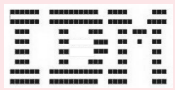
Bibliothèque Standard du Langage C 2005 - v 3.0

Bob CORDEAU
cordeau@onera.fr

Mesures Physiques
IUT d'Orsay

15 mai 2006



Avant-propos

Ce document présente la bibliothèque standard du langage C.

On y trouve d'une part un index alphabétique des fonctions standards, et d'autre part les fonctions et macro-définitions, les constantes, les variables et les types prédéfinis. Quelques exemples permettent d'illustrer les fonctions importantes.

Cette version est destinée à évoluer en fonction des remarques émises.

Sommaire 1/3

Sommaire 2/3

Sommaire 3/3

Première partie I

Index des fonctions standards

Index (1/n)

`void` `abort(void) ;`	interruption programme	*stdlib.h*
`int` `abs(int) ;`	valeur absolue entière	*stdlib.h*
`double` `acos(double) ;`	arc cosinus	*math.h*
`char*` `asctime(struct tm*) ;`	conversion heure/chaîne	*time.h*
`double` `asin(double) ;`	arc sinus	*math.h*
`void` `assert(int) ;`	vérification programme	*assert.h*

Index (2/n)

`double atan(double) ;`	arc tangente	*math.h*
`double atan2(double, double) ;`	arc tangente y/x	*math.h*
`int atexit(void(*)(void)) ;`	fonction à appeler en fin de programme	*stdlib.h*
`double atof(const char*) ;`	conversion chaîne/`double`	*stdlib.h*
`int atoi(const char*) ;`	conversion chaîne/`int`	*stdlib.h*
`long atol(const char*) ;`	conversion chaîne/`long`	*stdlib.h*

Index (3/n)

`void*`
`bsearch(const void*,`
`  const void*,`
`  size_t,`
`  size_t,`
`  int(*)(const void*,`
`  const void*));`
	recherche binaire dans un tableau	*stdlib.h*

`void*`
`calloc(size_t,`
`  size_t);`
	allocation initialisée de mémoire	*stdlib.h*

`double` `ceil(double);`	plus petit entier supérieur ou égal	*math.h*
`void` `clearerr(FILE*);`	suppression des indications d'erreurs et de fin de fichier	*stdio.h*

Index (4/n)

`clock_t` `clock(void) ;`	temps machine consommé par le programme appelant	*time.h*
`double` `cos(double) ;`	cosinus	*math.h*
`double` `cosh(double) ;`	cosinus hyperbolique	*math.h*
`char*` `ctime(const time_t*) ;`	conversion heure/chaîne	*time.h*
`double` `difftime(time_t, time_t) ;`	calcule une différence de temps en secondes	*time.h*
`div_t` `div(int, int) ;`	calcule du quotient et du reste	*stdlib.h*

Index (5/n)

`void exit(int) ;`	fin programme	*stdlib.h*
`double exp(double) ;`	exponentiation	*math.h*
`double fabs(double) ;`	valeur absolue d'un réel	*math.h*
`int fclose(FILE*) ;`	ferme fichier	*stdio.h*
`int fcloseall(void) ;`	ferme tous les fichiers	*stdio.h*
`int feof(FILE*) ;`	test de fin de fichier	*stdio.h*
`int ferror(FILE*) ;`	test d'erreur lecture/écriture	*stdio.h*
`int fflush(FILE*) ;`	vide tampon de fichier	*stdio.h*

Index (6/n)

`int fgetc(FILE*) ;`	lit caractère dans fichier	*stdio.h*
`int fgetpos(FILE*, fpos_t*) ;`	détermine position dans fichier	*stdio.h*
`char* fgets(char*, int, FILE*) ;`	lit chaîne dans fichier	*stdio.h*
`double floor(double) ;`	plus grand entier inférieur ou égal	*math.h*
`double fmod(double, double) ;`	reste de la division flottante	*math.h*
`FILE* fopen(const char*, const char*) ;`	ouvre fichier	*stdio.h*

Index (7/n)

`int fprintf(FILE*, const char*, ...);`	écriture formatée dans un fichier	*stdio.h*
`int fputc(int, FILE*);`	écrit caractère dans fichier	*stdio.h*
`int fputs(const char*, FILE*);`	écrit chaîne dans fichier	*stdio.h*
`size_t fread(void*, size_t, size_t, FILE*);`	lit bloc dans fichier	*stdio.h*
`void free(void*);`	libère mémoire	*stdlib.h*

Index (8/n)

`FILE*` `freopen(const char*, const char*, FILE*);`	ferme fichier et affecte pointeur à nouveau fichier	*stdio.h*
`double` `frexp(double, int*);`	sépare mantisse et exposant	*math.h*
`int` `fscanf(FILE*, const char*, ...);`	lecture formatée dans un fichier	*stdio.h*
`int` `fseek(FILE*, long, int);`	repositionne un pointeur dans fichier	*stdio.h*

Index (9/n)

