PROGRAMMATION AVANCÉE

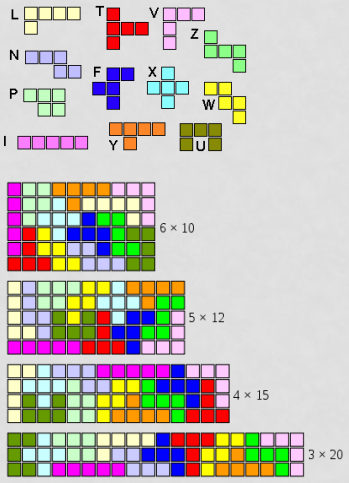
PHUC NGO

CONTENUE DU COURS

- Rappels
  - Fonctions et procédures
  - Structures de données
    - Enregistrements, tableaux, liste enchainé, Pile(FIFO), file(FILO), …
- Compilation avec Makefile/Cmake et la bibliothèque SDL
- Gestion de mémoire
  - Représentation de mémoire
  - Pointeurs et allocation dynamique
- **Gestion de fichiers**
  - **Lecture et écriture**
- Gestion des entrées et sorties
  - Ecran, souris et clavier
- Directives au préprocesseur
- Structures de données et algorithmes de graphes
  - Parcours de graphe, arbre couvrant, …

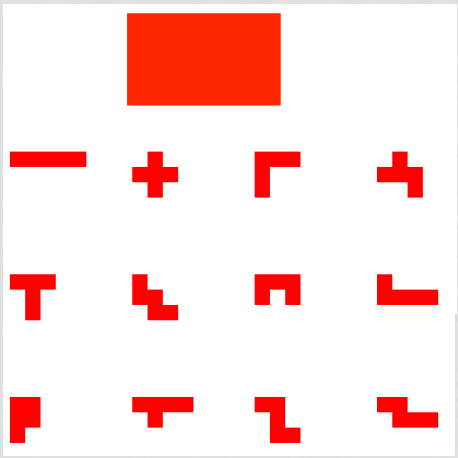
GESTION DE FICHIERS POUR LE PROJET

- Fichier de description du jeu : de la grille et des pièces
- Fichier de sauvegarde de la partie
- Fichier des niveaux
- Fichier de scores
- Et ???



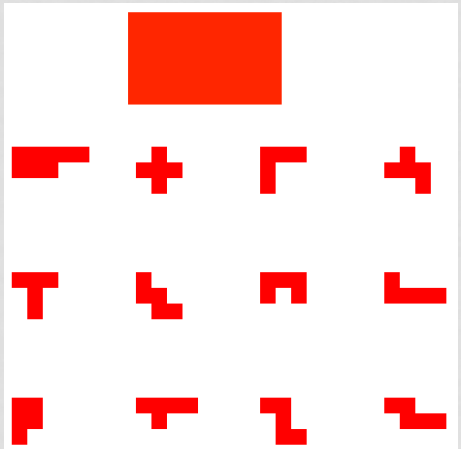
EXEMPLE DU FICHIER D'ENTRÉES

```
##########    #      ##
##########   ###      #
##########    #       ##
##########
##########   #        #
##########   ##      ###
              ##       #
#####
             ####    #
###           #      ####
 #
 #           ###     ##
             #        ###
##           #
##
#           ###
            # #
```



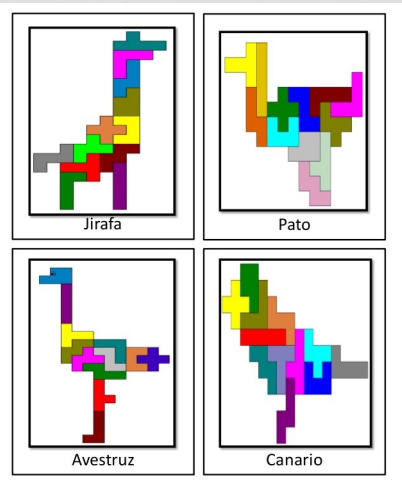
EXEMPLE DU FICHIER D'ENTRÉES

```
##########    #      ##
##########   ###      #
##########    #       ##
##########
##########   #         #
##########   ##       ###
              ##        #
#####
###          ####     #
              #       ####
###
 #           ###      ##
 #           #         ###
             #
##
##           ###
#            # #
```



EXEMPLE DE LA GRILLE DE PENTOMINO

```
      ##
    #####
    ###
    ##
    ##
    ##
    ##
   ###
  ####
  ####
 ######
#######
# #####
  ###  #
  #    #
  #    #
  #    #
```



FICHIERS

- Stocker les informations sur le disque dur
  - Variables sont stockées dans la mémoire vive et supprimées à la fin du programme
- Stocker les informations évoluées dans le temps
  - Constantes sont limitées en utilisation (pas modifiables)
- Différents types de données
  - Types simples et complexes
  - Données formatées
- Illimité en taille*
- Plus flexible et efficace pour la gestion des données du programme

