heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Analyse et programmation 2

# Le traitement de fichiers

heig-vd

## Thèmes abordés

- Applications.
- Les fichiers au format texte.
  - Création, ouverture, lecture, écriture,…
  - Fonctions particulières pour les fichiers texte.
- Les fichiers binaires.
  - Fonctions particulières pour les fichiers binaires.
  - L'alignement des structures.
  - Retour sur la fonction main : argc et argv.

heig-vd

## Le traitement de fichiers

Applications qui consomment ou produisent des fichiers



heig-vd

## Texte ou binaire

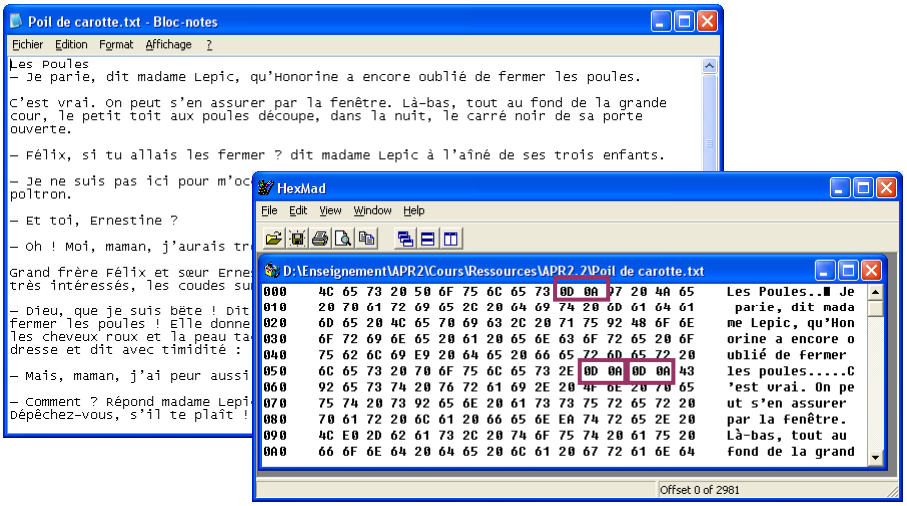
Définitions

- Un fichier texte
  - Utilise les lettres, les chiffres, les signes de ponctuation.
  - Représente l'information sous forme textuelle.
  - Souvent, le contenu est intelligible pour nous.
- Un fichier binaire
  - Contient directement des valeurs numériques.
  - Une application particulière est utilisée pour rendre le contenu intelligible.

heig-vd

## Texte ou binaire

Exemple – un fichier texte observé avec NotePad et HexMad

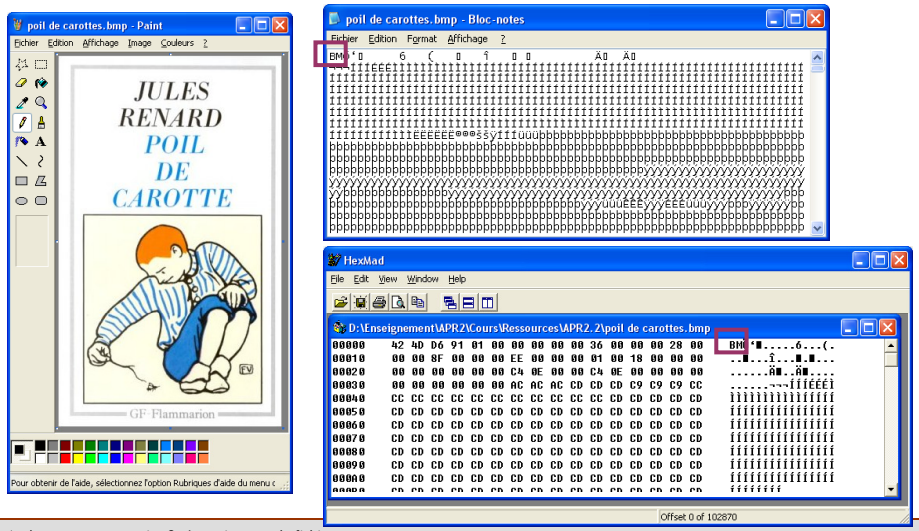





heig-vd

## Texte ou binaire

Exemple – un fichier binaire observé avec NotePad, Paint et HexMad



heig-vd

## Traitement des fichiers texte en C
Les fonctions

- La bibliothèque du langage C
  - Offre des fonctions standards pour la gestion de fichiers.
  - Rassemblées pour la plupart dans stdio.h
- Binaire ou texte ?
  - Certaines fonctions sont communes aux deux types.
  - D'autres sont plus spécifiques à un type de fichiers.



heig-vd

## Traitement des fichiers texte en C
Ouverture d'un fichier

- Pour pouvoir être utilisé, un fichier doit être « ouvert »
  - Cela permet au système d'exploitation de se préparer.
  - Chargement en mémoire d'informations nécessaires.
  - Fournir au programmeur un descripteur de fichier :
    - Pointeur désignant le fichier pour la suite du programme.
- Comment ouvrir un fichier : la fonction fopen

```
FILE* fichier; // Descripteur de fichier
fichier = fopen("Poil de carotte.txt", "r");

if (fichier != NULL) // ouverture réussie
```

chemin relatif ou absolu

mode

heig-vd

## Traitement des fichiers texte en C

### Ouverture d'un fichier – Chemin relatif et absolu

- Chemin absolu
  - On indique tous les répertoires depuis la racine.
  - Syntaxe dépendante du système d'exploitation.
  - Attention à la syntaxe du langage C : doubler les \\
  - Exemple :

```
"C:\\APR2\\Poil de carotte.txt"
```

