Tutoriel GTK+

E2I-3 PJI

Année 2009-2010

I Introduction

Le but de ce tutorial est d'apprendre à maîtriser **GTK+**. GTK+ (le GIMP ToolKit) est une bibliothèque écrite en C pour créer des interfaces graphiques. Vous pouvez créer fenêtres, boutons, menus, barres d'outils, barres de progression, etc ...GTK+ est sous licence libre LGPL, vous pouvez donc l'utiliser pour développer des programmes libres ou commerciaux.

GTK+ utilise le principe de la **Programmation Orientée Objet**(POO). La POO est un mode de programmation basée sur des objets. Plutôt que le déroulement du programme soit régie par une suite d'instruction, en POO il est régi par la création, l'interaction et la destruction d'**objets**, en résumé la vie des objets qui le composent. Un objets est une entité (le parallèle avec la vie courante est simple, un objet pourrait être un ordinateur), en C on pourrait se représenter un objet comme une structure de données. Un objet est composé de propriétés (ce qui correspond à nos variables) et de méthodes (des fonctions). Pour créer un objet, on fait appel à son constructeur, il s'agit d'une fonction qui a pour but de créer et d'initialiser l'objet. A la fin de sa vie, l'objet doit être détruit, c'est le but du destructeur.

Voici quelques choix qui ont été faits par l'équipe de développement de GTK+ :

- Tous les objets manipulés sont des `GtkWidget`, pour que GTK+ vérifie qu'il s'agit du bon type d'objet lors de l'appel à une fonction, chaque type de widget possède une macro de la forme `GTK_TYPE_DU_WIDGET` à utiliser pour transtyper vos widgets.
- Les noms de fonctions sont de la forme `gtk_type_du_widget_action`, ceci rend les noms des fonctions très explicites, en contre partie ils peuvent être très longs.
- La gestion des événements qui modifient le comportement de l'application (clique de souris, pression d'une touche du clavier...) se fait grâce à l'utilisation de fonctions de rappel (callback). Pour cela, on connecte les widgets à des fonctions qui seront appelées lorsque l'événement survient.

Documentation GTK+

`http://library.gnome.org/devel/gtk/unstable/index.html`

II Notre premier programme

Un programme qui utilise GTK+ est un programme écrit en C avant tout, il contient donc le code de base de tout programme C :

main.c
```
#include <stdlib.h>

int main (int argc, char **argv)
{
  /* ... */
  return EXIT_SUCCESS;
}
```

Pour pouvoir utiliser les fonctions de GTK+, il faut bien sûr inclure le fichier d'en-tête correspondant :

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

La première chose à faire est d'initialiser la machinerie GTK+ grâce à la fonction `gtk_init` :

```
void gtk_init (int *argc, char ***argv);
```

Cette fonction reçoit les arguments passés en ligne de commande. Ensuite, il nous faut créer tous les widgets dont nous avons besoin, si nécessaire modifier leurs paramètres par défaut, les connecter à des fonctions callback et ensuite demander leur affichage (tout ceci sera explicité dans la suite du tutoriel). Une fois la fenêtre principale crée, il suffit de lancer la boucle principale de GTK+ :

```
void gtk_main (void);
```

Cette fonction est une boucle sans fin (seule la fonction `gtk_main_quit` permet d'en sortir) qui se charge de gérer le déroulement de notre programme (affichage des widgets, envoi de signaux...). Voici donc notre premier programme en C/GTK+ qui ne fait rien :

main.c
```
#include <stdlib.h>
#include <gtk/gtk.h>

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

  /* Lancement de la boucle principale */
  gtk_main();
  return EXIT_SUCCESS;
}
```

En plus de ne rien faire notre programme ne peut être terminé que de façon brutale (Ctrl+C) puisque nous n'appelons pas la fonction `gtk_main_quit`.