FICHIERS SÉQUENTIELS

- Les informations sont stockées **consécutivement** dans l'ordre par leur entrée et peuvent seulement lus dans le même l'ordre
  - C'est comme une liste chainée, pour accéder à un élément précis dans un fichier séquentiels, on doit lire tout les éléments qui le précèdent, en commençant par le 1er

…	Alice	Jean	Clément	John	Florian	…

⇧
La tête de lecture / écriture

- Les fichiers séquentiels aurons les propriétés suivantes
  - Il est en état d'écriture ou lecture mais pas les deux à la fois
  - À un moment, on ne peut accéder qu'à l'élément qui se trouve en face de la tête de lecture/écriture
  - Après chaque accès, la tête de lecture/écriture se déplace à l'élément suivant

GESTION DE FICHIERS

- Accès à un fichier via une mémoire tampon (buffer) pour réduire le nombre d'accès aux périphériques
- L'information pour manipuler un fichier
  - L'adresse de la mémoire tampon
  - La position de la tête de lecture/écriture
  - Le mode d'accès au fichier : lecture, écriture, …
  - État d'erreur de la manipulation
- L'objet de type **FILE en C**
  - Structure permettant la manipulation de fichier
  - Un flot de données (stream)
  - Définir dans *stdlib.h* et *stdio.h*

OBJET FILE

- Définit dans *stdio.h*
  - Permet aux manipulations de fichiers
    - Gestion de buffer, curseur, …
    - Accès au fichier pour la lecture, l'écriture, …

```
typedef struct __sFILE {
    unsigned char *_p;   /* current position in (some) buffer */
    int   _r;            /* read space left for getc() */
    int   _w;            /* write space left for putc() */
    short _flags;        /* flags, below; this FILE is free if 0 */
    short _file;         /* fileno, if Unix descriptor, else -1 */
    struct __sbuf _bf;   /* the buffer (at least 1 byte, if !NULL) */
    int   _lbfsize;      /* 0 or -_bf._size, for inline putc */
    /* opérations */
    /* gestion de buffers */
} FILE;
```

OPÉRATIONS SUR LES FICHIERS

- Syntaxe C
  - Inclure la bibliothèque : #include <stdlib.h> et <stdio.h>
  - Ouvrir le fichier

    FILE* ptrFichier = **fopen**(**const char*** nom_fichier, **const char*** mode)

    mode :

    "r": lecture seule

    "w": écriture seule

    "a": ajout à la fin

    "r+": lecture et écriture pour un fichier existant

    "w+": lecture et écriture, avec suppression du contenu au préalable

    "a+": ajout et lecture/écriture à la fin
  - Fermer le fichier

    int fclose(**FILE*** ptrFichier)
    - Valeur de retourne : 0 si réussi, EOF sinon
  - Opérations à faire obligatoirement pour toute manipulation

TRAITEMENT DE FICHIERS

- Traitement par enregistrement
  - Les fichiers textes qui stockent des informations formatées (des enregistrements)
  - Ils sont organisés en ligne et la fin est marquée par '\n'
  - Pour pouvoir réaliser les traitements, on doit connaître la structure des informations stockées

```
typedef struct
{
        char nom[20];
        char prenom[20];
        bool sexe;
        int age;
} personne;
```

Fichier personne.txt

```
Alice Dupont F 20
Jean Pierre M 21
Clement Dubois M 20
```

TRAITEMENT DE FICHIERS

- Traitement par caractère
  - Permet d'une gestion plus flexible et pratique pour manipuler de façon confortable des textes écrits
  - Les traitements se réalisent au fur et à mesure en fonction du caractère lu

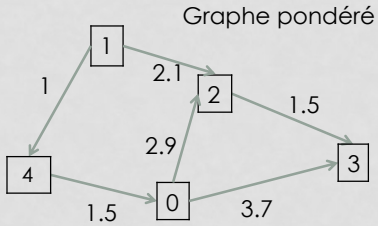
Fichier pieces.txt

```
#####

###
  #
  #

##
##
#
```

Fichier graphe.txt

```
5
0 0 2.9 3.7 0
0 0 2.1 0 1
0 0 0 1.5 0
0 0 0 0 0
1.5 0 0 0 0
```

Graphe pondéré



ÉCRITURE DANS UN FICHIER

- Syntaxe C
  - Inclure la bibliothèque : #include <stdlib.h> et <stdio.h>
  - Écrire dans le fichier
    - Écrit un caractère dans le fichier

      int fputc (**int** caractere, **FILE*** ptrFichier)
      - Valeur de retourne : le caractère écrit si réussi, EOF sinon
    - Écrit une chaîne dans le fichier

      char* fputs (**const char*** chaine, **FILE*** ptrFichier)
      - Valeur de retourne : non-négative value si réussi, EOF sinon
    - Écrit une chaîne formatée dans le fichier = même que printf

      void fprintf (**FILE*** ptrFichier , **const char*** format, arg1, arg2, …)