- Chemin relatif
  - Relatif au « répertoire courant ».
  - En général, au début, c'est le répertoire contenant l'exécutable.
  - Il peut être changé par programme (fonction chdir).
  - Syntaxe particulière
    - Répertoire parent : `".."`
    - Répertoire courant : `"."`
  - Exemples :

```
"Poil de carotte.txt"
".\\Poil de carotte.txt"
"..\\Labo2\\Poil de carotte.txt"
```



heig-vd

## Traitement des fichiers texte en C

### Ouverture d'un fichier – les différents modes

| Mode | Effet |
|---|---|
| "r" | Ouverture d'un fichier texte en lecture. |
| "w" | Création d'un fichier texte, ouverture en écriture.<br>Si le fichier existait déjà, il est écrasé. |
| "a" | Ouverture d'un fichier texte en écriture, positionnement à la fin (compléter).<br>Si le fichier n'existait pas, il est créé. |
| "r+" | Ouverture d'un fichier texte en lecture/écriture. |
| "w+" | Création d'un fichier texte, ouverture en lecture/écriture. |
| "a+" | Ouverture d'un fichier texte en lecture/écriture, écriture à partir de la fin. |

Peu utilisé, fonctionnement un peu déroutant

heig-vd

## Traitement des fichiers texte en C
Lecture dans un fichier texte

- Plusieurs fonctions de lecture pour le texte :
  - fgetc : lit un caractère.
    ```
    int fgetc(FILE* file);
    ```
  - fgets : lit une ligne de texte.
    ```
    char* fgets(char* buffer, int max_count, FILE* file);
    ```
    - Retourne buffer en cas de succès, NULL en cas d'échec.
    - Place au maximum max_count caractères dans buffer.
    - S'arrête à la fin d'une ligne.
  - fscanf : lecture selon une spécification de format
    ```
    int fscanf(FILE* file, const char* format, ...);
    ```
    - Fonctionne comme scanf, mais lit depuis un fichier.



heig-vd

## Traitement des fichiers texte en C
Détection de la fin d'un fichier texte

- Accès séquentiel
  - Le contenu du fichier est lu dans l'ordre, de façon séquentielle.
  - A partir de la position courante.
  - On finit par atteindre la fin du fichier.
- Comment détecter la fin de fichier par programme
  - Fonction spéciale feof (File End Of File)
    ```
    int feof(FILE* file);
    ```
    - Retourne
      - 0 si la fin de fichier n'est pas atteinte.
      - Une valeur différente de 0 quand la fin de fichier est atteinte.

heig-vd

## Traitement des fichiers texte en C

Fermeture d'un fichier texte

- Tant que le fichier est ouvert
  - De la mémoire est utilisée pour le descripteur de fichier.
  - Le fichier est verrouillé :
    - il ne peut pas être détruit, renommé, écrit.
- Lorsque le traitement de fichier est terminé
  - Il faut refermer le fichier.
  - La fonction fclose est prévue à cet effet.

```
int fclose(FILE* file);
```

    - Renvoie 0 si la fermeture a réussi.
    - Renvoie != 0 en cas de problème.



heig-vd

## Traitement des fichiers texte en C

Problèmes lors de la lecture d'un fichier

- Un problème peut survenir durant la lecture
  - Fichier sur un média amovible:
    - on retire le média pendant la lecture, une erreur survient.
  - Fichier sur un disque dur:
    - Les secteurs contenant le fichier peuvent être altérés.
- Détecter les erreurs pendant la lecture
  - Il est vivement recommandé de le faire.
  - La fonction ferror est prévue à cet effet.

```
int ferror(FILE* file);
```

    - Renvoie != 0 si une erreur est survenue.
    - Renvoie 0 si tout va bien.

heig-vd

## Traitement des fichiers texte en C

Exemple typique pour la lecture d'un fichier texte

```c
#include <stdio.h>
#define LONGUEUR_MAX 80

int main()
{
    FILE* fichier;
    char ligne[LONGUEUR_MAX];

    fichier = fopen("Poil de carotte.txt", "r");
    if (fichier != NULL)
    {
        while (!feof(fichier))
        {
            fgets(ligne, LONGUEUR_MAX, fichier);
            if (!ferror(fichier))
                puts(ligne);
            else
            {
                printf("Erreur lors de la lecture du fichier\n");
                break; // poursuite de la lecture impossible, sortir
            }
        }
        fclose(fichier);
    }
    else
        printf("Erreur lors de l'ouverture du fichier.\n");
    printf("Pressez une touche...");
    _getch();
}
```

Que se passerait-il si on ne testait pas les erreurs ?



heig-vd

## Traitement des fichiers texte en C

Ecriture dans un fichier texte

- Plusieurs fonctions d'écriture pour le texte :
  - fputc : écrit un caractère.

    ```c
    int fputc(char c, FILE* file);
    ```

    - Retourne le caractère écrit, ou EOF (-1) en cas d'erreur.
  - fputs : écrit une ligne de texte et un saut de ligne.

