COURS et TP DE LANGAGE C++

Chapitre 16

TP de prise en main de C++ Builder

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

INITIATION A L'OUTIL C++ BUILDER

Introduction

Cet outil logiciel BORLAND, basé sur le concept de programmation orientée objet, permet à un développeur, même non expérimenté, de créer assez facilement une interface homme/machine d'aspect « WINDOWS ».
Le programme n'est pas exécuté de façon séquentielle comme dans un environnement classique. Il s'agit de programmation « événementielle », des séquences de programme sont exécutées, suite à des actions de l'utilisateur (clique, touche enfoncée etc…), détectées par WINDOWS.

Ce TP, conçu dans un esprit d'autoformation, permet de s'initier à l'outil, en maîtrisant, en particulier, quelques aspects généraux des problèmes informatiques, à savoir :
- La saisie de données,
- Le traitement,
- La restitution.

Les thèmes proposés fonctionnent avec toutes les versions de « C++ Builder ».

Bibliographie

- L'aide du logiciel,
- BORLAND C++ BUILDER, Gérad LEBLANC, EYROLLES.

1- Création du projet

C++ Builder fonctionne par gestion de projet. Un projet contient l'ensemble des ressources à la construction du fichier exécutable final.
Par défaut, les fichiers utilisateurs sont rangés dans la répertoire
c:\Program Files\Borland\Cbuilder3\Projects (ou Cbuilder5)

Créer un sous-répertoire Travail dans le répertoire Projects de Builder.
Lancer C++ Builder.
Apparaissent
- La fenêtre graphique (***Form1***), dans laquelle seront posés les objets graphiques (menus, boutons etc …),
- Un fichier source ***Unit.cpp***, contenant l'application utilisateur,
- Un fichier source ***Project1.cpp*** qui sera utilisé par WINDOWS pour lancer l'application utilisateur (pour y accéder Voir → Gestionnaire du projet → Double Clique sur ***Project1***),
- un fichier d'en-tête ***Unit1.h*** contenant la déclaration des composants utilisés sous forme de variables globales, ainsi que les fonctions créées. (pour le lire **Fichier → Ouvrir → Unit.h**)
- Dans le bandeau supérieur, un choix de composants graphiques,
- Sur la gauche, «l'inspecteur d'objets », permettant de modifier les propriétés de ces composants graphiques, et de leur associer des événements détectés par WINDOWS.

Modifier le nom du projet : Fichier → Enregistrer le projet sous
- ***Unit.cpp*** devient ***Projects\Travail\Mon_Appli.cpp***
- ***Project1.bpr*** devient ***Projects\Travail\Mon_Proj.bpr***

alors,
- ***Unit.h*** devient automatiquement ***Projects\ Travail\Mon_Appli.h***
- ***Project1.cpp*** devient automatiquement ***Projects\ Travail\Mon_Proj.cpp***

Cliquer dans la fenêtre ***Form1***. Repérer dans l'inspecteur d'objets les 2 propriétés « Name » (nom de la fenêtre dans le programme), et « Caption » (Titre de la fenêtre).
D'une façon générale, la plupart des composants Builder possèdent 2 identifiants :
- « Caption » : nom apparaissant sur l'écran à l'exécution,
- « Name » : nom du composant dans le programme, et donc variable à manipuler.

Modifier le titre de la fenêtre : « Travaux Pratiques »

Exécuter le programme. Le fermer avec la croix en haut à droite de la fenêtre.

Sortir de C++ Builder. Vérifier la présence du fichier exécutable Mon_Proj.exe, dans le répertoire Travail. Le lancer directement et vérifier son fonctionnement.

Revenir à C++ Builder et recharger le projet Mon_Proj.

2- Création d'un menu

On souhaite créer un menu **Fichier**
Saisie
Affichage
Quitter

Dans la boîte à outils « Standard », sélectionner le composant « Main Menu » et le placer dans Form1.
Cliquer sur ce menu, et remplir les différentes cases « Caption ».

Double Cliquer sur la rubrique « Quitter ». On accède alors à la fonction qui sera exécutée lors d'un clique sur cette rubrique. Compléter cette fonction comme ci-dessous.

```
 void __fastcall TForm1::Quitter1Click(TObject *Sender)
{
   Form1-> Close();
}
```

Exécuter le programme et vérifier.
Quitter C++ Builder et vérifier la présence dans le répertoire Travail du fichier ***Mon_Appli.h.*** Le lister et conclure.
Revenir à C++ Builder.

Cliquer dans Form1 et modifier sa couleur, via l'inspecteur d'objets, (choisir le blanc, clWhite). Vérifier en exécutant.

On souhaite maintenant interdire, pour l'instant, l'accès à la rubrique « Affichage » du menu.

