ···**

Macro, définissant un nombre négatif utilisé par plusieurs fonctions pour marquer la fin d'un fichier.

#define **FOPEN_MAX ···**

Macro, définissant le nombre de fichiers que l'on peut ouvrir simultanément.

#define **FILENAME_MAX ···**

Macro, définissant la taille maximum des noms de fichiers.

#define **L_tmpnam···**

Macro, définissant la taille des noms de fichiers temporaires. Voir tmpnam.

#define **TMP_MAX ···**

Macro, définissant le nombre maximum de fichiers temporaires. Voir tmpnam.

#define **SEEK_SET ···**

Macro, rend un entier utilisé par fseek.

#define **SEEK_CUR ···**

Macro, rend un entier utilisé par fseek.

#define **SEEK_END ···**

Macro, rend un entier utilisé par fseek.

*FILE**stdin**

Variable, représente l'entrée standard pour les fonctions d'entrées-sorties.

*FILE**stdout**

Variable, représente la sortie standard pour les fonctions d'entrées-sorties.

*FILE**stderr**

Variable, représente la sortie d'erreur standard pour les fonctions d'entrées-sorties.

```
int remove(const char * filename);
```

Fonction, détruit le fichier nommé.
Rend 0 si succès, non-zéro sinon.

```
int rename(const char *old, const char *new);
```

Fonction, renomme le fichier "old" en "new".
Rend 0 si succès, non-zéro sinon.

```
FILE *tmpfile(void);
```

Fonction, crée un fichier temporaire de nom indifférent, qui sera détruit à la fin du programme.
Rend le fichier.

```
char *tmpnam(char *);
```

Fonction, crée un nom de fichier valide et qui n'existe pas. tmpnam peut être appelée TMP_MAX fois.
Rend le nom, qui n'aura pas plus de L_tmpnam caractères.

```
int fclose(FILE *);
```

Fonction, ferme un fichier.
Rend 0 si succès, EOF sinon.

```
int fflush(FILE *stream);
```

Fonction, écrit sur le disque les opérations bufferisées sur stream.
Rend 0 si succès, EOF sinon.

```
FILE *fopen(const char *filename, const char *mode);
```

Fonction, ouvre un fichier filename dans le mode spécifié.
Rend le fichier si succès, NULL sinon.

```
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Fonction, ouvre un fichier filename dans le mode spécifié, l'associe à stream. Si le fichier stream était déjà ouvert, il est fermé d'abord.
Rend stream si succès, NULL sinon.

```
void setbuf(FILE *, char *);
```

Fonction, gère le mode de bufferisation d'un fichier.

```
int setvbuf(FILE *, char *, int, size_t);
```

Fonction, gère le mode de bufferisation d'un fichier.

```
int fprintf(FILE *stream, const char * format, ...);
```

Fonction, écrit les arguments variables dans le fichier, selon le format.
Rend le nombre de caractères écrits, ou un nombre négatif si erreur.

```
int fscanf(FILE *, const char *, ...);
```

Fonction, écrit dans les arguments variables des valeurs prises dans le fichier, selon le format.
Rend le nombre d'items lus (éventuellement 0), ou EOF s'il y a une erreur avant la lecture du premier.

```
int printf(const char *format, ...);
```

Fonction, écrit les arguments variables dans stdout, selon le format.
Rend le nombre de caractères écrits, ou un nombre négatif si **erreur**.

```
int scanf(const char *format, ...);
```

Fonction, écrit dans les arguments variables des valeurs prises depuis stdin, selon le format.
Rend le nombre d'items lus (éventuellement 0), ou EOF s'il y a une erreur avant la lecture du premier.

```
int sprintf(char *s, const char *format, ...);
```

Fonction, écrit les arguments variables dans s, selon le format.
Rend le nombre de caractères écrits, ou un nombre négatif si erreur.

```
int sscanf(const char *s, const char *format, ...);
```

Fonction, écrit dans les arguments variables des valeurs prises depuis s, selon le format.
Rend le nombre d'items lus (éventuellement 0), ou EOF s'il y a une erreur avant la lecture du premier.

