ゲーム作りで学ぶ!
実践的C言語プログラミング

Nishio

はじめに

プログラミングには数学やアルゴリズムなどの知識は必要ない．このような話をよく耳にします．たしかにこれらの知識が無くても，プログラム（ゲーム，業務アプリケーション等）を作ることもできます．別に数学なんか一切できなくても（四則演算ができないとダメですが），アルゴリズムのリンクリストやらハッシュ等の知識なんてなくてもプログラムは組めるのです（まともに動くかどうかは別として）．

しかし，質の高いプログラムをつくるためには，数学的な論理的思考力やデータ構造などのアルゴリズムの知識はとても大切です．特にゲームプログラミングの分野においては数学的な知識を使わないと，思うようにゲームが作れないかもしれません．

例えば，あなたは今，シューティングゲーム作っているとします．自機が敵に向かって弾を発射したとしましょう．さて，弾が衝突したかどうかをどうやって判定するのでしょうか．この時利用するのが，円と円との衝突判定です．

具体的には，自弾を 円 C_1(中心点 (x_1, y_1), 半径 r_1) とし，敵機を 円 C_2(中心点 (x_2, y_2), 半径 r_2) とすると，$(x_2 - x_1)^2 + (y_2 - y_1)^2 \leq (r_1 + r_2)^2$ なんていう計算を行う必要があります．円と円だけでなく，もしかしたら点と円との衝突判定，四角形と円との衝突判定などが必要となるかもしれません．

アルゴリズムの知識が無いと，とんでもなく非効率なプログラムができあがってしまうかもしれません．特にゲームの分野では，1 秒間に 60 回 (60fps の場合）もの処理を高速で行う必要があります．たとえばシューティングゲームで，自機が 300 発もの弾を発射したとします．この弾を登録する時に，データ構造の知識（リンクリスト等）が必要となってくるかもしれません．

本書では，単にゲームの作り方を説明していくだけでなく，アルゴリズムや数学について，またこれらの知識をどういったときに使ったらよいか等を解説していきたいと思います．

本書の対象読者

本書は，C 言語の基礎を一通りマスターしたが，この先どのようなプログラムを作っていいかわからない方，ゲームの作り方を知りたい，アルゴリズムの知識を身につけたい等の方を対象としています．

本書を読むにあたっては基本的な C 言語の知識が必要です．基本的なとは，例えば if 構文の使い方が分かる，malloc 等を知っているくらいの知識で十分です．関数ポインタなどの難しい考え方は本書で丁寧に解説していくつもりです．

もし，C 言語の基礎から勉強したいのならば，著者の書いた C 言語プログラミング入門 (http://karetta.jp/book-cover/c-for-beginners) を是非読んでみてください．

DX ライブラリについて

C/C++言語は，Java 言語等と違い，標準で GUI を作成するためのライブラリは組み込まれてはいません．よって，企業や団体，個人が作成したライブラリを利用する事になります．C/C++言語で GUI を作成する為のライブラリ（GUI ツールキット）は数多く存在しています．Windows の場合，WindowsAPI や MFC 等が有名です．Linux の場合，Gtk や Qt 等が使われることが多いです．しかし，これらのライブラリはゲームを作成するには向いていません．ゲーム作成で最もよく使われるライブラリは OpenGL や DirectX 等でしょう．

OpenGL は Windows だけでなく，Linux や Unix，ゲーム機の PS2 等にも使われています．しかし，OpenGL はグラフィック専用のライブラリなので，グラフィック以外の機能（例えばサウンド再生，通信機能，入力関係等）は提供していません．

対して，DirectX は Windows（又はゲーム機の XBox 等）でしか利用できませんが，グラフィック処理だけでなく，サウンド再生，通信機能等，ゲーム作りには欠かせない機能が組み込まれています．Windows でゲームを作る際，一番多く採用されているライブラリでしょう．

しかし，この DirectX，機能は豊富なのですが，使い方がとても難しいです．覚えることが多く，ちょっとした処理をするにも一苦労です．著者もいままで DirectX を使ってゲームを作ろうと（何度も）企てていましたが，なにせ初期化だけでも 100 行近くのコードを書かねばならず，挫折してしまったものです．だんだんゲーム作りがいやにもなってきていました．

そんな時偶然見つけたのが山田 巧氏が作成した DX ライブラリでした．DirectX のラッパークラスである DX ライブラリですが，面倒な処理はすべて DX ライブラリが受け持ってくれます．例えば，初期化処理は DxLib_Init 関数を呼び出すだけで完了です．画像を表示する，音楽を再生する等の処理も 1 つの関数を呼び出すだけでいいのです．これには驚きました．いままでいくつかの DirectX のラッパーライブラリを使ってきましたが，DX ライブラリが一番直感的で，使いやすいと感じています．本書では，山田氏の作成した DX ライブラリを利用して，ゲームプログラミングの解説を行って行きたいと思います．

謝辞

著者は絵を書くことができません．音楽等についてのセンスもありません．しかし，画像等の素材が無くてはゲームとして成り立ちません（一部例外もありますが）．

ネット上にはゲーム素材をなんと無料で提供してくださる方がたくさんいます．著者のようにゲームは作りたいが，素材がないものにとっては，これは大変ありがたいことです．本書でゲームを作成するにあたっては，以下の方々の素材を利用させていただきました．このようなすばらしい素材を提供してくださった方々に深く感謝申し上げます．

ノベルゲーム作成に使用した素材の作者様

背景画像はぐったりにゃんこ様（http://guttari8.web.infoseek.co.jp/) の素材を利用させていただきました.

メッセージボックスはひだちー素材館様（http://www.vita-chi.net/sozai1.htm) の素材を利用させていただきました.

人物の立ち絵は蓮太郎様（http://sky.geocities.jp/dsrmg137/index.html）の素材を利用させていただきました.

シューティングゲーム作成に使用した素材の作者様

敵グラフィックは，野プリン様 (http://wild-pd.hp.infoseek.co.jp/) のものを利用させていただきました.

画面右側背景はシュガーポット様（ http://sugarpot2.fc2web.com/sozai/in_sozai.htm) のものを利用させていただきました.

背景画像は伊嘉智情大様 (http://www.ikachi.org/graphic/index.html) のものを利用させていただきました.

戦闘機画像は夢蒼/musou 様 (http://game.yu-nagi.com/) のものを利用させていただきました.

爆発画像は MIA 様の発色弾 (http://taillove.jp/mia/#) を利用させていただきました.

目 次

第I部

DXライブラリの基礎

第1章　開発環境を整える

この章では，DX ライブラリを利用するための開発環境を整えていきます．なお，C/C++ コンパイラは Visual C++ 2008 Express Edition を利用します．このコンパイラはインストール済みである事が前提となります．

1.1　DX ライブラリのダウンロード

まずは，DX ライブラリをダウンロードしましょう．以下のページから最新版を入手することができます．

```
http://homepage2.nifty.com/natupaji/DxLib/dxdload.html
```

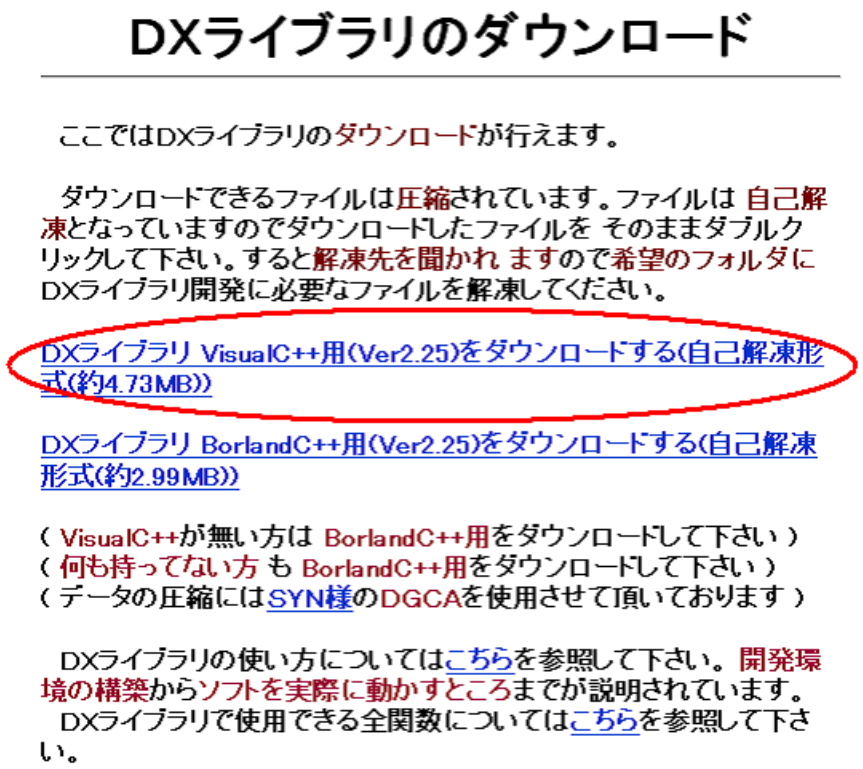

図 1.1: DX ライブラリのダウンロード

今回は VisualC++ Express Edition 2008 をコンパイラとして使用することとします．よって，DX ライブラリは VisualC++用をダウンロードしてください（図 1.1)．ダウンロードする場所は，仮に C:\Game\としましょう．なお，VisualC++2008 はインストール済みであることが前提です．もし，持っていないのなら，

```
http://www.microsoft.com/japan/msdn/vstudio/express/
```

から無償でダウンロード可能なので，インストールしておいてください．

ダウンロードが完了したら，DX ライブラリを展開してください．図 1.2 のように展開されれば成功です．

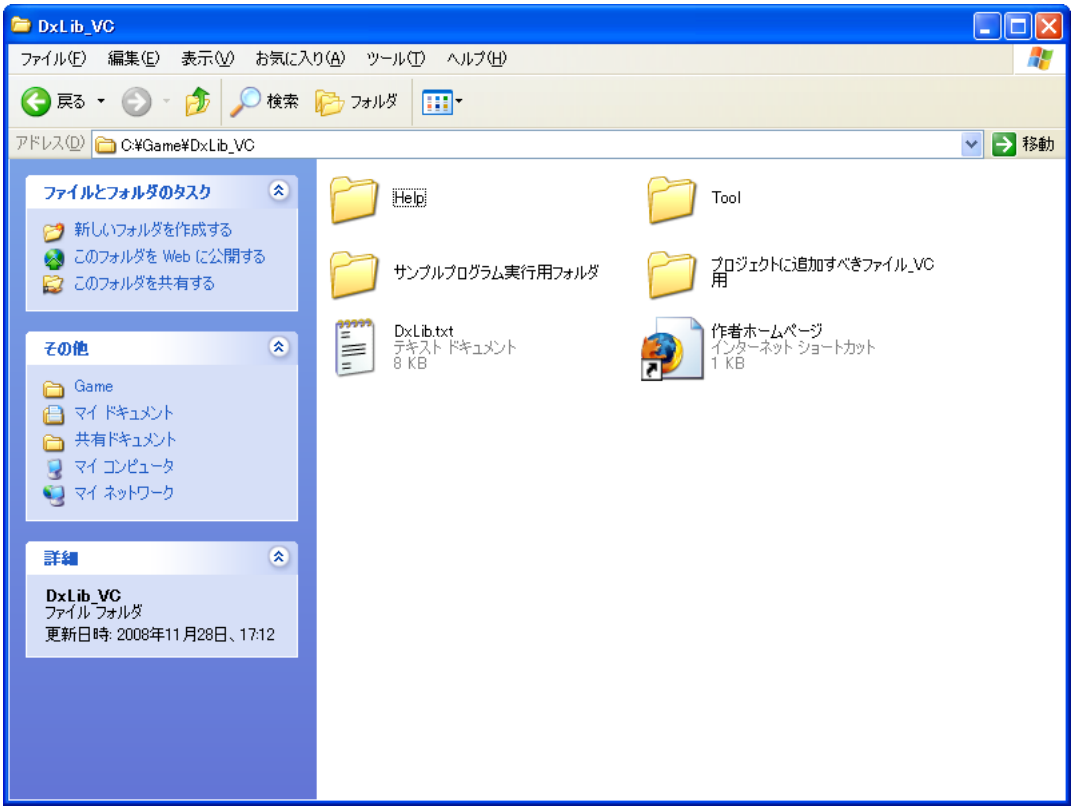

図 1.2: DX ライブラリの展開

1.2 VC++プロジェクトの環境設定

DX ライブラリを利用したプログラムをコンパイルするための VC++プロジェクトを作成してみましょう．まずは，Visual C++ 2008 を起動してください．

次に，メニューバーのファイル → 新規作成 → プロジェクトをクリックしてください (図 1.3）.

図 1.3: プロジェクトの作成

空のプロジェクトを選択してください．プロジェクトの種類，全般にあります（図 1.4）．インストール場所は，C:\Game とします．プロジェクト名は何でもよいですが，ここでは DxlibTest としましょう．

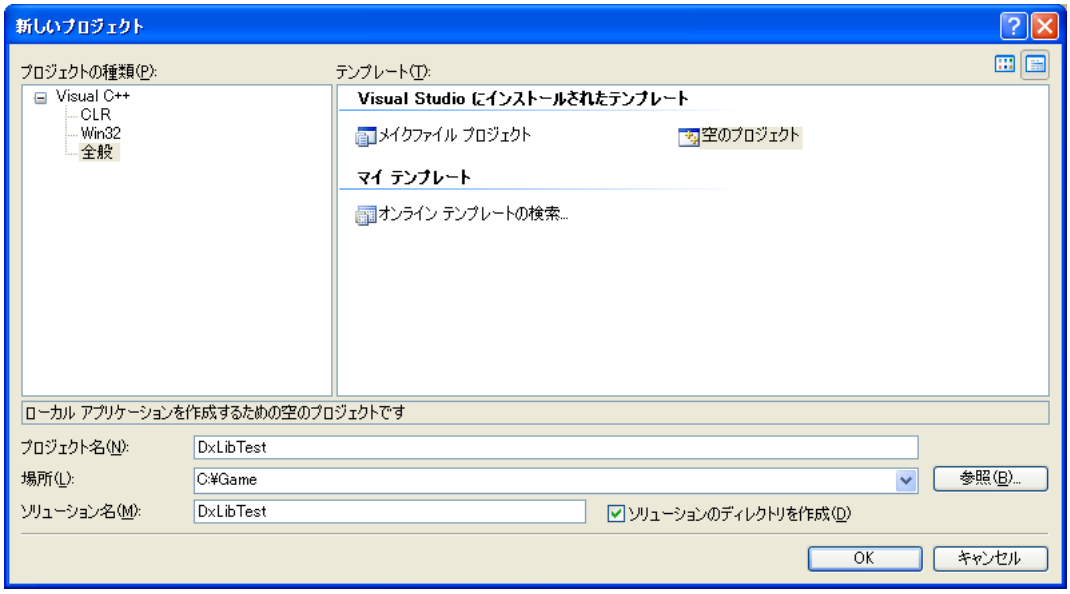

図 1.4: 空のプロジェクトの作成

プロジェクト作成完了後，ソリューションエクスプローラ内にある，ソースファイルフォルダを右クリックし，新しい項目をクリックします (図 1.5）．

C++ファイルを選択し，追加をクリックしてください（図 1.6）．ファイル名は何でもよいですが，ここでは test としましょう．

次に，DX ライブラリを利用するための設定を行います．メニューバーのツール → オプションを選択してください（図 1.7）．

オプションのプロジェクトおよびソリューションにある，VC++ディレクトリを選択します．そして，ディレクトリを表示するプロジェクトからインクルードファイルを選択し

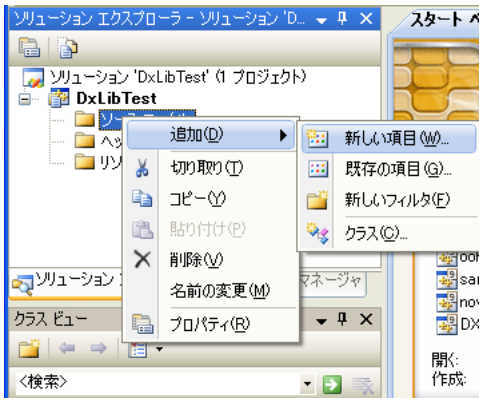

図 1.5: 新しい項目の追加

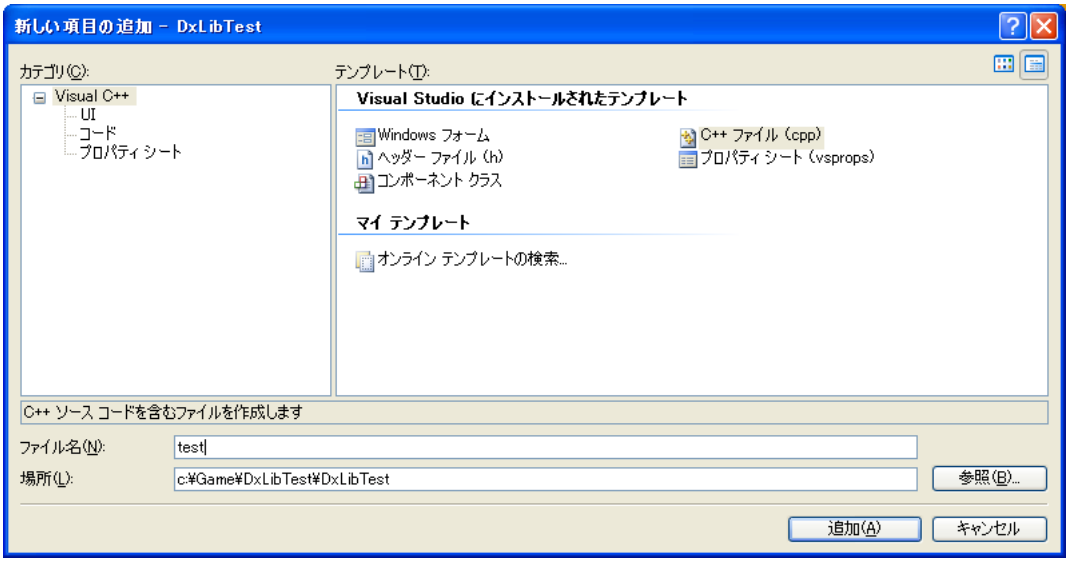

図 1.6: cpp ファイルの追加

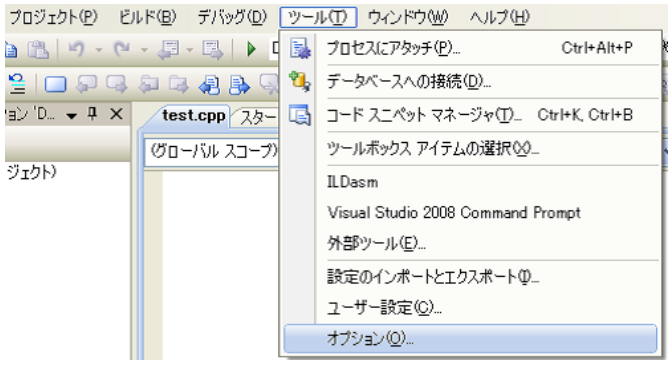

図 1.7: オプションの設定

てください（図 1.8）.

