Chapitre 5 : Entrées et sorties

Guy Francoeur

basé sur les travaux d'
Alexandre Blondin Massé, professeur

Département d'informatique
Université du Québec à Montréal

6 janvier 2019
Construction et maintenance de logiciels
INF3135

Table des matières

Table des matières

Caractère par caractère

La fonction int getchar(void) :

- Retourne le prochain caractère lu sur l'**entrée standard**;
- Retourne la valeur EOF (pour "end of file") si la lecture est **terminée**; La valeur EOF est également retournée lorsque le caractère CTRL-D est saisi au clavier.
- Notez que le type de **retour** est int : permet de traiter le code ASCII étendu.

La fonction int putchar(int c) :

- Ajoute un caractère sur la **sortie standard**.

La fonction int ungetc(int c, stdin) :

- Ajoute un caractère sur l'**entrée standard**.

Exemple

```c
//ex12.c
#include <stdio.h>

int main() {
    char c;

    while ((c = getchar()) != '\n') {
        putchar(toupper(c));
    }
    return 0;
}
```

Entrée : bonjour
Sortie : BONJOUR

Manipulation des entrées/sorties : ligne par ligne

La fonction char *gets(char *ligne) :

- Retourne la prochaine ligne lue sur l'**entrée standard**;
- Supprime le caractère \n en fin de ligne et ajoute le caractère \0 en fin de chaîne;
- Retourne NULL lorsque le caractère EOF est rencontré;
- **Aucun contrôle** sur la taille de la ligne lue.

La fonction int puts(const char *ligne) :

- Ajoute une ligne sur la **sortie standard**.

Exemple

```c
#include <stdio.h>

const unsigned int MAX_LIGNE = 20;

int main() {
    char ligne[MAX_LIGNE];
    int i = 0;
    while (gets(ligne) != NULL) {
        printf("%d : %s\n", i++, ligne);
    }
    return 0;
}
```

Résultat :

```
warning: this program uses gets(), which is unsafe.
Bonjour !
0 : Bonjour !
Comment allez-vous ?
1 : Comment allez-vous ?
Très bien.
2 : Très bien.
```

Saisie d'une ligne sécurisée

- Pour prévenir un **débordement**, on utilise la fonction fgets.

```
#include <stdio.h>

int main() {
    char ligne[10];
    fgets(ligne, 10, stdin);
    printf("Ligne : /%s/", ligne);
    return 0;
}
```

Entrée : Croissants et pâtisseries
Sortie : Ligne : /Croissant/

Formatage

- La fonction int printf(char *format, ...) permet d'afficher sur la **sortie standard** un texte **formaté**;
- La fonction int sprintf(char *chaine, char *format, ...) permet d'envoyer un **texte formaté** dans une **chaîne**;
- **Attention !** Assurez-vous que l'espace mémoire pointé par *chaine soit réservé.
- La variable format décrit la structure selon laquelle les éléments sont affichés et quel est le **type** de ces éléments.
- On utilise le symbole % pour indiquer les différents **code de formatage**.

Formatage des types de base

Code	Description
%c	Affichage d'un caractère
%d	Affichage d'un entier sous forme décimale
%hd	Affichage d'un entier court sous forme décimale
%ld	Affichage d'un entier long sous forme décimale
%u	Affichage d'un entier non signé
%o	Affichage d'un entier sous forme octale
%x	Affichage d'un entier sous forme hexadécimale
%e	Affichage d'un flottant en notation scientifique
%f	Affichage d'un flottant en notation décimale
%g	Affichage d'un flottant de façon compacte
%lf	Affichage d'un double en notation décimale
%L	Affichage d'un long double en notation décimale
%s	Affichage d'une chaîne de caractères
%p	Affichage d'un pointeur

Codes de formatage optionnels

Code	Description
%-	Alignement à gauche (par défaut à droite)
%+	Ajoute le symbole + aux nombres positifs
%	Ajoute un espace aux nombres positifs
%#	Ajoute un préfixe 0 ou 0X si octal ou hexadécimal
%8d	Affichage sous forme décimale de largeur au moins 8
%.4f	Affichage sous forme décimale avec quatre chiffres après la virgule.
	Remplissage avec espace ou 0 si alignement droit

