Entrées sorties

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

Flux

La bibliothèque `stdio.h` permet de réaliser des entrées/sorties portables du langage C.

Les entrées/sorties se font par des **flux** qui représentent des objets externes au programme, appelés fichiers :

- fichier texte ;
- fichier binaire (image, son, vidéo...).

Un flux est représenté par un pointeur sur une une structure de type `FILE`. Cette structure contient des informations concernant le fichier.

```
struct _IO_FILE {
  char* _IO_read_ptr; /* Current read pointer */
  char* _IO_read_end; /* End of get area. */
  char* _IO_read_base; /* Start of putback+get area. */
  char* _IO_write_base; /* Start of put area. */
  char* _IO_write_ptr; /* Current put pointer. */
  char* _IO_write_end; /* End of put area. */
  char* _IO_buf_base; /* Start of reserve area. */
  char* _IO_buf_end; /* End of reserve area. */

  struct _IO_FILE *_chain;
  int _fileno;
};
```

Buffer associé à un fichier

Pour limiter le nombre d'appels système, les caractères à lire/écrire dans un fichier sont placés dans un **buffer** (en pratique une chaîne de caractère de taille 1024) qui est vidé au fur et à mesure de la lecture/écriture.

Buffer associé à un fichier

Pour limiter le nombre d'appels système, les caractères à lire/écrire dans un fichier sont placés dans un **buffer** (en pratique une chaîne de caractère de taille 1024) qui est vidé au fur et à mesure de la lecture/écriture.

```
char c = 'a';
int e = 7;
scanf("%d", &e);
scanf("%c", &c);
printf("%c %d\n", c,  e);
```

Peut-on saisir une valeur pour `c` ?

Redirections des entrées sorties

Trois flux sont prédéfinis pour tout programme incluant la librairie `stdio.h` :

- `stderr` : flux d'erreurs standard, par défaut, l'écran ;
  - pas de buffer ;
- `stdout` : flux de sortie standard, par défaut, l'écran ;
  - `printf` écrit dans le flux `stdout` ;
- `stdin` : flux d'entrée standard, par défaut, le clavier.
  - `scanf` lit dans le flux `stdin`.

Redirections des entrées sorties

Trois flux sont prédéfinis pour tout programme incluant la librairie `stdio.h` :

- `stderr` : flux d'erreurs standard, par défaut, l'écran ;
  - pas de buffer ;
- `stdout` : flux de sortie standard, par défaut, l'écran ;
  - `printf` écrit dans le flux `stdout` ;
- `stdin` : flux d'entrée standard, par défaut, le clavier.
  - `scanf` lit dans le flux `stdin`.

L'utilisateur d'un programme peut modifier l'entrée et la sortie standard avec `<` et `>`

```
./prog <infile >outfile
```

`infile` et `outfile` ne sont pas considérés comme des arguments du programme.

Fonction fprintf

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

Similaire à la fonction `printf`, sauf que le fichier dans lequel on écrit est spécifié.

Fonction fprintf

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

Similaire à la fonction `printf`, sauf que le fichier dans lequel on écrit est spécifié.

`printf(const char *format, ...)` est équivalent à
`fprintf(stdout, const char *format, ...)`.

Fonction fprintf

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

Similaire à la fonction `printf`, sauf que le fichier dans lequel on écrit est spécifié.

`printf(const char *format, ...)` est équivalent à `fprintf(stdout, const char *format, ...)`.

L'équivalent existe pour `scanf`, i.e. la fonction `fscanf`.

Opérations sur les flux

La librairie `stdio.h` fournit des fonctions pour manipuler les flux :

- ouvrir un flux ;
- fermer un flux ;
- lecture dans un flux ;
- écriture dans un flux ;
- se déplacer dans un flux.

Ouverture d'un flux

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

- `path` est une chaîne de caractère qui contient le chemin (relatif ou absolu) du fichier ; il faut doubler l'antislash ;
- `mode` indique le mode d'ouverture du fichier. Par exemple pour la lecture au début du fichier on utilise `"r"`.

Ouverture d'un flux

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

- `path` est une chaîne de caractère qui contient le chemin (relatif ou absolu) du fichier ; il faut doubler l'antislash ;
- `mode` indique le mode d'ouverture du fichier. Par exemple pour la lecture au début du fichier on utilise `"r"`.

```
fopen("\\home\\user\\essai.c", "r");
fopen("fichierlocal", "r");
```

Ouverture d'un flux

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

- `path` est une chaîne de caractère qui contient le chemin (relatif ou absolu) du fichier ; il faut doubler l'antislash ;
- `mode` indique le mode d'ouverture du fichier. Par exemple pour la lecture au début du fichier on utilise `"r"`.

```
fopen("\\home\\user\\essai.c", "r");
fopen("fichierlocal", "r");
```

Retourne un pointeur sur le flux ou `NULL` si échec (par exemple le fichier n'existe pas).

