Formatage style langage C

- Formation et affichage d'une chaîne de caractères
  - **printf**(*format*, *variable_1*, *variable_2*, ...)
  - Affiche une chaîne de caractères formatée et composée des *variable_i*
  - Pour finir une ligne (créer une nouvelle), il faut utiliser le caractère spécial \n (c'est différent du *disp* où il y avait toujours un saut de ligne à la fin)
- Formation et renvoi d'une chaîne de caractères
  - *nouvelle_variable* = **sprintf**(*format*, *variable_1*, *variable_2*, ...)
  - renvoie une chaîne de caractères formatée et composée des *variable_i*
- Emprunté au C dans beaucoup de langages (Matlab, PHP, Java...)
- Le *format* est une chaîne de caractères qui définit comment composer la chaîne finale à partir des *variable_i*
- On se sert d'un caractère spécial, le pour cent (%), pour montrer où et comment insérer les valeurs des *variable_i* consécutives dans le *format* pour composer la chaîne
- L'ordre des % doit être le même que celui des *variable_i*

Spécification du format (1)

- Les caractères qui suivent le % définissent :
  1) comment interpréter une *variable_i* (nombre, caractère, chaîne...)
  2) comment formater la valeur de cette variable (point décimal, nombre de chiffres ...), c.-à-d. comment la transformer en chaîne de caractères
- Principales options :
  - **%d**
    nombre sans point décimal
    - printf("%d", 45) → 45
    - printf("%d", 45.98) → 45
  - %*n*d
    n – nombre de chiffres ; s'il y en a moins, des espaces sont insérées au début
    - printf("%6d", 45) → ····45
  - %0*n*d
    zéros ajoutés au lieu d'espaces
    - printf("%06d", 45) → 000045

Spécification du format (2)

- **%f**
  nombre avec point décimal
  - printf("%f", 45) → `45.000000`
  - printf("%f", 45.98) → `45.980000`
- %.*m*f
  m – nombre de chiffres après le point décimal
  - printf("%.1f", 45.98) → `46.0`
  - printf("%.0f", 45.98) → `46` (notez la différence avec %d)
- %*n*.*m*f
  n – nombre de caractères total (y compris le point décimal)
  m – nombre de chiffres après le point décimal
  - printf("%6.1f", 45.98) → `··46.0`
- **%e**, %.*m*e, %*n*.*m*e
  nombre sous la forme de « 0.0e+0 »
  - printf("%e", 45.98) → `4.598000e+001` ($4{,}598 \cdot 10^{1}$)
  - printf("%10.2e", 45.98) → `·4.60e+001`
  - %E pour avoir le « E » en majuscule

Spécification du format (3)

- pour le signe (peut être utilisé avec %d, %f et %e) :
  - (espace) – réserver la place pour un minus
    printf("% f", 45.98) → ·45.980000
    printf("% f", -45.98) → -45.980000
  - \+ – insérer un plus si le nombre est positif
    printf("%+f", 45.98) → +45.980000
- **%c**, *%n*c
  caractère (donnée alphanumérique)
  - printf("%c", "A") → A
    printf("%c", 65) → A
    printf("%d", "A") → 65
- **%s**, *%n*s
  chaîne de caractères
  - printf("%6s", "ABC") → ···ABC
- pour insérer le caractère « % » même : **%%**

- Formatage composé :
  - > printf("L'interet de **%d**%% vaut **%.2f** euros.\n", **40**, **625**)
    L'interet de **40**% vaut **625.00** euros.

Formatage style langage C d'entrée

- *variable* = **sscanf**(*chaîne*, *format*)
  - Obtenir la valeur de la *variable* à partir de la *chaîne* de caractères, suivant le *format* qui aussi définit le type de la *variable*
  - ```
    > a = "3.5"
    > n = sscanf(a, "%f")
    n = 3.5000
    > n+1
    ans = 4.5000
    ```
  - ```
    > n = ...
    sscanf("3.5 6.21", "%f %f")
    n =  3.5000
         6.2100
    ```
  - Normalement, on ne précise pas le nombre de caractères (comme %6.2f)
- [*valeurs*, *nombre_lues*] = ... sscanf(*chaîne*, *format*)
  - ```
    > [n,nd] = ...
    sscanf("3.5 6.21", "%f %f")
    n =  3.5000
         6.2100
    nd = 2
    ```
  - ```
    > [n,nd] = ...
    sscanf("3.5", "%f %f")
    n =  3.5000
    nd = 1
    ```
- La forme du résultat dépend fortement du langage de programmation spécifique
  - Normalement c'est plus compliquée qu'en Matlab p. ex. en C original

