1 Récapitulatif

1.1 Les 4 lignes à connaître

```
#include <gtk/gtk.h>          // Inclure les fichiers d'entête
main(int argc, char** argv)
{
    gtk_init(&argc, &argv);   // Initialiser l'environnement gtk
    gtk_main();               // Lancer la boucle événementielle
}
```

Dans une fonction callback :

```
    gtk_main_quit();          // Quitter la boucle infinie
```

1.1.1 void gtk_init(int*, char***)

Indispensable avant de faire quoi que ce soit en gtk : c'est seulement après avoir fait appel à la fonction gtk_init que l'on peut commencer à manipuler les fonctions gtk. Cette fonction reprend les arguments de la fonction main.

1.1.2 void gtk_main()

Lance la boucle événementielle infinie qui capte les signaux émis par les objets graphiques.

1.1.3 void gtk_main_quit()

Ferme une boucle événementielle lancée par ailleurs. Puisque la boucle lancée dans gtk_main() est infinie, elle ne peut être stoppée que par un événement externe.

1.2 Types manipulés

nom	type	déclaration	construction	cast
Fenêtre	GtkWindow*	GtkWidget* pFenetre;	pFenetre = gtk_window_new(...);	GTK_WINDOW(pFenetre);
Libellé	GtkLabel*	GtkWidget* pLabel;	pLabel = gtk_label_new(...);	GTK_LABEL(pLabel);
Bouton	GtkButton*	GtkWidget* pBouton;	pBouton = gtk_Button_new(...);	GTK_BUTTON(pBouton);

Tous les types que l'on manipule sont héritiers de GtkWidget et leur fonction de création (constructeur) renvoie toujours un pointeur sur GtkWidget.

⇒
- *toujours déclarer les objets Gtk comme des pointeurs sur GtkWidget* ;
- *toujours utiliser les macros de cast pour préciser le type de l'objet en fonction du contexte.*

*// Déclaration d'un bouton (plus exactement, d'un pointeur sur un bouton) => déclaration GtkWidget**
GtkWidget* pBouton;
// Instanciation du bouton => constructeur de GtkButton (qui renvoie un GtkWidget)*
pBouton = gtk_button_new();
// Attribution d'un libellé au bouton par "void gtk_button_set_label(GtkButton, const gchar*);"*
gtk_button_set_label(GTK_BUTTON(pBouton), "Quitter");
// Attachement d'un callback par "gulong gtk_signal_connect(GtkObject, const gchar*, GCallback, gpointer);"*
gtk_signal_connect(GTK_OBJECT(pBouton), "clicked", gtk_main_quit, NULL);
// Affichage de l'objet par "void gtk_widget_show(GtkWidget);"*
gtk_widget_show(GTK_WIDGET(pBouton)); *// ou* gtk_widget_show(pBouton);

1.3 Fonction utiles

1.3.1 Pour afficher

Afficher un objet graphique :	void gtk_widget_show(GtkWidget*);
Ex. :	`gtk_widget_show(pLabel);`
Afficher un objet graphique *et* son contenu :	void gtk_widget_show_all(GtkWidget*);
Ex. :	`gtk_widget_show_all(pFenetre);`
Associer un objet graphique à une fenêtre :	void gtk_container_add(GtkContainer*, GtkWidget*);
Ex. :	`gtk_container_add(GTK_CONTAINER(pFenetre), pLabel);`

On ne peut afficher un objet s'il n'est pas (directement ou indirectement) contenu dans une fenêtre, elle-même affichée. Par exemple, l'instruction "gtk_widget_show(pLabel);" n'aura aucun effet si le label pLabel n'est pas contenu dans une fenêtre, elle-même affichée. En revanche, si pLabel est contenu dans pFenetre, on peut l'afficher en écrivant :

gtk_widget_show(pLabel); // on affiche pLabel
gtk_widget_show(pFenetre); // on affiche pFenêtre (ce qui rend effectif l'affichage de pLabel)

Ou encore, en une seule instruction :
gtk_widget_show(pFenetre); // on affiche pFenêtre et tous les objets qu'elle contient

