6. Lecture/écriture formatées et Fichiers

- printf / scanf
- Fichier:
  - Mémoire tampon
  - Accès aux fichiers
  - Modes d'ouverture
  - Lecture / écriture

Références

- **Site du zèro : Programmer en C**

http://www.siteduzero.com/tutoriel-3-14189-apprenez-a-programmer-en-c.html

  - Téléchargeable en PDF, bon complément au cours
  - Plus facile d'accès mais moins complet que le cours

- Cours de F. Fabre : http://www.ltam.lu/cours-c/

Remarque : les subtilités de scanf présentées ici peuvent vous aider en TP et ne serviront pas de questions pièges pour l'examen.

printf

- **printf** transfère du texte vers 1 fichier spécial :

  stdout, la sortie standard (par défaut l'écran)

- **Prototype**: `int printf ("format", exp1, exp2, … );`

  - Valeur de retour: - nombre de caractères imprimés (sans le "\0" si chaîne)
    -nombre négatif si erreur.

  - "format": une chaine de caractère comprenant
    -le texte à afficher tel quel à l'écran ('\n' : char pour newline)
    - les spécificateurs de format de conversion commençant par %

    pour les expressions exp1, exp2, …

Note: il doit y avoir autant de spécificateur que d'expressions en paramètres

printf

- **Spécificateur de format (conversion)** :

  % + un caractère indiquant le format de conversion pour l'affichage du contenu de la variable.

spécificateur	affichage
%i, %d	un entier (int)
%u	un entier non signé (unsigned int)
%x	en entier en hexadécimal
%f, %e	un réel (float)
%lf, %le	un réel long (double)
%c	un caractère (char)
%s	une chaîne de caractères (char[] ou char *)

- %s prend en argument l'adresse de début (ou idem. le nom) de la chaîne. Le caractère nul de fin '\0' n'est pas imprimé.

printf

- Largeur du champ d'impression et précision :
  - `%(largeur).(précision)symbol_conversion`

  - Exemples:

    - %10d : affiche au moins sur un espace de 10 caractères
      (ajoute des espaces blancs si nécessaire)
    - %.1f : affiche le réel avec 1 seul chiffre derrière la virgule
    - %.5s : affiche 5 caractères de la chaîne
      (ou moins si la chaîne est plus petite)
    - %12.5s : affiche 5 caractère et 8 blancs (car largeur de 12 caractères)

printf

```
void main ()
{
    int i = 65, ret;
    double x = 34.06;
    char n[] = "je vais au bois";

    ret = printf("%i %x %c \n", i,i,i);
    ret = printf("%lf %.1lf \n", x,x);

    printf("code retour: %i\n", ret);
    printf("code retour: %8i\n", ret);

    ret=printf("%.5s | %s\n", n,n);
}
```

- 65 41 A
- 34.060000 34.1

- code retour: 16
- code retour: 16

- je va | je vais au bois

scanf

- **scanf** interprète un texte à partir d'un 1 fichier spécial :
  stdin, l'entrée standard (par défaut le clavier)
  puis mémorise les données reçues aux adresses indiquées

- **Prototype**: `int scanf ("format", &exp1, &exp2, … );`
  - Valeur de retour: - le nombre de conversion avec succès
    (judicieux de le tester)
  - "format": une chaine de caractère précisant comment interpréter les données reçues à stocker aux adresses &exp1, &exp2, …

scanf

- **Spécificateur de format (conversion)** :

  % + un caractère indiquant le format de conversion de la lecture.

spécificateur	lecture	type argument associé
%i, %d	d'un entier	int *
%x	un entier en hexadécimal	unsigned int*
%f, %e	un réel	float *
%lf, %le	un réel long	double*
%c	un caractère	char*
%s	une chaîne de caractères	char* ou char[]

- %s correspond à une chaîne de caractères sans espacement.

  L'affectation stoppe au premier espace.

  La chaîne de caractère associée doit être assez grande pour ajouter le caractère de fin '\0'.

scanf

- Largeur du champ de lecture:
  - `%(largeur)symbol_conversion`
  - précise la largeur maximale du champ (le nombre de caractères) à évaluer pour la conversion
  - **Attention**: le reste du champs reste dans le buffer du clavier et est attribué à la prochaine variable !

