第１章　プレゼンターを理解するための準備

ポイント：プレゼンターを理解するための準備を行う。

プレゼンターシステムを理解するためには、CG やアニメーションに関するテクニックが必要となる。ここでは、プレゼンターの開始と初期設定について、またプレゼンターではどのような技術が必要となるかについて学ぶ。

1.1 はじめに

このマニュアルは、読者がプレゼンターで使用されているテクニックを学び、CG やアニメーションなどに関する基本的なプログラム技術を習得するために書かれた教科書である。SPACE は、非常に複雑なシステムで、単に非線形解析を効率的に実行するだけでなく、得られた結果を処理し、ユーザーが理解し易いように色々な形式で情報を提供する。SPACE システムを理解するためには、数値解析の技術に加えて、これらの処理に関する各種の技術、特に、GUI（グラフィカルユーザーインターフェイス）に関する知識が必須となる。

演習システム：プレゼンター作成入門
このマニュアルは、図形処理用プログラム、特に、SPACE のプレゼンターシステムの中に含まれている図形処理に関するコードを例題とし、図形処理やウインドウ管理などを学ぶための演習システムである。

「プレゼンター作成入門編」は、演習システム「静的解析編」と「動的解析編」と対を成すマニュアルであり、SPACE のようなシミュレーションシステムを開発、あるいは勉強しようとしている学生、技術者に対する教科書となっている。「静的解析編」は、線形の静的解析と座屈解析について説明し、また、「同：動的解析編」では線形の動的解析と固有振動数と振動モードを求める固有問題解析について解説している。「プレゼンター作成入門編」は、図形処理、ウインドウの管理法、アニメーション技術について解説している。これら３つのマニュアルは、説明文が多く、分かり易く書かれている。これら３つの入門書に関するマニュアルを十分に理解することで、構造解析で用いる数値計算の基本や CG やアニメーションなどの技術を確実に把握でき、その結果、SPACE に関する他のマニュアルも理解できることになろう。

このマニュアルは、Visual C++と Fortran 言語が理解されていることを前提に書かれている。SPACE は Microsoft Visual Studio (Developer Studio)を利用して開発されていることから、このプログラム開発システムの操作法も理解していることを前提にしている。両者を十分にマスターしていない場合は、そちらを先に勉強し、慣れておいていただきたい。マニュアルの中には多くの関数が説明なしで直接表示されることもある。そこで、Visual C++言語のリファレンスマニュアルなどを傍に置き、新しい関数が出てくる毎に、その関数や引数の意味、あるいは使用

VC++に関する多くの関数が出現する。これらを理解するために、VC++のリファレンスマニュアルや参考書を常に参照されたい。

方法を調べると良い。標準的なテクニックが直ぐに理解できるようになろう。

1.2 イベント駆動型プログラミングとオブジェクト指向型言語

オブジェクト指向型言語であるVC++によるプログラムは、数値計算に用いるコードと大きく異なる部分があり、FORTRANで育った人達には理解しにくいかもしれない。その原因は、プログラムがイベント駆動型であることとオブジェクト、クラス、マルチスレッドなどの新しい概念が多く出現することにある。一般的に、プログラム中の関数は、メッセージによって駆動され、関数が実行される。このように逐次型のプログラムと全く異なった動作で処理されるため、プログラムを読むことも書くことも、最初は戸惑うことになろう。このイベント駆動型の構造を一旦理解すれば、プレゼンターシステムを理解することはそれほど難しくはない。

イベント駆動型プログラムの構造は、Microsoft Visual Studio (Developer Studio)と Microsoft Visual C++ 6.0 を使用することで、簡単に構築することができる。SPACEは、Microsoft Visual C++を使用して開発されていることもあって、ここでは、この Developer Studio で作成したプレゼンターを利用して、イベント駆動型プログラムの構造を学ぶことにする。

