La bibliothèque standard C

1 stdlib.h

1.1 Types

- `size_t` type entier non signé du résultat de `sizeof()`
- `div_t` type structure du résultat de `div()`
- `ldiv_t` type structure du résultat de `ldiv()`

1.2 Macros

- `NULL` pointeur nul
- `EXIT_FAILURE` argument de `exit()`
- `EXIT_SUCCESS` argument de `exit()`
- `RAND_MAX` valeur maximale retournée par `rand()`

1.3 Fonctions

Arithmétique en nombre entiers

- `int abs(int i)` valeur absolue d'un entier
- `long int labs(long int i)` valeur absolue d'un entier long
- `div_t div(int numerateur, int denominateur)` calcul du quotient et du reste
- `ldiv_t ldiv(long int numerateur, long int denumerateur)` calcul du quotient et du reste (avec des entiers longs)

Algorithmes

- `void * bsearch(const void *cle, const void *t, size_t n, size_t taille, int (*compare)(const void*, const void*))` cherche dans le tableau `t` de `n` éléments la valeur `cle` à l'aide de la fonction `compare` (chaque élément occupe `taille` octets)
- `void *qsort(const void*t, size_t n, size_t n, size_t taille, int (*compare)(const void*, const void*))` tri le tableau `t` de `n` éléments à l'aide de la fonction `compare` (chaque élément occupe `taille` octets)
- `int rand(void)` retourne un entier pseudo-aléatoire entre 0 et `RAND_MAX`
- `void srand(unsigned int d)` valeur de départ pour la suite de valeurs retournées par `rand`

Conversions de chaîne

- `double atof(const char *)` convertit une chaîne en `double`
- `int atoi(const char *)` convertit une chaîne en `int`
- `long atol(const char *)` convertit une chaîne en `long`

Allocation mémoire

- `void * calloc(size_t n, size_t taille)` alloue l'espace mémoire pour un tableau de `n` éléments occupant chacun `taille` octets. Tout l'espace est initialisé à 0.
- `void free (void *p)` désalloue l'espace mémoire alloué à l'adresse `p`
- `void realloc(void *p, size_t taille)` réalloue un espace mémoire de `taille` octets
- `void * realloc(void *p, size_t taille)` réalloue un espace mémoire de `taille` octets en remplacement de celui alloué à l'adresse `p`. Le conteny, si les tailles sont compatibles, est inchangé.

1.4 Interaction avec l'environnement

- `void abort (void)` provoque une fin anormale du programme sauf si le signal `SIGABRT` est intercepté
- `int atexit(void (*f)(void))` enregistre la fonction `f` qui sera appelée lors de la fin anormale du programme
- `void exit(int etat)` provoque une fin normale du programme
- `char * getenv(const char * env)` retourne la valeur associaée à la variable d'environnement `env`
- `int system(const char *commande)` passe la `commande` au processeur de commandes (shell) pour qu'elle soit exécutée

2 stdio.h

2.1 Types

- `size_t` type entier non signé du résultat de `sizeof()`
- `FILE` type structure permettant d'enregistrer toutes les informations écessaires au contrôle d'un flot
- `fpos_t` type structure permettant d'identifier une position dans un fichier

2.2 Macros

- `NULL` pointeur nul
- `EOF` fin de fichier
- `FOPEN_MAX` nombre de fichiers pouvant être ouverts simultanément
- `stderr` de type `FILE *` : flot standard d'erreur
- `stdin` de type `FILE *` : flot standard d'entrée. Par défaut le clavier
- `stdout` de type `FILE *` : flot standard de sortie
- `SEEK_SET` déplacement (`fseek`) à partir du début du fichier
- `SEEK_CUR` déplacement (`fseek`) à partir de la position courante
- `SEEK_END` déplacement (`fseek`) à partir de la fin du fichier

2.3 Fonctions

Opérations sur les fichiers

- `int remove(const char *nf)` efface le fichier de nom `nf`
- `int rename(const char *ancien, const char *nouveau)` renome le fichier de nom `ancien` en `nouveau`

Fonctions d'accès aux fichiers

- `FILE * fopen(const char* nf, const char * mode)` ouvre le fichier de nom `nf` et lui associe un flot.
  Le `mode` est :
  - `r` lecture
  - `w` création écriture
  - `a` ajout

  On peut également préciser :
  - `b` fichier binaire
  - `+` mise à jour
- `int fclose(FILE *flot)` purge `flot` et ferme le fichier associé
- `int fflush(FILE *flot)` puge `flot`

