Programmation en C
Compléments

Alain CROUZIL
alain.crouzil@irit.fr

Département d'Informatique (DdI)

Institut de Recherche en Informatique de Toulouse (IRIT)
Équipe Computational Imaging and Vision (MINDS)

Faculté Sciences et Ingénierie (FSI)
Université Toulouse III – Paul Sabatier (UPS)

Licence Informatique – Licence MIASHS
2019-2020

D[i] Département Informatique

UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté Sciences et Ingénierie

IRIT
Institut de Recherche en Informatique de Toulouse

Sommaire

Sommaire

Structures (1/1)

Une structure permet de déclarer des variables composées d'un ensemble de données (champs) pouvant être de types différents.

Définition et déclaration

- **Définition d'une structure**

```
struct identificateur
{
  déclaration_des_champs
};
```

Exemple :

```
struct point
{
  int numero;
  float x;
  float y;
};
```

Exemple : **struct** point pt1,pt2;

Structures (2/2)

Accès aux champs d'une variable structurée

identificateur_de_variable.identificateur_de_champ

Exemples :

```
pt1.numero=1;
pt1.x=0.5;
pt1.y=12.3;
printf("%f\n",pt1.x);
```

Initialisation

Une variable structurée peut être initialisée au moment de sa déclaration.
Exemple : **struct** point pt={12, 5.35, 10.4};

Affectation

Contrairement aux tableaux, on peut affecter en une seule fois le contenu d'une variable structurée à une autre variable structurée du même type.
Exemple : pt2=pt1;

Sommaire

Pointeurs et structures (1/)

Priorité des opérateurs

L'opérateur `.` est prioritaire sur l'opérateur `*`

Exemple : **struct** point {**float** x,y;} pt,*pp; pp=&pt;

On a alors : (*pp).x==pt.x et `(*pp).y==pt.y`, **mais** *pp.x n'a pas de sens car pp.x n'a pas de sens.

Opérateur ->

L'expression p->a désigne « le champ a de la variable structurée pointée par p ».

Dans l'exemple précédent, on a : (*pp).x == pp->x

Pointeurs et structures (2/2)

Exemple

Une liste est une suite finie, éventuellement vide, d'éléments de même type.
Exemples de représentation :

- tableaux ;
- listes chaînées : utilisation des variables structurées (les cellules de la liste) dont le dernier champ est un pointeur vers l'élément suivant (la cellule suivante).

Par exemple, pour représenter un polygone :

```
struct sommet
{
  float x;
  float y;
  struct sommet *pSuivant;
};
```

Le principe des listes chaînées sera utilisé avec la gestion dynamique de la mémoire pour constituer les listes chaînées dynamiques.

À vos boitiers !

14

Sommaire

Type énuméré

Le type énuméré est un cas particulier de type entier.

Exemples

- **enum** couleur {rouge, vert, bleu};
  définit un type **enum** couleur comportant trois valeurs possibles désignées par les constantes rouge, vert et bleu, qui sont des entiers ordinaires (ce qui implique une très grande souplesse d'utilisation...) valant respectivement 0, 1 et 2.
- avec **enum** couleur {rouge=2, vert=4, bleu};
  les trois constantes valent respectivement 2, 4 et 5.
- **enum** couleur c1,c2;
  déclare deux variables c1 et c2 de type **enum** couleur. On peut alors écrire :

```
c1=rouge;
printf("%d\n",c1);
c2=c1;
```

Mais il n'est pas possible d'écrire : rouge=3;

Sommaire

Définition de synonymes de types (1/2)

Principe

L'instruction `typedef` permet de définir des synonymes de types, c'est-à-dire de donner un nom à un type quelconque (sans créer de nouveau type), aussi complexe soit-il, puis d'utiliser ce nom comme spécificateur de type pour simplifier les déclarations.

