Introduction à la Programmation des Algorithmes

4.2. Langage C – stdio.h

François Fleuret

https://fleuret.org/11x001/

UNIVERSITÉ
DE GENÈVE

Nous avons utilisé dans la plupart de nos exemples la fonction `printf` de la librairie `stdio.h`, mais cette librairie offre d'autres fonctions, en particulier pour

- formater des entrées et des sorties,
- manipuler des fichiers (lecture / écriture).

scanf

Nous avons vu comment formater l'affichage de quantités calculées par un programme.

Il est souvent nécessaire de faire une opération inverse pour interpréter des caractères saisis par l'utilisateur ou venant d'un fichier comme des quantités numériques.

L'affichage se fait avec

```
printf(format, var1, var2, ...)
```

La fonction standard du C pour saisir des valeurs dans la console, aussi définie dans `stdio.h`, est

```
scanf(format, adr_var1, adr_var2, ...)
```

Elle prend comme argument une chaîne de caractères qui définit le format à interpréter, avec des spécifications de types préfixées par % similaires à celles de `printf`, suivie des **adresses** des variables où ces valeurs devront être stockées.

Par exemple pour saisir un nombre entier et afficher tout ses diviseurs:

```c
#include <stdio.h>

int main () {
  int n;

  printf("Entrez un entier: ");
  scanf("%d", &n);

  for(int k = 1; k <= n; k++)
    if(n % k == 0) printf("%d\n", k);

  return(0);
}
```

```
~/sources/11x001 ./a.out
Entrez un entier: 30
1
2
3
5
6
10
15
30
```

Le format peut être plus complexe et permet par exemple de saisir deux valeurs séparées par un espace:

```c
#include <stdio.h>

int main () {
  float x, y;

  printf("Entrez deux valeurs: ");
  scanf("%f %f", &x, &y);

  printf("%f\n", x * y);

  return(0);
}
```

affiche

```
~/sources/11x001 ./a.out
Entrez deux valeurs: 23 45
1035.000000
```

La fonction `scanf` retourne une valeur entière qui indique combien de valeurs ont été correctement saisies, et -1 si aucun caractère ne l'a été.

Pour ignorer une quantité, on peut ajouter le symbole `*` dans un spécificateur de format. Par exemple

```
scanf("%*d %*d %d", &n);
```

attend trois valeurs entières, ignore les deux premières, et mémorise la troisième dans la variable `n`.

Finalement, `scanf` permet aussi de lire des chaînes de caractères. Il existe plusieurs manières de faire, mais la plus pratique consiste à utiliser `%ms` qui alloue une zone mémoire de la taille nécessaire:

```
1  char *nom;
2  if(scanf("%ms", &nom) == 1) {
3    printf("nom=\"%s\"\n", nom);
4    free(nom);
5  } else {
6    printf("ERREUR!\n");
7  }
```

Il faut libérer la zone allouée avec `free` comme pour une allocation faite avec `malloc`.

Finalement, `scanf` permet aussi de lire des chaînes de caractères. Il existe plusieurs manières de faire, mais la plus pratique consiste à utiliser `%ms` qui alloue une zone mémoire de la taille nécessaire:

```
1  char *nom;
2  if(scanf("%ms", &nom) == 1) {
3    printf("nom=\"%s\"\n", nom);
4    free(nom);
5  } else {
6    printf("ERREUR!\n");
7  }
```

Il faut libérer la zone allouée avec `free` comme pour une allocation faite avec `malloc`.

Cette option ne permet pas de saisir des chaînes contenant des espaces ou certains caractères spéciaux, mais d'autres le permettent.

Les fonctions `printf` et `scanf` sont très complexes, et en particulier les spécificateurs de formats sont nombreux et offrent des options très riches.

Nous n'avons vu qu'une toute petite partie de ce dont elles sont capables.

Pour plus de détails, vous pouvez vous référer aux pages de manuel Linux:

https://man7.org/linux/man-pages/man3/printf.3.html

https://man7.org/linux/man-pages/man3/scanf.3.html

Lecture et écriture de fichiers

La gestion des données persistantes se fait à l'aide de fichiers, chacun constitué d'une suite d'octets, et généralement stocké sur un disque dur (hd pour *hard disk*) ou un disque à semi-conducteur (ssd pour *solid-state drive*).

Les contraintes physiques inhérentes à ces supports font que contrairement à l'accès aux données en mémoire, qui peut se faire directement, celui aux informations stockées dans un fichier se fait de manière sérielle: on doit [la plupart du temps] lire une portion consécutive de données.

La librairie `stdio.h` met à disposition pour lire et écrire dans des fichier:

- La structure de donnée `FILE`, qui contient les informations relatives à un fichier qui est en train d'être manipulé,
- la fonction `fopen` qui "ouvre" un fichier pour lire son contenu ou y écrire,
- la fonction `fclose` qui "ferme" un fichier et indique que l'on a fini de le manipuler,
- la fonction `fgets` pour lire une ligne complète depuis un fichier,
- la fonction `fprintf` qui permet de faire des écriture formatées dans un fichier, et
- la fonction `fscanf` pour lire des données depuis un fichier.

Un programme minimal pour créer un fichier pourrait être:

```
1   #include <stdio.h>
2
3   int main () {
4     FILE *file;
5
6     file = fopen("result.txt", "w");
7     fprintf(file, "Bonjour!\n");
8     fclose(file);
9
10    return(0);
11  }
```

La ligne 6 ouvre le fichier `result.txt` pour écrire dedans, la ligne 7 écrit dans ce fichier la chaîne de caractères “Bonjour!” suivie d’un retour à la ligne, et la ligne 8 ferme le fichier. On ne peut donc plus utiliser `file` après cette ligne.