  - Exemples:
    - %5d : n'attribue à la variable entière que les 5 premiers chiffres lus !
    - %3f : n'attribue au réel que les 3 premiers caractères lus (dont le '.', exemple: 2.3 pour 2.345)
    - %30s : n'attribue à la chaine de caractères que 30 caractères au maximum

scanf :

- Caractères d'espacement (espace , nouvelle ligne ou tabulation) précisé dans le format:

  correspond à un ou plusieurs caractères d'espacement quelconques dans la saisie.

- Exemples:

```
int day, month, year;
scanf( " %i  %i\n%i" , &day, &month, &year);
```

fonctionne correctement avec les saisies suivantes au clavier:

25 12 1978 ↵

ou

25 12 ↵
1978 ↵

ou encore

25↵
12↵
1979↵

scanf :

- Caractères d'espacement:

  Note : un espacement à la fin du format est une mauvaise idée.

~~`scanf("%i ")`~~ ou ~~`scanf("%i \n")`~~

- Autres caractères : la saisie au clavier doit les reproduire exactement

- Exemples:

```
int day, month, year;
scanf( "%i/%i/%i" , &jour, &mois, &year);
```

fonctionne correctement avec les saisies suivante au clavier:

25/12/1978↵ ou 25/012/01876↵

mais pas

~~25 12 1879↵~~ ou ~~25/ 12/1978↵~~

scanf : code de retour

- si le buffer du clavier est vide : attente de la validation d'une nouvelle saisie avec la touche return (↵)

  sinon lecture.

- lecture progressivement du buffer du clavier

  et comparaison avec le format annoncé

- arrêt : -lorsque toutes les données précisées dans le format sont reconnues.

  -dès qu'une donnée ne correspond pas au format annoncé

  (les dernières arguments reste alors non-initialisés)

Le code de retour indique le nombre de conversions effectivement réalisées.

scanf

```
void main ()
{
    int j, x, ret;
    char t[10];
    ret = scanf("%2i %i", &j, &x);
    printf("code: %i, lus: %i %i\n",
                          ret, j, x);

    ret = scanf("%s", t);
    printf("code: %i , t: %s\n",
                         ret, t);
}
```

23 3 mer ↵
code: 2, lus: 23 3
code: 1, toto

(2scanf à la suite: le premier n'a pas vidé le buffer)

234 6 mer↵
code : 2 , lus: 23 4
code : 1 , t: 6

(%2i -> le 4 de 234 reste dans le buffer)

23 ?3 mer ↵
code: 1, lus: 23 0
code: 1, t: ?3

(erreur 1er scanf)

getchar, puts et gets

- `int getchar ();` sans arguments

  - lire un caractère (même d'espacement) dans le buffer du clavier

- `int gets( char *s );`

  -lire une ligne (le buffer du clavier tant que le caractère '\n' n'est pas rencontré) et stocker la ligne à l'adresse de la chaîne.

  - '\0' est ajouté par la fonction à la fin de la chaîne

- `int puts( char *s );`

  -afficher la chaîne de caractère + un '\n' à l'écran

getchar

```
void main ()
{
   int j;
   scanf("%i", &j);

   /*get 1 char from keyboard
      after end of scanf*/

   j = getchar();
   printf("char %c (ascii %i)",
                          j, j);
}
```

23? ↵
char ? (ascii 63)

(? après que le scanf lit 23)

23↵
char
(ascii 10)

('\n' code ascii 10 :
le scanf laisse le \n de la touche ↵
dans le buffer du clavier après sa lecture de l'entier)

