Livre_ProgrammerC++.book Page 757 Mardi, 19. septembre 2017 9:18 09

Annexe G

Les principales fonctions de la bibliothèque C standard

La norme ANSI du langage C fournissait à la fois la description du langage C et le contenu d'une bibliothèque standard. Plus précisément, cette bibliothèque est subdivisée en plusieurs sous-bibliothèques ; à chaque sous-bibliothèque est associé un fichier « en-tête » comportant essentiellement :

- les en-têtes des fonctions correspondantes ;
- les définitions des macros correspondantes ;
- les définitions de certains symboles utiles au bon fonctionnement des fonctions ou macros de la sous-bibliothèque.

En théorie, en C++, toutes ces fonctions restent accessibles, mais certaines ne sont plus utilisées en C++. La présente annexe décrit les **principales fonctions** pouvant présenter un intérêt en C++[1]. Chaque paragraphe correspond à une sous-bibliothèque et précise quel est le nom du fichier en-tête correspondant.

1. Vous trouverez une description complète de la bibliothèque standard du C dans l'ouvrage *Langage C*, publié aux éditions Eyrolles.

Livre_ProgrammerC++.book Page 758 Mercredi, 20. septembre 2017 9:46 09

Remarque

Les fonctions décrites ici sont classées par fichier en-tête, et non par ordre alphabétique. Néanmoins, si vous cherchez la description d'une fonction précise, il vous suffit de vous reporter à l'index situé en fin d'ouvrage.

En C

Les noms des fichiers en-tête étaient légèrement différents de ceux de C++. Par exemple, on trouvait *stdio.h* au lieu de *cstdio* (le préfixe *c* traduisant, en quelque sorte, l'héritage du langage C.

1 Entrées-sorties (*cstdio*)

N.B. Le symbole *FILE* est défini par *typedef* comme un synonyme d'un type structure dont les champs contiennent les informations nécessaires à la gestion d'un fichier (nom, mode d'écriture ou de lecture, tampon pour stocker les données intermédiaires...). Les symboles *stdin* et *stdout* sont des noms prédéfinis de telles structures associées à l'entrée et à la sortie standard.

1.1 Gestion des fichiers

FOPEN ***FILE** * **fopen** (***const char** * **nomfichier, const char** * **mode**)

Ouvre le fichier dont le nom est fourni, sous forme d'une chaîne, à l'adresse indiquée par *nomfichier*. Fournit, en retour, un « flux » (pointeur sur une structure de type prédéfini *FILE*), ou un pointeur nul si l'ouverture a échoué. Les valeurs possibles de *mode* sont les suivantes :

- **r** : *lecture* seulement ; le fichier doit exister.
- **w** : *écriture* seulement. Si le fichier n'existe pas, il est créé. S'il existe, son (ancien) contenu est perdu.
- **a** : *écriture en fin de fichier* (*append*). Si le fichier existe déjà, il sera étendu. S'il n'existe pas, il sera créé – on se ramène alors au mode *w*.
- **r+** : *mise à jour* (lecture et écriture). Le fichier doit exister. Notez qu'alors il n'est pas possible de réaliser une lecture à la suite d'une écriture ou une écriture à la suite d'une lecture, sans positionner le pointeur de fichier par *fseek*. Il est toutefois possible d'enchaîner plusieurs lectures ou écritures consécutives (de façon séquentielle).
- **w+** : *création pour mise à jour*. Si le fichier existe, son (ancien) contenu sera détruit. S'il n'existe pas, il sera créé. Notez que l'on obtiendrait un

Livre_ProgrammerC++.book Page 759 Mardi, 19. septembre 2017 9:18 09

mode comparable à *w+* en ouvrant un fichier vide (mais existant) en mode *r+*.

- **a+** : *extension et mise à jour*. Si le fichier n'existe pas, il sera créé. S'il existe, le pointeur sera positionné en fin de fichier.
- **t** ou **b** : lorsque l'implémentation distingue les fichiers de texte des autres, il est possible d'ajouter l'une de ces deux lettres à chacun des 6 modes précédents. La lettre *t* précise que l'on a affaire à un fichier de texte ; la lettre *b* précise que l'on a affaire à un fichier binaire. (On dit aussi que *t* correspond au mode « translaté », pour spécifier que certaines substitutions auront lieu).

FCLOSE ***int* fclose (*FILE* * flux)**

Vide éventuellement le tampon associé au flux concerné, désalloue l'espace mémoire attribué à ce tampon et ferme le fichier correspondant. Fournit la valeur *EOF* en cas d'erreur et la valeur *0* dans le cas contraire.

1.2 Écriture formatée

Toutes ces fonctions utilisent une chaîne de caractères nommée *format*, composée à la fois de caractères quelconques et de codes de format dont la signification est décrite en détail à la fin du présent paragraphe.

