

Date: Samedi 27 janvier 2007 à 17:22:06
Sujet: Programmation Delphi

Cours Delphi 2 : Initiation à la programmation avec Delphi

Delphi est un outil de programmation puissant et simple à utiliser pour créer des applications sous Windows. Nous allons découvrir ensemble l'environnement de Delphi2005, l'organisation des projets, les différents fichiers de Delphi, les fonctions, les procédures ainsi que les bases pour programmer vos premières applications. Le cours sera suivi d'exercices corrigés qui vous permettront de vous exercer et de vous perfectionner.

Delphi est un environnement de développement intégré propriétaire (IDE) pour le langage Pascal sous Windows. Delphi implémente une version orientée objet du langage Pascal.

L'environnement de développement s'appuie sur un éditeur d'interface graphique associé à un éditeur de code source. Il doit son succès à sa facilité d'utilisation pour développer des applications graphiques et/ou liées aux bases de données. Un projet Delphi est constitué d'unités (units) correspondant à des fichiers sources et des fiches (forms) qui définissent les interfaces graphiques. Pour l'instant nous nous contenterons de coder des applications consoles et nous verrons plus tard les applications graphiques. Les fichiers de Delphi Liste des différents fichiers que vous pourrez créer, modifier ainsi que leur utilité respective.

Extension du fichier	Description
.DCR	Fichier projet
.DFM	Dessin de la fiche (form) contenant les propriétés des composants graphiques
.DPR	Fichier projet, c'est le coeur de votre application, il faudra lier tout vos différents fichiers contenant du code (.pas) au .DPR.
.PAS	Fichier source contenant le code de l'application
.BDSPROJ	Fichier projet associé au
.DPR	.RC Fichier de ressources Windows

Liste des fichiers générés après la compilation et qu'il n'est pas nécessaire de modifier vous-même.

Description	Extension du fichier	Fichier
d'un groupe de projet	.BPG	
Fichier créé lors de la compilation mais pas indispensable	.DCU	
	.DPK	Fichier

source d'un paquet .DOF
 Fichier d'option de compilation du projet
 .DSK Préférence de l'IDE
 .RES Fichier de ressources

Windows, il peut contenir des icônes, des bitmaps, des sons
 Je vais essayer de vous apprendre à coder proprement donc
 à partir de maintenant, il vous que vous souveniez qu'on code les
 fonctions et les procédures (vous comprendrez plus tard ce que c'est
 exactement) dans des fichiers .pas et qu'on les appelle dans le .dpr . Votre
 premier projet Pour créer un nouveau projet Delphi (.dpr) rendez
 dans

```
Fichier &gt; Nouveau &gt; Autres &gt; Projets Delphi &gt; Application
console Project1.dpr program
Project1; {$APPTYPE CONSOLE} uses
SysUtils; begin
{ TODO -oUser -cConsole Main : placez le code ici }
end.
```

à celui ci-dessus.

La première ligne définit le nom du programme. Vous
 pouvez changer le Projet1 en Premierprojet et enregistrer le fichier avec
 comme nom Premierprojet (.dpr ou .bdsproj). uses sert à inclure des
 fichiers de fonctions ou de procédures, souvent des .pas.

SysUtils contient les fonctions et procédures de base de
 Delphi, dont les procédures d'entrées et de sorties.
 Je vous conseille de l'inclure dans tous vos fichiers.

{ TODO -oUser -cConsole Main : placez le code ici } peut être
 supprimé. begin et end. définissent respectivement le
 début et la fin de votre code à proprement parler. Types de
 données Les variables et leur type doivent être
 déclarées avant d'être utilisées. Le type d'une
 variable doit être choisi en fonction de ce que contiendra cette variable.
 integer : ce sont les nombres entiers (exemple : 42 ou -5) single ou double :
 sont utilisés comme type pour les nombres à virgules
 (exemple : 3.3) On peut appliquer des opérations aux variables
 numériques :

- + : addition
- : soustraction
- * : multiplication
- / : division

mod : modulo (ne peut être appliqué qu'aux entiers (integer))
 char : est utilisé pour contenir un seul caractère
 (exemple : m) string : est utilisé pour contenir des chaînes de
 caractères, des phrases (exemple : Hello world) Pour
 concaténer (assembler) 2 chaînes de caractères on
 utilise l'opérateur + .

Exemple : 'Hello ' + 'world' équivaut à 'Hello
 world' boolean : ce sont des booléens. Il y a 2 valeurs
 booléennes True et False, respectivement Vrai et Faux. Il existe
 beaucoup d'autres types mais je vous ai présenté les plus
 utilisés. Variables, Constantes et Paramètres Un

paramètre représente les informations que l'on peut passer à une fonction ou à une procédure.