`int fsetpos(FILE*, const fpos_t*);`	positionnement dans fichier	*stdio.h*
`long ftell(FILE*);`	donne position dans fichier	*stdio.h*
`size_t fwrite(void*, size_t, size_t, FILE*);`	écrit bloc dans fichier	*stdio.h*
`int getc(FILE*);`	lit caractère dans fichier	*stdio.h*
`int getchar(void);`	lit caractère sur `stdin`	*stdio.h*
`char* getenv(const char*);`	recherche variable d'environnement	*stdlib.h*

Index (10/n)

`char*` `gets(char*) ;`	lit chaîne sur `stdin`	*stdio.h*
`struct tm*` `gmtime(const time_t*) ;`	conversion secondes/date et heure	*time.h*
`int` `isalnum(int) ;`	alphanumérique ?	*ctype.h*
`int` `isalpha(int) ;`	lettre ?	*ctype.h*
`int` `iscntrl(int) ;`	caractère de contrôle ?	*ctype.h*
`int` `isdigit(int) ;`	chiffre ?	*ctype.h*
`int` `isgraph(int) ;`	caractère graphique ?	*ctype.h*
`int` `islower(int) ;`	lettre minuscule ?	*ctype.h*

Index (11/n)

`int isprint(int);`	caractère imprimable ?	*ctype.h*
`int ispunct(int);`	signe de ponctuation ?	*ctype.h*
`int isspace(int);`	caractère d'espacement ?	*ctype.h*
`int isupper(int);`	lettre majuscule ?	*ctype.h*
`int isxdigit(int);`	chiffre hexadécimal ?	*ctype.h*
`long labs(long);`	valeur absolue d'un `long`	*math.h*
`double ldexp(double, int);`	exponentiation en base 2	*math.h*

Index (12/n)

`ldiv_t` `ldiv(long, long);`	quotient et du reste d'un `long`	*math.h*
`struct tm*` `localtime(const time_t);`	conv. secondes/date et heure	*time.h*
`double` `log(double);`	logarithme népérien	*math.h*
`double` `log10(double);`	logarithme décimal	*math.h*
`void` `longjmp(jmp_buf, int);`	restaure environnement	*setjmp.h*
`void*` `malloc(size_t);`	allocation mémoire	*stdlib.h*

Index (13/n)

`void*`
`memchr(const void*,`
`int,`
`size_t);`
	recherche première occurrence d'un caractère dans un buffer	*string.h*

`int`
`memcmp(const void*,`
`const void*,`
`size_t);`
	compare les premiers octets de deux buffers	*string.h*

`void*`
`memcpy(void*,`
`const void*,`
`size_t);`
	copie n octets de source vers cible	*string.h*

`void*`
`memmove(void*,`
`const void*);`
	déplace source vers cible	*string.h*

Index (14/n)

`void*` `memset(void*, int, size_t);`	remplit le début d'un buffer avec un caractère	*string.h*
`time_t` `mktime(struct tm*);`	conversion heure locale vers heure normalisée	*time.h*
`double` `modf(double, double);`	parties entière et décimale d'un `double`	*math.h*
`void` `perror(const char*);`	écrit message d'erreur sur `stderr`	*stdio.h*
`double` `pow(double, double);`	calcule x^y	*math.h*

Index (15/n)

`int printf(const char*, ...);`	écriture formatée sur `stdout`	*stdio.h*
`int putc(int, FILE*);`	écrit caractère dans fichier	*stdio.h*
`int putchar(int);`	écrit caractère sur `stdout`	*stdio.h*
`int puts(const char*);`	écrit chaîne sur `stdout`	*stdio.h*
`void qsort(void*, size_t, size_t, int(*)(const void*, const void*));`	tri rapide de tableau	*stdlib.h*

Index (16/n)

`int raise(int) ;`	envoie un signal au programme exécutant	*signal.h*
`int rand(void) ;`	génère un nombre pseudo-aléatoire	*stdlib.h*
`void* realloc(void*, size_t) ;`	change taille bloc mémoire	*stdlib.h*
`int remove(const char*) ;`	supprime fichier	*stdio.h*
`int rename(const char*, const char*) ;`	renomme fichier	*stdio.h*
`void rewind(FILE*) ;`	retour au début du fichier	*stdio.h*

Index (17/n)

`int scanf(const char*, ...);`	lecture formatée depuis `stdin`	*stdio.h*
`void setbuf(FILE*, char*);`	crée buffer de fichier	*stdio.h*
`int setjmp(jmp_buf);`	sauve l'environnement de pile	*stdio.h*
`int setvbuf(FILE*, char*, int, size_t);`	contrôle taille de buffer et type de bufférisation	*stdio.h*
`void (*signal(int, void(*)(int))(int);`	traitement signal	*signal.h*

Index (18/n)