FPRINTF ***int* fprintf (*FILE* * flux, *const char* * format, ...)**

Convertit les valeurs éventuellement mentionnées dans la liste d'arguments (...) en fonction du *format* spécifié, puis écrit le résultat dans le *flux* indiqué. Fournit le nombre de caractères effectivement écrits ou une valeur négative en cas d'erreur.

PRINTF ***int* printf (*const char* * format, ...)**

Convertit les valeurs éventuellement mentionnées dans la liste d'arguments (...) en fonction du *format* spécifié, puis écrit le résultat sur la sortie standard (*stdout*). Fournit le nombre de caractères effectivement écrits ou une valeur négative en cas d'erreur.

Notez que :

printf (format, ...) ;

est équivalent à :

fprintf (stdout, format, ...) ;

SPRINTF ***int* sprintf (*char* * ch, *const char* * format, ...)**

Convertit les valeurs éventuellement mentionnées dans la liste d'arguments (...) en fonction du *format* spécifié et place le résultat dans la chaîne

Livre_ProgrammerC++.book Page 760 Mardi, 19. septembre 2017 9:18 09

d'adresse *ch*, en le complétant par un caractère *\0*. Fournit le nombre de caractères effectivement écrits (sans tenir compte du *\0*) ou une valeur négative en cas d'erreur.

1.3 Les codes de format utilisables avec ces trois fonctions

Chaque code de format a la structure suivante :

% [drapeaux] [largeur] [.précision] [h|l|L] conversion

dans laquelle les crochets *[* et *]* signifient que ce qu'ils renferment est facultatif. Les différentes « indications » se définissent comme suit :

- **drapeaux** :

  - : justification à gauche ;

  + : signe toujours présent ;

  ^ : impression d'un espace au lieu du signe + ;

  # : forme alternée ; elle n'affecte que les types **o**, **x**, **X**, **e**, **E**, **f**, **g** et **G** comme suit :

  - **o** : fait précéder de *0* toute valeur non nulle ;
  - **x** ou **X** : fait précéder de 0x ou 0X la valeur affichée ;
  - **e**, **E** ou **f** : le point décimal apparaît toujours ;
  - **g** ou **G** : même effet que pour **e** ou **E**, mais de plus les zéros de droite ne seront pas supprimés.

- **largeur** (*n* désigne une constante entière positive écrite en notation décimale) :

  n : au minimum, *n* caractères seront affichés, éventuellement complétés par des blancs à gauche ;

  0n : au minimum, *n* caractères seront affichés, éventuellement complétés par des zéros à gauche ;

  ***** : la largeur effective est fournie dans la liste d'expressions.

- **précision** (*n* désigne une constante entière positive écrite en notation décimale) :

  .n : la signification dépend du caractère de conversion, de la manière suivante :

  - **d**, **i**, **o**, **u**, **x** ou **X** : au moins n chiffres seront imprimés. Si le nombre comporte moins de n chiffres, l'affichage sera complété à gauche par des zéros. Notez que cela n'est pas contradictoire avec l'indication de largeur, si celle-ci est supérieure à *n*. En effet, dans ce cas, le nombre pourra être précédé à la fois d'espaces et de zéros ;
  - **e**, **E** ou **f** : on obtiendra n chiffres après le point décimal, avec arrondi du dernier ;
  - **g** ou **G** : on obtiendra au maximum *n* chiffres significatifs ;
  - **c** : sans effet ;

Livre_ProgrammerC++.book Page 761 Mardi, 19. septembre 2017 9:18 09

- **s** : au maximum *n* caractères seront affichés. Notez que cela n'est pas contradictoire avec l'indication de largeur.

.0 : la signification dépend du caractère de conversion, comme suit :

- **d**, **i**, **o**, **u**, **x** ou **X** : choix de la valeur par défaut de la précision (voir ci-dessous) ;
- **e**, **E** ou **f** : pas d'affichage du point décimal ;

***** : la valeur effective de *n* est fournie dans la « liste d'expressions ».

rien : choix de la valeur par défaut, à savoir :

- *1* pour *d*, *i*, *o*, *u*, *x* ou *X* ;
- *6* pour *e*, *E* ou *f* ;
- tous les chiffres significatifs pour *g* ou *G*
- tous les caractères pour *s* ;
- sans effet pour *c*.

- **h|l|L** :

  h : l'expression correspondante est d'un type *short int* (signé ou non). En fait, il faut voir que, compte tenu des conversions implicites, *printf* ne peut jamais recevoir de valeur d'un tel type. Tout au plus peut-elle recevoir un entier dont on (le programmeur) sait qu'il résulte de la conversion d'un *short*. Dans certaines implémentations, l'emploi du modificateur *h* conduit alors à afficher la valeur correspondante suivant un gabarit différent de celui réservé à un *int* (c'est souvent le cas pour le nombre de caractères hexadécimaux). Ce code ne peut, de toute façon, avoir une éventuelle signification que pour les caractères de conversion : *d*, *i*, *o*, *u*, *x* ou *X*.

  l : Ce code précise que l'expression correspondante est de type *long int*. Il n'a de signification que pour les caractères de conversion : *d*, *i*, *o*, *u*, *x* ou *X*.