EXEMPLE : ÉCRITURE DE FICHIER

- Fonction **fputc**

```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "w") ;

char text[8] = "bonjour" ;

if (fichier != NULL) {
      for (int i = 0 ; i < 7; i++)
            fputc(text[i], fichier);
      fputc('\n', fichier);
      fclose(fichier);
}
```

Fichier exemple.txt

```
bonjour
```

EXEMPLE : ÉCRITURE DE FICHIER

- Fonction **fputs**

```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "w") ;

char text[8] = "bonjour" ;

if (fichier != NULL) {

      fputs(text, fichier);
      fputc('\n', fichier);
      fclose(fichier);
}
```

Fichier exemple.txt

```
bonjour
```

SORTIES MISES EN FORME

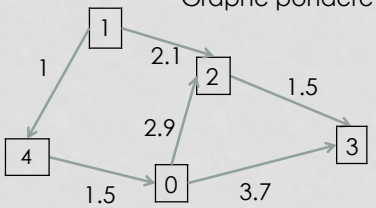
- La fonction fprintf

  int fprintf (**FILE** *ptrFichier, **const char** *format, arg1, arg2, ...)

- Lors que le format de données est bien précis
  - Un tableau des entiers, un enregistrement de personne, une liste chainée, …

Caractère	Type de l'argument
%d ou %i	int
%u	unsigned int
%f	float
%ld	double
%c	char
%s	char*

Graphe pondéré



EXEMPLE : SORTIES MISES EN FORME

- Fonction **fprintf**

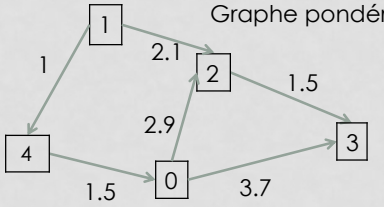
```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "w") ;

char text[8] = "bonjour" ;
int x = 3 ;
float y = 10.7 ;
char z = '\n' ;

if (fichier != NULL) {
    fprintf(fichier, "%s %d %f %c", text, x, y, z);
    fclose(fichier);
}
```

Fichier exemple.txt

```
bonjour 3 10.7
```

EXEMPLE : SORTIES MISES EN FORME

Graphe pondéré



G	[0]	[1]	[2]	[3]	[4]
[0]	0	0	2.9	3.7	0
[1]	0	0	2.1	0	1
[2]	0	0	0	1.5	0
[3]	0	0	0	0	0
[4]	1.5	0	0	0	0

```
float** G ; //G est le tableau représentant le graphe pondéré
FILE * pFile=NULL; int taille = 5;
pFile=fopen( "graph.txt" , "w");
if (pFile==NULL)  perror ("Error opening file");
else {
      fprintf(pFile, "%d\n" , taille );
      for(int i=0; i<taille ; i++) {
            for(int j=0; j<taille ; j++)
                  fprintf(pFile, "%f " , G[i][j]);
                  fprintf(pfile, "\n");
      }
      fclose (pFile);
}
```

graphe.txt

```
5
0 0 2.9 3.7 0
0 0 2.1 0 1
0 0 0 1.5 0
0 0 0 0 0
1.5 0 0 0 0
```

LECTURE DES FICHIERS

- Syntaxe C
  - Inclure la bibliothèque : #include <stdlib.h> et <stdio.h>
  - Lire dans le fichier
    - Lit un caractère dans le fichier

      int fgetc (**FILE*** ptrFichier)
      - Valeur de retourne : le caractère lu si réussi, EOF sinon
    - Lit une chaîne dans le fichier

      char* fgets (**char*** chaine, **int** nbCaractere, **FILE*** ptrFichier)
      - Valeur de retourne : le pointeur de la chaîne si réussi, NULL sinon
    - Lit une chaîne formatée dans le fichier = même que scanf

      void fscanf(**FILE*** ptrFichier, **const char** *format, arg1, arg2, ...)
      - Similaire que fprintf pour une lecture avec un format prédéfini

EXEMPLE : LECTURE DE FICHIER

- Fonction **fgetc**

```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "r") ;

int c ;

if (fichier != NULL) {
     do {
          c = fgetc(fichier);
          printf("%c ", c) ;
     } while (c != EOF) ;

      fclose(fichier);
}
```

Fichier exemple.txt

EXEMPLE : LECTURE DE FICHIER

- Fonction **fgets**

```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "r") ;
#define NB_MAX 100
char str[NB_MAX] = "" ;

if (fichier != NULL) {

     fgets(str, NB_MAX, fichier);
     for (int i=0; i < strlen(str); i++)
          printf("%c ", c) ;

      fclose(fichier);
}
```

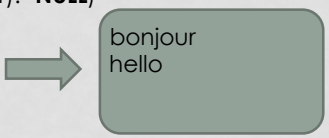
Fichier exemple.txt

EXEMPLE : LECTURE DE FICHIER

- Fonction **fgets**

```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "r") ;
#define NB_MAX 100
char str[NB_MAX] = "" ;

if (fichier != NULL) {

     //tantqu'on n'est pas fin de fichier
     while(fgets(str, NB_MAX, fichier)!=NULL)
          printf("%s ", str) ;

      fclose(fichier);
}
```

