[LC055]

La bibliothèque standard ANSI

• valeur retournée
◊ idem en cas d'erreur

(voir aussi (KR2) *pages 245 à 264;* (TCR)*;* (HS)*; ...)*

Sont données ci-dessous les déclarations en style moderne des fonctions les plus courantes de la bibliothèque ANSI. Fonctions, macros et types sont en fait déclarés (style?) dans les fichiers d'en-tête standard :

<assert.h>	<float.h>	<math.h>	<stdarg.h>	<stdlib.h>
<ctype.h>	<limits.h>	<setjmp.h>	<stddef.h>	<string.h>
<errno.h>	<locale.h>	<signal.h>	<stdio.h>	<time.h>

(I)Quelques fonctions de <stdio.h> sur les entrées/sorties:

Un flot (stream = flot ou flux) est une source ou une destination de données qui peut être associée à un fichier ou à un périphérique. L'ouverture d'un fichier lui associe un flot et retourne un pointeur sur un objet de type FILE que l'on peut identifier à ce flot. Les trois flots **stdin, stdout, stderr** sont prédéfinis.

opérations sur fichiers

```
FILE * fopen(char * chemin, char * mode)   ouverture • pointeur sur flot ouvert ◊ NULL
FILE * freopen(char * chemin, char * mode, FILE * fl)   réouverture (flot standard)
/* mode = "r" ouverture ,"w" création en ,"a" ajout en ,"r+" ouverture ,"w+" création ,"a+" ajout en   (texte)
      = "rb" en lect ,"wb" écriture ,"ab" écriture ,"r+b" écr-lect ,"w+b" écr-lect ,"a+b" écr-lect   (binaire)
```

En mode texte le caractère d'interligne '\n'= <LF> peut être traduit de différentes façons suivant l'environnement (<LF> sous UNIX; <CR><LF> sous TurboC). En mode binaire, au contraire, il n'est effectué aucun traitement sur les octets (caractères ou autres) qui "s'écoulent" dans le flot : Le caractère '\n' restera <LF>. */

```
int    fclose(FILE * fich)     fermeture                          • 0              ◊ EOF
FILE * tmpfile(void)           création fichier temporaire        • pointeur sur flot  ◊ NULL
int    remove(char * chemin)   suppression                        • 0              ◊ ≠0
int    rename(char * oldname, char * newname)   renomme (ou move) • 0              ◊ ≠0
   /* autres fonctions :  tmpnam   setwbuf   setbuf   open   close   ... */
```

entrées/sorties sur flots

```
int    fprintf(FILE * fl, char * format [,val,...])  sortie formatée   • nb car ◊ nég
int    fscanf(FILE * fl, char * format [,adr,...])   entrée formatée   • nb var lues ◊EOF
int    fputc(int ca, FILE * fl)   } sortie d'un caractère(octet)   • caractère ca écrit ◊ EOF
int    putc(int ca, FILE * fl)    }
int    fgetc(FILE * fl)           } entrée d'un caractère(octet)   • caractère lu    ◊ EOF
int    getc(FILE * fl)            }
int    fputs(char * ch,FILE * fl)          sortie chaîne            • positif ou nul ◊ EOF
char * fgets(char * ch,int n,FILE * fl)    entrée chaîne (au plus n-1 car)  • ch ◊ NULL
   /* le type size_t est le type (entier non_signé) retourné par l'opérateur sizeof */
size_t fwrite(void * ptr,size_t size,size_t nb,FILE * fl)   sortie bloc (tableau) (mode binaire)   • nbre élts écrits
size_t fread(void * ptr,size_t size,size_t nb,FILE * fl)    entrée bloc (tableau) (mode binaire)   • nbre élts lus
int    ungetc(int car, FILE * fl)   remise un caract sur flot d'entrée   • car ◊ EOF
   /* autres fonctions :  fflush   vfprintf   vfscanf   write   read   ... */
```