```
int vfprintf(FILE *stream, const char *format, va_list arg);
```

Fonction, équivalente à fprintf où la liste d'arguments variables est remplacée par arg qui doit avoir été initialisé par va_start.
N'appelle pas va_end.
Rend le nombre de caractères écrits, ou un nombre négatif si erreur.

```
int vprintf(const char *, void *);
```

Fonction, équivalente à printf où la liste d'arguments variables est remplacée par arg qui doit avoir été initialisé par va_start.
N'appelle pas va_end.
Rend le nombre de caractères écrits, ou un nombre négatif si erreur.

```
int vsprintf(char *, const char *, void *);
```

Fonction, équivalente à sprintf où la liste d'arguments variables est remplacée par arg qui doit avoir été initialisé par va_start.
N'appelle pas va_end.
Rend le nombre de caractères écrits, ou un nombre négatif si erreur.

```
int fgetc(FILE *);
```

Fonction de lecture.
Rend un caractère (unsigned char) pris dans le fichier.

```
char *fgets(char *s, int n, FILE *stream);
```

Fonction, charge s (qui doit être allouée) avec une chaîne de caractères pris dans stream. Lit jusqu'au premier \n ou fin-de-fichier, mais pas plus de n caractères.
Un caractère nul est mis à la fin de la chaîne.
Rend s.

```
int fputc(int c, FILE *stream);
```

Fonction, écrit le carctère c dans stream.
Rend c si succès, EOF sinon.

```
int fputs(const char *s, FILE *stream);
```

Fonction, écrit une chaîne de caractères s dans stream. Le caractère nul n'est pas écrit.
Rend un nombre positif ou nul si succès, EOF sinon.

```
int getc(FILE *stream);
```

Fonction ou Macro, équivalente à fgetc.
Rend un caractère (unsigned char) pris dans le fichier.

```
int getchar(void);
```

Fonction de lecture.
Rend un caractère lu sur stdin, ou EOF si erreur.

```
char *gets(char *s);
```

Fonction, charge s (qui doit être allouée) avec une chaîne de caractères pris dans stream. Lit jusqu'au premier \n ou fin-de-fichier.
Ne pas utiliser (utiliser fgets) car il n'y a aucun moyen d'arrêter la lecture si s est saturé.
Un caractère nul est mis à la fin de la chaîne.
Rend s.

```
int putc(int c, FILE *stream);
```

Fonction ou Macro, équivalente à fputc.
Rend c si succès, EOF sinon.

```
int putchar(int);
```

Fonction, écrit un caractère sur stdout.
Rend le caractère, ou EOF si erreur.

```
int puts(const char *s);
```

Fonction, écrit s sur stdout.
Rend un nombre >=0 si succès, EOF sinon.

```
int ungetc(int c, FILE *stream);
```

Fonction, repousse c dans le buffer de stream. c sera relu par la première lecture de stream.
On ne peut pousser ainsi qu'un caractère par fichier.
Rend le caractère, ou EOF si erreur.

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Fonction, lit dans stream nmemb blocs chacun de taille size et les écrit dans la mémoire pointée par ptr.
Rend le nombre d'éléments effectivement lus.

```
size_t fwrite(const *ptr, size_t size, size_t nmemb, FILE *stream);
```

Fonction, écrit dans stream nmemb blocs chacun de taille size lus dans la mémoire pointée par ptr.
Rend le nombre d'éléments effectivement écrits.

```
int fgetpos(FILE *stream, fpos_t *pos);
```

Fonction, positionne stream à la position pos.
Rend 0 si succès, autre chose sinon.

```
int fseek(FILE *stream, long offset, int whence);
```

Fonction, positionne stream
- a la position offset si whence vaut SEEK_SET.
- a la position courante + offset si whence vaut SEEK_CUR
- a la fin du fichier + offset si whence vaut SEEK_END (pas forcément implémenté)

Rend 0 si succés, autre chose sinon.

```
int fsetpos(FILE *stream, const fpos_t * pos);
```

Fonction, positionne stream a la position pos.
Rend 0 si succés, autre chose sinon.

