COURS 7 - ENTRÉES/SORTIES ET FICHIERS

G. Bianchi, G. Blin, A. Bugeau, S. Gueorguieva, R. Uricaru

2015-2016

Programmation 1 - uf-info.ue.prog1@diff.u-bordeaux.fr

université de BORDEAUX

UTILISATION AVANCÉE DE PRINTF

RETOUR DE PRINTF

▷ La fonction **printf** retourne le nombre de caractères écrits sur la sortie standard

```
int main(int argc,char* argv[]) {
    int n=printf("command=%s\n",argv[0]);
    int i;
    for (i=1;i<n;i++) { //1 to n to avoid \n
        printf("-");
    }
    printf("\n");
    return 0;
}
```

```
$>./test_printf
command=./test_printf
----------------------
```

RETOUR DE PRINTF

- La fonction printf retourne un entier négatif en cas d'erreur
- Evènement très rare lié à une sortie redirigée vers un fichier et une erreur d'écriture dans ce dernier
  - Plus de place sur le disque
  - Absence du support amovible
  - Problème de droits en écriture sur le fichier de redirection

AFFICHAGE DES ENTIERS

- %d - entiers signés, %u - entiers non signés
- %o - entiers non signés en octal, %x %X - entiers non signés en hexadécimal

```
int main(void) {
    int i=-45,j=43;
    printf("%d\n",i);
    printf("%u\n",i);
    printf("%o\n",j);
    printf("%x\n",j);
    printf("%X\n",j);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
-45
4294967251
53
2b
2B
```

AFFICHAGE DES RÉELS

- `%f` - standard avec 6 décimales
- `%g` - standard sans les zéros finaux
- `%e` `%E` - notation exponentielle

```
int main(void){
    double f=345.575;
    printf("%f\n",f);
    printf("%g\n",f);
    printf("%e\n",f);
    printf("%E\n",f);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
345.575000
345.575
3.455750e+002
3.455750E+002
```

FORMATS SPÉCIAUX

▷ %p - affiche en hexadécimal des adresses mémoires

▷ %n - stocke le nombre de caractères déjà écrits (nécessite une variable où stocker ce nombre)

```
int main(void){
    int n;
    double d=458.21;
    printf("%g%n",d,&n);
    printf("=%d chars\n",n);
    printf("&n=%p\n",&n);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
458.21=6 chars
&n=0022FF6C
```

LE GABARIT

▷ L'entier inséré entre le `%` et la lettre définissant le type d'affichage de la donnée correspond au nombre minimum de caractères à utiliser

▷ `printf` complète avec des espaces si nécessaire, ou des zéros si `%0` est utilisé à la place de `%`

```
int main(void){
    printf("%5d\n",4);
    printf("%5d\n",123456);
    printf("%5f\n",3.1f);
    printf("%5s\n","abc");
    printf("%05d\n",4);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
    4
123456
3.100000
  abc
00004
```

LA PRÉCISION

▷ L'entier inséré entre un `.` et la lettre définissant le type d'affichage de la donnée réelle correspond à la règle d'arrondi

▷ `%.3f` correspond à un arrondi à 3 décimales

▷ Compatible avec un gabarit

```
int main(void){
    double f=3.14159265;
    printf("%.3f\n",f);
    printf("%08.3f\n",f);
    printf("%.4e\n",f);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
3.142
0003.142
3.1416e+000
```

GABARITS ET PRÉCISIONS VARIABLES

▷ Possibilité de définir des gabarits et précisions variables en fournissant, à la place de l'entier, une variable en argument de `printf` localisée par un `'*'`

```
int main(int argc, char* argv[]){
    int i, tmp, max=0, k=6, n=3;
    float t=3.1455677;
    printf("%0*.*f\n", k, n, t);
    for (i=0;i<argc;i++) {
        tmp=strlen(argv[i]);
        if (tmp>max){
            max=tmp;
        }
    }
    for (i=0;i<argc;i++) {
        printf("arg #%d=%*s\n",i,max,argv[i]);
    }
    return EXIT_SUCCESS;
}
```

```
$>./a.out is the command
03.146
arg #0=./a.out
arg #1=     is
arg #2=    the
arg #3=command
```

SIGNES DES NOMBRES

- `%+d` force l'affichage du signe d'un entier
- `% d` met un espace si le signe de l'entier est un `'+'`

```
int main(void){
    float f=12.54f;
    float g=-12.54f;
    printf("%+f\n",f);
    printf("%+f\n",g);
    printf("% f\n",f);
    printf("% f\n",g);
    printf("%+.3f\n",f);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
+12.540000
-12.540000
 12.540000
-12.540000
+12.540
```

SPRINTF

▷ La fonction **sprintf** permet de stocker le résultat dans une chaîne de caractères supposée assez longue