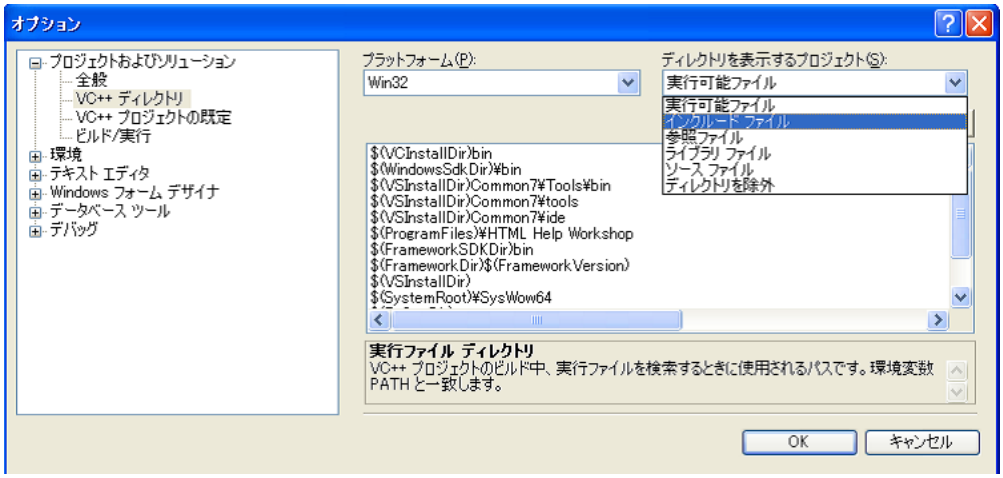

図 1.8: インクルードファイルの設定

図 1.9 の赤丸で囲ったフォルダマークをクリックします．すると，空白のテキストボックスの右側にボタンがでてきます．そのボタンをクリックし，ディレクトリの選択を行います．ディレクトリは，先ほど展開した DXLib フォルダの「プロジェクトに追加すべきファイル_VC 用」を選択してください．

同様に，今度はディレクトリを表示するプロジェクトのライブラリファイルを選択し，ディレクトリの選択を行ってください．選択するディレクトリは先ほどと同じ「プロジェクトに追加すべきファイル_VC 用」です．図 1.10 のように設定できれば完了です．

次に，メニューバーのプロジェクト → DxLibTest のプロパティをクリックしてください（図 1.11）.

構成プロパティ内の C/C++ → コード生成を選択します．そして，ランタイムライブラリを**マルチスレッドデバッグ DLL** から**マルチスレッドデバッグ**に変更してください（図 1.12）.

以上で VC++の環境設定は完了です.

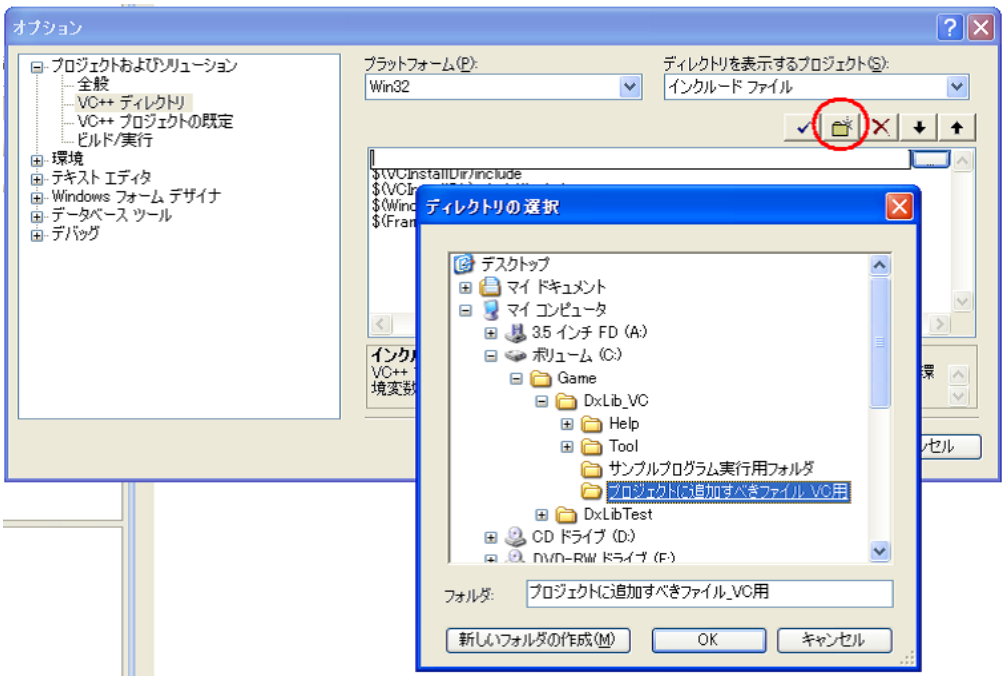

図 1.9: インクルードファイルの設定

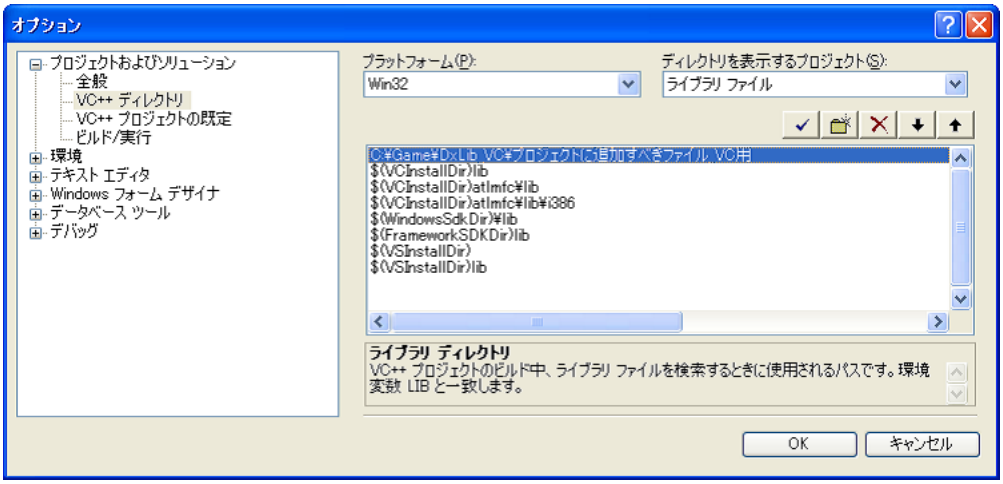

図 1.10: ライブラリファイルの設定

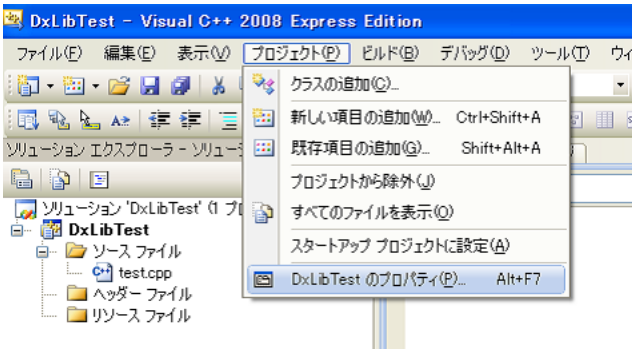

図 1.11: プロジェクトプロパティの設定

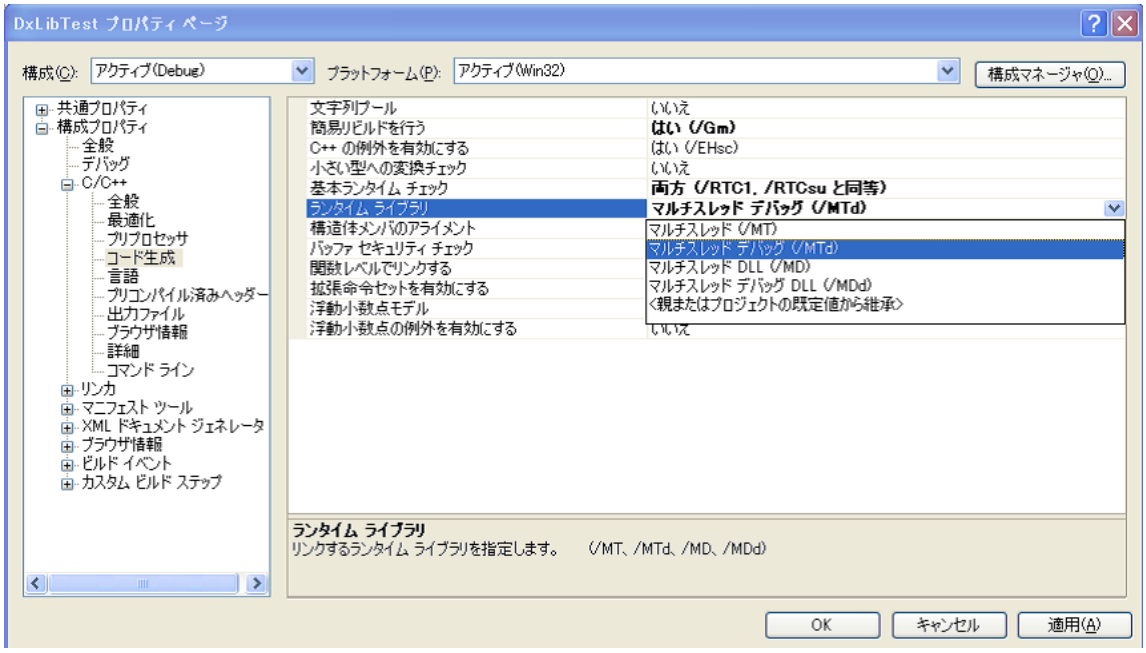

図 1.12: マルチスレッドデバッグの設定

1.3 サンプルプログラムの実行

正しく開発環境が整えられたかどうかを確かめましょう．以下のソースコードを test.cpp 内に書き込んでください．

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4                          LPSTR lpCmdLine, int nCmdShow )
5  {
6      if( DxLib_Init() == -1 ) {
7          return -1;
8      }
9      DrawString(50, 50, "Hello DX ライブラリ", GetColor(255,255,255) );
10     WaitKey();
11     DxLib_End();
12
13     return 0;
14 }
```

プロジェクトをコンパイルし，実行してみてください．画面に Hello DX ライブラリと表示されれば完了です．

第2章　文字列描画の基礎

この章では，文字列描画関係の関数について学んでいきます．

2.1　画面に文字列を表示する

Hello World プログラムを作成してみる事にしましょう．Hello World プログラムとは，画面にただ Hello World と表示するだけのプログラムです．まずは，プログラムをお見せします．

リスト 2.1: ”foundation-01.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //文字列表示
  DrawString( 20, 20, "Hello DX Library!", GetColor(255, 255, 255));

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

上記のプログラムは，座標 (20, 20) に Hello DX Library!と白文字で表示するプログラムです（図 2.1）．

図 2.1: Hello World プログラム

見慣れない関数がずらりと並んでいます．先頭から順番に説明していきたいと思います．

1 行目では，DxLib.h のインクルードを行っています．これは，DX ライブラリ特有の関数を利用する場合は必ず必要です．

次に，以下のコードをご覧下さい．

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4                                          LPSTR lpCmdLine, int nCmdShow )
5  {
6          //ウィンドウモードで起動
7          ChangeWindowMode( TRUE );
8          //DX ライブラリ初期化
9          if( DxLib_Init() == -1 ) {
10                 return -1;
11         }
12
13         //ここにプログラムを書いていく
14
15         //DX ライブラリ終了処理
16         DxLib_End();
17         return 0;
18 }
```

上記のコードは覚える必要はありません．「ここにプログラムを書いていく」という部分にプログラムを書いていくということを覚えて置いてください．このテンプレートは毎回利用します．

一応少し解説をしておきます．WinMain 関数ですが，これはプログラムが実行された際に，一番初めに呼び出される関数です．コンソールアプリケーションの場合，一番初めに呼び出される関数は main ですが，Windows アプリケーションの場合，WinMain 関数が呼び出されます．これは，DX ライブラリに限った話ではなく，Windows アプリケーションは，必ず WinMain 関数から始まります．

次に，ChangeWindowMode 関数ですが，これはフルスクリーンかウィンドウモードかを切り替えるための関数です．DX ライブラリは，標準ではフルスクリーンで起動されます．しかし，フルスクリーンで起動されるのは何かと嫌なものなので，ChangeWindowMode 関数を利用してウィンドウモードとしています．引数には TRUE 又は FALSE を指定できます．TRUE にするとウィンドウモードで起動されます．

DxLib_Init 関数は，DX ライブラリの初期化を行うプログラムです．この関数は，初期化に失敗した際，戻り値として-1 を返します．よって，何らかのトラブルが発生して初期化に失敗し，-1 が帰ってきた場合，プログラムを終了させる必要があります．

DxLib_End 関数は，DX ライブラリを終了する際に利用します．この関数は，メモリ領域の開放や，ウィンドウを閉じる等の処理を行ってくれます．

では，話を戻して，DrawString 関数の説明に入ります．DrawString 関数は文字列を画面に描画する際に利用します．関数の定義は次のようになっています．

```
int DrawString( int x , int y , char *String , int Color );
```

第 1, 第 2 引数は，文字を描画する座標を指定します．座標は画面左上を (0, 0) としています（図 2.2).

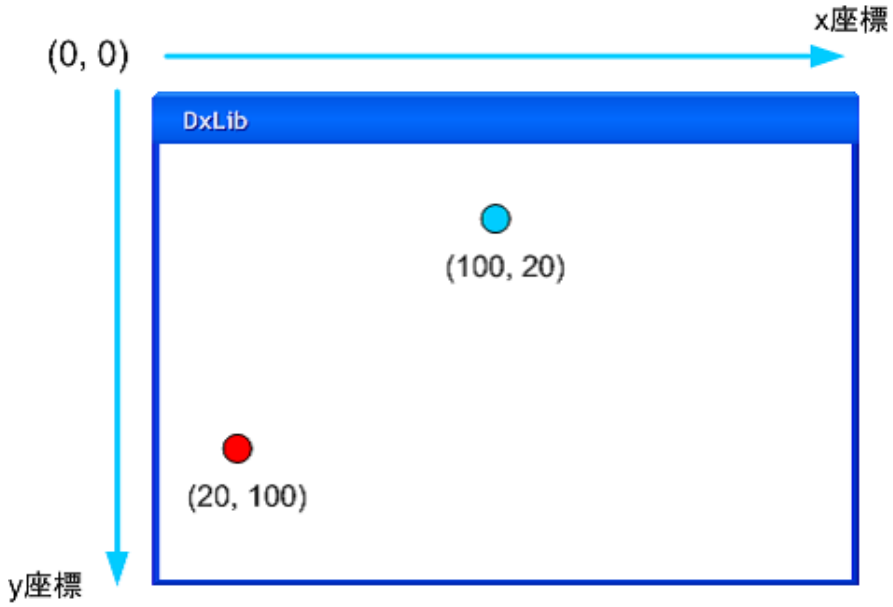

図 2.2: 座標

DrawString 関数の第 3 引数には描画したい文字列を，第 4 引数には文字列の色を指定します．

ここで，第 4 引数に色コードを渡すのですが，その際利用するのが GetColor 関数です．GetColor 関数の定義は次のようになっています．

```
int GetColor( int Red , int Green , int Blue );
```

Red, Green, Blue には取得したい色の輝度値を指定します．これらの要素の限界値は 255 です．例えば，赤色コードを取得したい場合，

```
GetColor(255, 0, 0)
```

とすればよいわけです．今回は白色を取得したいので，

```
GetColor(255, 255, 255)
```

としています．

WaitKey 関数ですが，これはキー（又はマウス）の入力待ちを行う際に利用します．

2.2 書式付き文字列の描画

DX ライブラリには書式付の文字列を表示できる DrawFormatString 関数が用意されています．これは，printf 関数と同じように使うことができる関数です．この関数を利用して，画面に文字を表示させてみましょう．

リスト 2.2: ”foundation-02.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4             LPSTR lpCmdLine, int nCmdShow )
5  {
6    char message[] = "Hello world";
7    int hoge = 35;
8
9    //ウィンドウモードで起動
10   ChangeWindowMode( TRUE );
11   //DXライブラリ初期化
12   if( DxLib_Init() == -1 ) {
13     return -1;
14   }
15
16   //文字列表示
17   DrawFormatString( 20, 20, GetColor(255, 255, 255),
18     "message : %s -- value : %d", message, hoge );
19
20   //キー入力待ち
21   WaitKey();
22   //DXライブラリ終了処理
23   DxLib_End();
24   return 0;
25 }
```

このプログラムの実行結果は図 2.3 となります．

図 2.3: 書式付文字列の表示

今回利用した DrawFormatString 関数の定義は以下のようになっています．

```
int DrawFormatString( int x , int y , int Color ,
                      char *FormatString , ... );
```

第 1, 第 2 引数は文字列を描画する座標を，第 3 引数には色コードを指定します．第 4 引数以降は，printf 関数と同じです．ただし，一部エスケープシークエンスを利用することができません．それは，\n や\t 等です．改行等の処理は自分で行う必要があります．

2.3 フォントサイズの変更

表示するフォントサイズを変更する事もできます．その際に利用するのが，SetFontSize 関数です．

リスト 2.3: ”foundation-03.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //フォントサイズを25にする
  SetFontSize( 25 );

  //文字列表示
  DrawString(20, 20, "Hello DX library!", GetColor(255, 255, 255) );

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は図 2.4 となります．

図 2.4: フォントサイズの変更

今回利用した SetFontSize 関数の定義は以下のようになっています．

```
int SetFontSize( int FontSize );
```

引数にはフォントの大きさを指定します．

2.4 フォントの変更

フォントを変更することもできます．利用するのは，ChangeFont 関数です．今回は，フォントを MS 明朝に変えて文字列を描画したいと思います．

リスト 2.4: ”foundation-04.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //白色
  int white = GetColor(255, 255, 255);

  //文字列表示
  DrawString(20, 20, "はろーわーるど", white );

  //フォントをMS明朝に変える
  ChangeFont("ＭＳ 明朝");

  //明朝文字で文字列描画
  DrawString(20, 50, "はろーわーるど", white );

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は図 2.5 となります.

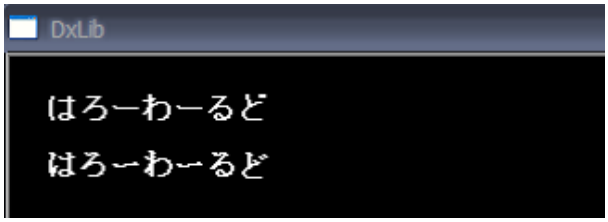

図 2.5: フォントの変更

今回利用した ChangeFont 関数の定義は以下のようになっています.