```
long ftell(FILE *);
```

Fonction de consultation.
Rend la position courante du fichier.

```
void rewind(FILE *);
```

Fonction, met la position courante du fichier au début.

```
void clearerr(FILE *);
```

Fonction, efface les statuts d'erreur éventuels sur le fichier.

```
int feof(FILE *);
```

Fonction de consultation.
Rend 0 si la fin du fichier n'est pas atteinte.

```
int ferror(FILE *);
```

Fonction de consultation.
Rend 0 s'il n'y a pas d'erreur positionnée pour le fichier.

```
void perror(const char *);
```

Fonction, charge la chaîne de caractères avec le texte du message correspondant à la dernière erreur pour le fichier.

<stdlib.h>

typedef struct { int quot, rem; } **div_t**;

Type utilisé dans la fonction div

typedef struct { long quot, rem; } **ldiv_t**;

Type utilisé dans la fonction ldiv

double **atof**(*const char *);*

Fonction, convertit la représentation imprimable d'un flottant dans ce flottant.
Rend le flottant convertit.

int **atoi**(*const char *);*

Fonction, convertit la représentation imprimable d'un entier dans cet entier.
Rend l'entier convertit.

long **atol**(*const char *);*

Fonction, convertit la représentation imprimable d'un entier long dans cet entier.
Rend l'entier long convertit.

double **strtod**(*const char *nptr, char **endptr);*

Fonction, convertit la représentation imprimable d'un double (chaîne pointée par nptr) en double.
endptr est positionné sur le premier caractère non interprété.
Rend le double.

long **strtol**(*const char *nptr, char **endptr, int base);*

Fonction, convertit la représentation imprimable d'un entier (chaîne pointée par nptr) en entier long.
endptr est positionné sur le premier caractère non interprété.
Rend l'entier long, lu dans la base spécifiée.

unsigned long **strtoul**(*const char *, char **, int);*

Fonction, convertit la représentation imprimable d'un entier long non signé (chaîne pointée par nptr) en entier long.
endptr est positionné sur le premier caractère non interprété.
Rend l'entier non signé, lu dans la base spécifiée.

#define **RAND_MAX ...**

Macro, utilisée par rand.

int **rand**(*void);*

Fonction qui calcule une séquence pseudo-aléatoire.
Rend un nombre pseudo-aléatoire, entre 0 et RAND_MAX.

```
void srand(unsigned);
```

Fonction qui initialise une séquence de nombres rendus par rand.

```
void *calloc(size_t nmemb, size_t size);
```

Fonction, alloue de la mémoire pour nmemb éléments chacun de taille size.
Initialise la mémoire à 0.
Rend un pointeur vers le début du bloc, ou NULL si erreur.

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

Fonction, libère la mémoire pointée par l'argument.
Attention, l'argument n'est pas modifié.

```
void *malloc(size_t);
```

Fonction, alloue de la mémoire le nombre d'octets spécifié.
Rend un pointeur vers le début du bloc, ou NULL si erreur.

```
void *realloc(void *ptr, size_t size);
```

Fonction, réalloue la mémoire pointée par ptr avec le nombre d'octets spécifié.
Ce nombre peut être plus grand, égal ou plus petit.
Rend un pointeur vers le début du bloc qui peut être l'ancien, ou NULL si erreur.

```
void abort(void);
```

Fonction, envoie un signal de terminaison anormale.

```
int atexit(void (*)(void));
```

Fonction, inscrit la fonction spécifiée comme devant être exécutée à la fin du programme.
Rend 0 si succès, autre chose sinon.

```
void exit(int);
```

Fonction, provoque la terminaison du programme. L'entier spécifié est passé au système. 0 signifie une terminaison normale.

```
char *getenv(const char *name);
```

Fonction, traduit name en une chaîne de caractères dépendant de l'environnement. Par exemple, le PATH sous UNIX peut être traduit par getenv.
Rend la traduction.

```
int system(const char *string );
```

Fonction, appelle l'hôte avec la chaîne de caractère spécifiée.
Si string est NULL,
Rend 0 si l'hôte ne sait pas quoi faire des appels de system.
Si string est ≠ NULL,
Rend une valeur dépendante de l'implémentation.