    ```c
    int fputs(const char* text, FILE* file);
    ```

    - Retourne une valeur non négative en cas de succès, EOF si échec.
    - Ecrit une ligne et un saut de ligne dans le fichier.
  - fprintf : écriture selon une spécification de format

    ```c
    int fprintf(FILE* file, const char* format, ...);
    ```

    - Fonctionne comme printf, mais écrit dans un fichier.
    - Retourne le nombre de caractères écrits, une valeur négative si erreur

heig-vd

## Traitement des fichiers texte en C

Ecriture dans un fichier texte

- Particularités lors de l'écriture de fichiers
  - Les fonctions d'écriture préparent les données à écrire.
  - Elles ne sont pas nécessairement écrites immédiatement.
  - La fermeture du fichier force l'écriture sur le disque.
  - Si fclose retourne une erreur, il y a eu un problème d'écriture.
- Forcer l'écriture des données

```
int fflush(FILE* file);
```

  - Retourne 0 en cas de succès, EOF en cas d'erreur.



heig-vd

## Traitement des fichiers texte en C

Exemple typique pour l'écriture d'un fichier texte

```
#include <stdio.h>

int main()
{
    FILE * fichier;
    int i;

    fichier = fopen("Essai.txt", "w");
    if (fichier != NULL)
    {
        for (i = 0; i < 10; i++)
        {
            fprintf(fichier, "ligne %d\n", i);
            if (ferror(fichier))
            {
                printf("Erreur lors de l'ecriture.\n");
                break;
            }
        }
        if (fclose(fichier))
            printf("Le fichier n'a pas ete correctement ecrit.\n");
    }
    else
        printf("Erreur lors de l'ouverture du fichier.\n");
    printf("Pressez une touche...");
    _getch();
}
```

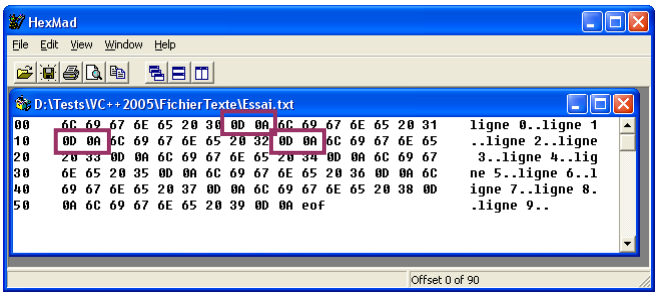
Que se passerait-il si:
- Le fichier était en lecture seule ?
- on ne testait pas les erreurs ?

heig-vd

## Traitement des fichiers texte en C

Gestion des fins de ligne

- Les fins de lignes dans les fichiers sous Dos/Windows
  - Elles sont marquées par la paire CR-LF (\r\n)
  - Que se passe-t-il lorsqu'on écrit '\n' simplement ?
- Les fonctions de fichier lisent et écrivent automatiquement les fins de ligne dans le format Windows.






heig-vd

## Traitement des fichiers textes en C

Les flux standards

- Il existe 3 flux standards
  - Utilisables avec les fonctions fichier.
  - stdin : fichier correspondant à la saisie dans la console.
  - stdout : fichier correspondant à l'affichage console.
  - stderr : flux de sortie pour les messages d'erreur, en général identique à stdout.
- `printf(format, …)` est équivalent à

```
fprintf(stdout, format, …);
```

heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé - XML

- Besoin important d'échanger de l'information structurée
  - Commande informatisée de pièces auprès d'un fournisseur.
  - Suivi automatisé de cotations boursières.
  - Envoi d'une trajectoire à une commande de robot.
  - Envoi de la liste des titres musicaux sur un lecteur MP3.
  - …



heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé - XML

- Base commune pour les formats texte
  - Normalisée par le w3c (world wide web consortium).
  - Première version en 1998.
- XML
  - eXtended Markup Language : langage à balises étendu.
- Nombreux outils disponibles pour exploiter ce format.
- Donc, il est intéressant à connaître…

heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé – XML – Codage des structures

- Exemple

```
<Vecteur>
<X>1000</X>
<Y>2000</Y>
<Z>3000</Z>
</Vecteur>
```

- Structure
  - Format purement textuel.
  - Valeurs encadrées par une balise ouvrante et fermante.
  - Hiérarchique.
  - Les balises peuvent être librement définies selon les besoins de l'application.
  - Les sauts de ligne et espaces sont non significatifs.
  - Un fichier doit contenir un nœud principal unique.



heig-vd

## Application répandue des fichiers texte

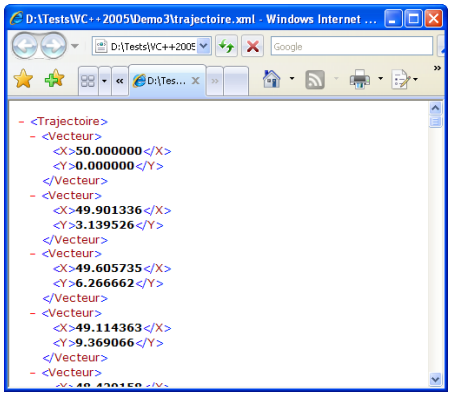
L'échange de données informatisé – XML – Codage des tableaux

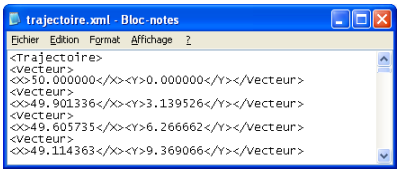
```
<Positions>
    <Vecteur>
        <X>1000</X>
        <Y>2000</Y>
        <Z>3000</Z>
    </Vecteur>
    <Vecteur>
        <X>1100</X>
        <Y>2100</Y>
        <Z>3100</Z>
    </Vecteur>
    <Vecteur>
        <X>1200</X>
        <Y>2200</Y>
        <Z>3200</Z>
    </Vecteur>
</Positions>
```