`double sin(double);`	sinus	*math.h*
`double sinh(double);`	sinus hyperbolique	*math.h*
`int sprintf(char*, const char*, ...);`	écriture formatée dans buffer	*stdio.h*
`double sqrt(double);`	racine carrée	*math.h*
`void srand(unsigned int);`	initialise générateur aléatoire	*stdlib.h*
`int sscanf(const char*, const char*, ...);`	lecture formatée depuis buffer	*stdio.h*

Index (19/n)

`char* strcat(char*, char*);`	concaténation de chaînes	*string.h*
`char* strchr(const char*, int);`	recherche première occurrence d'un caractère dans une chaîne	*string.h*
`int strcmp(const char*, const char*);`	comparaison de chaînes	*string.h*
`char* strcpy(char*, const char*);`	copie de chaînes	*string.h*
`int strcspn(const char*, const char*);`	recherche d'une sous-chaîne sans les motifs d'une autre	*string.h*

Index (20/n)

`char*` `strerror(int) ;`	pointe sur le message d'erreur en paramètre	*string.h*
`size_t` `strlen(const char*) ;`	calcule longueur de chaîne	*string.h*
`char*` `strncat(char*, const char*, size_t) ;`	concaténation de *n* caractères	*string.h*
`int` `strncmp(const char*, const char*, size_t) ;`	compare début de chaînes	*string.h*

Index (21/n)

`char* strncpy(char*, const char*, size_t);`	copie début de chaîne	*string.h*
`char* strpbrk(const char*, const char*);`	recherche dans une chaîne de caractères contenus dans une autre	*string.h*
`char* strrchr(const char*, int);`	recherche dernière occurrence d'un caractère dans une chaîne	*string.h*
`int strspn(const char*, const char*);`	longueur d'une sous-chaîne contenant uniquement des caractères d'une autre	*string.h*

Index (22/n)

`char*` `strstr(const char*, const char*);`	vérifie que la première chaîne est contenue dans la seconde	*string.h*
`double` `strtod(const char*, char**);`	conversion chaîne/`double`	*stdlib.h*
`char*` `strtok(char*, const char*);`	extrait des tokens de la première chaîne séparés par les caractères de la seconde	*string.h*
`long` `strtol(const char*, char**, int);`	conversion chaîne/`long`	*stdlib.h*

Index (23/n)

`unsigned long` `strtoul(const char*, char**, int) ;`	conversion chaîne/ `unsigned long`	*stdlib.h*
`int` `system(const char*) ;`	fait exécuter une commande par le système d'exploitation	*stdlib.h*
`double` `tan(double) ;`	tangente	*math.h*
`double` `tanh(double) ;`	tangente hyperbolique	*math.h*
`time_t` `time(time_t*) ;`	heure courante en secondes	*time.h*
`FILE*` `tmpfile(void) ;`	crée fichier temporaire	*stdio.h*
`char*` `tmpnam(char*) ;`	crée nom pour fichier temporaire	*stdio.h*

Index (24/n)

`int tolower(int) ;`	conversion en minuscules	*ctype.h*
`int toupper(int) ;`	conversion en majuscules	*ctype.h*
`int ungetc(int, FILE*) ;`	réécrit caractère dans fichier lu	*stdio.h*
`<type> va_arg(va_list, <type>) ;`	donne paramètre suivant de la fonction	*stdarg.h*
`void va_end(va_list) ;`	fixe pointeur d'argument sur `NULL`	*stdarg.h*
`void va_start(va_list, <name_last_fixparameter>) ;`	initialise pointeur d'argument	*stdarg.h*

Index (25/n)

`int vfprintf(FILE*, const char*, va_list);`	comme `fprintf`, avec un pointeur vers une liste de paramètres	*stdio.h*
`int vprintf(const char*, va_list);`	comme `printf`, avec un pointeur vers une liste de paramètres	*stdio.h*
`int vsprintf(char*, const char*, va_list);`	comme `sprintf`, avec un pointeur vers une liste de paramètres	*stdio.h*

Deuxième partie II

Bibliothèque standard C 1/2

Macro-définition et exemple

`assert()` assure que l'assertion est vérifiée.

```
#include <stdio.h>
#include <assert.h>

struct ITEM {
  int key, value;
};

void addItem(struct ITEM *itemptr) {
  assert(itemptr != NULL); // echec : on sort.
  // add item to list...
}

int main(void) {
  addItem(NULL); // on tente d'ajouter un item nul
  return 0;
}
```

Les fonctions (1/2)

- `isalnum()` teste si le caractère est alphanumérique.
- `isalpha()` teste si le caractère est alphabétique.
- `iscntrl()` teste si l'argument est un caractère de contrôle.
- `isdigit()` teste si le caractère est numérique.
- `isgraph()` teste si le caractère est visible.
- `islower()` teste si le caractère représente une lettre minuscule.
- `isprint()` teste si le caractère est imprimable.