  L : Ce code précise que l'expression correspondante est de type *long double*. Il n'a de signification que pour les caractères de conversion : *e*, *E*, *f*, *g* ou *G*.

- **conversion** : il s'agit d'un caractère qui précise à la fois le type de l'expression (nous l'avons noté ici en italique) et la façon de présenter sa valeur. Les types numériques indiqués correspondent au cas où aucun modificateur n'est utilisé (voir ci-dessus) :
  - **d** : *signed int*, affiché en décimal ;
  - **o** : *unsigned int*, affiché en octal ;
  - **u** : *unsigned int*, affiché en décimal ;
  - **x** : *unsigned int*, affiché en hexadécimal (lettres minuscules) ;
  - **X** : *signed int*, affiché en hexadécimal (lettres majuscules) ;
  - **f** : *double*, affiché en notation décimale ;
  - **e** : *double*, affiché en notation exponentielle (avec la lettre *e*) ;
  - **E** : *double*, affiché en notation exponentielle (avec la lettre *E*) ;

Livre_ProgrammerC++.book Page 762 Mardi, 19. septembre 2017 9:18 09

- **g** : *double*, affiché suivant le code *f* ou *e* (ce dernier étant utilisé lorsque l'exposant obtenu est soit supérieur à la précision désirée, soit inférieur à *-4*) ;
- **G** : *double*, affiché suivant le code *f* ou *E* (ce dernier étant utilisé lorsque l'exposant obtenu est soit supérieur à la précision désirée, soit inférieur à *-4*) ;
- **c** : *char* ;
- **s** : pointeur sur une « chaîne » ;
- *%* : affiche le caractère %, sans faire appel à aucune expression de la liste ;
- **n** : place, à l'adresse désignée par l'expression de la liste (du type pointeur sur un entier), le nombre de caractères écrits jusqu'ici ;
- **p** : pointeur, affiché sous une forme dépendant de l'implémentation.

1.4 Lecture formatée

Ces fonctions utilisent une chaîne de caractères nommée *format*, composée à la fois de caractères quelconques et de codes de format dont la signification est décrite en détail à la fin du présent paragraphe. On y trouvera également les règles générales auxquelles obéissent ces fonctions (arrêt du traitement d'un code de format, arrêt prématuré de la fonction).

FSCANF ***int* fscanf (*FILE* * flux, *const char* * format, ...)**

Lit des caractères sur le flux spécifié, les convertit en tenant compte du *format* indiqué et affecte les valeurs obtenues aux différentes variables de la liste d'arguments (...). Fournit le nombre de valeurs lues convenablement ou la valeur *EOF* si une erreur s'est produite ou si une fin de fichier a été rencontrée avant qu'une seule valeur ait pu être lue.

SCANF ***int* scanf (*const char* * format, ...)**

Lit des caractères sur l'entrée standard (*stdin*), les convertit en tenant compte du *format* indiqué et affecte les valeurs obtenues aux différentes variables de la liste d'arguments (...). Fournit le nombre de valeurs lues convenablement ou la valeur *EOF* si une erreur s'est produite ou si une fin de fichier a été rencontrée avant qu'une seule valeur ait pu être lue.

Notez que :

scanf (format, ...)

est équivalent à :

fscanf (stdin, format, ...)

SSCANF ***int* sscanf (*char* * ch, *const char* * format, ...)**

Lit des caractères dans la chaîne d'adresse *ch*, les convertit en tenant compte du *format* indiqué et affecte les valeurs obtenues aux différentes

Livre_ProgrammerC++.book Page 763 Mardi, 19. septembre 2017 9:18 09

variables de la liste d'arguments (...). Fournit le nombre de valeurs lues convenablement.

1.5 Règles communes à ces fonctions

a) Il existe six caractères dits « séparateurs », à savoir : l'espace, la tabulation horizontale (*\t*), la fin de ligne (*\n*), le retour chariot (*\r*), la tabulation verticale (*\v*) et le changement de page (*\f*). En pratique, on se limite généralement à l'espace et à la fin de ligne.

b) L'information est recherchée dans un tampon, image d'une ligne. Il y a donc une certaine désynchronisation entre ce que l'on frappe au clavier (lorsque l'unité standard est connectée à ce périphérique) et ce que lit *la fonction*. Lorsqu'il n'y a plus d'information disponible dans le tampon, il y a déclenchement de la lecture d'une nouvelle ligne. Pour décrire l'exploration de ce tampon, il est plus simple de faire intervenir un indicateur de position que nous nommerons pointeur.