Fonctions d'entrées-sorties avec format

- `int fprintf(FILE *flot, const char *format, ...)` écrit sur `flot` sous le contrôle de `format`
- `int fscanf(FILE * flot, const char * format, ...)` lit sur `flot` sous le contrôle de `format`
- `int printf(const char *format, ...)` écrit sur `stdout` sous le contrôle de `format`
- `int scanf(const char *format, ...)` lit sur `stdin` sous le contrôle de `format`
- `int sprintf(char *s, const char *format, ...)` écrit sur `s` sous le contrôle de `format`
- `int sscanf(char *s, const char *format, ...)` lit sur `s` sous le contrôle de `format`

Fonctions d'entrées-sorties caractères

- `int fgetc(FILE *flot)` lit le prochain caractère sur `flot`
- `int ungetc(int c, FILE *flot)` remet le caractère `c` sur `flot`
- `int getc(FILE *flot)` lit le prochain caractère sur `flot`. Peut être réalisée sous la forme de macro
- `int getchar(void)` lit le prochain caractère sur `stdin`
- `int fputc(int c, FILE *flot)` écrit le caractère `c` sur `flot`
- `int putc(int c, FILE * flot)` écrit le caractère `c` sur `flot`. Peut être réalisée sous la forme de macro
- `int putchar(int c)` écrit le caractère `c` sur `stdout`

- `char *fgets(char *s, int n, FILE *flot)` lit au plus `n-1` caractères sur `flot`. La lecture s'arrête au caractère fin de ligne (qui est conservé) ou à la fin de fichier. Un caractère nul est ajouté comme dernier caractère
- `char *gets(char *s)` lit des caractères sur `stdin`. La lecture s'arrête au caractère de fin de ligne (non conservé) ou à la fin du fichier. Un caractère nul est ajouté comme dernier caractère
- `fputs(const char *s, FILE *flot)` écrit `s` sur `flot`. Le caractère nul de fin n'est pas écrit
- `puts(const char *s)` écrit `s` et ajoute un caractère de fin de ligne sur `stdout`. Le caractère nul de fin n'est pas écrit

Fonctions d'entrée-sorties directes

- `size_t fread(void *t, size_t taille, size_t n, FILE *flot)` place dans `t` jusqu'à `n` éléments de taille `taille` lus sur le flot. Retourne le nombre d'éléments effectivement lus
- `size_t fwrite(void *t, size_t taille, size_t n, FILE *flot)` écrit jusqu'à `n` éléments de taille `taille` de `t` sur `flot`. Retourne le nombre d'éléments effectivement écrits

Fonctions de positionnement de fichier

- `long int ftell(FILE *flot)` retourne la position courante de `flot`
- `void rewind(FILE * flot)` possitionnement de `flot` au début du fichier
- `int fseek(FILE *flot, long int n, int mode)` positionnement de `flot` à `n` caractères (fichier binaire) selon `mode` (`SEEK_SET`, `SEEK_CUR` ou `SEEK_END`). Pour un fichier texte `n` doit être égal à 0 ou une valeur retournée par `ftell)` (`mode=SEEK_SET`)
- `int fgetpos(FILE *flot, fpos_t *pos)` range dans `pos` la valeur courante de l'indicateur de positionnement de fichier de `flot`. À utiliser avec les gros fichiers
- `int fsetpos(FILE *flot, fpos_t *pos)` remet l'indicateur de positionnement de fichier `flot` à la valeur qui est dans `pos`. À utiliser sur de gros fichiers.

Fonctions de traitement d'erreur

- `void clearerr(FILE *flot)` met à zéro les indicateurs d'erreur et de fin de fichier de `flot`
- `int feof(FILE *flot)` teste l'indicateur de fin de fichier de `flot`
- `int ferror(FILE *flot)` teste l'indicateur d'erreur de `flot`
- `void perror(char *message)` affiche sur `stderr` message suivi de " :"et d'un message d'erreur circonstancié

3 string.h

3.1 Groupes de fonctions

Les fonctions dont le nom commence par `mem` manipulent des séquences de caractères quelqconques. Un premier argument `void *s` pointe sur le début de la chaîne, un second argument `size_t n` indique le nombre maximum de caractères.

Les fonctions dont le nom commence par `strn` manipulent des séquences de caractères non nuls. Un premier argument `char *s` pointe sur le début de la châine, un second argment `size_t n` indique le nombre de maximum de caractères. La séauence se termine au premier caractère nul ou au plus au bout de `n` caractères.

Les fonctions dont le nom commence par `str` manipulent des séquences de caractères terminées par un caractère nul. Un unique argument `char *` pointe sur le début de la chaîne.