プレゼンターはオブジェクト指向型言語である VC++を用いて記述され、イベント駆動型プログラムとなっている。
プレゼンターの開発コード名は sf31 である。

動的解析用のプレゼンターシステムの中で、最も大切なCSf31Viewクラスの最初の部分を以下に示す。このクラスは、プレゼンターの子ウインドウを管理し、その中に表示する様々なグラフや表を表示する機能を有している。

イベント駆動型プログラムの基本構造が、メッセージマップと呼ばれる次のコード：

```
BEGIN_MESSAGE_MAP(CSf31View, CView)
 •
ON_WM_LBUTTONDOWN()
ON_COMMAND(ID_SWND_ST, OnSwndSt)
 •
END_MESSAGE_MAP()
```

イベント駆動型プログラムの基本構造であるメッセージマップ

で挟まれた記述の中に見られる。例えば、ON_WM_LBUTTONDOWN()は、マウスの左ボタンが押されたとき、この関数に処理が移ることになる。また、ON_COMMAND(ID_SWND_ST,OnSwndSt)は、システムからのメッセージID_SWND_STをこのクラスが受け取るとき、関数 OnSwndSt()が実行され

ることになる。これらの関数はこのクラスのメンバー関数であり、その処理内容は、このプログラムの後段で記述される。メッセージによって駆動された関数内の処理が全て終了すると、プレゼンターは、次のメッセージを受け取るまで何もしない、つまり停止した状態となる。これがイベント駆動型システムの基本的な動作法である。

以下に、CSf31View クラスのメッセージマップの全てを示す。この子ウインドウの View クラスでは、多くのメッセージを受け取り、多数の処理が実現されていることを理解されたい。また、このメッセージマップは、後章でしばしば参照されることになる。

```
//    ――――――――――――――――――――――――――――――――
//    ●   sf31View.cpp : CSf31View クラスの動作の定義
//    ――――――――――――――――――――――――――――――――
//    ●    マルチウインドウシステム定義
//    ――――――――――――――――――――――――――――――――
#include "malloc.h"
#include "stdafx.h"
#include "sf31.h"
#include "sf31dat.h"
#include "sf31Doc.h"
#include "sf31View.h"
#include "fort.h"
#include "ProgressDlg.h"
#include "Dlgstress_sp.h"
#include "Dig_prop_mem.h"
#include "Digme_spring.h"
#include "MainFrm.h"
#include "winuser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//    ――――――――――――――――――――――――――――――――
//    ●    ウインドウメッセージマップ
//    ――――――――――――――――――――――――――――――――
IMPLEMENT_DYNCREATE(CSf31View, CView)
BEGIN_MESSAGE_MAP(CSf31View, CView)
    //{{AFX_MSG_MAP(CSf31View)
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONUP()
    ON_WM_RBUTTONDOWN()
    ON_WM_RBUTTONUP()
    ON_WM_TIMER()
    ON_COMMAND(ID_SWND_ST, OnSwndSt)
    ON_COMMAND(ID_SWND_WV, OnSwndWv)
    ON_COMMAND(ID_SWND_SS, OnSwndSs)
```