c) La rencontre dans le format d'un caractère séparateur provoque l'avancement du pointeur jusqu'à la rencontre d'un caractère qui ne soit pas un séparateur.

d) La rencontre dans le format d'un caractère différent d'un séparateur (et de *%*) provoque la prise en compte du caractère courant (celui désigné par le pointeur). Si celui-ci correspond au caractère du format, *la fonction* poursuit son exploration du format. Dans le cas contraire, il y a arrêt prématuré de *la fonction*.

e) Lors du traitement d'un code de format, l'exploration s'arrête :

- à la rencontre d'un caractère invalide par rapport à l'usage qu'on doit en faire (point décimal pour un entier, caractère différent d'un chiffre ou d'un signe pour du numérique...). Si *la fonction* n'est pas en mesure de fabriquer une valeur, il y a arrêt prématuré de l'ensemble de la lecture ;
- à la rencontre d'un séparateur ;
- lorsque la longueur (si elle a été spécifiée) a été atteinte.

1.6 Les codes de format utilisés par ces fonctions

Chaque code de format a la structure suivante :

% [*] [largeur] [h|l|L] conversion

dans laquelle les crochets *[* et *]* signifient que ce qu'ils renferment est facultatif. Les différentes « indications » se définissent comme suit :

- ***** : la valeur lue n'est pas prise en compte ; elle n'est donc affectée à aucun élément de la liste ;

Livre_ProgrammerC++.book Page 764 Mardi, 19. septembre 2017 9:18 09

- **largeur** : nombre maximal de caractères à prendre en compte (on peut en lire moins s'il y a rencontre d'un séparateur ou d'un caractère invalide) ;
- **h|l|L** :

  h : l'élément correspondant est l'adresse d'un *short int*. Ce modificateur n'a de signification que pour les caractères de conversion : *d*, *i*, *n*, *o*, *u*, ou *x* ;

  l : l'élément correspondant est l'adresse d'un élément de type :

  - *long int* pour les caractères de conversion *d*, *i*, *n*, *o*, *u* ou *x* ;
  - *double* pour les caractères de conversion *e* ou *f* ;

  L : l'élément correspondant est l'adresse d'un élément de type *long double*. Ce modificateur n'a de signification que pour les caractères de conversion *e*, *f* ou *g*.

- **conversion** : ce caractère précise à la fois le type de l'élément correspondant (nous l'avons indiqué ici en italique) et la manière dont sa valeur sera exprimée. Les types numériques indiqués correspondent au cas où aucun modificateur n'est utilisé (voir ci-dessus). Il ne faut pas perdre de vue que l'élément correspondant est toujours désigné par son adresse. Ainsi, par exemple, lorsque nous parlons de *signed int*, il faut lire : « adresse d'un *signed int* » ou encore « pointeur sur un *signed int* ».
  - **d** : *signed int* exprimé en décimal ;
  - **o** : *signed int* exprimé en octal ;
  - **i** : *signed int* exprimé en décimal, en octal ou en hexadécimal ;
  - **u** : *unsigned int* exprimé en décimal ;
  - **x** : *int* (*signed* ou *unsigned*) exprimé en hexadécimal ;
  - **f**, **e** ou **g** : *float* écrit indifféremment en notation décimale (éventuellement sans point) ou exponentielle (avec e ou E) ;
  - **c** : suivant la longueur, correspond à :
    - un *caractère* lorsqu'aucune longueur n'est spécifiée ou que celle-ci est égale à *1* ;
    - une *suite de caractères* lorsqu'une longueur différente de *1* est spécifiée. Dans ce cas, il ne faut pas perdre de vue que la fonction reçoit une adresse et que donc, dans ce cas, elle lira le nombre de caractères spécifiés et les rangera à partir de l'adresse indiquée. Il est bien sûr préférable que la place nécessaire ait été réservée. Notez bien qu'il ne s'agit pas ici d'une véritable chaîne, puisqu'il n'y aura pas (à l'image de ce qui se passe pour le code *%s*) d'introduction du caractère *\0* à la suite des caractères rangés en mémoire ;
  - **s** : *chaîne de caractères*. Il ne faut pas perdre de vue que la fonction reçoit une adresse et que donc, dans ce cas, elle lira tous les caractères jusqu'à la rencontre d'un séparateur (ou un nombre de caractères égal à la longueur éventuellement spécifiée) et elle les rangera à partir de l'adresse indiquée. Il est donc préférable que la place nécessaire ait été réservée. Notez bien qu'ici un caractère *\0* est stocké à la suite des caractères rangés en

Livre_ProgrammerC++.book Page 765 Mardi, 19. septembre 2017 9:18 09

mémoire et que sa place aura dû être prévue (si l'on lit *n* caractères, il faudra de la place sur *n+1*) ;

- **n** : *int*, dans lequel sera placé le nombre de caractères lus correctement jusqu'ici. Aucun caractère n'est donc lu par cette spécification ;
- **p** : *pointeur* exprimé en hexadécimal, sous la forme employée par *printf* (elle dépend de l'implémentation).

