

# INF3135

## Construction et maintenance de logiciels

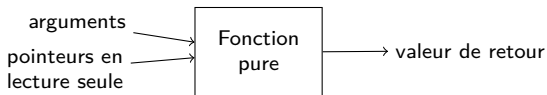
### **Chapitre 4: Entrées et sorties**

Alexandre Blondin Massé

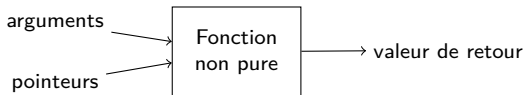
Université du Québec à Montréal  
Département d'informatique

Été 2020

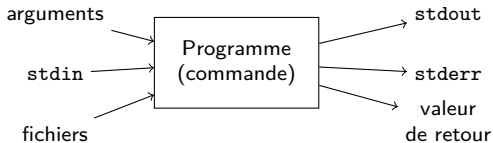
# Entrées et sorties



Entrées



Sorties



# Table des matières

- 1 La bibliothèque `stdio.h`
- 2 Canaux standards
- 3 Valeur de retour
- 4 Quelques commandes utiles

## La bibliothèque `stdio.h`

# La bibliothèque `stdio.h`

- *stdio* = *standard input output*
- Inclusion avec `#include <stdio.h>`

## Macros:

- EOF: caractère de fin de fichier
- `stdin`, `stdout`, `stderr`: canaux standards
- `NULL`: pointeur nul, ...

## Types:

- `FILE`: flux (*stream*)
- `size_t`: taille en octets
- `fpos_t`: position dans un flux, ...

**Variables externes:** `optarg`, `opterr`, `optind`, `optopt` (gestion des arguments)

Plusieurs dizaines de **fonctions** (`man stdio`)

# Opérations générales (1/2)

## Ouverture et fermeture:

```
// Ferme un flux
int      fclose(FILE *);
// Ouvre un flux avec un descripteur de fichier
FILE     *fdopen(int, const char *);
// Ouvre un flux
FILE     *fopen(const char *, const char *);
// Réouvre un flux
FILE     *freopen(const char *, const char *, FILE *);
```

## Suppression et renommage:

```
// Supprime un fichier ou un répertoire
int      remove(const char *);
// Renomme un fichier ou un répertoire
int      rename(const char *, const char *);
```

## Manipulation de fichier temporaires:

```
// Retourne un nom de fichier temporaire
char     *tmpnam(char *);
// Crée un fichier temporaire
FILE     *tmpfile(void);
```

# Opérations générales (2/2)

## Informations générales

```
// Vérifie si la fin du flux est atteinte
int      feof(FILE *);
// Récupère le descripteur d'un flux
int      fileno(FILE *);
// Récupère le nom du terminal courant
char     *ctermid(char *);
// Récupère l'utilisateur courant
char     *cuserid(char *); (LEGACY)
```

## Gestion des erreurs:

```
// Réinitialise les indicateurs de fin de fichier et d'erreur
void     clearerr(FILE *);
// Vérifie si une erreur est survenue
int      ferror(FILE *);
// Écrit un message d'erreur sur stderr
void     perror(const char *);
```

# Manipulation de caractères

```
// Lit le prochain caractère d'un flux
int      fgetc(FILE *);
// Équivalent à fgetc, avec passes multiples du flux
int     getc(FILE *);
// Écrit un caractère sur un flux
int      fputc(int, FILE *);
// Équivalent à fputc, avec passes multiples du flux
int      putc(int, FILE *);
// Remet un caractère sur un flux
int      ungetc(int, FILE *);
```

## Canaux standards:

```
// Équivalent à getc(stdin)
int      getchar(void);
// Écrit un caractère sur stdout
int      putchar(int);
```



# Manipulation d'octets (*binary stream*)

```
// Lit des octets provenant d'un flux
size_t  fread(void *, size_t, size_t, FILE *);
// Écrit des octets sur un flux
size_t  fwrite(const void *, size_t, size_t, FILE *);
// Lit un mot (int) provenant d'un flux
int      getw(FILE *);
// Écrit un mot (int) sur un flux
int      putw(int, FILE *);
```

# Manipulation de chaînes de caractères

```
// Lit une ligne provenant d'un flux
char    *fgets(char *, int, FILE *);
// Écrit une chaîne sur un flux
int      fputs(const char *, FILE *);
```

## Canaux standards:

```
// Lit une chaîne sur stdin (obsolète, deprecated)
// Préférer fgets(..., ..., stdin)
char    *gets(char *);
// Écrit une chaîne sur stdout
int      puts(const char *);
```