1.3.2 Pour interagir

Associer une fonction à un couple (signal, objet)

fonction :	gulong gtk_signal_connect(GtkObject* object, const gchar* name, GCallback func, gpointer data);
appel :	`gtk_signal_connect(GTK_OBJECT(nomObjet), nomSignal, G_CALLBACK(nomFonction), nomDonnee);`

Conséquence de cette instruction : lorsque l'objet graphique *nomObjet* émettra le signal *nomSignal*, la boucle événementielle appellera la fonction *nomFonction* en lui passant pour arguments, d'une part, l'objet graphique *nomObjet* qui a émis le signal et, d'autre part, la donnée *nomDonnee* (qui peut être nulle) :

nomFonction(nomObjet, nomDonnee);

Ce dernier argument est un pointeur générique de type gpointer (équivalent GLib du type void*).

Définir une fonction de callback

En conséquence de l'attachement fait par la fonction gtk_signal_connect(), la seule restriction pour définir une fonction de callback *func* est qu'elle doit avoir la signature suivante :

void func(GtkWidget* object, gpointer data);

1.4 Types et fonctions supplémentaires qui seront utilisés dans ce TP

GtkLabel	
GtkWidget* gtk_label_new(const gchar* str);	*crée un objet GtkLabel de libellé* str
void gtk_label_set_label(GtkLabel* label, const gchar* str);	*affecte le libellé* str *au label* label

GtkButton	
fonction	GtkWidget* gtk_button_new(); *crée un objet GtkButton*
fonction	void gtk_button_set_label(GtkButton* button, const gchar* label); *affecte le libellé* label *au bouton* button
fonction	G_CONST_RETURN gchar* gtk_button_get_label(GtkButton* button); *récupération du libellé du bouton* button
signal	"clicked", *émis par clic de la souris / clavier sur le bouton*

GtkWindow	
fonction	GtkWidget* gtk_window_new(GtkWindowType type); *crée un objet GtkWindow* paramètre type == GTK_WINDOW_TOPLEVEL
fonction	void gtk_window_set_title(GtkWindow* window, const gchar* title); *affecte le titre* title *à la fenêtre* window
fonction	void gtk_window_set_default_size(GtkWindow* window, gint width, gint height); *affecte la largeur* width *et la hauteur* height *à la fenêtre* window *(valeurs en pixels)*
signal	"destroy", *émis lorsque la fenêtre est fermée*

Exercice 1 : prendre ses marques

1. Faire un programme Gk qui affiche une fenêtre (utiliser “gtk_widget_show()”).
2. Associer à la fenêtre et au signal de destruction la fonction “gtk_main_quit()”.
3. Et si la fenêtre avait pour titre “Vive Gtk !" ?
4. On pourrait aussi donner à la fenêtre une taille respectable : 500 pixels de largeur, 250 pixels de hauteur ?

Exercice 2 : labelliser

1. Créer le label “C’est encore plus joli comme cela, non ?”.
2. Affichez le label...
3. ... affichez-le *vraiment* !
4. Et avec “gtk_widget_show_all()”, ça marche comment ?
5. Finalement, changez le label : écrivez plutôt “Y’en avait marre du HTML...” (utiliser “gtk_label_set_label()”)

Exercice 3 : bouton

1. Créer un bouton.
2. Ajoutez ce bouton à la fenêtre ; que se passe-t-il ? Retirez alors le label de la fenêtre.
3. Écrivez “Quitter” sur le bouton.
4. Joignez donc le geste à la parole : faites-en sorte que l’on quitte l’application en cliquant sur le bouton.
5. Changez le libellé du bouton en “Traduire”. Écrire aussi une fonction “traduction” qui :
   (a) ne renvoie rien (type retour “void”) ;
   (b) prend deux arguments, le premier, pBouton, de type GtkWidget* et le second, pData, de type gpointer ;
   (c) opère le traitement suivant (on suppose que pBouton pointe sur une structure GtkButton) : si le libellé de pButon est “Traduire”, le changer en “Translate” et réciproquement.
6. Faire en sorte que, lorsque l’on clique sur le bouton, son libellé passe de français en anglais et réciproquement.
7. Un peu plus délicat à présent : puisque le bouton est capable de traduire son propre libellé lorsque l’on clique dessus, pourquoi ne traduirait-il pas aussi le titre de la fenêtre ?