1.7 Entrées-sorties de caractères

FGETC ***int* fgetc (*FILE* * flux)**

Lit le caractère courant du *flux* indiqué. Fournit :

- le résultat de la conversion en *int* du caractère *c* (considéré comme *unsigned int*) si l'on n'était pas en fin de fichier ;
- la valeur *EOF* si la fin de fichier était atteinte.

Notez que *fgetc* ne fournit de valeur négative qu'en cas de fin de fichier, quel que soit le code employé pour représenter les caractères et quel que soit l'attribut de signe attribué par défaut au type *char*.

FGETS ***char* * fgets (*char* * ch, *int* n, *FILE* * flux)**

Lit au maximum *n-1* caractères sur le *flux* mentionné (en s'interrompant éventuellement en cas de rencontre d'un caractère *\n*), les range dans la chaîne d'adresse *ch*, puis complète le tout par un caractère *\0*. Le caractère « *\n* », s'il a été lu, est lui aussi rangé dans la chaîne (donc juste avant le *\0*). Cette fonction fournit en retour :

- la valeur *NULL* si une éventuelle erreur a eu lieu ou si une fin de fichier a été rencontrée ;
- l'adresse *ch*, dans le cas contraire.

FPUTC ***int* fputc (*int* c, *FILE* * flux)**

Écrit sur le *flux* mentionné la valeur de *c*, après conversion en *unsigned char*. Fournit la valeur du caractère écrit (qui peut donc, éventuellement, être différente de celle du caractère reçu) ou la valeur *EOF* en cas d'erreur.

FPUTS ***int* fputs (*const char* * ch, *FILE* * flux)**

Écrit la chaîne d'adresse *ch* sur le *flux* mentionné. Fournit la valeur *EOF* en cas d'erreur et une valeur non négative dans le cas contraire.

GETC ***int* getc (*FILE* * flux)**

Macro effectuant la même chose que la fonction *fgetc*.

Livre_ProgrammerC++.book Page 766 Mardi, 19. septembre 2017 9:18 09

GETCHAR ***int* getchar (*void*)**

Macro effectuant la même chose que l'appel de la macro :

fgetc (stdin)

GETS ***char** * gets (*char** ch)**

Lit des caractères sur l'entrée standard (*stdin*), en s'interrompant à la rencontre d'une fin de ligne (*\n*) ou d'une fin de fichier, et les range dans la chaîne d'adresse *ch*, en remplaçant le *\n* par *\0*. Fournit :

- la valeur *NULL* si une erreur a eu lieu ou si une fin de fichier a été rencontrée, alors qu'aucun caractère n'a encore été lu ;
- l'adresse *ch*, dans le cas contraire.

PUTC ***int* putc (*int* c, *FILE** flux)**

Macro effectuant la même chose que la fonction *fputc*.

PUTCHAR ***int* putchar (*int* c)**

Macro effectuant la même chose que l'appel de la macro *putc*, avec *stdout* comme adresse de flux. Ainsi :

putchar (c)

est équivalent à :

putc (c, stdout)

PUTS ***int* puts (*const char** ch)**

Écrit sur l'unité standard de sortie (*stdout*) la chaîne d'adresse *ch*, suivie d'une fin de ligne (*\n*). Fournit *EOF* en cas d'erreur et une valeur non négative dans le cas contraire.

1.8 Entrées-sorties sans formatage

FREAD ***size_t* fread (*void** adr, *size_t* taille, *size_t* nblocs, *FILE** flux)**

Lit, sur le *flux* spécifié, au maximum *nblocs* de *taille* octets chacun et les range à l'adresse *adr*. Fournit le nombre de blocs réellement lus.

FWRITE ***size_t* fwrite (*const void** adr, *size_t* taille, *size_t* nblocs, *FILE** flux)**

Écrit, sur le *flux* spécifié, *nblocs* de *taille* octets chacun, à partir de l'adresse *adr*. Fournit le nombre de blocs réellement écrits.

Livre_ProgrammerC++.book Page 767 Mardi, 19. septembre 2017 9:18 09

1.9 Action sur le pointeur de fichier

FSEEK ***int*** **fseek (*FILE* * flux, *long* noct, *int* org)**

Place le pointeur du *flux* indiqué à un endroit défini comme étant situé à *noct* octets de l' « origine » spécifiée par *org :*

org = SEEK_SET correspond au début du fichier ;

org = SEEK_CUR correspond à la position actuelle du pointeur ;

org = SEEK_END correspond à la fin du fichier ;

Dans le cas des fichiers de texte (si l'implémentation les différencie des autres), les seules possibilités autorisées sont l'une des deux suivantes :

- *noct = 0* ;
- *noct* a la valeur fournie par *ftell* (voir ci-dessous) et *org = SEEK_SET*.