Cliquer sur cette rubrique, modifier la propriété « Enabled » en « false » et vérifier en exécutant le programme.

Ajouter maintenant au menu une rubrique « Couleur » :

Fichier	**Couleur**
Saisie	Blanc
Affichage	Gris
Quitter	

Puis en double cliquant sur la rubrique « Blanc », compléter la fonction comme ci-dessous :

```
void __fastcall TForm1::Blanc1Click(TObject *Sender)
{
   Form1->Color = clWhite;
}
```

Enfin en double cliquant sur la rubrique « Gris », compléter la fonction comme ci-dessous :

```
void __fastcall TForm1::Gris1Click(TObject *Sender)
{
   Form1->Color = clGray;
}
```

Vérifier en exécutant.

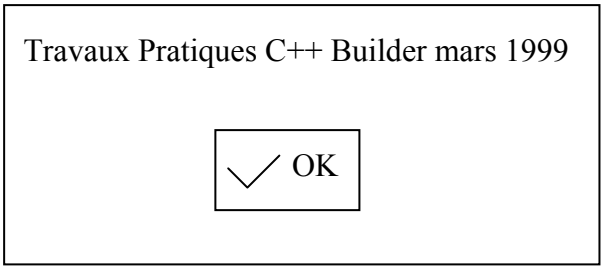
3- Création d'une boîte de dialogue

Ajouter maintenant au menu une rubrique « Outils » :

Fichier	**Couleur**	**Outils**
Saisie	Blanc	A propos de …
Affichage	Gris	
Quitter		

Poser sur la fenêtre ***Form1*** une « Label » (dans les composants « Standard »), avec les propriétés suivantes :
Alignment : taCenter
AutoSize : false
Caption : TP Builder mars 1999
Color : clAqua
Font : Times New Roman / Normal / Taille 12
Height : 80
Left : 150
Name : Label1
Top : 80

Transparent : false
Visible : true
Width : 145
Garder les valeurs par défaut des autres propriétés.
Ajouter au milieu de la Label1 un « BitBtn» (dans les composants « Supplement »), avec les propriétés suivantes :
Name : BitBtn1
Kind : bkOK
Visible : true



Exécuter pour vérifier.

Donner maintenant à Label1 et à Bitbtn1 la propriété Visible = false.

Exécuter pour vérifier.

Cliquer sur la ligne A propos de … du menu et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Aproposde1Click(TObject *Sender)
{
   Label1->Visible = true;
   BitBtn1->Visible = true;
}
```

Exécuter pour vérifier.

Cliquer sur le bouton OK et compléter la fonction comme ci-dessous :

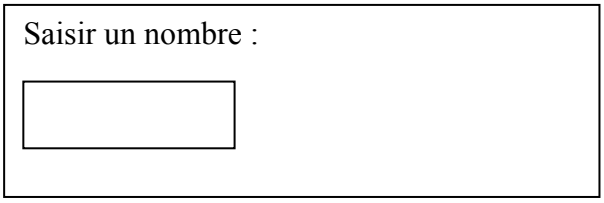
```
 void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   Label1->Visible = false;
   BitBtn1->Visible = false;
}
```

Exécuter pour vérifier.

4- Création d'une boîte de saisie

Ajouter sur Form1 une « Label » (dans les composants « Standard »), avec les propriétés suivantes :
Alignment : taLeftJustify
AutoSize : false
Caption : Saisir un nombre :
Color : clYellow
Font : Times New Roman / Normal / Taille 10
Height : 80
Left : 50
Name : Label2
Top : 150
Transparent : false
Visible : true
Width : 160

Ajouter au milieu de la Label2 un « Edit» (ab dans les composants « Standard »), avec les propriétés suivantes:
AutoSize : false
Name : Edit1
Color : clWindow
Text :
Visible : true



Exécuter pour vérifier.

Donner maintenant à Label2 et à Edit1 la propriété Visible = false.

Exécuter pour vérifier.

Cliquer sur la rubrique « Saisie » du menu et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true;
   Edit1->Visible = true;
}
```

Exécuter pour vérifier.

Dans la boîte Edit1, double cliquer sur l'événement **OnKeyDown** et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   float r;
   if(Key == VK_RETURN)
   {r = (Edit1->Text).ToDouble();}
}
```

Puis, ajouter au début du source Mon_Appli.cpp la déclaration
#include <vcl\dstring.h>

C++ Builder propose une bibliothèque de manipulations de chaînes de caractères. La classe *AnsiString* y a été créée, munie des méthodes appropriées.
Edit1->Text est de type *AnsiString*.
ToDouble est une méthode qui transforme une chaîne de type *AnsiString* en un nombre réel.

Attention, aucune précaution n'a été prise, ici, en cas d'erreur de saisie (ceci sera étudié à la fin du TP).
Si on effectue une erreur de saisie (par exemple en tapant une lettre), il y aura traitement d'une exception par WINDOWS. Concrètement, pour nous, ceci se traduit, pour l'instant, par un « plantage ».

L'événement **OnKeyDown** détecte l'appui sur une touche du clavier. On valide ici la saisie lorsque l'utilisateur appuie sur la touche « ENTREE ». Celle-ci n'est pas prise en compte dans la saisie.
Exécuter pour vérifier (avec et sans erreur de saisie).

5- Traitement et affichage d'un résultat

Modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true; Saisie1->Enabled = false;
   }
}
```

Exécuter pour vérifier.