```
int ChangeFont( char *FontName );
```

引数にはフォントの名前を指定します.

2.5 コンソールへの文字列出力

複雑なプログラムを作成していると，デバッグ用メッセージを表示したくなる場合があります．しかし，DX ライブラリを利用している場合，コンソール画面が起動されていな

いので，printf 関数を利用することができません．そこで，今回はコンソールを起動する方法をお見せします．

リスト 2.5: ”foundation-05.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //デバッグ用にコンソールを呼び出す
  AllocConsole();
  freopen("CONOUT$", "w", stdout);
  freopen("CONIN$", "r", stdin);

  //printf関数を利用
  printf("Hello world!");

  //キー入力待ち
  WaitKey();
  //コンソール解放
  FreeConsole();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は図 2.6 となります．

図 2.6: コンソールの呼び出し

コンソールを呼び出すには，AllocConsole 関数を利用します．これは，DX ライブラリの関数ではなく，Windows 用の関数です．しかし，ただコンソールを呼び出しただけでは，画面に文字を表示することができません．そこで，標準入力，標準出力を freopen 関数を利用してコンソールへと割り当てています．これらの関数の意味は理解しなくてかまいません．とにかく，

```
AllocConsole();
```

```
2   freopen("CONOUT$", "w", stdout);
3   freopen("CONIN$", "r", stdin);
```

とすれば，コンソールが呼び出せるということを覚えておいてください．また，コンソールを使い終わったら，FreeConsole 関数を利用して，コンソールを解放してやる必要があります.

第3章　画像データ制御の基礎

この章では，画像描画関係の関数について学んでいきます．

3.1　画像ファイルの表示

画像を画面に表示するプログラムを作成しましょう．今回表示させる画像は図 3.1 です．まずは，作成したプログラムをお見せします．

図 3.1: カエル (蓮太郎氏作)

リスト 3.1: ”foundation-06.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //画像描画
  LoadGraphScreen( 20, 20, "kaeru1.png", TRUE );

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

上記のプログラムの実行結果は図 3.2 となります．

画像を表示させるために，LoadGraphScreen 関数を利用しました．この関数の定義は以下のようになっています．

```
int LoadGraphScreen( int x , int y , char *GraphName , int TransFlag );
```

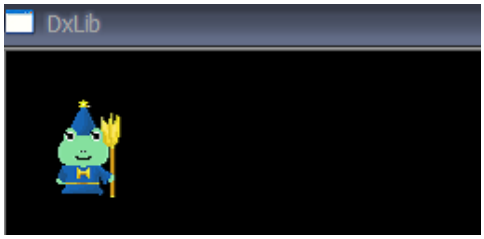

図 3.2: 画像の表示

第 1，第 2 引数には画像を表示する座標を指定します．第 3 引数は，表示したい画像のパスを指定します．ここで，kaeru1.png を読み込みたい場合は，kaeru1.png を Visual C++ プロジェクトがある場所と同じフォルダに置いてください（図 3.3).

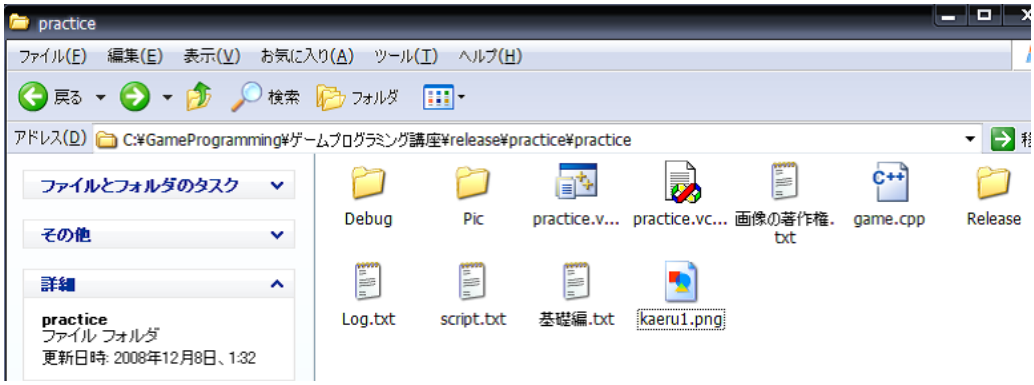

図 3.3: 画像ファイルの保存場所

また，例えばプロジェクトがある下のフォルダ，Pic に保存されている kaeru1.png を読み込みたい場合は，`"Pic/kaeru1.png"` 又は `"./Pic/kaeru1.png"` 等と指定します．

LoadGraphScreen 関数が読み込むことができる画像ファイル形式は，BMP, JPEG, PNG, DDS, ARGB, TGA の 6 種類です. GIF 画像等，読み込めない画像形式に注意してください．

第 4 引数には透過色を入れるか入れないかを指定します．TRUE を指定することで，透過色が有効となります．この第 4 引数に関しては，次節で詳しく説明します．

3.2 画像の透過処理

LoadGraphScreen 関数の第 4 引数 (TransFlag) を TRUE とすることで，画像の透過処理を行うこともできます．透過処理とは，画像の一部分を透明化させることです．まずは，透過処理の例をお見せします．今回は背景画像 back.png とカエルの画像 kaeru1.png を利用しています．

リスト 3.2: ”foundation-07.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //背景描画
  LoadGraphScreen( 0, 0, "back.png", FALSE );
  //画像描画 透過処理無し
  LoadGraphScreen( 20, 20, "kaeru1.png", FALSE );
  //画像描画 透過処理有り
  LoadGraphScreen( 100, 20, "kaeru1.png", TRUE );

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は，図 3.4 となります.

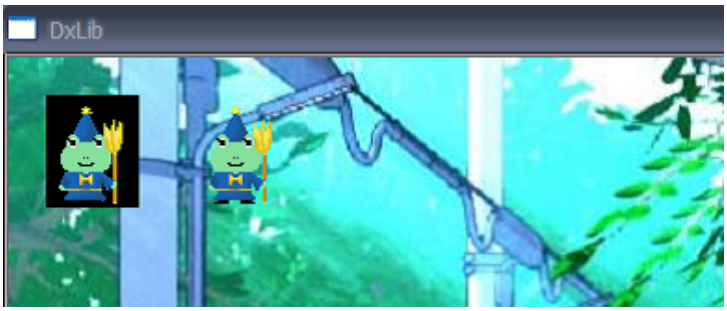

図 3.4: 透過処理の例

図 3.4 の左側のカエルは透過処理を行っておらず，右側のカエルは透過処理をしています．透過処理をするためには，TransFlag を TRUE とすることで，画像ファイルの限りなく黒色に近い部分，又はあらかじめ透過処理されている部分を透明色とします.

透過処理の対象となる色は，デフォルトでは黒となっています．しかし，黒色を透過して欲しくない場合もあります．そういった時には，透過色を設定する SetTransColor 関数を利用します.

```
int SetTransColor( int Red , int Green , int Blue );
```

例えば，赤色を透過色としたい場合は，`SetTransColor(255, 0, 0);` とします.

3.3 メモリ上にある画像を表示

メモリ上に画像をあらかじめロードしておき，その後画面に表示させてみましょう．通常は，画像を画面に表示させる際，LoadGraphScreen 関数は利用しません．それは，LoadGraphScreen 関数は呼び出された時点で画像ファイルをディスクから読み込んでくるからです．ハードディスクからファイルを読み込むのは，メモリからの読み込みと比べかなり処理に時間がかかってしまうものです．そこで，画像をメモリ上に読み込んでおき，そこから必要なときに画面に表示させるといった方法が取られます．

メモリ上に画像を読み込むには，LoadGraph 関数を利用します．

```
int LoadGraph( char *FileName );
```

FileName には読み込みたい画像のパスを指定します．この関数は画像の読み込みに成功した際，戻り値にグラフィックハンドルと呼ばれる画像識別番号が返ってきます（画像読み込み失敗時には-1 が返ってきます）．この識別番号を DrawGraph 関数に渡してやることで画像を表示することができます．

DrawGraph 関数の定義は次のようになっています．

```
int DrawGraph( int x, int y, int GrHandle, int TransFlag );
```

使い方は LoadGraphScreen 関数とほぼ同じです．第 3 引数にグラフィックハンドルを渡すことで，画像を表示させることができます．

では，LoadGraph 関数及び DrawGraph 関数を利用したプログラムをお見せします．

リスト 3.3: ”foundation-08.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //画像グラフィックハンドル
  int gHandle;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //画像をメモリに読み込む
  gHandle = LoadGraph("kaeru1.png");
  //メモリ上の画像を描画
  DrawGraph( 20, 20, gHandle, TRUE );
  DrawGraph( 80, 20, gHandle, TRUE );

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は図 3.5 となります．

図 3.5: メモリ上に読み込んだ画像の表示

3.4 Windows からのメッセージ処理

次のようなプログラムを書いてみました．これは，一定時間画像を表示するプログラムです．

リスト 3.4: ”foundation-09.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  //画像グラフィックハンドル
  int gHandle;
  int i;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //画像をメモリに読み込む
  gHandle = LoadGraph("kaeru1.png");

  //Windowsからの要求を受け付けない
  for(i = 0; i < 50; i++ ) {
    //メモリ上の画像を描画
    DrawGraph( 20, 20, gHandle, TRUE );
    DrawGraph( 80, 20, gHandle, TRUE );

    //100ms待つ
    Sleep( 100 );
    //画面上に描かれたものを削除
    ClearDrawScreen();
  }

  //キー入力待ち
  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

時間待ちには Sleep 関数を利用しています．この関数には，引数として待ち時間 (ms 単位）を指定します．例えば，0.1 秒 = 100 ミリ秒の時間待ちをしたい場合は，100 を渡します．また，ClearDrawScreen 関数は，画面に描画されたものを削除するための関数です．

さて，一見何も問題が無い様に見えますが，実行するとやや違和感を感じるかもしれません．たしかに画像は表示されているのですが，ウィンドウの移動や×ボタンを押しての終了等ができません．なぜでしょうか．

実は，これは Windows 側から送られてくるメッセージの処理をしていないからなのです．Windows は，様々なメッセージをプログラムに送っているのです．例えば，画面を移動してくれ，プログラムを終了してくれ等のメッセージです．このようなメッセージを処理しないと，ゲームが不安定になってしまいます．

通常の Windows プログラミングにおいては，この処理は自分で書く必要があるのですが，DX ライブラリでは，そのような面倒なメッセージ処理を肩代わりしてくれる関数が存在しています．それが，ProcessMessage 関数です．上記のプログラムのように，長い間ループ内で処理をしている時には，ProcessMessage 関数を呼びだして，メッセージ処理をする必要があります．

ProcessMessage の定義は次のようになっています．

```
int ProcessMessage( void );
```

この関数は，エラーが発生した際には，戻り値として-1 を返してきます．よって，エラー発生時には，直ちにプログラムを終了させてやる必要があります．

では，先ほどのプログラムのループ内に ProcessMessage 関数を入れてみましょう．

リスト 3.5: ”foundation-10.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4             LPSTR lpCmdLine, int nCmdShow )
5  {
6    //画像グラフィックハンドル
7    int gHandle;
8    int i;
9
10   //ウィンドウモードで起動
11   ChangeWindowMode( TRUE );
12   //DXライブラリ初期化
13   if( DxLib_Init() == -1 ) {
14     return -1;
15   }
16
17   //画像をメモリに読み込む
18   gHandle = LoadGraph("kaeru1.png");
19
20   for(i = 0; i < 50; i++ ) {
21     //メモリ上の画像を描画
22     DrawGraph( 20, 20, gHandle, TRUE );
23     DrawGraph( 80, 20, gHandle, TRUE );
24
25     //Windowsからのメッセージを処理
26     if( ProcessMessage() == -1 ) {
27       break;
28     }
29
30     //100ms待つ
31     Sleep( 100 );
32     //画面上に描かれたものを削除
33     ClearDrawScreen();
34   }
```

```
35
36    //キー入力待ち
37    WaitKey();
38    //DXライブラリ終了処理
39    DxLib_End();
40    return 0;
41  }
```

今度はどうでしょうか．ウィンドウを動かしたり，×ボタンを押してプログラムの終了ができるはずです．

3.5 ダブルバッファリング

今回は画像を動かしてみたいと思います．次のようなプログラムを書いてみました．

リスト 3.6: ”foundation-11.cpp”

```
1   #include "DxLib.h"
2
3   int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4               LPSTR lpCmdLine, int nCmdShow )
5   {
6     //画像グラフィックハンドル
7     int gHandle;
8     int i;
9
10    //ウィンドウモードで起動
11    ChangeWindowMode( TRUE );
12    //DXライブラリ初期化
13    if( DxLib_Init() == -1 ) {
14      return -1;
15    }
16
17    //画像をメモリに読み込む
18    gHandle = LoadGraph("kaeru1.png");
19
20    for(i = 0; i < 500; i++ ) {
21      //メモリ上の画像を描画
22      DrawGraph( i, 20, gHandle, TRUE );
23
24      //Windowsからのメッセージを処理
25      if( ProcessMessage() == -1 ) {
26        break;
27      }
28
29      //10ms待つ
30      Sleep( 10 );
31      //画面上に描かれたものを削除
32      ClearDrawScreen();
33    }
34
35    //キー入力待ち
36    WaitKey();
37    //DXライブラリ終了処理
38    DxLib_End();
39    return 0;
40  }
```

画面上を画像が動いていますが，よくよく見ると画像にちらつきが生じています．これは，画像を表示途中の画面が表示されているためです．これを解消するために，ダブルバッファリングと呼ばれるちらつき回避の技術が必要となります．

ダブルバッファリングは画面を 2 つ利用します．一つは実際に表示する表画面（フロントバッファ）と，計算途中の画像を書いておく裏画面（バックバッファ）です．裏画面で画像描画等の処理を行い，すべての処理が完了したら，裏画面と表画面を入れ替える（スワッピング）処理を行います．これにより，画面の変更が一度に行われるため，見た目にはスムーズに表示されるわけです．

さて，DX ライブラリでダブルバッファリングを行うのは簡単です．まずは，描画先の画面を変更します．通常は表画面に直接描画されますが，ダブルバッファリングを利用するために，描画先を裏画面に変える必要があります．その際利用するのが SetDrawScreen 関数です．

```
int SetDrawScreen( int DrawScreen );
```

DrawScreen には描画先の画面を指定します．描画先を表画面とするには，DX_SCREEN_FRONT を，裏画面とするには DX_SCREEN_BACK を指定します．

裏画面の内容を表画面に反映させるには，ScreenFlip 関数を利用します．

```
int ScreenFlip( void );
```

注意点として，**ScreenFlip 関数を呼び出した後は，ClearDrawScreen 関数を呼び出して裏画面の情報を削除**してください．

では，先ほどのプログラムにダブルバッファリング処理を加えてみましょう．

リスト 3.7: ”foundation-12.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4             LPSTR lpCmdLine, int nCmdShow )
5  {
6    //画像グラフィックハンドル
7    int gHandle;
8    int i;
9
10   //ウィンドウモードで起動
11   ChangeWindowMode( TRUE );
12   //DXライブラリ初期化
13   if( DxLib_Init() == -1 ) {
14     return -1;
15   }
16
17   //裏画面への描画をデフォルトとする
18   SetDrawScreen( DX_SCREEN_BACK );
19
20   //画像をメモリに読み込む
21   gHandle = LoadGraph("kaeru1.png");
22
23   for(i = 0; i < 500; i++ ) {
24     //メモリ上の画像を描画
25     DrawGraph( i, 20, gHandle, TRUE );
26
```

```
27      //Windowsからのメッセージを処理
28      if( ProcessMessage() == -1 ) {
29        break;
30      }
31      //裏画面反映
32      ScreenFlip();
33
34      //10ms待つ
35      Sleep( 10 );
36      //画面上に描かれたものを削除
37      ClearDrawScreen();
38    }
39
40    //キー入力待ち
41    WaitKey();
42    //DXライブラリ終了処理
43    DxLib_End();
44    return 0;
45  }
```

実行結果は図 3.6 となります．どうでしょうか．ちらつきが解消されているはずです．

図 3.6: ダブルバッファリング

第4章　入力関係処理の基礎

この章では，キーボードやマウスからの入力処理等の基礎について解説していきます.

4.1　特定のキーの入力状態の取得

特定のキーが押されているかを判定するには，CheckHitKey 関数を利用します.

```
int CheckHitKey( int KeyCode );
```

KeyCode には入力状態を取得したいキーコードを渡します（表 4.1）．例えば，A キーが押されているかを判定したい場合は，表 4.1 より KeyCode に KEY_INPUT_A を渡してやればよいわけです．また，この関数はキーが押されている場合には 1 を，押されていない場合は 0 を戻り値として返してきます.

では，A キーが押されるまで処理を続けるプログラムを作成してみましょう.

リスト 4.1: ”foundation-13.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  int white;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //白色取得
  white = GetColor( 255, 255, 255 );

  //Aキーが押されるまでプログラムを続行
  while( CheckHitKey( KEY_INPUT_A ) == 0 ) {
    if( ProcessMessage() == -1 ) {
      break;
    }
    DrawString( 20, 20, "Aキーでプログラム終了", white );
  }

  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

このプログラムの実行結果は，図 4.1 となります.

図 4.1: キー入力を受け付けるプログラム

キーコード	キー	キーコード	キー
KEY_INPUT_BACK	BackSpace	KEY_INPUT_RALT	右 ALT
KEY_INPUT_TAB	Tab	KEY_INPUT_SCROLL	ScrollLock
KEY_INPUT_RETURN	Enter	KEY_INPUT_SEMICOLON	；
KEY_INPUT_LSHIFT	左シフト	KEY_INPUT_COLON	：
KEY_INPUT_RSHIFT	右シフト	KEY_INPUT_LBRACKET	［
KEY_INPUT_LCONTROL	左コントロール	KEY_INPUT_RBRACKET	］
KEY_INPUT_RCONTROL	右コントロール	KEY_INPUT_AT	@
KEY_INPUT_ESCAPE	Escape	KEY_INPUT_BACKSLASH	＼
KEY_INPUT_SPACE	Space	KEY_INPUT_COMMA	,
KEY_INPUT_PGUP	PageUp	KEY_INPUT_MULTIPLY	テンキー＊
KEY_INPUT_PGDN	PageDown	KEY_INPUT_ADD	テンキー＋
KEY_INPUT_END	End	KEY_INPUT_SUBTRACT	テンキー－
KEY_INPUT_HOME	Home	KEY_INPUT_DECIMAL	テンキー.
KEY_INPUT_LEFT	左	KEY_INPUT_DIVIDE	テンキー／
KEY_INPUT_UP	上	KEY_INPUT_NUMPADENTER	テンキーの Enter
KEY_INPUT_RIGHT	右	KEY_INPUT_NUMPAD0	テンキー 0
KEY_INPUT_DOWN	下	…	
KEY_INPUT_INSERT	Insert	KEY_INPUT_NUMPAD9	テンキー [
KEY_INPUT_DELETE	Delete	KEY_INPUT_F1	F1
KEY_INPUT_MINUS	-	…	
KEY_INPUT_CAPSLOCK	CaspLock	KEY_INPUT_F12	F12
KEY_INPUT_PAUSE	PauseBreak	KEY_INPUT_A	A
KEY_INPUT_YEN	￥	…	
KEY_INPUT_PREVTRACK	^	KEY_INPUT_Z	Z
KEY_INPUT_PERIOD	.	KEY_INPUT_0	0
KEY_INPUT_SLASH	／	…	
KEY_INPUT_LALT	左 ALT	KEY_INPUT_9	9

表 4.1: キーとキーコードの関係

4.2 キーの入力状態の取得

前節で利用した CheckHitKey 関数は，一つのキーの状態を取得する為のものでした．しかし，複数のキーが押されているかを調べるために，何回も CheckHitKey 関数を呼び出すのは，処理の効率がよくありません．そこで，今回はすべてのキーの入力状態を一度に取得することができる CheckHitKeyStateAll 関数を利用してみます．

CheckHitKeyStateAll 関数は以下のように定義されています．

```
int GetHitKeyStateAll( char *KeyStateBuf );
```

KeyStateBuf には char 型配列（要素数 256 個）を渡します．要素数は多くても少なくてもいけません．これは DX ライブラリの仕様によるものです．

KeyStateBuf には，全てのキーの入力状態が格納されます．特定のキーが押されているかを調べるには，配列の要素番号をキーコードとして，値が 1 か 0 かをチェックすればよいわけです．

例えば A キーが押されているかを調べるには，以下のコードを利用します．

```
1  char keyStatus[256];
2  GetHitKeyStateAll( keyStatus );
3
4  if( keyStatus[ KEY_INPUT_A ] == 1 ) {
5          //A キーが押されている時の動作
6  }
```

では，今回は GetHitKeyStateAll 関数を利用して，A キーと B キーが同時に押された際に終了処理を行うプログラムを作成してみましょう．

リスト 4.2: ”foundation-14.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4             LPSTR lpCmdLine, int nCmdShow )
5  {
6    int white;
7    //キー状態
8    char keyStatus[ 256 ];
9
10   //ウィンドウモードで起動
11   ChangeWindowMode( TRUE );
12   //DXライブラリ初期化
13   if( DxLib_Init() == -1 ) {
14     return -1;
15   }
16
17   //白色取得
18   white = GetColor( 255, 255, 255 );
19
20   //A + B キーが押されるまでプログラムを続行
21   while( 1 ) {
22     if( ProcessMessage() == -1 ) {
23       break;
24     }
```

```
25      //キーの状態を取得
26      GetHitKeyStateAll( keyStatus );
27
28      if( keyStatus[ KEY_INPUT_A ] && keyStatus[ KEY_INPUT_B ] ) {
29        break;
30      }
31      DrawString( 20, 20, "A + Bキーでプログラム終了", white );
32    }
33
34    //DXライブラリ終了処理
35    DxLib_End();
36    return 0;
37  }
```

このプログラムの実行結果は図 4.2 となります.

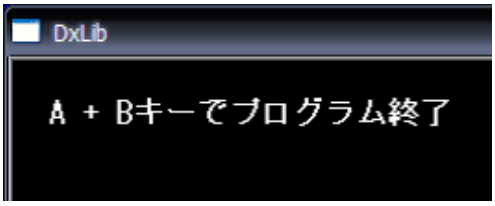

図 4.2: 複数のキー入力を受け付けるプログラム

4.3 マウス座標の取得

マウスカーソルの位置情報を取得するには，GetMousePoint 関数を利用します．この関数の定義は以下のようになっています.

```
int GetMousePoint( int *XBuf, int *YBuf );
```

XBuf, YBuf にはマウスカーソルの x 座標，y 座標が代入されます.