▷ ⚠Attention aux débordements

```
int main(void){
    char* fname="John";
    char* lname="Doe";
    char email[256];
    sprintf(email,"%s.%s@labri.fr",fname,lname);
    printf("email=%s\n",email);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
email=John.Doe@labri.fr
```

GETCHAR ET GETS

▷ Rappel: La fonction `int getchar()` renvoie le caractère lu ou la constante `EOF` en cas d'erreur

▷ La fonction `char* gets(char *s);` permet la lecture d'une chaîne de caractères, terminée par `'\n'`, sur l'entrée standard et place le résultat dans la variable `s`

```
int main(void){
    char line[10];
    printf("At most 9 characters ?\n");
    if(gets(line)==NULL){
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

```
$>./a.out
At most 9 characters ?
test
```

▷ Cette fonction retourne `NULL` en cas d'erreur et `s` en cas de succès

GETCHAR ET GETS

▷ Rappel: La fonction `int getchar()` renvoie le caractère lu ou la constante `EOF` en cas d'erreur

▷ La fonction `char* gets(char *s);` permet la lecture d'une chaîne de caractères, terminée par `'\n'`, sur l'entrée standard et place le résultat dans la variable `s`

```
int main(void){
    char line[10];
    printf("At most 9 characters ?\n");
    if(gets(line)==NULL){
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

```
$>./a.out
At most 9 characters ?
test
```

▷ Le caractère `'\n'` n'est pas recopié dans la chaîne pointée par `s` et un caractère `nul` (`'\0'`) est placé à la fin de la chaîne

GETCHAR ET GETS

▷ Rappel: La fonction `int getchar()` renvoie le caractère lu ou la constante `EOF` en cas d'erreur

▷ La fonction `char* gets(char *s);` permet la lecture d'une chaîne de caractères, terminée par `'\n'`, sur l'entrée standard et place le résultat dans la variable `s`

```
int main(void){
    char line[10];
    printf("At most 9 characters ?\n");
    if(gets(line)==NULL){
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

```
$>./a.out
At most 9 characters ?
testtroplong
Segmentation fault
```

▷ La fonction `gets` ne gère pas les débordements éventuels sur la chaîne passée en paramètre

LES FICHIERS

Qu'est-ce qu'un fichier ?

- Un fichier est un espace de stockage de données
- Les opérations permises sont
  - Lire, écrire et tronquer
- Il n'est pas prévu de mécanisme pour enlever un morceau au milieu du fichier
- Après ouverture du fichier, toute opération de lecture ou d'écriture s'effectue relativement à une position courante dans le fichier
- Cette position indique la position de fin de la dernière lecture ou écriture dans le fichier

E/S DE LA LIBRAIRIE STDIO.H

- ▷ La librairie `stdio.h` définit un type particulier nommé `FILE` pour décrire un fichier
- ▷ Cette structure varie selon les implémentations
  - ▷ ⚠Il ne faut jamais utiliser ses champs directement
- ▷ Les fonctions de `stdio.h`

```
FILE* fopen(const char* path, const char* mode);
int fclose(FILE* stream);
int fprintf(FILE* stream, const char* format, ...);
int fgetc(FILE* stream);
int fputc(int c, FILE* stream);
char* fgets(char* s, int n, FILE* stream);
int fputs(const char* s, FILE* stream);
size_t fread(void* ptr, size_t size, size_t n, FILE* stream);
size_t fwrite(const void* ptr, size_t size, size_t n, FILE* stream);
```

OUVRIR UN FICHIER

▷ L'ouverture d'un fichier s'effectue à l'aide de la fonction `FILE* fopen(const char* path, const char* mode);` où `path` correspond au chemin de ce dernier et `mode` aux options d'accès

  ▷ `"r"` ouvre le fichier en lecture seule

  ▷ `"w"` ouvre le fichier en écriture seule (le fichier est créé si nécessaire, écrasé si existant)

  ▷ `"a"` ouvre le fichier en écriture seule en fin de fichier (le fichier est créé si nécessaire)

  ▷ `"r+"`, `"w+"`, `"a+"` ouvrer le fichier en lecture et écriture

▷ Les opérations s'effectuent, par défaut, en mode texte

▷ ⚠ Il faut toujours tester la valeur de retour

OUVRIR UN FICHIER

▷ Si le chemin du fichier est relatif, il l'est par rapport au répertoire courant d'exécution

▷ En cas de création du fichier, les droits dépendent du `umask` – user file creation mode mask * (`-rw-r--r--` par défaut)

```
int main(void){
    FILE* f=fopen("foo","w");
    if (f==NULL){
        exit(EXIT_FAILURE);
    }
    fprintf(f,"hello\n");
    fclose(f);
    return EXIT_SUCCESS;
}
```