Les fonctions (2/2)

- `ispunct()` teste si le caractère est un signe de ponctuation.
- `isspace()` teste si le caractère est un espacement.
- `isupper()` teste si le caractère représente une lettre majuscule.
- `isdigit()` teste si le caractère est un chiffre hexadécimal valide.
- `tolower()` convertit le caractère en sa représentation minuscule.
- `toupper()` convertit le caractère en sa représentation majuscule.

Exemple : fonction isdigit()

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char c = 'C';

  if (isdigit(c))
    printf("%c est un digit\n",c);
  else
    printf("%c n'est pas un digit\n", c);

  return 0;
}
```

Exemple : fonction tolower()

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(void)
{
    int length;
    char string[] = "CECI est UNE chaine DE caracteres";

   length = strlen(string);
   for (int i = 0; i < length; ++i)
   {
       string[i] = tolower(string[i]);
   }
   printf("%s\n", string);

   return 0;
}
```

Les constantes et variables prédéfinies

Constantes :

- `EDOM` : Error code for math domain error
- `EILSEQ`
- `ERANGE` : Code d'erreur d'un calcul en dehors des limites

Variable :

- `extern int errno ;`

Exemple : affichage de la liste des erreurs disponibles

```
#include <errno.h>
#include <stdio.h>

extern char *_sys_errlist[];

main()
{
   int i = 0;

   while(_sys_errlist[i++]) printf("%s\n", _sys_errlist[i]);
   return 0;
}
```

Les constantes prédéfinies 1/5

- `FLT_RADIX` : base de l'exposant dans la représentation d'un flottant
- `FLT_MANT_DIG` : taille de la mantisse dans la représentation d'un réel flottant
- `DBL_MANT_DIG` : idem pour un `double`
- `LDBL_MANT_DIG` : idem pour un `long double`
- `FLT_EPSILON` : différence entre la valeur réelle 1.0 et le plus petit flottant immédiatement supérieur
- `DBL_EPSILON` : idem pour un `double`
- `LDBL_EPSILON` : idem pour un `long double`

Les constantes prédéfinies 2/5

- `FLT_DIG` : nombre de chiffres de la partie décimale d'un réel ne subissant aucune modification après conversion dans un réel flottant
- `DBL_DIG` : idem pour un `double`
- `LDBL_DIG` : idem pour un `long double`
- `FLT_MIN_EXP` : le plus petit entier négatif, x, tel que FLT_RADIX^{x-1} est un flottant normalisé
- `DBL_MIN_EXP` : idem pour un `double`
- `LDBL_MIN_EXP` : idem pour un `long double`

Les constantes prédéfinies 3/5

- `FLT_MIN` : le plus petit flottant positif normalisé
- `DBL_MIN` : idem pour un `double`
- `LDBL_MIN` : idem pour un `long double`
- `FLT_MIN_10_EXP` : le plus petit entier négatif, x, tel que 10^x est un réel flottant normalisé
- `DBL_MIN_10_EXP` : idem pour un `double`
- `LDBL_MIN_10_EXP` : idem pour un `long double`

Les constantes prédéfinies 4/5

- `FLT_MIN_10_EXP` : le plus petit entier négatif, x, tel que 10^x est un réel flottant normalisé
- `DBL_MIN_10_EXP` : idem pour un `double`
- `LDBL_MIN_10_EXP` : idem pour un `long double`
- `FLT_MAX_EXP` : le plus grand entier positif, x, tel que FLT_RADIX^{x-1} est un flottant
- `DBL_MAX_EXP` : idem pour un `double`
- `LDBL_MAX_EXP` : idem pour un `long double`

Les constantes prédéfinies 5/5

- `FLT_MAX` : le plus grand flottant représentable
- `DBL_MAX` : idem pour un `double`
- `LDBL_MAX` : idem pour un `long double`
- `FLT_MAX_10_EXP` : le plus grand entier, x, tel que 10^x est un flottant
- `DBL_MAX_10_EXP` : idem pour un `double`
- `LDBL_MAX_10_EXP` : idem pour un `long double`
- `FLT_ROUNDS` : mode courant de calcul d'arrondi

Les constantes prédéfinies 1/2

- `CHAR_BIT` : nombre de bits dans un `char`
- `MB_LEN_MAX` : le plus grand nombre d'octets nécessaires à la contenance d'un caractère international
- `CHAR_MIN` : la plus petite valeur pouvant être contenue dans un `char`
- `INT_MIN` : idem pour un `int`
- `LONG_MIN` : idem pour un `long int`
- `SCHAR_MIN` : idem pour un `signed char`
- `SHRT_MIN` : idem pour un `short int`

Les constantes prédéfinies 2/2

- `CHAR_MAX` : la plus grande valeur pouvant être contenue dans un `char`
- `INT_MAX` : idem pour un `int`
- `LONG_MAX` : idem pour un `long int`
- `SCHAR_MAX` : idem pour un `signed char`
- `SHRT_MAX` : idem pour un `short int`
- `UCHAR_MAX` : idem pour un `unsigned char`
- `USHRT_MAX` : idem pour un `unsigned short int`
- `UINT_MAX` : idem pour un `unsigned int`
- `ULONG_MAX` : idem pour un `unsigned long int`

