

Mémento de l'UE LU2IN018

Ce document a pour vocation de présenter un bref descriptif des fonctions offertes par les bibliothèques standards et qui sont susceptibles d'être utilisées dans l'UE.

Entrées - sorties

Prototypes disponibles dans `stdio.h`.

Entrées, sorties formatées

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

La fonction `printf` écrit sur la sortie standard (par défaut le terminal), un message correspondant au texte spécifié dans la chaîne `format`. Le texte affiché peut contenir des portions *variables* spécifiées par des codes de conversion précédés par le caractère `%`. Les arguments suivant `format` doivent correspondre (ordre et type) aux codes de conversion présents. Voici quelques codes de conversion courants :

- `%d` : entier
- `%c` : caractère
- `%s` : chaîne de caractères
- `%f` : nombre réel

`printf` renvoie le nombre de caractères imprimés et la chaîne de format peut contenir des codes de contrôle permettant le formatage comme `\n` pour forcer le passage à la ligne et `\t` pour insérer une tabulation.

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

Les fonctions `scanf` et `sscanf` permettent de saisir et analyser un texte saisi sur l'entrée standard, par défaut le clavier (`scanf`) ou depuis une chaîne de caractères passée en argument (`sscanf`). Le texte devra respecter le `format` spécifié et les arguments suivants doivent correspondre à des pointeurs sur des variables de type appropriés. Les codes de conversion son identiques à ceux de `printf`.

Entrées, sorties caractères

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

Lit un caractère dans le flux d'entrée standard (par défaut le clavier). La fonction retourne le code `EOF` en cas d'erreur, le caractère lu dans le cas contraire.

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

Affiche un caractère dans le flux de sortie standard (par défaut le terminal). La fonction retourne le code `EOF` en cas d'erreur, le caractère lu dans le cas contraire.

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

Affiche une chaîne de caractères dans le terminal et passe à la ligne, renvoie `EOF` en cas d'erreur.

Fichiers

Prototypes disponibles dans `stdio.h`.

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

Ouvre un fichier dont le chemin est spécifié par la chaîne `path` et retourne un pointeur de type `FILE *` (NULL en cas d'échec). L'argument `mode` permet de spécifier le type d'accès à réaliser sur le fichier :

- `[r]` pour un accès en lecture,
- `[w]` pour un accès en écriture et le contenu précédent du fichier est écrasé,
- `[a]` pour un accès en écriture, le contenu du fichier est préservé et les écritures sont effectuées à la suite des contenus déjà présents.

En cas d'erreur la fonction retourne le code `NULL` sinon un pointeur vers le fichier ouvert.

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

Cette fonction provoque la fermeture du fichier pointé par `fp`. En cas d'erreur la fonction retourne le code `EOF` sinon 0.

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

Identique à `printf` mais l'argument `stream` permet de spécifier le flux de sortie.

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

Identique à `scanf` mais l'argument `stream` permet de spécifier le flux d'entrée.

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

Identique à `putchar` mais l'argument `stream` permet de spécifier le flux de sortie.

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

Identique à `puts` mais l'argument `stream` permet de spécifier le flux de sortie et il n'y a pas d'ajout de passage à la ligne.

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

Identique à `getchar` mais l'argument `stream` permet de spécifier le flux d'entrée.

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

Lit au plus `size-1` octets dans le flux `stream`. La lecture s'arrête dès qu'un passage à la ligne est rencontré. Les octets lus sont stockés dans `s`. La fonction retourne `s` en cas de succès et `NULL` en cas d'erreur.

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

Lecture binaire de **nmemb** éléments de **size** octets dans le fichier **stream**. Les données lues sont stockées en mémoire à partir de l'adresse **ptr**. La fonction retourne le nombre d'éléments effectivement lus.

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

Écriture de **nmemb** éléments de **size** octets dans le fichier **stream**. Les données à écrire sont lues en mémoire à partir de l'adresse **ptr**. La fonction retourne le nombre d'éléments effectivement écrits.

Chaînes de caractères

Prototypes disponibles dans **string.h**.

Une chaîne de caractères correspond à un tableau de caractère et doit contenir un marqueur de fin \0.

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

Renvoie la longueur d'une chaîne de caractères (marqueur de fin \0 non compris).

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

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

Comparaison entre chaînes de caractères éventuellement limité aux **n** premiers caractères. La valeur renournée est :

- 0 si les deux chaînes sont identiques,
- négative si **s1** précède **s2** dans l'ordre lexicographique (généralisation de l'ordre alphabétique),
- positive sinon.

```
char *strcpy(char *dest, const char *src);
```

```
char *strncpy(char *dest, const char *src, size_t n);
```

Copie le contenu de la chaîne **src** dans la chaîne **dest** (marqueur de fin \0 compris). La chaîne **dest** doit avoir précédemment été allouée. La copie peut être limitée à **n** caractères et la valeur renournée correspond au pointeur de destination **dest**.

```
void *memcpy(void *dest, const void *src, size_t n);
```

Copie **n** octets à partir de l'adresse contenue dans le pointeur **src** vers l'adresse stockée dans **dest**. **dest** doit pointer vers une zone mémoire préalablement allouée et de taille suffisante. **memcpy** renvoie la valeur de **dest**.

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

Retourne le nombre de caractères de la chaîne **s** (marqueur de fin \0 non compris).

```
char *strdup(const char *s);
```

Cette fonction permet de dupliquer une chaîne de caractères, elle retourne un pointeur vers la chaîne nouvellement allouée. La nouvelle chaîne pourra être libérée avec la fonction **free**.

```
char *strcat(char *dest, const char *src);  
char *strncat(char *dest, const char *src, size_t n);
```

Ajoute la chaîne **src** à la suite de la chaîne **dst**. La chaîne **dest** devra avoir été allouée et être de taille suffisante. La fonction retourne **dest**.

```
char *strstr(const char *haystack, const char *needle);
```

La fonction renvoie un pointeur sur la première occurrence de la sous-chaîne **needle** rencontrée dans la chaîne **haystack**. Si la chaîne recherchée n'est pas présente, la fonction retourne NULL.

Conversion de chaînes de caractères

Prototypes disponibles dans **stdlib.h**.

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

La fonction convertit le début de la chaîne pointée par **nptr** en un entier de type **int**.

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

Cette fonction convertit le début de la chaîne pointée par **nptr** en un **double**.

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

Convertit le début de la chaîne **nptr** en un entier long. L'interprétation tient compte de la **base** et la variable pointée par **endptr** est affectée avec l'adresse du premier caractère invalide (au sens de la conversion).

Allocation dynamique de mémoire

Prototypes disponibles dans **stdlib.h**.

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

Alloue **size** octets de mémoire et retourne un pointeur générique correspondant à l'adresse du premier octet de la zone, renvoie NULL en cas d'échec.

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

Permet de modifier la taille d'une zone de mémoire allouée dynamiquement. **ptr** doit correspondre à l'adresse du premier octet de la zone précédemment allouée par **malloc** ou **realloc**. **size** correspond à la taille en octet de la nouvelle zone allouée. **realloc** garantie que la nouvelle zone contiendra les données présentes dans la zone initiale.

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

Libère une zone mémoire allouée dynamiquement. **ptr** doit correspondre à l'adresse du premier octet de la zone précédemment allouée par **malloc** ou **realloc**.