Fichiers

- Un fichier est un ensemble de données structuré, portant un nom et enregistré sur un support
  - support = disque dur, mémoire flash...
- Un fichier sert à
  - mémoriser des résultats de façon permanente sous une forme toujours compréhensible pour l'ordinateur
- pour
  - ne pas surcharger la mémoire
  - garder les résultats pour le futur
  - traiter les résultats avec un autre programme
  - échanger les résultats
- ou parce que
  - elles ont été obtenues hors un ordinateur
    - elles existent alors sous la forme écrite, imprimé, dessiné, photographié traditionnellement...
    - le support serait normalement un papier
    - la forme n'est pas compréhensible pour l'ordinateur
  - elles ont été obtenues sur un autre ordinateur
    - elles doivent alors être transmises

Les deux types de base de fichiers

- Fichier binaire
  - les nombres peuvent être enregistrés directement alors plus vite
  - occupe moins d'espace
  - incompréhensible pour l'humain
  - nombres directement compréhensibles pour l'ordinateur
  - traitable seulement avec un programme qui connaît la méthode du codage des données (le format du fichier)
- Fichier texte
  - les nombres enregistrés comme texte
  - occupe plus d'espace
  - compréhensible pour l'humain
  - conversion texte ⟶ nombre nécessaire pour l'ordinateur alors plus lent
  - importation possible (document texte, classeur...)
  - bon pour l'échange entre des programmes qui ne connaissent pas leurs formats de fichiers

Gestion de fichiers en Matlab

- Modèle très pareil au langage C et apparentés
- Séquence typique d'actions :
  - ouverture – premier accès qui rend possible la lecture et l'écriture
  - lecture – obtention des données
  - écriture – enregistrement (ajout ou modification) des données
  - fermeture – fin d'accès
- On ne peut effectuer aucune autre action avant l'ouverture
- Après fermeture, l'unique action qui peut être effectuée est une nouvelle ouverture
- Dès l'ouverture jusqu'à la fermeture, la lecture et/ou l'écriture peuvent s'effectuer n'importe combien de fois et dans n'importe quelle ordre

Ouverture et fermeture

- Ouverture
  - [*no_fichier*, *mess_erreur*] = **fopen**(*nom_fichier*, *mode*)
  - si une erreur arrive, *no_fichier* ⟵ –1 et un message est renvoyé
- Modes d'accès :
  - "r" – lecture seulement
  - "w" – écriture seulement, le contenu présent est effacé *
  - "a" – écriture à la fin du contenu présent *
  - "r+" – lecture et écriture
  - "w+" – lecture et écriture, le contenu présent est effacé *
  - "a+" – lecture ou écriture à la fin du contenu présent *
  - * le fichier est créé s'il n'existe pas
  - minimisent le risque d'effacement ou modification non volontaires
- Le fichier sera reconnu par son nombre renvoyé par *fopen* et qui doit alors être enregistré dans une variable
- Fermeture
  - **fclose**(*no_fichier*)

Écriture

- Écriture simple
  - **fputs**(*no_fichier*, *chaine_caracteres*)
  - un argument seulement, une chaîne de caractères
- Écriture avec formatage
  - **fprintf**(*no_fichier*, *format*, *variable1*, *variable2*, ...)
  - comme *printf* et *sprintf*
  - le résultat enregistré est une chaîne de caractères
- Écriture binaire
  - **fwrite**(*no_fichier*, *matrice_donnees*, *precision*)
  - *precision* = combien de bits pour chaque élément (optionnel)