各種のヘッダーファイルをインクルードしている。

マウスからのメッセージに対するメッセージマップを表す。

各種のメッセージに対するメッセージマップ。第１引数がメッセージ、第２引数が呼び出す関数を表す。

```
        ON_COMMAND(ID_WND_PROP, OnWndProp)
        ON_COMMAND(ID_OP_ROT, OnOpRot)
        ON_COMMAND(ID_OP_ROT_E, OnOpRotE)
        ON_COMMAND(ID_OP_ROT_1, OnOpRot1)
        ON_COMMAND(ID_OP_ROT_E1, OnOpRotE1)
        ON_COMMAND(ID_DYN_START, OnDynStart)
        ON_COMMAND(ID_DYN_STOP, OnDynStop)
        ON_WM_DESTROY()
        ON_COMMAND(ID_SWND_WV_Y, OnSwndWvY)
        ON_COMMAND(ID_SWND_WV_Z, OnSwndWvZ)
        ON_COMMAND(ID_SWND_SL, OnSwndSl)
        ON_COMMAND(ID_SWND_MD, OnSwndMd)
        ON_COMMAND(ID_SWND_EX, OnSwndEx)
        ON_COMMAND(ID_SWND_WV_DIS, OnSwndWvDis)
        ON_COMMAND(ID_SWND_MDW, OnSwndMdw)
        ON_COMMAND(ID_SWND_NO, OnSwndNo)
        ON_COMMAND(ID_SWND_SM, OnSwndSm)
        ON_COMMAND(ID_MEMO_MEM, OnMemoMem)
        ON_COMMAND(ID_MEMO_MEM_CR, OnMemoMemCr)
        ON_COMMAND(ID_MEMO_NOD, OnMemoNod)
        ON_COMMAND(ID_MEMO_NOD_CR, OnMemoNodCr)
        ON_COMMAND(ID_SWND_SS_MEM, OnSwndSsMem)
        ON_COMMAND(ID_SWND_SM_MEM, OnSwndSmMem)
        ON_COMMAND(ID_VIEW_TOOLBAR_X, OnViewToolbarX)
        ON_UPDATE_COMMAND_UI(ID_VIEW_TOOLBAR_X, OnUpdateViewToolbarX)
        ON_COMMAND(ID_DYN_RESET, OnDynReset)
        ON_COMMAND(ID_DYN_STEP, OnDynStep)
        ON_COMMAND(ID_DYN_BACK, OnDynBack)
        ON_COMMAND(ID_DYN_MAG, OnDynMag)
        ON_COMMAND(ID_SWND_MX_DIS, OnSwndMxDis)
        ON_COMMAND(ID_SWND_MX_STRESS, OnSwndMxStress)
        ON_COMMAND(ID_DYN_STEP_X, OnDynStepX)
        ON_COMMAND(ID_SWND_DEMDW, OnSwndDemdw)
        ON_COMMAND(ID_SWND_DEMDWX, OnSwndDemdwx)
        ON_COMMAND(ID_SWND_MDRES, OnSwndMdres)
        ON_COMMAND(ID_SWND_OUTW, OnSwndOutw)
        ON_COMMAND(ID_SWND_OUTDT, OnSwndOutdt)
        ON_COMMAND(ID_FILE_PRINT_COMENT, OnFilePrintComent)
        ON_COMMAND(ID_SWND_SECTION, OnSwndSection)
        ON_COMMAND(ID_SEC_MEM, OnSecMem)
        ON_COMMAND(ID_MEMO_SEC_CR, OnMemoSecCr)
        ON_COMMAND(ID_SWND_ISO, OnSwndIso)
        ON_WM_LBUTTONDBLCLK()
        //}}AFX_MSG_MAP
        // 標準印刷コマンド
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
        ON_MESSAGE(IDS_MESSAGE_WIND, OnMessageWind)
    END_MESSAGE_MAP()
    //    ――――――――――――――――――――――――――――――――
    //    ●    CSf31View クラスの構築
    //    ――――――――――――――――――――――――――――――――
    //
```

これらはプレゼンターの CSf31View クラスで、そのメッセージマップを表し、各種のメッセージに対応する。

```
CSf31View::CSf31View()
{
//      ●      ウインドウ管理データの初期設定
・
・
・
・
}
```

クラス CSf31View のコンストラクタである。

このように、イベント駆動型プログラムでは、ユーザーからのメッセージ、例えばマウスのボタンを押したり、キー入力したりすることで、あるいは OS からのメッセージでクラスの中の関数が実行される。このようにユーザーからの要求でプログラムが適切に動作し、情報が処理されるシステムであり、この積み重ねでイベント駆動型システムが構成されるわけである。各クラスには、上記のようなメッセージを受け取るメッセージマップが存在し、その部分によってメッセージが処理されるわけであるが、実際はもう少し複雑なメカニズムとなっている。このメカニズムの詳細については参考書を参照されたい。