Fichier exemple.txt

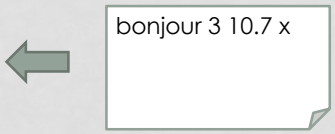


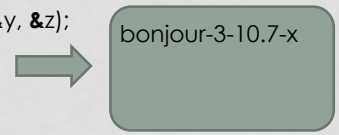
EXEMPLE : SORTIES MISES EN FORME

- Fonction **fscanf**

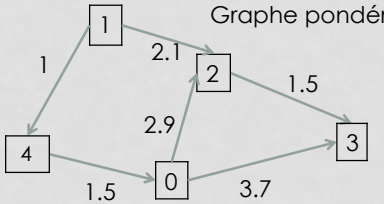
```
FILE* fichier = NULL ;
fichier = fopen("exemple.txt", "r") ;
#define NB_MAX 100
char str[NB_MAX] = "" ;
int x ;
float y ;
char z ;

if (fichier != NULL) {
     fscanf(fichier, "%s %d %f %c", str, &x, &y, &z);
     printf("%s-%d-%f-%c", str, x, y, z);
     fclose(fichier);
}
```

Fichier exemple.txt





EXEMPLE : SORTIES MISES EN FORME



G	[0]	[1]	[2]	[3]	[4]
[0]	0	0	2.9	3.7	0
[1]	0	0	2.1	0	1
[2]	0	0	0	1.5	0
[3]	0	0	0	0	0
[4]	1.5	0	0	0	0

```
float** G ; //G est le tableau représentant le graphe pondéré
FILE * pFile=NULL;
int taille;
pFile=fopen( "graph.txt" , "r");
if (pFile==NULL)  perror ("Error opening file");
else {
      fscanf( pFile, "%d" , &taille) ;
      for(int i=0; i<taille ; i++) {
            for(int j=0; j<taille ; j++)
                  fscanf (pFile, "%f " , &G[i][j]);
      }
      fclose (pFile);
}
```

graphe.txt

```
5
0 0 2.9 3.7 0
0 0 2.1 0 1
0 0 0 1.5 0
0 0 0 0 0
1.5 0 0 0 0
```

OPÉRATIONS SUR LES FICHIERS

- Syntaxe C
  - Se déplacer dans un fichier = curseur
    - Indique la position actuelle dans le fichier
      long ftell (**FILE*** ptrFichier)
      - Valeur de retourne : la position actuelle si réussi, -1 sinon
    - Positionne le curseur à un endroit précis
      int fseek (**FILE*** ptrFichier **, long** déplacement**, int** origine)
      - Valeur de retourne : 0 si réussi, non-zéro sinon
    - Remet le curseur au début du fichier = fseek
      void rewind (**FILE*** ptrFichier)
  - Renommer un fichier
    int rename (**const char*** ancienNom, **const char*** nouveauNom)
  - Supprimer un fichier
    int remove (**const char*** fichierASupprimer)

OPÉRATIONS SUR LES FICHIERS

- Syntaxe C
  - Vérifier l'ouverture du fichier
    ptrFichier == NULL
  - Vérifier la fin du fichier : EOF = End Of File
    fgetc(ptrFichier) == EOF ou **fgets**(str, NB_MAX, ptrFichier) == NULL
    int feof (**FILE*** ptrFichier)
    - Valeur de retourne : 0 si EOF, non-zéro sinon
  - Vérifier fin de la ligne
    fgetc(ptrFichier) == '**\n**' ou '**\r** ' (selon le système)
  - Vérifie si une erreur sur un flux de données
    int ferror (**FILE*** ptrFichier)
    - Valeur de retourne : 0 si ok, non-zéro sinon
    - Exemple : après une écriture dans un fichier

AJOUTER DANS UN FICHIER

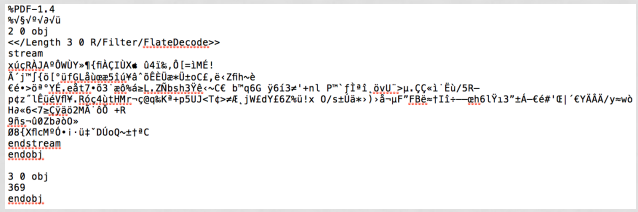
- Ajouter à la fin du fichier
  - L'ouverture avec le mode "a" ou "a+"
  - L'ancien fichier est entièrement copié dans le nouveau fichier, suivi du élément ajouté
- Ajoute au début du fichier
  - L'ancien fichier est copié derrière le nouvel élément qui est écrit en premier lieu
- Insertion dans un fichier
  - Copier les éléments de l'ancien fichier qui précèdent l'élément à ajouter,
  - Écrire le nouvel élément,
  - Copier le reste dans l'ancien fichier