3.2 Fonctions

Fonctions de copie

- `void *memcpy(void *s1, const void *s2, size_t n)` copie de `n` caractères de `s2` dans `s1`. Si les objets se recouvrent le comportement est **indéterminé**
  N.B. *: Si les objets se recoupent, cette fonction est plus efficace qu'un* `for`
- `void *memmove(void *s1, const void*s2, size_t n)` copie de `n` caractères `s2` dans `s1`. Si les objets se recouvrent le comportement est correct
- `char *strcpy(char *s1, const char *s2)` copie la chaîne `s2` dans `s1`. Si les objets se recouvrent les comportement est **indéterminé**
  N.B. *: vérifie la taille*
- `char *strncpy(char *s1, const char *s2, size_t n)` copie de au plus `n` caractères de `s2` dans `s1`. Si les objets se recouvrent les comportement est **indéterminé**. On ajoute éventuellement des caractères nuls dans `s2` pour arriver à `n`

Fonctions de concaténation

- `char *strcat(char *s1, const char *s2)` copie la chaîne `s2` (y compris le caractère nul de fin) à la fin de `s1` en écrasant son caractère nul. Si les objets se recouvrent le comportement est **indéterminé**
  `N.B.` *: Cette fonction ne fait pas d'allocation*
- `char *strncat(char *s1, const char *s2, size_t n)` ajoute une copie de au plus `n` caractères de `s2` dans `s1`. On écrase le caractère nul de fin de `s1` et on ajoute un caractère nul au résultat. Si les objets se recouvrent le comportement est **indéterminé**

Fonctions de comparaison

- `char *memcmp(const *s1, const char *s2, size_t n)` compare les `n` premiers caractères de `s1` aux `n` premiers caractères `s2`. La valeur retournée est négative, nulle ou positive suivant que `s1` inférieur, égal ou supérieur à `s2`
- `char *strcmp(const char *s1, const char *s2)` compare la chaîne `s1` à la chaîne `s2`. La valeur retournée est négative, nulle ou positive suivant que `s1` inférieur, égal ou supérieur à `s2`
- `char *strncmp(const char *s1, const char *s2, size_t n)` compare au plus `n` caractères de `s1` et de `s2`. La valeur retournée est négative, nulle ou positive suivant que `s1` inférieur, égal ou supérieur à `s2`

Fonctions de recherche

- `char *memchr(const vois *s, int c, size_t n)` repère la première occurence de `c` dans les `n` premiers caractères de `s`. La fonction retourne un pointeur sur le caractère trouvé ou `NULL` s'il n'y est pas
- `char *strchr(const char *s, int c)` repère la premiere occurence de `c` dans `s`. La fonction retourne un pointeur sur le caractère trouvé ou `NULL` s'il n'y est pas

- `char *strrchr(const char *s, int c)` repère la dernière occurence de `c` dans `s`. La fonction retourne un pointeur sur le caractère trouvé ou `NULL` s'il n'y est pas
- `size_t strspn(const char *s1, const char *s2)` calcule la longeur du plus grand préfixe de `s1` ne contenant que des caractères de `s2`
- `size_t strcspn(const char *s1, const char *s2)` calcule la longeur du plus grand préfixe de `s1` ne contenant aucun caractère de `s2`
- `char *strpbrk(const char *s1, const char *s2)` repère dans `s1` la première occurrence d'un caractère de `s2`. La fonction retourne un pointeur sur le caractère trouvé ou `NULL` s'il n'y a aucun caractère de `s2` dans `s1`
- `char *strstr(const char *s1, const char *s2)` repère dans `s1` la première occurrence de `s2`. La fonction retourne un pointeur sur la chaîne trouvée ou `NULL` si `s2` n'est pas facteur de `s1`
- `char *strtok(char *s1, const char *s2)` fragmente `s1` selon les séparateur donnés dans `s2`. Au premier appel la fonction modifie `s1`. Pour les autres appels on met `NULL` comme premier argument. À chaque appel la fonction retourne un pointeur sur un nouveau mot puis `NULL` quand il n'y en a plus
  `N.B.` *: à utiliser avec précautions !*

Fonctions diverses

- `void *memset(void *s, int c, size_t n)` copie la valeur de `c` dans les `n` premiers caractères de `s`
- `char *strerror(int no)` retourne le message associé au numéro d'erreur `no`
- `size_t strlen(const char *s)` retourne la longueur de `s`

4 ctype.h

4.1 Fonctions de test

- `int isalpha(int c)` lettre ?
- `int islower(int c)` minuscule ?
- `int isupper(int c)` majuscule ?
- `int isdigit(int c)` chiffre ?
- `int isxdigit(int c)` chiffrehexadécimal ?
- `int isalnum(int c)` lettre ou chiffre ?
- `int isprint(int c)` imprimable (y compris l'espace) ?
- `int isgraph(int c)` imprimable (sauf l'espace) ?
- `int isspace(int c)` espace ?
- `int ispunct(int c)` ponctuation (autre que espace, lettre ou chiffre) ?
- `int iscntrl(int c)` contrôle ?