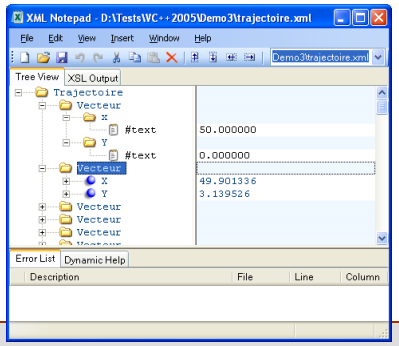
heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé – XML – Visualiseurs/Editeurs gratuits






heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé – XML – Générer par programme

- La génération de fichiers XML ne pose aucune difficulté.
- Il suffit de générer un fichier texte, selon les spécifications du format souhaité.
- Pas d'outil spécial nécessaire.

heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé – XML – Générer par programme

```
int main()
{
    const int NOMBRE_POINT = 100;
    const double RAYON = 50;
    const double CENTREX = 0.0, CENTREY = 0.0;
    int i;
    FILE *f;
    f = fopen("trajectoire.xml", "w");
    if (f != NULL)
    {
        fprintf(f, "<Trajectoire>\n");
        for (i = 0; i < NOMBRE_POINT; i++)
        {
            fprintf(f, "<Vecteur>\n");
            fprintf(f, "<X>%lf</X><Y>%lf</Y>\n",
                CENTREX + RAYON * cos(2 * M_PI * i / NOMBRE_POINT),
                CENTREY + RAYON * sin(2 * M_PI * i / NOMBRE_POINT));
            fprintf(f, "</Vecteur>\n");
        }
        fprintf(f, "</Trajectoire>\n");
        fclose(f);
    }
    system("PAUSE");
    return 0;
}
```

```
<Trajectoire>
- <Vecteur>
    <X>50.000000</X>
    <Y>0.000000</Y>
  </Vecteur>
- <Vecteur>
    <X>49.901336</X>
    <Y>3.139526</Y>
  </Vecteur>
- <Vecteur>
    <X>49.605735</X>
    <Y>6.266662</Y>
  </Vecteur>
- <Vecteur>
    <X>49.114363</X>
    <Y>9.369066</Y>
  </Vecteur>
- <Vecteur>
    <X>48.429158</X>
    <Y>12.434494</Y>
  </Vecteur>
- <Vecteur>
    <X>47.552826</X>
```



heig-vd

## Application répandue des fichiers texte

L'échange de données informatisé – XML – Lire par programme

- La lecture de fichiers XML est plus délicate.
- Bibliothèques logicielles spécialisées
  - XML Parser
- Simplification possible
  - Formater le XML pour en simplifier l'interprétation
  - Ex : 1 balise ou 1 valeur par ligne seulement.

```
<Positions>
<Vecteur>
<X>
1000
</X>
<Y>
2000
</Y>
```






## Traitement des fichiers binaires en C

Ouverture

- Principe similaire aux fichiers textes.
  - Ouverture du fichier avec fopen.
  - La gestion binaire est précisée dans le mode d'ouverture.
- 2 types d'accès
  - Accès séquentiel
    - Les éléments sont lus ou écrits l'un après l'autre dans le fichier.
  - Accès direct
    - Les éléments sont écrits à une position donnée.
    - La position courante d'écriture est déplacée en cours d'utilisation.
    - Application typique : les bases de données.

heig-vd

## Traitement des fichiers binaires en C

Ouverture d'un fichier – les différents modes

| Mode | Effet |
|---|---|
| "rb" | Ouverture d'un fichier binaire en lecture. |
| "wb" | Création d'un fichier binaire, ouverture en écriture. |
| "ab" | Ouverture d'un fichier binaire en écriture, positionnement à la fin (compléter). |
| "rb+" | Ouverture d'un fichier binaire en lecture/écriture. |
| "wb+" | Création d'un fichier binaire, ouverture en lecture/écriture. |
| "ab+" | Ouverture d'un fichier binaire en lecture/écriture, écriture à partir de la fin. |



heig-vd

## Traitement des fichiers binaires en C

Lecture et écriture

- Les fonctions de lecture écriture du mode texte
  - Sont utilisables avec les fichiers binaires.
- Quelle est finalement la différence entre binaire et texte ?
  - Interprétation de certains caractères en lecture
    - En mode texte : '\r' '\n' -> '\n'
    - CtrlZ (26) : en mode texte, représente la fin du fichier.
  - Interprétation de certains caractères en écriture
    - Écriture en mode texte : '\n' -> '\r' '\n'
    - Ecriture en mode binaire : '\n' -> '\n'

heig-vd

## Traitement des fichiers binaires en C
Ecriture

- fwrite
  - Permet d'écrire une ou plusieurs variables dans un fichier.
  - Ecriture exacte du contenu de la mémoire.
- Prototype

```
int fwrite(const void* data, int size, int count,
    FILE* file);
```

  - data : pointeur sur une ou plusieurs variables, de n'importe quel type.
  - size: taille d'un élément à écrire
  - count :nombre d'éléments à écrire.
  - file: fichier.
  - Valeur retournée :
    - le nombre d'éléments effectivement écrits, count si tout a bien fonctionné.