Fichiers

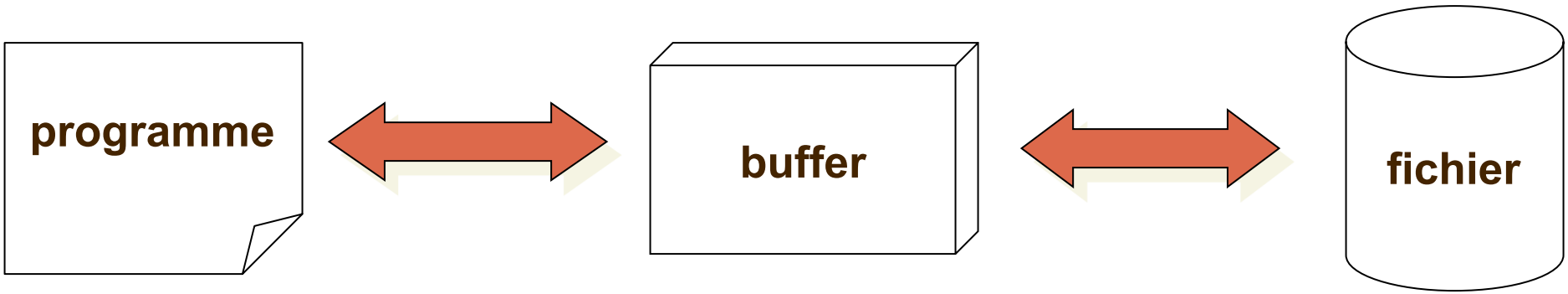
- 2 impératifs :
  - stockage permanent de l'information
  - stockage de grandes quantités d'information

Fichiers : Ensembles structurés de données stockées sur un support externe (suite d'enregistrements)

- On peut distinguer 2 types de fichiers :
  - **texte :** * organisé en lignes (séparées par '\n')
    * traduction 1 octet = 1 caractère
    * affichable sur écran
  - **binaire :** * séquence d'octets les uns derrière les autres sans traduction
    * adapté au données hétérogènes et structures complexes
    * gain de place mais risque d'incompatibilité entre machines (sizeof)

Accès aux fichiers

- Accès disques sont très lents comparés aux accès mémoire
- Important de limiter les accès physiques → "buffering"
- Mémoire tampon (buffer) est une zone de mémoire centrale où l'information est mémorisée temporairement



L'utilisation d'un buffer est transparent pour l'utilisateur et ne change rien à l'accès.

Accès aux fichiers

- Fichier est repéré par un nom externe sur le médium (disque dur)

`(/home/morane/fichiers/myfile.dat)`

Comment relier les instructions d' I/O avec le nom externe ?

- On passe par une structure de type FILE

  regroupe toutes les informations nécessaires au programme pour manipuler les données du fichier (voir <stdio.h>)

  - adresse de la mémoire tampon
  - position actuelle de la tête de lecture/écriture
  - type d'accès au fichier : lecture, écriture, …
  - état du fichier
  - …

Accès aux fichiers

- Une structure FILE doit être créée par `fopen(…)` lorsqu'on veut utiliser un fichier

- `f = fopen("nom_fichier" , "mode_ouverture")`
  - accepte en entrée un nom externe
  - négocie avec le système une structure FILE
  - initialise cette structure
  - rend un pointeur sur cette structure (NULL si erreur d'ouverture)

- Exemple : `FILE *f ;` (déclaration)

  `f = fopen("truc.dat", "w") ;` (ouverture)

- la commande `fclose(f)` annule cette liaison et ferme le fichier

Modes d'ouverture des fichiers

	Mode d'ouverture	Position après ouverture	Si fichier déjà existant	Si fichier non existant
`"r"`	**lecture seule**	**début**		**erreur**
`"w"`	**écriture seul**	**début**	**écrasement !**	**création**
`"a"`	**écriture seul**	**fin**		**erreur**
`"r+"`	lecture/écriture	début		erreur
`"w+"`	lecture/écriture	début	écrasement !	création
`"a+"`	lecture/écriture	fin		erreur