Makefile

Il faut ajouter au Makefile des flags pour qu'il gère la dépendance GTK+.

```
EXE = hello
SRC = main.c
OBJ = $(SRC:.c=.o)
CC  = g++

# Flags d'erreurs:
ERROR_FLAGS = -Wall -W -pedantic

# Flags pour le compilateur:
GTK_CFLAGS = $$(pkg-config --cflags gtk+-2.0)
CFLAGS = $(ERROR_FLAGS) $(GTK_CFLAGS)

# Flags pour l'editeur de lien:
GTK_LDFLAGS = $$(pkg-config --libs gtk+-2.0)
LDFLAGS = $(ERROR_FLAGS) $(GTK_LDFLAGS)

all: $(EXE)

$(EXE): $(OBJ)
        $(CC) -o $@ $^ $(LDFLAGS)

%.o: %.cpp
        $(CC) -o $@ -c $< $(CFLAGS)

clean:
        @rm -rf $(EXE) $(OBJ) *~
```

III Notre première fenêtre

La fenêtre principale est représentée par la classe `GtkWindow`. La création d'une fenêtre se fait simplement en appelant le constructeur de cette classe, la fonction `gtk_window_new` :

```
GtkWidget *gtk_window_new (GtkWindowType type);
```

Le seul paramètre de cette fonction est le type de fenêtre souhaité. Il n'y a que deux possibilités :

- `GTK_WINDOW_TOPLEVEL` : une fenêtre classique (c'est la base de notre application)
- `GTK_WINDOW_POPUP` : il s'agit d'une fenêtre avec seulement l'espace de travail (pas de bordure ni de menu système).

Cette fonction renvoie un pointeur sur une structure de type `GtkWindow` (transformer en `GtkWidget` grâce au polymorphisme) qui est l'entité qui va nous servir à manipuler notre fenêtre.

La définition du titre de la fenêtre se fait en appelant la fonction `gtk_window_set_title` :

```
void gtk_window_set_title(GtkWindow *window, const gchar *title);
```

Si vous essayez d'écrire un code maintenant, vous ne verrez rien, pourquoi ? Tout simplement parce qu'il faut préciser à GTK+ qu'il faut rendre notre fenêtre visible grâce à la fonction `gtk_widget_show` :

```
void gtk_widget_show (GtkWidget *widget);
```

Voici le code qui permet d'afficher notre première fenêtre :

```
main.c
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

#include "interface.h"

int main(int argc, char **argv){
  GtkWidget *p_window = NULL;

  /* Initialisation de GTK+ */
  gtk_init (&argc, &argv);

  /* Creation de la fenetre principale */
  create_window();

  /* Lancement de la boucle principale */
  gtk_main ();

  return EXIT_SUCCESS;
}
```

```
interface.h
#include <gtk/gtk.h>
void create_window();

interface.c
#include "interface.h"

void create_window(){
  GtkWidget *p_window = NULL;

  /* Creation de la fenetre principale */
  p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(p_window), "Hello");

  /* Affichage de la fenetre principale */
  gtk_widget_show (p_window);
}
```

Dans l'exemple précédent, si vous avez essayé de terminer l'application en fermant la fenêtre (Alt+F4 ou en cliquant sur la petite croix), vous vous êtes peut-être aperçu que l'application tournait encore en fond, c'est le même problème que pour le chapitre précédent : on ne fait pas appel à `gtk_main_quit`. Mais comment appeler cette fonction puisque qu'une fois `gtk_main` appelée nous ne pouvons rien faire ? C'est là qu'intervient le mécanisme des callback. A chaque événement qui se produit, la bibliothèque `GObject` produit un signal, si nous souhaitons modifier le comportement par défaut du signal, il suffit de connecter notre fonction callback à l'événement souhaité :

```
#define g_signal_connect(instance, detailed_signal, c_handler, data);
```

Cette macro permet d'intercepter l'événement `detailed_signal` de l'objet instance grâce à la fonction `c_handler` qui doit être du type `GCallback` :

```
void (*GCallback) (void);
```

Les fonctions callback peuvent bien sûr recevoir des paramètres mais leur nature et leur nombre dépendent du contexte, tout est géré par la bibliothèque `GObject`. Le prototype le plus courant pour une fonction de rappel est le suivant :

```
void callback (GtkWidget *p_widget, gpointer *user_data);
```

