Gtkmm : TD1

Exercice 1 - Interface simple

```
1.  class Interface : public Gtk::Window {
2.  private:
3.      Gtk::VBox vb;
4.      Gtk::HBox hb1, hb2;
5.      Gtk::Label l('Interface'), l2('texte');
6.      Gtk::Entry e;
7.      Gtk::Button b1('Valider'), b2('Ok'), b3('Cancel');
8.      Gtk::Alignment align(Gtk::ALIGN_RIGHT, Gtk::ALIGN_CENTER, 0, 0);
9.  public:
10.     Interface() {
11.             add(vb);
12.             vb.pack_start(l, GTK::PACK_EXPAND_WIDGET, 0);
13.             vb.pack_start(hb1, GTK::PACK_EXPAND_PADDING, 0);
14.             hb1.pack_start(l2, GTK::PACK_EXPAND_PADDING, 0);
15.             hb1.pack_start(entry, GTK::PACK_EXPAND_WIDGET, 10);
16.             hb1.pack_start(b1, GTK::PACK_EXPAND_PADDING, 0);
17.             vb.pack_start(align, GTK::PACK_EXPAND_PADDING, 0);
18.             align.add(hb2);
19.             hb2.pack_start(b2, GTK::PACK_SHRINK, 10);
20.             hb2.pack_start(b3, GTK::PACK_SHRINK, 10);
21.             show_all();
22.     }
23. }
```

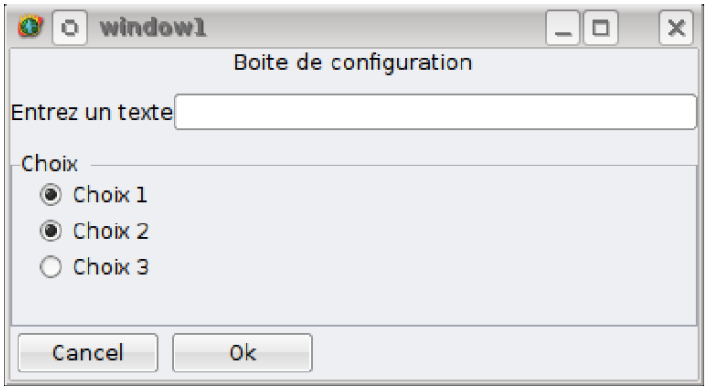
Q.1) Construire un arbre représentant la structure hiérarchique des widgets de l'interface

Q.2) En déduire l'allure générale de l'interface

NB : les l. 5 et 7 ne sont pas syntaxiquement correctes, mais simplifient l'écriture du code pour les exercices (voir les corrigés en ligne).

Exercice 2 - Conception d'une interface simple

On souhaite concevoir une classe `MaFenetre` qui hérite de la classe `Gtk::Window` et qui ait l'allure suivante :



Les widgets nécessaire à la réalisation de cette interface sont :

- Conteneurs multiples : `HBox`, `VBox`,
- Conteneurs simples : `Windows`, `Frame`, `Alignment`
- Widgets : `Button`, `RadioButton`, `Label`, `Entry`

Q.1) Construire un arbre représentant la structure hiérarchique des widgets de l'interface en tenant compte des configurations géométriques

Q.2) En déduire le code nécessaire à la réalisation de l'interface

Pour tous les widgets, l'ajout de texte se fera à l'aide de la fonction `set_label`. Pour rappel, les containers simples proposent une fonction `add()` pour ajouter un widget et les conteneurs multiples utilisent les fonctions `pack_start()` et `pack_end()`.

Exercice 3 - Connexion de signaux

Q.1) Dessiner la fenêtre obtenue par ce code ?

Q.2) Indiquer quelle(s) connexion(s) ne sont pas valide(s) ?