Les constantes prédéfinies

- NULL : peut être utilisé comme représentant le pointeur nul.
- LC_CTYPE : définition de l'alphabet
- LC_COLLATE : définition de l'ordre des caractères
- LC_TIME : définition du format de l'heure et de la date
- LC_NUMERIC : définition du format de représentation des nombres
- LC_MONETARY : définition du format de représentation des masses monétaires
- LC_MESSAGES : gestion des messages multilingues
- LC_ALL : catégorie prioritaire par défaut

Les fonctions de manipulation des diverses catégories :

- `setlocale()` modifie ou retrouve la valeur d'une catégorie.
- `localeconv()` retrouve la définition du format de représentation.

Type :

```
struct lconv {
  char *decimal_point;   // decimal point character
  char *thousands_sep;   // thousands separator
  char *grouping;        // digit grouping
  char *int_curr_symbol; // international currency symbol
  char *currency_symbol; // national currency symbol
  char *mon_decimal_point; // currency decimal point
  char *mon_thousands_sep; // currency thousands separator
  char *mon_grouping;    // currency digits grouping
  char *positive_sign;   // currency plus sign
  char *negative_sign;   // currency minus sign
  char int_frac_digits;  // internal curr. fract. digits
  char frac_digits;      // currency fractional digits
  char p_cs_precedes;    // currency plus location
  char p_sep_by_space;   // currency plus space ind.
  char n_cs_precedes;    // currency minus location
  char n_sep_by_space;   // currency minus space ind.
  char p_sign_posn;      // currency plus position
  char n_sign_posn;      // currency minus position
} ;
```

Constantes prédéfinies

- HUGE_VAL : indique que la valeur n'est pas représentable

M_E : e
M_LN2 : $\ln 2$
M_PI_2 : $\pi/2$
M_2_PI : $2/\pi$
M_SQRT1_2 : $\sqrt{1/2}$

M_LOG2E : $\log_2 e$
M_LN10 : $\ln 10$
M_PI_4 : $\pi/4$
M_2_SQRTPI : $2/\sqrt{\pi}$

M_LOG10E : $\log e$
M_PI : π
M_1_PI : $1/\pi$
M_SQRT2 : $\sqrt{2}$

Remarque importante : Toutes les fonctions mathématiques manipulent des variables de type double

Les fonctions trigonométriques :

- `acos()` : arc cosinus.
- `asin()` : arc sinus.
- `atan()` et `atan2()` : arc tangente.
- `cos()` : cosinus.
- `sin()` : sinus.
- `tan()` : tangente.

Les fonctions trigonométriques hyperboliques :

- `cosh()` : cosinus hyperbolique.
- `sinh()` : sinus hyperbolique.
- `tanh()` : tangente hyperbolique.

Les fonctions exponentielles et logarithmiques :

- `exp()` : exponentielle (e^x).
- `frexp()` : extraction de la mantisse et de l'exposant d'un nombre.
- `ldexp()` : exponentiation en base 2 ($x.2^y$).
- `log()` : logarithme népérien.
- `log10()` : logarithme décimal.
- `modf()` : décomposition d'un réel en partie entière et décimale.
- `pow()` : élévation à la puissance (x^y).
- `sqrt()` : extraction de racine carrée.

Les fonctions diverses :

- `ceil()` : calcul de l'entier inférieur le plus proche (fonction plancher).
- `fabs()` : valeur absolue.
- `floor()` : calcul de l'entier supérieur le plus proche (fonction plafond).
- `fmod()` : reste de la division.

Exemple : fonction atan2()

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    double result;
    double x = 90.0, y = 45.0;

  result = atan2(y, x);
  printf("L'arc tangente de %lf est %lf", (y / x), result);

  return 0;
}
```

Exemple : fonction pow()

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 2.0, y = 3.0;

  printf("%lf puissance %lf : %lf\n", x, y, pow(x, y));

  return 0;
}
```

Exemple : fonction modf()

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double fraction, entier;
    double nombre = 100000.567;

  fraction = modf(nombre, &entier);
  printf("Soit le nombre : %lf \n", nombre);
  printf("sa partie entiere : %lf \n", entier);
  printf("sa partie fractionnaire : %lf", fraction);