```
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, int
(*compare)(const void *, const void *));
```

Fonction, cherche dans un tableau (*base) de nmemb objets, chacun de taille size, un élément qui corresponde à key.
Pour décider de la correspondance, la fonction compare sera appelée et doit être définie à l'extérieur.

```
void qsort( const void *base, size_t nmemb, size_t size, int (*compare)(const void *, const
void *));
```

Fonction, trie un tableau (*base) de nmemb objets, chacun de taille size.
Pour décider de l'ordre, la fonction compare sera appelée et doit être définie à l'extérieur.
Elle doit rendre -1, 0 ou +1 selon l'ordre des deux arguments.

```
int abs(int);
```

Fonction valeur absolue.
Rend la valeur absolue de son argument entier.

```
div_t div(int numer, int denom);
```

Fonction division rationnelle sur entiers. Divise numer par denom, et charge la structure div_t avec le résultat.
Rend la structure représentant le rationnel.

```
long labs(long);
```

Fonction valeur absolue.
Rend la valeur absolue de son argument long.

```
ldiv_t ldiv(long, long);
```

Fonction division rationnelle sur longs. Divise numer par denom, et charge la structure div_t avec le résultat.
Rend la structure représentant le rationnel.

<string.h>

```
void *memcpy(void * s1, const void * s2, size_t n);
```

Fonction. Copie n octets de s2 dans s1. Risque de se passer mal s'il y a chevauchement entre s1 et s2.

```
void *memmove(void * s1, const void * s2, size_t n);
```

Fonction. Idem, mais se passe toujours bien même si recouvrement, au prix d'un peu de performance.

```
char *strcpy(char *s1, const char *s2);
```

Fonction. Copie s2 dans s1, jusqu'au caractère nul.
Rend s1.

```
char *strncpy(char *s1, const char *s2, size_t n);
```

Fonction. Copie s2 dans s1, jusqu'au caractère nul mais pas plus que n caractères.
Rend s1.

```
char *strcat(char *s1, const char *s2);
```

Fonction. Copie s2 à la fin de s1. Le caractère nul de s1 est écrasé.
s1 est modifiée, pas s2.
Rend s1.

```
char *strncat(char *s1, const char *s2, size_t n);
```

Fonction. Copie s2 à la fin de s1, mais pas plus de n caractères. s1 est modifiée, pas s2.
Un caractère nul est toujours mis a la fin, même si la limite n est atteinte.
Rend s1.

```
int memcmp(const void *s1, const void *s2, size_t n);
```

Fonction. Compare lexicalement les n premiers caractères de s1 et s2 (nuls ou non nuls)
Rend -1 si s1 < s2, 0 si s1 == s2, 1 si s1 > s2.

```
int strcmp(const char * s1, const char * s2);
```

Fonction. Compare lexicalement s1 et s2.
Rend -1 si s1 < s2, 0 si s1 == s2, 1 si s1 > s2.

```
int strncmp( const char * s1, const char * s2, size_t);
```

Fonction. Compare lexicalement s1 et s2, mais pas plus de n caractères.
Rend -1 si s1 < s2, 0 si s1 == s2, 1 si s1 > s2.

```
void *memchr(const void *s, int c, size_t n);
```

Fonction. Trouve la première occurence de c (converti en unsigned char) parmi les n premiers caractères de s.

Rend le pointeur sur le caractère trouvé, ou NULL si insuccès.

```
char *strchr(const char *s, int c);
```

Fonction. Trouve la première occurence de c (converti en unsigned char) dans s.
Remarque: si c = 0, le caractère nul de fin de chaîne sera trouvé.
Rend le pointeur sur le caractère trouvé, ou NULL si insuccès.

```
size_t strcspn(const char *s1, const char *s2);
```

Fonction. Rend la longueur du segment de chaîne commençant en s1, composé uniquement de caractères qui ne sont pas dans la chaîne pointée par s2.