Exemples

- **typedef int** Entier;
  int a,b; $\Longleftrightarrow$ Entier a,b;
- **typedef int** $*$Ptr;
  int $*$p1,$*$p2; $\Longleftrightarrow$ Ptr p1,p2;
- **typedef double** Point[3];
  double p[3],q[3]; $\Longleftrightarrow$ Point p,q;

Définition de synonymes de types (2/2)

Exemples (suite)

- **typedef struct**
```
{
   char Nom[30];
   char Appellation[20];
   int Millesime;
   int Quantite;
} Vin;
Vin v1,v2;
```

- **typedef enum** {faux=0, vrai=1} Booleen;
```
Booleen Indic;
Indic=vrai;
...
if (Indic)
...
```

Sommaire

Qualifieur const (1/3)

Rôle

- **const** permet de préciser que la valeur d'une variable ne doit pas changer lors de l'exécution du programme.
- Les éventuelles instructions permettant la modification de la valeur de cette variable sont signalées par le compilateur sous la forme d'avertissements (*warnings*).

Exemples

- **const int** a=12; déclare un entier constant de valeur 12. La valeur de a ne peut pas être changée par la suite sans avertissement :

```
a=20; /* produit un avertissement */
a++; /* produit un avertissement */

int *p;
p=&a; /* produit un avertissement */
*p=20;
```

Qualifieur const (2/3)

Exemples (suite)

- Avec les pointeurs :

```
const char * pc1; /* la zone pointée est constante, le pointeur est modifiable */
char c;
char * const pc2=&c; /* la zone pointée est modifiable, le pointeur est constant */
const char * const pc3=&c; /* la zone pointée et le pointeur sont constants */
```

Qualifieur const (3/3)

Exemples (fin)

- Avec les variables structurées :
  Exemple 1 :

```
struct date
{
   unsigned int jour;
   unsigned int mois;
   unsigned int annee;
};
const struct date revolution={14,7,1789}; /* tous les champs sont constants */
```

Exemple 2 :

```
struct date
{
   unsigned int jour;
   const unsigned int mois;
   const unsigned int annee;
};
struct date revolution={14,7,1789}; /* le champ jour est modifiable */
```

Sommaire

Compléments sur les entrées-sorties

Support complémentaire

Travaillez à partir du support complémentaire (chapitre 3).
Ici, nous ne verrons que le fonctionnement des tampons (*buffers*).

Unités standard

- l'entrée standard (`stdin`) : par défaut, le clavier ;
- la sortie standard (`stdout`) : par défaut, l'écran ;
- la sortie standard des erreurs (`stderr`) : par défaut, l'écran.

L'entrée est ouverte en lecture et les sorties sont ouvertes en écriture.

Fonctionnement du tampon d'entrée (1/3)

Les tampons

À chaque flux d'entrée et de sortie est associée une zone mémoire appelée tampon ou *buffer* qui sert d'intermédiaire entre le programme et le périphérique concerné.

Le tampon d'entrée

- Il contient les caractères tapés par l'utilisateur.
- Il fonctionne selon le principe de la file (premier arrivé = premier sorti).
- Lors de l'exécution de scanf("%c",&car); (ou car=getchar();) deux situations :
  - tampon d'entrée pas vide : la variable car reçoit le caractère le plus ancien se trouvant dans le tampon d'entrée et ce caractère est enlevé du tampon d'entrée (consommé) ;
  - tampon d'entrée vide : le déroulement du programme est stoppé jusqu'à ce que le tampon d'entrée reçoive un ou plusieurs caractères.
- L'ajout de caractères dans le tampon d'entrée ne se fait qu'au moment ou l'utilisateur valide sa frappe avec la touche « Entrée » (↩) correspondant à `'\n'`.
  Exemple : si l'utilisateur tape les caractères texTe␣TApé↩
  - le contenu du tampon d'entrée n'est modifié que lors de la frappe de ↩ ;
  - il reçoit 11 caractères.