マウスカーソルの座標を取得する際，マウスカーソルが表示されていなければ意味がありません．そこで，SetMouseDispFlag 関数を利用して，マウス表示の有無を設定する必要があります.

```
int SetMouseDispFlag( int DispFlag );
```

DispFlag には TRUE 又は FALSE を渡します．TRUE でマウスカーソルが表示されます.

では，これらの関数を利用して，マウスの座標を取得するプログラムを作成してみましょう.

リスト 4.3: ”foundation-15.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4              LPSTR lpCmdLine, int nCmdShow )
5  {
```

```
6   int white, posX, posY;
7
8   //ウィンドウモードで起動
9   ChangeWindowMode( TRUE );
10  //DXライブラリ初期化
11  if( DxLib_Init() == -1 ) {
12    return -1;
13  }
14  SetDrawScreen( DX_SCREEN_BACK );
15
16  //マウスを表示状態にする
17  SetMouseDispFlag( TRUE );
18  //白色取得
19  white = GetColor( 255, 255, 255 );
20
21  //ESCAPEキーが押されるまでプログラムを続行
22  while( CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
23    ClearDrawScreen();
24    if( ProcessMessage() == -1 ) {
25      break;
26    }
27    //マウスポインタの座標を取得
28    GetMousePoint( &posX, &posY );
29    //座標を表示
30    DrawFormatString( 20, 20, white, "マウスの座標( %d, %d )", posX, posY );
31    //裏画面反映
32    ScreenFlip();
33  }
34
35  //DXライブラリ終了処理
36  DxLib_End();
37  return 0;
38 }
```

このプログラムの実行結果は，図 4.3 となります．

図 4.3: マウスの位置を表示するプログラム

4.4 マウスボタンの状態の取得

マウスのどのボタンが押されたかどうかを判定するには，GetMouseInput 関数を利用します．この関数の定義は以下のようになっています．

```
int GetMouseInput( void );
```

戻り値にはマウスの入力状態が入っています．どのボタンが押されているかは，表 4.2 の値との AND 演算を行い，結果が 0 でなければボタンが押されていると判定することができます．

定義値	マウスボタン
MOUSE_INPUT_LEFT	マウス左ボタン
MOUSE_INPUT_RIGHT	マウス右ボタン
MOUSE_INPUT_MIDDLE	マウス中央ボタン

表 4.2: マウスの定義値

例えば，マウス左ボタンが押されているかを調べるには，GetMouseInput 関数の戻り値と MOUSE_INPUT_LEFT との AND 演算を行えばよいわけです.

```
if( ( GetMouseInput() & MOUSE_INPUT_LEFT ) != 0 ) {
    //左クリック時の処理
}
```

では，前節のプログラムを改良して，左クリック時に文字列を表示するプログラムを作成してみましょう.

リスト 4.4: ”foundation-16.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  int white, posX, posY;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }
  SetDrawScreen( DX_SCREEN_BACK );

  //マウスを表示状態にする
  SetMouseDispFlag( TRUE );
  //白色取得
  white = GetColor( 255, 255, 255 );

  //ESCAPEキーが押されるまでプログラムを続行
  while( CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
    ClearDrawScreen();
    if( ProcessMessage() == -1 ) {
      break;
    }
    //マウスポインタの座標を取得
    GetMousePoint( &posX, &posY );
    //座標を表示
    DrawFormatString( 20, 20, white, "マウスの座標( %d, %d )", posX, posY );
    //マウス左ボタンが押されているかどうか
    if( (GetMouseInput() & MOUSE_INPUT_LEFT ) != 0 ) {
      DrawString( 20, 50, "マウス左ボタンが押されています", white );
    }
    //裏画面反映
    ScreenFlip();
  }

```

```
39    //DXライブラリ終了処理
40    DxLib_End();
41    return 0;
42  }
```

このプログラムの実行結果は図 4.4 となります.

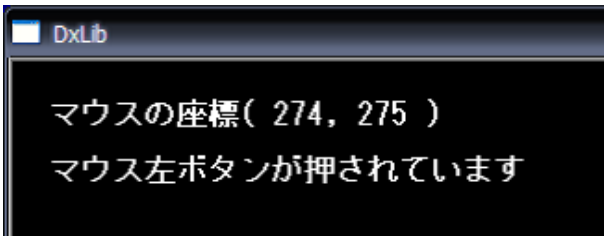

図 4.4: マウスボタンの状態を取得するプログラム

第5章　サウンド取り扱いの基礎

この章では，音楽，効果音の再生方法の基礎について解説していきます．

5.1　サウンドの再生

WAV,MP3,OGG ファイルを再生するには，PlaySoundFile 関数を利用します．この関数の定義は以下のようになっています．

```
int PlaySoundFile( char *FileName , int PlayType );
```

この関数は再生に成功すると戻り値として 0 を，失敗すると-1 を返してきます．第 1 引数 Filename には音声ファイルのパスを指定します．第 2 引数 PlayType には音声の再生方法を指定します（表 5.1）．

定義	再生形式
DX_PLAYTYPE_NORMAL	通常再生
DX_PLAYTYPE_BACK	バックグラウンド再生
DX_PLAYTYPE_LOOP	ループ再生

表 5.1: 音声再生方式

通常再生は，サウンド再生が終わるまで処理を停止します．それに対し，バックグラウンド再生は再生を始めると同時に処理を返してきます．通常はバックグラウンド再生を利用します．ループ再生は，バックグラウンド再生とほとんど同じですが，再生を停止するまでループして音を再生します．

では，音楽を再生するプログラムをお見せします．今回利用した音声ファイルは loop3.mp3 です．

リスト 5.1: ”foundation-17.cpp”

```
1  #include "DxLib.h"
2
3  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
4             LPSTR lpCmdLine, int nCmdShow )
5  {
6    int white;
7
8    //ウィンドウモードで起動
9    ChangeWindowMode( TRUE );
10   //DXライブラリ初期化
```

```
11    if( DxLib_Init() == -1 ) {
12      return -1;
13    }
14
15    //白色取得
16    white = GetColor(255,255,255);
17
18    //音楽loop3.mp3の再生
19    if( PlaySoundFile( "loop3.mp3", DX_PLAYTYPE_LOOP ) == 0 ) {
20      DrawString( 30, 30, "loop3.mp3を再生中", white );
21    }
22
23    WaitKey();
24    //DXライブラリ終了処理
25    DxLib_End();
26    return 0;
27  }
```

このプログラムの実行結果は図 5.1 となります.

図 5.1: 音楽の再生

5.2 メモリ上にあるサウンドの再生

前節で利用した PlaySoundFile 関数は，関数を呼び出した際にハードディスクから音声ファイルを読み込んできます．このため，PlaySoundFile 関数は高速な処理が求められるゲームには向いていません．今回は，メモリ上にサウンドをあらかじめロードしておき，必要に応じて再生するようにしましょう.

音声ファイルをメモリ上に読み込むには LoadSoundMem 関数を利用します．この関数の定義は以下のようになっています.

```
int LoadSoundMem( char *FileName );
```

FileName には音声ファイルのパスを指定します．この関数は，音声ファイル読み込み成功時には，戻り値としてサウンドハンドルと呼ばれるサウンド認識番号を返してきます.

メモリ上に読み込んだ音声を再生するには，PlaySoundMem 関数を利用します.

```
int PlaySoundMem( int SoundHandle , int PlayType );
```

第 1 引数 SoundHandle には，サウンド認識番号を渡します．第 2 引数 PlayType には，表 5.1 の音声再生形式を渡します.

では，これらの関数を利用して，音楽及び効果音を再生するプログラムをお見せします．A,B,C いずれかのキーを押すと，効果音 hapyou1.wav, hapyou2.wav, toujyo.wav が再生されます．また，常に loop3.mp3 がバックグラウンドミュージックとして再生されています．

リスト 5.2: ”foundation-18.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  int white;
  int music;
  int soundA, soundB, soundC;
  //キー状態
  char keyStatus[ 256 ];

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }

  //白色取得
  white = GetColor(255,255,255);

  //音楽読み込み
  music = LoadSoundMem( "loop3.mp3" );
  //サウンド読み込み
  soundA = LoadSoundMem( "hapyou1.wav" );
  soundB = LoadSoundMem( "hapyou2.wav" );
  soundC = LoadSoundMem( "toujyo.wav" );

  //音楽再生
  PlaySoundMem( music, DX_PLAYTYPE_LOOP );

  while( 1 ) {
    if( ProcessMessage() == -1 ) {
      break;
    }
    //キーの状態を取得
    GetHitKeyStateAll( keyStatus );

    //A, B, Cキーでサウンド再生
    if( keyStatus[ KEY_INPUT_A ] ) {
      PlaySoundMem( soundA, DX_PLAYTYPE_BACK );
    }else if ( keyStatus[ KEY_INPUT_B ] ) {
      PlaySoundMem( soundB, DX_PLAYTYPE_BACK );
    }else if ( keyStatus[ KEY_INPUT_C ] ) {
      PlaySoundMem( soundC, DX_PLAYTYPE_BACK );
    }

    DrawString( 20, 20, "A, B, Cキーでサウンド再生", white );
  }

  WaitKey();
  //DXライブラリ終了処理
  DxLib_End();
  return 0;
}
```

音声ファイルのため，実行結果は省略します．

5.3 サウンドデータの音量調節

音声データの音量を設定するには ChangeVolumeSoundMem 関数を利用します．この関数の定義は以下のようになっています．

```
int ChangeVolumeSoundMem( int VolumePal, int SoundHandle );
```

第 1 引数 VolumePal には音量 (最小値 0, 最大値 255) を渡します．第 2 引数にはサウンド識別番号（サウンドハンドル）を渡します．

では，この関数を利用して，音楽のボリューム調節を行うプログラムの例をお見せします．上ボタンでボリュームアップ，下ボタンでボリュームダウンを行います．

リスト 5.3: ”foundation-19.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow )
{
  int white;
  int music;
  //キー状態
  char keyStatus[ 256 ];
  //ボリューム (0～255)
  int volume = 100;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //DXライブラリ初期化
  if( DxLib_Init() == -1 ) {
    return -1;
  }
  //バックバッファへの描画
  SetDrawScreen( DX_SCREEN_BACK );
  //白色取得
  white = GetColor(255,255,255);

  //音楽読み込み
  music = LoadSoundMem( "loop3.mp3" );
  //音楽再生
  PlaySoundMem( music, DX_PLAYTYPE_LOOP );

  while( 1 ) {
    ClearDrawScreen();
    if( ProcessMessage() == -1 ) {
      break;
    }
    //キーの状態を取得
    GetHitKeyStateAll( keyStatus );

    //上，下ボタンでボリューム調節
    if( keyStatus[ KEY_INPUT_UP ] ) {
      //ボリュームを上げる
      volume++;
      if( volume > 255 ) {
        volume = 255;
      }
      //ボリューム設定
      ChangeVolumeSoundMem( volume, music );
    }else if ( keyStatus[ KEY_INPUT_DOWN ] ) {
      //ボリュームを下げる
```

```
48        volume--;
49        if( volume < 0 ) {
50          volume = 0;
51        }
52        //ボリューム設定
53        ChangeVolumeSoundMem( volume, music );
54      }
55      DrawFormatString( 20, 20, white, "ボリューム %d(最小0, 最大255)", volume );
56      ScreenFlip();
57    }
58
59    WaitKey();
60    //DXライブラリ終了処理
61    DxLib_End();
62    return 0;
63  }
```

このプログラムの実行結果は図 5.2 となります.

図 5.2: 音量の調節

第II部

ノベルゲームの作成

ノベルゲームは，別名サウンドノベルやビジュアルノベル等と呼ばれるゲームジャンルの一つです．このゲームの完成図は図 5.3 のようになります．

図 5.3: ノベルゲーム

ノベルゲームは，シューティングゲームやアクションゲーム等とは違い，数学の知識は必要ありませんし，難しいアルゴリズムもほとんど登場しません．よって，初心者が作成するにはうってつけのジャンルであると思われます．

今回作成するノベルゲームに組み込む機能は次の通りです．

- メッセージ表示機能
- グラフィック表示機能
- 選択肢の実装
- 簡易スクリプト言語によるストーリー作成機能

上記の中で唯一難しいものは，ノベルゲーム用のスクリプト言語を作成する事です．まともなスクリプト言語を作ろうと作ろうと思うと，それだけで本が一冊できてしまうほどの難しさです．しかし，今回はノベルゲーム用なので，難しい考えは全て排除する事とし

ました．しかし，実装が簡単な分，制約も多く出てしまいます．本書を理解し，ノベルゲームが作れるようになったら，是非独自のスクリプト言語を実装してみてください．

さて，プログラムを作成していくには，次の二つの方針があります．

- 機能をどんどん拡張していく方法 (図 5.4)．
- ゲームに必要な部品を作っていき，最後に結合する方法（図 5.5)．

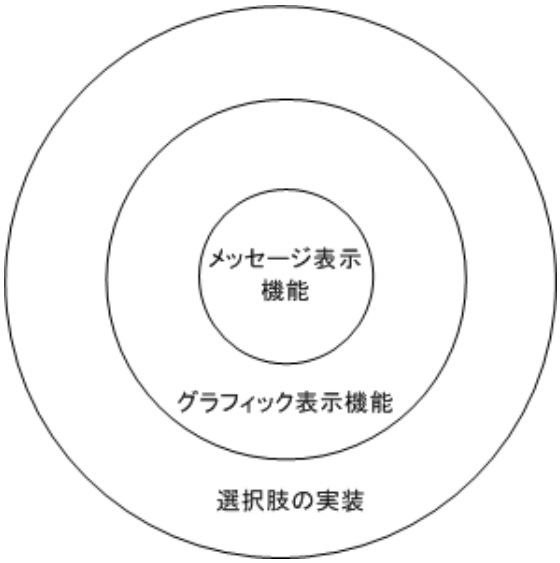

図 5.4: 機能拡張していく方法

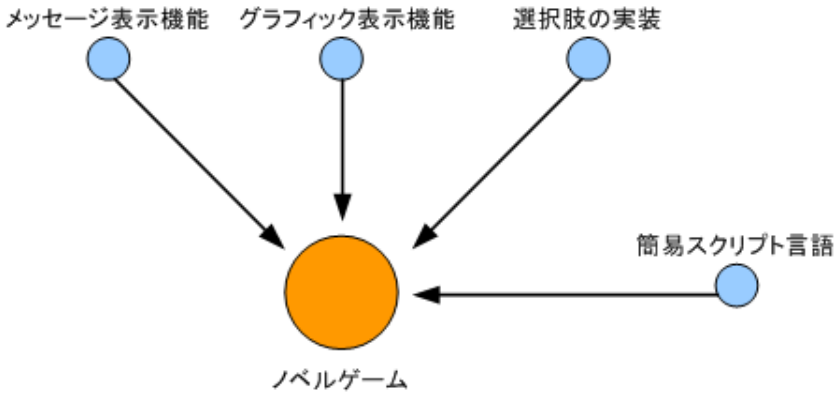

図 5.5: 機能ごとの部品を作っていき，最後に結合する方法

本書では，後者の方針を取る事にします．それは，前者の方法では機能を拡張するごとにステップ数が増えていき，ソースがごちゃごちゃになってしまう可能性があるからです．後者の方法では，メッセージ表示機能とグラフィック表示機能などは切り離して考えておき，最後に結合するので，設計が楽になります．

第6章　サウンドノベル風メッセージ表示

サウンドノベル風にメッセージを表示するプログラムを作っていきたいと思います．サウンドノベル風メッセージとは，画面に一文字づつ文字を描画していくものです．完成イメージは図 6.1 です．

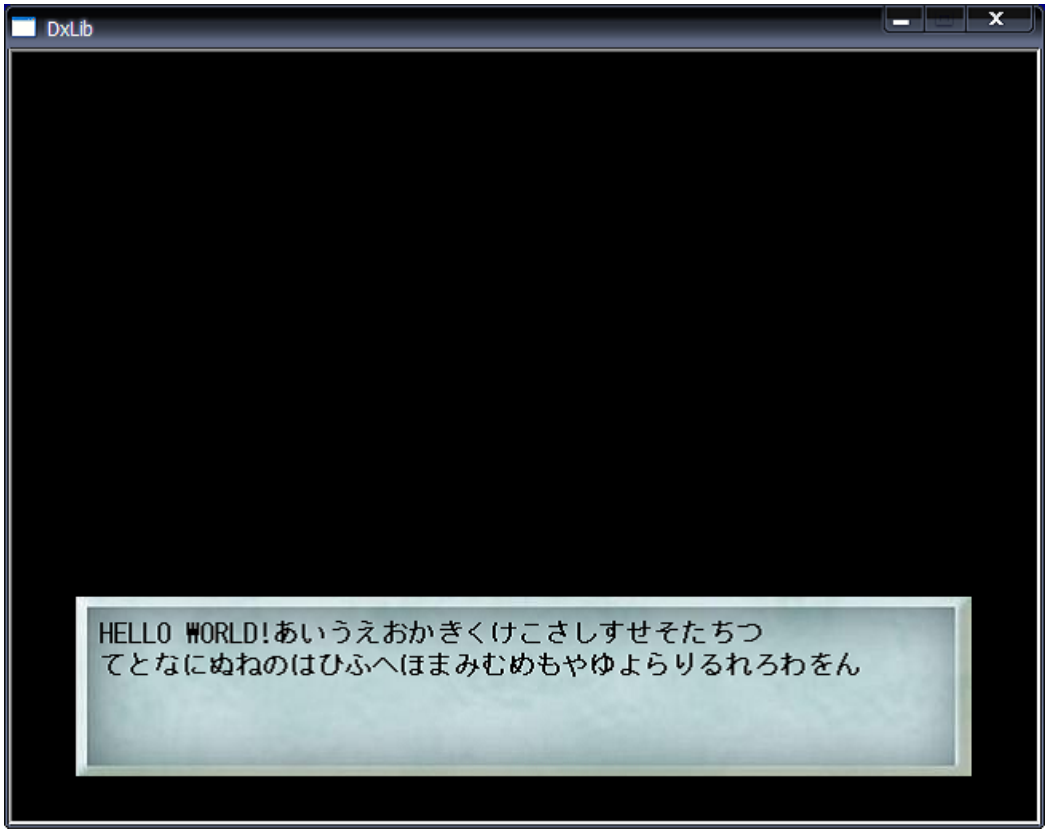

図 6.1: サウンドノベル風メッセージ表示

いきなり GUI で作り始めるのは大変なので，まずは CUI で部品を作っていき，その後 GUI に移植していきます．

6.1 指定範囲の英字文字列を表示

まずは指定範囲の文字列を表示するプログラムを作成してみましょう．これは，メッセージを複数行表示するときに使用します．

例えば，表示したいメッセージが 50 文字あった場合，すべてを 1 行で表示すると画面からはみ出してしまいます．コンソールで文字を表示する時とは違い，DX ライブラリで文字列を表示するときは，改行などの処理は自分で行わなければなりません．よって，1 行目を 1 文字目から 25 文字目まで表示させ，2 行目は 26 文字目から 50 文字目まで表示するといった作業が必要となります．こういったときに利用する部品を作っていきます．

では，まずは次のような関数を作ってみます．

リスト 6.1: ”message-cui-01.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //表示したい文字列
5  char g_message[256] = "HelloWorld";
6
7  //messageで指定した文章を start の位置から len 文字分表示する
8  void writeSubstring(char* message, int start, int len)
9  {
10   int i;
11   //文字数
12   int maxLen = strlen( message );
13
14   //startの位置が表示したい最大文字数より大きい場合
15   if( start >= maxLen ) {
16     return;
17   }
18
19   //指定した位置からlen文字分表示する
20   for( i = 0; i < len && message[ start + i ] != '\0'; i++ ) {
21     printf("%c", message[ start + i ] );
22   }
23   printf("\n");
24 }
25
26 int main()
27 {
28   //g_messageを表示する
29   writeSubstring(g_message, 0, strlen( g_message) );
30   //g_message 2文字目( g_message[1] )から5文字分表示する
31   writeSubstring(g_message, 1, 5);
32   //g_message 6文字目( g_message[5] )から10文字分表示する
33   //10文字は表示できないので，文字列の最後まで表示
34   writeSubstring(g_message, 5, 10);
35   //g_message 20文字目から表示（なにも表示されない）
36   writeSubstring(g_message, 20, 10);
37 }
```

writeSubstring 関数は message で渡した文字列の start から len 文字分表示する関数です．main 関数でどのような動作をするか確認してみましょう．実行結果は以下のようになりました．

実行結果