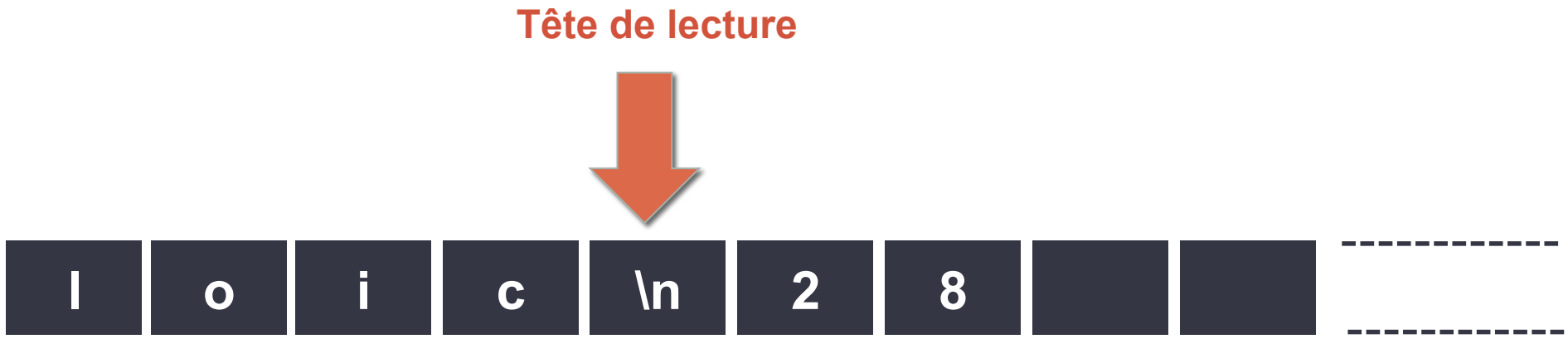
Nom du fichier

- Eviter les accents et espaces dans les noms de fichiers

- Chemin absolu : **“/home/etudiant/tp1/ex1.txt“**

- Chemin relatif (préférable) par rapport au répertoire ou est exécuté le programme :
  - **“ex1.txt“** ou **“./ex1.txt“** pour un fichier dans le répertoire courant
  - **“TP/ex1.txt“ ou “./TP/ex1.txt“** pour un fichier dans sous-répertoire TP
  - **“../ex1.txt“** pour un fichier dans le répertoire supérieur

Lecture / écriture séquentielle



- On ne peut lire ou écrire que sous la tête de lecture
- La tête de lecture avance après chaque lecture ou écriture sous la donnée suivante

Lecture / écriture en mode texte

- ligne par ligne
- caractère par caractère

- ligne par ligne avec (fichiers texte)
  - `int fprintf (file *F, "format", expr1, expr2,…);`
    Ecrit la chaîne de caractères dans le fichier **F**.
    Retourne le nombre de caractères écrits ou une valeur <0 si erreur
    Note: fprintf(stdout, ..) est la même chose que printf

  - `int fscanf (file *F, "format", &expr1, &expr2,…);`
    Lit les données dans le fichier **F** et les stocke aux adresses indiquées.
    Retourne le nombre de données lues avec succès.
    Note: fscanf(stdin, …) est la même chose que scanf

  - `int feof(file *F)` Renvoie une valeur différente de 0 si EOF (End Of File), la fin du fichier **F**, est atteinte.

Ecriture en mode texte

```
int main()
{
   FILE *F_out;
   char name[] = "loic";
   int age = 18, poids = 70;

   F_out=fopen("liste.txt","w");

   if (F_out ==NULL)
       {printf("Erreur"); return(-2);}

   fprintf(F_out, "%s %i %i\n",
                  name, age, poids);
   fprintf(F_out, "%s %i %i\n",
                  "eric", 56, 82);

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

Sort du programme si échec ouverture du fichier.

liste.txt:

```
loic 18 70
eric 56 82
```

Lecture en mode texte

```
int main()
{
   FILE *F_in;
   char name[];
   int age, poids;

   F_in = fopen("liste.txt","r");
   if (F_in == NULL)
            return(-2);

   while ( !feof(F_in) )
      { fscanf(F_in, "%s %i %i\n",
                  name, &age, &poids));
        printf("%s %i %i\n",
                  name, age, poids);
      }
   fclose(F_out);
   return(0);
}
```

liste.txt:

```
loic 18 70
eric 56 82
```

Tant que la fin du fichier n'est pas atteinte : lire et afficher le contenu

Lecture du même format que celui écrit.

> loic 18 70
> eric 56 82

Lecture / écriture en mode texte

- ligne par ligne
- caractère par caractère

- caractère par caractère (fichiers texte)
  - `int fputc (int c, file *F);`
    Ecrit le caractère (0-255) dans le fichier **`F`**.
    Retourne le caractère écrit ou EOF si une erreur.
  - `int fgetc (file *F);`
    Lit un caractère du fichier **`F`**.
    Retourne le caractère sous forme d'un entier ou EOF à la fin du fichier.
  - `int feof(file *F)`

Lecture en mode texte

