

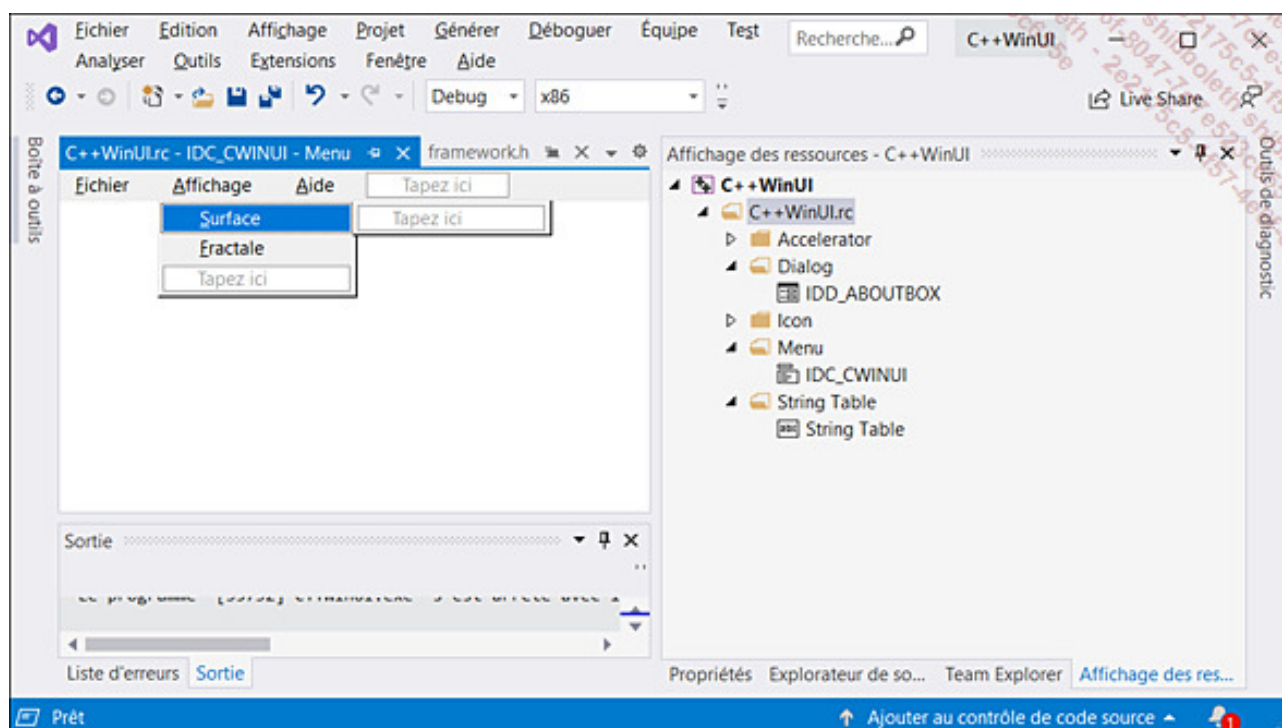
Travaux pratiques

1. Réaliser une application de dessin Win32

Nous reprenons le format d'application Win32 étudié en début de chapitre pour réaliser un programme capable d'afficher deux types de dessins générés par calcul C++.

Notre projet s'appelle "**C++ Win UI**".

La première étape consiste à ajouter de nouvelles entrées **Affichage - Surface** et **Affichage - Fractale** au menu de l'application. Pour cela, il faut ouvrir le fichier de ressource et éditer le menu `IDC_CWINUI` :



Le caractère souligné sert de raccourci-clavier (on dit aussi accélérateur). Il s'obtient en précédant la lettre **A** (puis respectivement S et F) par un signe **&**.

Pour désigner chaque type de dessin, on définit une énumération dans le fichier C++ `WinUI.cpp` :

```
enum Affichage { Surface, Fractale } affichage;
```

La fonction `wWinMain()` se substitue à la traditionnelle fonction `main()`. C'est un emplacement possible pour initialiser la variable `affichage` :

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine,
    _In_ int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Placez le code ici.
    affichage = Affichage::Surface;
```

La fonction `WndProc()` traite les messages de l'application. À chaque entrée dans le menu correspond une action de `WM_COMMAND` :

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
        {
            int wmlId = LOWORD(wParam);
            // Analyse les sélections de menu :
            switch (wmlId)
            {
                case ID_AFFICHAGE_SURFACE:
                    affichage = Affichage::Surface;
                    InvalidateRect(hWnd, nullptr, true);
                    break;
                case ID_AFFICHAGE_FRACTALE:
                    affichage = Affichage::Fractale;
                    InvalidateRect(hWnd, nullptr, true);
                    break;
```

```

case IDM_ABOUT:
    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
    break;
case IDM_EXIT:
    DestroyWindow(hWnd);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
}
break;

```

L'appel à `InvalidateRect()` force le rafraîchissement de la zone cliente (le contenu de la fenêtre principale). Cela provoque l'émission d'un message `WM_PAINT` lui-même intercepté et traité dans la fonction `WndProc`. Attention, c'est Windows qui rappelle la fonction en passant les paramètres indispensables pour le traitement du rafraîchissement graphique.

```

case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // TODO: Ajoutez ici le code de dessin qui utilise hdc...
    switch (affichage)
    {
        case Affichage::Surface:
            OnDrawG(hWnd, hdc, ps);
            break;
        case Affichage::Fractale:
            OnDrawF(hWnd, hdc, ps);
            break;
    }

    EndPaint(hWnd, &ps);
}
break;

```

```

case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

La fonction `OnDrawG()` utilise quelques primitives **GDI** (*Graphic Design Interface*, le module graphique de Windows) pour dessiner point par point une figure 3D représentée en perspective cavalière :

```

#pragma region OnDrawG
void OnDrawG(HWND hWnd, HDC hdc, PAINTSTRUCT ps)
{
    // MX et MY déterminent la qualité de l'image
    int MX, MY;
    MX = 350;
    MY = 350;

    // dimensionner l'écran
    RECT rc;
    GetClientRect(hWnd, &rc);

    SIZE lp;
    lp.cx = rc.right;
    lp.cy = -rc.bottom;
    SetViewportExtEx(hdc, lp.cx, lp.cy, &lp);
    POINT p;
    p.x = rc.right / 2;
    p.y = rc.bottom / 2;
    SetViewportOrgEx(hdc, p.x, p.y, &p);

    lp.cx = 2 * MX;
    lp.cy = 2 * MY;
    SetWindowExtEx(hdc, lp.cx, lp.cy, &lp);

    RECT zone_invalide;    // zone à rafraîchir
}

```

```

POINT test;          // un point
GetClipBox(hdc, &zzone_invalide);

// -----
// vue en 3D
double a1, a2, a3, b1, b2, b3;    // coefficients de
projection 3D
double l, L;                  // latitude et longitude

// point de vue
l = 1.1;
L = 1.20;

// ces coefficients sont calculés pour une projection orthogonale
a1 = -sin(L);
a2 = cos(L);
a3 = 0;
b1 = -sin(l) * cos(L);
b2 = -sin(l) * cos(L);
b3 = cos(l);