先に示した CSf31View は、プレゼンターの子ウインドウを管理し、各種のグラフなどを表示するクラスである。プレゼンターが実際に起動されて子ウインドウが表示されると、このクラスから一つのオブジェクトが作り出される。上記の処理を実際に実行するのは、このオブジェクトということになる。このクラスはオブジェクトを生み出すための鋳型であり、子ウインドウが 2、3 個と表示されると、同じ処理内容を有するオブジェクトが 2、3 と増えていく。ただし、このオブジェクトが管理するメンバー変数は異なったメモリー領域にあり、異なった値を保持することになり、当然動作も異なったものとなる。

プレゼンターでは各種のグラフや表が描かれる。そのためウインドウの管理が必要となる。

このように、クラスを一つ用意するのみで、子ウインドウが増えても、それに対応してオブジェクトが増え、各々のウインドウ管理が可能となる。ただし、子ウインドウに何が表示されているか、あるいは、どのような状態となっているかなど、ウインドウ管理が必要となってくる。これについては、第３章で詳細に解説する。

1.3 プレゼンター開始と初期設定

プレゼンターが起動すると、図 1-1 に示すプレゼンターの主ウインドウに一つの子ウインドウが表示され、動作は停止した状態となる。この状態になる前に、多くの初期設定が実施される。ここでは、この初期設定の内容とそれがプログラムのどの部分で実際に処理されているかについて観察する。

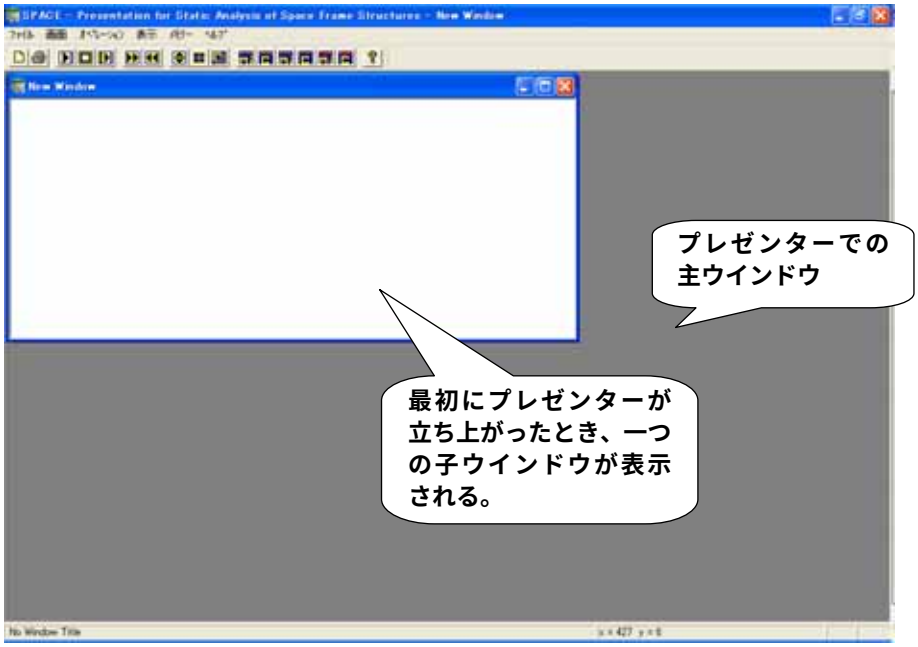

図 1-1 プレゼンター起動画面

図 1-1 が表示される前までに多くの関数が実行され、初期設定が行われる。その中で読者が知る必要のある関数は、以下の 2 つである。

```
InitInstance()
CMainFrame()
```