```
#include <stdio.h>
int main()
{
  FILE *F_in;
  char c;

  F_in=fopen("liste.txt","r");
  if (F_in == NULL)
      {printf("erreur"); return(-2);}

  while ( (c=fgetc(F_in)) != EOF )
            printf("%c", c);

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

liste.txt:

```
loic 18 70
eric 56 82
```

Tant que c (le caractère lu) est différent de EOF :

> loic 18 70
> eric 56 82

Lecture / écriture en mode binaire

- Efficace surtout pour les tableaux et les structures complexes
- Principales fonctions (fichiers binaires)

  - `int fwrite (type *bloc, int taille, int nombre, FILE *F);`

    Ecrit **nombre** éléments de taille **taille** (octets) à partir de l'emplacement mémoire pointé par **bloc** dans le fichier **F**.

    Rend le nombre d'éléments écrits.

  - `int fread (type *bloc, int taille, int nombre, FILE *F);`

    Lit **nombre** éléments de taille **taille** (octets) dans le fichier **F**. Le résultat est stocké à l'emplacement pointé par **bloc**.

    Rend le nombre d'éléments lus (peut être <**nombre** ou 0 si eof)

  - `int feof (File *F);`

Ecriture en mode texte

```
#include <stdio.h>
int main()
{
    FILE *OUT;
    int tab[8] = {7,6,5,4,3,2,1,-1};

    OUT = fopen("tableau.txt","w");

    if (OUT == NULL)
    {printf ("erreur"); return(-2);}

    fwrite(tab,8,sizeof(int),OUT);

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

tableau.txt: (ouvert par un éditeur de texte)

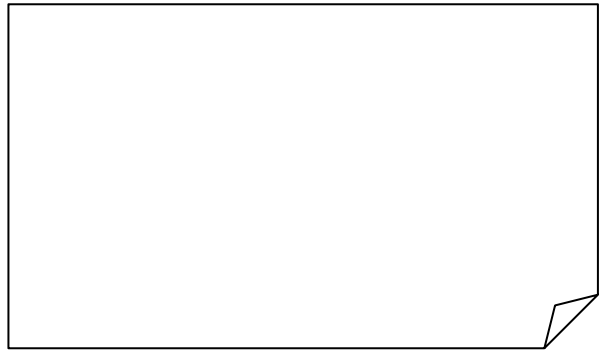
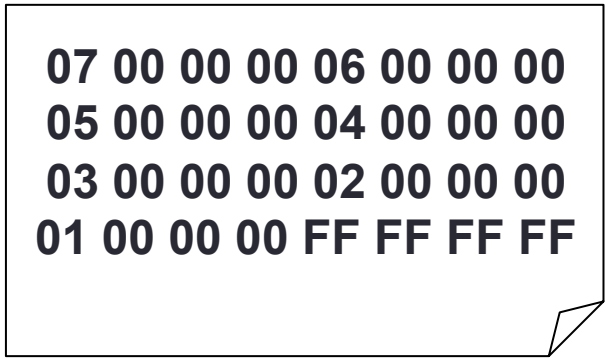


tableau.txt: (ouvert par un éditeur hexadécimal, + complément à 2)



Lecture en mode texte

```
int main()
{   FILE *IN;
    int i=0, n=10, *tab;
    tab=(int*)malloc(n*sizeof(int));

    IN = fopen("tableau.txt","r");
    /*ajouter test ouverture IN*/

    while (!feof(IN))
       { fread(tab+i,1,sizeof(int),IN);
                i++;
       }
    printf("tab[7]: %i",tab[7]);
    fclose(IN);
    return(0);
}
```

tableau.txt: (ouvert par un éditeur hexadécimal, + complément à 2)

```
07 00 00 00 06 00 00 00
05 00 00 00 04 00 00 00
03 00 00 00 02 00 00 00
01 00 00 00 FF FF FF FF
```

→ **tab[7] = -1;**