- Exemple
  - ```
    f = fopen("a.txt", "w");
    fputs(f, "La ligne 1 est plus longue\n");
    fputs(f, "Ligne 2");
    fclose(f);
    ```
  - Le fichier nommé *a.txt* dans le répertoire de travail en cours est ouvert et son contenu est effacé (ou bien il est créé)
  - Son contenu est :
    ```
    La ligne 1 est plus longue
    Ligne 2
    ```
  - Ne pas oublier le caractère de fin de ligne *\n* ou *\r\n* (selon le système d'exploitation)
    ```

Lecture simple

- Lecture simple par ligne
  - *ligne* = **fgetl**(*no_fichier*, *nombre_caracteres*)
  - Lit une ligne du fichier portant le numéro *no_fichier*
  - *nombre_caracteres* est facultatif ; s'il est passé, la lecture est finie au plus tard après un tel nombre de caractères (si la ligne ne finit avant)
  - Le « \n » rencontré à la fin de la ligne n'est pas renvoyé (contrairement à certains autres langages)
  - Le nombre –1 est renvoyé s'il n'y a plus de données dans le fichier (fin du fichier)
- Exemple
  - On assume le fichier *a.txt* créé précédemment
  - ```
    > f = fopen("a.txt", "r");
    > s1 = fgetl(f, 18)
    s1 = La ligne 1 est plu
    > s2 = fgetl(f, 18)
    s2 = s longue
    > s3 = fgetl(f, 18)
    s3 = Ligne 2
    > s4 = fgetl(f, 18)
    s4 = -1
    > fclose(f);
    ```
- Afin de lire tout le contenu du fichier
  - on utilisera une boucle *do/until* avec *s==-1* comme condition
    ```

Lecture avec formatage

- Permet d'obtenir d'une manière simple des valeurs numériques
- Tout comme pour formatage d'entrée de chaînes de caractères, la forme du résultat dépend fortement du langage de programmation particulier
  - Les formes les plus faciles à utiliser sont propres à Matlab
- Forme langage C propre
  - [*variable1*, *variable2*, ..., *nb_lues*] = **fscanf**(*no_fichier*, *format*, "C")
  - Les *variable1*, *variable2* etc. sont remplis avec les valeurs consécutives trouvées dans le fichier selon le *format* donné
  - *nb_lues* est le nombre de valeurs lues avec succès

Lecture avec formatage (suite)

- Forme matricielle (propre à Matlab)
  - [*matrice_valeurs*, *nb_lues*] = **fscanf**(*no_fichier*, *format*, *taille*)
  - Au lieu de plusieurs variables, une matrice contenant plusieurs valeurs sera remplie
  - La *taille* peut être :
    - Inf – lecture jusqu'à la fin, les valeurs étant mises dans une seule colonne
    - *nombre* – lecture du *nombre* de valeurs donné, les valeurs étant mises dans une seule colonne
    - [*nombre1*, *nombre2*] – lecture ligne par ligne, enregistrement dans la matrice colonne par colonne ; *nombre1* d'éléments par colonne de la matrice (ou par ligne du fichier) ; *nombre2* de colonnes (ou lignes) maximum
    - [*nombre1*, Inf] – comme le précédent mais sans limiter le nombre de colonnes de la matrice (ou lignes du fichier)
- Cette forme d'appel n'existe pas en C original

