minishell :>

Lexique des fonctions autorisées

```
pr
                REDIRECTION_OUTFILE,
                REDIRECTION_INFILE,
                REDIRECTION_APPEND,
                // REDIRECTION_HEREDOC
        } type;intf             int printf(const char *format, ...);
```
Affiche du texte formaté sur la sortie standard.

```
malloc          void * malloc( size_t memorySize );
```
Allocation dynamiques de mémoire
```
free            void free( void * pointer );
```
Libération dynamiques de mémoire

```
fork            pid_t fork(void);
```
Créer un processus fils

```
wait            pid_t wait(int *status);
waitpid         pid_t waitpid(pid_t pid, int *status, int options);
```
wait, waitpid - Attendre qu'un processus change d'état

```
wait3           pid_t wait3(int *status, int options, struct rusage *rusage);
wait4           pid_t wait4(pid_t pid, int *status, int optio, struct rusage *rusage);
```
Attendre qu'un processus change d'état - Style BSD

```
execve          int execve(const char *file, char *const argv[], char *const envp[]);
```
Exécuter un programme
```
access            int access(const char *pathname, int mode);
```
Vérifie l'accès aux fichiers et test les permissions.
```
getenv            char *getenv(const char *name);
```
Récupère la valeur de l'environnement pour la variable d'environnement spécifiée.

```
dup             int dup(int oldfd);
dup2            int dup2(int oldfd, int newfd);
```
dup, dup2 - Dupliquer un descripteur de fichier

```
pipe            int pipe(int pipefd[2]);
```
Créer un pipe

```
open            int open(const char *pathname, int flags);
                int open(const char *pathname, int flags, mode_t mode);
```
Ouvrir ou créer éventuellement un fichier ou un périphérique
```
close           int close(int fd);
```
Fermer un fd
```
read            ssize_t read(int fd, void *buf, size_t count);
```
Lire depuis un fd
```
write           ssize_t write(int fd, const void *buf, size_t count);
```
Écrire dans un fd
```
unlink            int unlink(const char *pathname);
```
Supprime un lien vers un fichier.

opendir `DIR *opendir(const char *name);`
Ouvrir un répertoire
readdir `int readdir(DIR *dir, struct dirent *entry, struct dirent **result);`
Consulter un répertoire
closedir `int closedir(DIR *dir);`
Fermer un répertoire
signal `sighandler_t signal(int signum, sighandler_t handler);`
Gestion de signaux ANSI C
kill `int kill(pid_t pid, int sig);`
Envoyer un signal à un processus

exit `void exit(int status);`
Terminer normalement un processus

getcwd `char *getcwd(char *buf, size_t size);`
Obtenir le répertoire de travail courant
chdir `int chdir(const char *path);`
Changer le répertoire courant

stat `int stat(const char *path, struct stat *buf);`
lstat `int lstat(const char *path, struct stat *buf);`
fstat `int fstat(int fd, struct stat *buf);`
Obtenir l'état d'un fichier (file status)

strerror `char *strerror(int errnum);`
Obtenir le libellé d'un numéro d'erreur
errno `errno`
Code de la dernière erreur

sigaction `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
Gère les signaux en définissant des actions à exec lorsqu'un signal donné est reçu.
sigemptyset `int sigemptyset(sigset_t *set);`
Initialise un ensemble de signaux vide.
sigaddset `int sigaddset(sigset_t *set, int signum);`
Ajoute un signal spécifié à un ensemble de signaux.

isatty `int isatty(int fd);`
Vérifie si le descripteur de fichier spécifié correspond à un terminal.
ttyname `char *ttyname(int fd);`
Renvoie le nom du terminal associé au descripteur de fichier spécifié.
ttyslot `int ttyslot(void);`
Renvoie le num de l'entrée de la table des terminaux pour le term contrôlant le processus.

ioctl `int ioctl(int fd, unsigned long request, ...);`
Effectue diverses opérations de contrôle sur le périphérique associé au descripteur de fichier spécifié.

readline `char *readline(const char *prompt);`

Affiche un prompt et attend la saisie d'une ligne de texte par l'utilisateur.

rl_clear_history `void rl_clear_history(void);`

Efface l'historique des commandes précédemment saisies par l'utilisateur.

rl_on_new_line `void rl_on_new_line(void);`

Notifie GNU Readline qu'une nouvelle ligne de cmd est sur le point d'être affichée.

rl_replace_line `void rl_replace_line(const char *text, int clear_undo);`

Remplace la ligne éditée dans l'éditeur de ligne par la chaîne de texte spécifiée.

rl_redisplay `void rl_redisplay(void);`

Actualise l'affichage de la ligne de commande éditée dans l'éditeur de ligne.

add_history `void add_history(const char *line);`

Ajoute la ligne spécifiée à l'historique des commandes.

tcsetattr `int tcsetattr(int fd, int optional_actions, const struct termios *termios_p);`

Définit les paramètres du terminal pour le descripteur de fichier spécifié.

tcgetattr `int tcgetattr(int fd, struct termios *termios_p);`

Récupère les paramètres du terminal pour le descripteur de fichier spécifié.

tgetent `int tgetent(char *bp, const char *name);`

Charge les informations terminfo pour un terminal spécifié.

tgetflag `int tgetflag(const char *id);`

Obtient la valeur d'un drapeau terminfo pour le terminal actuel.

tgetnum `int tgetnum(const char *id);`

Obtient la valeur d'une capacité numérique terminfo pour le terminal actuel.

tgetstr `char *tgetstr(const char *id, char **area);`

Obtient la valeur d'une capacité de chaîne terminfo pour le terminal actuel.

tgoto `char *tgoto(const char *cap, int col, int row);`

Construit une séquence de commande terminfo en fonction de la capacité spécifiée.

tputs `int tputs(const char *str, int affcnt, int (*putc)(int));`

Écrit la chaîne terminfo spécifiée sur le terminal.

#includes

```
/* Pour readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay,
add_history */
# include <readline/readline.h>
# include <readline/history.h>
/* Pour printf, perror */
# include <stdio.h>
/* Pour malloc, free, exit */
# include <stdlib.h>
/* Pour write, access, open, read, close, fork, wait, waitpid, wait3, wait4,
isatty, ttyname, ttyslot, ioctl */
# include <unistd.h>
# include <fcntl.h>
# include <sys/types.h>
# include <sys/wait.h>
/* Pour signal, sigaction, sigemptyset, sigaddset, kill */
# include <signal.h>
/* Pour getcwd, chdir, stat, lstat, fstat, unlink, execve */
# include <sys/stat.h>
/* Pour dup, dup2, pipe */
# include <unistd.h>
/* Pour opendir, readdir, closedir */
# include <dirent.h>
/* Pour strerror, errno */
# include <string.h>
# include <errno.h>
/* Pour getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto,
tputs */
# include <term.h>
# include <termios.h>
# include <curses.h>

// C'est pas inclus dans le sujet, pourquoi c'est là?
/* Pour gettimeofday */
# include <sys/time.h>
/* Pour getrusage */
# include <sys/resource.h>
```