```
char *strpbrk(const char *s1, const char *s2);
```

Fonction.Rend un pointeur sur la première occurence dans s1 d'un des caractères qui sont dans s2, NULL si insuccès.

```
char *strrchr(const char *s, int c);
```

Fonction. Trouve la dernière occurence de c (converti en unsigned char) dans s.
Remarque: si c = 0, le caractère nul de fin de chaîne sera trouvé.
Rend le pointeur sur le caractère trouvé, ou NULL si insuccès.

```
size_t strspn(const char *s1, const char *s2);
```

Fonction.
Rend la longueur du segment de chaîne commençant en s1, composé uniquement de caractères qui sont dans la chaîne pointée par s2.

```
char *strstr(const char *s1, const char *s2);
```

Fonction.
Rend un pointeur sur la première séquence de caractères dans s1 qui est égale à s2.
Rend NULL si insuccès.

```
char *strtok(char *s1, const char *s2);
```

Fonction. Tronçonne s1 en "mots" délimités par un des caractères qui composent s2.
Le premier appel passe s1, les suivants passent NULL en premier argument.
s2 peut être différent à chaque appel.
Rend (à chaque appel) un pointeur sur le "mot" suivant, ou NULL si insuccès.

```
void *memset(void * s, int c, size_t n);
```

Fonction. Copie c dans les n premiers caractères pointés par s.

```
char *strerror(int errnum);
```

Fonction.
Rend une chaîne de caractères représentant le message associé à l'erreur errnum.

```
size_t strlen(const char *s);
```

Fonction.
Rend la longueur de s (le nombre de caractères avant le caractère nul).

<time.h>

typedef **...** **clock_t**;

Type, capable de représenter le résultat de clock.

typedef **...** **time_t**;

Type, capable de représenter le temps.

struct **tm** *{*

```
int tm_sec;
int tm_min;
int tm_hour;
int tm_mday;
int tm_mon;
int tm_year;
int tm_wday;  /* 0=dimanche, 1=lundi, etc. */
int tm_yday;  /* nombre de jours depuis le 1er Janvier */
int tm_isdst; /* booléen, 1 si heure d'été */
};
```

Type, représentant la date, l'heure et diverses informations. Voir mktime, asctime et strftime.

clock_t **clock***(void);*

Fonction.
Rend le nombre de 'ticks' depuis le lancement du programme. La durée d'un 'tick' est dépendante de l'implémentation.

double **difftime***(time_t, time_t);*

Fonction.
Rend le nombre de secondes entre les deux temps.

time_t **mktime***(struct tm *);*

Fonction, convertit la date exprimée par une structure tm dans le type time_t.
Ignore les champs tm_wday et tm_yday.
Rend l'objet du type time_t.

time_t **time***(time_t * timer);*

Fonction.
Rend la date et l'heure courante. Si timer n'est pas nul, la même valeur est affectée à *timer.

*char **asctime***(const struct tm *);*

Fonction.
Rend une chaîne de caractères représentant la date et l'heure, de la forme: « Sun Sep 16 01:03:52 1995\n\0 »

```
char *ctime(const time_t* timer);
```

Fonction, équivalente à asctime (localtime (timer)). Voir asctime et localtime.
Rend une chaîne de caractères.

```
struct tm *gmtime(const time_t *);
```

Fonction, charge une structure tm de façon cohérente, exprimé en temps universel, à partir d'un time_t.
Rend cette structure.

```
struct tm*localtime(const time_t *);
```

Fonction, charge une structure tmde façon cohérente, exprimé en temps local, à partir d'un time_t.
Rend cette structure.

