IN3R11-2 – C

Cours 4

Damien MASSON
d.masson@esiee.fr

http://esiee.fr/~massond/Teaching/IN3R11-2/

22 novembre 2010

printf

- renvoit le nombre de caractères affichés
- retour négatif en cas d'erreur
- très rare : sortie redirigée vers un fichier et plus de place sur le disque
- les autres cas d'erreur (droits, disque amovibles débranché...) sont gérés par le système

Affichage des entiers

- %d : entier signé
- %u : entier non signé
- %o : entier non signé en octal
- %x %X : entier non signé en hexadécimal

```c
int main(int argc,char* argv[]){
    int i=-45,j=43;
    printf("%d\n",i); /* -45 */
    printf("%u\n",i); /* 4294967251 */

    printf("%o\n",j); /* 53 */
    printf("%x\n",j); /* 2b */
    printf("%X\n",j); /* 2B */
    return 0;
}
```

Affichage des réels

- %f : standard, 6 décimales
- %g : standard, sans les zéros finaux
- %e %E : notation exponentielle

```c
int main(int argc,char* argv[]) {
    double f=345.575;
    printf("%f\n",f); /* 345.575000     */
    printf("%g\n",f); /* 345.575        */
    printf("%e\n",f); /* 3.455750e+002 */
    printf("%E\n",f); /* 3.455750E+002 */
    return 0;
}
```

Formats spéciaux

- %p : affichage en hexadécimal des adresses mémoires
- %n : stocke le nombre de caractères déjà écrits

```c
int main(int argc,char* argv[]) {
    int n;
    double d=458.21;
    printf("%g%n",d,&n);       /* 458.21=6 chars */
    printf("=%d␣chars\n",n); /* &n=0022FF6C */
    printf("&n=%p\n",&n);
    return 0;
}
```

le Gabarit

- entier après le % = nombre minimum de caractères à utiliser
- complète avec des espaces si nécessaire, ou des zéros si %0 avant un nombre gabarit != précision ! !

```
int main(int argc,char* argv[]) {
    printf("%5d\n",4);          /*     4    */
    printf("%5d\n",123456);     /* 123456   */
    printf("%5f\n",3.1f);       /* 3.100000 */
    printf("%5s\n","abc");      /*   abc    */
    printf("%05d\n",4);         /* 00004    */
    return 0;
}
```

Précision

- %.3f : 3 décimales
- règle d'arrondi
- compatible avec un gabarit

```c
int main(int argc,char* argv[]) {
    double f=3.14159265;
    printf("%.3f\n",f);    /* 3.142         */
    printf("%08.3f\n",f);  /* 0003.142      */
    printf("%.4e\n",f);    /* 3.1416e+000   */
    return 0;
}
```

Gabarit et précision variables

si on ne les connaît pas à la compilation : * au lieu d'un entier + variable en argument de printf

```
int main(int argc,char* argv[]) {
  int i,tmp;
  int max=0;
  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 0;
}
/*
arg #0=    ./t
arg #1=     is
arg #2=    the
arg #3=command
*/
```

Signes des nombres

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

```c
int main(int argc,char* argv[]) {
    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 0;
}
/*
+12.540000
-12.540000
 12.540000
-12.540000
+12.540
*/
```

sprintf

Idem que printf mais écrit dans une chaîne de caractères supposée assez longue. Attention aux débordements

```
int main(int argc,char* argv[]) {
    char* firstname="John";
    char* lastname="Doe";
    char email[256];
    sprintf(email,"%s.%s@esiee.fr",
                    firstname,lastname);
    printf("email=%s\n",email);
    return 0;
}
```

scanf

% ∗ [*cd*...] : saisir sans sauver le résultat dans une variable :

```
int scan_positive_integer() {
    int res,n=-1;
    do {
        res=scanf("%d",&n);
        if (res==-1) return -1;
        if (res==0) {
            scanf("%*c"); /* ici */
        }
    } while (n<0);
    return n;
}
```

sscanf

Permet de lire dans une chaîne de caractère

```
int main(int argc,char* argv[]) {
    char* date="Tue␣Jul␣17␣14:50:00␣CEST␣2007";
    char day[4],month[4];
    int nday,year;
    sscanf(date,"%s␣%s␣%d␣%*s␣%*s␣%d",
                  day,month,&nday,&year);
    printf("%s␣%d␣%s␣%d\n",day,nday,month,year);
    return 0;
}
```

Autres entrées/sorties bufferisées

Entrées sorties bufferisées sans formats :

- getchar : lit un char
- putchar : écrit un char
- gets : lit une chaîne de caractères
- puts : écrit une chaîne de caractères

Entrées/sorties non buffereisées

Appels systèmes : read/write (voir fichiers)

Fichiers

- espace de stockage de données
- opérations permises : lire, écrire, tronquer
- il n'est pas prévu de mécanisme pour enlever un morceau au milieu d'un fichier
- tous les fichiers ne sont pas en random access (ex : /dev/mouse)

primitives

- primitives systèmes :

```
int open(const char* pathname,int flags,mode_t mode);
int close(int fd);
ssize_t read(int fd,void* buf,size_t count);
ssize_t write(int fd,const void* buf,size_t count);
```

- peu pratiques car ne manipulent que des octets

structure FILE

- FILE = structure décrivant un fichier
- varie selon les implémentations ne jamais utiliser ses champs directement
- fonctions de stdio.h :

```
FILE* fopen(const char* path,const char* mode);
int fclose(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);
int fscanf(FILE* stream,const char* format, ...);
int fprintf(FILE* stream,const char* format, ...);
```

Ouvrir un fichier

- options de fopen :
  - "r" : lecture seule
  - "w" : écriture écriture seule (fichier créé si nécessaire, écrasé si existant)
  - "a" : ajout en fin (fichier créé si nécessaire)
- toujours tester la valeur de retour
- par défaut, en mode texte, pour opération en mode binaire : "rb" "wb" "ab"
- en mode lecture et écriture : "r+" "w+" "a+"

Fermer un fichier

- fclose(f) ;
- f doit être non NULL
- f doit représenter une adresse valide obtenue avec fopen
- f ne doit pas avoir déjà été fermé
- tout fichier ouvert doit être fermé ! ! (fuite mémoire)

entrés/sorties formatées

- fprintf, fscanf : fonctionnent comme printf et scanf
- printf = fprintf(stdout...
- scanf = fscanf(stdin...
- stdin, stdout, stderr dont des FILE*
- fgetc, fputc fonctionnent comme getchar putchar
- fgets, fputs fonctionnent comme gets puts
- toujours lire le man de ces fonctions avant de les utiliser (pour être sur de ce qu'elles font et ne font pas)

Fin de fichier en lecture

- fscanf : EOF
- fgetc : EOF
- fgets : NULL
- fread : 0

```
void copy(FILE* src,FILE* dst) {
    char buffer[4096];
    size_t n;
    while ((n=fread(buffer,sizeof(char),4096,src))>0) {
        fwrite(buffer,sizeof(char),n,dst);
    }
}
```