# Lecture et écriture formatées

```
// Écrit des données formatées sur un flux
int      fprintf(FILE *, const char *, ...);
// Lit des données formatées provenant d'un flux
int      fscanf(FILE *, const char *, ...);
// Équivalent à fprintf avec va_list
int      vfprintf(FILE *, const char *, va_list);
// Équivalent fprintf, mais sur stdout
int      printf(const char *, ...);
// Équivalent à fscanf, mais provenant de stdin
int      scanf(const char *, ...);
// Écrit des données formatées dans une chaîne
int      sprintf(char *, const char *, ...);
// Équivalent à sprintf, mais avec écriture tronquée
int      snprintf(char *, size_t, const char *, ...);
// Lit des données formatées depuis une chaîne
int      sscanf(const char *, const char *, int ...);
// Équivalent à printf, mais avec a va_list
int      vprintf(const char *, va_list);
// Équivalent à snprintf, mais avec va_list
int      vsnprintf(char *, size_t, const char *, va_list);
// Équivalent à sprintf, mais avec va_list
int      vsprintf(char *, const char *, va_list);
```

# Navigation

```
// Modifie la position courante dans un flux
int      fseek(FILE *, long int, int);
// Modifie la position courante dans un flux (avec off_t)
int      fseeko(FILE *, off_t, int);
// Récupère la position courante dans un flux
long int ftell(FILE *);
// Récupère la position courante dans un flux (avec off_t)
off_t     ftello(FILE *);
// Modifie la position courante dans un flux
int      fsetpos(FILE *, const fpos_t *);
// Récupère la position courante dans un flux
int      fgetpos(FILE *, fpos_t *);
// Remet la position courante d'un flux au début
void      rewind(FILE *);
```

## Autres fonctions (1/2)

### Traitement des arguments:

```
// Traite les options d'une commande
int      getopt(int, char * const[], const char); (LEGACY)
```

### Tampons (*buffer*):

```
// Vide un tampon
int      fflush(FILE *);
// Modifie la stratégie de tampon
// Possibilités: line buffered, unbuffered, fully buffered
int      setvbuf(FILE *, char *, int, size_t);
// Modifie la stratégie de tampon à unbuffered ou fully buffered
void     setbuf(FILE *, char *);
```

### Tubes (communication inter-processus):

```
// Ferme un tube
int      pclose(FILE *);
// Ouvre un tube
FILE     *popen(const char *, const char *);
```

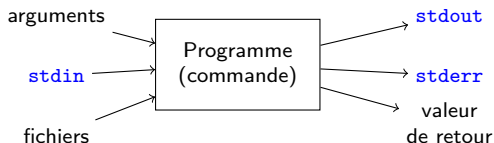
## Autres fonctions (2/2)

### Verrouillage:

```
// Verrouille un flux (thread-safety)
void      flockfile(FILE *);
// Version non bloquante de verrouillage
int       ftrylockfile(FILE *);
// Déverrouille un flux
void      funlockfile(FILE *);
// Lit un caractère provenant d'un flux (non sécuritaire)
int       getc_unlocked(FILE *);
// Écrit un caractère (non sécuritaire)
int       putc_unlocked(int, FILE *);
// Équivalent à getc(stdin) (non sécuritaire)
int       getchar_unlocked(void);
// Écrit un caractère sur stdout (non sécuritaire)
int       putchar_unlocked(int);
```

## Canaux standards

# Canaux



## 3 canaux standards

- `stdin`: entrée standard (canal 0)
- `stdout`: sortie standard (canal 1)
- `stderr`: sortie d'erreur standard (canal 2)

## Comportement par défaut (peut être redéfini)

- `stdin`: saisie clavier (*line buffered*)
- `stdout`: affichage sur le terminal (*line buffered*)
- `stderr`: affichage sur le terminal (*unbuffered*)



# L'entrée standard (stdin)

- Un programme peut lire des données sur l'**entrée standard**
- Par défaut, lit la **saisie clavier**

```
#include <stdio.h>

#define BUFFER_SIZE 20

int main(void) {
    char c1 = getchar(), c2 = getchar(), c3 = getchar();
    char line[BUFFER_SIZE];
    fgets(line, BUFFER_SIZE, stdin);
    printf("3 premiers caractères: %c %c %c\n", c1, c2, c3);
    printf("Reste de ligne: %s\n", line);
    return 0;
}
```

## Résultat:

```
$ gcc stdin.c -o stdin && ./stdin
Git est un super logiciel!
3 premiers caractères: G i t
Reste de ligne:  est un super logic
```