Lecture avec formatage – exemple

- Contenu du fichier *f2.txt* :
  - ```
    1 5.2543 1.2e-14
    2 4.1311 3.5e-15
    3 3.8723 1.8e-15
    4 3.2367 4.5e-14
    ```
- Lecture :
  - ```
    f = fopen("f2.txt", "r");
    [m, nbElem] = fscanf(f, "%d %f %e", [3,Inf])
    fclose(f);
    ```
  - ```
    m =
      1.0000e+000   2.0000e+000   3.0000e+000   4.0000e+000
      5.2543e+000   4.1311e+000   3.8723e+000   3.2367e+000
      1.2000e-014   3.5000e-015   1.8000e-015   4.5000e-014
    nbElem =  12
    ```
  - La ligne 1 du fichier devient la colonne 1 de la matrice
  - La colonne 1 du fichier devient la ligne 1 de la matrice
- Cas particulier : format %c du premier élément dans une ligne
  - Il faut marquer la fin de ligne dans le format ("%c ... \n") pour que le caractère « nouvelle ligne » ne soit pas traité du premier élément de la série suivante
    ```

Lecture avec formatage – le paramètre « taille »

- Avec *taille* = Inf :
  - [m, nbElem] = ...
    fscanf(f, "%d %f %e", Inf)
  - m =
    1.0000e+000
    5.2543e+000
    1.2000e-014
    2.0000e+000
    4.1311e+000
    3.5000e-015
    3.0000e+000
    3.8723e+000
    1.8000e-015
    4.0000e+000
    3.2367e+000
    4.5000e-014
    nbElem = 12
- Ligne par ligne :
  - une ligne chaque fois *fscanf* est appelée
  - mettre *nombre2* = 1
  - [m, nbElem] = ...
    fscanf(f, "%d %f %e", [3,1])
  - m =
    1.0000e+000
    5.2543e+000
    1.2000e-014
    nbElem = 3

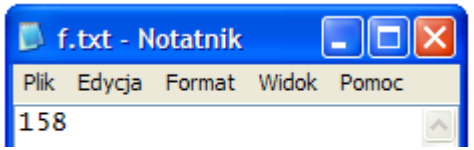
Lecture avec formatage – façon langage C

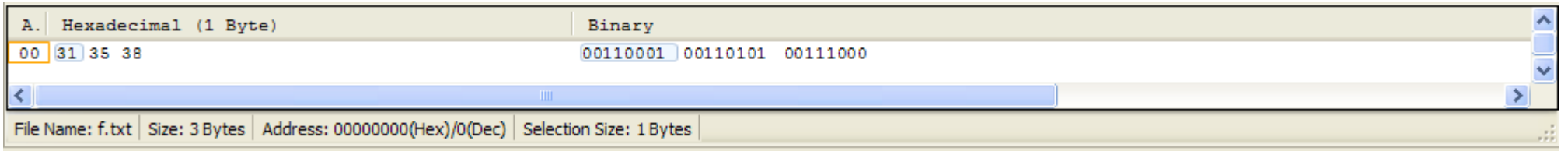
- Autant de valeurs sont lues (au maximum) qu'il y a de variables à gauche de l'affectation (*noElem* exclu)
  - [a, b, c, nbElem] = fscanf(f, "%d %f %e", "C")
  - a = 1
    b = 5.2543
    c = 1.2000e-014
    nbElem = 3
- Le nombre des variables (*noElem* exclu) doit être égal au nombre des %
  - [a, b, c, d, g, h, nbElem] = fscanf(f, "%d %f %e %d %f %e", "C")
  - a = 1
    b = 5.2543
    c = 1.2000e-014
    d = 2
    g = 4.1311
    h = 3.5000e-015
    nbElem = 6
  - le découpage du fichier en lignes n'a aucune influence
- C'est pareil en C original et d'autres langages

Fichiers texte et fichiers binaires – comparaison des résultats de l'enregistrement

- Enregistrement du nombre 158
  - dans un fichier texte

```
fTxt=fopen("f.txt","w");
fputs(fTxt,"158");
fclose(fTxt);
```

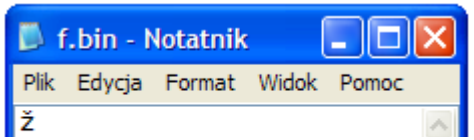


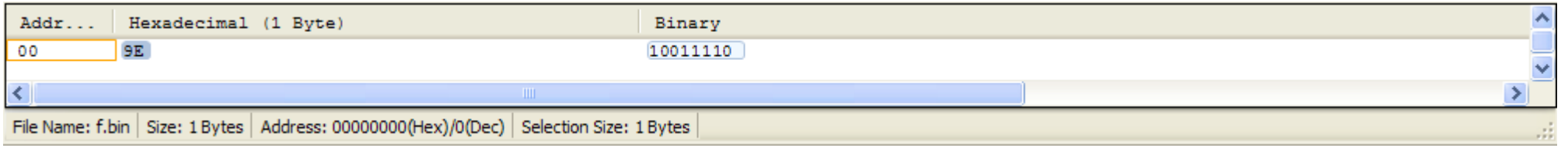


  - on voit le nombre en Bloc-notes
  - mais pour l'ordinateur ce sont 3 nombres à 1 o (3 o en total)
  - le Bloc-notes suppose que ce sont des codes de caractères

  - dans un fichier binaire

```
fBin=fopen("f.bin","w");
fwrite(fBin,158);
fclose(fBin);
```





  - pour l'ordinateur c'est resté le nombre 158 (9E) qui occupe 1 o
  - mais le Bloc-notes affiche un caractère bizarre : c'est « ž », la lettre qui porte le code 158