heig-vd

## Traitement des fichiers binaires en C
Ecriture – exemple typique

```
#include <stdio.h>

typedef struct
{
    char code;
    short taille;
    double prix;
} article;

int main()
{
    FILE * fichier;
    article article1 = { 'A', 42, 25.50 };

    fichier = fopen("article1.raw", "wb");
    if (fichier != NULL)
    {
        fwrite(&article1, sizeof(article), 1, fichier);
        if (fclose(fichier))
            printf("Le fichier n'a pas ete correctement ecrit.\n");
    }
    else
        printf("Erreur lors de l'ouverture du fichier.\n");
    printf("Pressez une touche...");
    _getch();
}
```

heig-vd

# Traitement des fichiers binaires en C

## Ecriture – exemple typique avec un tableau

```c
#include <stdio.h>

typedef struct
{
    char code;
    short taille;
    double prix;
} article;


int main()
{
    FILE * fichier;
    article articles[3] =
    { { 'A', 42, 25.50 },
      { 'A', 43, 26.50 },
      { 'B', 40, 32.00 } };

    fichier = fopen("articles.raw", "wb");
    if (fichier != NULL)
    {
        fwrite(articles, sizeof(article), 3, fichier);
        if (fclose(fichier))
            printf("Le fichier n'a pas ete correctement ecrit.\n");
    }
    else
        printf("Erreur lors de l'ouverture du fichier.\n");
    printf("Pressez une touche...");
    _getch();
}
```

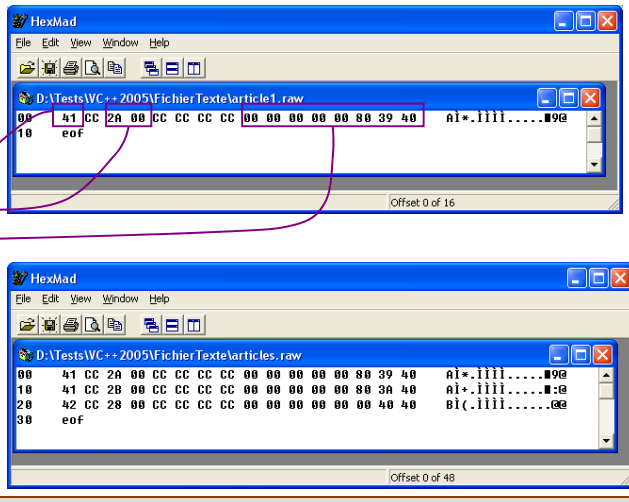


heig-vd

# Traitement des fichiers binaires en C

## Ecriture – analyse du résultat

```c
typedef struct
{
    char code;
    short taille;
    double prix;
} article

article article1 =
{'A', 42, 25.50};
```



heig-vd

# Traitement des fichiers binaires en C

## Alignement de structures

- Observation
  - On a écrit des informations supplémentaires non désirées.
  - Liées à l'alignement interne des variables sur des mots du processeur.
- Parades
  - Ecrire les données champ par champ.
    - Fastidieux.
  - Informer le compilateur qu'on ne souhaite pas aligner la structure sur les mots du processeur.
    - Dégradation des performances.



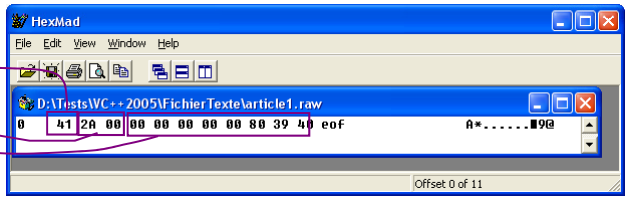
heig-vd

# Traitement des fichiers binaires en C

## Alignement de structures - solution 1: écrire champ par champ

```
. . .
    if (fichier != NULL)
    {
        fwrite(&article1.code, sizeof(char), 1, fichier);
        fwrite(&article1.taille, sizeof(short), 1, fichier);
        fwrite(&article1.prix, sizeof(double), 1, fichier);
        if (fclose(fichier))
            printf("Le fichier n'a pas ete correctement ecrit.\n");
    }
. . .
```

```
typedef struct
{
    char code;
    short taille;
    double prix;
} article

article article1 =
{'A', 42, 25.50};
```



heig-vd

# Traitement des fichiers binaires en C

## Alignement de structures – solution 2: forcer l'alignement sur des octets.

```
. . .
#pragma pack(push, 1) // Spécifique à Visual C++

typedef struct
{
    char code;
    short taille;
    double prix;
} article;

#pragma pack(pop) // Spécifique à Visual C++
. . .
    fwrite(&article1, sizeof(article), 1, fichier);
. . .
```

Mémorise les options d'alignement sur une pile
Force l'alignement des variables sur 1 octet.

Récupère les options d'alignement depuis la pile

```
typedef struct
{
    char code;
    short taille;
    double prix;
} article

article article1 =
{'A', 42, 25.50};
```




heig-vd

# Traitement des fichiers binaires en C

## Lecture

- fread
  - Permet de lire une ou plusieurs variables depuis un fichier.
  - Lecture du fichier et écriture directe dans la mémoire.
- Prototype

```
int fread(void* buffer, int size, int count,
    FILE* file);
```

  - buffer: pointeur sur la variable de destination, de n'importe quel type.
  - size: taille d'un élément à lire.
  - count :nombre d'éléments à lire.
  - file: fichier.
  - Valeur retournée :
    - le nombre d'éléments effectivement lus, count si tout a bien fonctionné, ou moins s'il y en avait moins de disponible.