Q.3) Expliquer ce qui s'affichera lors qu'on cliquera sur chacun des boutons valides ?

```
1.  void on_button_clicked() {
2.       std::cout << "fonction globale\n";
3.  };
4.
5.  class SomeButton : public Gtk::Button {
6.  public:
7.      SomeButton (const Glib::ustring& label, bool mnemonic=false) : Gtk::Button(label,mnemonic) {};
8.
9.       void on_clicked() {
10.                      std::cout << "fonction on_clicked() de "<< get_label() <<"\n";
11.     }
12.      void on_button_clicked() {
13.                      std::cout << "fonction on_button_clicked() de "<< get_label() <<"\n";
14.     }
15.  };
16.
17.  class some_class
18.  {
19.  public:
20.      void on_button_clicked() {
21.                      std::cout << "fonction on_button_clicked de some_class\n";
22.     };
23.  };
24.
25.  some_class some_object;
26.
27.  int main(int argc, char** argv)
28.  {
29.    Gtk::Main kit(argc, argv);
30.
31.    // Construction de la fenêtre
32.    Gtk::Window fenetre;
```

```
33.   fenetre.set_title("ma fenetre");
34.   Gtk::HBox bH;
35.   fenetre.add(bH);
36.   Gtk::Button buttonA("A");
37.   Gtk::Button buttonB("B");
38.   Gtk::Button buttonC("C");
39.   SomeButton buttonD("D");
40.   SomeButton buttonE("E");
41.   SomeButton buttonF("F");
42.   bH.pack_start(buttonA);
43.   bH.pack_start(buttonB);
44.   bH.pack_start(buttonC);
45.   bH.pack_start(buttonD);
46.   bH.pack_start(buttonE);
47.   bH.pack_start(buttonF);
48.
49.   //Affichage des widgets
50.   fenetre.show_all();
51.
52.   // Connexions
53.   buttonA.signal_clicked().connect( sigc::ptr_fun(&on_button_clicked) );
54.   buttonB.signal_clicked().connect( sigc::mem_fun(some_object, &some_class::on_button_clicked) );
55.   buttonC.signal_clicked().connect( sigc::ptr_fun(&some_class::on_button_clicked) );
56.   buttonE.signal_clicked().connect( sigc::mem_fun(buttonE, &SomeButton::on_button_clicked) );
57.   buttonF.signal_clicked().connect( sigc::mem_fun(buttonF, &SomeButton::on_button_clicked) );
58.
59.   //Lancement de la fenêtre
60.   Gtk::Main::run(fenetre);
61.   return 0;
62. }
```

Exercice 4 - Binding des signaux

Le code *incomplet* de cet exercice est donné en fin d'énoncé.

Q.1) Compréhension du code

q.1.1) Représenter graphiquement l'interface obtenue

q.1.2) Expliquer ce qui va se passer lorsque l'utilisateur cliquera sur le `Drawing Area`

Q.2) Quel sera le profil de la fonction `on_button_pressed` *pour permettre une connexion au signal* `signal_button_press_event()` *de* `widg` *? (On pourra consulter le cours pour connaître la forme du signal correspondant)*

Q.3) Compléter la connexion du signal `signal_button_release_event()` *de* `widg` *à la fonction* `on_button_released()`.

Dans l'état actuel, le bouton « A » ne sert à rien, on va utiliser se bouton pour faire afficher un texte qui dépend des arguments passés au programme (par exemple « `le premier paramètre est : coucou` », si on a lancé le programme ainsi :

```
$ prog coucou
```

Le programme vérifie déjà qu'il y a bien un paramètre passé (ligne 12). Je rappelle que le premier paramètre passé à la commande est récupéré dans `argv[1]`.

> *Q.4) Ajouter **une fonction** et **une connexion** qui permette d'afficher le premier argument du programme lorsqu'on clique sur le bouton « A ».*

```
1.  ???? on_button_pressed( ???? ) {
2.      std::cout << "fonction pressed \n";
3.  }
4.
5.  bool on_button_released(GdkEventButton*, Glib::ustring str, int t) {
6.      std::cout << "fonction pressed (" << str << ", "<< t << ")\n";
7.  }
8.
9.  int main(int argc, char** argv) {
10.   Gtk::Main kit(argc, argv);
11.
12.   if(argc < 2) return 1;
13.
14.   // Construction de la fenêtre
15.   Gtk::Window fenetre;
16.   fenetre.set_title("ma fenetre");
17.   Gtk::HBox bH;
18.   fenetre.add(bH);
19.   Gtk::Button buttonA("A");
20.   bH.pack_start(buttonA);
21.
22.   Gtk::DrawingArea widg;
23.   bH.pack_start(widg);
24.   widg.add_events(Gdk::BUTTON_PRESS_MASK | Gdk::BUTTON_RELEASE_MASK);
25.
26.   //Affichage des widgets
27.   fenetre.show_all();
28.
29.   //Connexions
30.   widg.signal_button_press_event().connect( sigc::ptr_fun(&on_button_pressed) );
31.   widg.signal_button_release_event().connect(  //// BRANCHER on_button_released //// );
32.
33.   //Lancement de la fenêtre
34.   Gtk::Main::run(fenetre);
35.   return 0;
36. }
```