SCIENCES ET MONTAGNE
UNIVERSITÉ SAVOIE MONT BLANC

UNIVERSITÉ SAVOIE MONT BLANC

INFO 505 - Programmation C II
L3 – 2024-25

CM5 : Fichiers, GUI avec GTK, retours sur qq concepts…

J.Y. RAMEL

Bureau 2D-204 - Polytech'Savoie - Bourget du Lac

Lecture et Écriture dans les fichiers

- Types de flux
  - Flux standard : stdin, stdout, stderr
  - Flux de fichiers : Gestion avec FILE *

- Intérêts
  - Stocker des données de manière persistante.
  - Écrire des résultats ou des logs dans des fichiers de sortie
  - Récupérer des paramètres / données stockés

- 2 types de fichiers
  - Binaire
  - Texte

Comment choisir ?

Ouverture/fermeture

Syntaxe :

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

fclose(FILE *)

Modes courants :

- r : Lecture seule, le fichier doit exister.
- r+ : Lecture et écriture, le fichier doit exister, pas d'effacement.
- w : Écriture seule, création du fichier s'il n'existe pas, contenu effacé s'il existe.
- w+ : Lecture et écriture, création du fichier s'il n'existe pas, contenu effacé s'il existe.
- a : Ajout, création du fichier s'il n'existe pas, écriture toujours à la fin.
- a+ : Lecture et ajout, création du fichier s'il n'existe pas, écriture toujours à la fin.
- **b à ajouter pour fichier binaires**

Exemple :

```
FILE *f = fopen("example.bin", "rb");
if (f == NULL) { perror("Erreur d'ouverture du fichier"); }
fclose(f);        // pour bien libérer les ressources associées
```

Fichiers Texte

```
int main() {
    FILE *file = fopen("exemple.txt", "w");
    if (file == NULL) {
        perror("Erreur ouverture fichier");
            return 1;
    }
    fprintf(file, "Hello, world!\n");
    char *chaine = "igne suivante\n";
    fputc('L', file);              // 1 char
    fputs(chaine, file);           // 1 string

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

```
int main() {
    char buffer[100];
    FILE *file = fopen("exemple.txt", "r");
    if (file == NULL) {
     perror("Erreur ouverture fichier");
     return 1;
    }
    while (fgets(buffer, sizeof(buffer), file)) {
     printf("%s", buffer);
    }
    fclose(file);
    return 0;
}
```

int fseek(FILE *stream, long offset, int whence);
permet de repositionner le curseur dans un fichier à une position précise.
whence (point de référence) = SEEK_SET, SEEK_CUR ou SEEK_END.

Fichiers binaires

```c
int main() {
    int data[] = {1, 2, 3, 4, 5};
    FILE *file = fopen("exemple.bin", "wb");
    if (file == NULL) {
        perror("Erreur ouverture fichier");
        return 1;
    }
    fwrite(data, sizeof(int), 5, file);
    fclose(file);
    return 0;
}
```

```c
int main() {
    int data[5];
    FILE *file = fopen("exemple.bin", "rb");
     if (file == NULL) {
      perror("Erreur ouverture fichier");
      return 1;
    }
    fread(data, sizeof(int), 5, file);
    for (int i = 0; i < 5; i++) {
      printf("%d ", data[i]);
    }
    fclose(file);
    return 0;
}
```

Fichiers binaires

Faire un prg C qui écrit puis lit un Element (de liste chainée) dans un fichier binaire

Fichiers binaires

Faire un prg C qui ecrit puis lit une structure typedef dans un fichier binaire

```
typedef struct {
    int id;
    float valeur;
   char nom[20];
} Element;

int main() {
    FILE *file = fopen("data.bin", "wb");
    if (file == NULL) {
        perror("Erreur à l'ouverture");
        return 1;    }
    Element e = {1, 3.14, "Element1"};
    fwrite(&e, sizeof(Element), 1, file);
    fclose(file);    return 0;
}
```