```
HelloWorld
elloW
World
```

さて，writeSubstring 関数が正しく動作しているか確認していきましょう．

実行結果 1 行目は g_message をただ表示するものです．よって HelloWorld が表示されています．この関数で気をつけなければならないことは，1 文字目が 0 の位置から始まることです．

2 行目は 2 文字目から 5 文字文表示するものです．うまくいっていますね．3 行目は 5 文字目から 10 文字分表示するものです．といっても，HelloWorld の 6 文字目から 10 文字も文字が存在しません．よって 6 文字目から文字列の最後まで表示しています．最後の writeSubstring 関数は 20 文字目から 10 文字分表示するものですが，HelloWorld は 20 文字もないので，画面には何も表示されません．

6.2 指定範囲の日本語文字列を表示

さて，前節の writeSubstring 関数ですが，実は日本語を表示しようとするとうまくいきません．

ここで文字コードの問題が浮上します．日本語は char1 文字（1byte) だけでは表現することができないのです．よって，char 2 文字で表現しているのです．ただし，これは文字コードが SHIFT_JIS の場合です[1]．ほかの文字コードでは，もしかしたら日本語を 3 バイト (char3 つ分）で表しているかもしれません．

文字コードの問題はとても厄介です．著者も Windows プログラムを Linux 用に移植した時に，文字コードの問題によく悩まされたものです．まあ，ここでは Windows でゲームを作るので，文字コードは SHIFT_JIS としましょう．

では，前節のプログラムを日本語に対応させてみましょう．

リスト 6.2: ”message-cui-02.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //注意点：文字コードはSHIFT_JIS（Windows標準）を前提としている
5  //      他の文字コードでは正常に動作しない
6  //SHIFT_JISの場合，日本語は2バイトで表される
7
8  //表示したい文字列
9  char g_message[256] = "はろーわーるど";
10
11 //messageで指定した文章を start の位置から len 文字分表示する
12 void writeSubstring(char* message, int start, int len)
13 {
14   int i;
15   //文字数
16   int maxLen = strlen( message );
17
```

[1] SHIFT_JIS は Windows 標準の文字コードです．

```
18    //日本語の場合，位置を2倍する
19    start *= 2;
20    len *= 2;
21
22    //startの位置が表示したい最大文字数より大きい場合
23    if( start >= maxLen ) {
24      return;
25    }
26
27    //指定した位置からlen文字分表示する
28    for( i = 0; i < len && message[ start + i ] != '\0'; i += 2 ) {
29      printf("%c", message[ start + i ] );
30      printf("%c", message[ start + i + 1 ]);
31    }
32    printf("\n");
33  }
34
35  int main()
36  {
37    //g_messageを表示する
38    writeSubstring(g_message, 0, strlen( g_message) );
39    //g_message 2文字目から5文字分表示する
40    writeSubstring(g_message, 1, 5);
41    //g_message 6文字目から10文字分表示する
42    //10文字は表示できないので，文字列の最後まで表示
43    writeSubstring(g_message, 5, 10);
44    //g_message 20文字目から表示（なにも表示されない）
45    writeSubstring(g_message, 20, 10);
46  }
```

実行させて正しく動作するかどうか見てみましょう．

実行結果

```
はろーわーるど
ろーわーる
るど
```

どうやらうまく動いているようです．

6.3 日本語文字の判定方法

前節で日本語文字列を切り出すことができるようになりました．しかし，ここで新たな問題が浮上してしまいました．それは，日本語と英文字が混ざっている場合です．

よって，いまから表示する文字が日本語かどうかを判定する必要があるわけです．もし，日本語かどうかが判定できれば，英字の場合はインデクスを 1 進め，日本語の場合はインデクスを 2 進めればよいわけです．

SHIFT_JIS の場合，日本語は 2 バイトで表現しているわけですが，実は 1 バイト目を調べるだけで日本語（2 バイト文字）なのかそうでないのかが判定できる仕組みになっています．

SHIFT_JIS の 2 バイト文字の場合，第一バイトが 0x81〜0x9f または 0xe0〜0xfc の範囲にあります．よって，次のようなコードで指定された char 文字が日本語かそうでないのかを判定できます．

```
1  //code が日本語であるか判定する
2  //戻り値 1:日本語 0:日本語ではない
3  int isJapaneseCharacter(unsigned char code)
4  {
5          if( (code >= 0x81 && code <= 0x9F) ||
6                  (code >= 0xE0 && code <= 0xFC) ) {
7                          return 1;
8          }
9          return 0;
10 }
```

では次のようなコードを書いて正しく動作するかどうか確認していきましょう.

リスト 6.3: ”message-cui-03.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //SHIFT_JISの場合，上位バイトが0x81～0x9F、0xE0～0xFCの範囲に収まる
5
6  //code が日本語であるか判定する
7  //戻り値 1:日本語 0:日本語ではない
8  int isJapaneseCharacter(unsigned char code)
9  {
10   if( (code >= 0x81 && code <= 0x9F) ||
11     (code >= 0xE0 && code <= 0xFC) ) {
12       return 1;
13   }
14   return 0;
15 }
16
17 int main()
18 {
19   char check_first[3] = "え";
20   char check_second[3] = "下";
21   char check_third[3] = "h";
22
23   printf("「え」の場合, %d\n", isJapaneseCharacter( check_first[0] ) );
24   printf("「下」の場合, %d\n", isJapaneseCharacter( check_second[0] ) );
25   printf("「h 」の場合, %d\n", isJapaneseCharacter( check_third[0] ) );
26
27   return 0;
28 }
```

実行結果

「え」の場合, 1
「下」の場合，1
「h 」の場合，0

どうやらうまく判定できているようです．次節ではこの関数を使って writeSubstring 関数を改良していきましょう.

6.4 指定範囲の文字列を表示

やっと指定範囲の文字列を表示させる準備が整いました．改良した writeSubstring 関数は英字，日本語文字が混ざっていても正しく表示させることができます．表示させる文字列は，

はろー Hello わーるど World

です．コードは次のようになりました．

リスト 6.4: ”message-cui-04.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //注意点：文字コードはSHIFT_JIS（Windows標準）を前提としている
5  //       他の文字コードでは正常に動作しない
6  //SHIFT_JISの場合，日本語は2バイトで表される
7  //上位バイトが0x81～0x9F、0xE0～0xFCの範囲に収まる
8
9  //表示したい文字列
10 char g_message[256] = "はろーHelloわーるどWorld";
11
12
13 //code が日本語であるか判定する
14 //戻り値 1:日本語 0:日本語ではない
15 int isJapaneseCharacter(unsigned char code)
16 {
17   if( (code >= 0x81 && code <= 0x9F) ||
18     (code >= 0xE0 && code <= 0xFC) ) {
19       return 1;
20   }
21   return 0;
22 }
23
24
25 //messageで指定した文章を start の位置から len 文字分表示する
26 void writeSubstring(char* message, int start, int len)
27 {
28   int i;
29   //文字数
30   int maxLen = strlen( message );
31
32   //startの位置を変更する
33   //startの位置までに日本語がでてきていたら，1を足していく
34   for( i = 0; i < start && message[i] != '\0'; ) {
35     if( isJapaneseCharacter( message[i] ) ) {
36       //日本語の場合，2バイト分すすめる
37       i += 2;
38       //startに1バイト分足す
39       start++;
40     }else {
41       //半角文字の場合，1バイト分進める
42       i++;
43     }
44   }
45
46   //startの位置が表示したい最大文字数より大きい場合
47   if( start >= maxLen ) {
48     return;
49   }
50
51   //指定した位置からlen文字分表示する
```

```
52    for( i = 0; i < len && message[ start + i ] != '\0'; ) {
53      if( isJapaneseCharacter( message[ start + i ] ) ) {
54        //日本語の場合, 2文字分表示する
55        printf("%c", message[ start + i ] );
56        printf("%c", message[ start + i + 1 ]);
57        //lenは日本語なので, 1バイト分追加する
58        len++;
59        //2バイト分進める
60        i += 2;
61      }else {
62        //半角文字1文字を表示
63        printf("%c", message[ start + i ] );
64        //1バイト分進める
65        i++;
66      }
67    }
68    printf("\n");
69  }
70
71  int main()
72  {
73    //g_messageを表示する
74    writeSubstring(g_message, 0, strlen( g_message) );
75    //g_message 2文字目から5文字分表示する
76    writeSubstring(g_message, 1, 5);
77    //g_message 6文字目から30文字分表示する
78    //30文字は表示できないので, 文字列の最後まで表示
79    writeSubstring(g_message, 5, 30);
80    //g_message 20文字目から表示（なにも表示されない）
81    writeSubstring(g_message, 20, 10);
82  }
```

実行結果

はろー Hello わーるど World
ろー Hel
llo わーるど World

ようやく完成しました．うまく動作していますね.

6.5 指定範囲の文字列を表示 GUI 版

ここからは DX ライブラリを使って画面に文字を表示させていきましょう．前節のプログラムを GUI に移植してみましょう．文字列を表示させるには DrawString 関数を利用します.

```
int DrawString( int x , int y , char *String , int Color );
```

描画する文字列の左上の座標を (x,y), 表示したいメッセージは String, メッセージの色は Color となっています.

では,「はろー hello わーるど」という文字列を 1 文字目から 5 文字分表示するプログラムを作成してみましょう．なんてことはありません．前節のプログラムを少し書き換えただけです.

リスト 6.5: ”message-gui-01.cpp”

```
#include "DxLib.h"

//注意点：文字コードはSHIFT_JIS（Windows標準）を前提としている
//       他の文字コードでは正常に動作しない
//SHIFT_JISの場合，日本語は2バイトで表される
//上位バイトが0x81～0x9F、0xE0～0xFCの範囲に収まる

int isJapaneseCharacter(unsigned char code);
void writeSubstring(char* message, int start, int len, int posX, int posY, int color);

//メッセージのフォントの大きさ
#define MESSAGE_FONT_SIZE 20
//仮想バッファの最大文字数
#define MESSAGE_MAX_LENGTH 30
//仮想バッファの最大行数
#define MESSAGE_MAX_LINE 5

//表示したいメッセージ
char g_message[MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE] = "はろーhelloわーるど";

//画面にメッセージを表示する際にしようする仮想テキストバッファ
char g_messageBuffer[MESSAGE_MAX_LINE][MESSAGE_MAX_LENGTH];


//code が日本語であるか判定する
//戻り値 1:日本語 0:日本語ではない
int isJapaneseCharacter(unsigned char code)
{
  if( (code >= 0x81 && code <= 0x9F) ||
    (code >= 0xE0 && code <= 0xFC) ) {
      return 1;
  }
  return 0;
}


//messageで指定した文章を start の位置から len 文字分表示する
//文字列左側の座標は(posX, posY)，文字の色をcolorとする
void writeSubstring(char* message, int start, int len, int posX, int posY, int color)
{
  int i;
  //文字数
  int maxLen = strlen( message );

  //startの位置を変更する
  //startの位置までに日本語がでてきていたら，1を足していく
  for( i = 0; i < start && message[i] != '\0'; ) {
    if( isJapaneseCharacter( message[i] ) ) {
      //日本語の場合，2バイト分すすめる
      i += 2;
      //startに1バイト分足す
      start++;
    }else {
      //半角文字の場合，1バイト分進める
      i++;
    }
  }

  //startの位置が表示したい最大文字数より大きい場合
  if( start >= maxLen ) {
    return;
  }

  //指定した位置からlen文字分表示する
```

```
65    for( i = 0; i < len && message[ start + i ] != '\0'; ) {
66      if( isJapaneseCharacter( message[ start + i ] ) ) {
67        //日本語の場合, 2文字分表示する
68        g_messageBuffer[0][ i ] = message[ start + i ];
69        g_messageBuffer[0][ i + 1 ] = message[ start + i + 1 ];
70        //lenは日本語なので, 1バイト分追加する
71        len++;
72        //2バイト分進める
73        i += 2;
74      }else {
75        //半角文字1文字を表示
76        //printf("%c", message[ start + i ] );
77        g_messageBuffer[0][ i ] = message[ start + i ];
78        //1バイト分進める
79        i++;
80      }
81    }
82    g_messageBuffer[0][i] = '\0';
83
84    //メッセージ描画
85    DrawString(posX, posY, g_messageBuffer[0], color );
86  }
87
88  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
89                      LPSTR lpCmdLine, int nCmdShow )
90  {
91    //ウィンドウモードで起動
92    ChangeWindowMode( TRUE );
93    //画面の大きさは640 * 480
94    SetGraphMode( 640 , 480 , 16 ) ;
95    //DxLib初期化
96    if( DxLib_Init() == -1 ) {
97       return -1;
98    }
99
100   // 描画先を裏画面にセット
101   SetDrawScreen( DX_SCREEN_BACK ) ;
102
103   int whiteColor = GetColor(255,255,255);
104
105   //メインループ
106   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
107     //g_messageの文字を1文字目から5文字分表示する
108     //文字列描画の位置は(50, 100), 色は白とする
109     writeSubstring( g_message, 0, 5, 50,100, whiteColor );
110     Sleep( 100 );
111     ScreenFlip();
112   }
113
114   DxLib_End();
115   return 0;
116 }
```

g_messageBuffer は画面にメッセージを表示する文字をためておくものです. 今回は g_messageBuffer[0], つまり 1 行目までしか利用していないわけですが, 今後メッセージが 2 行,3 行に渡って表示するときに g_messageBuffer[1], g_messageBuffer[2] を利用していきます.

このプログラムの実行結果は, 図 6.2 となります.

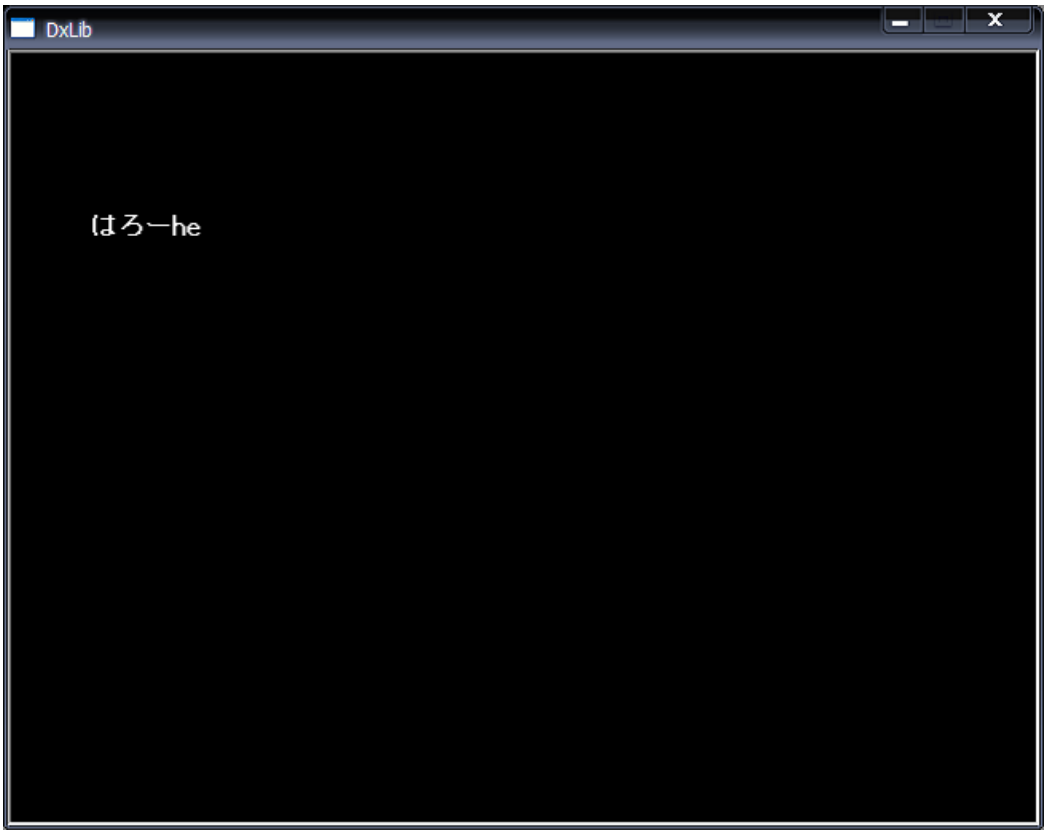

図 6.2: 指定した範囲の文字列を表示

6.6 メッセージを 1 文字ずつ表示

メッセージを 1 文字ずつ画面に表示させてみましょう．特に難しい処理はしていません．ただ，最初は 1 文字だけ表示，約 100ms 後に 2 文字目まで表示，さらに 100ms 後に 3 文字目まで表示 と繰り返しているだけです．

リスト 6.6: ”message-gui-02.cpp”

```
#include "DxLib.h"

int isJapaneseCharacter(unsigned char code);
void writeSubstring(char* message, int start, int len, int posX, int posY, int color);

//メッセージのフォントの大きさ
#define MESSAGE_FONT_SIZE 20
//仮想バッファの最大文字数
#define MESSAGE_MAX_LENGTH 30
//仮想バッファの最大行数
#define MESSAGE_MAX_LINE 5

//表示したいメッセージ
char g_message[MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE] = "はろーhelloわーるど";

//画面にメッセージを表示する際にしようする仮想テキストバッファ
char g_messageBuffer[MESSAGE_MAX_LINE][MESSAGE_MAX_LENGTH];


//code が日本語であるか判定する
//戻り値 1:日本語 0:日本語ではない
int isJapaneseCharacter(unsigned char code)
{
  //省略（前回と同じ）
}


//messageで指定した文章を start の位置から len 文字分表示する
//文字列左側の座標は(posX, posY), 文字の色をcolorとする
void writeSubstring(char* message, int start, int len, int posX, int posY, int color)
{
  //省略（前回と同じ）
}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
             LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //画面の大きさは640 * 480
  SetGraphMode( 640 , 480 , 16 ) ;
  //DxLib初期化
  if( DxLib_Init() == -1 ) {
     return -1;
  }

  // 描画先を裏画面にセット
  SetDrawScreen( DX_SCREEN_BACK ) ;

  //現在何文字目までを表示しているか
  int currentCursor = 0;
  int whiteColor = GetColor(255,255,255);

  //メインループ
  while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
    if( g_message[currentCursor] != '\0' ) {
```

```
57        currentCursor++;
58      }
59
60      writeSubstring( g_message, 0, currentCursor, 50,100, whiteColor );
61      Sleep( 100 );
62      ScreenFlip();
63    }
64
65    DxLib_End();
66    return 0;
67  }
```

メインループ内で currentCursor という変数を利用していますが，この値ははじめ 0 にセットされており，時間が立つにつれて 1,2,3 と増えていきます．g_message[currentCursor] がナル文字 (\0) と等しくなるまでこの増加を繰り返します．また，文字描画部分では，0 文字目から currentCursor 文字目までを画面に描画しています．このような処理をすることで，画面に 1 文字ずつ文字が描画しているようにみえるわけです．

6.7 改行処理を加えたメッセージ描画

さて，前節のプログラムでやっとサウンドノベル風のメッセージ描画ができたわけですが，1 つ問題があります．それは，メッセージが長すぎる場合に画面からはみ出してしまうことです．この問題を解決するために，改行処理を加えて見ましょう．

まずは writeSubstring 関数に少し手を加えましょう．関数の宣言を以下のように変更しました．

```
void writeSubstring(char* message, int start, int len,
                    int posX, int posY, int color, int bufferLine)
```

bufferLine という引数を一つ増やしました．これは，いま何行目の文字列を描画しているかを表すものです．

さて，メインループの中身も書き換えて見ましょう．どのような処理を行ったらよいでしょうか．ここで，メッセージが，例えば 70 文字あったとしましょう．また，1 行で表すことができる文字数は 30 とします．まず，1 行目を描画しているときは 1 文字目から 30 文字分を 1 文字ずつ g_messageBuffer[0] に格納し，画面に描画します．つぎに 31 文字目の描画に入るわけですが，そのときは 1 行目にはもう文字を追加することができません．よって，g_messageBuffer[1] に今度は 1 文字ずつ文字を格納していくわけです．3 行目の時も同様に処理を行います．