Une variable est une donnée déclarée à l'intérieur d'une fonction, et qui n'est accessible que dans celle-ci. C'est une donnée locale, vous pouvez la modifier comme bon vous semble. Il existe également des variables globales qui sont accessibles dans tout le DPR et le projet mais il ne faut les utiliser qu'exceptionnellement. Quand on utilise une variable, elle doit être déclarée dans tous les fichiers où elle est utilisée. Seules les variables passées en paramètre d'une fonction/procédure n'ont pas besoin d'être déclarées dans la fonction/procédure ou elles sont utilisées. Pour déclarer une variable, on utilise le mot clé var. On peut déclarer les variables dans le .DPR ou dans les fonctions/procédures de vos fichiers .pas .

Déclarer une variable dans un .dpr, juste après les uses.

```
Premierprojet.dpr
program Premierprojet;      {$APPTYPE CONSOLE}      uses
    SysUtils;      var
        nom_variable1 : type_variable1;
        nom_variable2 : type_variable2;      begin
        { TODO -oUser -cConsole Main : placez le code ici }
        end.
```

La déclaration des variables dans les fichiers .pas peut se faire de 2 façons (vous allez bientôt vous en servir). La seule chose qui change par rapport à la déclaration dans le .dpr, c'est l'emplacement où l'on déclare les variables, le code reste le même.

Vous pouvez placer un code similaire à celui ci-dessous après l'interface dans votre .pas (ne le faites pas encore).

```
var
nom_variable1 : type_variable1;
nom_variable2 : type_variable2;
```

Mais le mieux est de déclarer les variables en local dans les fonctions/procédures, c'est ce que nous ferons dans le reste du cours.

Souvenez-vous qu'elles doivent être déclarées entre la ligne de déclaration de la fonction et le begin.

```
function nom_fonction(nom_variable1 : type_variable1) : type_resultat;

var
nom_variable2 : type_variable2;
nom_variable3 : type_variable3;

begin
// instructions de votre fonction
```



```

fichiersource1.pas                                unit fichiersource1;
interface
    function nom_fonction(nom_variable : type_variable;
nom_variable2 : type_variable2) : type_resultat;      implementation
    function nom_fonction(nom_variable : type_variable;
nom_variable2 : type_variable2) : type_resultat;

    begin
        result := nom_variable * nom_variable2 ;
    end;      end.

```

Essayez de modifier au fur et à mesure la structure du code. Cette ligne permet de déclarer la fonction.

nom_fonction c'est le nom de la fonction, choisissez des noms de fonctions clairs et différents du nom du fichier qui contient ces fonctions.

nom_variable : type_variable choisissez également des noms de variables clairs. Le type de la variable dépend de ce qu'elle contient, on l'a déjà vu plus haut dans le cours (ex : integer, string etc.).

Les variables qui se trouvent entre les parenthèses qui suivent le nom de la fonction sont les variables passées en paramètres. On peut mettre autant de paramètres que l'on veut, il faut juste respecter la syntaxe et séparer par un point virgule (;) les différents couples nom_variable : type_variable.

: type_resultat ; définit le type du résultat que renverra la fonction result := nom_variable * nom_variable2 ; result est spécifique aux fonctions, c'est le résultat que renvoie la fonction. Dans notre cas la fonction renverra le résultat de la multiplication des 2 variables que nous avons passées en paramètres. On va donc donner aux variables des noms plus clairs et leur associer des types adaptés.

Votre fichier doit maintenant ressembler à cela :

```

fichiersource1.pas                                unit fichiersource1;
interface
    function multiplication(var1 : integer ; var2 : integer) : integer;
implementation
    function multiplication(var1 : integer ; var2 : integer) : integer;
    begin
result := var1 * var2 ;
    end;      end.

```

Nous verrons dans la suite du cours comment appeler une fonction et afficher le résultat.

Nous nous servirons également de cette fonction dans la suite du cours. Une procédure peut également prendre des paramètres mais elle ne renvoie aucun résultat. Cela signifie pas qu'on ne peut rien afficher avec, au contraire. Nous allons

```

créer une procédure dans le même fichier.
fichiersource1.pas          unit fichiersource1;
interface
    fonction multiplication(var1 : integer ; var2 : integer) : integer;

    procedure hello(nom1 : string);          implementation
    fonction multiplication(var1 : integer ; var2 : integer) : integer;
    begin
    result := var1 * var2 ;
    end;
    procedure hello(nom1 : string);
    begin
    writeln('Bonjour ' + nom1) ;
    end;
    end.