// -----
int i, j;    // coordonnées dans l'espace logique
int X, Y;    // coordonnées projetées (X,Y)=proj(x,y,z)

int coul;    // couleur des points
double cote;
double x, y;

for (i = -MX; i < MX; i++)
    for (j = -MY; j < MY; j++)
    {
        x = (double)i / MX * 2;
        y = (double)j / MY * 2;

        cote = 60 * y * sin(x);
        if (i > 0)
        {
            if (j > 0)

```

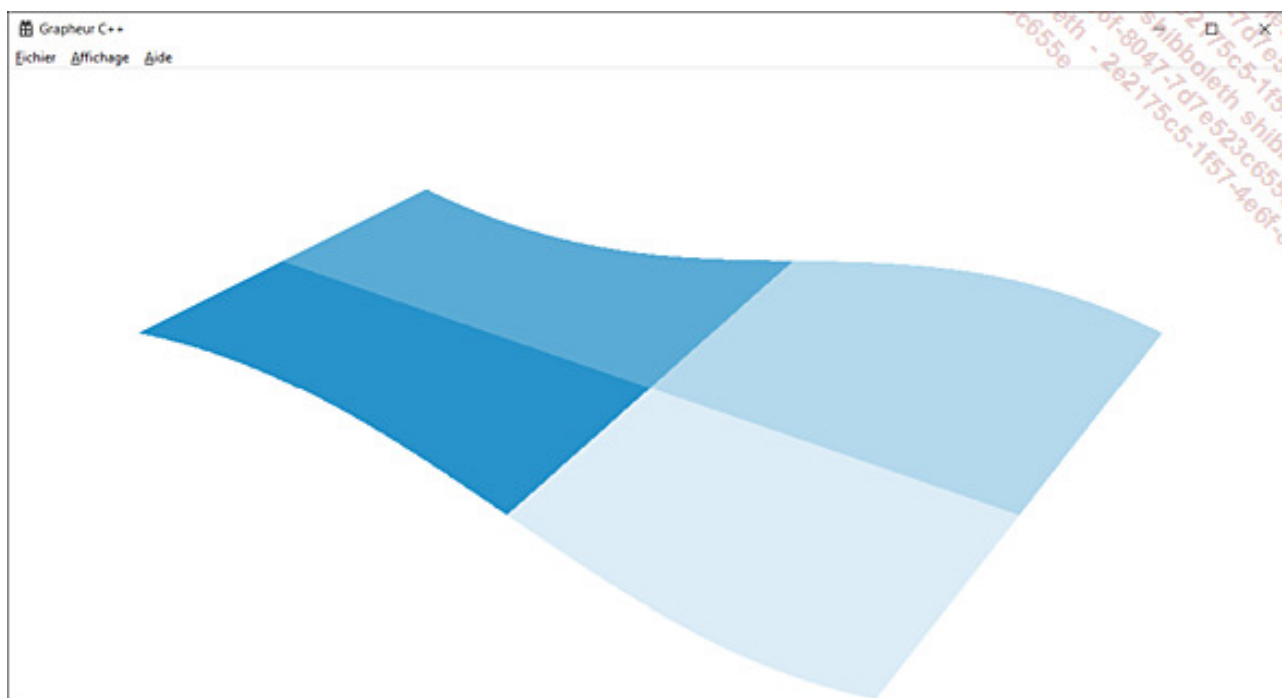
```

        coul = 90;
    else
        coul = 40;
    }
    else
    {
        if (j > 0)
            coul = 180;
        else
            coul = 220;
    }

    // projection sur un plan
    test.x = a1 * i + a2 * j + a3 * cote;
    test.y = b1 * i + b2 * j + b3 * cote;

    if (PtInRect(&zone_invalide, test)) //
le point est-il invalide ?
    {
        SetPixel(hdc, test.x, test.y,
        RGB(coul,127+coul/2,192+coul/4));
    }
}
#pragma endregion

```



Le code fonction `OnDrawF()` est similaire pour représenter en 3D une version stylisée de l'ensemble fractal de Mandelbrot :

```
#pragma region OnDrawF
void OnDrawF(HWND hWnd, HDC hdc, PAINTSTRUCT ps)
{
    // MX et MY déterminent la qualité de l'image
    int MX, MY;
    MX = 1500;
    MY = 1500;

    // dimensionner l'écran
    RECT rc;
    GetClientRect(hWnd, &rc);

    SIZE lp;
    lp.cx = rc.right;
    lp.cy = -rc.bottom;
    SetViewportExtEx(hdc, lp.cx, lp.cy, &lp);
    POINT p;
    p.x = rc.right / 2;
    p.y = rc.bottom / 2;
```

```

SetViewportOrgEx(hdc, p.x, p.y, &p);

lp.cx = 2 * MX;
lp.cy = 2 * MY;
SetWindowExtEx(hdc, lp.cx, lp.cy, &lp);

RECT zone_invalide;    // zone à rafraîchir
POINT test;           // un point
GetClipBox(hdc, &zone_invalide);

// -----
// vue en 3D
double a1, a2, a3, b1, b2, b3;    // coefficients de
projection 3D
double l, L;                      // latitude et longitude

// point de vue
l = 1.1;
L = 1.20;

// ces coefficients sont calculés pour une projection orthogonale
a1 = -sin(L);
a2 = cos(L);
a3 = 0;
b1 = -sin(l) * cos(L);
b2 = -sin(l) * cos(L);
b3 = cos(l);

// -----
// tracé de l'ensemble de Mandelbrot en 3D
Complexe z1, z2, c, t;
int i, j;    // coordonnées dans l'espace logique
int n;       // nombre d'itération
int X, Y;    // coordonnées projetées (X,Y)=proj(x,y,z)

double seuil = 3; // détection de divergence
int coul;        // cote et couleur des points
double cote;

int max = 20;

```



```

for (i = -MX; i < MX; i++)
  for (j = -MY; j < MY; j++)
  {
    c.re = (2.0 * i) / MX;
    c.im = (2.0 * j) / MY;

    z1.re = 0;
    z1.im = 0;
    for (n = 0; n < max; n++)
    {
      // z2=z1*z1+c
      z2.re = z1.re * z1.re - z1.im * z1.im + c.re;
      z2.im = 2 * z1.re * z1.im + c.im;

      t.re = z2.re - z1.re;
      t.im = z2.im - z1.im;

      // le point c est-il instable ?
      if (t.re * t.re + t.im * t.im > seuil)
      {
        double z = seuil / (t.re * t.re + t.im * t.im);
        cote = n * 1.5 * (-4) * (z);
        if (cote > 2)
          cote = 2;
        if (cote < -2)
          cote = -2;
        coul = (int)cote;
        // projection sur un plan
        X = a1 * i + a2 * j + a3 * coul;
        Y = b1 * i + b2 * j + b3 * coul;

        test.x = X;
        test.y = Y;
        if (PtInRect(&zone_invalide, test))
// le point est-il invalide ?
        {
          int zm = (int)sqrt((t.re * c.re + t.im * c.im));
          if (zm != 0)
            zm = 127+127 / zm;

```

```

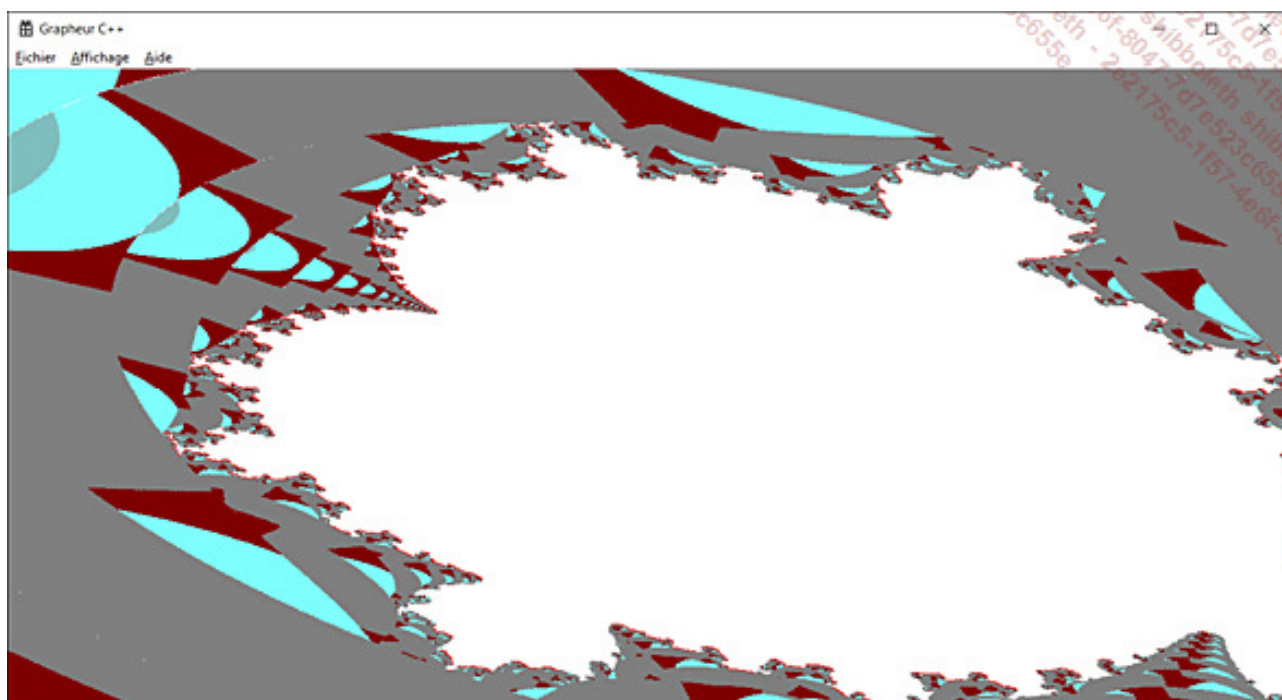
        int couleur = RGB(127,(int)zm,(int)zm);
        SetPixel(hdc, X, Y, couleur);
    }
    break;
}