さて，これらを考慮してプログラムを組むと次のようになりました．

リスト 6.7: ”message-gui-03.cpp”

```
1  #include "DxLib.h"
2
3  int isJapaneseCharacter(unsigned char code);
4  void writeSubstring(char* message, int start, int len,
5          int posX, int posY, int color, int bufferLine);
6
7  //メッセージのフォントの大きさ
```

```
8  #define MESSAGE_FONT_SIZE 20
9  //仮想バッファの最大文字数
10 #define MESSAGE_MAX_LENGTH 30
11 //仮想バッファの最大行数
12 #define MESSAGE_MAX_LINE 5
13
14 //表示したいメッセージ
15 char g_message[MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE] =
16   "はろーhelloわーるどWorldあいうえおかきくけこさしすせそたちつてとな" \
17   "にぬねのはひふへほまみむめもやゆよらりるれろわをん";
18
19 //画面にメッセージを表示する際にしようする仮想テキストバッファ
20 char g_messageBuffer[MESSAGE_MAX_LINE][MESSAGE_MAX_LENGTH];
21
22
23 //code が日本語であるか判定する
24 //戻り値 1:日本語 0:日本語ではない
25 int isJapaneseCharacter(unsigned char code)
26 {
27   if( (code >= 0x81 && code <= 0x9F) ||
28     (code >= 0xE0 && code <= 0xFC) ) {
29       return 1;
30   }
31   return 0;
32 }
33
34
35 //messageで指定した文章を start の位置から len 文字分表示する
36 void writeSubstring(char* message, int start, int len,
37      int posX, int posY, int color, int bufferLine)
38 {
39   int i;
40   //文字数
41   int maxLen = strlen( message );
42
43   //startの位置を変更する
44   //startの位置までに日本語がでてきていたら, 1を足していく
45   for( i = 0; i < start && message[i] != '\0'; ) {
46     if( isJapaneseCharacter( message[i] ) ) {
47       //日本語の場合, 2バイト分すすめる
48       i += 2;
49       //startに1バイト分足す
50       start++;
51     }else {
52       //半角文字の場合, 1バイト分進める
53       i++;
54     }
55   }
56
57   //startの位置が表示したい最大文字数より大きい場合
58   if( start >= maxLen ) {
59     return;
60   }
61
62   //指定した位置からlen文字分表示する
63   for( i = 0; i < len && message[ start + i ] != '\0'; ) {
64     if( isJapaneseCharacter( message[ start + i ] ) ) {
65       //日本語の場合, 2文字分bufferにセット
66       g_messageBuffer[ bufferLine ][ i ] = message[ start + i ];
67       g_messageBuffer[ bufferLine ][ i + 1 ] = message[ start + i + 1 ];
68       //lenは日本語なので, 1バイト分追加する
69       len++;
70       //2バイト分進める
71       i += 2;
72     }else {
```

```
73        //半角文字1文字をセット
74        g_messageBuffer[ bufferLine ][ i ] = message[ start + i ];
75        //1バイト分進める
76        i++;
77      }
78    }
79    g_messageBuffer[ bufferLine ][i] = '\0';
80
81    //メッセージ描画
82    DrawString(posX, posY, g_messageBuffer[ bufferLine ], color );
83  }
84
85  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
86                LPSTR lpCmdLine, int nCmdShow )
87  {
88    //ウィンドウモードで起動
89    ChangeWindowMode( TRUE );
90    //画面の大きさは640 * 480
91    SetGraphMode( 640 , 480 , 16 ) ;
92    //DxLib初期化
93    if( DxLib_Init() == -1 ) {
94       return -1;
95    }
96
97    // 描画先を裏画面にセット
98    SetDrawScreen( DX_SCREEN_BACK ) ;
99
100   int i;
101
102   //現在何文字目までを表示しているか
103   int currentCursor = 0;
104   //何行目の文字を表示しているか
105   int currentLineCursor = 0;
106   //白
107   int whiteColor = GetColor(255,255,255);
108
109   //メインループ
110   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
111     if( g_message[currentCursor] != '\0' ) {
112       currentCursor++;
113     }
114
115     //画面クリア
116     ClearDrawScreen();
117
118     //MESSAGE_MAX_LENGTH まで文字を描画したら段落を切り替える
119     if( currentCursor % MESSAGE_MAX_LENGTH == 0 ) {
120       if( g_message[currentCursor] != '\0' ) {
121         currentLineCursor++;
122       }
123     }
124
125     for( i = 0; i < MESSAGE_MAX_LINE; i++ ) {
126       if( i == currentLineCursor ) {
127         //サウンドノベルメッセージ風に表示
128         writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
129                                    currentCursor - MESSAGE_MAX_LENGTH * i,
130           50, 100 + MESSAGE_FONT_SIZE * i, whiteColor, i );
131         break;
132       }else {
133         //メッセージをそのまま表示
134         writeSubstring( g_message, i * MESSAGE_MAX_LENGTH , MESSAGE_MAX_LENGTH, 50,
135           100 + MESSAGE_FONT_SIZE * i, whiteColor, i );
136       }
137     }
```

```
138     Sleep( 100 );
139     ScreenFlip();
140   }
141
142   DxLib_End();
143   return 0;
144 }
```

メインループ内の currentLineCursor は現在描画中のメッセージの行番号を表しています．段落を切り替える際に currentCursor % MESSAGE_MAX_LENGTH という処理をしています．これは，currentCursor が MESSAGE_MAX_LENGTH で割り切れた時に if 文を実行することを表しています．具体的には，MESSAGE_MAX_LENGTH は 30 なわけですが，現在描画中の文字が 30 の倍数番目に差し掛かったときに改行するということです．よって，30 文字目，60 文字目，90 文字目の時に改行処理を行います．

メッセージ描画部分ですが，途中以下のような処理をしているところがあります．

```
1   for( i = 0; i < MESSAGE_MAX_LINE; i++ ) {
2           if( i == currentLineCursor ) {
3                   //サウンドノベルメッセージ風に表示
4                   writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
5                           currentCursor - MESSAGE_MAX_LENGTH * i,
6                           50, 100 + MESSAGE_FONT_SIZE * i, whiteColor, i );
7                   break;
8           }else {
9                   //メッセージをそのまま表示
10                  writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
11                          MESSAGE_MAX_LENGTH, 50,
12                          100 + MESSAGE_FONT_SIZE * i, whiteColor, i );
13          }
14  }
```

これは，これから表示する 1 文字（つまり currentCursor 番目の文字）が currentLineCursor と同じ行にある場合，サウンドノベル風にメッセージを表示させ，そうでない場合は，その行のメッセージをそのまま画面に表示させます．

さて，このプログラムの実行結果は 6.3 となりました．ちょっと見た目に問題があります．それは日本語文字の幅と英文字の幅が違うため，文字の折り返しを行う場所が 1 行目と 2 行目でずれてしまっています．これを修正するためにはいろいろと厄介な処理が必要なので，これで妥協するとしましょう．気に食わない方は，是非この問題を修正してみてください．

6.8 メッセージボックスの表示

メッセージをメッセージボックスの画像の上に表示させてみましょう．また，前回のプログラムはメインループの中にいろいろと処理を書きすぎました．よって，これらの処理

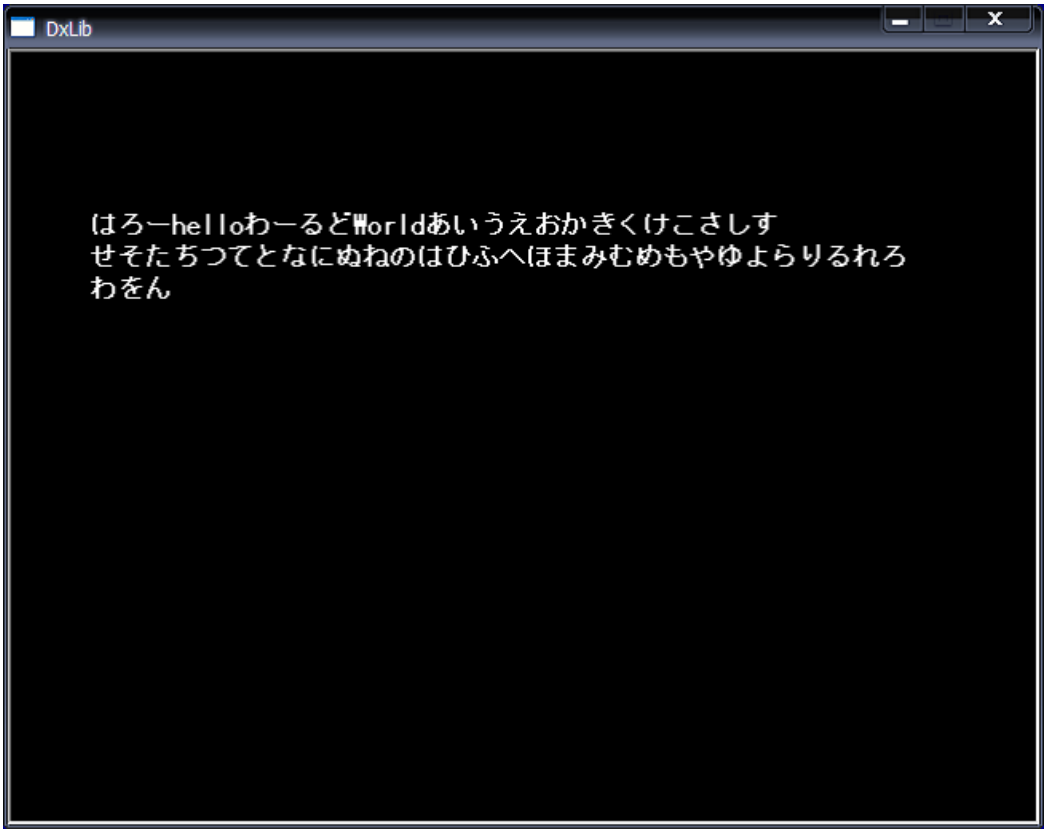

図 6.3: 改行処理を加えたメッセージ描画

を関数に分割していきたいと思います．関数に分割するにあたって，メインループ内に存在していたローカル変数，例えば currentCursor 変数などをグローバル変数にする必要もあります．

プログラムは以下のようになりました．

リスト 6.8: ”message-gui-04.cpp”

```
#include "DxLib.h"

int isJapaneseCharacter(unsigned char code);
void writeSubstring(char* message, int start, int len,
       int posX, int posY, int color, int bufferLine);
void drawMessage();
void initGame();

//メッセージのフォントの大きさ
#define MESSAGE_FONT_SIZE 20
//仮想バッファの最大文字数
#define MESSAGE_MAX_LENGTH 30
//仮想バッファの最大行数
#define MESSAGE_MAX_LINE 5
//メッセージボックスのX座標
#define MESSAGE_BOX_X_POS 40
//メッセージボックスのY座標
#define MESSAGE_BOX_Y_POS 340
//メッセージボックスの画像ファイル
#define MESSAGE_BOX_GRAPHIC_FILENAME "./Pic/boxd3.jpg"

//表示したいメッセージ
char g_message[MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE] =
  "はろーhelloわーるどWorldあいうえおかきくけこさしすせそたちつてとな" \
  "にぬねのはひふへほまみむめもやゆよらりるれろわをん";

//画面にメッセージを表示する際にしようする仮想テキストバッファ
char g_messageBuffer[MESSAGE_MAX_LINE][MESSAGE_MAX_LENGTH];

//メッセージボックス関係

//現在何文字目までを表示しているか
static int g_currentCursor = 0;
//何行目の文字を表示しているか
static int g_currentLineCursor = 0;
//白
static int g_whiteColor;
//黒
static int g_blackColor;
//メッセージボックスの画像
static int g_messageBoxGraphicHandle;


//code が日本語であるか判定する
//戻り値 1:日本語 0:日本語ではない
int isJapaneseCharacter(unsigned char code)
{
  //省略（前回と同じ）
}


//messageで指定した文章を start の位置から len 文字分表示する
void writeSubstring(char* message, int start, int len,
     int posX, int posY, int color, int bufferLine)
{
  //省略（前回と同じ）
}
```

```
58
59
60  //メッセージ描画
61  void drawMessage()
62  {
63    int i;
64
65    //メッセージボックス描画
66    DrawGraph( MESSAGE_BOX_X_POS, MESSAGE_BOX_Y_POS, g_messageBoxGraphicHandle, FALSE );
67
68    if( g_message[g_currentCursor] != '\0' ) {
69      g_currentCursor++;
70    }
71
72    //MESSAGE_MAX_LENGTH まで文字を描画したら段落を切り替える
73    if( g_currentCursor % MESSAGE_MAX_LENGTH == 0 ) {
74      if( g_message[g_currentCursor] != '\0' ) {
75        g_currentLineCursor++;
76      }
77    }
78
79    //メッセージ描画部分
80    for( i = 0; i < MESSAGE_MAX_LINE; i++ ) {
81      if( i == g_currentLineCursor ) {
82        //メッセージ風に表示
83        writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
84                              g_currentCursor - MESSAGE_MAX_LENGTH * i,
85           MESSAGE_BOX_X_POS + 15,
86                                   MESSAGE_BOX_Y_POS + MESSAGE_FONT_SIZE * i + 15,
87                                    g_blackColor, i );
88        break;
89      }else {
90        //メッセージをそのまま表示
91        writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
92                             MESSAGE_MAX_LENGTH, MESSAGE_BOX_X_POS + 15,
93          MESSAGE_BOX_Y_POS + MESSAGE_FONT_SIZE * i + 15,
94                                  g_blackColor, i );
95      }
96    }
97  }
98
99  //初期化処理
100 void initGame()
101 {
102   //白
103   g_whiteColor = GetColor(255,255,255);
104   //黒
105   g_blackColor = GetColor(0, 0, 0);
106   //メッセージボックス
107   g_messageBoxGraphicHandle = LoadGraph( MESSAGE_BOX_GRAPHIC_FILENAME );
108 }
109
110
111 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
112               LPSTR lpCmdLine, int nCmdShow )
113 {
114   //ウィンドウモードで起動
115   ChangeWindowMode( TRUE );
116   //画面の大きさは640 * 480
117   SetGraphMode( 640 , 480 , 16 ) ;
118   //DxLib初期化
119   if( DxLib_Init() == -1 ) {
120      return -1;
121   }
122
```

```
123   // 描画先を裏画面にセット
124   SetDrawScreen( DX_SCREEN_BACK ) ;
125
126   //初期化処理
127   initGame();
128
129   //メインループ
130   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
131
132     //画面クリア
133     ClearDrawScreen();
134
135     //メッセージ描画
136     drawMessage();
137
138     Sleep( 100 );
139     ScreenFlip();
140   }
141
142   DxLib_End();
143   return 0;
144 }
```

処理を分割した以外は特に前回と変わっていません．このプログラムの実行結果は図 6.4 となります．

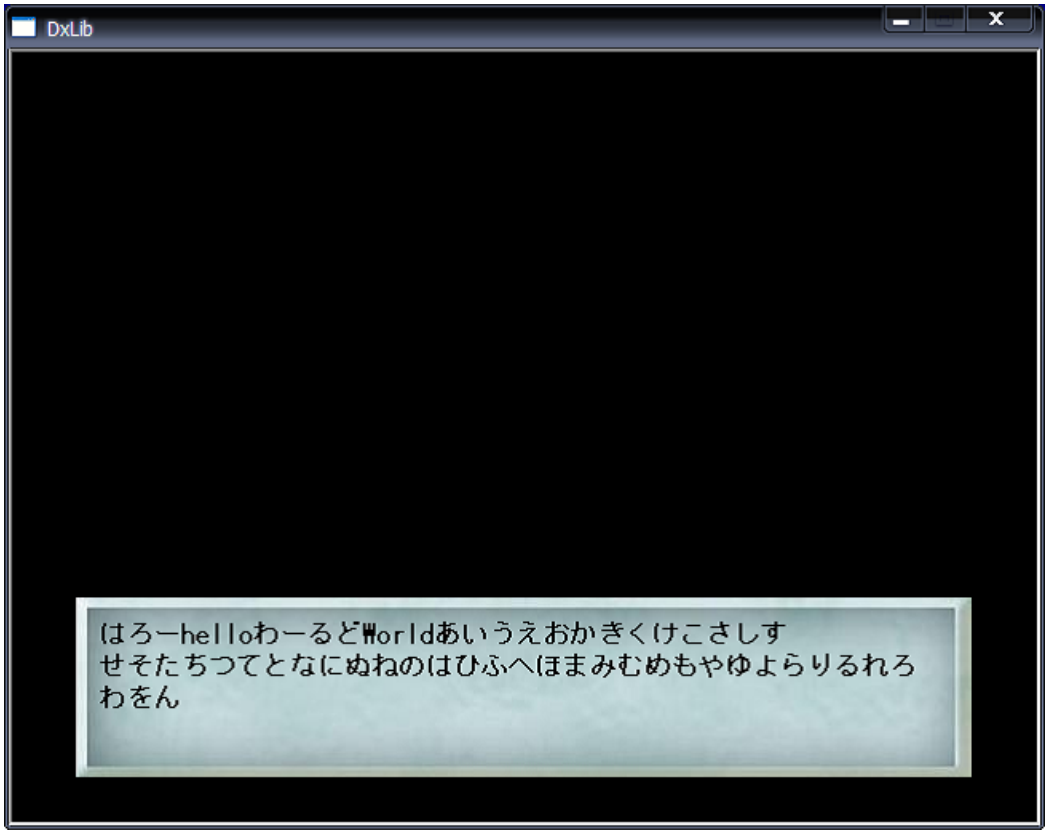

図 6.4: メッセージボックスの上にメッセージを表示

6.9 サウンドノベル風メッセージ表示プログラム

やっとサウンドノベル風メッセージ表示プログラムの完成です．前節のプログラムでほぼ完成したようなものですが，ここで少し手を加えてみます．

表示したい文字をボタンを押すごとに変えるプログラムを作ってみたいと思います．いままでのプログラムは g_message に格納された文字を描画していただけわけですが，この g_message の中身を書き換えれば，思い通りのメッセージを画面に表示させることができます．今回は F1, F2, F3 キーを押すとメッセージが切り替わるようにしましょう．