AUTRES TRAITEMENTS

- Supprimer un élément dans un fichier
  - Copier les éléments de l'ancien fichier qui précèdent l'élément à supprimer
  - Copier tous ce qui suivent l'élément à supprimer
- Modifier un élément dans un fichier
  - Copier les éléments de l'ancien fichier qui précèdent l'élément à modifier
  - Écrire l'élément modifier
  - Copier tous ce qui le suivent

DES FORMATS DES FICHIERS

- Chaque type de fichier a une spécification
  - Normes, standards, conseils d'usage
  - Exemple : la spécification PDF/A-1 (un fichier pdf)
    - La norme ISO 19005-1 : Gestion de document – Format de document portable
    - Référence : https://www.adobe.com/devnet/pdf/pdf_reference.html



- Bien compris le format permet de
  - Ouvrir et lire correctement le fichier en format donné
    - Pdf, png, mp4,...
  - Convertir entre les différents formats
    - Word ←→ pdf, png ←→ jpeg, ...

ENCODAGE DE CARACTÈRES

- L'encodage des caractères dans les fichiers peut être différent selon
  - Le système d'exploiration que le fichier est crée
  - Le logiciel sous lequel le fichier est créé
  - Exemple : '\n' ou '\r' pour le saut de ligne
- Pour le projet, à faire attention
  - Contrôler bien des caractères lus à partir des fichiers
    - Les caractères autorisés
    - Les caractères non autorisés : afficher message d'erreurs
    - Le nombre de caractères par ligne pour allocation dynamique
  - Minimiser le nombre d'accès au fichier
    - La lecture de fichiers du disque dur est coûteuse en temps
    - L'allocation dynamique avec les fichiers est coûteuse en espace
    - Pas faire l'accès aux fichiers (textes, images, ...) dans des boucles

RÉCAPITULATIF GESTION DES FICHIERS

- Lire dans un fichier

```
#include <stdio.h>
int main () {
        FILE * pFile=NULL;
        int c, n = 0;
        pFile=fopen ("myfile.txt","r");
        if (pFile==NULL)
                perror ("Error opening file");
        else {
                do {
                        c = fgetc (pFile);
                        printf(" %c ", c);
                        n++;
                } while (c != EOF);
                fclose (pFile);
                printf ("File contains %d caracteres\n",n);
        }
    return 0;
}
```

Ouverture du fichier, et la vérification

Lecture caractère par caractère

Jusqu'à la fin du fichier

Fermeture du fichier

RÉCAPITULATIF GESTION DES FICHIERS

- Écrire dans un fichier

```
#include <stdio.h>
int main () {
        FILE * pFile=NULL;
        pFile=fopen("myfile.txt","w");
        if (pFile==NULL)
                perror ("Error opening file");
        else {
                fputc ('x',pFile);
                if (ferror (pFile))
                        perror ("Error Writing to myfile.txt\n");
                fclose (pFile);
        }
        return 0;
}
```

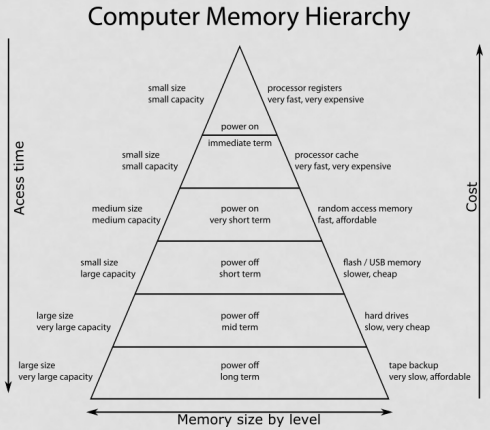
Ouverture du fichier, et la vérification

Écrire le caractère 'x' dans le fichier et vérifier

Fermeture du fichier

HIÉRARCHIE MÉMOIRE

- Temps d'accès à la mémoire dépend le type de mémoires
  - Augemente de la mémoire interne à la mémoire externe
- **Éviter de lire les fichiers dans les boucles !!!**
  - Faire une seule fois au début du programme



wikipédia

TEMPS D'ACCÈS À LA MÉMOIRE

Type de mémoire		Taille	Temps d'accès
Registres de processeur		qq milliers d'octets	1 cycle processeur (~ ns)
Mémoire cache CPU (SRAM)	N0 : mémoire des microcodes	6 Ko	--
	N1 : mémoire cache des instructions / données	128 Ko	700 Go/seconde
	N2,3,4 : cache partagée des instructions / données	1, 6, 128 Mo	200, 100, 40 Go/seconde
Mémoire vive (DRAM)		qq Go	10 Go/seconde
Disque dur		qq To	~ 75 Mo/seconde