  return 0;
}
```

Troisième partie III

Bibliothèque standard C 2/2

Fonction, macro-définition et type

Fonction :

- `longjmp()` : restaure l'environnement d'exécution.

Macro-définition :

- `setjmp()` : sauve l'environnement d'exécution.

Type :

- `struct jmp_buf ;`

Les constantes prédéfinies

- `SIGABRT` : arrêt anormal.
- `SIGFPE` : erreur dans un calcul arithmétique.
- `SIGILL` : tentative d'exécution d'une instruction interdite.
- `SIGINT` : interruption logicielle.
- `SIGSEGV` : violation d'accès à la mémoire.
- `SIGTERM` : demande de terminaison.

Fonctions, macro-définitions et type

Fonctions :

- `signal()` : définition de l'action à réaliser à l'instant de la délivrance d'un signal.
- `raise()` : envoi d'un signal.

Macro-définitions :

- `SIG_DFL` : comportement par défaut.
- `SIG_ERR` : erreur.
- `SIG_IGN` : ignore la délivrance.

Type :

- `sig_atomic_t ;`

Macro-définitions et type

Macro-définitions :

- `va_start()` : pointe sur le début de la liste des arguments.
- `va_arg()` : décode l'argument courant et pointe sur le suivant.
- `va_end()` : termine le balayage des arguments.

Type :

- `va_list ;`

Constante prédéfinie, macro-définition et types prédéfinis

Constante prédéfinie :

- `NULL` : peut être utilisé comme représentant le pointeur nul.

Macro-définition :

- `offsetof()` : calcule l'adresse relative d'un champ par rapport au début de la structure.

Types prédéfinis :

- `size_t` : type des tailles d'objets.
- `ptrdiff_t` : type du résultat de la soustraction de deux adresses.
- `wchar_t` : type des caractères étendus.

Les constantes prédéfinies 1/2

- `_IOFBF` : vidage du tampon lors de son débordement.
- `_IOLBF` : vidage du tampon après écriture d'une fin de ligne.
- `_IONBF` : vidage à chaque écriture.
- `BUFSIZ` : taille du tampon par défaut.
- `EOF` : valeur indiquant la fin du flot.
- `FILENAME_MAX` : longueur maximum d'une référence de fichier.
- `FOPEN_MAX` : nombre maximum d'objets que l'environnement permet de manipuler simultanément.
- `L_TMPNAM` : longueur des références d'objets temporaires créés dans l'environnement.

Les constantes prédéfinies 2/2

- `NULL` : peut être utilisé comme représentant le pointeur nul.
- `SEEK_CUR` : les déplacements sont mesurés relativement à la position courante.
- `SEEK_END` : les déplacements sont mesurés relativement à la fin du flot.
- `SEEK_SET` : les déplacements sont mesurés relativement au début du flot.
- `TMP_MAX` : nombre minimum de fichiers temporaires distincts que l'environnement est capable de fournir.
- `stdin` : objet représentant le flot d'entrée standard.
- `stdout` : objet représentant le flot de sortie standard.
- `stderr` : objet représentant le flot de sortie d'erreurs standard.

Les fonctions (1/5)

- `clearerr()` repositionne les indicateurs d'erreur et de marque de fin.
- `fclose()` libère le flot.
- `feof()` teste la fin du flot.
- `ferror()` teste l'indicateur d'erreur du flot.
- `fflush()` vide le tampon associé.
- `fgetc()` lit un caractère dans le flot.
- `fgetpos()` récupère la position courante du flot.
- `fgets()` lit une chaîne dans le flot.

Les fonctions (2/5)

- `fopen()` alloue un flot.
- `fprintf()` réalise une écriture formatée dans un flot.
- `fputc()` écrit un caractère dans un flot.
- `fputs()` écrit une chaîne dans un flot.
- `fread()` lit une suite d'octets dans un flot.
- `freopen()` réalloue un flot.
- `fscanf()` réalise une lecture formatée dans un flot.
- `fseek()` positionnement dans le flot.

Les fonctions (3/5)

- `fsetpos()` modifie la position courante dans le flot.
- `ftell()` récupère la position courante dans le flot.
- `fwrite()` écrit une suite d'octets dans un flot.
- `getc()` lit un caractère dans un flot.
- `getchar()` lit un caractère dans le flot d'entrée standard.
- `gets()` lit une chaîne de caractères dans le flot d'entrée standard.
- `perror()` écrit un message d'erreur sur le flot de sortie d'erreurs standard.
- `printf()` écrit un message formaté sur le flot de sortie standard.

Les fonctions (4/5)

- `putc()` écrit un caractère sur un flot.
- `putchar()` écrit un caractère sur le flot de sortie standard.
- `puts()` écrit une chaîne de caractères sur le flot de sortie standard.
- `remove()` supprime un fichier.
- `rename()` renomme un fichier.
- `rewind()` remet la position courante au début du flot.
- `scanf()` réalise une lecture formatée sur le flot d'entrée standard.
- `setbuf()` modifie la zone utilisée pour contenir le tampon.
- `setvbuf()` modifie les caractéristiques du tampon.

Les fonctions (5/5)

- `sprintf()` réalise une écriture formatée dans une zone mémoire.
- `sscanf()` réalise une lecture formatée depuis une zone mémoire.
- `tmpfile()` alloue un flot temporaire.
- `tmpnam()` créé une référence de fichier temporaire unique.
- `ungetc()` remet un caractère dans le flot.
- `vfprintf()` réalise une écriture formatée dans un flot.
- `vprintf()` réalise une écriture formatée dans le flot de sortie standard.
- `vsprintf ()` réalise une écriture formatée dans une zone mémoire.

Les types

- `FILE` : type des objets utilisés pour réaliser des opérations d'entrées/sorties.
- `fpos_t` : type des mesures de positionnement.
- `size_t` : type des mesures de taille.

Exemple : fonctions fopen(), fclose(), printf(), scanf() et sprintf()

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    FILE *fp;
    char buf[13] = "0123456789_", c;

  printf("entrez un caractere : "); scanf("%c", &c);
  // concatenation dans un buffer
  sprintf(buf, "%s%c", buf, c);
  // creation d'un fichier de 10 caracteres
  fp = fopen("FICTIF.txt", "w");
  fwrite(&buf, strlen(buf), 1, fp);
  fclose(fp); // fermeture du flot

  return 0;
}
```