関数 InitInstance()は、ウインドウ生成に関連する処理を行う関数であり、Visual Studio　が自動的に生成する。一般的には、この関数を変更する必要はないが、ウインドウをカスタマイズする場合は、手を加える必要がある。その場合は、マニュアル等を良く理解した上で実行されたい。この関数では、主ウインドウの作成や、ツールバー、メニューの作成を定義する。以下に、この関数を示す。

```
////////////////////////////////////////////////////////////////////////////
// CSf31App クラスの初期化
////////////////////////////////////////////////////////////////////////////
BOOL CSf31App::InitInstance()
{
    // 標準的な初期化処理
    // もしこれらの機能を使用せず、実行ファイルのサイズを小さく
    // したければ以下の特定の初期化ルーチンの中から不必要なもの
    // を削除してください。

#ifdef _AFXDLL
    Enable3dControls(); // 共有 DLL の中で MFC を使用する場合にはここをコールしてください。
#else
    Enable3dControlsStatic();// MFC と静的にリンクしている場合にはここをコールしてください。
```

```
#endif

    LoadStdProfileSettings();  // 標準の INI ファイルのオプションをロードします (MRU を含む)
    // アプリケーション用のドキュメント テンプレートを登録します。ドキュメント テンプレート
    // はドキュメント、フレーム ウィンドウとビューを結合するために機能します。
    //CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_SF31TYPE,
        RUNTIME_CLASS(CSf31Doc),
        RUNTIME_CLASS(CChildFrame), // カスタム MDI 子フレーム
        RUNTIME_CLASS(CSf31View));
    AddDocTemplate(pDocTemplate);

// メイン MDI フレーム ウィンドウを作成
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

// DDE、file open など標準のシェル コマンドのコマンドラインを解析します。
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

// コマンドラインでディスパッチ コマンドを指定します。
    if (!ProcessShellCommand(cmdInfo))
      return FALSE;

// メイン ウィンドウが初期化されたので、表示と更新を行います。
    pMainFrame->ShowWindow(SW_SHOWMAXIMIZED);
    pMainFrame->UpdateWindow();
    return TRUE;
}
```

ウインドウの外観を変更する場合は、Visual Studio のマニュアルなどをよく読んで更新されたい。

次の関数 CMainFrame()は、このクラスのコンストラクタであり、プレゼンターシステムの初期設定を行う場所でもある。プレゼンターでは、ここで、多くの初期設定やグローバル変数の動的領域の確保、ファイルからのデータ入力、描画用ペンやフォントの設定などを行っている。このことについては、第３章でさらに詳しく説明する。

```
//     ――――――――――――――――――――――――――――――――
//     ●    CMainFrame クラスのコンストラクタ
//     ――――――――――――――――――――――――――――――――
//
CMainFrame::CMainFrame()
{
//――――――――――――――――――――――――――  ●
       int ihan,ndim,nnode;
       F_time_ii =0;
       F_Speed = 4;
        FI_Speed = 1;
       ihan=0;
```

```
        ndim=3;
        nnode=1;
        int xxfig[3];
        F_radio_color_pr =0;
//――――――――――――――――――――――――――――――  ●  ペン用のデータをセット
        create_penx();
//――――――――――――――――――――――――――――――  ●  システムデータのセット
        SYSNAM(&F_idfile[0],F_title);
//――――――――――――――――――――  ● 透視図用データのセット
        PERSCT(&xxfig[0],&xxfig[1],&xxfig[2],&F_scrnps[0],
               &F_viewps[0],&F_scalps,&F_scalep[0],&F_mag[0]);
        FF_scalps = F_scalps;
//――――――――――――――――――――  ● 動的用コントロールデータその１のセット
        DYCTL1(&ihan,&F_nindis,&F_gindis,&F_f1sec);
//――――――――――――――――――――  ● 動的用コントロールデータその２のセット
        DYCTL2(&ihan,&F_nstep,&F_delt,&F_igra[0],&F_ibata,
         &F_beta,&F_gamma,&F_xgal[0],&F_ntime,&F_dlamst);
//――――――――――――――――――――  ● 出力用データのセット
        DOUTCL(&ihan,&F_iwstp,&F_soutsc,&F_dmaxck);
//――――――――――――――――――――――――――――――  ●  配列データ設定用の値を計算
        ・
        ・
}
```