La fonction pour ouvrir un fichier est

```
fopen(filename, mode)
```

où `filename` est une chaîne de caractères spécifiant le nom du fichier et `mode` est une chaîne de caractères spécifiant comment le fichier sera manipulé. Les trois principaux modes sont:

- `"r"` pour le lire (*read*),
- `"w"` pour écrire (*write*), auquel cas le contenu du fichier est perdu dès que `fopen` est executée, et
- `"a"` pour rajouter du contenu à la fin du fichier (*append*).

La valeur retournée est 0 en cas d'erreur et un pointeur en cas de succès.

La fonction pour fermer un fichier est

```
fclose(stream)
```

où `stream` est un `FILE *` obtenu avec `fopen`.

La fonction pour lire une ligne complète depuis un fichier

```
char *fgets(buffer, taille, stream)
```

où `buffer` est un `char *`, `taille` est le nombre maximum de caractères qui peuvent être stockés là, et `stream` est un `FILE *` obtenu avec `fopen`.

Si une ligne a bien été lue, la valeur retournée est `taille`, et zéro sinon.

Les fonctions

```
fprintf(stream, format, var1, ...)
```

et

```
fscanf(stream, format, adr_var1, ...)
```

sont identiques à `printf` et `scanf`, mais attendent un argument de plus de type `FILE *` qui spécifie sur quel fichier opérer, et qui doit être accessible respectivement en écriture et en lecture.

```c
char buffer[1024];
char *filename = "truc";
FILE *f;

f = fopen(filename, "w");
fprintf(f, "Premiere ligne\n");
fprintf(f, "Deuxieme ligne\n");
fprintf(f, "Troisieme ligne\n");
fclose(f);

f = fopen(filename, "r");
while(fgets(buffer, sizeof(buffer) / sizeof(char), f))
  printf(buffer);
fclose(f);
```

affiche

```
Premiere ligne
Deuxieme ligne
Troisieme ligne
```

```c
char buffer[1024];
char *filename = "truc";
FILE *f;

f = fopen(filename, "w");
fprintf(f, "Premiere ligne\n");
fprintf(f, "Deuxieme ligne\n");
fclose(f);

f = fopen(filename, "w");
fprintf(f, "Troisieme ligne\n");
fclose(f);

f = fopen(filename, "r");
while(fgets(buffer, sizeof(buffer) / sizeof(char), f))
  printf(buffer);
fclose(f);
```

affiche

```
Troisieme ligne
```

```c
char buffer[1024];
char *filename = "truc";
FILE *f;

f = fopen(filename, "w");
fprintf(f, "Premiere ligne\n");
fprintf(f, "Deuxieme ligne\n");
fclose(f);

f = fopen(filename, "a");
fprintf(f, "Troisieme ligne\n");
fclose(f);

f = fopen(filename, "r");
while(fgets(buffer, sizeof(buffer) / sizeof(char), f))
  printf(buffer);
fclose(f);
```

affiche

```
Premiere ligne
Deuxieme ligne
Troisieme ligne
```

Si nous voulons calculer la moyenne et l'écart-type d'une séries de valeurs stockées dans un fichier `valeurs.dat`

```
6.9085
-0.8443
0.9509
3.2033
6.0919
-2.6881
-6.4554
1.6049
4.5141
10.5301
-0.6678
-1.6742
...
```

```c
FILE *fh;

fh = fopen("valeurs.dat", "r");

float x, sum = 0, sum_sq = 0;
int nb = 0;
while(fscanf(fh, "%f", &x) == 1) {
  sum += x;
  sum_sq += x * x;
  nb++;
}

fclose(fh);

float moyenne = sum / nb;
float var = (sum_sq - sum * moyenne) / (nb - 1);
printf("nb=%d moyenne=%f ecart_type=%f\n", nb, moyenne, sqrt(var));
```

affiche

```
nb=100 moyenne=1.907111 ecart_type=3.483990
```

Gestion des erreurs

Comme nous l'avons vu, une fonction peut échouer, auquel cas elle retourne une valeur qui le signal, par exemple -1 pour `scanf` ou 0 pour `fopen`.

Il est en général important de fournir des informations plus précises sur la cause de l'erreur.

Les librairies `errno.h` et `strings.h` permettent d'avoir accès à de telles informations, et en particulier de fournir un message d'erreur compréhensible par un utilisateur.

En cas d'erreur, la plupart des fonctions de la librairie C standard stockent un code d'erreur dans la variable globale `errno` qui est définie dans `errno.h`

Par exemple si le fichier `toto` n'existe pas

```
1  FILE *f;
2
3  f = fopen("toto", "r");
4
5  if(f) {
6    /* ... */
7    fclose(f);
8  } else {
9    printf("errno=%d\n", errno);
10 }
```

affiche

```
errno=2
```

Ces codes d'erreur peuvent être transcrit en erreur intelligible grâce à la fonction `strerror` de `strings.h`.

Le programme suivant essaye de créer un fichier à la "racine", ce qui sous Linux demande des droits d'administrateur

```
1  FILE *f;
2
3  f = fopen("/toto", "w");
4
5  if(f) {
6    /* ... */
7    fclose(f);
8  } else {
9    printf("errno=%d (%s)\n", errno, strerror(errno));
10 }
```

affiche

```
errno=13 (Permission denied)
```

Fin