z1.re = z2.re;
z1.im = z2.im;
}

if (n >= max)
{
    // le point est resté stable, il fait partie du chapeau
    test.x = a1 * i + a2 * j + a3 * n * 20;
    test.y = b1 * i + b2 * j + b3 * n * 20;

    if (PtInRect(&zone_invalide, test))
// le point est-il invalide ?
    {
        SetPixel(hdc, X, Y, RGB(255, 0, 0));
    }
}
}
}
}
#pragma endregion

```



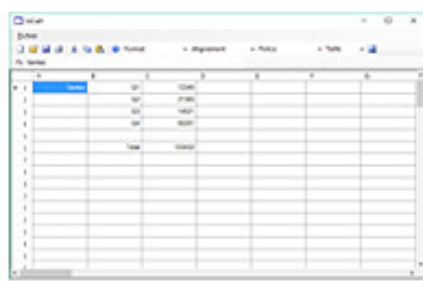
2. Une application en C++ pour .NET : le tableur InCell

Nous proposons maintenant d'étudier la réalisation d'un tableur graphique en C++ CLI. Le but ici est d'expliquer la mise en œuvre de mécanismes de C++ CLI au travers d'une application conséquente, sans rentrer dans l'analyse fonctionnelle. Pour ceux qui souhaitent aller plus loin, l'ensemble du code source est documenté et fourni en téléchargement parmi les fichiers complémentaires pour cet ouvrage.

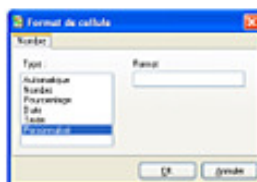
a. Architecture du tableur

L'architecture repose sur deux assemblages distincts : l'interface graphique InCell et l'unité de calcul CLITableur. Le premier assemblage est un exécutable WinForm tandis que le second est une bibliothèque de classes .DLL.

Assemblage InCell (C#)



Form1:WinForm



FormFormat:WinForm



FormAbout:WinForm

Assemblage CLITableur (C++)

Feuille de calcul :
Ensemble de cellules
Persistance

Cellule

Ordonnanceur :
détermine le plan
de calcul

Evaluateur de formules

Fichiers *.icls

L'exécutable est principalement constitué de formulaires (d'écrans) `WinForm` qui définissent l'apparence et le comportement de l'application graphique. On y trouve le formulaire principal `Form1`, la boîte de dialogue de formatage de cellule `FormFormat` et la boîte de dialogue d'à propos `FormAbout`. La fonction principale `main()` instancie et affiche `Form1` qui contrôle l'exécution de l'application jusqu'à sa fermeture. La programmation est événementielle et c'est au travers de menus et de barres d'outils que l'essentiel des opérations sont déclenchées. Comme nous le verrons un peu plus loin, des événements particuliers sont également gérés au niveau de la grille de donnée pour recalculer la feuille lorsque cela est nécessaire.

Le second assemblage est indépendant de l'interface graphique. Pour éviter les interdépendances, nous avons utilisé des événements pour ne pas tomber dans le piège de l'interface graphique qui commande un calcul, lequel doit ensuite être affiché sur l'interface graphique. La classe `Feuille` est un tableau de `Cellule`, cette classe

représentant une valeur, une formule, un type, un format d'affichage, un style...

Dans un tableur, les formules ne peuvent pas toujours s'évaluer de gauche à droite et de haut en bas, un ordonnanceur est intégré à la feuille de calcul pour déterminer l'ordre adéquat dans lequel les opérations d'actualisation auront lieu. Il manque encore l'évaluateur de formules qui calcule des expressions à base de constantes, de fonctions intrinsèques (cosinus, somme, moyenne...) ou de références à des cellules.

b. La feuille de calcul

Avant d'expliciter la classe `Feuille`, commençons par la classe `Cellule`. Une cellule est une formule dont l'évaluation donne une valeur. Cette valeur est ensuite formatée et affichée sur la grille de données en respectant un certain style. L'extrait qui suit donne un aperçu de cette classe où plusieurs mécanismes propres à C++ CLI sont utilisés : propriétés, énumérations, ainsi que les commentaires `///` destinés à fabriquer la documentation du projet au format XML.

```
/// <summary>
/// Cellule représente une valeur ou une formule formatée
/// </summary>
public ref class Cellule
{

private:
    /// <summary>
    /// Valeur "calculée" d'une cellule (membre privé)
    /// </summary>
    String^ value;

    /// <summary>
    /// Formule (membre privé)
    /// </summary>
    String^ formula;

    /// <summary>
    /// Style gras (membre privé)
    /// </summary>
    bool isBold;
```

```

/// <summary>
/// Style italique (membre privé)
/// </summary>
bool isItalic;

/// <summary>
/// Style souligné (membre privé)
/// </summary>
bool isUnderlined;

/// <summary>
/// Style couleur de fond (membre privé)
/// </summary>
Color backgroundColor;

/// <summary>
/// Style couleur d'avant-plan (membre privé)
/// </summary>
Color foreColor;

/// <summary>
/// Style police (membre privé)
/// </summary>
String^ fontName;

/// <summary>
/// Style taille de la police (membre privé)
/// </summary>
float fontSize;

/// <summary>
/// Style alignement (membre privé)
/// </summary>
DataGridViewContentAlignment alignment;

public:
/// <summary>
/// Formatages possibles de la valeur
/// </summary>

```

```

enum class FormatType
{
    AUTOMATIQUE, NOMBRE, POURCENTAGE, DATE, TEXT, PERSONNALISE
};

private:
    /// <summary>
    /// Formatage de la valeur (membre privé)
    /// </summary>
    FormatType format;

    /// <summary>
    /// Chaîne de formatage (membre privé)
    /// </summary>
    String^ formatString;

public:
    /// <summary>
    /// Indique si la valeur peut être reconnue comme nombre
    /// </summary>
    bool IsNumber();

    /// <summary>
    /// Constructeur
    /// </summary>
    Cellule();

    /// <summary>
    /// Sérialise la cellule au format XML
    /// </summary>
    XmlNode^ ToXml(XmlDocument^ doc, int col, int row);

    /// <summary>
    /// Désérialise la cellule à partir d'un noeud XML
    /// </summary>
    void FromXml(XmlNode^ n);
};

```

La classe `Feuille` contient un tableau à deux dimensions représentant la matrice de `Cellule`. Là encore, des mécanismes de C++ CLI sont employés, comme les délégués et les événements. Nous verrons un peu plus loin comment l'interface graphique s'y "connecte". Nous trouvons aussi des propriétés facilitant les translations d'un domaine à l'autre, par exemple, pour retrouver le nom court d'un fichier.