entrées/sorties standard

```
int    printf(char * format [,val,...])   ⇔ fprintf(stdout,...)   • nb caract. écrits ◊ nég
int    scanf(char * format [,adr,...])    ⇔ fscanf(stdin,...)     • nb variables lues ◊ EOF
int    putchar(int car)                   ⇔ putc(c,stdout)        • caractère car écrit ◊ EOF
int    getchar()                          ⇔ getc(stdin)           • caractère lu ◊ EOF
int    puts(char * ch)    sortie ligne(s) (avec rempl. \0 par \n )   • positif ou nul ◊ EOF
char * gets(char * ch)    entrée ligne (avec remplacemt \n par \0 )  • adresse ch ◊ NULL
   /* autres fonctions :  vprintf   vscanf   ... */
```

pseudo entrées/sorties sur chaînes

```
int    sprintf(char * s, char * format [,val,...])
int    sscanf(char * s, char * format [,adr,...])
```
/* autres fonctions : vsprintf vsscanf ... */

positionnement dans fichiers

```
int    fseek(FILE * fl, long offset, int origin)
```
positionne pointeur de fichier • 0 (*zéro*) ◊ ≠0

```
long   ftell(FILE * fl)
```
retourne : • position courante pointeur fichier ◊ -1

/* autres fonctions : rewind fgetpos fsetpos ... */

gestion des erreurs

```
void   clearerr(FILE * fl)
int    feof(FILE * fl)
int    ferror(FILE * fl)
void   perror(char * s)
```
remet à zéro les indicateurs eof et erreur du flot
retourne : • VRAI (≠0) si fin de fichier pour ce flot
retourne : • VRAI (≠0) si erreur pour ce flot
affiche message dernière erreur (n° dans errno)

/* autres fonctions : strerror ... */

(II) Quelques fonctions de <ctype.h> pour tester la classe de caractères ..:

/* retournent VRAI (valeur non nulle) si le caractère c est de la classe testée */

`isalnum(int c)`	*est alphanumérique*	`isalpha(int c)`	*est alphabétique*
`iscntrl(int c)`	*est car. de contrôle*	`isspace(int c)`	*est un blanc*
`islower(int c)`	*est une minuscule*	`isupper(int c)`	*est une majuscule*
`isdigit(int c)`	*est un chiffre*	`isxdigit(int c)`	*ext un chiffre hexadécimal*

... ou pour convertir :

`int tolower(int c)` *en minuscule* `int toupper(int c)` *en majuscule*

(III) Le traitement des chaines de caractères : <string.h>

/* le type **size_t** est le type (entier non_signé) retourné par l'opérateur sizeof */

```
char * strcpy(char * dest, char * src)
char * strncpy(char * dest, char * src, size_t nbmax)
void * memcpy(void * dest, void * src, size_t nb)
void * memmove(void * dest, void * src, size_t nb)
char * strcat(char * dest, char * suite)
char * strncat(char * dest, char * suite, size_t nbmax)
```
copie chaîne (y compris octet nul) • **dest**
idem (nbmax caract. max)
idem (nb octets)
idem même si chevauchement
concaténation chaînes • **dest**
id (nbmax caract. max)

```
int    strcmp(char * ch1, char * ch2)
```
comparaison chaînes • { + si **ch1>ch2** ; 0 si **ch1=ch2** ; - si **ch1<ch2** }

```
int    strncmp(char * ch1, char * ch2, size_t nbmax)
int    memcmp(void * ch1, void * ch2, size_t nb)
size_t strlen(char * ch)
char * strchr(char * ch, int c)
void * memchr(void * ch, int c, size_t nb)
char * strstr(char * ch, char * ssch)
void * memset(void * ch, int c, size_t nb)
```
idem (nbmax caract. max)
idem (nb octets)
retourne : • longueur de la chaîne **ch**
retourne : • pointeur sur 1è occurence du caractère **c** dans la chaîne **ch** ◊ NULL
idem (rech. dans nb premiers octets de ch)
retourne : • pointeur sur 1è occurence de la sous-chaîne **ssch** dans la chaîne **ch** ◊ NULL
remplace nb octets de ch par car. c • **ch**