```

Vous avez vite dû comprendre que la procédure hello affichait Bonjour suivi de la variable rentrée en paramètre. Comme je l'ai déjà dit, je vous conseille de coder vos fonctions et vos procédures dans un fichier .pas que vous inclurez dans les uses du .dpr. Votre premier programme On va maintenant mettre en pratique toutes ces informations et utiliser les fichiers que nous avons créés jusqu'à présent. Mais avant cela nous allons voir quelques règles de base de la syntaxe Delphi.

- chaque instruction doit être suivie d'un ;
- l'opérateur d'affectation est :=
- les commentaires n'ont aucune influence sur le code de votre application, vous pouvez en mettre pour vous aider quand vous relirez votre code. Pour insérer des commentaires dans le code source, on peut utiliser trois techniques différentes :

On peut écrire un commentaire sur plusieurs lignes en plaçant le texte entre bornant le texte entre accolades { } ou entre une paire astérisques/parenthèses (* *). Ou on peut écrire un commentaire d'une ligne en la commençant par deux barres obliques (slash) : //.

Nous avons créé une fonction et une procédure dans fichiersource1.pas, nous allons maintenant les utiliser dans notre fichier projet (.dpr ou .bdsproj).

```

Premierprojet.dpr (ou
.bdsproj)
program Premierprojet;
{$APPTYPE CONSOLE}
uses
    SysUtils,
    fichiersource1 in 'fichiersource1.pas';
begin
    writeln(multiplication(3,5)) ;
    readln ;
end.

```

Vous vous souvenez que l'on a créé la fonction multiplication ?

multiplication(3,5) : va donc appeler la fonction multiplication créée dans fichiersource1.pas avec en paramètres 3 et 5 (les paramètres doivent être rangés dans le bon ordre mais si ici cela n'a pas d'influence sur le résultat).

On utilise la procédure de sortie writeln pour afficher le résultat de multiplication(3,5) dans la console.

multiplication(3,5) va renvoyer le résultat de la fonction multiplication

mais on doit l'afficher c'est pour cela qu'on utilise writeln.

readln est normalement utilisée pour permettre à l'utilisateur d'entrer une variable mais on l'utilise ici (sans les parenthèses et le nom d'une variable) pour empêcher l'application de se fermer jusqu'a ce que l'utilisateur appuie sur une touche, c'est une sorte de pause.

Exécuté ce code en cliquant sur la flèche verte dans la barre d'outil ou en appuyant sur F9, une console s'ouvrira et vous verrez affiché le résultat de la multiplication de 3 et 5, c'est à dire 15.

Essayez d'enlever le readln, vous constaterez que la console se ferme automatiquement. Nous avons vu comment afficher le résultat d'une fonction grâce a la fonction write (ou writeln), nous allons maintenant voir que la fonction write n'est pas nécessaire dans le cas d'une procédure.

Contrairement à la fonction, la procédure renvoie tout ce qu'elle a fait, c'est comme si l'on incluait la procédure dans le dpr.

Le writeln qui est dans la procedure hello va donc être affiché. Ajoutons un appel à la procédure hello avec en paramètres une chaîne de caractère (= string) entre des quotes (ex : ' votre chaîne de caractère').

Premierprojet.dpr (ou .bdsproj) program
Premierprojet;

```
    {$APPTYPE CONSOLE}          uses
    SysUtils,
    fichiersource1 in 'fichiersource1.pas';          begin
    &nbsp;&nbsp;&nbsp;writeln(multiplication(3,5)) ;
    &nbsp;&nbsp;&nbsp;readln ;
    &nbsp;&nbsp;&nbsp;hello('zmaster') ;
    &nbsp;&nbsp;&nbsp;readln ;
    end.
```

Entraînez-vous à utiliser les procédures write et read ainsi qu'a vous servir des fonctions déclarées dans un autre fichier.

On va améliorer un peu notre programme pour le rendre plus dynamique et fonctionnel.

On va faire en sorte que notre programme nous demande notre prénom et qu'il nous dise Bonjour grâce à la procedure hello et qu'il nous demande ensuite les 2 chiffres qu'on veut multiplier avant d'afficher le résultat.

On va coder cette partie dans le dpr. Comme vous avez dû le comprendre, il va nous falloir 3 variables car on va entrer dans le programme notre prénom (type string) et les 2 chiffres à multiplier qui seront des integer (entiers). On commence par déclarer les variables, ce qui nous donne : Premierprojet.dpr (ou .bdsproj)
program Premierprojet;

```

        {$APPTYPE CONSOLE}          uses
        SysUtils,
        fichiersource1 in 'fichiersource1.pas';      var
        mult1, mult2 : integer ;
        prenom : string ;          begin
        &nbsp;&nbsp;&nbsp;writeln(multiplication(3,5)) ;
        &nbsp;&nbsp;&nbsp;readln ;
        &nbsp;&nbsp;&nbsp;hello('zmaster') ;
        &nbsp;&nbsp;&nbsp;readln ;
        end.