```

/// <summary>
/// Feuille de calcul
/// </summary>
public ref class Feuille
{
public:
    /// <summary>
    /// Fonction appelée quand une cellule est mise à jour
    /// </summary>
    delegate void del_cellUpdated(int col, int row, String^ value);

    /// <summary>
    /// Déclenché lorsqu'une cellule est mise à jour (calculée)
    /// </summary>
    event del_cellUpdated^ OnCellUpdated;

    /// <summary>
    /// Fonction appelée lorsque le style est modifié
    /// </summary>
    delegate void del_cellStyleUpdated(int col, int row);

    /// <summary>
    /// Déclenché lorsque le style est modifié
    /// </summary>
    event del_cellStyleUpdated^ OnCellStyleUpdated;

    // taille de la feuille

    /// <summary>
    /// Nombre de lignes max
    /// </summary>
    const static int ROWS = 100;

    /// <summary>

```



```

/// Nombre de colonnes max
/// </summary>
const static int COLS = 52;

private:
    int rows;
    int cols;

    /// <summary>
    /// Cellules composant la feuille de calcul
    /// </summary>
    array<Cellule^,2>^ cells;

public:
    /// <summary>
    /// Accès indexé aux cellules
    /// </summary>
    Cellule^ Cells(int col,int row);

    /// <summary>
    /// Constructeur
    /// </summary>
    Feuille();

private:
    /// <summary>
    /// Évalue le contenu d'une cellule
    /// </summary>
    String^ evaluate(Cellule^ cell, int col, int row);

    /// <summary>
    /// Appelé pour capturer la valeur d'une cellule
    /// </summary>
    void ev_OnGetCellValue(String^ name, double & value);

    /// <summary>
    /// Identifie les références des cellules contenues dans une plage
    /// </summary>
    array<Cellule^>^ GetRange(String^ range);

```

```

/// <summary>
/// Instance d'évaluateur
/// </summary>
Evaluator^ ev;

/// <summary>
/// Erreurs
/// </summary>
StringBuilder^ errors;

public:
    /// <summary>
    /// Recalcule la feuille après détermination du plan
    /// </summary>
    void recal();

private:
    /// <summary>
    /// Chemin et nom de fichier (membre privé)
    /// </summary>
    String^ filename;

    /// <summary>
    /// Nom du document (membre privé)
    /// </summary>
    String^ fileshortname;

    /// <summary>
    /// Indique si une modification a eu lieu
    /// depuis la dernière sauvegarde (membre privé)
    /// </summary>
    bool isModified;

    /// <summary>
    /// Enregistre le document vers le fichier filename
    /// </summary>
    void SaveToFilename();

    /// <summary>
    /// Produit un flux XSLX

```

```

/// </summary>
String^ GetExcelStream();

public:
    /// <summary>
    /// Constructeur applicatif
    /// </summary>
    void Init();

    /// <summary>
    /// Indique si le document a été modifié
    /// depuis la dernière sauvegarde
    /// </summary>
    property bool IsModified
    {
        bool get()
        {
            return isModified;
        }
    }

    /// <summary>
    /// Obtient ou définit le nom/chemin du document
    /// </summary>
    property String^ Filename
    {
        String^ get()
        {
            return filename;
        }

        void set(String^ value)
        {
            this->filename = value;

            FileInfo^ f = gcnew FileInfo(filename);
            this->fileshortname = f->Name;
        }
    }
}

```

```

/// <summary>
/// Obtient le nom du document
/// </summary>
property String^ Fileshortname
{
    String^ get()
    {
        return fileshortname;
    }
}

/// <summary>
/// Enregistre le document
/// </summary>
void Save();

/// <summary>
/// Enregistre sous un autre nom
/// </summary>
void Save(String^ filename);

/// <summary>
/// Charge le document
/// </summary>
void Load(String^ filename);

/// <summary>
/// Enregistre une copie au format XLSX
/// </summary>
void SaveCopyExcel(String^ filename);
};

```

Comme détail d'implémentation de la classe `Feuille`, nous avons choisi la méthode `GetRange` qui emploie des listes génériques :

```

/// <summary>
/// Identifie les références des cellules contenues dans une plage

```

```

/// </summary>
array<Cellule^>^ Feuille::GetRange(String^ range)
{
    array<String^>^ names = range->Split(gcnew array<wchar_t> { '.' });
    System::Collections::Generic::List<Cellule^>^ res =
        gcnew System::Collections::Generic::List<Cellule^>();

    Evaluator^ ev = gcnew Evaluator();
    int col1 = 0, col2 = 0;
    int row1 = 0, row2 = 0;

    ev->CellNameToColRow(names[0], col1, row1);
    ev->CellNameToColRow(names[1], col2, row2);

    for (int i = col1; i <= col2; i++)
        for (int j = row1; j <= row2; j++)
        {
            res->Add(cells[i, j]);
        }

    return res->ToArray();
}

```

La méthode `SaveToFilename` met en œuvre quant à elle l'API XML de .NET que nous avons étudiée précédemment :

```

/// <summary>
/// Enregistre le document vers le fichier filename
/// </summary>
void Feuille::SaveToFilename()
{
    XmlDocument^ doc = gcnew XmlDocument();
    XmlNode^ root = doc->CreateElement("document");
    doc->AppendChild(root);
    for (int col=0; col<cols; col++)
        for (int row = 0; row < rows; row++)
        {
            if (cells[col, row]->Value != "" ||
                cells[col, row]->Formula != "")

```

```

    {
        XmlNode^ n = cells[col, row]->ToXml(doc,col,row);
        root->AppendChild(n);
    }
}

StringBuilder^ sb = gcnew StringBuilder();
XmlTextWriter^ writer =
gcnew XmlTextWriter(gcnew StringWriter(sb));
writer->Formatting = Formatting::Indented;

doc->WriteTo(writer);
writer->Flush();

FileStream^ fs =
gcnew FileStream(filename, FileMode::Create, FileAccess::Write);
StreamWriter^ sw = gcnew StreamWriter(fs);
sw->Write(sb->ToString());
sw->Flush();
fs->Close();

isModified = false;
}

```

c. L'ordonnanceur de calcul

L'ordonnanceur, qui détermine dans quel ordre les formules des cellules doivent être évaluées, repose sur l'implémentation d'un graphe. Cette structure de donnée n'existant pas nativement, nous l'avons créée en C++. Ici, il n'y a pas de mécanisme propre à C++ CLI, si ce n'est l'emploi des structures de données de `System::Collection::Generic (List, Stack)`, ainsi que de structures non typées comme `Hashtable`.

```

/// <summary>
/// Représente un sommet au sein d'un graphe
/// </summary>
public ref class Sommet
{

```

```

public:
    /// <summary>
    /// Nom du sommet
    /// </summary>
    String^ name;

    /// <summary>
    /// Sommets du graphe adjacents
    /// </summary>
    List<Sommet^>^ adjacents;

public:
    Sommet()
    {
        adjacents = gcnew List<Sommet^>();
        name = "";
    }
};

/// <summary>
/// Représente un graphe dirigé
/// </summary>
public ref class Graphe
{
private:
    /// <summary>
    /// Ensemble des sommets du graphe (membre privé)
    /// </summary>
    List<Sommet^>^ g;

public:
    /// <summary>
    /// Constructeur
    /// </summary>
    Graphe()
    {
        g = gcnew List<Sommet^>();
    }

public:

```

```

/// <summary>
/// Ajoute un sommet au graphe, avec ses dépendances
/// (sommets adjacents)
/// </summary>
void AjouterSommet(String^ name, array<String^>^ dependances)
{
    Sommet^ adj;
    Sommet^ s;
    if ((s = GetSommetByName(name)) == nullptr)
    {
        // le sommet n'existe pas encore
        s = gnew Sommet();
        s->name = name->ToUpper();
        g->Add(s);
    }

    // ajoute ses dépendances
    for (int i = 0; i < dependances->Length; i++)
        if ((adj = GetSommetByName(dependances[i])) != nullptr)
        {
            s->adjacents->Add(adj);
        }
        else
        {
            // crée un nouveau sommet adjacent
            adj = gnew Sommet();
            adj->name = dependances[i]->ToUpper();
            g->Add(adj);
            s->adjacents->Add(adj);
        }
    }

public:
    /// <summary>
    /// Identifie le sommet appelé name
    /// </summary>
    Sommet^ GetSommetByName(String^ name)
    {
        for (int i=0;i<g->Count;i++)
        {

```