Fonctionnement du tampon d'entrée (2/3)

Vidage du tampon d'entrée

Nécessité de vider le tampon d'entrée lors de la lecture d'un caractère, à un moment où le tampon d'entrée risque de ne pas être vide.
Exemple : la séquence suivante n'est pas correctement écrite :

```
printf("Etes-vous␣d'accord␣(o/n)␣?\n"); /* M1 */
rep=getchar();
while ((rep!='o') && (rep!='n'))
{
   printf("Tapez␣o␣ou␣n␣:\n"); /* M2 */
   rep=getchar();
}
```

En effet, si l'utilisateur tape, par mégarde, la séquence a↩, en réponse au message M1, alors le message M2 s'affichera deux fois au lieu d'une, puisque le tampon d'entrée contient deux caractères (a et ↩) différents des deux caractères autorisés (o et n). Pour remédier à ce problème, il faut procéder à un vidage conditionnel du tampon d'entrée avant l'instruction de lecture de la boucle.

Fonctionnement du tampon d'entrée (3/3)

Exemple de vidage conditionnel du tampon d'entrée

```
printf("Etes-vous␣d'accord␣(o/n)␣?\n"); /* M1 */
rep=getchar();
while ((rep!='o') && (rep!='n'))
{
  printf("Tapez␣o␣ou␣n␣:\n"); /* M2 */
  if (rep!='\n') while (getchar()!='\n');
  rep=getchar();
}
```

La condition (rep!='\n') permet de prendre en compte la possibilité que l'utilisateur ait pu taper ↩ seulement. En effet, dans ce cas, si l'on omet la condition, la boucle **while** (getchar()!='\n'); serait bloquante.

Fonctionnement du tampon de sortie

Vidage du tampon de sortie

- Les fonctions d'affichage envoient les caractères non pas directement à l'écran, mais dans le tampon de sortie.
- L'affichage sur l'écran accompagné du vidage du tampon de sortie se fait dans deux cas :
  - soit quand le caractère `'\n'` doit être affiché ;
  - soit quand on utilise explicitement l'instruction fflush(stdout);

Pour éviter des confusions dues à une mauvaise synchronisation entre les entrées clavier et les affichages à l'écran, en phase de mise au point, il est conseillé de faire suivre les instructions d'affichage de l'instruction fflush(stdout); si la chaîne à afficher ne se termine pas par la caractère `'\n'`.

À vos boitiers !

15

À vos boitiers !

16

Sommaire

Introduction

Deux types de fichiers

Tous les fichiers sont des suites d'octets.

- Fichiers de texte : chaque octet qui les compose représente un caractère et ces caractères sont organisés en lignes (le contenu du fichier apparaît de manière lisible dans la fenêtre d'un éditeur de texte).
- Fichiers binaires : ce sont tous les autres fichiers (leur contenu apparaît sous la forme de caractères illisibles ou partiellement lisibles dans la fenêtre d'un éditeur).

Différence entre les deux types de fichiers : manière dont les informations y sont codées.
Exemple : l'entier 24 est codé par :

- `00110010 00110100` dans un fichier de texte : un octet pour le code ASCII du caractère `2`, qui vaut 50, et un octet pour le code ASCII du caractère `4`, qui vaut 52 ;
- `00000000 00000000 00000000 00011000` dans un fichier binaire si les entiers sont codés sur 4 octets.

Entrées-sorties dans les fichiers

Utilisation de la bibliothèque standard

- Nécessité de la directive `#include <stdio.h>`.
- En général, on n'utilise pas les mêmes fonctions pour réaliser des entrées-sorties dans les fichiers de texte et dans les fichiers binaires.

Ouverture (1/2)

Identificateur de fichier

La manipulation d'un fichier va se faire à travers un identificateur de fichier (aussi appelé *flux* ou *stream*) de type `FILE *`.