1.4 プレゼンターで使用しているテクニック

プレゼンターでは、多くのアニメーションや CG に関するテクニックを使用している。この本では、これらを例題にして基本的な技術を説明する。ここで学ぶ基本的な技術は、

１．図形処理のための基本的な手続き
２．図形の描き方（荷重履歴、地震波形、節点変位）
３．OnDraw 関数の使い方
４．図形の拡大・縮小、回転
５．マウスによる図形の位置取得
６．マルチウインドウの管理法
７．アニメーションの仕組み
８．ワイヤーフレームによる透視図の描き方
９．図形処理に関する各手法（ラバーバンド、プログレスバー、ポップアップメニュー）
10．書類出力の仕組み

などである。また、動的ソルバーで使用しているマルチスレッド技術についても学ぶ。

1.5 マルチ言語処理（FORTRAN と VC++)

SPACE システムでは､開発言語として FORTRAN と VC++を併用している。両言語間のインターフェイスやデータの交換について､簡単に復習しておこう。各々の言語仕様については、各自勉強されたい。

両言語間のインターフェイスには､特別なインターフェイス文が必要となる。ここでは、SPACE で良く使用する VC++で書かれたコードから FORTRAN で書かれたサブルーチンの呼び出し方法を示す。SPACE では、VC++のインクルードファイルとして fort.h に全て書き込むようになっている。以下に、fort.h を示す。

```
//    ――――――――――――――――――――――――――――――――
//    ●   include /fort.h
//    ――――――――――――――――――――――――――――――――
//    ●    FORTRAN 用の VC++の外部宣言(ok)
//    ――――――――――――――――――――――――――――――――
extern "C"{
void __stdcall SYSNAM(int*,char*);
void __stdcall INPTFX(int*,int*,int*,int*,int*,int*);
void __stdcall INPTFY(int*,int*,int*,int*,int*,float*,
                    int*,int*,int*,int*,int*,int*,float*,float*,
                    float*,int*,int*,float*,int*,float*,int*,int*,
                    float*,int*,int*,int*,int*,int*);
void __stdcall GET_S_COMP_MODEL_X(int*);
void __stdcall CHKFF(int*,int*,int*,float*,int*);
void __stdcall OUTFNO(int*,int*);
void __stdcall OUTFNX(int*,int*);
void __stdcall OUTFNY(int*,float*);
void __stdcall PERSCT(int*,int*,int*,float*,float*,float*,float*,float*);
void __stdcall TOSHIZ(float*,int*,float*,float*,int*,float*,float*,
                                          int*,int*,int*,float*);
void __stdcall TOSHPK(float*,int*,float*,float*,int*,float*,float*,
int*,int*,int*,float*,int*,int*,float*,int*,int*,int*);
void __stdcall TOSHIP(int*,float*,int*,float*,int*,float*,float*,int*,
                    float*,float*,int*,int*,int*,float*,float*,int*);
・
・

}
```

インクルード用ヘッダーファイル fort.h では、インターフェイスは、外部関数であることを示す extern “C”{　}で囲まれており、その中に関数名を書くことになる。標準的な書き方は以下のようである。

```
void __stdcall SYSNAM(int*,char*,int*);
```

関数名、つまり、FORTRAN のサブルーチン名あるいは関数名は大文字でなくてはならない。これは、言語仕様からくるもので、たとえプログラムの表記に小文字を含んでいても変数、関数、サブルーチン名等は、

FORTRAN では全て大文字とみなすからである。従って、大文字と小文字を区別する VC++では、FORTRAN のサブルーチンを呼ぶ場合は、大文字でなくてはならない。

例として動的ソルバーの主サブルーチンとなる SUBMAIN_DYNAMIC_A() を取り上げ、FORTRAN と VC++のコードを示す。まず、インターフェイス文は、

```
void __stdcall SUBMAIN_DYNAMIC_A(int*,int*,int*,int*,double*,double*,int*,int*,double*,int*,
  int*,float*,int*,float*,int*,float*,float*,float*,float*,int*,int*,int*,int*);
```