また，描画したいメッセージが存在しない場合はメッセージボックスを表示しないようにしてみます．

プログラムは以下のようになりました．今回は関数の中身も省略せずに書いています．少々長いですが，頑張って読んでみてください．

リスト 6.9: ”message-gui-05.cpp”

```
#include "DxLib.h"

//注意点：文字コードはSHIFT_JIS（Windows標準）を前提としている
//       他の文字コードでは正常に動作しない
//SHIFT_JISの場合，日本語は2バイトで表される
//上位バイトが0x81～0x9F、0xE0～0xFCの範囲に収まる

int isJapaneseCharacter(unsigned char code);
void writeSubstring(char* message, int start, int len,
     int posX, int posY, int color, int bufferLine);
void drawMessage();
void initGame();

//メッセージのフォントの大きさ
#define MESSAGE_FONT_SIZE 20
//仮想バッファの最大文字数
#define MESSAGE_MAX_LENGTH 30
//仮想バッファの最大行数
#define MESSAGE_MAX_LINE 5
//メッセージボックスのX座標
#define MESSAGE_BOX_X_POS 40
//メッセージボックスのY座標
#define MESSAGE_BOX_Y_POS 340
//メッセージボックスの画像ファイル
#define MESSAGE_BOX_GRAPHIC_FILENAME "./Pic/boxd3.jpg"

//表示したいメッセージ
char g_message[MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE];

//画面にメッセージを表示する際にしようする仮想テキストバッファ
char g_messageBuffer[MESSAGE_MAX_LINE][MESSAGE_MAX_LENGTH];

//メッセージボックス関係

//現在何文字目までを表示しているか
static int g_currentCursor = 0;
//何行目の文字を表示しているか
static int g_currentLineCursor = 0;
//白
static int g_whiteColor;
//黒
static int g_blackColor;
//メッセージボックスの画像
static int g_messageBoxGraphicHandle;
```

```
45
46
47  //code が日本語であるか判定する
48  //戻り値 1:日本語 0:日本語ではない
49  int isJapaneseCharacter(unsigned char code)
50  {
51    if( (code >= 0x81 && code <= 0x9F) ||
52      (code >= 0xE0 && code <= 0xFC) ) {
53        return 1;
54    }
55    return 0;
56  }
57
58
59  //messageで指定した文章を start の位置から len 文字分表示する
60  void writeSubstring(char* message, int start, int len,
61       int posX, int posY, int color, int bufferLine)
62  {
63    int i;
64    //文字数
65    int maxLen = strlen( message );
66
67    //startの位置を変更する
68    //startの位置までに日本語がでてきていたら, 1を足していく
69    for( i = 0; i < start && message[i] != '\0'; ) {
70      if( isJapaneseCharacter( message[i] ) ) {
71        //日本語の場合, 2バイト分すすめる
72        i += 2;
73        //startに1バイト分足す
74        start++;
75      }else {
76        //半角文字の場合, 1バイト分進める
77        i++;
78      }
79    }
80
81    //startの位置が表示したい最大文字数より大きい場合
82    if( start >= maxLen ) {
83      return;
84    }
85
86    //指定した位置からlen文字分表示する
87    for( i = 0; i < len && message[ start + i ] != '\0'; ) {
88      if( isJapaneseCharacter( message[ start + i ] ) ) {
89        //日本語の場合, 2文字分bufferにセット
90        g_messageBuffer[ bufferLine ][ i ] = message[ start + i ];
91        g_messageBuffer[ bufferLine ][ i + 1 ] = message[ start + i + 1 ];
92        //lenは日本語なので, 1バイト分追加する
93        len++;
94        //2バイト分進める
95        i += 2;
96      }else {
97        //半角文字1文字をセット
98        g_messageBuffer[ bufferLine ][ i ] = message[ start + i ];
99        //1バイト分進める
100       i++;
101     }
102   }
103   g_messageBuffer[ bufferLine ][i] = '\0';
104
105   //メッセージ描画
106   DrawString(posX, posY, g_messageBuffer[ bufferLine ], color );
107 }
108
109
```

```
110 //メッセージ描画
111 void drawMessage()
112 {
113   int i;
114 
115   //文字が1文字もセットされていなかったらメッセージボックスを表示しない
116   if( strnlen(g_message, MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE ) <= 0 ) {
117     return;
118   }
119 
120   //メッセージボックス描画
121   DrawGraph( MESSAGE_BOX_X_POS, MESSAGE_BOX_Y_POS, g_messageBoxGraphicHandle, FALSE );
122 
123   if( g_message[g_currentCursor] != '\0' ) {
124     g_currentCursor++;
125   }
126 
127   //MESSAGE_MAX_LENGTH まで文字を描画したら段落を切り替える
128   if( g_currentCursor % MESSAGE_MAX_LENGTH == 0 ) {
129     if( g_message[g_currentCursor] != '\0' ) {
130       g_currentLineCursor++;
131     }
132   }
133 
134   //メッセージ描画部分
135   for( i = 0; i < MESSAGE_MAX_LINE; i++ ) {
136     if( i == g_currentLineCursor ) {
137       //メッセージ風に表示
138       writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
139                           g_currentCursor - MESSAGE_MAX_LENGTH * i,
140         MESSAGE_BOX_X_POS + 15,
141                                 MESSAGE_BOX_Y_POS + MESSAGE_FONT_SIZE * i + 15,
142                                 g_blackColor, i );
143       break;
144     }else {
145       //メッセージをそのまま表示
146       writeSubstring( g_message, i * MESSAGE_MAX_LENGTH ,
147                           MESSAGE_MAX_LENGTH, MESSAGE_BOX_X_POS + 15,
148         MESSAGE_BOX_Y_POS + MESSAGE_FONT_SIZE * i + 15,
149                                 g_blackColor, i );
150     }
151   }
152 }
153 
154 //初期化処理
155 void initGame()
156 {
157   //白
158   g_whiteColor = GetColor(255,255,255);
159   //黒
160   g_blackColor = GetColor(0, 0, 0);
161   //メッセージボックス
162   g_messageBoxGraphicHandle = LoadGraph( MESSAGE_BOX_GRAPHIC_FILENAME );
163 }
164 
165 //描画したいメッセージをセット
166 void setMessage(const char* message)
167 {
168   //カーソルを初期化
169   g_currentCursor = 0;
170   g_currentLineCursor = 0;
171 
172   //メッセージをコピー
173   strncpy( g_message, message, MESSAGE_MAX_LENGTH * MESSAGE_MAX_LINE);
174 }
```

```
175
176 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
177              LPSTR lpCmdLine, int nCmdShow )
178 {
179   //ウィンドウモードで起動
180   ChangeWindowMode( TRUE );
181   //画面の大きさは640 * 480
182   SetGraphMode( 640 , 480 , 16 ) ;
183   //DxLib初期化
184   if( DxLib_Init() == -1 ) {
185      return -1;
186   }
187
188   // 描画先を裏画面にセット
189   SetDrawScreen( DX_SCREEN_BACK ) ;
190
191   //初期化処理
192   initGame();
193
194   //メインループ
195   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
196
197     //画面クリア
198     ClearDrawScreen();
199
200     //F1 F2を押すとメッセージをセットする
201     //F3を押すと空の文字列をセットする
202     if( CheckHitKey( KEY_INPUT_F1 ) ) {
203       setMessage("はろーわーるど");
204     }else if( CheckHitKey( KEY_INPUT_F2 ) ) {
205       setMessage("HELLO WORLD!あいうえおかきくけこさしすせそたちつてとな"
206       "にぬねのはひふへほまみむめもやゆよらりるれろわをん");
207     }else if( CheckHitKey( KEY_INPUT_F3 ) ) {
208       setMessage("");
209     }
210
211     //メッセージ描画
212     drawMessage();
213
214     Sleep( 100 );
215     ScreenFlip();
216   }
217
218   DxLib_End();
219   return 0;
220 }
```

第7章　グラフィック管理

ゲームには欠かせないグラフィック管理部分を作っていきたいと思います．グラフィック管理とは，ゲーム画面上にどの画像を表示させるか等の情報を管理する部分です．完成イメージは図 7.1 です．

ただ単に画像を画面に貼り付けているように見えますが，色々と工夫を凝らしています．例えば，画像を画面に表示する都度，ハードディスクからイメージを読み込むと時間がかかってしまいます．そこで，あらかじめ画像をメモリ上に読み込んでおき，必要な時に表示させる方法をとっています．

また，画像を表示させる際には，フェードイン・フェードアウトを行うようにしてあります．

図 7.1: グラフィック管理

7.1 メモリ上に画像を読み込む

まずは，メモリ上に画像をあらかじめロードしておき，その後画面に表示させてみましょう．なぜこのような方法を取るのでしょうか．DX ライブラリにはファイルからその都度画像をロードして表示する LoadGraphScreen 関数が存在しています．

```
int LoadGraphScreen( int x , int y , char *GraphName , int TransFlag );
```

今回はこの関数は使わないわけですが，一応使い方を説明しておきます．引数 x, y には画像を表示したい座標を，GraphName には読み込む画像を，TransFlag は透過処理を行うかどうかを指定するものです．たとえば，フォルダ Pic 内の画像 kaeru1.png を座標 (50,50) に表示させるには以下のように引数を指定します．

```
LoadGraphScreen(50, 50, "./Pic/kaeru1.png", TRUE);
```

では，kaeru1.png を 10 箇所に表示させたい場合はどうでしょうか．LoadGraphScreen 関数を 10 回呼び出す事になるのですが，これはハードディスクから同じ画像を 10 回呼び出すことになり，処理時間の遅れにつながります．ハードディスクからファイルを読み込むのは，メモリからの読み込みと比べかなり処理に時間がかかってしまうものです．

そこで，ハードディスクから画像をメモリ上に読み込んでおき，そこから必要なときに画面に表示させるといった方法が取られます．

さて，あらかじめ画像をメモリ上に読み込んでおくには，LoadGraph 関数を利用します．

```
int LoadGraph( char *FileName );
```

FileName には読み込みたい画像のパスを指定します．この関数は画像の読み込みに成功した際，戻り値にグラフィックハンドルと呼ばれる画像識別番号が返ってきます（画像読み込み失敗時には-1 が返ってきます）．この識別番号を DrawGraph 関数に渡してやることで画像を表示することができます．さきほど LoadGraphScreen 関数を利用して画面に画像を表示させた方法を，LoadGraph 関数及び DrawGraph 関数を利用して書き直すと以下のようになります．

```
int kaeru = LoadGraph( "./Pic/kaeru1.png" );
DrawGraph( 50, 50, kaeru, TRUE );
```

DrawGraph 関数の第 3 引数にはグラフィックハンドル（画像識別番号）を渡します．その他は LoadGraphScreen 関数と全く同じです．

ではメモリに画像を読み込んでおき，画像に表示させるプログラムをお見せします．まずは一通りソースを眺めてみてください．ややステップ数が多いですが，頑張ってください．

リスト 7.1: ”graphic-01.cpp”

```
1 #include "DxLib.h"
2
3 typedef struct GraphicNode_tag{
```

```
4     int id;      //画像のid 利用していない場合は0がセットされている
5     int graphicHandle; //画像のグラフィックハンドル
6   } GraphicNode;
7
8   //グラフィックの最大登録数
9   #define GRAPHIC_MAX_NUM 3
10
11  //グラフィック管理
12  GraphicNode g_graphicManager[GRAPHIC_MAX_NUM];
13
14  //プロトタイプ宣言
15  void initGraphicNode();
16  int addGraphicNode(int id, const char* graphFilename);
17
18  //初期化
19  void initGraphicNode()
20  {
21    //グラフィック管理初期化
22    memset( &g_graphicManager, 0, sizeof(GraphicNode) * GRAPHIC_MAX_NUM );
23
24    //デバッグ用にコンソールを呼び出す
25    AllocConsole();
26    freopen("CONOUT$", "w", stdout);
27    freopen("CONIN$", "r", stdin);
28
29    //本当は使い終わったらFreeConsole()を呼ばなければならない
30    //ここでは省略
31  }
32
33  //グラフィックを読み込む
34  //戻り値 -1: 失敗 0: 成功
35  int addGraphicNode(int id, const char* graphFilename)
36  {
37    int i;
38    //idが重複していないか確認
39    for(i = 0; i < GRAPHIC_MAX_NUM ; i++) {
40      if( id == g_graphicManager[i].id ) {
41        printf("idが重複しています(id %d)\n", id);
42        return -1;
43      }
44    }
45
46    //利用していないノードを見つける
47    for(i = 0; i < GRAPHIC_MAX_NUM; i++) {
48      if( g_graphicManager[i].id == 0 ) {
49        break;
50      }
51    }
52    //グラフィックノードの空きがない
53    if( i == GRAPHIC_MAX_NUM ) {
54      printf("グラフィックノードの空きがありません(id %d)\n", id);
55      return -1;
56    }
57
58    //idをセット
59    g_graphicManager[i].id = id;
60    //画像読み込み
61    g_graphicManager[i].graphicHandle = LoadGraph( graphFilename );
62
63    //読み込み失敗時
64    if( g_graphicManager[i].graphicHandle == -1 ) {
65      printf("画像読み込みに失敗しました(id %d)\n", id);
66      //グラフィックノードを空き状態にする
67      g_graphicManager[i].id = 0;
68      g_graphicManager[i].graphicHandle = 0;
```

```
69      return -1;
70    }
71    return 0;
72  }
73
74  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
75                LPSTR lpCmdLine, int nCmdShow )
76  {
77    //ウィンドウモードで起動
78    ChangeWindowMode( TRUE );
79    //画面の大きさは640 * 480
80    SetGraphMode( 640 , 480 , 16 ) ;
81    //DxLib初期化
82    if( DxLib_Init() == -1 ) {
83       return -1;
84    }
85
86    // 描画先を裏画面にセット
87    SetDrawScreen( DX_SCREEN_BACK ) ;
88
89    //初期化処理
90    initGraphicNode();
91
92    //画像追加
93    addGraphicNode(1, "./pic/kaeru1.png");
94    addGraphicNode(1, "./pic/kaeru2.png"); //重複チェック
95    addGraphicNode(2, "./pic/kaeru2.png");
96    addGraphicNode(3, "./pic/kaeru3.png"); //存在しない
97    addGraphicNode(3, "./pic/kaeru2.png");
98    addGraphicNode(4, "./pic/kaeru2.png"); //ノード不足
99
100   int i;
101
102   //メインループ
103   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
104
105     //画面クリア
106     ClearDrawScreen();
107
108     for(i = 0; i < GRAPHIC_MAX_NUM; i++ ) {
109       //描画可能なグラフィックノードが存在するとき
110       if( g_graphicManager[i].id != 0 ) {
111         //とりあえず適当な場所に描画
112         DrawGraph(30, i * 70, g_graphicManager[i].graphicHandle, TRUE);
113       }
114     }
115     ScreenFlip();
116   }
117
118   DxLib_End();
119   return 0;
120 }
```

今回はGraphicNodeという構造体を作りました．この構造体は，画像のIDとグラフィックハンドル（画像識別番号）をセットにしたものです．また，グラフィックノードを配列(g_graphicManager)とし，IDをキーとして，グラフィックハンドルを呼び出すような仕組みを作っていきます（図7.2）．

このようなデータ構造は，マップと呼ばれます．マップはキーと値のペアであり，キーに対して値が一意に定まるような構造をしています．よってIDが重複することは決してありません．マップを実装するためには，さまざまなデータ構造が使われます．単純なマップは単方向リストで実装されていたり，ハッシュ関数を利用したものなども存在します．

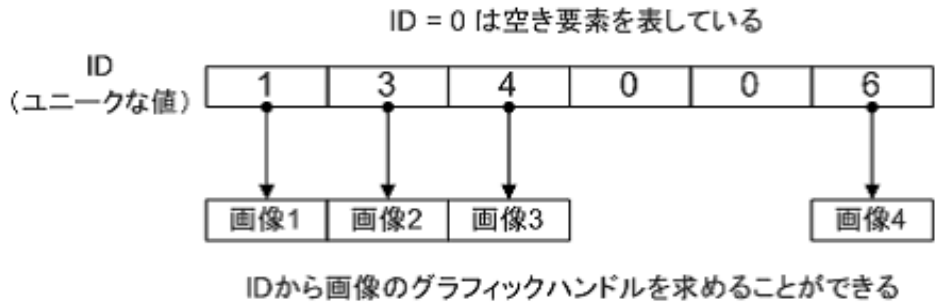

図 7.2: GraphicNode を利用したマップ構造

詳しくはアルゴリズムの部をご覧ください.

さて，addGraphicNode 関数は GraphicNode の要素を登録するためのものです．id は重複して登録できない様に工夫しています．id に 0 がセットされている時はデータが登録されていないものと見なすようにしています．よって，新たにノードを追加する際は，g_graphicManager 配列の先頭要素から id が 0 の場所を見つけ，その場所にデータを追加するようにしましょう．もし，g_graphicManager の最後の要素まで検索しても利用可能な場所が確保できない場合は，登録失敗とします (図 7.3）.

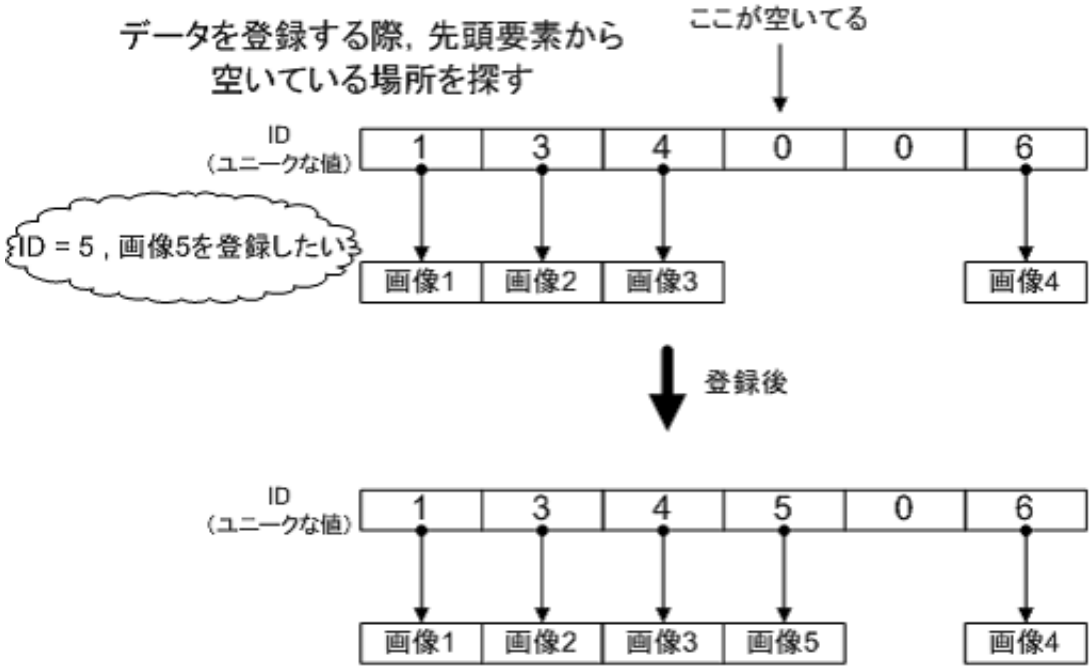

図 7.3: 画像データの登録

今はあまり気にしなくてもよいですが，実はこのような方法では問題がたくさんあります．まず，g_graphicManager を配列で実現している点です．これでは，配列の要素数分しか画像が登録できないことになります．また，利用していないノードを見つける方法も，先頭から要素を検索していく方法では効率が悪いです．このような時には，データ構造のリンクリストや，データ検索時にはハッシュ等を使うべきですが，いきなりそのような設計とするとプログラムが複雑になってしまうので，今回はそれらのアルゴリズムは利用していません．この章の最後には改良のポイントを示してあります．興味のある方は，ヒントを参照しながら，適切なアルゴリズムを利用してプログラムを作成してみてください．

さて，addGraphicNode 関数では，printf 関数をデバッグメッセージを表示するために利用しています．しかし，DX ライブラリは，標準ではコンソール画面を表示させることができません．そこで，initGraphicNode 関数では，コンソールが利用できるよう以下のようなコードを書いています．

```
AllocConsole();
freopen("CONOUT$", "w", stdout);
freopen("CONIN$", "r", stdin);
```

このコードの意味は理解しなくてかまいません．とにかく，このようにすればコンソールを呼び出すことができます．

では，main 関数内で正しい動作をするか確認していきましょう．画像をただメモリにロードしただけでは，正しく動作しているのかが見えづらいです．そこで，今回はメインループ内で読み込んだ画像を適当な場所に表示させています．

実行結果は図 7.4 及び以下のようになります．

実行結果

```
id が重複しています (id 1)
画像読み込みに失敗しました (id 3)
グラフィックノードの空きがありません (id 4)
```

7.2 画面に表示する画像の管理

前節で画像をメモリ上に読み込む仕組みを作ることができました．今回は，読み込んだ画像を画面に表示させる仕組みを作っていきましょう．

画面に画像を表示するためには，画像を表示する位置情報が必要です．そこで，位置情報とグラフィックハンドルを持った構造体，VisibleGraphicNode を作っていきましょう．また，これらのノードを管理する配列，g_visibleGraphic も作成します．