# La sortie standard (stdout)

- Un programme peut écrire des données sur la **sortie standard**
- Par défaut, l'affichage se fait sur le **terminal**

```
#include <stdio.h>

int main(void) {
    char c = 'A';
    putchar(c);
    putchar('\n');
    puts("C est un langage particulier");
    fwrite("Incroyable!", sizeof(char), 4, stdout);
    printf("\n%s %c %p %lf\n",
           "Fantastique", '!', &c, 1.23456789);
    return 0;
}
```

## Résultat:

```
$ gcc stdout.c -o stdout && ./stdout
A
C est un langage particulier
Incr
Fantastique ! 0x7ffe6e743c27 1.234568
```

# La sortie d'erreur standard (stderr)

- On peut aussi écrire sur la **sortie d'erreur standard**
- Par défaut, l'affichage se fait aussi sur le **terminal**

```
#include <stdio.h>

int main(void) {
    char c = 'A';
    fputc(c, stderr);
    fputc('\n', stderr);
    fputs("C est un langage particulier", stderr);
    fwrite("Incroyable!", sizeof(char), 4, stderr);
    fprintf(stderr, "\n%s %c %p %lf\n",
            "Fantastique", '!', &c, 1.23456789);
    return 0;
}
```

## Résultat:

```
$ gcc stderr.c -o stderr && ./stderr
A
C est un langage particulierIncr
Fantastique ! 0x7ffd48babbd7 1.234568
```

## Redirections (1/2)

- Par défaut, `stdin` lit la saisie clavier
- Et `stdout`/`stderr` écrivent sur le terminal
- Ces comportements peuvent être modifiés avec des **redirections**
- Les redirections sont gérées par le **shell**
- Elles ne sont donc **pas gérées** par `argc` et `argv`

### Syntaxe

- `commande < fichier`: redirige fichier sur `stdin`
- `commande > fichier`: redirige `stdout` dans fichier
- `commande 2> fichier`: redirige `stderr` dans fichier

## Redirections (2/2)

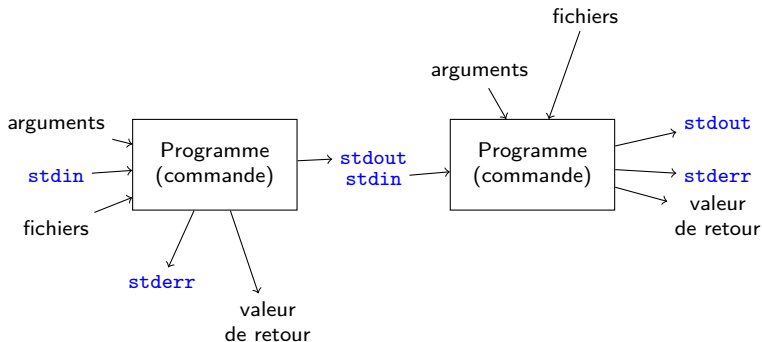
```
#include <stdio.h>

int main(void) {
    char c = getchar();
    if (c == 'y')
        printf("Yes!\n");
    else
        fprintf(stderr, "No!\n");
    return 0;
}

$ cat redirections.y
yoyo
yaourt
$ cat redirections.z
zèbre
zoo
$ gcc redirections.c -o redirections
$ ./redirections < redirections.y
Yes!
$ ./redirections < redirections.z
No!
$ ./redirections < redirections.z 2> /dev/null
```

# Tubes

- Permet d'**enchaîner** des programmes
- Le contenu écrit sur stdout par la première commande
- Est lu sur stdin par la deuxième commande
- **Syntaxe:** commande1 | commande2 | ... | commandeN



## Filtres utiles

**Filtre:** programme souvent utilisé dans un tube

- `sort`: trie les lignes d'un flux
- `uniq`: supprime les doublons consécutifs
- `grep`: filtre selon une expression régulière
- `fmt`: formate des données
- `pr`: formate du texte pour impression
- `head`: affiche les premières lignes d'un flux
- `tail`: affiche les dernières lignes d'un flux
- `tr`: traduit caractère par caractère
- `sed`: transforme du texte
- `awk`: transforme du texte

Plus de détails dans **INF1070**

Consulter le manuel (`man`)

# Exemple de filtres

## Fichier maj.c:

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

int main(void) {
    char c;
    while ((c = getchar()) != EOF) {
        putchar(toupper(c));
    }
    return 0;
}
```