であり、この記述は fort.h に含まれている。次に、サブルーチンを呼ぶ方の VC++コードと、呼ばれる方の FORTRAN コードを以下に示す。

VC++で書かれたコード：

```
//    ――――――――――――――――――――――――――――――――
//      ●　時刻歴解析開始
//    ――――――――――――――――――――――――――――――――
//    ――――――――――――――――――――
//      ●　予備計算：シングルスレッド
//    ――――――――――――――――――――――
      SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
                        &nx_step,&ns_step,&d_max_v,&id_max_v,
                        &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
                         &F_read_spring,F_fay,F_n_spring,F_my_spring,
                        F_mz_spring,F_stat_spring,&n_iterate,
                       &nm_iterate,&numb_method);
```

FORTRAN で書かれたサブルーチン：

```
c――――――――――――――――――――――――――――――――――――――――
      subroutine submain_dynamic_a(i_calnum,iend_code,icontrol,ierr_dat,
     *           T,dt,n_step,ns_step,d_max_v,id_max_v,
     *           i_read_disp,F_disp,i_read_ndbalanceF,F_ndbalanceF,
     *           i_read_spring,F_fay,F_n_spring,F_my_spring,
     *           F_mz_spring,i_stat_spring,n_iterate,
     *           nm_iterate,numb_method)
```

上記のサブルーチンの括弧内は引数を表しており、当然、呼ぶ方と呼ばれる方、共に引数の型と数が一致しなければならない。また、VC++側における引数の前の記号 & は、アドレス渡しを意味しており、VC++と FORTRAN 間では、全てこのアドレス渡しとなっている。従って、VC++のコードには注意が必要であり、変数名の前に& 記号が必要となる。ただし、配列は VC++でもアドレス渡しであるため、& 記号を付けてはならない。例えば、F_disp、F_ndbalanceF などがそれに該当する。詳細は文法書を見られたい。

関数間のデータ渡しは、アドレス渡しとデータ渡しがある。一般にFORTRANはアドレス渡しであり、またVC++では、単独の変数はデータ渡しで、配列はアドレス渡しとなっている。アドレス渡しは、受け渡す関数に引数となっている記憶番地を渡すため、受け渡される関数内でデータの変更を行うと、受け渡す方の引数である変数の値も変更されてしまう。一方、データ渡しは、受け渡す方の引数に入っている値をコピーして、受け渡される関数の中の対応する変数にこの値をセットする。そのため、呼ばれる方でデータを変更したとしても、呼び出し側では、値は変更されていないことになる。このように、両者は記述の仕方も動作も異なっているので注意して使用する必要がある。

次に、両言語間で配列の記述を考えてみよう。両者では、配列の記述方法と記憶番地の順序が異なっている。例を用いて説明しよう。まず、２次元配列を以下に書いてみる。

```
FORTRAN：  A(10,10)
VC++    ：  A[10][10]
```

何も指定しないと、配列の記憶順は、FORTRANではA(1,1)から始まって、A(2,1)、A(3,1)となり、最後が、A(10,9)、A(10,10)の順番となる。一方、VC++では、A[0][0]、A[0][1]から始まって、最後がA[8][9]、A[9][9]となる。このように、文法の相違として、一般的にはFORTRANは１から始まるが、VC++は０から始まり、また、FORTRANは最も左から記憶番地が始まるが、VC++は外側から順次内側に変化する。例えば、A(5,6)はA[5][4]と同じ番地を表すことになる。両言語間でデータ交換を行う場合、これらのことを良く考慮して、データの受け渡しを行う必要がある。

以上で、FORTRANとVC++を併用する場合の最も基本的な注意点を述べた。後は、Visual C++とVisual FORTRANの文法書や(株)マイクロソフトのDeveloper Studioに関する文書を参照されたい。