# Programmation et structures de données en C cours 3: Chaînes de caractères et entrées/sorties

Jean-Lou Desbarbieux, Stéphane Doncieux
et Mathilde Carpentier
LU2IN018 Sorbonne Université 2021/2022

## Sommaire



# Chaînes de caractères

## Chaînes de caractères : rappels

- Dans un tableau, terminé par un `'\0'`
- Exemple :

```
char nom[]="david";
char nom2[8]="goliath";
/* il faut une case pour '\0' */

char nom3[]="david\0goliath";
printf("nom3=%s\n",nom3); /* resultat ? */

/* et s'il n'y a pas assez de cases ? */
char nom4[7]="goliath";
printf("nom4=%s\n",nom4); /* resultat ? */
```

## Quelques fonctions utiles

```c
#include <string.h>

 /* renvoie la longueur de la chaine */
size_t strlen (const char *s);

/* Comparaison entre chaines. Renvoie:
 * 0 si les deux chaines sont egales
 * une valeur negative si s1 arrive avant s2
 * une valeur positive sinon */
int strcmp (const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2,
            size_t n);
```

## Quelques fonctions utiles

```c
#include <string.h>

/* Copie entre deux chaines de caracteres
 (pas d'allocation), renvoie un pointeur sur dest */
char *strcpy (char *dest, const char *src);
char *strncpy(char *dest, const char *src,
              size_t n);

/* Alloue une nouvelle chaine de longueur strlen(s1)+1
 et l'initialise avec la chaine pointee par s1
  (equivalent a malloc, puis strcpy).
    ATTENTION: il faut bien penser a liberer la chaine
    ainsi allouee avec un free */
char *strdup(const char *s1);
```

## De la convention utilisée en C pour les chaînes de caractères...

Avantages :

- ►
- ►

Inconvénients :

- ►
- ►

## scanf

- ► Lecture de données tapées au clavier
- ► Renvoie un entier : nombre d'entités lues
- ► Exploite également un format avec des codes de format

```c
float x;
int n;
scanf("%f␣%d",&x,&n);
char m[20];
scanf("%s", m); /* attention, danger ! */
scanf("%19s", m); /* plus securise...
     lecture de 19 caracteres max */
```

## scanf

ATTENTION : lors de la lecture, seul ce qui est demandé est lu !
Exemple pathologique :

```
char c;
scanf("%c",&c);
printf("Char lu : %c\n", c);
scanf("%c",&c);
printf("Char lu : %c\n", c);
```

Que fait le programme ?
Lecture d'un seul caractère ! Le 2eme étant le retour à la ligne...
Solution ajouter un ' ' avant le premier code de format :

```
scanf(" %c",&c);
```

→ l'espace indique que le `scanf` doit "consommer" tous les espaces, retour à la ligne et autre tabulation avant de lire quoi que ce soit...

## getchar

ATTENTION : même problème sur le `getchar` que sur `scanf`
Exemple de solution :

```
int c;
do
  c=getchar();
while (c=='\n');
```

ou pour éviter aussi espaces, retours à la ligne et autres tabulations :

```
#include <ctype.h>

int c;
do
  c=getchar();
while (isspace(c));
```

## getchar/putchar

- `int getchar()` : lecture d'un caractère. Renvoie le code du caractère lu, `EOF` si problème. Équivalent à `scanf("%c",&c_lu)`
- `int putchar(int c)` : écriture d'un caractère. Renvoie le code du caractère écrit, `EOF` si problème. Équivalent à `printf("%c",c_a_ecrire)`
- Fonctions plus simples et plus rapides