```
$ gcc maj.c -o maj
$ head -n 2 maj.c | ./maj
#include <STDIO.H>
#include <CTYPE.H>
$ head -n 2 maj.c | ./maj | tail -n 1
#include <CTYPE.H>
$ grep 'char' maj.c | ./maj
CHAR C;
WHILE ((C = GETCHAR()) != EOF) {
    PUTCHAR(TOUPPER(C));
```



Valeur de retour

# Valeur de retour

- En Unix, tout programme retourne une **valeur entière**
- Lorsque son exécution est **terminée**

## Sémantique

- 0: le programme s'est terminé « **normalement** »
- $\neq 0$ : le programme s'est terminé « **anormalement** »

## Récupérer la valeur de retour

- Contenue dans la **variable spéciale** \$?
- Valeur de retour de la **dernière commande**

# Valeur de retour en C

## Fonction main

- Valeur retournée à l'aide de `return`
- Doit être **entière**
- Peut être **négative**
- Par défaut, retourne 0
- **Bonne pratique**: toujours spécifier la valeur de retour

## La fonction exit

```
void exit(int status);
```

- Permet de **terminer** l'exécution du programme proprement
- Vide et ferme les **flux** encore ouverts
- Supprime les **fichiers temporaires**

# Combinaisons de commandes

- On peut **combinaisonner** des commandes avec ; && et ||
- Le comportement dépend de la **valeur de retour**
- ;: deux commandes consécutives indépendantes
- &&: 2e commande exécutée seulement si la 1re réussit
- ||: 2e commande exécutée seulement si la 1re échoue

```
$ echo "commande" && echo $?
```

```
commande
```

```
0
```

```
$ echo "commande" || echo $?
```

```
commande
```

```
$ cat fichier.inexistant && echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
$ cat fichier.inexistant ; echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
1
```

```
$ cat fichier.inexistant || echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
1
```

## Quelques commandes utiles

# La suite Graphviz

- **Bibliothèques** et **programmes**
- Permettant de générer des **graphes**
- **Site officiel**: <https://graphviz.org/>
- **Dépôt Git**: [sur GitLab](#)
- **Licence**: [Common Public License](#)

## Le format DOT

- Permet de décrire un graphe
- **Attributs**: noeuds, arcs, graphe, sous-graphe
- **Type d'attributs**: forme, couleur, police, espacement, ...

## Utilisation

- En ligne de commande: `dot`, `neato`, `circo`
- Sous forme de bibliothèque C: `#include <gvc.h>`

# Le format DOT

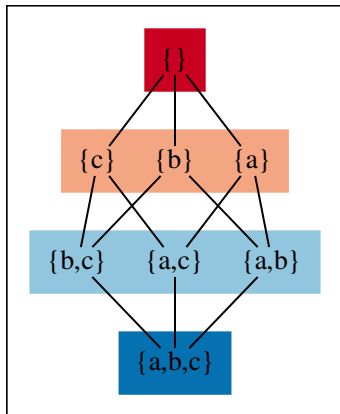
```
// Fichier graphviz.dot
digraph G {
    // Style des sous-graphes
    style=filled; labeljust=l; colorscheme=rdbu4;
    // Style des noeuds et des flèches
    node [shape=plain]; edge [arrowhead=none];
    // Sous-graphes
    subgraph cluster_0 {
        e [label="{ }"];
        color=1;
    }
    subgraph cluster_1 {
        a [label="{a}"]; b [label="{b}"]; c [label="{c}"];
        color=2;
    }
    subgraph cluster_2 {
        ab [label="{a,b}"]; ac [label="{a,c}"]; bc [label="{b,c}"];
        color=3;
    }
    subgraph cluster_3 {
        abc [label="{a,b,c}"];
        color=4;
    }
    // Autres relations
    e -> {a b c};
    a -> ab; a -> ac;
    b -> ab; b -> bc;
    c -> ac; c -> bc;
    {ab ac bc} -> abc;
}
```

# Invocation et résultat

## Invocation:

```
$ dot -Tpdf -o graphviz.pdf < graphviz.dot
```

## Résultat:





# Le programme QPDF

- Programme en **ligne de commande**
- Permettant de manipuler des **documents pdf**: concaténation, extraction de pages, affichage d'information, rotation, chiffrement, ...
- **Site officiel**: <http://qpdf.sourceforge.net/>
- **Dépôt Git**: [sur Github](#)
- **Licence**: [Apache](#)