Exemple

```c
#include <stdio.h>
#include <math.h>

int main() {
    printf("i    sqrt(i)   cos(i)\n");
    printf("-----------------------\n");
    for (int i = 8; i < 10000; i *= 2) {
        printf("%4.4d %-8.4f %8.4f\n", i, sqrt(i), cos(i));
    }
  return 0;
}
```

```
i    sqrt(i)   cos(i)
---------------------
0008 2.8284    -0.1455
0016 4.0000    -0.9577
0032 5.6569     0.8342
0064 8.0000     0.3919
0128 11.3137   -0.6929
0256 16.0000   -0.0398
0512 22.6274   -0.9968
1024 32.0000    0.9874
2048 45.2548    0.9497
4096 64.0000    0.8040
8192 90.5097    0.2928
```

Entrées formatées

- La fonction int scanf(char *format, ...) permet de lire une chaîne de caractères **formatées** sur l'**entrée standard**;
- La fonction int sscanf(char *chaine, char *format, ...) permet de lire du **texte formaté** d'une chaîne;

```
#include <stdio.h>

int main() {
    double somme, valeur;

    somme = 0.0;
    while (scanf("%lf", &valeur)) {
        somme += valeur;
        printf("Total : %.2f\n", somme);
    }
    return 0;
}
```

Exemple de calculatrice simple

Sortie :

```
34
Total : 34.00
28.5
Total : 62.50
10.1
Total : 72.60
fini
```

Sortie :

```
1.23456789012345678900
Total : 1.23
2.4
Total : 3.63
-0.1
Total : 3.53
^D
```

Extraction de données formatées

```c
#include <stdio.h>

int main() {
    char nom[30], prenom[30], ligne[60];
    int naissance;

    while (fgets(ligne, 30, stdin) &&
           sscanf(ligne, "%s %s %d", nom, prenom,
           &naissance) == 3) {
        printf("Nom                    : %s\n", nom);
        printf("Prenom                 : %s\n", prenom);
        printf("Date de naissance      : %d\n", naissance);
    }
    return 0;
}
```

Sortie

```
Jean Cote 1978
Nom             : Jean
Prenom          : Cote
Date de naissance : 1978
Victor Hugo 1802
Nom             : Victor
Prenom          : Hugo
Date de naissance : 1802
Nelson Mandela 1918
Nom             : Nelson
Prenom          : Mandela
Date de naissance : 1918
ok
```

Entrées et sorties

- Faire attention au **débordement**;
- Lors de l'utilisation de la fonction fgets, ne pas oublier que les caractères supplémentaires sont encore dans l'entrée standard stdin;

```c
#include <stdio.h>

int main() {
    char ligne[10];
    fgets(ligne, 10, stdin);
    printf("%s\n", ligne);
    printf("%c", getchar());
    return 0;
}
```

Sortie :

```
abcdefghijklmnopqr
abcdefghi
j
```

Table des matières

Manipulation de fichiers

- Toutes les fonctions interagissant avec stdin et stdout s'étendent naturellement aux **fichiers**;
- On distingue **deux types** de fichiers :
  - Les fichiers **textes**;
  - Les fichiers **binaires**.
- Il existe un type fichier FILE prédéfini dans la bibliothèque stdio.h;

Ouverture et fermeture

- La fonction

```
FILE *fopen(const char *nomFichier, const char *mode)
```

  permet d'**ouvrir** un fichier selon un **mode** donné;

- Remarquez que c'est un **pointeur** vers un fichier qui est retourné;
- La valeur NULL est retournée si un **problème survient**;
- La fonction

```
int fclose(FILE *fichier)
```

  permet de **fermer** un fichier;

- Une valeur de 0 est retournée si tout se déroule **bien**.

Mode d'ouverture

Mode	Description
"r"	Ouvre un fichier en mode lecture. Le fichier doit exister.
"w"	Ouvre un fichier vide en mode écriture. Si un fichier avec le même nom existe, il est écrasé.
"a"	Ouvre un fichier vide en mode "append". Si le fichier n'existe pas, alors il est créé.
"r+"	Ouvre un fichier en mode lecture et écriture. Le fichier doit exister.
"w+"	Ouvre un fichier vide en mode lecture et écriture. Si un fichier avec le même nom existe, il est écrasé.
"a+"	Ouvre un fichier vide en mode lecture et "append". Si le fichier n'existe pas, alors il est créé.

L'ajout du caractère b en suffixe pour les modes d'ouvertures spécifie que le fichier est **binaire** plutôt que **texte**.