```

        Sommet^s = g[i];
        if (s->name == name)
            return s;
    }
    return nullptr;
}

/*
 * Tri topologique
 */
private:
    Hashtable^ visites;
    void init_visites()
    {
        visites = gcnew Hashtable();
    }

    System::Collections::Generic::Stack<Sommet^>^ pile;
    Sommet^ depart;

public:
    /// <summary>
    /// Effectue un tri topologique à partir des adjacences
    /// entre sommets (dépendances inverses)
    /// </summary>
    array<String^>^ TriTopologique()
    {
        pile = gcnew System::Collections::Generic::Stack<Sommet^>();
        List<String^>^ plan = gcnew List<String^>();
        hasError = false;
        errors = gcnew StringBuilder();

        for (int i=0;i<g->Count;i++)
        {
            Sommet^s = g[i];

            init_visites();
            depart = s;

```

```

    // le tri est basé sur un parcours en profondeur d'abord
    parcours_profondeur(s);
}

while (pile->Count != 0)
    plan->Add(pile->Pop()->name);

return plan->ToArray();
}

private:
void parcours_profondeur(Sommet^ s)
{
    if (s == nullptr)
        return;

    if (visites[s->name] != nullptr /*&& visites[s->name]
is bool*/)
        return;
    else
        visites[s->name] = true;

    pile->Push(s);
    for(int i=0;i<s->adjacents->Count;i++)
    {
        Sommet^ adj=s->adjacents[i];
        if (adj == depart)
        {
            // déjà visité : erreur, graphe cyclique
            hasError = true;
            errors->Append("Référence circulaire sur
" + depart->name + " : " + s->name + "\n");
            return;
        }

        parcours_profondeur(adj);
    }
}

};

```

d. Zoom sur l'évaluateur

L'évaluateur étant une classe assez longue, nous avons recouru aux régions pour délimiter certaines parties du code :

```

Evaluator.cpp  Evaluator.h
{} CLITableur

namespace CLITableur
{
    /*
     * Analyseur grammatical
     */
    Analyseur grammatical

    /*
     * Analyseur lexical
     */
    #pragma region Analyseur lexical
    #pragma region back
    void Evaluator::back()
    {
        next_value = current_value;
        next_token = current_token;
        has_next = true;
    }
    #pragma endregion

    look_ahead

    next

    GetNames
    #pragma endregion
}

```

La méthode suivante est assez importante : elle utilise un événement pour solliciter une valeur issue d'une classe de niveau supérieur.

```

void Evaluator::cellule(String^ name, Variant^ f)
{
    double value = 0;

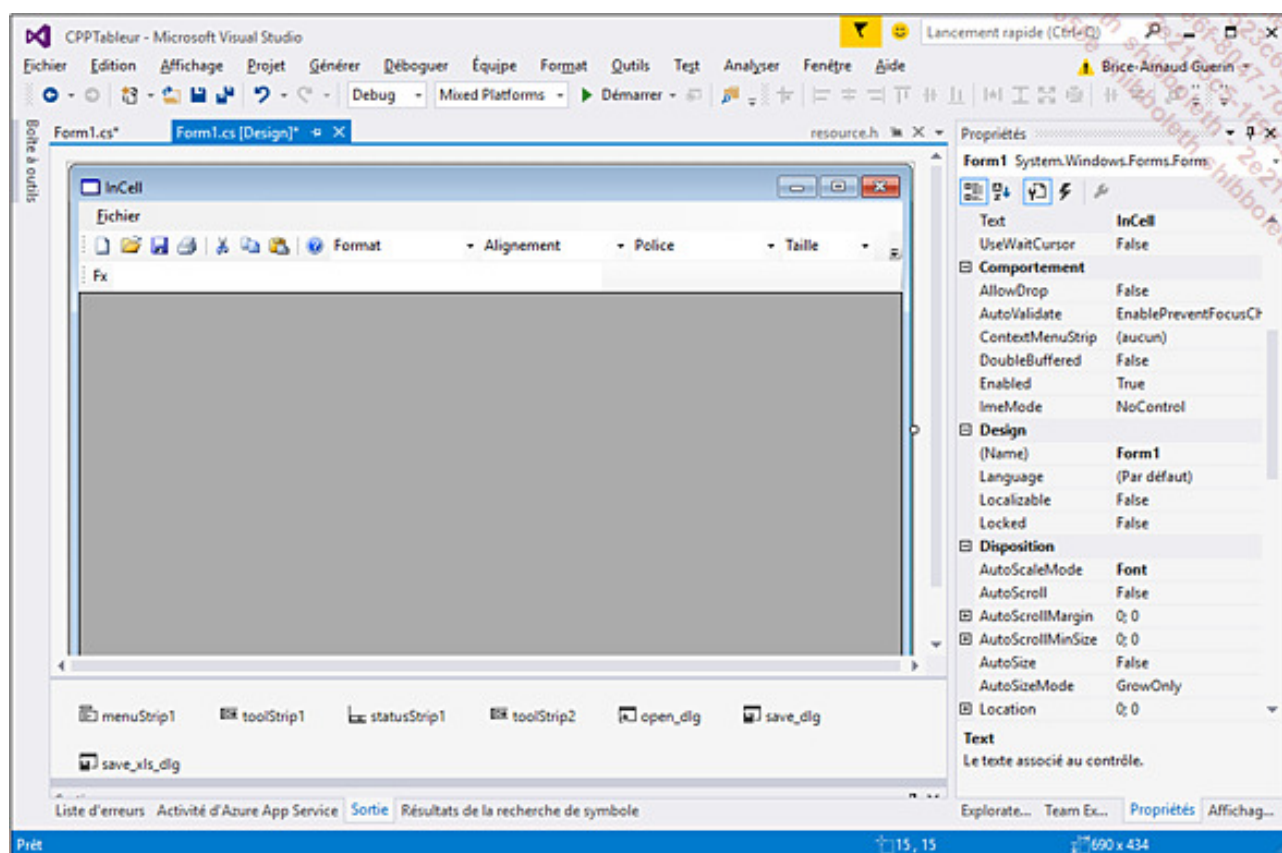
    // un événement cherche la valeur de la cellule
    // sans ajouter une référence à Cells
    OnGetCellValue(name, value);
    f->nValue = value;
}

```

Sans événement, il aurait fallu transmettre une référence de `Cells` à l'objet `Evaluator`, ce qui aurait créé une référence circulaire. Ces anomalies perturbent le fonctionnement du GC et provoquent des fuites mémoire (la mémoire est allouée et ne peut être relâchée).

e. L'interface graphique

La présence d'un concepteur visuel est quasiment indispensable s'il l'on souhaite créer des interfaces de haut niveau. Malheureusement, Microsoft ne propose plus de concepteur WinForm en C++, aussi l'interface graphique est-elle implantée en C#.



Ce concepteur laisse toute la liberté au programmeur pour positionner les contrôles sur un écran, modifier les propriétés d'apparence et de comportement, et enfin créer le code événementiel.

The screenshot shows the InCell application window. The menu bar includes 'Fichier', 'Format', 'Alignement', 'Police', and 'Taille'. The formula bar displays $Fx = C1 + C2 + C3 + C4$. The spreadsheet contains the following data:

	A	B	C	D	E	F	G
1	Ventes	Q1	45213				
2		Q2	58471				
3		Q3	23796				
4		Q4	61015				
5							
6		Total	188495				
7							
8							
9							
10							
11							
12							

3. Un module de gestion de données pour tiny-lisp

Nous avons étudié au chapitre Programmation orientée objet l'aspect modulaire de l'interprète tiny-lisp. Cette modularité est illustrée dans cet exemple de base de données qui se montre très économe en code.

Le module DB crée des listes persistantes, c'est-à-dire enregistrées sur disque d'une session à l'autre de l'interprète. Son activation se fait au niveau de la classe `Configuration` :

```
Configuration* Configuration::instance=0;

void Configuration::define_modules()
{
    Configuration::getInstance()->add_module(new MathModule());
    Configuration::getInstance()->add_module(new DBModule());
}
```

```
}
```

La méthode virtuelle `register_globals()` définit les points d'entrée du module, c'est-à-dire les symboles et les procédures qui le composent. On remarque que l'environnement global est lui-même conservé car le module DB va instancier de nouveaux symboles lorsque des listes persistantes seront fabriquées.