heig-vd

## Traitement des fichiers binaires en C

### Lecture - exemple typique

```
#include <stdio.h>

#pragma pack(push, 1) // Spécifique à Visual C++

typedef struct
{
    char code;
    short taille;
    double prix;
} article;

#pragma pack(pop) // Spécifique à Visual C++

int main()
{
    FILE * fichier;
    article articles[3];

    fichier = fopen("articles.raw", "rb");
    if (fichier != NULL)
    {
        fread(articles, sizeof(article), 3, fichier);
        fclose(fichier);
    }
    else
        printf("Erreur lors de l'ouverture du fichier.\n");
    printf("Pressez une touche...");
    _getch();
}
```



heig-vd

## Traitement des fichiers binaires en C

### Gérer la position courante de lecture/écriture dans le fichier

- ftell
  `long ftell(FILE* file);`
  - Retourne la position courante par rapport au début du fichier.
  - Exprimée en octets par rapport au début, le premier = 0.
- fseek
  `int fseek(FILE* file, long offset, int origin);`
  - Change la position courante du fichier.
  - origin peut prendre le valeurs :
    - SEEK_SET : offset est donné par rapport au début du fichier.
    - SEEK_CUR: offset est donné par rapport à la position courante.
    - SEEK_END: offset est donné par rapport à la fin du fichier.
  - Résultat: 0 en cas de succès, différent de 0 sinon.
- rewind
  `void rewind(FILE* file);`
  - Remet la position courante au début du fichier.
  - Equivalent à : `fseek(file, 0L, SEEK_SET);`

heig-vd

## Traitement des fichiers binaires en C

Gérer la position courante de lecture/écriture dans le fichier

- fgetpos
  - Permet d'acquérir un marqueur sur la position courante.
  - Le contenu de fpos_t est privé (pas exploitable directement).

```
int fgetpos(FILE* file, fpos_t* pos);
```

- fsetpos
  - Permet de se repositionner sur un marqueur précédemment acquis.

```
int fsetpos(FILE* file, const fpos_t* pos);
```

- Exemple

```
fpos_t marqueur1;

fgetpos(fichier, &marqueur1);
. . .
fsetpos(fichier, &marqueur1);
```



heig-vd

## Traitement des fichiers en C

Autres opérations sur les fichiers

- Renommer

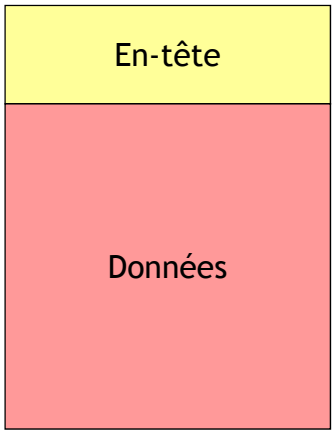
```
int rename(const char* filename, const char*
  new_filename);
```

- Supprimer

```
int remove(const char filename);
```

- Résultats
  - Ces fonctions renvoient 0 en cas de succès, != 0 sinon.
  - Le fichier ne doit pas être ouvert (empêche le renommage et la suppression)

## Application répandue des fichiers binaires
Les fichiers sonores (.wav) - structure








## Application répandue des fichiers binaires
Les fichiers sonores (.wav) – en-tête

| endian | File offset (bytes) | field name | Field Size (bytes) | |
|---|---|---|---|---|
| big | 0 | ChunkID | 4 | **The "RIFF" chunk descriptor**<br>The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data" |
| little | 4 | ChunkSize | 4 | |
| big | 8 | Format | 4 | |
| big | 12 | Subchunk1 ID | 4 | **The "fmt " sub-chunk**<br>describes the format of the sound information in the data sub-chunk |
| little | 16 | Subchunk1 Size | 4 | |
| little | 20 | AudioFormat | 2 | |
| little | 22 | NumChannels | 2 | |
| little | 24 | SampleRate | 4 | |
| little | 28 | ByteRate | 4 | |
| little | 32 | BlockAlign | 2 | |
| little | 34 | BitsPerSample | 2 | |
| big | 36 | Subchunk2ID | 4 | **The "data" sub-chunk**<br>Indicates the size of the sound information and contains the raw sound data |
| little | 40 | Subchunk2Size | 4 | |
| little | 44 | data | Subchunk2Size | |

heig-vd

# Application répandue des fichiers binaires

## Les fichiers sonores (.wav) – en-tête – représentation en C

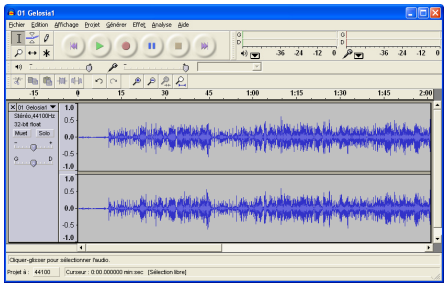
```
typedef struct
{
    char riff[4]; // must contain "RIFF"
    long length;  // must contain total length of file - 8
    char wav[4];  // must contain "WAVE"
} RIFF_HEADER;

typedef struct
{
    char fmt[4]; // must contain "fmt "
    long length; // must contain 0x10
    short audio_format; // must contain 1 for PCM
    short channel_number; // 1 = mono, 2 = stereo, ...
    long sample_rate; // in Hz
    long byte_rate; // == sample_rate * channel_number * BitsPerSample/8
    short bytes_per_sample; // 1=8 bit Mono, 2=8 bit Stereo or 16 bit Mono, 4=16 bit Stereo
    short bits_per_sample; // = 8, 16
} FORMAT_HEADER;

typedef struct
{
    char data[4] ; // must contain data
    long length; // length of data;
} DATA_HEADER;

typedef struct
{
    RIFF_HEADER riff;
    FORMAT_HEADER format;
    DATA_HEADER data;
} WAV_HEADER;

void wav_prepare_header(WAV_HEADER * header, int channel_number, int sample_rate_hz,
                        int sample_count, int bits_per_sample);
```