```
int main()
{
        FILE *file = fopen("data.bin", "rb");
        if (file == NULL) {
            perror("Erreur à l'ouverture");
            return 1;
        }
        Element e;
        fread(&e, sizeof(Element), 1, file);
        printf("ID : %d\n - Valeur : %.2f \n
        - Nom : %s\n", e.id, e.valeur, e.nom);
        fclose(file);
        return 0;
}
```

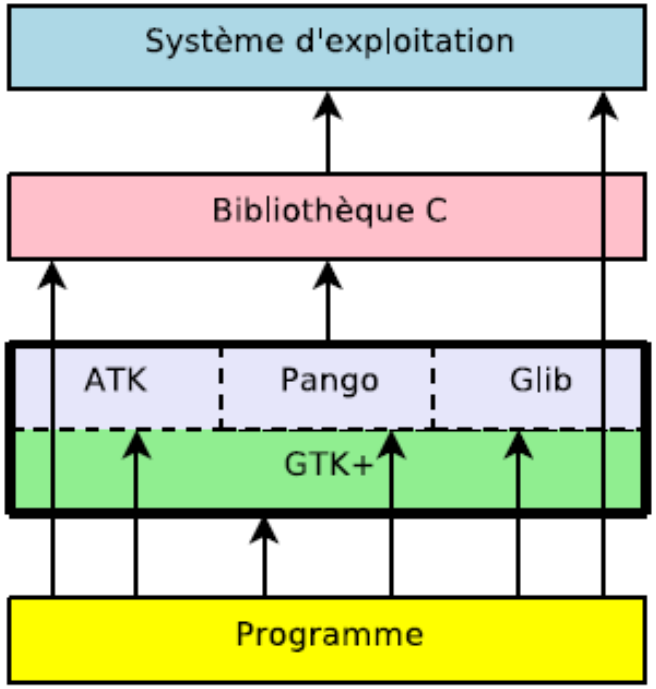
GUI – Interfaces graphique

Que savez vous ?

Notions de base

GTK (GIMP Toolkit)

- Ecrite en C
- Nombreuses liaisons pour d'autres langages et librairies
- Basée sur une architecture orientée objets
- Multiplateforme : Fonctionne sous Linux, Windows et macOS.
- Open-source : Entièrement libre et maintenue par une grande communauté.



Installation (plutôt version 3)

sudo apt-get install libgtk-3-dev

gcc -o my_program my_program.c `pkg-config --cflags --libs gtk+-3.0`

A expliquer ?

Notions de base

Des Widgets

Eléments de l'interface (boutons, labels, fenêtres, zones de saisie, . . .)

GtkButton : Un bouton cliquable.

GtkLabel : Une étiquette pour afficher du texte.

GtkEntry : Un champ de saisie de texte

GtkDialog: fenêtres de dialogue pré-existantes

Menus

GtkMenuBar : La barre de menu qui contient les différents sous-menus.

GtkMenu : Le menu déroulant associé à chaque option de la barre

GtkMenuItem : Les éléments dans chaque sous-menu

Conteneurs de disposition des Widgets

GTK propose plusieurs conteneurs :

- GtkBox : Dispose les widgets en ligne ou en colonne.
- GtkGrid : Dispose les widgets dans une grille 2D

Notions de base

Des signaux et Callbacks associés

- Signal = Evénement déclenchant une ou plusieurs actions (click sur un bouton, . . .)
- Fonction rappel/callback = Fonction appelée lorsque survient un signal

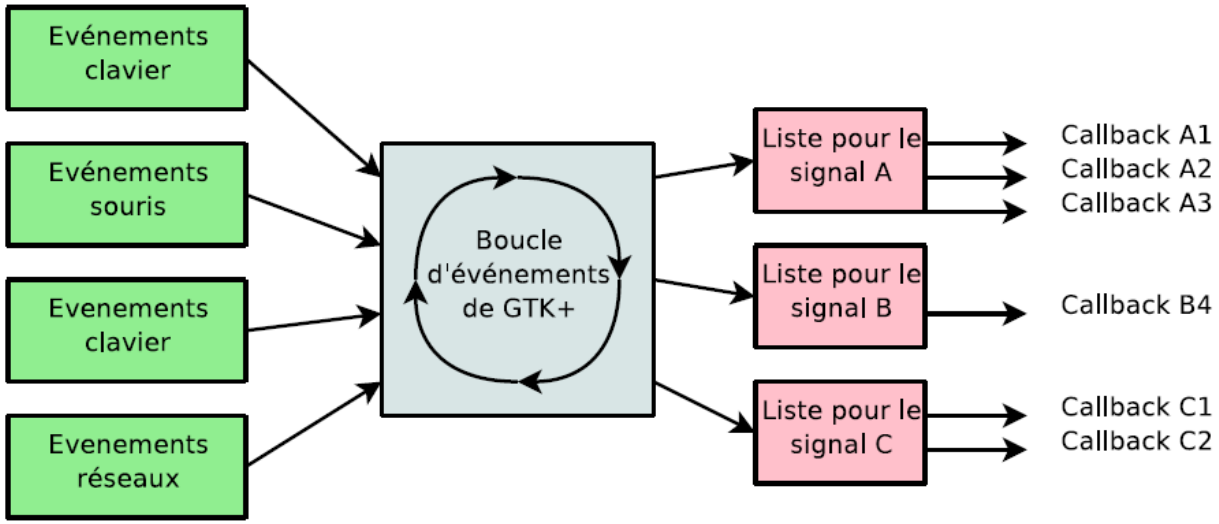
- Association événement ←→ Callback