Lecture et écriture formatées

Des analogues des fonctions printf, scanf, etc. existent pour les fichiers :

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

int fputc(int c, FILE *fichier)

int fputs(const char *s, FILE *fichier)

int fscanf(FILE *fichier, const char *format, ...)

int fgetc(FILE *fichier)

char *fgets(char *s, int n, FILE *fichier)
```

Accès direct

- Lorsqu'un fichier est ouvert, on lui associe un **curseur** qui se déplace lors de l'écriture et de la lecture de données;
- Il est possible d'accéder directement à une **adresse donnée**, au ***i*-ème octet** du fichier;
- La fonction

```
long int ftell(FILE *fichier);
```

  donne la **position courante** du curseur par rapport au début du fichier **en octets**;
- La fonction

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

  positionne le curseur en **début de fichier**.

Positionnement du curseur

- La fonction

```
int fseek(FILE *fichier, long int decalage, int début);
```

  positionne le **curseur courant** à la position début + décalage. La fonction retourne 0 si le positionnement se déroule bien, une autre valeur sinon.

- En général, pour le paramètre début, on utilise des constantes retenant des positions de base dans un fichier :
  - SEEK_SET correspond au début du fichier;
  - SEEK_CUR correspond à la position actuelle du curseur;
  - SEEK_END correspond à la fin du fichier.

Table des matières

Trois canaux

- L'**entrée standard** (stdin, canal 0). Par défaut, elle correspond aux saisies **clavier**;
- La **sortie standard** (stdout, canal 1). Par défaut, elle correspond à l'affichage au **terminal**;
- Le **canal d'erreur** (stderr, canal 2). Par défaut, s'affiche également au **terminal**;
- Ces entrées et sorties peuvent être **redirigées** (dans des fichiers par exemple) grâce aux opérateurs Unix $<$, $<<$, $>$, etc.

Canaux standards en C

- Les trois canaux standards stdin, stdout et stderr sont des **fichiers prédéfinis**;
- Il sont **ouverts** par défaut;
- Pas besoin de les **fermer** à la fin d'un programme non plus;
- Pour afficher sur le **canal d'erreur** :

```
fprintf(stderr, "Erreur %d : %s", noErreur, msgErreur);
```

- La fonction fgets peut également être utilisée pour éviter les **débordements** :

```
char ligne[TAILLE_LIGNE];
fgets(ligne, TAILLE_LIGNE, stdin);
```

Redirections

- On peut utiliser les **redirections** :

```
$ ls -lhs > output.txt        # liste les fichiers dans output.txt
$ ./tp1 -c FRAG > out.txt  # redirige la sortie standard dans out.txt
$ ./tp2 2> err.log     # Les erreurs stderr sont redirigées dans err.log
```

- Pour rendre un processus **complètement silencieux** :

```
$ python process.py &> /dev/null
```

- Pour **sauvegarder** ce qui est affiché :

```
$ make > compilation.log # Rien n'est affiche, tout est redirige
$ make | tee compilation.log   # Affiche et redirige le résultat
```

Tubes (*pipes*)

- Le caractère | est utilisé pour créer un **tube**;
- Un des aspects les plus **importants** des commandes Unix est qu'elles peuvent être combinées à l'aide de **tubes** (*pipes*);
- Essentiellement, un **tube** permet de récupérer la **sortie** (sur **stdout**) d'un processus et de la passer en **entrée** (sur **stdin**) à un autre processus.
- Le caractère utilisé est la barre verticale |;
- Téléchargement et traitement :

```
$ wget https://github.com/guyfrancoeur/tp1/data.zip |
    unzip -d ./data
```

Table des matières

Rappel

- La **syntaxe** générale d'un Makefile est

  <cible>: <dépendances>
  <tab><commande>

- Par exemple :

  ```
  tp1: tp1.o
      gcc -o tp1 tp1.o
  ```

  indique comment générer l'**exécutable** tp1 à partir du fichier **objet binaire** tp1.o.

Utilisation de variables

- Une première **amélioration** qu'on peut apporter à un Makefile est d'employer des **variables** :
- Une variable est **initialisée** en écrivant simplement

  <nom variable> = <valeur>

- On récupère ensuite son **contenu** en utilisant l'opérateur $:

  $(<nom variable>)

Exemple

```
CC = gcc
CFLAGS = -Wall
SRC = tp1.c
OBJ = tp1.o
EXEC = tp1

$(EXEC): $(OBJ)
	$(CC) $(CFLAGS) -o $(EXEC) $(OBJ)