TEMPS D'ACCÈS À LA MÉMOIRE

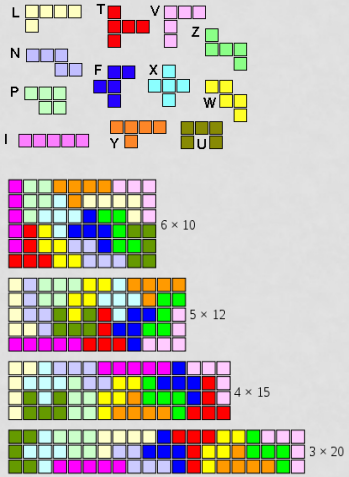
- Comparaison pour 1000 transfers de données
  - Les registres de processeur (~ 1 ns) : 10^{-3} ms
  - Mémore cache CPU : 5 x 10^{-3} ms
    - ➔ 5 fois le temps qu'il faut pour les registres
  - Mémoire vive : 10^{-2} ms
    - ➔ 2 fois le temps qu'il faut à la mémoire cache
  - Disque dur : 5000 ms
    - ➔ **500 000 fois le temps** qu'il faut à la mémoire vie

CONTENUE DU COURS

- Rappels
  - Fonctions et procédures
  - Structures de données
    - Enregistrements, tableaux, liste enchainé, Pile(FIFO), file(FILO), …
- Compilation avec Makefile/Cmake et la bibliothèque SDL
- Gestion de mémoire
  - Représentation de mémoire
  - Pointeurs et allocation dynamique
- Gestion de fichiers
  - Lecture et écriture
- **Gestion des entrées et sorties**
  - **Ecran, souris et clavier**
- Directives au préprocesseur
- Structures de données et algorithmes de graphes
  - Parcours de graphe, arbre couvrant, …

GESTION DES ENTRÉES/SORITES POUR LE PROJET

- Graphique du jeu sur l'écran
- Affichage de menu du jeu
- Affichage de scores
- Gestion de la souris pour déplacer les pièces
- Gestion de la clavier pour choisir dans le menu
- Et ???



GESTION DES ENTRÉES/SORTIES

- Clavier = l'entrée standard
  - Envoyer au programme les données
  - Communiquer avec les utilisateurs
- Écran = la sortie standard
  - Afficher les données à l' écran
  - Annoncer des erreurs
  - Débugger le programme
- Échanger avec les utilisateurs
  - Interaction via la souris, le clavier et l'écran
- Fichiers = entrées et sorties indirect du programme
  - Rediriger les entrées et sorties standards vers fichiers
  - Écrire/lire des données sur un flux de sortie/entrée standard

GESTION DES ENTRÉES/SORTIES

- Les flux d'entrées et de sorties
  - **stdin** : l'entrée standard (clavier)
  - **stdout** : la sortie standard (écran)
  - **stderr** : la sortie standard des erreurs (écran ou fichier log)
  - **FILE** : les fichiers
- Les entrées/sorties sont bufférisées par défaut
- Ils sont fermés automatiquement à la fin du programme
- Effectuer des opérations avec les fonctionnalités offertes par le système
- Bibliothèque standard avec les fonctions permettant d'interagir avec les entrées/sorties
  - Fonctions: printf, scanf, fprintf, fscanf, getchar(), getc(stdin), putchar(c), putc(c, stdout)

GESTION DES ENTRÉES/SORITES

- Syntaxe en C
  - Inclure la bibliothèque : #include <stdio.h>
  - Écrire les données formatées *sur* le flux de sortie standard *(stdout)*

    int printf (**const char*** la_chaine_formatée, ...)
    - formats : %c (char), %d (int), %f (float), %s (string),…
    - Valeur de retourne : le nombre de caractères écrits si réussi, négative valeur sinon
  - Écrire un caractère sur un flux de sortie *(stdout ou fichier)*

    int putchar (**int** character)
    int putc (**int** character, **FILE*** flux_donnees)
    int fputc (**int** character, **FILE*** flux_donnees)
    - Valeur de retourne : le caractère écrite si réussi, EOF sinon
  - Écrire une chaine de caractères sur un flux de sortie

    char* puts (**const char*** chaine)
    char* fputs (**const char*** chaine, **FILE*** flux_donnees)
    - Valeur de retourne : le pointeur de la chaîne si réussi, NULL sinon

GESTION DES ENTRÉES/SORITES

- Syntaxe en C
  - Inclure la bibliothèque : #include <stdio.h>
  - Lire les données formatées sur le flux d'entrée standard (*stdin*)

    int scanf (**const char*** la_chaine_formatée, ...)
    - formats : %c (char), %d (int), %f (float), %s (string),…
    - Valeur de retourne : le nombre de caractères écrits si réussi, négative valeur sinon
  - Lire un caractère sur un flux d'entrée (*stdin ou fichier*)

    int getchar ()
    int getc (**FILE*** flux_donnees)
    int fgetc (**FILE*** flux_donnees)
    - Valeur de retourne : le caractère écrite si réussi, EOF sinon
  - Lire une chaine de caractères sur un flux d'entrée

    char* gets (**const char*** chaine)
    char* fgets (**const char*** chaine, **int** nbCaractere, **FILE*** flux_donnees)
    - Valeur de retourne : le pointeur de la chaîne si réussi, NULL sinon