Fonction `fopen`

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

ouvre le fichier de désignation *nom* dans le mode *mode* et retourne son identificateur ou `NULL` s'il y a un problème.
Le *mode* est une chaîne de 1, 2 ou 3 caractères :

- le premier caractère indique si le fichier doit être ouvert en lecture (`r` pour *read*), en écriture (`w` pour *write*) ou en écriture en fin de fichier (`a` pour *append*) ;
- le deuxième caractère, qui peut être ajouté au premier, est le signe `+` qui précise que l'on souhaite effectuer une mise à jour (lecture et écriture) ;
- le troisième caractère indique si le fichier doit être considéré comme un fichier de texte (`t` pour *text*), qui est le type par défaut, ou comme un fichier binaire (`b` pour *binary*).

Ouverture (2/2)

Exemple

```
FILE *id_fich;
id_fich=fopen("data.txt","rt");
```

Unités standard

Trois identificateurs de fichiers particuliers sont disponibles et n'ont besoin ni d'être ouverts, ni d'être fermés : `stdin` (entrée standard), `stdout` (sortie standard) et `stderr` (sortie standard des erreurs).

Lecture (1/4)

Lecture dans un fichier de texte

- Lecture d'un caractère :

  `int fgetc(FILE *id_fich)`

  - lit le caractère courant dans le fichier d'identificateur *id_fich* ;
  - retourne ce caractère ou `EOF` si la fin du fichier est atteinte.

  Remarque : la macro `getc` effectue la même chose que `fgetc` et la macro `getchar` effectue la même chose que `fgetc(stdin)`.

- Lecture d'une chaîne de caractères :

  `char *fgets(char *ch, int n, FILE *id_fich)`

  - lit au maximum $n-1$ caractères dans le fichier d'identificateur *id_fich*, s'arrête si elle rencontre le caractère `'\n'`, les range (y compris l'éventuel `'\n'`) dans la chaîne d'adresse *ch* en complétant par le caractère `'\0'` ;
  - retourne l'adresse de la chaîne ou `NULL` s'il y a un problème ou si la fin de fichier a été atteinte.

- Lecture formatée :

  `int fscanf(FILE *id_fich, const char *format[,liste_d_expr])`

  se comporte de la même manière que `scanf` mais effectue la lecture dans le fichier d'identificateur *id_fich* au lieu du clavier.

Lecture (2/4)

Lecture dans un fichier de texte : exemple

```c
#include <stdio.h>

int main(void)
{
   FILE *f; /* Identificateur (interne) de fichier */
   char c1,c2,ch[81];
   int n;

   /* Ouverture du fichier */
   f=fopen("fich1.txt","rt"); /* r : lecture, t : texte */

   c1=fgetc(f); /* c1 recoit 'a' */
   c2=fgetc(f); /* c2 recoit '\n' */
   fgets(ch,81,f); /* ch recoit "bc d\n" */
   fscanf(f,"%d",&n); /* n recoit 3794 */

   /* Fermeture du fichier */
   fclose(f);

   printf("%c%c%s%d\n",c1,c2,ch,n);
   return 0;
}
```

fich1.txt

```
a
bc␣d
3794
```

Lecture (3/4)

Accès à la taille d'un emplacement en mémoire

`sizeof` *`type`*

L'opérateur `sizeof` permet de connaître la taille en octets occupée par une variable d'un type donné. La valeur obtenue est de type `size_t`.

Lecture dans un fichier binaire

```
size_t fread(void *pt, size_t taille, size_t nb_infos,
             FILE *id_fich)
```

- lit dans le fichier d'identificateur *`id_fich`* *`nb_infos`* informations (blocs d'octets) de *`taille`* octets chacune et les range en mémoire à l'adresse *`pt`* (la zone mémoire doit avoir été préalablement allouée) ;
- retourne le nombre d'informations effectivement lues. Ce nombre peut être inférieur à *`nb_infos`* si la fin du fichier est atteinte.