```
$>./a.out
$>ls -l foo
-rw-r--r--  1 gblin uf-info 6 sep 17 15:38 foo
```

*permissions par défaut d'un répertoire ou d'un fichier créé

FERMER UN FICHIER

- La fermeture d'un fichier s'effectue à l'aide de la fonction `int fclose(FILE* stream);`
- Le paramètre `stream`
  - Doit être non `NULL`
  - Doit représenter une adresse valide
  - Ne doit pas avoir déjà été fermé
- ⚠ Tout fichier ouvert doit être fermé

FPRINTF

▷ La fonction `int fprintf(FILE* stream, const char* format, ...);` fonctionne comme la fonction `printf`

▷ Trois fichiers spéciaux (`stdout`, `stdin` et `stderr`) de type `FILE*` permettent respectivement d'interagir avec la sortie standard, l'entrée standard et la sortie standard d'erreur

▷ Le shell appelant le programme peut traiter spécifiquement et indépendamment chacun de ses fichiers spéciaux (pratique, par exemple, pour récupérer uniquement la sortie d'erreur)

▷ `printf(...)` est équivalent à `fprintf(stdout,...)`

RETOUR SUR LES ERREURS GRAVES

▷ En cas d'erreur insurmontable, il est possible d'interrompre le programme en appelant l'instruction `exit` définie dans `stdlib.h` qui prend en paramètre `EXIT_FAILURE` ou `EXIT_SUCCESS`

▷ Il faut fournir un message d'explication clair en utilisant la sortie d'erreur standard

```
int foo(){
    ...
    if(erreur_fatale){
        fprintf(stderr,"A fatal error occured");
        exit(EXIT_FAILURE);
    }
    ...
}
```

FGETC ET FPUTC

- Il est possible de lire et écrire un unique caractère à l'aide des fonctions `int fgetc(FILE* stream);` et `int fputc(int c, FILE* stream);`
- Ces deux fonctions retourent `EOF` en cas d'erreur
  - Il n'existe pas de caractère de fin de fichier
  - `EOF` n'est pas un caractère

FGETS

- La fonction `char* fgets(char* s, int n, FILE* stream);` permet de lire des caractères dans un fichier et les stocke dans `s` jusqu'à
  - Un `'\n'` (copié dans le résultat)
  - La fin du fichier
  - Avoir lu `n-1` caractères
- La fonction retourne `NULL` en cas d'erreur, `s` sinon
- Le paramètre `n` permet d'éviter les problèmes de débordement

FPUTS

▷ La fonction `int fputs(const char* s, FILE* stream);` permet d'écrire la chaîne `s` dans le fichier `stream` en suivant le mode sélectionné lors de l'ouverture

▷ La fonction retourne `EOF` en cas d'erreur, `0` sinon

FEOF

▷ La fonction `int feof(FILE* stream);` permet de tester si la fin du fichier a été atteinte

▷ La fonction retourne `0` si la fin de fichier est atteinte

▷ Pas très pratique car cette fonction nécessite l'échec d'une lecture pour renvoyer la bonne valeur

```
int main(void){
    char s[4096];
    FILE* f=fopen("foo","r");
    if (f==NULL){
        exit(EXIT_FAILURE);
    }
    while (!feof(f)) {
        fgets(s,4095,f);
        printf("** %s **\n",s);
    }
    fclose(f);
    return EXIT_SUCCESS;
}
```

```
$>cat foo
abc toto
23
$>./a.out
** abc toto **
** 23 **
** 23 **
```

FSEEK

- La fonction `int fseek(FILE* stream, long offset, int whence);` modifie la position courante dans le fichier
- La position est définie comme la somme `offset+whence`
- `whence` peut prendre les valeurs
  - `SEEK_SET` - début du fichier
  - `SEEK_CUR` - position courante
  - `SEEK_END` - fin du fichier

POSITIONNEMENT

▷ La fonction `void rewind(FILE* stream);` revient au début du fichier

▷ La fonction `long ftell(FILE* stream);` retourne la position courante du fichier

```
long filesize(FILE* f) {
    long old_pos=ftell(f);
    fseek(f,0,SEEK_END);
    long size=1+ftell(f);
    fseek(f,old_pos,SEEK_SET);
    return size;
}
```

DOGGY BAG

TO TAKE AWAY ...

▷ Les formats d'affichage peuvent être finement paramétrés à l'aide des gabarits et précisions

▷ L'ensemble des fonctions de la bibliothèque standard `stdio.h` permettent aisément une manipulation textuelle des données d'un fichier

▷ L'interaction avec le monde extérieur peut se faire à l'aide des trois fichiers spéciaux que sont `stdout`, `stdin` et `stderr`

Questions?