4.2 Fonctions de conversion

- `int tolower(int c)` en minuscule
- `int toupper(int c)` en majucule

5 time.h

5.1 Types

- `size_t` type entier non signé du résultat de `sizeof`
- `clock_t` type arithmétique utilisé pour représenter le temps machine
- `time_t` type arithmétique utilisé pour représenter la date (temps interne)
- `struc tm` type structure utilisé pour représenter la date (temps externe)
  - `int tm_sec` seconde $[0, 61]$
  - `int tm_min` minute $[0, 59]$
  - `int tm_hour` heure $[0, 23]$
  - `int tm_mday` jour du mois $[1, 31]$
  - `int tm_mon` mois $[1, 31]$
  - `int tm_year` année depuis 1900
  - `int tm_wday` jour $[dimanche = 0, 6]$
  - `int tm_yday` jour de l'année $[0, 365]$
  - `int tm_isdst` indicateur d'heure d'été

5.2 Macros

- `NULL` pointeur nul
- `CLOCKS_PER_SEC` pour convertir la valeur retournée par `clock` en secondes

5.3 Fonctions

Fonctions de manipulation de la date

- `time_t time(time_t *p_time)` retourne la date sous forme arithmétique (en paramètre `NULL` ou adresse où mettre le résultat)
- `double difftime (time_t date1, time_t date2)` différence en secondes entre deux dates sosu forme arithmétique
- `clock_t clock(void)` retourne le temps macjine utilisé (depuis l'appel de programme)

Fonctions de conversion de date

- `char *asctime(const struct tm *p_tm)` convertit la date sous forme structure en chaîne de caractères
- `char *ctime(const time_t *p_time)` convertut la date sous forme arithmétique en chaîne de caractère
- `time_t mktime(struct tm *p_tm)` convertit la date sous forme structure en date sous forme arithmétique
- `struct to *gmtime(const time_t *p_time)` convertit la date sous forme arithmétique en date sous forme structure (temps universel)
- `struct to *localtime(const time_t *p_time)` convertit la date sous forme arithmétique en date sous forme structure (temps local)
- `size_t *sfrftime(char *s, size_t max, const char *format, const struct tm *p_tm)` convertit la date sous forme structure en une chaîne `s` d'au plus `max` caractères sous le contrôle de `format`. Retourne le nombre de caractères de la chaîne créée

6 signal.h

6.1 Type

- `sig_atomic_t` type entier (aucun signal ne peut survenir pendant sa modification)

6.2 Macros

- `SIGABRT` signal de terminaison anormale
- `SIGFOE` signal d'opération arithmétique incorrecte
- `SIGILL` signal de fonction ou d'instruction interdite
- `SIGINT` signal d'interruption
- `SIGSEGV` signal d'accès incorrect à la mémoire
- `SIGTERM` signal de demande de terminaison envoyée au programme
- `SIG_DFL` second paramètre de `signal` : traitement par défaut
- `SIG_IGN` second paramètre de `signal` : ignorer le signal
- `SIG_ERR` valeur retournée par `signal` en cas d'échec

6.3 Fonctions

- `void(*signal(int signal, void (*f)(int)))(int)` fonction de récupération de signal : la fonction `f` sera appelée lors de la réception du signal `signal`. Retourne le précédent paramétrage (un pointeur de fonction) ou `SIG_ERR`
- `int raise(int signal)` envoie le `signal` au programme en cours d'exécution

7 math.h

7.1 Fonctions trigonométriques

- `double acos(double x)`
- `double asin(double x)`
- `double atan(double x)`
- `double atan2(double x, double y)`
- `double cos(double x)`
- `double sin(double x)`
- `double tan(double x)`

7.2 Fonctions hyperboliques

- `double sinh(double x)`
- `double cosh(double x)`
- `double tanh(double x)`

7.3 Fonctions exponentielles et logarithmiques

- `double exp(double x)`
- `double ldexp(double x, int n)` retourne $x \times 2^n$
- `double log(double x)`
- `double log10(double x)`

7.4 Fonctions d'élévation à une puissance

- `double pow(double x, double y)` retourne x^y
- `double sqrt(double x)` retourne $\sqrt{x}$

7.5 Fonctions diverses

- `double ceil(double x)` retourne la plus petite valeur entière supérieure ou égale à `x`
- `double (double x)` retourne la plus grand valeur entière supérieure ou égale à `x`
- `double fabs(double x)` retourne la valeur absolue de `x`
- `double fmod(double x, double y)` retourne le reste de la division de `x` par `y`
- `double modf(double x, double *p_partie_entiere)` décompose `x` en partie entière et en partie fractionnaire
- `double frexp(double x, double *p_exposant)` décompose `x` en partie fractionnaire (dans l'intervalle $[0.5, 1[$) et une puissance de 2