Lecture (4/4)

Lecture dans un fichier binaire : exemple

```c
#include <stdio.h>

int main(void)
{
   FILE *f; /* Identificateur (interne) de fichier */
   float tab[100];
   int i,n;

   /* Ouverture du fichier */
   f=fopen("fich1.dat","rb"); /* r : lecture, b : binaire */

   n=fread(tab,sizeof(float),100,f);
   for (i=0;i<n;i++) printf("%f\n",tab[i]);

   /* Fermeture du fichier */
   fclose(f);

   return 0;
}
```

Écriture (1/5)

Écriture dans un fichier de texte

- Écriture d'un caractère :

  `int fputc(int c, FILE *id_fich)`

  - écrit dans le fichier d'identificateur *id_fich* le caractère *c* (convertit en `unsigned char`) ;
  - retourne le caractère écrit ou `EOF` s'il y a un problème.
- Écriture d'une chaîne de caractères :

  `int fputs(const char *ch, FILE *id_fich)`

  - écrit dans le fichier d'identificateur *id_fich* la chaîne d'adresse *ch* ;
  - retourne `EOF` s'il y a un problème ou une valeur négative dans le cas contraire.
- Écriture formatée :

  `int fprintf(FILE *id_fich, const char *format[,liste_d_expr])`

  se comporte de la même manière que `printf` mais effectue l'écriture dans le fichier d'identificateur *id_fich* au lieu de l'écran.

Écriture (2/5)

Écriture dans un fichier de texte : exemple

```c
#include <stdio.h>

int main(void)
{
   FILE *f; /* Identificateur (interne) de fichier */

   /* Ouverture du fichier */
   f=fopen("fich2.txt","wt"); /* w : Ecriture, t : texte */

   fputc('z',f);
   fputc('\n',f);
   fprintf(f,"Entier : %d\nFlottant : %f\n",12,3.14);
   fputs("Chaine\n",f);

   /* Fermeture du fichier */
   fclose(f);

   return 0;
}
```

fich2.txt

```
z
Entier : 12
Flottant : 3.140000
Chaine
```

Écriture (3/5)

Affichage des messages d'erreur

- Avec la fonction `fprintf` :

```
FILE *f;
f=fopen("toto.txt","rt");
if (f==NULL)
  fprintf(stderr,"Fichier toto.txt inexistant.\n");
```

- Avec la fonction `perror` :

  `void perror(const char *message)`

  permet d'afficher sur `stderr` la chaîne passée en paramètre suivie du caractère `:`, suivi éventuellement du message associé à l'erreur (dépend du système d'exploitation).

```
FILE *f;
f=fopen("fich","rt");
if (f==NULL)
  perror("fich");
```

Si le fichier `fich` n'existe pas, `perror` va afficher un message ressemblant à :
`fich: No such file or directory`

Écriture (4/5)

Écriture dans un fichier binaire

```
size_t fwrite(void *pt, size_t taille, size_t nb_infos,
              FILE *id_fich)
```

- écrit dans le fichier d'identificateur *id_fich* *nb_infos* informations (blocs d'octets) de *taille* octets chacunes situées en mémoire à l'adresse *pt* ;
- retourne le nombre d'informations effectivement écrites ; ce nombre peut être inférieur à *nb_infos* dans le cas d'un disque saturé.

Écriture (5/5)

Écriture dans un fichier binaire : exemple

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     FILE *f; /* Identificateur (interne) de fichier */
6     int tab[5]={1,2,3,4,5};
7
8     /* Ouverture du fichier */
9     f=fopen("fich2.dat","wb"); /* w : Ecriture, b : binaire */
10
11    fwrite(tab,sizeof(int),5,f);
12
13    /* Fermeture du fichier */
14    fclose(f);
15
16    return 0;
17 }
```

Fermeture

Fermeture d'un fichier ouvert

