Université de Nice-Sophia Antipolis — juin 2018
Polytech/Xidian — Durée: 1h30

Travaux Pratiques – Séance n° 13

Convertisseur Euros/Yuans

1 Présentation

Le but de ce TD est de construire une interface graphique pour un convertisseur Euros/Yuans selon le taux de change en vigueur. Ce convertisseur aura l'apparence donnée par la figure 1.

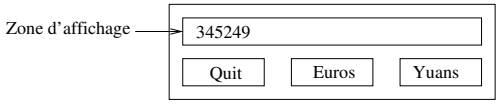

FIGURE 1 – Interface graphique du convertisseur.

La manipulation du convertisseur se fera avec le clavier pour entrer la somme d'argent à convertir, et la souris. Un clic sur :

— la touche « Euros » affiche la somme convertie en euros :
— la touche « Yuans » affiche la somme convertie en dollars :
— la touche « Quit » met fin au programme.

La plupart des environnements de programmation offre des boîtes à outils graphiques (graphical toolkit) pour programmer des interfaces graphiques. Une boîte à outils est un ensemble de composants graphiques préfabriqués (boutons, menus déroulants, ascenseurs, etc.), appelés *widgets*, qu'il suffit d'assembler pour composer l'interface graphique. La boîte à outils graphique que nous allons utiliser s'appelle `libsx` et a l'avantage d'être très simple à utiliser.

La construction de notre programme se fera en deux temps :

1. assemblage des composants graphiques ;
2. connexion des composants graphiques avec les actions de change dollars/euros.

2 Assemblage des composants graphiques

Pour `libsx`, un composant graphique est de type `Widget`. `Libsx` propose plusieurs types de composants graphiques. Pour notre programme, nous en utiliserons deux : les entrées de texte et les boutons.

Une entrée est une simple zone graphique qui permet l'édition d'un message de type chaîne de caractères. Elle va nous servir pour visualiser et éditer la valeur courante traitée par le convertisseur. La fonction `MakeStringEntry` crée une entrée. Elle possède quatre paramètres. Son prototype est le suivant :

```
Widget MakeStringEntry(char *txt, int size, StringCB func, void *data);
```


Pour l'instant, seul le deuxième paramètre nous intéresse, c'est la taille en pixels de l'entrée. Pour les trois autres, vous passerez la valeur `NULL` au moment de l'appel de la fonction.

Un bouton est un widget qui affiche un message et qui peut prendre deux états, enfoncé ou non enfoncé, en réaction aux clics de la souris. Les boutons nous serviront à représenter toutes les touches de la calculette. La fonction `MakeButton` crée un bouton. Son prototype est le suivant :

```
Widget MakeButton (char *s, ButtonCB func, void *data);
```

Pour l'instant, seul le premier paramètre nous intéresse, il correspond au texte à afficher sur le bouton. Pour les deux autres, vous passerez la valeur `NULL` au moment de l'appel de la fonction.

Une fois les widgets créés, il faut les assembler en les positionnant les uns par rapport aux autres, pour obtenir la figure 1. C'est la fonction `SetWidgetPos` qui se charge de ce travail. Son prototype est le suivant :

```
void SetWidgetPos (Widget w, int pos1, Widget w1, int pos2, Widget w2);
```

Cette fonction assemble le widget `w` en position `pos1` par rapport au widget `w1` et en position `pos2` par rapport au widget `w2`. Les positions valides sont `PLACE_UNDER` et `PLACE_RIGHT`. Ainsi, l'appel

```
SetWidgetPos(a, PLACE_RIGHT, b, PLACE_UNDER, c);
```

assemble le widget `a` à droite du widget `b` et sous le widget `c`.

Il est également possible de placer un widget par rapport à un seul autre widget. Dans ce cas, les deux derniers paramètres prendront respectivement les valeurs `NO_CARE` et `NULL`.