FTELL ***long*** **ftell (*FILE* *flux)**

Fournit la position courante du pointeur du *flux* indiqué (exprimée en octets par rapport au début du fichier) ou la valeur *-1L* en cas d'erreur.

1.10 Gestion des erreurs

FEOF ***int*** **feof (*FILE* * flux)**

Fournit une valeur non nulle si l'indicateur de fin de fichier du *flux* indiqué est activé et la valeur *0* dans le cas contraire.

2 Tests de caractères et conversions majuscules-minuscules (*cctype*)

ISALNUM ***int*** **isalnum (*char* c)**

Fournit la valeur *1* (vrai) si *c* est une lettre ou un chiffre et la valeur *0* (faux) dans le cas contraire.

ISALPHA ***int*** **isalpha (*char* c)**

Fournit la valeur *1* (vrai) si *c* est une lettre et la valeur *0* (faux) dans le cas contraire.

ISDIGIT **int isdigit (char c)**

Fournit la valeur *1* (vrai) si *c* est un chiffre et la valeur *0* (faux) dans le cas contraire.

Livre_ProgrammerC++.book Page 768 Mardi, 19. septembre 2017 9:18 09

ISLOWER ***int*** **islower (*char* c)**

Fournit la valeur *1* (vrai) si *c* est une lettre minuscule et la valeur *0* (faux) dans le cas contraire.

ISSPACE ***int*** **isspace (*char* c)**

Fournit la valeur 1 (vrai) si *c* est un séparateur (espace, saut de page, fin de ligne, tabulation horizontale ou verticale) et la valeur 0 (faux) dans le cas contraire.

ISUPPER ***int*** **isupper (*char* c)**

Fournit la valeur *1* (vrai) si *c* est une lettre majuscule et la valeur *0* (faux) dans le cas contraire.

3 Manipulation de chaînes (*cstring*)

STRCPY ***char*** *** strcpy (*char* * but, *const char* * source)**

Copie la chaîne *source* à l'adresse *but* (y compris le *\0* de fin) et fournit en retour l'adresse de *but*.

STRNCPY ***char*** *** strncpy (*char* * but, *const char* * source, *int* lgmax)**

Copie au maximum *lgmax* caractères de la chaîne *source* à l'adresse *but* en complétant éventuellement par des caractères *\0* si cette longueur maximale n'est pas atteinte. Fournit en retour l'adresse de *but*.

STRCAT ***char*** *** strcat (*char* * but, *const char* * source)**

Recopie la chaîne *source* à la fin de la chaîne *but* et fournit en retour l'adresse de *but*.

STRNCAT ***char*** *** strncat (*char* * but, *const char* * source, *size_t* lgmax)**

Recopie au maximum *lgmax* caractères de la chaîne *source* à la fin de la chaîne *but* et fournit en retour l'adresse de *but*.

STRCMP ***int*** **strcmp (*const char* * chaine1, *const char* * chaine2)**

Compare *chaine1* et *chaine2* et fournit :

- une valeur négative si *chaine1 < chaine2* ;
- une valeur positive si *chaine1 > chaine2* ;
- zéro si *chaine1 = chaine2*.

Livre_ProgrammerC++.book Page 769 Mardi, 19. septembre 2017 9:18 09

STRNCMP ***int*** **strncmp (*const char* * chaine1, *const char* * chaine2, *size_t* lgmax)**

Travaille comme *strcmp*, en limitant la comparaison à un maximum de *lgmax* caractères.

STRCHR ***char*** *** strchr (*const char* * chaine, *char* c)**

Fournit un pointeur sur la première occurrence du caractère *c* dans la chaîne *chaine*, ou un pointeur nul si ce caractère n'y figure pas.

STRRCHR ***char*** *** strrchr (*const char* * chaine, *char* c)**

Fournit un pointeur sur la dernière occurrence du caractère *c* dans la chaîne *chaine* ou un pointeur nul si ce caractère n'y figure pas.

STRSPN ***size_t*** **strspn (*const char* * chaine1, *const char* * chaine2)**

Fournit la longueur du segment initial de *chaine1* formé entièrement de caractères appartenant à *chaine2*.

STRCSPN ***size_t*** **strcspn (*const char* * chaine1, *const char* * chaine2)**

Fournit la longueur du segment initial de *chaine1* formé entièrement de caractères n'appartenant pas à *chaine2*.

STRSTR ***char*** *** strstr (*const char* * chaine1, *const char* * chaine2)**

Fournit un pointeur sur la première occurrence dans *chaine1* de *chaine2* ou un pointeur nul si *chaine2* ne figure pas dans *chaine1*.