Déclarer une variable globale comme ci-dessous :
float resultat ;

Puis modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true;
      Saisie1->Enabled = false; resultat = r*2 - 54;
   }
}
```

Construire le projet pour vérifier qu'il n'y a pas d'erreur (Menu Projet → Construire).

Aller chercher dans la palette standard le composant « ListBox » (boîte liste)
Le poser sur la fenêtre principale, vers la droite, avec les propriétés suivantes :
Visible = false
Color = clYellow
Height = 65
Left = 280
Top = 24
Width = 120
Name : ListBox1

Modifier la fonction **Saisie1Click** comme ci-dessous

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true;
   Edit1->Visible = true;
   ListBox1->Visible = false;
}
```

Cliquer sur la rubrique « Affichage » du menu. Ecrire la fonction de gestion de l'événement OnClick comme ci-dessous :

```
 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   chaine= FloatToStrF(resultat,AnsiString::sffGeneral,7,10);
   ListBox1->Clear(); ListBox1->Items->Add(chaine);
   Affichage1->Enabled = false; Saisie1->Enabled = true;
}
```

La fonction **FloatToStrF** permet de convertir un nombre réel en une chaîne de caractères. Voir l'aide pour plus d'informations.
Exécuter pour vérifier.

6- Affichage d'une série de valeurs

Déclarer une variable globale **float tab[3]**, supprimer la variable **resultat**.
Modifier les fonctions **Affichage1Click** et **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(int i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
}

 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      tab[0] = r*2 - 54; tab[1] = r*3 + 100; tab[2] = r*2.5 - 8.5;
      Saisie1->Enabled = false; Affichage1->Enabled = true;
   }
}
```

Exécuter pour vérifier.

7- Fichiers

Ajouter au menu dans la rubrique « Fichier », une sous-rubrique « Ouvrir » et une sous-rubrique « Enregistrer »
La sous-rubrique « Ouvrir », possède la propriété Enabled = true.
La sous-rubrique « Enregistrer », possède la propriété Enabled = false.

Modifier les fonctions **Affichage1Click** et **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true; Enregistrer->Enabled = true;
      Saisie1->Enabled = false; Ouvrir1->Enabled = false;
      tab[0] = r*2 - 54; tab[1] = r*3 + 100;
      tab[2] = r*2.5 - 8.5;
   }
}

 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   int i;
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
   Enregistrer1->Enabled = true;
}
```

Exécuter pour vérifier.

Aller chercher dans la palette Dialogues le composant « SaveDialog » (boîte de sauvegarde)
Le poser sur la fenêtre principale, avec les propriétés suivantes :

FileName : tp1.dat (nom apparaissant par défaut)
InitialDir : c:\Program Files\Borland\Cbuilder3\Projects\Sei
Filter : *.dat
Name : SaveDialog1.

Cliquer sur la rubrique « Enregistrer » du menu et rédiger la fonction **Enregistrer1Click** comme ci-dessous.

```
 void __fastcall TForm1::Enregistrer1Click(TObject *Sender)
{
   bool res = SaveDialog1->Execute();
}
```

Exécuter pour vérifier.
C'est au programme de sauvegarder réellement le fichier. La variable « res » vaut ***false*** si l'utilisateur a annulé l'opération, ***true*** s'il a réellement sélectionné un fichier.

Ajouter le fichier ***stdio.h*** dans la lister des ***#include***
Modifier fonction **Enregistrer1Click** comme ci-dessous :

```
 void __fastcall TForm1::EnregistrerClick(TObject *Sender)
{
   FILE *f;
   bool res = SaveDialog1->Execute();
   if(res==true)
   {
      f = fopen((SaveDialog1->FileName).c_str(), "wb");
      fwrite(tab,4,3,f); fclose(f) ;
   }
}
```

On utilise ici les fonctions de traitement de fichier classiques de BORLAND C++. La fonction membre ***c_str()*** de la classe ***AnsiString*** permet de convertir une « AnsiString » en tableau de caractères classique, nécessaire ici à la fonction ***fopen***.