REDIRIGER LES FLUX STANDARDS

- Rediriger les entrée et sortie standards vers fichier
  - Opérations : **<** (entrée) et **>** (sortie)

    ./programme > fichier_sortie.txt

    ./programme < fichier_donnee.txt

    ./programme < fichier_donnee.txt > fichier_sorite.txt
- Écrire sur la sortie standard avec **fprintf** (cf. les fichiers)

  int fprintf (**FILE*** flux_donnees , **const char*** la_chaine_formatée, …)

  fprintf (**stdout**, "Bonjour !\n")
- Lire sur l'entrée standard avec **fscanf** (cf. les fichiers)

  int fscanf (**FILE*** flux_donnees , **const char*** la_chaine_formatée, …)

  fscanf (**stdin,** "%d", **&**variable)

REDIRIGER LES FLUX STANDARDS

- Afficher le message d'erreur sur la sortie d'erreur **stderr**

  void perror (**const char*** message_erreur)
- Par défaut, stderr est associée à l'écran

```
#include <stdio.h>
int main( void ) {
    FILE * myFile = fopen("exemple.txt", "r" );
    if(myFile==NULL) {
        perror( "Error: " );
        return -1; }
    fclose( myFile );
    return 0; }
```

Sortie standard (stdout) ➔ écran

```
Error: No such file or directory
```

- Rediriger vers un fichier

  ./programme > log.txt

Sortie fichier (FILE) ➔ log.txt

```
Error: No such file or directory
```

EXEMPLE : GESTION DES FLUX DE DONNÉES

```
#include <stdio.h>
int main () {
        FILE * pFile=NULL;
        int age = 19;
        pFile=fopen ("myfile.txt","a+");
        if (pFile==NULL)
                perror ("Error opening file");
        else {
                printf("Entrez votre age");
                scanf("%d",&age);
                fprintf(pFile, "L'utilisateur a %d ans", age);
                fclose(pFile);
        }
    return 0;
}
```

Ouverture du fichier, et la vérification

Écrit dans le fichier la chaine formatée

Fermeture du fichier

Fichier mytext.txt

age=19 → L'utilisateur a 19 ans

EXEMPLE : GESTION DES FLUX DE DONNÉES

```
#include <stdio.h>
int main () {
        FILE * pFile=NULL;
        int age;
        pFile=fopen ("myfile.txt","r");
        if (pFile==NULL)
                perror ("Error opening file");
        else {
                fscanf(pFile, "L'utilisateur a %d ans", &age);
                printf("Age d'utilisateur est %d", age);
                fclose(pFile);

        }
    return 0;
}
```

Ouverture du fichier, et la vérification

Lit dans le fichier la chaine formatée

Fermeture du fichier

Fichier mytext.txt

L'utilisateur a 19 ans → Age d'utilisateur est 19

GESTION DES ENTRÉES ET SORTIES AVEC SDL

- SDL dispose des API permettant de gérer des événements lié à l'écran, le clavier, la souris, …
  - https://wiki.libsdl.org/SDL_Event

SDL_Event	Type	Description
SDL_WindowEvent	window	window event data
SDL_KeyboardEvent	key	keyboard event data
SDL_MouseMotionEvent	motion	mouse motion event data
SDL_MouseButtonEvent	button	mouse button event data
SDL_MouseWheelEvent	wheel	mouse wheel event data
SDL_QuitEvent	quit	quit request event data
SDL_JoyAxisEvent	jaxis	joystick axis event data

DÉTECTION DES ÉVÈNEMENTS AVEC SDL

```
#include "SDL2/SDL.h"

int main( void ) {
//créer la variable pour les événements
SDL_Event evt;

//Boucle pour la detection des événements
while (SDL_PollEvent(&evt)) {
        //Detection du type d'évéments
        switch (evt.type) {
                case SDL_KEYDOWN:
                        printf( "Key press detected\n" ); break;
                case SDL_KEYUP:
                        printf( "Key release detected\n" ); break;
                default: break;
        }
}
```

SDL : ÉVÈNEMENTS CLAVIER

- Les événements du clavier :
  - SDL_KEYDOWN : une touche est enfoncée
  - SDL_KEYUP : une touche est relâchée

```
switch (evt.type) {
  case SDL_KEYDOWN : // Si une touche est appuyée
     switch(evt.key.keysym.sym) { // Teste la touche appuyée
        case SDLK_ESCAPE : // Touche échapé
           terminer = true; break;
        case SDLK_UP: // Flèche haut
           position.y--; break;
        case SDLK_DOWN: // Flèche bas
           position.y++; break;
        case SDLK_RIGHT: // Flèche droite
           position.x++; break;
        case SDLK_LEFT: // Flèche gauche
           position.x--; break;
     }
}
```