$(OBJ): $(SRC)
	$(CC) $(CFLAGS) -c $(SRC)
```

Variante :

```
FILENAME = tp1
EXEC = tp1

$(EXEC): $(FILENAME).o
	$(CC) $(CFLAGS) -o $(EXEC) $(FILENAME).o

$(FILENAME).o: $(FILENAME).c
	$(CC) $(CFLAGS) -c $(FILENAME).c
```

Cibles quelconques

- Il est également possible de définir des **cibles** qui ne sont pas des noms de fichier;
- Quelques cibles classiques :
  - clean : pour nettoyer le répertoire du projet;
  - all : pour générer l'ensemble du projet;
  - install : pour installer le projet sur la machine;
  - test : pour lancer une suite de tests;
  - doc : pour générer la documentation, etc.

La cible .PHONY

- Lorsqu'on utilise des **cibles** qui ne sont pas des noms de fichier, il est préférable de les **déclarer**;
- Ceci se fait à l'aide de la cible .PHONY;
- **Exemple :**

```
FILENAME = tp1
EXEC = tp1
CC = gcc
CFLAGS = -Wall

$(EXEC): $(FILENAME).o
	$(CC) $(CFLAGS) -o $(EXEC) $(FILENAME).o

$(FILENAME).o: $(FILENAME).c
	$(CC) $(CFLAGS) -c $(FILENAME).c

.PHONY: clean

clean:
	rm -f $(EXEC) $(FILENAME).o
```

Table des matières

Utilitaires Linux

Il existe de nombreux programmes UNIX très **pratiques** :

- **grep** : recherche de motifs;
- **tr** : remplace/supprime des caractères;
- **sed** : modifie en fonction d'expressions régulières;
- **awk** : traitement de fichiers textes;
- **echo** : affiche une chaine vers **stdout**;
- **tree** : affiche la structure et les fichiers;
- **find** : recherche l'existence d'un fichier dans le système;
- **time** : affiche des statistiques sur l'exécution du processus;
- **latex** : produit des documents pdf, etc.

Le programme grep

- Identifie des **motifs** dans un texte;
- Basé sur les **expressions régulières**;
- Par défaut, affiche les **lignes** dans lesquelles le motif apparaît;
- Permet de faire une **recherche rapide** dans un projet;
- Documentation : man grep.
- Très utile lorsque **combiné** à d'autres programmes.

Exemple

- Supposons que je souhaite avoir une idée rapide des différents **statuts** de vos travaux pratiques;
- Alors je peux entrer la commande suivante :

  $ grep -R 'Statut' -A5 --color -h *

- -R signifie que la recherche est **récursive** dans les sous-dossiers;
- -A5 signifie que les **5 lignes** suivantes sont affichées pour chaque correspondance;
- –color pour colorier le motif trouvé;
- -h n'affiche pas les noms de fichiers (pour garder l'anonymat.

Le programme tr

- Permet de **traduire** ou **supprimer** certains caractères;
- Spécialisé dans le traitement **caractère** par **caractère**;
- Utile pour résoudre des problèmes d'**encodages** de fichier;
- Syntaxe générale :

  tr [OPTION] SET1 [SET2]

- Convertit les minuscules en majuscules :

  tr [:lower:] [:upper:]

- Remplace les espaces par des tabulations :

  tr [:space:] '\t'

Le programme sed

- À l'instar de grep, il est basé sur les expressions **régulières**;
- Permet de faire des modifications de type **"find and replace"**;
- Remplacer les **tabulations** par des **espaces** :

  sed $'s/\t/   /g' fichier.txt

  - $ signifie que la syntaxe Bash doit être utilisée (sinon la tabulation n'est pas reconnue);
  - s signifie qu'on fait une **substitution**;
  - / est un séparateur;
  - **g** signifie qu'on fait la substitution **globalement**.

LaTeX

- Permet de produire des **documents** structurés;
- Principe WYSIWYM;
- Même idée que le format **Markdown**;
- Séparation de la **forme** et du **contenu**;
- Par opposition au principe WYSIWYG : Word, LibreOffice, Pages, etc.;
- Permet de produire des **affiches**, des **diapositives**, des **rapports**, des **examens**, etc.
- Format produit : généralement **pdf**.
- Utile pour la production **automatique** de **rapports**, de **documentation**, etc.