STRLEN ***size_t*** **strlen (*const char* * chaine)**

Fournit la longueur de *chaine*.

MEMCPY ***void*** *** memcpy (*void* * but, *const void* * source, *size_t* lg)**

Copie *lg* octets depuis l'adresse *source* à l'adresse *but* qu'elle fournit comme valeur de retour (il ne doit **pas** y avoir **de recoupement** entre *source* et *but*).

MEMMOVE ***void*** *** memmove (*void* * but, *const void* * source, *size_t* lg)**

Copie *lg* octets depuis l'adresse *source* à l'adresse *but* qu'elle fournit comme valeur de retour (il peut y avoir **recoupement** entre *source* et *but*).

Livre_ProgrammerC++.book Page 770 Mardi, 19. septembre 2017 9:18 09

4 Fonctions mathématiques (cmath)

SIN ***double*** **sin (*double* x)**

COS ***double*** **cos (*double* x)**

TAN ***double*** **tan (*double* x)**

ASIN ***double*** **asin (*double* x)**

ACOS ***double*** **acos (*double* x)**

ATAN ***double*** **atan (*double* x)**

ATAN2 ***double*** **atan2 (*double* y, *double* x)**

Fournit la valeur de arctan(y/x).

SINH ***double*** **sinh (*double* x)**

Fournit la valeur de sh(x).

COSH ***double*** **cosh (*double* x)**

Fournit la valeur de ch(x).

TANH ***double*** **tanh (*double* x)**

Fournit la valeur de th(x).

EXP ***double*** **exp (*double* x)**

LOG ***double*** **log (*double* x)**

Fournit la valeur du logarithme népérien de x : Ln(x) (ou Log(x)).

LOG10 ***double*** **log10 (*double* x)**

Fournit la valeur du logarithme à base 10 de x : log(x).

POW ***double*** **pow (*double* x, *double* y)**

Fournit la valeur de x^y.

SQRT ***double*** **sqrt (*double* x)**

CEIL ***double*** **ceil (*double* x)**

Fournit (sous forme d'un *double*) le plus petit entier qui ne soit pas inférieur à x.

Livre_ProgrammerC++.book Page 771 Mardi, 19. septembre 2017 9:18 09

FLOOR ***double* floor (*double* x)**

Fournit (sous forme d'un *double*) le plus grand entier qui ne soit pas supérieur à *x*.

FABS ***double* fabs (*double* x)**

Fournit la valeur absolue de *x*.

5 Utilitaires (*cstdlib*)

ATOF ***double* atof (*const char* * chaine)**

Fournit le résultat de la conversion en *double* du contenu de *chaine*. Cette fonction ignore les éventuels séparateurs de début et, à l'image de ce que fait le code format *%f*, utilise les caractères suivants pour fabriquer une valeur numérique. Le premier caractère invalide arrête l'exploration.

ATOI ***int* atoi (*const char* * chaine)**

Fournit le résultat de la conversion en *int* du contenu de *chaine*. Cette fonction ignore les éventuels séparateurs de début et, à l'image de ce que fait le code format *%d*, utilise les caractères suivants pour fabriquer une valeur numérique. Le premier caractère invalide arrête l'exploration.

ATOL ***long* atol (*const char* * chaine)**

Fournit le résultat de la conversion en *long* du contenu de *chaine*. Cette fonction ignore les éventuels séparateurs de début et, à l'image de ce que fait le code format *%ld*, utilise les caractères suivants pour fabriquer une valeur numérique. Le premier caractère invalide arrête l'exploration.

RAND ***int* rand (*void*)**

Fournit un nombre entier aléatoire (en fait pseudo-aléatoire), compris dans l'intervalle [0, *RAND_MAX*]. La valeur prédéfinie *RAND_MAX* est au moins égale à *32767*.

SRAND ***void* srand (*unsigned int* graine)**

Modifie la « graine » utilisée par le « générateur de nombres pseudo-aléatoires » de *rand*. Par défaut, cette graine a la valeur *1*.

CALLOC ***void** * calloc (*size_t* nb_blocs, *size_t* taille)**

Alloue l'emplacement nécessaire à *nb_blocs* consécutifs de chacun *taille* octets, initialise chaque octet à zéro et fournit l'adresse correspondante lorsque l'allocation a réussi ou un pointeur nul dans le cas contraire.

Livre_ProgrammerC++.book Page 772 Mardi, 19. septembre 2017 9:18 09

MALLOC ***void** * **malloc (*size_t* taille)**

Alloue un emplacement de *taille* octets, sans l'initialiser, et fournit l'adresse correspondante lorsque l'allocation a réussi ou un pointeur nul dans le cas contraire.