# Fichiers

## Fichiers : principes

- Un fichier est trouvé sur le disque dur à partir de son nom complet (incluant le répertoire) :
  - `mon_fichier` : fichier nommé `mon_fichier` dans le répertoire courant. Équivalent à `./mon_fichier`
  - `../repertoire/mon_fichier` : fichier nommé `mon_fichier` dans le répertoire `repertoire` qui est dans le répertoire au-dessus du répertoire courant
  - `/home/utilisateur/mon_fichier` : fichier nommé `mon_fichier` dans le répertoire `/home/utilisateur`
- Un programme C voit le fichier comme une suite d'octets (même si le système d'exploitation le gère autrement)
- L'accès est (essentiellement) séquentiel : on lit/écrit les octets les uns après les autres
- La lecture/écriture passe par un buffer intermédiaire (c'est en général transparent, sauf en cas d'arrêt prématuré)

## Fichiers : principes

- Manipulation d'un fichier : de quoi a besoin le programme ?
  - Savoir où en est la lecture/écriture dans le fichier
  - Indicateurs d'erreurs éventuelles
  - Indicateur de fin de fichier
  - Gestion du tampon de lecture/écriture
- Tout est géré au travers d'une structure (`FILE`)
- Les fonctions de lecture/écriture mettent à jour automatiquement la structure

## Fichiers : principes

- La structure `FILE` est initialisée par une fonction d'ouverture de fichier, et est détruite par une fonction de fermeture de fichier

Exemple :

```
#include <stdio.h>
int main() {
  FILE *mon_fichier = fopen("fichier.txt","r")
  int x,y,z;
  if (mon_fichier==NULL) {
    printf("Erreur a l'ouverture de fichier.txt\n");
    return 1;
  }
  fscanf(mon_fichier,"%d %d %d",&x,&y,&z);
  printf("Valeurs lues: %d %d %d\n",x,y,z);
  fclose(mon_fichier);
  return 0;
}
```

## Ouverture d'un fichier : `fopen/fclose`

- `FILE *fopen(const char *nom_fichier, const char *mode)` : ouvre le fichier selon le mode indiqué ("r", "w", "a",...)
- `int fclose(FILE *flux)` : vide éventuellement le tampon de lecture/écriture, libère la mémoire correspondante et ferme le fichier

## Lecture/écriture formatée

- `int fscanf(FILE *flux,const char *format, ...)` : lecture de données à partir d'un format depuis un fichier
- `int fprintf(FILE *flux, const char *format,...)` : écriture de données selon un format spécifié dans un fichier
- `int fputc(int c, FILE *flux)` : écrit un caractère dans un fichier
- `int fgetc(FILE *flux)` : lit un caractère depuis un fichier
- `char * fgets(char *buffer, int n, FILE *flux)` : lit une ligne dans un fichier (longueur maximale=n)

## Ecriture

```
int main(){
 FILE *pFi=NULL;
 float x=10, y=12, z=13.5,
 char nom[]="Dupont", prenom[]="Eugene";

 pFi=fopen("out.txt", "w");
  if(pFi==NULL){
     printf ("Erreur a l 'ouverture du fichier\n");
     return 1;
}
 fprintf(pFi, "%s %s %f %f %f\n", nom, prenom, x,y,z);
 fclose(pFi);
return 0;
}
```

## Lecture d'un fichier ligne à ligne

Lecture d'un fichier contenant 3 entiers par ligne avec `fgets` et `sscanf`

```
#include <stdio.h>
#define LONGUEURLIGNE 128
 int main() {
   FILE *src;
   if ((src= fopen("source.txt","r"))==NULL) {
     printf("Erreur a l'ouverture de source.txt\n");
     return 1;
   }
   char buffer[LONGUEURLIGNE];
   char *res=fgets(buffer, LONGUEURLIGNE, src);
   int a,b,c;
   while (res != NULL) {
      sscanf(buffer,"%d %d %d",&a, &b, &c);
      res=fgets(buffer, LONGUEURLIGNE, src);
      printf("Lu les entiers a=%d, b=%d, c=%d\n",a,b,c);
   }

   fclose(src);
   return 0;
 }
```

## La lecture binaire

```
size_t fread (void *ptr, size_t size,
              size_t nmemb, FILE *stream);
```

La fonction fread lit nmemb éléments de données, chacun d'eux représentant size octets de long, depuis le flux pointé par stream, et les stocke à l'emplacement pointé par ptr.