```
size_t strftime(char *s, size_t maxsize, const char * format, const struct tm * timeptr);
```

Fonction, écrit dans s une chaîne de caractères maîtrisée par format représentant le temps pointé par timeptr.
Le format comprend une liste de switchs spécifiques.
Rend le nombre de caractères écrits.

Opérateurs

+	addition
-	soustraction
+	plus unaire
-	moins unaire
*	multiplication
/	division
%	modulo
>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
==	égal
!=	différent
? et :	test
~	non bit à bit
!	non logique
<<	décalage gauche
>>	décalage droit
&	ET bit à bit
&&	ET logique
|	OU bit à bit
||	OU logique
^	OU exclusif bit à bit
=	affectation
+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=	opération et affectation
,	évalue termes gauche et droit, rend le droit
sizeof	rend la taille de l'argument en octets.
(*type*) *-cast-*	rend un objet visible sous un type donné.
()	associé des arguments à une fonction
[]	indexe un tableau
.	sélectionne un champ.
->	sélectionne le champ d'un objet pointé.
*	déréférence.
&	rend l'adresse de la lvalue argument

Spécifieurs

Chaque spécifieur peut être augmenté d'une information de cadrage, par exemple: %5d ou %-5d cadre l'entier à droite ou à gauche, sur 5 caractères.

Famille …printf

specifieur	**argument**	**représentation**
%	aucun	écrit un %
%c	int	caractère litéral
%d ou %i	int	séquence de chiffres (base 10)
%hd ou %hi	short	séquence de chiffres (base 10)
%ld ou %li	long	séquence de chiffres (base 10)
%e	double	d.ddde±dd
%Le	long double	d.ddde±dd
%e	double	d.dddE±dd
%LE	long double	d.dddE±dd
%f	double	d.ddd
%Lf	long double	d.ddd
%g	double	d.ddd ou d.ddde±dd selon valeur
%Lg	long double	d.ddd ou d.ddde±dd selon valeur
%G	double	d.ddd ou d.dddE±dd selon valeur
%LG	long double	d.ddd ou d.dddE±dd selon valeur
%o	int	représentation octale après conversion en unsigned
%ho	short	représentation octale après conversion en unsigned
%lo	long	représentation octale après conversion en unsigned
%p	void*	<défini par l'implémentation>
%s	char*	la chaîne de caractères
%u	int	représentation décimale après conversion en unsigned
%hu	short	représentation décimale après conversion en unsigned
%lu	long	représentation décimale après conversion en unsigned
%x	int	représentation hexa après conversion en unsigned
%hx	short	représentation hexa après conversion en unsigned
%lx	long	représentation hexa après conversion en unsigned
%X	int	représentation hexa après conversion en unsigned
%hX	short	représentation hexa après conversion en unsigned
%lX	long	représentation hexa après conversion en unsigned
%n	&int	
%hn	&short	
%ln	&long	

Famille …scanf

specifieur	argument
%c	*char(s)
%d	*int base 10
%hd	*short
%ld	*long
%e,%E,%f,%g,%G	*float
%le,%lE,%lf,%lg,%lG	*double
%Le,%LE,Lf,%Lg,%LG	*long double
%i	*int, base 16,8 ou 10 selon écriture : 0xABC, O123 ou 789
%hi	*short, idem
%li	*long, idem
%o	*unsigned base 8
%ho	*unsigned short base 8
%lo	*unsigned long base 8
%p	*(void*) , dépendant de l'implémentation
%s	*char
%u	*unsigned, base 10
%hu	*unsigned short, base 10
%lu	*unsigned long, base 10
%x, %X	*unsigned, base 16
%hx,%hX	*unsigned short, base 16
%lx,%lX	*unsigned long, base 16
%n	&int
%hn	&short
%ln	&long

strftime

specifieur	représentation
%	écrit un %
%a	jour de la semaine abrégé
%A	jour de la semaine complet
%b	mois abrégé
%B	mois complet
%c	date/heure
%d	jour du mois 00-31
%H	heure 00-23
%I	heure 01-12
%j	jour de l'année 000-366
%m	mois 01-12
%M	minute 00-59
%p	AM/PM
%S	secondes 00-61 (la dernière minute de l'année peut avoir 61 secondes !)
%U	numéro de la semaine avec premier dimanche premier jour semaine 1 : 00-53
%w	jour de la semaine 0-6
%W	numéro de la semaine avec premier lundi premier jour semaine 1 : 00-53
%x	dépendant des locales
%X	dépendant des locales
%y	année 00-99
%Y	année complète
%Z	nom du fuseau horaire

Index Alphabétique