Mode d'ouverture d'un fichier

- `"r"` : lecture à partir du début du fichier ; le fichier indiqué par le premier argument doit obligatoirement exister, sinon la fonction échoue ;
- `"w"` : écriture à partir du début du fichier ; si le fichier n'existe pas, il est créé ; si il existe, son contenu est effacé ;
- `"a"` : écriture à partir de la fin du fichier ; si le fichier n'existe pas, il est créé ;
- `"r+"` : lecture et écriture à partir du début du fichier ; le fichier indiqué par le premier argument doit obligatoirement exister, sinon la fonction échoue ;
- `"w+"` : lecture et écriture à partir du début du fichier ; si le fichier n'existe pas, il est créé ; si il existe, son contenu est effacé ;
- `"a+"` : lecture et écriture à partir de la fin du fichier ; si le fichier n'existe pas, il est créé.

Fermeture d'un fichier

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

- Libère le flux `stream`;
- renvoie 0 si réussit, `EOF` sinon.

```c
#include <stdio.h>

int main(void)
{
    FILE * fichier = fopen("fichier.txt", "r");
    if(fichier == NULL)
    {
        printf("erreur\n");
        /* ... */
    }

    /* Opérations sur le fichier ... */

    fclose(fichier);
    return 0;
}
```

Lecture d'un caractère

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

- Renvoie le caractère pointé par l'indicateur de position, et `EOF` si l'indicateur de position est à la fin du fichier ;
- avance l'indicateur de position d'un caractère si possible.

Lecture d'un caractère

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

- Renvoie le caractère pointé par l'indicateur de position, et `EOF` si l'indicateur de position est à la fin du fichier ;
- avance l'indicateur de position d'un caractère si possible.

La fonction `getchar` est équivalent à `fgetc(stdin)`.

Lecture d'un caractère

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

- Renvoie le caractère pointé par l'indicateur de position, et `EOF` si l'indicateur de position est à la fin du fichier ;
- avance l'indicateur de position d'un caractère si possible.

La fonction `getchar` est équivalent à `fgetc(stdin)`.

Aller à la fin d'un fichier texte :

```
int c;
do {
    c = fgetc(fichier);
} while (c != EOF);
```

Lecture d'une chaîne de caractères

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

Lit le flux caractère par caractère et s'arrête dès que

- `size-1` caractères ont été lus ;
- ou un retour à la ligne `'\n'` est lu ;
- ou `EOF` est lu.

Lecture d'une chaîne de caractères

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

Lit le flux caractère par caractère et s'arrête dès que

- `size-1` caractères ont été lus ;
- ou un retour à la ligne `'\n'` est lu ;
- ou `EOF` est lu.

Le pointeur `s` contient la chaîne lue, avec le caractère de fin de chaîne ajouté à la fin.

Retourne `s` ou `NULL` si erreur ou fin de fichier.

Implémentation de la fonction fgets

```
char* ffgets(char* s, int n, FILE* iop){
  int c;
  char* cs;

  cs = s;
  while(--n > 0 && (c = getc(iop)) != EOF)
    if((*cs++ = c) == '\n')
      break;
  *cs = '\0';
  return (c == EOF && cs == s) ? NULL : s;
}
```

Fonction fgets (suite)

```
int main(int argc, char** argv){
  char str[256];
  while(ffgets(str, 256, stdin) != NULL)
    printf("%s", str);
  return 0;
}
```

Écriture dans un flux

Écriture d'un caractère :

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

Écriture dans un flux

Écriture d'un caractère :

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

Écriture d'une chaîne :

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

Commande unix cat

```
int main(int argc, char** argv){
  FILE* fp;
  if(argc == 1) /* pas d'arguments: copie entree std */
    filecopy(stdin, stdout);
  else
    while(--argc > 0){
      if((fp = fopen(*++argv, "r")) == NULL){
        fprintf(stderr, "cat: ne peut ouvrir %s\n", *argv);
        exit(1);
      }
      else{
        filecopy(fp, stdout);
        fclose(fp);
      }
    }
  return 0;
}
```

Commande unix cat (suite)

```
void filecopy(FILE* ifp, FILE* ofp){
  int c;
  while((c = getc(ifp)) != EOF)
    putc(c, ofp);
}
```

Lecture/écriture formatée

Entrée/sortie standard : `printf` et `scanf`.