Les données peuvent être de type quelconque, `ptr` peut être un pointeur sur un int, un float, une struct... Il peut être un tableau de données d'un type quelconque.

ATTENTION : il n'y a aucune espèce de format, les données écrites dans le fichier sont recopiées directement en mémoire.

## La lecture binaire

```
FILE *fichier;
/*  ouverture du fichier */
if ((fichier=fopen(nom_fichier,"rb"))==NULL) {
  printf("Erreur lors de la lecture de %s\n",nom_fichier);
  exit(1);
}
int i[10];

int nb_lu=fread(&i, sizeof(int),10,fichier);

if (nb_lu == 10) {
  printf(``Lecture OK\n'');
}
else {
  printf(``Lecture KO\n'');

}
```

## L'écriture binaire

```
size_t fwrite (const void *ptr, size_t size,
               size_t nmemb, FILE *stream);
```

La fonction `fwrite` écrit `nmemb` éléments de données, chacun d'eux représentant `size` octets de long, dans le flux pointé par stream, après les avoir lus depuis l'emplacement pointé par `ptr`.

## L'écriture binaire

```
    int tab[10];

    /* Initialisation de tab */
    ...

  /*  ouverture du fichier */
  if ((fichier=fopen(nom_fichier,"wb"))==NULL) {
    printf("Erreur lors de l'ecriture de %s\n",nom_fichier);
    exit(1);
  }

/* ATTENTION: pas de '&' car tab est un tableau ! */
  fwrite(tab,sizeof(int),10,fichier);

  /* fermeture du fichier */
  fclose(fichier);
```

# Structures et entrées/sorties

## Structures et entrées/sorties

```c
typedef struct _point {
  float x;
  float y;
  float z;
} point;
point p1={ 1.0, 2.0, 3.0},p2;

FILE *f=fopen("mon_fichier","wb");
fwrite(&p1,sizeof(point),1,f);
fclose(f);
f=fopen("mon_fichier","rb");
fread(&p2,sizeof(point),1,f);
```

## Structures et entrées/sorties

```c
// Avec un tableau ?

typedef struct _contact {
  char nom[30];
  char prenom[30];
} contact;
contact c1={"Tyrell", "Eldon" };

FILE *f=fopen("mon_fichier","wb");
fwrite(&c1,sizeof(contact),1,f);
fclose(f);

 // ca marche !
 // Que contient le fichier ?
```

## Structures et entrées/sorties

```c
// Avec des pointeurs ?

typedef struct _contact {
  char * nom;
  char * prenom;
} contact;
contact c1;
c1.nom = strdup("Tyrell");
c1.prenom = strdup("Eldon");

FILE *f=fopen("mon_fichier","wb");
fwrite(&c1,sizeof(contact),1,f);
fclose(f);

 // qu'est-ce qui est ecrit ?
```

## Structures et entrées/sorties

```c
// Avec des pointeurs ? Ecriture:

typedef struct _contact {
  char * nom;
  char * prenom;
} contact;
contact c1;
c1.nom = strdup("Tyrell");
c1.prenom = strdup("Eldon");

FILE *f=fopen("mon_fichier","w");
fprintf(f,"%s\n",c1.nom);
fprintf(f,"%s\n",c1.prenom);
fclose(f);

// quel est le contenu du fichier ?
```

## Structures et entrées/sorties

```
// Avec des pointeurs ? Lecture
contact c2;

FILE *f=fopen("mon_fichier","r");
char buffer[128];
fgets(buffer,128,f);
// pour enlever le '\n'
buffer[strlen(buffer)-1]='\0';
c2.nom=strdup(buffer);

fgets(buffer,128,f);
// pour enlever le '\n'
buffer[strlen(buffer)-1]='\0';
c2.prenom=strdup(buffer);

fclose(f);
```