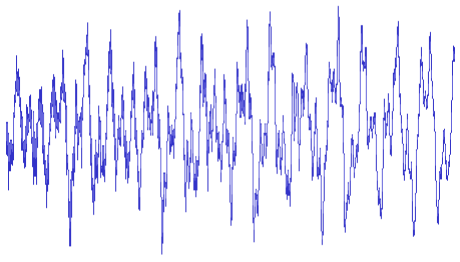


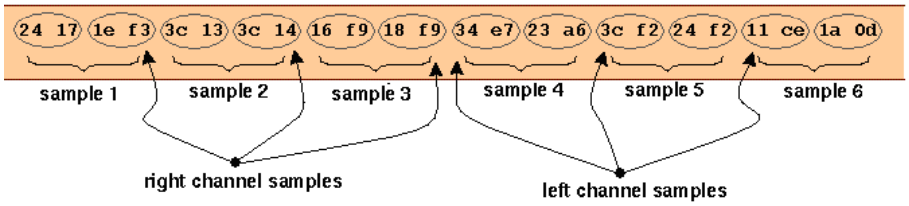
heig-vd

# Application répandue des fichiers binaires

## Les fichiers sonores (.wav) – les échantillons (données du son)



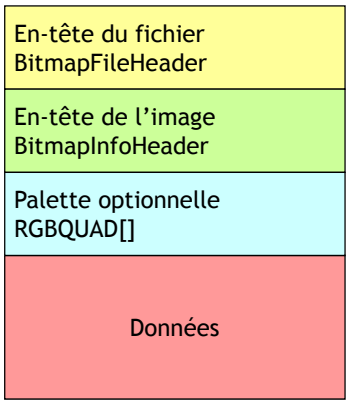




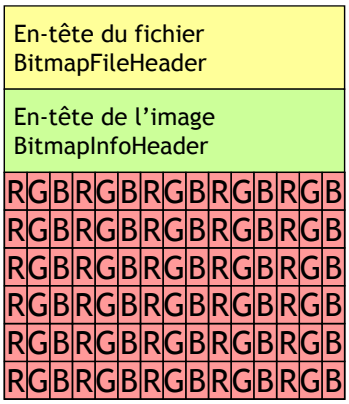
## Application répandue des fichiers binaires

Les fichiers image Bitmaps (.bmp)

### Format général




### Cas des photos 24 bits RGB








## Application répandue des fichiers binaires

Les fichiers image Bitmaps (.bmp) – les structures

```
typedef struct {
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;

typedef struct {
        DWORD       biSize;
        LONG        biWidth;
        LONG        biHeight;
        WORD        biPlanes;
        WORD        biBitCount;
        DWORD       biCompression;
        DWORD       biSizeImage;
        LONG        biXPelsPerMeter;
        LONG        biYPelsPerMeter;
        DWORD       biClrUsed;
        DWORD       biClrImportant;
} BITMAPINFOHEADER;
```

heig-vd

## Application répandue des fichiers binaires

Les fichiers image Bitmaps (.bmp) - lecture

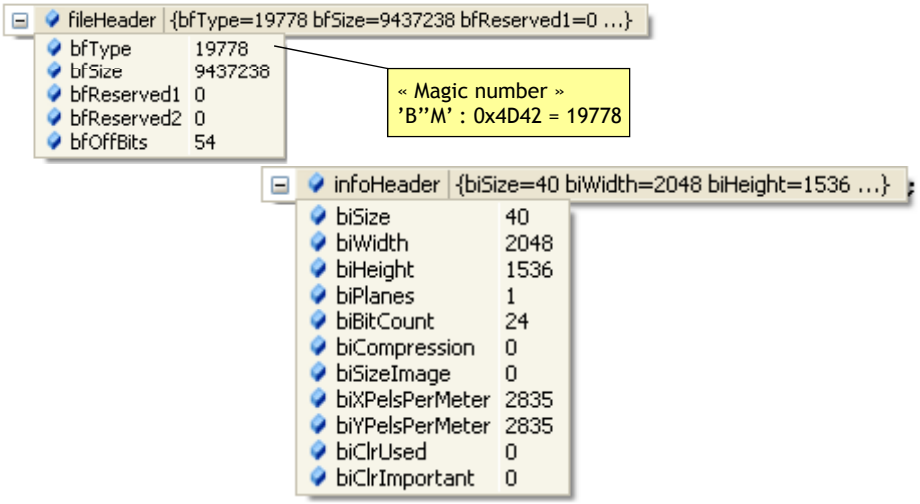
```
BITMAPFILEHEADER fileHeader;
BITMAPINFOHEADER infoHeader;
int data_size;
unsigned char * data;
FILE * source;

source = fopen("Aletsch.bmp", "rb");
if (source != NULL)
{
    fread(&fileHeader, sizeof(fileHeader), 1, source);
    fread(&infoHeader, sizeof(infoHeader), 1, source);
    data_size = fileHeader.bfSize - sizeof(fileHeader) -
        sizeof(infoHeader);
    data = malloc(data_size);
    fread(data, 1, data_size, source);
    fclose(source);
}
```



heig-vd

## Application répandue des fichiers binaires

Les fichiers image Bitmaps (.bmp)



heig-vd

## Application répandue des fichiers binaires
Traitement d'image

- Comment transformer une image ?