REALLOC ***void* realloc (*void** adr, *size_t* taille)**

Modifie la taille d'une zone d'adresse *adr* préalablement allouée par *malloc* ou *calloc*. Ici, *taille* représente la nouvelle taille souhaitée, en octets. Cette fonction fournit l'adresse de la nouvelle zone ou un pointeur nul dans le cas où la nouvelle allocation a échoué (dans ce dernier cas, le contenu de la zone reste inchangé). Lorsque la nouvelle taille est supérieure à l'ancienne, le contenu de l'ancienne zone est conservé (il a pu éventuellement être alors recopié). Dans le cas où la nouvelle taille est inférieure à l'ancienne, seul le début de l'ancienne zone (c'est-à-dire *taille* octets) est conservé.

FREE ***void* free (*void** adr)**

Libère la mémoire d'adresse *adr*. Ce pointeur doit obligatoirement désigner une zone préalablement allouée par *malloc*, *calloc* ou *realloc*. Si *adr* est nul, cette fonction ne fait rien.

EXIT ***void* exit (*int* etat)**

Termine l'exécution du programme. Cette fonction ferme les fichiers ouverts en vidant les tampons, détruit les objets dynamiques (mais pas les objets automatiques) et rend le contrôle au système, en lui fournissant la valeur *etat*. La valeur *0* est considérée comme une « fin normale ».

ABS ***int* abs (*int* n)**

Fournit la valeur absolue de *n*.

LABS ***long* abs (*long* n)**

Fournit la valeur absolue de *n*.

6 Macro de mise au point (*cassert*)

ASSERT ***void* assert (*int* exptest)**

Si le symbole *NDEBUG* est défini au moment où le préprocesseur rencontre la directive *#include <assert.h>*, la macro *assert* sera sans effet et sans valeur. Dans le cas contraire, la macro *asssert* introduit une instruction d'arrêt conditionnel de l'exécution. Plus précisément, si l'expression

Livre_ProgrammerC++.book Page 773 Mardi, 19. septembre 2017 9:18 09

exptest vaut 0, il y aura impression, sur la sortie standard d'erreur, d'un message de la forme :

Assertion failed : exptest, nom_fichier, line xxxx

On y trouve :

- l'expression concernée : *exptest* ;

- le nom du fichier source concerné, *nom_fichier* ;

- le numéro de la ligne correspondante du fichier source *xxxx*.

Il y aura ensuite appel de la fonction *abort* qui interrompra l'exécution du programme. Si l'expression *exptest* a une valeur différente de 0, la macro *assert* ne fera rien.

Notez que le symbole *NDEBUG* n'est défini dans aucun fichier en-tête. C'est au programme de le prévoir s'il souhaite inhiber l'effet des appels de *assert*.

Cette macro ne peut jamais être redéfinie sous la forme d'une fonction.

7 Gestion des erreurs (*cerrno*)

ERRNO **errno**

Représente une *lvalue* de type *int* qui peut être utilisée par certaines fonctions de la bibliothèque standard. Sa valeur est initialisée à zéro au démarrage du programme. Elle doit être modifiée comme indiqué par la norme dans quelques rares cas ; en dehors de cela, elle peut être modifiée par n'importe quelle fonction, même en dehors d'une situation d'erreur.

La norme ne précise pas si *errno* doit être défini sous forme d'une macro ou d'un symbole global. Le comportement du programme est indéterminé si l'on annule la définition de *errno* ou si l'on définit un autre identificateur de même nom.

8 Branchements non locaux (*csetjmp*)

Le symbole *jmp_buf* est un synonyme d'un type tableau permettant de sauvegarder l'état de l'environnement et une adresse de retour, pour assuer le bon fonctionnement de *setjmp* et *longjmp*.

SETJMP ***int* setjmp (*jmp_buf* env)**

Cette **macro** sauvegarde l'environnement actuel et l'adresse d'appel dans la variable *env*. Fournit 0 comme valeur de retour en cas d'appel direct et une valeur non nulle lorsque l'appel s'est fait par l'intermédiaire de *longjmp*. La

Livre_ProgrammerC++.book Page 774 Mardi, 19. septembre 2017 9:18 09

norme laisse la liberté à l'implémentation de définir *setjmp* comme une macro ou comme un identificateur global. Si le programme annule la définition de *setjmp* ou s'il définit un autre identificateur de même nom, le comportement est indéterminé.

LONGJMP ***void* longjmp (*jmp_buf* env, *int* etat)**

Restaure l'environnement (préalablement sauvegardé par *setjmp*), à partir du contenu de la variable *env*. Reprend l'exécution à l'adresse précédemment conservée, comme si la valeur *etat* était la valeur de retour de *setjmp*. Si l'on appelle *longjmp* avec 0 comme valeur de *etat*, *longjmp* « force » une valeur de retour égale à 1.