# Aide mémoire sur les entrées/sorties

## Comment faire ...

... pour écrire des données à l'écran :
méthode 1

- utiliser une fonction transformant les variables à écrire directement en chaînes de caractères (`printf`)

méthode 2

- écrire caractères par caractères (`putchar`)

## Comment faire ...

... pour lire des données tapées depuis le clavier :
méthode 1

- utiliser une fonction de lecture traduisant directement la chaîne lue dans le(s) type(s) souhaité(s) (`scanf`)

méthode 2

- lire caractères par caractères (`getchar`)

## Comment faire ...

... pour lire des données depuis un fichier :

méthode 1

- ouvrir le fichier (`fopen`)
- utiliser une fonction de lecture traduisant directement la chaîne lue dans le(s) type(s) souhaité(s) (`fscanf` ou `fgets` + `sscanf`)
- fermer le fichier (`fclose`)

méthode 2

- ouvrir le fichier (`fopen`)
- utiliser une fonction de lecture caractères par caractères (`fgetc`)
- fermer le fichier (`fclose`)

méthode 3

- ouvrir le fichier (`fopen`, mode binaire)
- utiliser une fonction de lecture copiant les données brutes depuis le fichier vers la mémoire (lecture dite binaire) (`fread`)
- fermer le fichier (`fclose`)

# Exemple : charger/sauver une image (TME0)

## Comment faire ...

... pour écrire des données dans un fichier :

méthode 1

- ouvrir le fichier (`fopen`)
- utiliser une fonction d'écriture transformant les variables à écrire directement en chaînes de caractères (`fprintf`)
- fermer le fichier (`fclose`)

méthode 2

- ouvrir le fichier (`fopen`)
- utiliser une fonction d'écriture caractères par caractères (`fputc`)
- fermer le fichier (`fclose`)

méthode 3

- ouvrir le fichier (`fopen`)
- utiliser une fonction d'écriture copiant la mémoire vers le fichier sans transformation (écriture dite binaire) (`fwrite`)
- fermer le fichier (`fclose`)

## Format d'une image

Les fichiers d'image au format PGM peuvent être en ASCII ou en BINAIRE.
Les premières lignes constituent l'entête de l'image et sont toujours les mêmes :

- `P2` ou `P5` : format du fichier (resp. ASCII ou BINAIRE)
- largeur et hauteur de l'image
- valeur maximum pour les niveaux de gris

Les lignes commençant par '#' peuvent être ignorées.
Ensuite : valeurs des pixels en ASCII (ligne par ligne, pixels indiqués en ASCII séparés par des espaces) ou en binaire.
0 correspond au noir et la valeur maximale indiquée correspond au blanc.
Pour simplifier, on suppose que l'image a toujours, au maximum 255 niveaux de gris (chaque pixel est sur 1 seul octet).

## Format d'une image

Exemple de fichier en ASCII :

```
P2
# Commentaire bla bla
4 7
15
0 0 0 0
0 3 3 3
0 3 0 0
0 7 7 3
0 3 0 0
0 3 0 0
0 0 0 0
```

## Ecriture d'une image (1)

```
int sauver_image_pgm(char *nom_fichier, image_t *img)
{
  FILE *f;

  f=fopen(nom_fichier, "w");

  if (!f)
    {
      fprintf(stderr, "impossible d'ouvrir le fichier %s\n", nom_fichier);
      return 0;
    }

  fprintf(f, "P5\n");
  fprintf(f, "#Genere par notre programme\n");
  fprintf(f, "%ld %ld\n", img->w, img->h);
  fprintf(f, "255\n");

  fwrite(img->buff, img->w, img->h, f);
  fclose(f);
  return 1;
}
```