Les constantes prédéfinies

- `EXIT_FAILURE` : permet de renvoyer une valeur indiquant l'échec de la commande.
- `EXIT_SUCCESS` : permet de renvoyer une valeur indiquant la terminaison normale de la commande.
- `MB_CUR_MAX` : donne le nombre maximal de caractères nécessaires pour contenir un caractère étendu dans l'internationalisation courante.
- `NULL` : peut être utilisé comme représentant le pointeur nul.
- `RAND_MAX` : indique la valeur maximum pouvant être renvoyée par `rand()`.

Les types prédéfinis 1/2

- `div_t` : est utilisé pour représenter le quotient et le reste d'une division entière :

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

- `ldiv_t` : est utilisé pour représenter le quotient et le reste d'une division entière sur des entiers longs :

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

Les types prédéfinis 2/2

- `size_t` : permet de ranger une valeur représentant une longueur.
- `wchar_t` : est le type permettant de ranger un caractère étendu.

Les fonctions (1/4)

- `abort()` permet d'arrêter brutalement l'exécution du processus courant en envoyant un signal `SIGABRT`. Le contrôle n'est jamais rendu à l'appelant même si ce signal est capturé. La terminaison est alors forcée en utilisant `exit(EXIT_FAILURE)`.
- `abs()` renvoie la valeur absolue d'un entier.
- `atexit()` permet d'enregistrer une fonction qui sera appelée au moment de la terminaison du processus par `exit()` ou par la sortie de `main()`.
- `atof()` convertit une chaîne de caractères représentant un nombre réel en un `double`.
- `atoi()` convertit une chaîne de caractères représentant un nombre entier en un `int`.
- `atol()` convertit une chaîne de caractères représentant un entier en un `long`.

Les fonctions (2/4)

- `bsearch()` effectue la recherche par dichotomie d'un élément dans un tableau d'éléments ordonnés.
- `calloc()` réalise l'allocation initialisée d'un tableau sur le tas.
- `div()` calcule le quotient et le reste d'une division entière.
- `exit()` réalise la terminaison du processus.
- `free()` libère une zone mémoire précédemment allouée sur le tas.
- `getenv()` permet de consulter l'environnement.
- `labs()` renvoie la valeur absolue d'un entier long.

Les fonctions (3/4)

- `ldiv()` calcule le quotient et le reste d'une division entière effectuée sur des entiers longs.
- `malloc()` alloue une zone mémoire sur le tas.
- `mblen()` calcule le nombre de caractères étendus contenus dans la chaîne de caractères.
- `mbstowcs()` convertit une chaîne de multi-caractères en chaîne de caractères étendus.
- `mbtowc(*)` convertit un multi-caractère en caractère étendu.
- `qsort()` applique le *quick-sort* sur un tableau d'éléments.
- `rand()` renvoie un nombre pseudo-aléatoire.
- `realloc()` modifie la taille d'une zone mémoire située sur le tas.

Les fonctions (4/4)

- `srand()` initialise le germe d'une séquence aléatoire.
- `strtod()` convertit une chaîne de caractères représentant un réel en `double`.
- `strtol()` convertit une chaîne de caractères représentant un entier long en `long`.
- `strtoul()` convertit une chaîne de caractères représentant un entier long non signé en `unsigned long`.
- `system()` permet de lancer un interpréteur de commande afin de réaliser la commande.
- `wcstombs()` convertit une chaîne de caractères étendus en une chaîne de multi-caractères.

Exemple : fonctions malloc(), free()

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void)
{
    char *str;

  // alloue une chaine de 6 caracteres
  str = (char *) malloc(6*sizeof(char));

  // copie "Hello" dans la chaine et l'affiche
  strcpy(str, "Hello");
  printf("Chaine : %s\n", str);

  // libere l'allocation
  free(str);

  return 0;
}
```

Exemple : fonctions atof()

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   float f;
   char *str = "12345.67";

 f = atof(str);
 printf("Chaine : %s\t flottant = %f\n", str, f);

 return 0;
}
```

Constante et type prédéfinis

Constante prédéfinie :

- `NULL` : peut être utilisé comme représentant le pointeur nul.