heig-vd

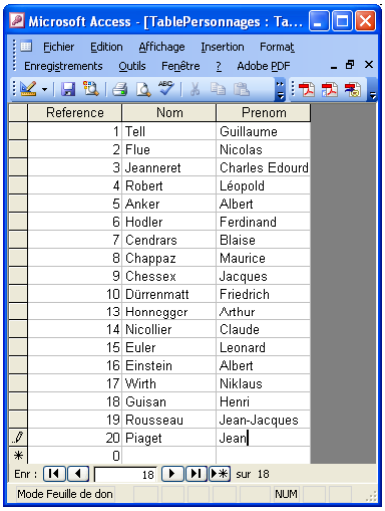
## Application répandue des fichiers binaires
Les bases de données

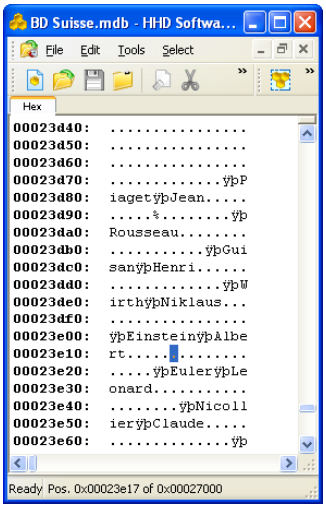
- Domaine d'application
  - Stockage structuré et exploitation de gros volumes de données.
  - Accès concurrent par plusieurs utilisateurs.
  - Banque, assurances, administration, comptabilité, …
- Principe de fonctionnement
  - Fichier binaire ouvert en permanence.
  - Découpage en « enregistrements », blocs de taille connue.
  - Déplacement du pointeur de lecture/écriture dans le fichier

heig-vd

## Application répandue des fichiers binaires

Les bases de données – Exemple élémentaire






heig-vd

## Retour sur la fonction main

Les paramètres de la ligne de commande - principe

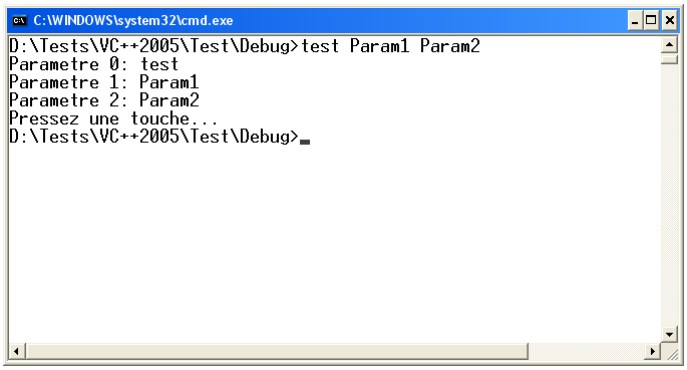
- Utilité des paramètres en ligne de commande
  - Communiquer une information à un programme.
    - Notepad fichier1.txt
  - Modifier son comportement
    - dir /s

heig-vd

## Retour sur la fonction main

Les paramètres de la ligne de commande - principe

- Tapés sur la ligne de commande lors du lancement.



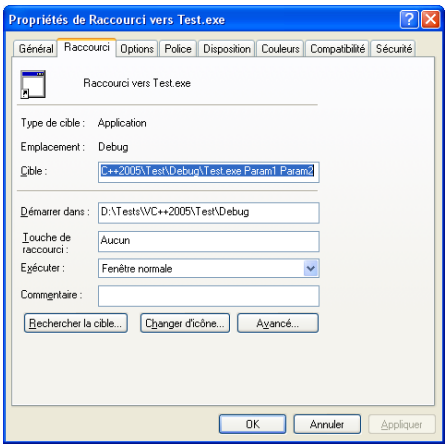



heig-vd

## Retour sur la fonction main

Les paramètres de la ligne de commande - principe

- Saisis dans un raccourci.

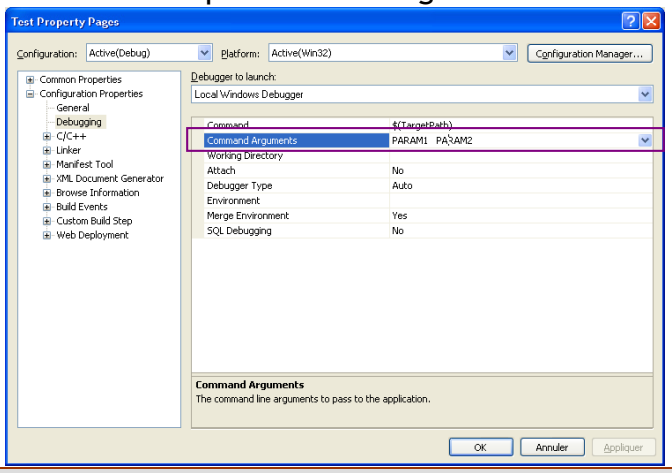


heig-vd

## Retour sur la fonction main

Les paramètres de la ligne de commande - principe

- Introduits dans les options de debug dans Visual Studio






heig-vd

## Retour sur la fonction main

Les paramètres de la ligne de commande – gestion en langage C

- Ils sont passés en paramètre à la fonction main
  - Dans ce cas, les indiquer dans le prototype de main.
- Exemple

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;

    for (i = 0; i < argc; i++)
        printf("Parametre %d: %s\n", i, argv[i]);
    printf("Pressez une touche...");
    _getch();
}
```

Nombre d'arguments

Valeurs des arguments (tableau de chaînes de caractères)

heig-vd

## Qu'avons-nous appris ?

- Les gestion des fichiers en C
  - Différents formats
    - Textes
    - Binaires
  - Différents accès
    - Séquentiel
    - Direct (pour les fichiers binaires)
- L'utilisation des paramètres de la ligne de commande.



heig-vd

## Vos questions