Structure du programme

La fonction `main` de votre programme aura dans un premier temps la forme suivante :

```
#include <stdio.h>
#include <stdlib.h>
#include <libsx.h>

int main (int argc, char **argv)
{
  if (OpenDisplay(argc, argv) == 0) {
    fprintf(stderr,"Can't open display\n");
    return EXIT_FAILURE;
  }

  init_display(argc, argv, NULL);
  MainLoop();
  return EXIT_SUCCESS;
}
```

Le premier test vérifie que l'affichage sur l'écran est possible. La fonction `init_display` contiendra le code de création des widgets et leur assemblage. Enfin, la fonction `MainLoop` lance le fonctionnement interactif de l'application.

Au dessus de la fonction `main`, vous déclarerez la fonction `init_display` comme suit :

```
void init_display (int argc, char **argv, void *d)
{
   /* création et assemblage des widgets */

   `.... à vous de jouer .....`

   /* pour gérer les couleurs */
   GetStandardColors();
```

```
    /* pour afficher l'interface */
    ShowDisplay();
}
```

1) Placez dans un fichier `conv.c` le code C précédent. Puis, à l'aide des fonctions `MakeStringEntry`, `MakeButton` et `SetWidgetPos`, complétez la fonction `init_display` afin de produire l'interface donnée par la figure 1.

2) Pour compiler votre programme, il faut indiquer au compilateur que vous utilisez la boîte à outils `libsx`. Pour cela, vous ajouterez `-lsx` à la fin de votre commande de compilation.

Pour compiler le programme, il sera plus simple d'utiliser la commande `make` avec le fichier `makefile` donné en annexe. Il suffit de taper la commande :

```
make
```

Le programme exécutable qui a été fabriqué se nomme `conv`.

Vous pouvez maintenant exécuter le programme :

```
./conv
```

3 Donner vie à l'interface graphique

3.1 Les callbacks

Les widgets que vous venez de créer sont sans « vie », ils ne produisent aucune action. Pour qu'ils fassent des actions, il faut leur associer des fonctions, appelées *callbacks*, qui sont déclenchées lorsqu'un *événement* se produit sur le widget. Un événement est, par exemple, l'appui sur une touche du clavier ou un clic de souris.

Dans notre application, il faudra associer un callback à chacun des boutons de l'interface. Pour `libsx`, un callback associé à un bouton est une fonction dont le prototype est le suivant :

```
void func(Widget, void *);
```

Prenons l'exemple du bouton « Quit ». L'action à déclencher lorsque l'on appuie sur ce bouton est de terminer l'exécution du programme par un simple appel à la fonction `exit`. Le callback à associer au bouton « Quit », que nous appellerons `quit`, s'écrira simplement

```
/* Callback bouton quit.
   Rôle : terminer l'application
*/
void quit(Widget w, void *d) {
   exit(EXIT_SUCCESS);
}
```

et son prototype est :

```
void quit(Widget, void *);
```

Le premier paramètre est le widget qui a déclenché le callback (ici le bouton Quit). Le second est un pointeur sur une donnée partagée (voir la section suivante).

3) Placez la fonction `quit` dans le fichier `callbacks.c` et son prototype dans `callbacks.h`.

L'association entre le callback et le widget se fait au moment de la création du widget. Dans le cas d'un bouton, il suffit de passer le nom du callback comme deuxième paramètre de la fonction `MakeButton`.

4) Modifiez l'appel à `MakeButton` dans `init_display` pour avoir quelque chose de la forme :

```
BQuit = MakeButton("Quit", quit, NULL);
```

Remarque : `Bquit` est le nom de la variable de type widget qui désigne le widget « Quit ». Ce nom de variable est quelconque.

5) Placez **#include** *"callbacks.h"* en tête du fichier `calc.c` et recompilez votre programme. Attention, votre programme comporte maintenant deux fichiers sources.

6) Exécutez votre programme. Vérifiez que le bouton « Quit » achève bien l'exécution de l'application.

3.2 Les données

L'interface graphique peut être amenée, par l'intermédiaire des callbacks, à manipuler des données de l'application. Remarquez que cela n'a pas été le cas pour le callback `quit`, mais cela le sera pour les callbacks des autres boutons de la calculette.

Par exemple, le clic sur une touche « Euros » doit convertir la valeur de la zone d'affichage en euros. Il faut donc mémoriser cette valeur et la rendre accessible aux callbacks des touches « Euros » et « Yuans ». Toutes les données accessibles par les callbacks seront placées dans une structure, dont l'adresse sera transmise lors de la création du widget.