## Ecriture d'une image (2)

```
int sauver_image_pgm(char *nom_fichier, image_t *img)
{
  FILE *f;
  f=fopen(nom_fichier, "w");
  if (!f)
    {
      fprintf(stderr, "impossible d'ouvrir le fichier %s\n", nom_fichier);
      return 0;
    }
  fprintf(f, "P2\n");
  fprintf(f, "#Genere par notre programme\n");
  fprintf(f, "%ld %ld\n", img->w, img->h);
  fprintf(f, "255\n");
  int i, j;
  unsigned char *p=img->buff;
  for (i=0;i<img->h;i++) {
    for (j=0;j<img->w; j++) {
      fprintf(f,"%hhu ", *p);
      p++;
    }
    fprintf(f,"\n");
  }
  fwrite(img->buff, img->w, img->h, f);
  fclose(f);
  return 1;
}
```

## Lecture d'une image (1)

```
image_t *charger_image_pgm(char *nom_fichier)
{
  FILE *f;
  image_t *img;
  unsigned int nb_ng;
  char tmp_str[TMP_STR_SIZE];
  enum format {BIN, ASCII} pgm_form;

  /* Ouverture du fichier en lecture */
  f=fopen(nom_fichier, "r");


  if (!f)
    {
      /* on ecrit sur le flux d'erreur */
      fprintf(stderr, "impossible d'ouvrir le fichier %s\n", nom_fichier);
      return NULL;
    }

  /* to be continued... */
```

## Lecture d'une image (2)

```
  /* ... */

  /* on lit une ligne en supprimant les eventuels commentaires */
  do
    fgets(tmp_str, TMP_STR_SIZE, f);
  while (*tmp_str == '#');

  /* on determine le format */
  if ( !strcmp(tmp_str, "P5\n"))
    pgm_form = BIN;
  else if ( !strcmp(tmp_str, "P2\n"))
    pgm_form = ASCII;
  else
    {
      fprintf(stderr, "format fichier non pgm\n");
      return NULL;
    }

  img = creer_image();

  /* to be continued... */
```

## Lecture d'une image (3)

```
    /* ... */
  /* on lit une ligne en supprimant les eventuels commentaires */
  do
    fgets(tmp_str, TMP_STR_SIZE, f);
  while (*tmp_str == '#');

  /* on determine la largeur et la hauteur de l'image */
  if (sscanf(tmp_str, "%ld %ld\n", &(img->w), &(img->h)) != 2)
    {
      /* si le sscanf n'a pas lu les deux entiers longs attendus: */
      fprintf(stderr, "format fichier non pgm\n");
      detruire_image(img);
      return NULL;
    }
  /* on lit une ligne en supprimant les eventuels commentaires */
  do
    fgets(tmp_str, TMP_STR_SIZE, f);
  while (*tmp_str == '#');
  /* on lit le niveau de gris */
  if (sscanf(tmp_str, " %d ", &nb_ng) != 1)
    {
      /* si le sscanf n'a pas lu l'entier attendu: */
      fprintf(stderr, "format fichier non pgm\n");
      detruire_image(img);
      return NULL;
    }
  /* to be continued... */
```

## Lecture d'une image (4)

```
    /* ... */

  /* on alloue l'espace pour stocker l'image */
  img->buff = (unsigned char *) malloc(img->w * img->h * sizeof (unsigned char));

  /* on lit l'image en prenant en compte le format */
  if ( pgm_form == BIN )
    {
      if (fread(img->buff, img->w, img->h, f) != img->h)
        {
          fprintf(stderr, "fichier image imcomplet!\n");
          return NULL;
        }
    }
  else
    {
      unsigned int i,j;
      unsigned char *p = img->buff;
      for ( i = 0; i < img->h; i++)
        for ( j = 0; j < img->w; j++)
          fscanf(f, "%hhu ", p++);
    }

  fclose(f);
  return img;
}
```

C'est tout pour aujourd'hui...