```

Quand on déclare finit plusieurs variables du même type, on peut les déclarer sur la même ligne en les séparant par des virgules.

On va maintenant demander rentrer les instructions qui nous permettront d'entrer ces variables. Premierprojet.dpr (ou .bdsproj)

```

program Premierprojet;

```

```

        {$APPTYPE CONSOLE}          uses
        SysUtils,
        fichiersource1 in 'fichiersource1.pas';      var
        mult1, mult2 : integer ;
        prenom : string ;          begin
        &nbsp;&nbsp;&nbsp;readln(prenom) ; //demande le prenom
        &nbsp;&nbsp;&nbsp;hello(prenom) ; (*ecris Bonjour suivit du prenom

```

*)

```

        &nbsp;&nbsp;&nbsp;readln(mult1) ; //demande le 1er chiffre
        &nbsp;&nbsp;&nbsp;readln(mult2); //demande le chiffre 2

```

```

        &nbsp;&nbsp;&nbsp;writeln(multiplication(mult1,mult2)) ;
        (*affiche la multiplication de mult1 et mult2 *)

```

```

        &nbsp;&nbsp;&nbsp;readln ; // fais une pause

```

end. Vous pouvez exécuter (F9) ce code pour voir.

Vous devrez entrer votre prénom puis appuyer sur Entrée > le programme renverra Bonjour suivit de votre prénom > vous devez rentrer un chiffre entier qui sera affecté à mult1 puis appuyer sur Entrée > rentrez alors un autre chiffre qui sera affecté à mult2 puis appuyez sur Entrée > le programme affichera le résultat de la multiplication des 2 chiffres que vous avez entrés > pour quitter appuyez sur n'importe quelle touche.

Vous pouvez améliorer un peu votre programme pour le rendre plus clair. Exemple :

```

Premierprojet.dpr (ou .bdsproj)
program Premierprojet;          {$APPTYPE
CONSOLE}          uses
        SysUtils,
        fichiersource1 in 'fichiersource1.pas';      var

```



```

    mult1, mult2 : integer ;
    prenom : string ;          begin

    write('Entrez votre prenom : ') ;
    readln(prenom) ;
    hello(prenom) ;

    write('Entrez un chiffre a multiplier : ') ;
    readln(mult1) ;
    write('Entrez un deuxieme chiffre a multiplier : ') ;
    readln(mult2) ;

    writeln(IntToStr(mult1) + ' x ' + IntToStr(mult2) + ' = ' +
IntToStr(multiplication(mult1,mult2))) ;
    readln ;

end.

```

Si vous avez bien suivis le cours, vous devriez comprendre toutes les lignes sauf :

```

writeln(inttostr(mult1) + ' x ' + inttostr(mult2) + ' = ' +
inttostr(multiplication(mult1,mult2))) ;

```

Je vais vous expliquer l'utilit  de chaque  ment de cette ligne.

D j, vous avez du comprendre en ex cutant le programme que c' tait la ligne qui afficher : mult1 x mult2 = r sultat

On ne doit pas m langer des variables de types diff rents dans un writeln or on essaie d'afficher des caract res de caract res (string) : ' x ', ' = ' et ainsi que des integer : mult1, mult2 et le r sultat de la multiplication.

Pour afficher diff rents types de donn es dans un write/writeln, il faut convertir ces donn es. IntToStr() sert a convertir des integer en string (d'ou son nom), on s'en sert ici pour pouvoir afficher les integer avec le reste.

Les + qui s pare chaque partie dans le writeln servent a concat ner (assembler) les diff rentes parties. C'est la fin de ce 2 me cours Delphi. C'est assez dur d'expliquer les bases de la programmation mais une fois que vous avez compris,   devient tr s facile. Dans le prochains cours, vous apprendrez les conditions, cela fait toujours partie des bases, c'est ce qui permet de dire a un programme : si condition alors faire instructions sinon faire autres instructions. Vous pouvez maintenant passer aux exercices du cours 2 : Exercices Delphi 2 : Entraînement Variables, Fonctions et Affichage (en cours de r alisation) Si vous avez besoin d'aide, il y a la partie programmation du forum : [Forum Programmation](#)

Les autres cours Delphi de www.zmaster.fr :

[Cours Delphi 1 : T cher et installer Delphi](#)

[2005 gratuitement](#)

[Cours Delphi 2 : Initiation à la programmation avec Delphi](#)

[Cours Delphi 3 : Les conditions \(if then else\) \(en cours de réalisation\)](#)

Publication de Tout sur l'informatique - Programmation C#,
Sécurité, Divx, P2P:

<http://www.zmaster.fr>

URL de cette publication

<http://www.zmaster.fr/modules.php?name=News&file=article&sid=202>