```
int fclose(FILE *id_fich)
```

- ferme le fichier d'identificateur *id_fich* ;
- retourne 0 ou `EOF` s'il y a un problème.

Remarques

- Il ne faut fermer un fichier que s'il est ouvert.
- Il est conseillé de fermer un fichier dès que son traitement est terminé.

Gestion de la position courante (1/4)

« Pointeur de fichier »

- À chaque fichier ouvert est associé un « pointeur de fichier » qui donne la position courante dans le fichier, c'est-à-dire le rang du prochain octet à lire ou à écrire.
- Après chaque opération de lecture ou d'écriture, ce « pointeur » est automatiquement incrémenté du nombre d'octets transférés.
- On parle d'*accès séquentiel* au fichier.

Détection de fin de fichier

`int feof(FILE *id_fich)`

- retourne une valeur non nulle si la fin du fichier d'identificateur *`id_fich`* est atteinte, ou 0 sinon ;
-  la valeur de retour de `feof` n'est valide qu'après avoir effectué au moins une lecture ;
- dans le cas d'une lecture dans un fichier binaire, la valeur de retour de la fonction `fread` peut remplacer avantageusement l'appel à la fonction `feof`.

Gestion de la position courante (2/4)

Positionnement dans un fichier

Il est possible d'accéder à un fichier de manière directe (accès direct) en modifiant sa position courante grâce à la fonction :

```
int fseek(FILE *id_fich, long decalage, int depart)
```

- *id_fich* est l'identificateur du fichier ;
- *decalage* est le nombre d'octets dont on veut se décaler par rapport à la position *depart* ;
- *depart* peut être égal à :
  - SEEK_SET ou 0 : début de fichier,
  - SEEK_CUR ou 1 : position courante dans le fichier,
  - SEEK_END ou 2 : fin de fichier.

La fonction fseek retourne 0 si tout s'est bien passé ou -1 sinon, en particulier dans le cas où on essaie de revenir en arrière avant le début du fichier. Néanmoins, quand on essaie d'avancer au-delà de la fin du fichier, fseek retourne 0 (tout se passe comme si on « piétinait » à la fin du fichier).

Gestion de la position courante (3/4)

Positionnement dans un fichier (suite)

- `void rewind(FILE *id_fich)`
  place le pointeur du fichier d'identificateur *id_fich* à son début.
  `rewind(f);` $\Longleftrightarrow$ `fseek(f,0,SEEK_SET);`
- `long ftell(FILE *id_fich)`
  retourne la position courante du fichier d'identificateur *id_fich* ou -1 s'il y a un problème.

Gestion de la position courante (4/4)

Exemples

On suppose que f est l'identificateur d'un fichier ouvert en lecture.

- Mémorisation de la position courante :

```
long position;
position=ftell(f);
```

- Retour à une position mémorisée :

```
fseek(f,position,SEEK_SET);
```

- Détermination de la taille d'un fichier :

```
long taille;
fseek(f,0,SEEK_END);
taille=ftell(f);
```

- Accès direct au contenu d'un fichier binaire :

```
int Nieme(int n, FILE *f)
{
  int Entier;

  fseek(f,sizeof(int)*(n-1),SEEK_SET);
  fread(&Entier,sizeof(int),1,f);
  return Entier;
}
```

Suppression d'un fichier

Fonction `remove`

```
int remove(const char *nom_fichier)
```

- supprime le fichier de nom *nom_fichier* ;
- retourne 0 ou une valeur non nulle s'il y a un problème.

Conseils

Principales fonctions utilisées selon le type de fichier

	Lecture			**Écriture**	
Type fichier	*Mode ouverture*	*Fonction*	*Détection fin de fichier*	*Mode ouverture*	*Fonction*
Texte	`"rt"`	`fscanf`	`feof`	`"wt"` ou `"at"`	`fprintf`
Binaire	`"rb"`	`fread`	valeur de retour de `fread`	`"wb"` ou `"ab"`	`fwrite`

À vos boitiers !

17

À vos boitiers !

18