SDL : ÉVÈNEMENTS SOURIS

- Les événements de la souris :
  - SDL_MOUSEMOTION : la souris est en motion
  - SDL_MOUSEBUTTONDOWN : bouton de la souris est enfoncée
  - SDL_MOUSEBUTTONUP : bouton la souris est relâchée
  - SDL_BUTTON_LEFT, SDL_BUTTON_MIDDLE, SDL_BUTTON_RIGHT, SDL_BUTTON_WHEELUP, SDL_BUTTON_WHEELDOWN, …

```
switch (evt.type) {
  case SDL_MOUSEMOTION:
     position.x=evt.motion.x;
     position.y=evt.motion.y;
     break;
  case SDL_MOUSEBUTTONDOWN:
     …
  case SDL_MOUSEBUTTONUP:
     …
  }
}
```

SDL : ÉVÈNEMENTS FENÊTRE

- Les événements de la fenêtre :
  - SDL_VIDEORESIZE : la fenêtre est redimensionnée
  - SDL_WINDOWEVENT_CLOSE : la fenêtre est fermée
  - SDL_WINDOWEVENT_FOCUS_LOST : la fenêtre perd le focus
  - SDL_WINDOWEVENT_MINIMIZED, SDL_WINDOWEVENT_MAXIMIZED, SDL_WINDOWEVENT_RESTORED, SDL_WINDOWEVENT_ENTER, SDL_WINDOWEVENT_FOCUS_GAINED, …

```
switch (evt.type) {
  case SDL_WINDOWEVENT_CLOSE :
     terminer = true; break;
  case SDL_WINDOWEVENT_FOCUS_LOST :
     …
  case SDL_VIDEORESIZE :
     …
  }
}
```

SDL : GESTION DE SONS

- La bibliothèque tierce nommée **SDL_sound** ou **SDL_mixer**
  - Documentation : https://www.libsdl.org/projects/SDL_mixer
  - API : https://www.libsdl.org/projects/SDL_mixer/docs/SDL_mixer.html
    Include : #include <SDL2/SDL_mixer.h>
- Les fonctions utilisées
  - Charger le fichier d'audio
    - SDL_AudioSpec* SDL_LoadWAV(const char* file, SDL_AudioSpec* spec, Uint8** audio_buf, Uint32* audio_len)
  - Libérer la mémoire chargée : void SDL_FreeWAV(Uint8* audio_buf)
  - Initialise SDL_Mixer : int Mix_Init(int flags)
  - Quitter SDL_Mixer : void Mix_Quit()
  - Ouvrir un audio mixer
    - int Mix_OpenAudio(int frequency, Uint16 format, int channels, int chunksize)
  - Fermer un audio mixer : void Mix_CloseAudio()
  - Pause l'audio : void Mix_Pause(int channel)
  - Résume l'audio : void Mix_Resume(int channel)
  - Contrôler le volume :
    - int Mix_VolumeChunk(Mix_Chunk *chunk, int volume)

EXEMPLE : GESTION DE SONS

```
// Initialiser le mixer
Mix_Init(MIX_INIT_OGG|MIX_INIT_MOD);
SDL_AudioSpec audioBufferSpec;
Uint8 *audioBuffer;
Uint32 audioBufferLen;
SDL_LoadWAV("audio.wav",&audioBufferSpec,&audioBuffer, &audioBufferLen);
Uint32 len=5, audioPos = 0;

// Lecture du buffer audio
while(audioPos < audioBufferSpec.len) {
        SDL_MixAudio(stream,audioBuffer+audioPos,len,SDL_MIX_MAXVOLUME);
        audioPos += len;
}

//Libérer la mémoire
SDL_FreeWAV(&audioBuffer);
//Quitter le mixer
Mix_Quit();
```

RÉCAPITULATIFS

- Gestion de fichiers
  - Accès au fichier : ouverture et fermertures
  - Manipultations des fichiers : écriture et lecture
  - Opérations sur les fichiers : curseur, véficiation, …
- Gestion d'entrée et de sorite
  - Le clavier = flux d'entrée standard (**stdin**)
  - L'écran = flux de sortie standard (**stdout**)
  - Les fichiers = flux d'entrée et de sortie (**FILE**)
  - Redirection des flux standards vers fichiers
  - Évènements avec SDL : clavier, souris, fenêtre, …

CONTENUE DU COURS

- Rappels
  - Fonctions et procédures
  - Structures de données
    - Enregistrements, tableaux, liste enchainé, Pile(FIFO), file(FILO), …
- Compilation avec Makefile/Cmake et la bibliothèque SDL
- Gestion de mémoire
  - Représentation de mémoire
  - Pointeurs et allocation dynamique
- Gestion de fichiers
  - Lecture et écriture
- Gestion des entrées et sorties
  - Ecran, souris et clavier
- **Directives au préprocesseur**
- **Structures de données et algorithmes de graphes**
  - **Parcours de graphe, arbre couvrant, …**