/* autres fonctions : strrchr strspn strcspn strpbrk strtok ... */

(IV) Les fonctions mathématiques : <math.h>

/* x et y sont de type 'double' et toutes ces fonctions retournent un résultat de type 'double' */

```
sin(x)          cos(x)          tan(x)      /* x en radians */
asin(x)         acos(x)         atan(x)
```

`atan2(y,x)` *qui retourne :* la valeur de $\arctan\left(\frac{x}{y}\right)$ dans $[-\pi,\pi]$ tels que sin et cos soient respectivement de même signes que **x** et **y**.

```
sinh(x)         cosh(x)         tanh(x)
exp(x)          log(x)          log10(x)
```

`pow(x,y)` *qui retourne :* x^y

`sqrt(x)` *qui retourne :* $\sqrt{x}$

`fmod(x,y)` *qui retourne :* x modulo y */* x et y étant réels flottant de type 'double' */*

`fabs(x)` *qui retourne :* valeur absolue du réel **x** 'double' */* voir aussi abs et labs ci-dessous */*

/* autres fonctions : ceil floor ldexp frexp modf ... */

(V) Les limites définies par l'implémentation : <limits.h> et <float.h>

→ Constantes correspondant aux valeurs min et max et autres caractéristiques des types entiers (limits.h) ou réels (float.h). *Exemples* : INT_MAX et INT_MIN ; CHAR_BIT ; DBL_MAX et DBL_MIN ; FLT_RADIX ; FLT_DIG ...

(VI) Les fonctions utilitaires : <stdlib.h>

```
int     abs(int n)          retourne :   • valeur absolue de l' entier (int) n
long    labs(long n)        retourne :   • valeur absolue de l' entier long n
```

/* autres fonctions : div ldiv ... */

```
double atof(char * s)     conv ascii to float double
int    atoi(char * s)     conv ascii to int
long   atol(char * s)     conv ascii to long
```

/* autres fonctions : strtod strto strtoul ... */

```
int    rand(void)         retourne :   • nbre aléatoire entre 0 et RAND_MAX /* qui vaut 32767 */
void   srand(unsigned int s)   idem à partir de la graine s
```

/* le type **size_t** est le type (entier non_signé) retourné par l'opérateur sizeof */

```
void * malloc(size_t size)   allocation mémoire • adresse (pointeur sur) zone allouée ◊ NULL
void * calloc(size_t nb, size_t size)   idem
void * realloc(void * p, size_t size)   réallocation (chgmt taille) de la mémoire
                  anciennement allouée au pointeur p   • nouvelle adresse zone allouée ◊ NULL
void   free(void * pt)      libération mémoire allouée
void   abort(void)          arrêt anormal du programme
void   exit(int status)     sortie = arrêt normal du programme
int    system(char * ch_com)   passe une chaîne de commande(s) au système
```

/* autres fonctions : getenv atexit ... */

```
void * bsearch  /* recherche binaire dans un tableau déjà trié • pointeur sur élt si trouvé ◊ NULL */
(void * clé, void * tbl ,size_t n,size_t tai ,int (* cmp)(void * d1,void * d2))
void   qsort    /* tri suivant algorithme quicksort */
          (void * tbl ,size_t n,size_t tai ,int (* cmp)(void * d1,void * d2))
```

clé — tableau — nb élt — taille élt — fonct. de comparaison

(VII) Les messages pour mise au point : <assert.h>

`void assert(int expression)` */* si expression non nulle, affichage du message :*

`Assertion failed : expression, file nomfichier, line n°ligne` *puis arrêt du programme.* En insérant `#define NDEBUG` *avant l'inclusion de <assert.h>, la macro assert n'est pas prise en compte */*