```
#pragma region register_globals
void DBModule::register_globals(Environment& env)
{
    env[TOKEN_USE] = Variant(&DBModule::db_use);
    env[TOKEN_CREATE] = Variant(&DBModule::db_create);
    env[TOKEN_INSERT] = Variant(&DBModule::db_insert);
    env[TOKEN_SELECT] = Variant(&DBModule::db_select);
    env[TOKEN_SAVE] = Variant(&DBModule::db_save);

    DBModule::global_env = &env;
}
#pragma endregion
```

a. Création de listes persistantes `db_create`

L'implémentation de `db_create()` ne présente aucune difficulté. Une liste est instanciée vide, puis matérialisée au format JSON dans un fichier portant son nom :

```
#pragma region db_create
Variant DBModule::db_create(const variants& c)
{
    if (c.size() > 1)
        throw ParseError(ERROR_CODE_WRONG_ARGC,
            std::string("wrong argument list for db_create"));

    Variant db(List);
    string filename = (*(c.begin())).val;
    db.val = filename;
    (*global_env)[filename.c_str()] = db;
}
```

```

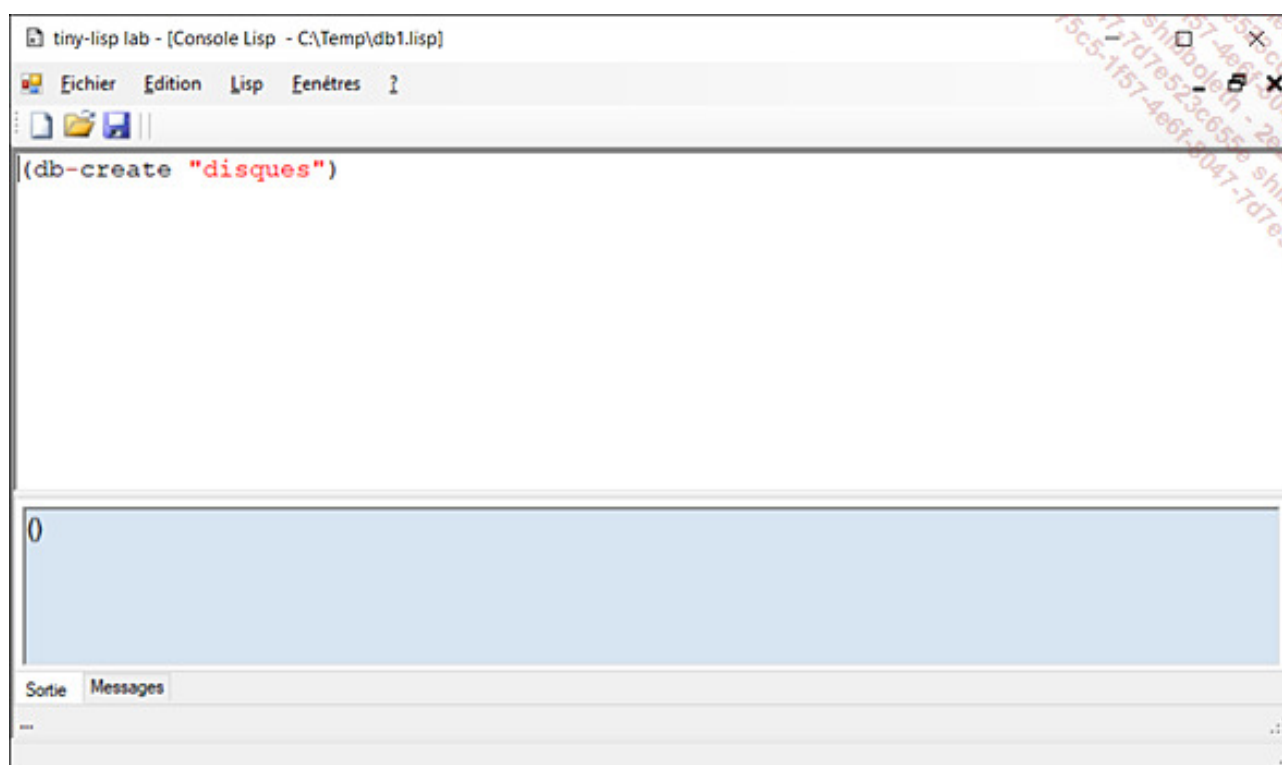
ofstream db_file;

db_file.open(DATA_PATH + filename);
db_file << db.to_json_string();
db_file.close();

return db;
}
#pragma endregion

```

Pour tester la création d'une liste persistante, il faut appeler `db_create` en indiquant son nom entre guillemets :



L'interprète renvoie bien une liste vide `()`.

b. Accès aux listes persistantes `db_use`

La procédure `db_use()` est le symétrique de `db_create()` ; elle recharge une liste sérialisée au format JSON et l'insère dans l'environnement global :


```

#pragma region db_use
Variant DBModule::db_use(const variants& c)
{
    if (c.size() > 1)
        throw ParseError(ERROR_CODE_WRONG_ARGC,
std::string("wrong argument list for db_use"));

    Variant db;
    ifstream db_file;
    string table;

    string filename = (*(c.begin())).val;
    db_file.open(DATA_PATH + filename);
    getline(db_file, table);
    db_file.close();

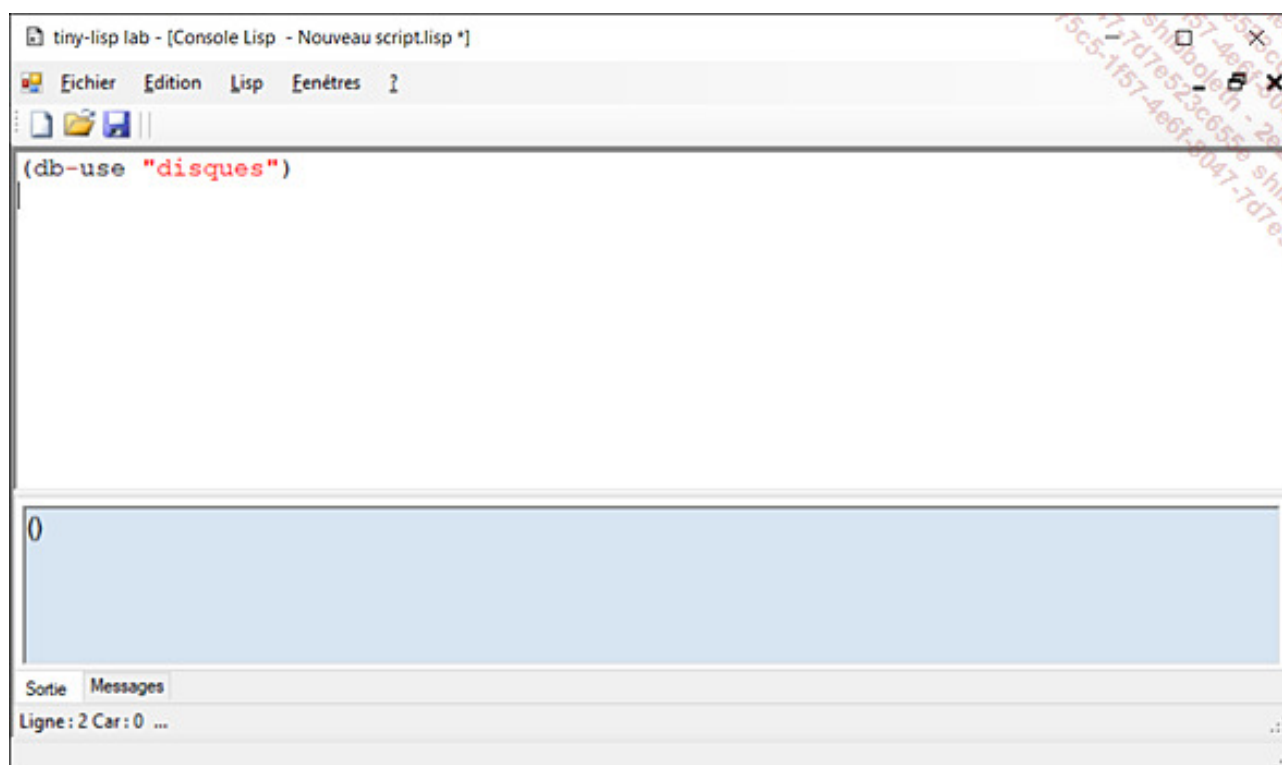
    db = Variant::from_json_string(table);
    db.val = filename;