Dont le premier paramètre est le widget qui a reçu le signal et le second correspond au paramètre data de la fonction `g_signal_connect`, qui nous permet de passer des informations aux fonctions callback. Pour finir proprement notre application, il suffit d'intercepter le signal destroy de notre fenêtre et d'y assigner la fonction `gtk_main_quit` qui ne prend pas d'argument. Voici donc notre première application fonctionnelle :

```
interface.c
#include "interface.h"

void create_window(){
  GtkWidget *p_window = NULL;

  /* Creation de la fenetre principale */
  p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(p_window), "Hello");
  g_signal_connect (G_OBJECT (p_window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  /* Affichage de la fenetre principale */
  gtk_widget_show (p_window);
}
```

Il est plus courant de créer notre propre fonction de rappel pour quitter le programme ; ceci permet de libérer la mémoire allouée, fermer les fichiers ouverts.

```
interface.c
#include "interface.h"
#include "callback.h"

void create_window(){
  GtkWidget *p_window = NULL;

  /* Creation de la fenetre principale */
  p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(p_window), "Hello");
  g_signal_connect (G_OBJECT (p_window), "destroy",
                    G_CALLBACK (cb_quit), NULL);

  /* Affichage de la fenetre principale */
  gtk_widget_show (p_window);
}
```

```
callback.h
#include <gtk/gtk.h>
void cb_quit (GtkWidget *p_widget,
              gpointer user_data);

callback.c
#include "callback.h"

void cb_quit (GtkWidget *p_widget,
              gpointer user_data)
{
  gtk_main_quit();

  /* Parametres inutilises */
  (void)p_widget;
  (void)user_data;
}
```

Le préfixe `cb_` permet de différencier les fonctions callback, qu'il est préférable de regrouper dans un ou plusieurs fichiers séparés.

A la fin de la fonction, je transtype les paramètres inutilisés pour éviter que mon compilateur m'indique des variables non utilisées, c'est le problème avec les fonctions callback, l'API nous impose un prototype.

Exercice

Vous devez personnaliser la fenêtre de la manière suivante :
- Définir la taille par défault
- Positionner la fenêtre au centre de l'écran

IV Fermer notre fenêtre grâce aux boutons

Maintenant que notre programme se termine proprement, il est plus convivial de proposer à l'utilisateur un bouton pour quitter. Pour créer un bouton, il existe plusieurs possibilités :

```
GtkWidget *gtk_button_new (void);
GtkWidget *gtk_button_new_with_label (const gchar *label);
GtkWidget *gtk_button_new_with_mnemonic (const gchar *label);
GtkWidget *gtk_button_new_from_stock (const gchar *stock_id);
```

La première fonction crée un bouton qui peut servir pour contenir n'importe quel autre widget (label, image...). Si vous souhaitez créer un bouton avec du texte dessus, utilisez directement la seconde fonction qui prend en argument le texte à afficher. Si vous souhaitez ajouter un raccourci clavier (accessible avec la touche Alt), la troisième fonction permet d'en spécifier un en faisant précéder une lettre (généralement la première) d'un underscore '_' (si vous souhaitez afficher le caractère '_', il faut en mettre deux). Enfin la dernière fonction permet d'utiliser les Stock Items qui sont un ensemble d'éléments prédéfinis par GTK+ pour les menus et barres d'outils. Le seul paramètre de cette fonction est le nom de l'élément et GTK+ se charge d'afficher l'icône appropriée ainsi que le texte suivant la langue choisie et un raccourci.

C'est bien sûr cette dernière fonction que nous allons utiliser et ce le plus souvent possible. Voici le code pour créer un tel bouton :

```
GtkWidget *p_button = NULL;

p_button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
```

Pour obtenir tous les types de stocks items disponibles : `GtkStockItem`.