```
g_signal_connect(button, "clicked", G_CALLBACK(on_button_clicked), NULL);
```

Remarques

– Plusieurs fonctions callback peuvent être associées à un signal

– Les fonctions sont appelées dans l'ordre de leurs ajouts



Quelques éléments et exemples

Exemple 1 : une simple fenêtre vide

```
#include <gtk/gtk.h>

int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);                    // Initialisation de GTK

    // Création d'une nouvelle fenêtre
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Première fenêtre GTK");
    gtk_window_set_default_size(GTK_WINDOW(window), 400, 300);

    // Connecter le signal "destroy" (fermeture de la fenêtre)
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

    gtk_widget_show_all(window);

    gtk_main();
    return 0;
```

A expliquer ?

Quelques éléments et exemples

Exemple 2 : fenêtre avec boutons dans un container

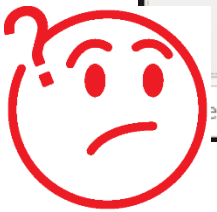
```
int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), « Ma fenetre");
    gtk_window_set_default_size(GTK_WINDOW(window), 400, 300);

    GtkWidget *box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    GtkWidget *button1 = gtk_button_new_with_label("Bouton 1");
    GtkWidget *button2 = gtk_button_new_with_label("Bouton 2");

    gtk_box_pack_start(GTK_BOX(box), button1, TRUE, TRUE, 0);
    gtk_box_pack_start(GTK_BOX(box), button2, TRUE, TRUE, 0);
    gtk_container_add(GTK_CONTAINER(window), box);

    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
    gtk_widget_show_all(window);

    gtk_main();
```

Quelques éléments et exemples

Exemple 3 : Saisie de texte avec un callback


```
// Callback saisie de texte
void on_open_from_entry(GtkWidget *entry, gpointer data) {
        const char *msg = (const char *)data;
        const char *txt = gtk_entry_get_text(GTK_ENTRY(entry));
        printf(" - Texte saisi dans %s = %s\n", msg, txt);
}

int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Disposition des Widgets");
    gtk_window_set_default_size(GTK_WINDOW(window), 400, 300);

    GtkWidget *entry1 = gtk_entry_new();
    gtk_entry_set_placeholder_text(GTK_ENTRY(entry1), "Entrez du texte...");

    // Connecter le signal pour la saisie (appui sur la touche "Entrée")
    g_signal_connect(entry1, "activate", G_CALLBACK(on_open_from_entry), « text1");

    gtk_widget_show_all(window);

    gtk_main();
```

A expliquer ?



Retours sur quelques notions

Makefile

- Un fichier de config texte spécifiant des cibles, des dépendances et commandes associées

- Parsées par l'**utilitaire make**

Expliquer chaque bloc ?

```
# Les variables
CC = gcc
CFLAGS = -Wall -g
OBJ = main.o math_ops.o

# Les cibles, dépendances, commandes
program: $(OBJ)
	$(CC) $(OBJ) -o program

main.o: main.c math_ops.h
	$(CC) $(CFLAGS) -c main.c

math_ops.o: math_ops.c math_ops.h
	$(CC) $(CFLAGS) -c math_ops.c

clean:
	rm -f *.o program
```

Makefile plus sophistiqué

Expliquer chaque bloc ?