Pour le convertisseur, il faut mémoriser la valeur à convertir et le taux de change sous forme de **double** ainsi que la zone d'affichage :

```
typedef struct {
  double valeur;
  double tauxDeChange;
  Widget ZoneAffich;
} ValeurCourante;
```

7) Placez cette déclaration dans le fichier `data.h`. Vous définirez les fonctions de manipulation de la structure de données `ValeurCourante` dans un fichier `data.c`. Définissez dans un premier temps une fonction d'initialisation de la structure de données.

Pour faire le lien entre les callbacks et les données, nous déclarerons et initialiserons une variable de type `ValeurCourante` dans la fonction `main` et nous la transmettrons à la fonction `init_display`.

```
#include <stdio.h>
#include <stdlib.h>
#include <libsx.h>
#include <callbacks.h>
#include <data.h>
#include <conv.h>

int main(int argc, char **argv) {
  ValeurCourante d;

  if (OpenDisplay(argc, argv) == 0) {
    fprintf(stderr,"Can't open display\n");
    return EXIT_FAILURE;
  }
```

```
  initValeurCourante(&d);
  init_display(argc, argv, &d);
  MainLoop();
  return EXIT_SUCCESS;
}
```

Dans la fonction `init_display`, il faudra passer l'adresse de cette variable comme troisième paramètre du constructeur de bouton, à la place de la constante `NULL`. Ainsi, pour le bouton « Yuans », on aura quelque chose de la forme :

```
void init_display (int argc, char **argv, void *d) {
   /* création et assemblage des widgets */

   ...

   BYuans = MakeButton("Yuans", dollars, d);

   ...

   /* pour gérer les couleurs */
   GetStandardColors();

   /* pour afficher l'interface */
   ShowDisplay();
}
```

Le rôle du callback `dollars` est de convertir la somme courante en dollars et d'afficher la valeur convertie dans la zone d'affichage. Le callback peut maintenant accéder à la valeur courante grâce à son second paramètre. Cette fonction a la forme suivante :

```
/*
 * Rôle : convertit la valeur courante en dollars
 *        et l'affiche dans la zone d'affichage
 */
void dollars(Widget w, void *d) {
{
    /* d désigne la ValeurCourante */
    /* convertir en dollars la valeur courante */
    ....
    /* afficher la nouvelle ValeurCourante
       dans la zone d'affichage
     */
}
```

L'affichage de la nouvelle valeur courante dans la zone d'affichage, correspond à la modification du texte du widget créé par la fonction `MakeStringEntry`. Cette modification peut être faite grâce à la fonction `SetStringEntry`. Cette fonction possède deux paramètres, le premier est le widget « entrée » à modifier, et le second est une chaîne de caractères qui représente le nouveau texte à afficher.

8) Programmez les callbacks des boutons « Yuans » et « Euros » Compilez et testez votre application.

9) Ajoutez le bouton « Taux de change » qui ouvre une fenêtre pour obtenir le taux de change euros/dollars à appliquer lors des conversions.

4 Annexe : le fichier Makefile

```
CC = gcc            # le compilateur à utiliser
CFLAGS = -ansi -Wall # les options du compilateur
LDFLAGS = -lsx      # les options pour l'éditeur de liens
SRC = conv.c callbacks.c data.c # les fichiers sources
PROG = conv         # nom de l'exécutable
OBJS =  $(SRC:.c=.o) # les .o qui en découlent
.SUFFIXES: .c .o    # lien entre les suffixes

all: $(PROG)

# étapes de compilation et d'édition de liens
# $@ la cible  $^ toutes les dépendances
$(PROG): $(OBJS)
        $(CC)  -o $@ $^ $(LDFLAGS)

callbacks.o: callbacks.h data.h
data.o: data.h
conv.o: data.h callbacks.h
# le lien entre .o et .c
# $< dernière dépendance
%.o: %.c
        $(CC) $(CFLAGS) -c $<

# pour faire propre
.PHONY: clean
clean:
        rm -f *.o *~ core $(PROG)
```