Comme pour la fenêtre si l'on veut voir quelque chose, il faut demander à GTK+ d'afficher notre widget :

```
gtk_widget_show(p_button);
```

Mais pas seulement ! En effet, il faut préciser à GTK+ où doit être affiché notre bouton. Pour cela, la classe `GtkWindow` est dérivée de la classe `GtkContainer` qui est, comme son nom le laisse penser, un conteneur pour widget. Pour ajouter un widget, il suffit de faire appel à la fonction `gtk_container_add` :

```
void gtk_container_add (GtkContainer *container, GtkWidget *widget);
```

Le premier paramètre de cette fonction est un `GtkContainer`, or nous souhaitons passer notre fenêtre qui est un `GtkWidget`, pour ne pas avoir de problèmes, il faut utiliser le système de macro offert par GTK+ pour transtyper notre `GtkWidget` en `GtkContainer` (eh oui en C, le polymorphisme a ses limites) :

```
gtk_container_add (GTK_CONTAINER (p_window), p_button);
```

Ce mécanisme de transtypage permet à GTK+ de vérifier que notre `GtkWidget` est bien compatible avec l'utilisation que l'on souhaite en faire. En cas d'échec, GTK+ nous prévient en affichant un message dans la console :

```
(gtk_bouton.exe:1044): Gtk-CRITICAL **: gtk_container_add: assertion 'GTK_IS_CONTAINER (container)' failed
```

Pour finir, il faut connecter notre bouton pour appeler notre fonction `cb_quit` lorsque l'utilisateur clic dessus (ce qui correspond à l'événement "clicked") :

```
g_signal_connect (G_OBJECT (p_button), "clicked", G_CALLBACK (cb_quit), NULL);
```

Pour éviter d'appeler la fonction `gtk_widget_show` pour tous les widgets de notre application, on peut se contenter d'un seul appel à la fonction `gtk_widget_show_all` qui permet d'afficher tous les widgets contenus dans celui passé à la fonction.

```
interface.c
#include "interface.h"
#include "callback.h"

void create_window(){
  GtkWidget *p_window = NULL;

  /* Creation de la fenetre principale */
  p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(p_window), "Hello");
  g_signal_connect (G_OBJECT (p_window), "destroy",
                    G_CALLBACK (cb_quit), NULL);

  GtkWidget *p_button = NULL;

  p_button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
  gtk_container_add (GTK_CONTAINER (p_window), p_button);
  g_signal_connect (G_OBJECT (p_button), "clicked",
                    G_CALLBACK (cb_quit), NULL);

  /* Affichage de la fenetre principale */
  gtk_widget_show_all (p_window);
}
```

Le préfixe `cb_` permet de différencier les fonctions callback, qu'il est préférable de regrouper dans un ou plusieurs fichiers séparés.

A la fin de la fonction, je transtype les paramètres inutilisés pour éviter que mon compilateur m'indique des variables non utilisées, c'est le problème avec les fonctions callback, l'API nous impose un prototype.

Exercice

Vous devez faire un programme qui fait ceci lorsque on clique sur la croix :
- Ecrire le titre de la fenêtre dans la console
- Quitter le programme

V Comment afficher plusieurs widgets

Dans la partie précédente, nous avons réussi à afficher un bouton dans la fenêtre de notre application. Si vous avez essayé d'ajouter un second widget, GTK+ à dû vous répondre poliment :

```
(gtk_box.exe:492) : Gtk-WARNING **: Attempting to add a widget with type
GtkButton to a GtkWindow, but as a GtkBin subclass a GtkWindow can only contain
one widget at a time; it already contains a widget of type GtkButton
```

Les plus anglophiles d'entre vous auront compris que GTK+ n'accepte pas l'ajout un second widget dans notre fenêtre tout simplement parce qu'il y en a déjà un et que la sous classe `GtkBin` (classe mère de notre `GtkWindow`) ne peut contenir qu'un seul widget.

Pour contourner ce problème, il faut commencer par créer un widget qui accepte d'en contenir plusieurs autres, pour ensuite y ajouter tout ce dont nous avons besoin. Pour l'instant, nous utiliserons qu'un seul widget (il existe trois classes qui permettent ceci), il s'agit des `GtkBox`. Elle consiste à créer une boîte puis à y ajouter les widgets les uns après les autres (soit au début soit à la fin). Il existe deux classes héritant de GtkBox : les `GtkVBox` qui empilent les widgets dans le sens vertical et les `GtkHBox` qui font de même mais dans le sens horizontal. Les fonctions pour manipuler ces deux classes sont les mêmes, seule la fonction pour les créer portent un nom différent :

```
GtkWidget *gtk_vbox_new (gboolean homogeneous, gint spacing);
GtkWidget *gtk_hbox_new (gboolean homogeneous, gint spacing);
```