```
# Liste des fichiers sources et objets
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)

# Cible principale
all: myprogram

# Règle pour construire l'exécutable
myprogram: $(OBJECTS)
    gcc -o $@ $^

# Règle générique pour compiler les fichiers .c en fichiers .o
%.o: %.c
    gcc -c $< -o $@

# Cible pour nettoyer les fichiers objets et l'exécutable
clean:
    rm -f $(OBJECTS) myprogram
```

ANNEXE : MEMO MAKEFILE GCC

- Gcc
  - gcc main.c –o myprg
  - gcc –c main.c + gcc main.o –o myprg
  - gcc –g main.c –o myprog + gdb ./myprg
  - gcc -c math_ops.c + ar crs libmath_ops.a math_ops.o + gcc main.c -L. -lmath _ops -o myprg
  - gcc -fPIC -c math_ops.c + gcc -shared -o libmath _ops.so math_ops.o + gcc main.c -L. -lmath _ops -o program + LD_LIBRARY_PATH=. ./program

- Make
  - % représente un modèle qui peut correspondre à n'importe quelle chaîne de caractères dans le nom de fichier. Utilisation :

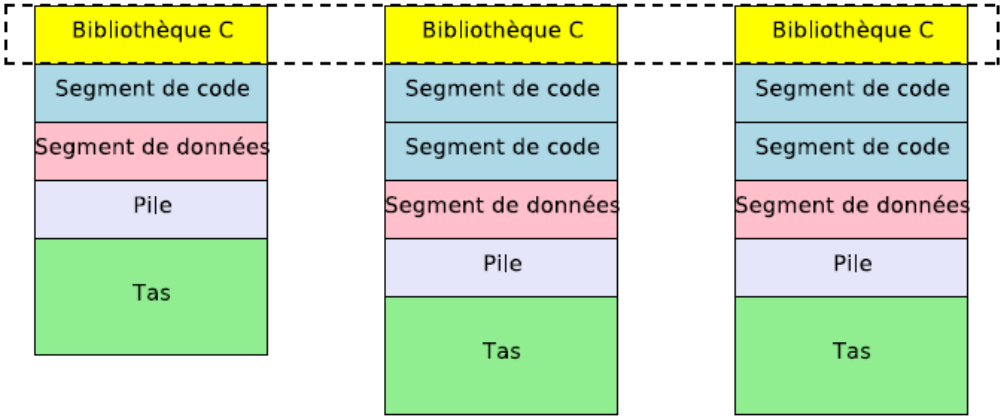
    ```
    %.o : %.c
        gcc  -c  $<  -o  $@
    ```
  - $@ : Nom de la cible
  - $^ : Liste des dépendances
  - $< : La première dépendance d'une règle.
  - SRCS = $(wildcard *.c) renvoie tous les fichiers .c dans le répertoire courant et les stocke dans la variable SRCS.
  - OBJS = $(patsubst %.c, %.o, $(SRCS)) : substitution de pattern

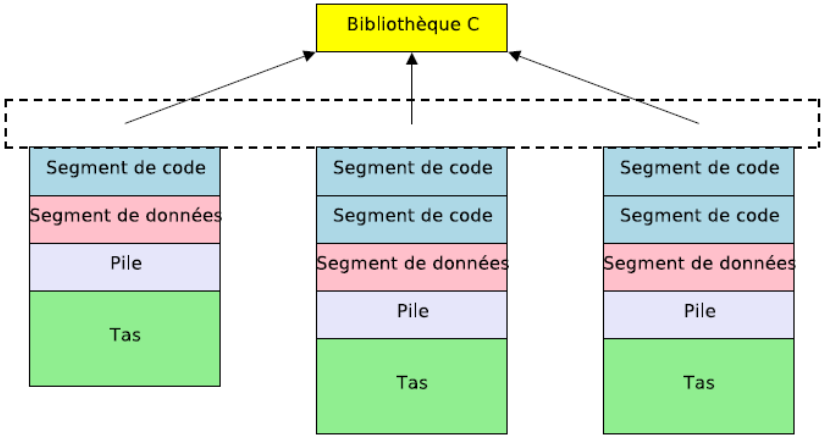
Création puis utilisation de librairies

- **Création**
  - Statiques : **ar -crs libmylib.a file1.o file2.o**
  - Dynamique : **gcc -shared -fPIC -o libmylib.so file1.o file2.o**

Différences ?

- **Utilisation**
  - **gcc main.c -L./lib –lmylib1 –lmylib2 -o myprogram**
  - **gcc -o my_program my_program.c `pkg-config --cflags --libs gtk4`**





Bilan du cours

- Précompilation, compilation, édition de liens
- Création / utilisation de librairies
- Makefile, commentaires, doxygen
- Passage par valeurs ou par adresse
- Pointeurs
- Définition de types et tableaux (listes chainées)
- Allocation mémoire, libération → Pile vs Tas
- Pointeurs de fonctions
- Débugage, tests, erreurs
- Fichiers
- GUI

Bilan du cours

- **Examen le 25 novembre**
- Sans support papier / numérique