    (*global_env)[filename] = db;

    return db;
}
#pragma endregion

```

Le test est similaire à `db_create` . Pour l'instant, la liste est vide :



c. Insertion d'items db_insert

La procédure `db_insert()` réalise un parcours de collection pour insérer les items dans la liste persistante :

```
#pragma region db_insert
Variant DBModule::db_insert(const variants& c)
{
    if (c.size() < 2)
        throw ParseError(ERROR_CODE_WRONG_ARGC,
            std::string("wrong argument list for db_insert"));

    Variant db = (*global_env)[(*c.begin()).val];

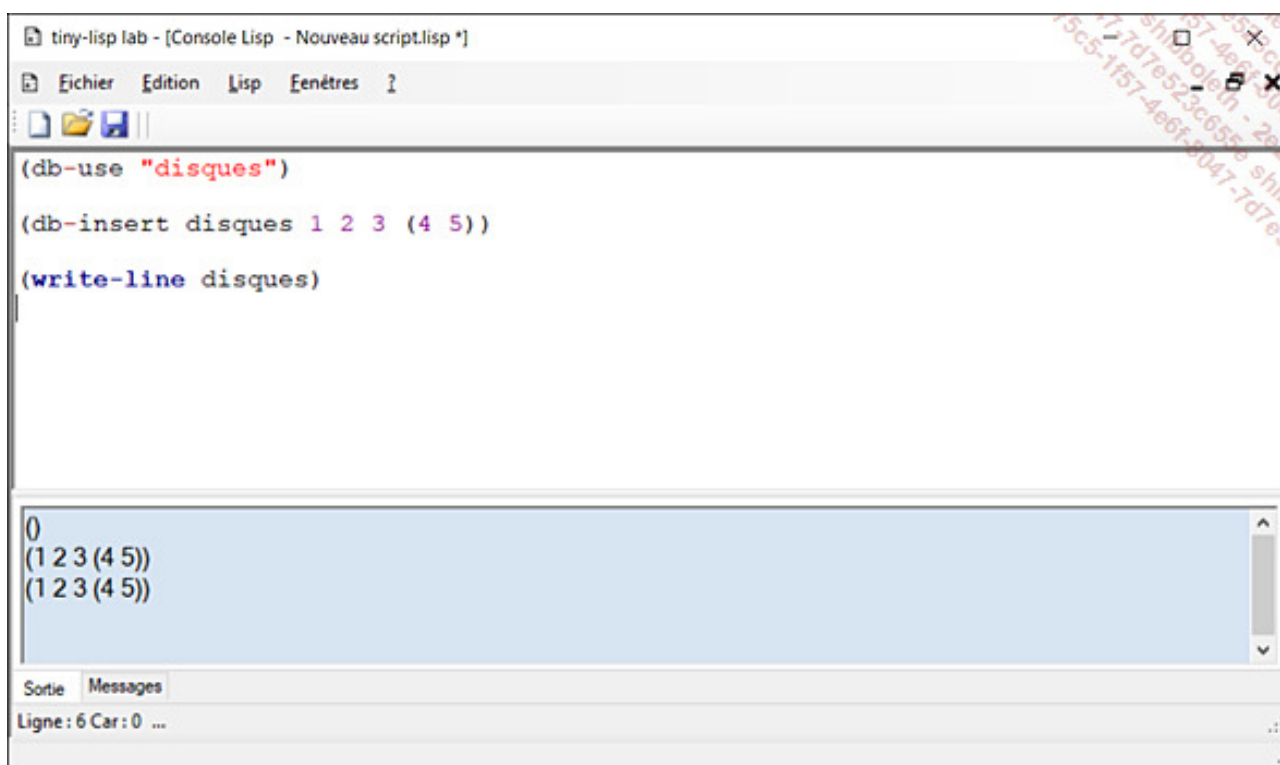
    for (auto ic = c.begin()+1; ic != c.end(); ic++)
    {
        Variant nc = *ic;
        db.list.push_back(nc);
    }
    (*global_env)[db.val] = db;
}
```

```

    return db;
}
#pragma endregion

```

À l'exécution, une fois la liste modifiée par l'ajout d'éléments (quatre items dans cet exemple), elle s'utilise comme n'importe quelle liste :



d. Enregistrement des listes persistantes db_save

À la différence de `db_create()`, `db_save()` n'instancie pas la liste puisqu'elle la tire de l'environnement global :

```

#pragma region db_save
Variant DBModule::db_save(const variants& c)
{
    if (c.size() > 1)
        throw ParseError(ERROR_CODE_WRONG_ARGC,
            std::string("wrong argument list for db_save"));
}

```

```

Variant db = (*global_env)[(*c.begin()).val];
string filename = (*(c.begin())).val;
db.val = filename;
(*global_env)[filename.c_str()] = db;

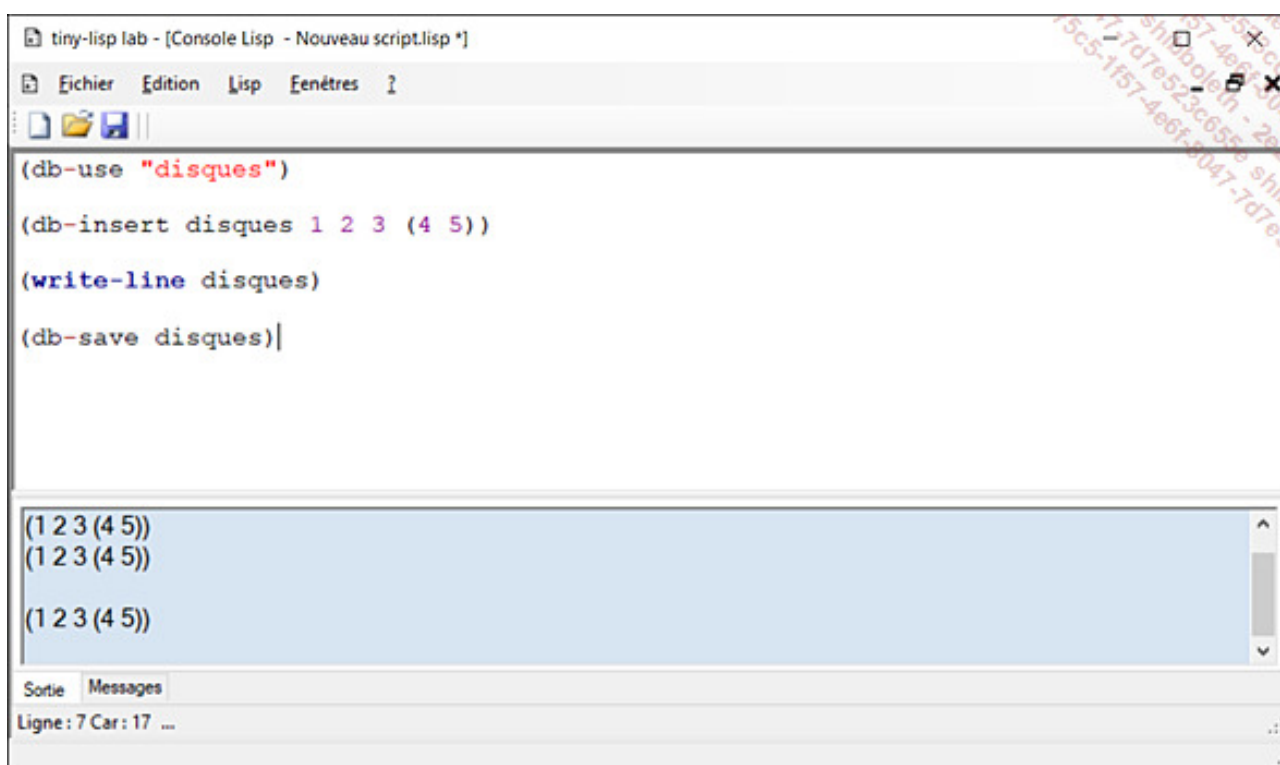
ofstream db_file;

db_file.open(DATA_PATH + filename);
db_file << db.to_json_string();
db_file.close();

return db;
}
#pragma endregion

```

Pour le prochain exemple, il faudra mettre en commentaire la ligne `db_insert` sous peine de dupliquer les items !



The screenshot shows a window titled "tiny-lisp lab - [Console Lisp - Nouveau script.lisp *]". The menu bar includes "Fichier", "Edition", "Lisp", "Fenêtres", and "?". The toolbar has icons for file operations. The main text area contains the following Lisp code:

```

(db-use "disques")

(db-insert disques 1 2 3 (4 5))

(write-line disques)

(db-save disques)|

```

The output area at the bottom shows three lines of results: "(1 2 3 (4 5))", "(1 2 3 (4 5))", and "(1 2 3 (4 5))". The status bar at the bottom indicates "Ligne : 7 Car : 17 ...".

e. Sélection d'items dans les listes persistantes `db_select`

L'instruction `db_select` est un peu plus élaborée. Elle reçoit comme paramètre une lambda-expression tiny-lisp appliquée à chaque item de la liste et renvoyant un booléen ; si le résultat est `#t` (true), l'item est sélectionné, sinon il est écarté de la sélection.

```
#pragma region db_select
Variant DBModule::db_select(const variants& c)
{
    if (c.size() < 2)
        throw ParseError(ERROR_CODE_WRONG_ARGC,
std::string("wrong argument list for db_select"));

    Variant db;
    string filename = (*(c.begin())).val;
    db = (*global_env)[filename];

    // (proc exp*)
    Variant proc(c[1]);
    Parser p(global_env);

    Variant select;
    select.type = List;
    for (auto& e : db.list)
    {
        // évaluer les paramètres
        variants exps;
        exps.push_back(e);

        // utiliser des smart pointers
        Environment* newenv = new Environment(/*parms*/proc.list[1].list,
/*args*/ exps,
/*outer env*/ proc.env);

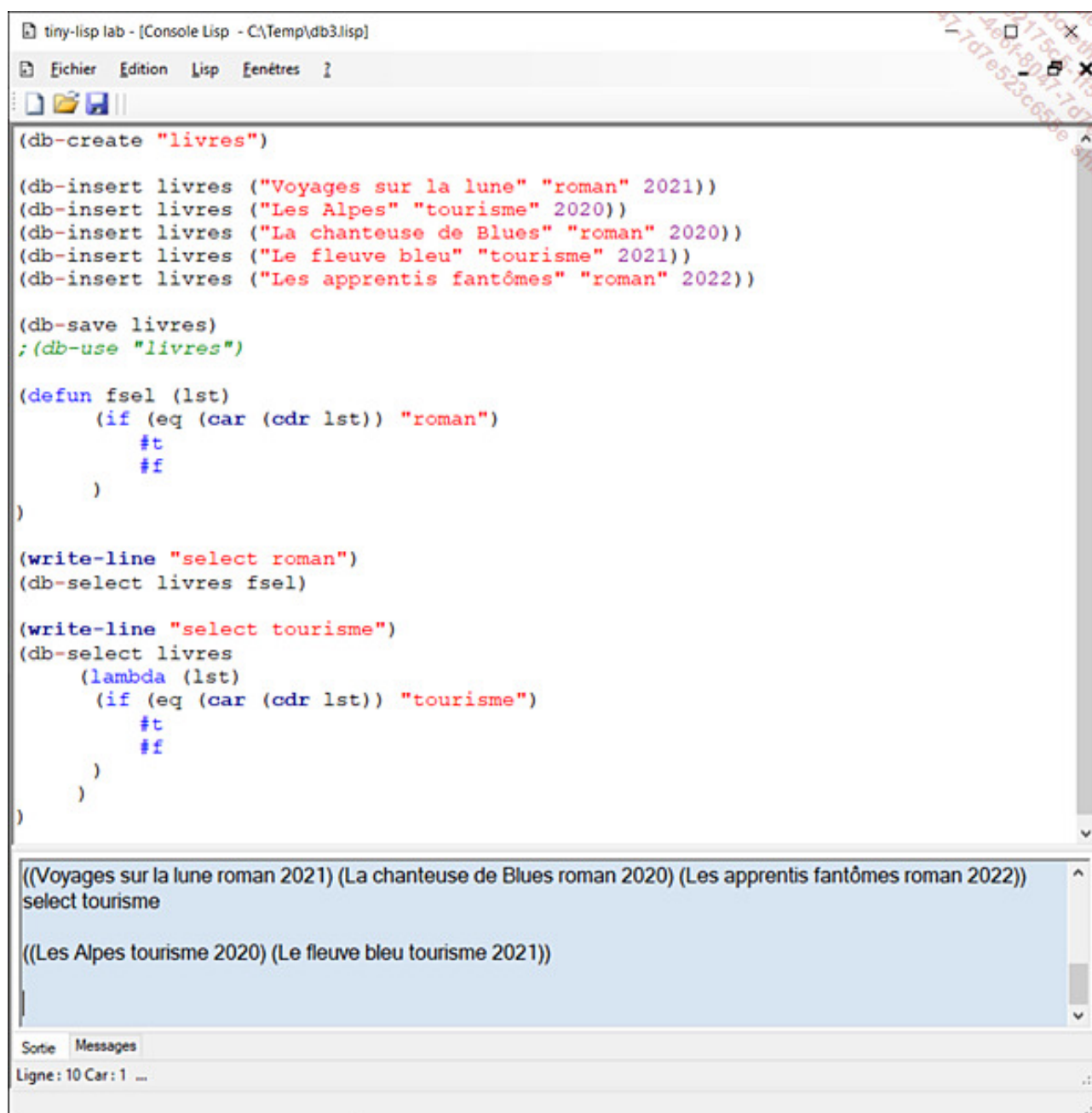
        Variant ret = p.eval(/*body*/proc.list[2], newenv);

        // vérifier si impact sur la variable globale
        delete newenv;

        if (ret.val == "#t")
            select.list.push_back(e);
    }
}
```

```
    return select;  
}  
#pragma endregion
```

Dans l'exemple d'application on définit une liste de livres de différents genres (roman, tourisme...). Pour la sélection des romans, une lambda-expression tiny-lisp est définie avant l'appel de `db_select`. Pour la sélection des ouvrages de tourisme, la lambda-expression tiny-lisp est définie directement dans le corps de `db_select` :



```

tiny-lisp lab - [Console Lisp - C:\Temp\db3.lisp]
Fichier  Edition  Lisp  Fenêtres  ?

(db-create "livres")

(db-insert livres ("Voyages sur la lune" "roman" 2021))
(db-insert livres ("Les Alpes" "tourisme" 2020))
(db-insert livres ("La chanteuse de Blues" "roman" 2020))
(db-insert livres ("Le fleuve bleu" "tourisme" 2021))
(db-insert livres ("Les apprentis fantômes" "roman" 2022))

(db-save livres)
;(db-use "livres")

(defun fsel (lst)
  (if (eq (car (cdr lst)) "roman")
      #t
      #f)
)

(write-line "select roman")
(db-select livres fsel)

(write-line "select tourisme")
(db-select livres
  (lambda (lst)
    (if (eq (car (cdr lst)) "tourisme")
        #t
        #f)
    )
)

((Voyages sur la lune roman 2021) (La chanteuse de Blues roman 2020) (Les apprentis fantômes roman 2022))
select tourisme

((Les Alpes tourisme 2020) (Le fleuve bleu tourisme 2021))

Sortie  Messages
Ligne: 10 Car: 1 ...

```