En direction d'un fichier : `fprintf` et `fscanf`.

En direction d'une chaîne de caractères : `sprintf` et `sscanf`.

Lecture/écriture formatée

Entrée/sortie standard : `printf` et `scanf`.

En direction d'un fichier : `fprintf` et `fscanf`.

En direction d'une chaîne de caractères : `sprintf` et `sscanf`.

La chaîne de caractères passée en argument (appelée **format**) de ces fonctions réalise des conversions désignées par le symbole %.

Règles pour les formats

Pour la lecture, les formats peuvent contenir

- des caractères blancs (espace, tabulation) : tout caractère blanc lu est ignoré ;
- des spécifications de conversion commençant par le caractère `%` ;
- les autres caractères doivent être rencontrés à la lecture.

Règles pour les formats

Pour la lecture, les formats peuvent contenir

- des caractères blancs (espace, tabulation) : tout caractère blanc lu est ignoré ;
- des spécifications de conversion commençant par le caractère `%` ;
- les autres caractères doivent être rencontrés à la lecture.

Pour l'écriture, les formats peuvent contenir

- des caractères qui sont recopiés dans la chaîne engendrée par l'écriture ;
- des spécifications de conversion commençant par le caractère `%`.

Exemples

```
scanf("%d%c", &e, &c);
```

est différent de

```
scanf("%d %c", &e, &c);
```

Exemples

```
scanf("%d%c", &e, &c);
```

est différent de

```
scanf("%d %c", &e, &c);
```

Soit `str` une chaîne de caractères :

```
printf(str);
```

est différent de

```
printf("%s", str);
```

Problème spécifique à scanf

```
scanf("%s", str)
```

Cette instruction récupère le premier mot de l'entrée standard.
Le reste de `stdin` dans le buffer reste.

Problème spécifique à scanf

```
scanf("%s", str)
```

Cette instruction récupère le premier mot de l'entrée standard.
Le reste de `stdin` dans le buffer reste.

```
void videBuffer(void){
    int c;
    do {
        c = getchar();
    } while (c != '\n' && c != EOF);
}
```

Fonction fflush

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

Vide le buffer associé au flux de sortie (écriture) `stream`.

Fonction fflush

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

Vide le buffer associé au flux de sortie (écriture) `stream`.

Ne fonctionne pas pour les flux d'entrée.

```
printf("phrase sans retour a la ligne");
fflush(stdout);
```

Autres problèmes spécifiques à scanf

La chaîne de caractère lue par `scanf` peut être trop longue et dépasser en mémoire.
Autant que possible, utiliser `fgets`.

Autres problèmes spécifiques à scanf

La chaîne de caractère lue par `scanf` peut être trop longue et dépasser en mémoire.
Autant que possible, utiliser `fgets`.

L'utilisateur peut rentrer des valeurs erronées (caractère au lieu d'entier par exemple).
Effectuer un filtre :

```
char c;
do c = getchar();
while(c < '0' || c > '9');
```

Manipulation du curseur de position dans le fichier

```
long ftell(FILE * stream);
```

Renvoie la position courante dans le fichier.

Manipulation du curseur de position dans le fichier

```
long ftell(FILE * stream);
```

Renvoie la position courante dans le fichier.

```
int fseek (FILE * stream, long offset, int whence);
```

Décale le curseur de `offset` positions à partir de `whence` (qui peut prendre les valeurs constantes `SEEK_SET`, `SEEK_CUR` ou `SEEK_END`).

Manipulation du curseur de position dans le fichier

```
long ftell(FILE * stream);
```

Renvoie la position courante dans le fichier.

```
int fseek (FILE * stream, long offset, int whence);
```

Décale le curseur de `offset` positions à partir de `whence` (qui peut prendre les valeurs constantes `SEEK_SET`, `SEEK_CUR` ou `SEEK_END`).

```
void rewind(FILE * stream);
```

Repositionne le curseur au début du fichier.

Lecture/écriture de fichier binaires

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

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

Lit/écrit `nmemb` éléments de taille `size` de ou vers le fichier `stream` et de ou vers l'adresse `ptr`.

Attention : contrairement à la lecture/écriture de fichiers texte, le résultat obtenu dépend du système.