Exécuter pour vérifier, en choisissant le nom de fichier proposé par défaut, puis un autre.
Vérifier la présence de ces fichiers dans l'explorateur de WINDOWS.

Aller chercher dans la palette Dialogues le composant « OpenDialog »
Le poser sur la fenêtre principale, avec les propriétés suivantes :

FileName : tp1.dat (nom apparaissant par défaut)
InitialDir : c:\Program Files\Borland\Cbuilder3\Projects\Travail
Filter : *.dat
Name : OpenDialog1.

Cliquer sur la rubrique « Ouvrir » du menu et rédiger la fonction **Ouvrir1Click** comme ci-dessous.

```
 void __fastcall TForm1::Ouvrir1Click(TObject *Sender)
{
   FILE *f;
   bool res = OpenDialog1->Execute();
   if(res==true)
   {
      f = fopen((OpenDialog1->FileName).c_str(), "rb");
      fread(tab,4,3,f); fclose(f); Affichage1->Enabled = true;
   }
}
```

C'est au programme d'ouvrir réellement le fichier. La variable « res » vaut ***false*** si l'utilisateur a annulé l'opération, ***true*** s'il a réellement sélectionné un fichier.

Exécuter pour vérifier.

8- Amélioration de la zone de saisie

Un composant a le « focus d'entrée », lorsqu'il est actif dès la prochaine action sur le clavier.

Modifier la fonction comme **Saisie1Click** ci-dessous :

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true; Edit1->Visible = true;
   ListBox1->Visible = false;
   Edit1->SetFocus(); // donne le focus d'entrée à la zone d'édition
}
```

Exécuter pour vérifier.

Modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      try
      {
         float r = (Edit1->Text).ToDouble();
         Affichage1->Enabled = true; Enregistrer->Enabled = true;
         Saisie1->Enabled = false; Ouvrir1->Enabled = false;
         tab[0] = r*2 - 54 tab[1] = r*3 + 100; tab[2] = r*2.5 - 8.5;
      }
          catch(...)
         {ShowMessage("Ce n'est pas un réel");}
   }
}
```

On traite ici l'exception générée par WINDOWS lors d'une erreur de saisie. Si tout se passe bien, le bloc « try » est exécuté, sinon, le bloc « catch » est exécuté.
On découvre ici, au passage la fonction **ShowMessage** qui permet d'afficher très facilement un message soumis à validation sur l'écran.

Exécuter sous Builder pour vérifier, en effectuant une erreur de saisie. WINDOWS génère une erreur. Cliquer sur OK et continuer l'exécution.
Lancer maintenant directement l'exécutable Mon_Proj.exe depuis l'explorateur de WINDOWS, en effectuant une erreur de saisie.
Ca marche !

9- Utilisation d'une ressource système

On souhaite, ici, mettre en œuvre la ressource TIMER.

Sélectionner le composant TIMER dans la palette « système ».
Lui attribuer les propriétés suivantes :

Enabled : false
Interval : 2000 (ce sont des millisecondes)
Name : Timer1

Double Cliquer sur l'événement **OnTimer** et compléter la fonction comme ci-dessous (analogue à Affichage1Click)

```
 void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
   Timer1->Enabled = false;  // inhibition du timer
   AnsiString chaine;
   ListBox1->Visible = true;
   Edit1->Visible = false; Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(int i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
   Enregistrer->Enabled = true;
}
```

Compléter la fonction **Edit1KeyDown** comme ci-dessous :
On lance dons le timer chaque fois que l'on saisit une valeur dans la boîte de saisie. On obtient dons un affichage 2 secondes après la saisie.

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      try
      {
         float r = (Edit1->Text).ToDouble();
         Affichage1->Enabled = true; Enregistrer->Enabled = true;
         Saisie1->Enabled = false; Ouvrir1->Enabled = false;
         tab[0] = r*2 - 54;
         tab[1] = r*3 + 100;
         tab[2] = r*2.5 - 8.5;
         Timer1->Enabled = true ;
      }
          catch(...)
         {
            ShowMessage("Ce n'est pas un réel");
         }
   }
}
```

Exécuter pour vérifier.

On pourra utiliser le timer, par exemple pour lancer une acquisition sur une carte d'interface, pour provoquer des événements périodiques s'il est toujours validé etc..
Il y a toutefois une limite : la milliseconde !

Remarque : On se prive ici de la rubrique « Affichage » du menu. Il ne faut donc pas faire n'importe quoi !

10- Conclusion

Ce n'est plus qu'une question de temps passé et de bonne documentation …

Si on connaît un peu de programmation orienté objet, quoique la notion de pointeur de structure suffise ici, on peut, à ce stade, analyser le fichier d'en-tête ***Mon_Appli.h,*** et comprendre un peu mieux la construction des composants manipulés dans ce TP.