Type prédéfini :

- `size_t` : type des tailles d'objets.

Les fonctions (1/3)

- `memchr()` recherche d'un caractère dans une zone mémoire.
- `memcmp()` comparaison lexicographique de deux zones mémoire.
- `memcpy()` copie d'une zone mémoire.
- `memmove()` copie d'une zone mémoire avec recouvrement.
- `memset()` initialisation d'une zone mémoire.
- `strcat()` concaténation de deux chaînes de caractères.
- `strncat()` concaténation limitée en longueur de deux chaînes.

Les fonctions (2/3)

- `strchr()` recherche la première occurrence d'un caractère dans un chaîne.
- `strrchr()` recherche la dernière occurrence d'un caractère dans une chaîne.
- `strcmp()` comparaison lexicographique de deux chaînes de caractères.
- `strncmp()` comparaison lexicographique limitée en longueur.
- `strcoll()` comparaison lexicographique de deux chaînes de caractères internationaux.
- `strcpy()` copie d'une chaîne de caractère.
- `strncpy()` copie limitée en longueur.

Les fonctions (3/3)

- `strcspn()` calcule la longueur du plus grand préfixe ne contenant pas certains caractères.
- `strspn()` calcule la longueur du plus grand préfixe ne contenant que certains caractères.
- `strpbrk()` calcule l'adresse du plus long suffixe commençant par un caractère choisi dans une liste.
- `strerror()` retrouve le message d'erreur associé.
- `strlen()` calcule la longueur d'une chaîne.
- `strstr()` calcule la position d'une sous-chaîne dans une chaîne.
- `strtok()` découpe une chaîne en lexèmes.
- `strxfrm()` transforme une chaîne en sa version internationalisée.

Exemple : fonctions strcat(), strcpy()

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char destination[8];
    char *car = "_", *c = "C++", *DevCpp = "Dev";

  strcpy(destination, DevCpp);
  strcat(destination, car);
  strcat(destination, c);

  printf("%s\n", destination);

  return 0;
}
```

Exemple : fonctions strcmp()

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
    int ptr1 = strcmp(buf2, buf1), ptr2 = strcmp(buf2, buf3);

  if (ptr1 > 0)
    printf("buffer 2 est plus grand que buffer 1\n");
  else
    printf("buffer 2 est plus petit que buffer 1\n");

  if (ptr2 > 0)
    printf("buffer 2 est plus grand que buffer 3\n");
  else
    printf("buffer 2 est plus petit que buffer 3\n");

  return 0;
}
```

Les constante prédéfinies

- `CLOCKS_PER_SEC` : indique combien de tops d'horloge sont générés par seconde.
- `NULL` : peut être utilisé comme représentant le pointeur nul.

Les types prédéfinis

- `clock_t` : représente un nombre de tops d'horloge.
- `size_t` : représente une longueur.
- `time_t` : représente une mesure de temps sur laquelle des calculs arithmétiques peuvent être réalisés.
- `struct tm` : représente une mesure de temps calendaire :

```
struct tm {
  int tm_sec;  // seconds after the minute (from 0)
  int tm_min;  // minutes after the hour (from 0)
  int tm_hour; // hour of the day (from 0)
  int tm_mday; // day of the month (from 1)
  int tm_mon;  // month of the year (from 0)
  int tm_year; // years since 1900 (from 0)
  int tm_wday; // days since Sunday (from 0)
  int tm_yday; // day of the year (from 0)
  int tm_isdst; // Daylight Saving Time flag
};
```

Les fonctions

- `asctime()` convertit un temps calendaire en chaîne de caractères.
- `clock()` renvoie le nombre de tops d'horloge consommés par le processus courant.
- `ctime()` convertit un temps arithmétique en une chaîne de caractères.
- `difftime()` calcule la différences de deux temps arithmétiques.
- `gmtime()` convertit un temps arithmétique en temps calendaire UTC.
- `localtime()` convertit un temps arithmétique en temps calendaire local.
- `mktime()` normalise un temps calendaire.
- `strftime()` formate un temps calendaire.
- `time()` renvoie le temps arithmétique courant.

Pour aller plus loin...


P. J. Plauger
The Standard C Library
PRENTICE hall.

S.P. Harbisson et G.L. Steele Jr.
Langage C, manuel de référence
Masson.

Peter Prinz et Ulla Kirch–Prinz
C précis et concis
O'Reilly.

Pour aller plus loin...


P. J. Plauger
The Standard C Library
PRENTICE hall.


S.P. Harbisson et G.L. Steele Jr.
Langage C, manuel de référence
Masson.

Peter Prinz et Ulla Kirch–Prinz
C précis et concis
O'Reilly.

Pour aller plus loin...


P. J. Plauger
The Standard C Library
PRENTICE hall.


S.P. Harbisson et G.L. Steele Jr.
Langage C, manuel de référence
Masson.


Peter Prinz et Ulla Kirch–Prinz
C précis et concis
O'Reilly.