```
# Nombre de pages
$ qpdf --show-npages a.pdf
4
$ qpdf --show-npages b.pdf
7
# Concatène a.pdf et b.pdf et place le résultat dans o.pdf
$ qpdf --empty --pages a.pdf b.pdf -- o.pdf
$ ls
a.pdf  b.pdf  o.pdf
$ qpdf --show-npages o.pdf
11
# Extrait les pages 1, 3 et 4 de a.pdf
$ qpdf --empty --pages a.pdf 1,3-4 -- n.pdf
$ qpdf --show-npages n.pdf
3
```

# La suite ImageMagick

- Permet de manipuler des **images matricielles** (*bitmap*)
- Peut aussi convertir en **format vectoriel** (PDF, SVG, ...)
- **Site officiel**: <https://imagemagick.org/index.php>
- **Dépôt Git**: [sur Github](#)
- Licence: [personnalisée](#), de type *copyleft*

## Plusieurs opérations

- Extraction d'**informations**: dimensions, format, ...
- **Transformations**: rognage, rotations, changements d'échelle, ...
- Application de **filtres**: flou, colorisation, convolutions, ...
- **Combinaison** d'images: concaténation, superposition, ...

## Utilisation

- En ligne de commande: `magick`, `convert`, `montage`, `mogrify`, ...
- **API** pour plusieurs langages de programmation

# Exemples

**# Redimensionne une image**

```
$ mogrify -resize 50% photo.jpg
```

**# Transforme une image couleur en niveau de gris**

```
$ convert image.png -set colorspace Gray -separate \  
> -average result.png
```

**# Produit une animation GIF (0.2 seconde par image)**

```
$ convert -delay 20 -loop 0 image*.gif animation.gif
```

**# Produit une image avec le mot ImageMagick**

```
$ convert -size 300x60 xc:skyblue -fill white -stroke black \  
> -pointsize 40 -gravity center \  
> -draw "text 0,0 'ImageMagick'" text.png
```

**# Combine plusieurs images dans une matrice (spritesheet)**

```
$ montage walking*.png -tile 4x2 -geometry 128x128+0+0 \  
> -background transparent walking-spritesheet.png
```

# La suite ffmpeg

- Permet de manipuler des fichiers **audio** et **vidéo**
- **Site officiel:** <https://ffmpeg.org/>
- **Dépôt Git:** plusieurs dépôts
- Licence: [de type GPL](#)

```
# Convertit un fichier MKV au format MP4
```

```
$ ffmpeg -i input.mkv -codec copy output.mp4
```

```
# Affiche la durée d'un vidéo au format H:M:S:MS
```

```
$ ffprobe -i input.mp4 -sexagesimal -show_entries format=duration \
> -v quiet -of csv="p=0"
0:15:14.400000
```

```
# Supprimer les 2 premières secondes d'un vidéo
```

```
$ ffmpeg -i input.mkv -ss 2 copy output.mkv
```

```
# Concatène 3 vidéos MP4 en un seul
```

```
$ ffmpeg -i "concat:input1.mp4|input2.mp4|input3.mp4" \
> -c copy output.mp4
```

# Le programme Gnuplot

- Permet de générer des **graphiques** statistiques
- Plusieurs **formats** de sortie supportés: PNG, PDF, SVG, ...
- **Site officiel**: <http://www.gnuplot.info/>
- **Dépôt Git**: [sur Github](#)
- Licence: [personnalisée](#), permissive, mais limites sur la distribution

## Plusieurs types de graphiques

- Histogrammes, lignes brisées, diagrammes à secteurs, ...
- Tracé de fonctions, de fonction paramétrées, fonction implicites
- Courbes dans l'espace, surfaces, ...

## Utilisation

- De façon interactive en lançant `gnuplot`
- À l'aide d'un **script**

# Exemple

## Fichier gnuplot.gp:

```
# Sélectionne le format de sortie
set terminal png
# Supprime la légende
set nokey
# Spécifie le titre
set title "Taille de fichier en octets"
# Rotation de 90 degrés des noms de fichier
set xtics rotate
# Spécifie le style de graphique (histogramme)
set style data histograms
# Lit les données sur stdin
# La colonne 1 en abscisse et la colonne 2 en ordonnée
plot '/dev/stdin' using 2:xtic(1)
```

## Invocation:

```
# La commande stat affiche des statistiques sur des fichiers
#      %n: nom du fichier, %s: taille du fichier en octets
#      [ms]*.c: tous les fichiers commençant par m ou s
#              et finissant par .c
$ stat -c '%n %s' [ms]*.c | gnuplot gnuplot.gp > histogram.png
```

# Résultat

Taille de fichier en octets