Le paramètre homogeneous permet de réserver pour chaque widget une zone de taille identique (zone que le widget n'est pas obligé de remplir) et spacing permet d'ajouter une bordure en pixels (espacement autour de la `GtkBox`). Ensuite, il suffit d'ajouter les différents widgets grâce aux fonctions :

```
void gtk_box_pack_start (GtkBox *box, GtkWidget *child, gboolean expand, gboolean fill, guint padding);
void gtk_box_pack_end (GtkBox *box, GtkWidget *child, gboolean expand, gboolean fill, guint padding);
```

Qui ajoutent réciproquement le widget child au début et à la fin de box. Le paramètre expand permet au widget d'avoir le plus de place possible (si le paramètre homogeneous vaut TRUE, cette option a aucun effet). Si plusieurs widget ont ce paramètre à TRUE, ils se partagent de façon égale l'espace. L'option fill permet au widget de remplir tout l'espace qui lui ait réservé. Pour réellement comprendre l'influence de ces paramètres, il faut faire des tests en modifiant un à un chaque paramètre et observer les effets d'un redimentionnement de la fenêtre principale. Le plus gênant avec cette méthode c'est qu'il faut jongler entre les `GtkHBox` et les `GtkVBox` en les imbriquant pour obtenir le résultat souhaiter : n'hésitez pas à utiliser une feuille et un crayon ;)

Pour en revenir à notre éditeur de texte, nous allons préparer notre application à contenir les futurs widgets. Nous utilisons un `GtkVBox` pour l'ensemble des widgets (il s'agit de la boîte principale qui pourra contenir d'autres `GtkContainer` selon nos besoins) :

```
interface.c
#include "interface.h"
#include "callback.h"

void create_window(){
  GtkWidget *p_window = NULL;

  /* Creation de la fenetre principale */
  p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(p_window), "Hello");
  g_signal_connect (G_OBJECT (p_window), "destroy",
                    G_CALLBACK (cb_quit), NULL);

  /* Creation du conteneur principal */
```

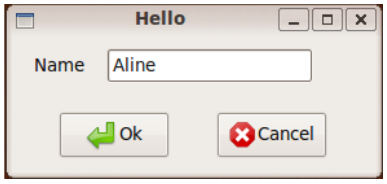

FIG. 1: Interface graphique de l'exercice 1.

```
  GtkWidget *p_main_box = NULL;
  p_main_box = gtk_vbox_new (FALSE, 0);
  gtk_container_add (GTK_CONTAINER (p_window), p_main_box);

  /* Creation du bouton "Quitter" */
  GtkWidget *p_button = NULL;
  p_button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
  gtk_container_add (GTK_CONTAINER (p_main_box), p_button);
  g_signal_connect (G_OBJECT (p_button), "clicked",
                    G_CALLBACK (cb_quit), NULL);

  /* Affichage de la fenetre principale */
  gtk_widget_show_all (p_window);
}
```

Exercice

1 - Vous devez faire un programme avec l'interface graphique présenté sur la figure 1. Lorsque on clique sur :

- le bouton Ok, la phrase "Hello *Aline*!" doit être affiché dans la console. Bien sûr que *Aline* est le valeur informé dans le `GtkEntry`.
- le bouton Cancel, le contenu écrit dans le `GtkEntry` doit être supprimé.
- la croix, le programme doit être quitté.

2 - Vous devez faire un programme avec l'interface graphique présenté sur la figure 2. Lorsque on clique sur :

- le bouton Ok, la phrase "Hello *name*! Your address is *address*!" doit être affiché dans la console. *name* et *address* sont les valeurs informés dans les `GtkEntrys`.
- le bouton Cancel, le contenu écrit dans les `GtkEntrys` devent être supprimés.
- la croix, le programme doit être quitté.

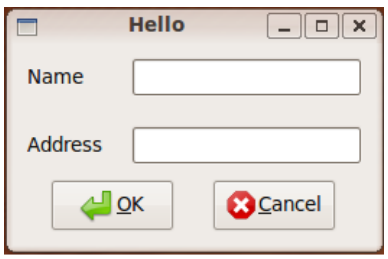

FIG. 2: Interface graphique de l'exercice 2.