では，画面に表示する画像を管理するプログラムをお見せします．

リスト 7.2: ”graphic-02.cpp”

```
1 #include "DxLib.h"
2
3 //グラフィックを管理
```

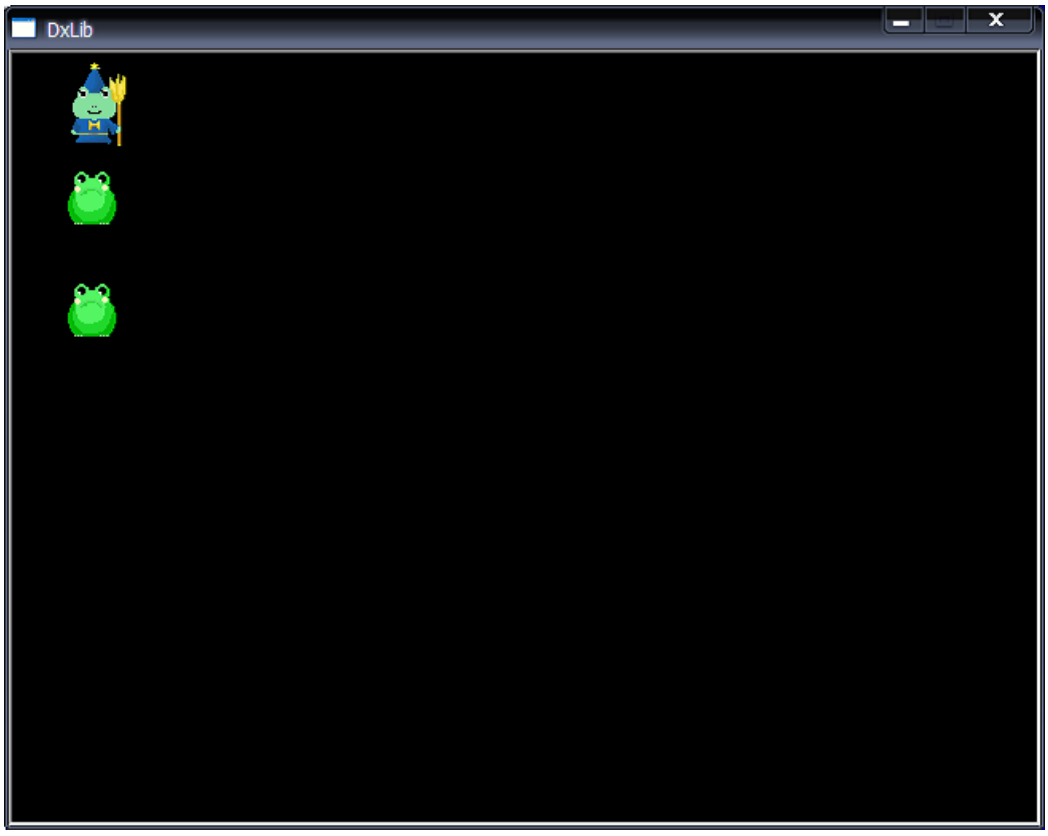

図 7.4: メモリ上の画像を表示するプログラム

```
4  typedef struct GraphicNode_tag{
5    int id;       //画像のid 利用していない場合は0がセットされている
6    int graphicHandle; //画像のグラフィックハンドル
7  } GraphicNode;
8
9  //画面に表示するグラフィックを管理
10 typedef struct VisibleGraphicNode_tag{
11   int graphicId; //画像のid GraphicNodeのidとは別物なので注意
12               //利用していない場合は0がセットされている
13   int graphicHandle; //画像のグラフィックハンドル
14   int x,y; //表示する座標
15 } VisibleGraphicNode;
16
17
18 //グラフィックの最大登録数
19 #define GRAPHIC_MAX_NUM 3
20 //画面に表示できる最大の画像数
21 #define VISIBLE_GRAPHIC_MAX_NUM 10
22
23 //グラフィック管理
24 GraphicNode g_graphicManager[GRAPHIC_MAX_NUM];
25
26 //表示するグラフィックの管理
27 VisibleGraphicNode g_visibleGraphic[VISIBLE_GRAPHIC_MAX_NUM];
28
29 //プロトタイプ宣言
30 void initGraphicNode();
31 int addGraphicNode(int id, const char* graphFilename);
32 int getGraphicHandle(int id);
33
34 int addVisibleGraphic(int id, int graphicId, int posX, int posY);
35 void drawVisibleGraphic();
36
37 //初期化
38 void initGraphicNode()
39 {
40   //グラフィック管理初期化
41   memset( &g_graphicManager, 0, sizeof(GraphicNode) * GRAPHIC_MAX_NUM );
42   //表示するグラフィックの管理初期化
43   memset( &g_visibleGraphic, 0, sizeof(VisibleGraphicNode) * VISIBLE_GRAPHIC_MAX_NUM );
44
45   //デバッグ用にコンソールを呼び出す
46   AllocConsole();
47   freopen("CONOUT$", "w", stdout);
48   freopen("CONIN$", "r", stdin);
49
50   //本当は使い終わったらFreeConsole()を呼ばなければならない
51   //ここでは省略
52 }
53
54 //グラフィックを読み込む
55 //戻り値 -1: 失敗 0: 成功
56 int addGraphicNode(int id, const char* graphFilename)
57 {
58   //省略（前回と同じ）
59 }
60
61 //idからグラフィックハンドルを取得する
62 int getGraphicHandle(int id)
63 {
64   int i = 0;
65   for(i = 0; i < GRAPHIC_MAX_NUM; i++ ) {
66     if( id == g_graphicManager[i].id ) {
67       return g_graphicManager[i].graphicHandle;
68     }
```

```
69    }
70    return -1;
71  }
72
73
74  //画面に画像を表示する
75  //idにはGraphicNodeのidを指定,  graphicIdはいまから描画する画像にidを付ける
76  //posX, posYは画像を表示する位置
77  //戻り値 -1: 失敗 0: 成功
78  int addVisibleGraphic(int id, int graphicId, int posX, int posY)
79  {
80    int i;
81    //idからグラフィックハンドルを取得
82    int handle = getGraphicHandle( id );
83
84    //graphicHandleの取得失敗
85    if( handle == -1 ) {
86      printf("ID: %d の画像は登録されていません\n", id);
87      return -1;
88    }
89
90    //graphicIdが重複していないか確認
91    for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM ; i++) {
92      if( graphicId == g_visibleGraphic[i].graphicId ) {
93        printf("graph idが重複しています(id %d)\n", graphicId);
94        return -1;
95      }
96    }
97
98    //利用していないノードを見つける
99    for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++) {
100     if( g_visibleGraphic[i].graphicId == 0 ) {
101       break;
102     }
103   }
104
105   //ノードの空きがない
106   if( i == VISIBLE_GRAPHIC_MAX_NUM ) {
107     printf("画面にこれ以上画像を表示できません(id %d)\n", graphicId);
108     return -1;
109   }
110
111   //情報登録
112   //グラフィックハンドル登録
113   g_visibleGraphic[i].graphicHandle = handle;
114   //graphicIdを登録
115   g_visibleGraphic[i].graphicId = graphicId;
116   //座標を登録
117   g_visibleGraphic[i].x = posX;
118   g_visibleGraphic[i].y = posY;
119
120   return 0;
121 }
122
123 //画像描画
124 void drawVisibleGraphic()
125 {
126   int i;
127   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++ ) {
128     //graphicIdが0でなければ（画像が存在すれば）
129     if( g_visibleGraphic[i].graphicId != 0 ) {
130       //指定した座標に画像描画
131       DrawGraph( g_visibleGraphic[i].x, g_visibleGraphic[i].y,
132         g_visibleGraphic[i].graphicHandle, TRUE);
133     }
```

```
134   }
135 }
136
137
138 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
139                LPSTR lpCmdLine, int nCmdShow )
140 {
141   //ウィンドウモードで起動
142   ChangeWindowMode( TRUE );
143   //画面の大きさは640 * 480
144   SetGraphMode( 640 , 480 , 16 ) ;
145   //DxLib初期化
146   if( DxLib_Init() == -1 ) {
147      return -1;
148   }
149
150   // 描画先を裏画面にセット
151   SetDrawScreen( DX_SCREEN_BACK ) ;
152
153   //初期化処理
154   initGraphicNode();
155
156   //画像追加
157   addGraphicNode(1, "./pic/kaeru1.png");
158   addGraphicNode(2, "./pic/kaeru2.png");
159   addGraphicNode(3, "./pic/kaeru2.png");
160
161   //表示する画像追加
162   addVisibleGraphic(1, 1, 10, 10);
163   addVisibleGraphic(2, 2, 50, 10);
164   addVisibleGraphic(3, 3, 50, 50);
165   //同じgraphicIdを2回登録(エラー)
166   addVisibleGraphic(1, 3, 10, 10);
167   //存在しないgraphicNodeのidを指定（エラー）
168   addVisibleGraphic(4, 4, 10, 10);
169   //画像をたくさん追加
170   addVisibleGraphic(1, 4, 100, 50);
171   addVisibleGraphic(1, 5, 200, 50);
172   addVisibleGraphic(1, 6, 300, 50);
173   addVisibleGraphic(1, 7, 100, 200);
174   addVisibleGraphic(1, 8, 200, 200);
175   addVisibleGraphic(1, 9, 300, 200);
176   addVisibleGraphic(2, 10,350, 300);
177   //これ以上画像を追加できない
178   addVisibleGraphic(2, 11, 300, 300);
179
180   //メインループ
181   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
182
183     //画面クリア
184     ClearDrawScreen();
185
186     //画像描画
187     drawVisibleGraphic();
188
189     ScreenFlip();
190   }
191
192   DxLib_End();
193   return 0;
194 }
```

画像の位置情報を持った VisibleGraphicNode の構造も，前回の GraphicNode と同じようにマップ構造となっています．ここで，VisibleGraphicNode の持っている graphicId と，

GraphicNode の id とは別物なので注意してください．graphicId は画像の位置情報及びグラフィックハンドルのキーとなるものです．

addVisibleGraphic 関数は画面に表示する画像を登録するためのものです．登録された画像は g_visibleGraphic 配列で管理されています．第 1 引数の id には GraphicNode の id を，第 2 引数には VisibleGraphicNode のキーとなる graphicId を，第 3，第 4 引数には画像を表示する場所を指定します．

drawVisibleGraphic 関数は，g_visibleGraphic 配列に保存されている画像を画面に表示するためのものです．

main 関数では，ただしくプログラムが動作するか確認のテストを行っています．このプログラムの実行結果は図 7.5 及び以下のようになります．

実行結果

```
graph id が重複しています (id 3)
ID: 4 の画像は登録されていません
画面にこれ以上画像を表示できません (id 11)
```

図 7.5: 画面への画像の表示

7.3 画面に表示された画像の削除

今回は，前節で addVisibleGraphic 関数を利用して画面上に描画した画像を削除する関数を作ってみたいと思います．特に難しい処理はありません．単に graphicId を 0 に書き換えれば画像は消えた事になるのです．

では，画像削除機能を書き加えたプログラムをお見せします．

リスト 7.3: ”graphic-03.cpp”

```
#include "DxLib.h"

//グラフィックを管理
typedef struct GraphicNode_tag{
  int id;     //画像のid 利用していない場合は0がセットされている
  int graphicHandle; //画像のグラフィックハンドル
} GraphicNode;

//画面に表示するグラフィックを管理
typedef struct VisibleGraphicNode_tag{
  int graphicId; //画像のid GraphicNodeのidとは別物なので注意
        //利用していない場合は0がセットされている
  int graphicHandle; //画像のグラフィックハンドル
  int x,y; //表示する座標
} VisibleGraphicNode;


//グラフィックの最大登録数
#define GRAPHIC_MAX_NUM 3
//画面に表示できる最大の画像数
#define VISIBLE_GRAPHIC_MAX_NUM 10

//グラフィック管理
GraphicNode g_graphicManager[GRAPHIC_MAX_NUM];

//表示するグラフィックの管理
VisibleGraphicNode g_visibleGraphic[VISIBLE_GRAPHIC_MAX_NUM];

//プロトタイプ宣言
void initGraphicNode();
int addGraphicNode(int id, const char* graphFilename);
int getGraphicHandle(int id);

int addVisibleGraphic(int id, int graphicId, int posX, int posY);
void drawVisibleGraphic();
int removeVisibleGraphic(int graphicId);

//初期化
void initGraphicNode()
{
  //省略（前回と同じ）
}

//グラフィックを読み込む
//戻り値 -1: 失敗 0: 成功
int addGraphicNode(int id, const char* graphFilename)
{
  //省略（前回と同じ）
}

//idからグラフィックハンドルを取得する
int getGraphicHandle(int id)
{
  //省略（前回と同じ）
```

```
55  }
56
57
58  //画面に画像を表示する
59  //idにはGraphicNodeのidを指定， graphicIdはいまから描画する画像にidを付ける
60  //posX, posYは画像を表示する位置
61  //戻り値 -1: 失敗 0: 成功
62  int addVisibleGraphic(int id, int graphicId, int posX, int posY)
63  {
64    //省略（前回と同じ）
65  }
66
67  //指定したgraphicIdの画像を削除する
68  //戻り値 -1: 失敗 0:成功
69  int removeVisibleGraphic(int graphicId)
70  {
71    int i;
72    for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM ; i++) {
73      if( graphicId == g_visibleGraphic[i].graphicId ) {
74        //指定したグラフィックが見つかった
75        //データを削除
76        g_visibleGraphic[i].graphicHandle = 0;
77        g_visibleGraphic[i].graphicId = 0;
78        g_visibleGraphic[i].x = 0;
79        g_visibleGraphic[i].y = 0;
80        return 0;
81      }
82    }
83    return -1;
84  }
85
86
87  //画像描画
88  void drawVisibleGraphic()
89  {
90    //省略（前回と同じ）
91  }
92
93
94  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
95                LPSTR lpCmdLine, int nCmdShow )
96  {
97    //ウィンドウモードで起動
98    ChangeWindowMode( TRUE );
99    //画面の大きさは640 * 480
100   SetGraphMode( 640 , 480 , 16 ) ;
101   //DxLib初期化
102   if( DxLib_Init() == -1 ) {
103      return -1;
104   }
105
106   // 描画先を裏画面にセット
107   SetDrawScreen( DX_SCREEN_BACK ) ;
108
109   //初期化処理
110   initGraphicNode();
111
112   //画像追加
113   addGraphicNode(1, "./pic/kaeru1.png");
114   addGraphicNode(2, "./pic/kaeru2.png");
115   addGraphicNode(3, "./pic/kaeru2.png");
116
117   //表示する画像追加
118   addVisibleGraphic(1, 1, 10, 10);
119   addVisibleGraphic(2, 2, 50, 10);
```

```
120   addVisibleGraphic(3, 3, 50, 50);
121   addVisibleGraphic(1, 4, 100, 50);
122   addVisibleGraphic(1, 5, 200, 50);
123   addVisibleGraphic(1, 6, 300, 50);
124   addVisibleGraphic(1, 7, 100, 200);
125   addVisibleGraphic(1, 8, 200, 200);
126   addVisibleGraphic(1, 9, 300, 200);
127   addVisibleGraphic(2, 10,350, 300);
128   //これ以上画像を追加できない
129   addVisibleGraphic(2, 11, 300, 300);
130   //画像を削除
131   removeVisibleGraphic(4);
132   removeVisibleGraphic(5);
133   removeVisibleGraphic(6);
134   removeVisibleGraphic(7);
135   removeVisibleGraphic(8);
136   removeVisibleGraphic(9);
137   removeVisibleGraphic(10);
138   //こんどは画像を登録できる
139   addVisibleGraphic(1, 4, 300, 300);
140
141
142   //メインループ
143   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
144
145     //画面クリア
146     ClearDrawScreen();
147
148     //画像描画
149     drawVisibleGraphic();
150
151     ScreenFlip();
152   }
153
154   DxLib_End();
155   return 0;
156 }
```

今回追加した部分は removeVisibleGraphic 関数だけです．このプログラムの実行結果は図 7.6 及び以下のようになりました．

実行結果

画面にこれ以上画像を表示できません (id 11)

7.4 画像のフェードイン・フェードアウト

画像を画面に表示する際，だんだん明るくしていったり，逆に画像を画面から削除するときはだんだん暗くしていったりといったフェードイン，フェードアウト効果を付けてみましょう．画像の明るさ（輝度）を変えるには SetDrawBright 関数を利用します．

```
int SetDrawBright( int RedBright , int GreenBright , int BlueBright );
```

RedBright, GreenBright, BlueBright はそれぞれ赤，緑，青の輝度を表しています．指定できる値は，0〜255 の範囲です．0 は真っ暗，255 は最高輝度です．

さて，フェードイン・フェードアウトを行うプログラムをお見せします．特に難しいことはありません．時間が立つにつれて，ただ輝度を少しずつ変えていけばいいのです．

図 7.6: 画像の削除機能を備えたプログラム

リスト 7.4: ”graphic-04-01.cpp”

```
#include "DxLib.h"


int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
             LPSTR lpCmdLine, int nCmdShow )
{
  int i;
  int graphicHandle;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //画面の大きさは800 * 600
  SetGraphMode( 800, 600, 16 ) ;
  //DxLib初期化
  if( DxLib_Init() == -1 ) {
     return -1;
  }

  // 描画先を裏画面にセット
  SetDrawScreen( DX_SCREEN_BACK );

  //画像読み込み
  graphicHandle = LoadGraph("./pic/man.png");

  //フェードイン
  for(i = 0; i < 255; i++ ) {
    //輝度をだんだんあげていく
    SetDrawBright(i, i, i);
    //画像描画
    DrawGraph(50,100, graphicHandle, TRUE);
    ScreenFlip();
  }

  //フェードアウト
  for(i = 255; i >= 0; i--) {
    //輝度をだんだんさげていく
    SetDrawBright(i, i, i);
    //画像描画
    DrawGraph(50,100, graphicHandle, TRUE);
    ScreenFlip();
  }

  WaitKey();

  DxLib_End();
  return 0;
}
```

フェードイン・フェードアウトはできました．しかし，背景画像の上に画像をフェードイン・フェードアウトさせた場合，どのように表示されるでしょうか．是非ご自身で試してみてください．なんだか，画像が黒くなっていくだけで違和感があります（図 7.7）．

さて，背景画像の上にだんだん画像を表示させていく，またはだんだん画像を消していく為には，アルファブレンドと呼ばれる手法を使います．アルファブレンドを用いると，画像が透けている感じを出すことができます (図 7.8）．

アルファブレンドを行うには SetDrawBlendMode 関数を利用します．

```
int SetDrawBlendMode( int BlendMode , int Pal );
```

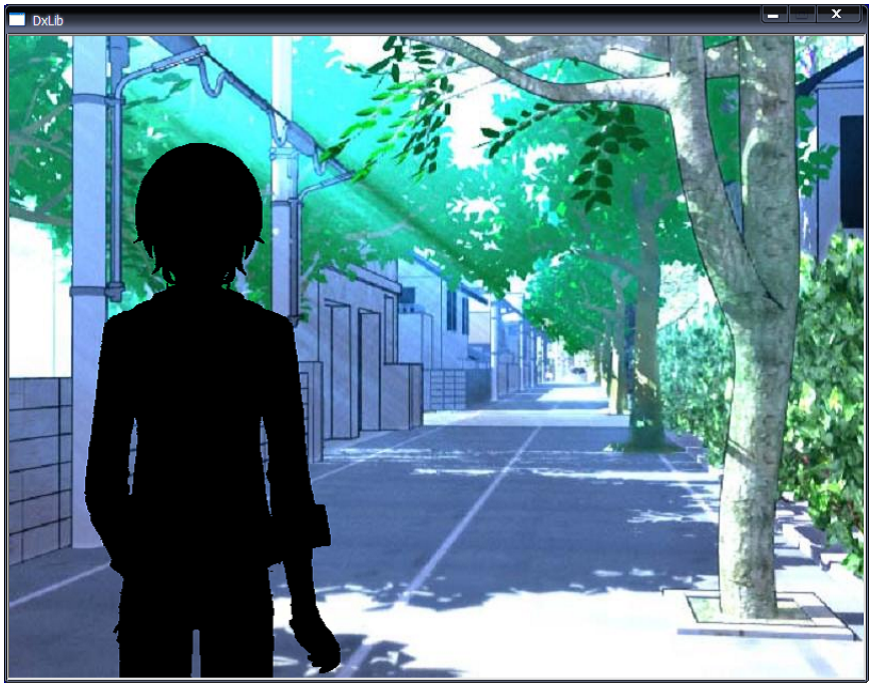

図 7.7: 違和感のある画像のフェードアウト

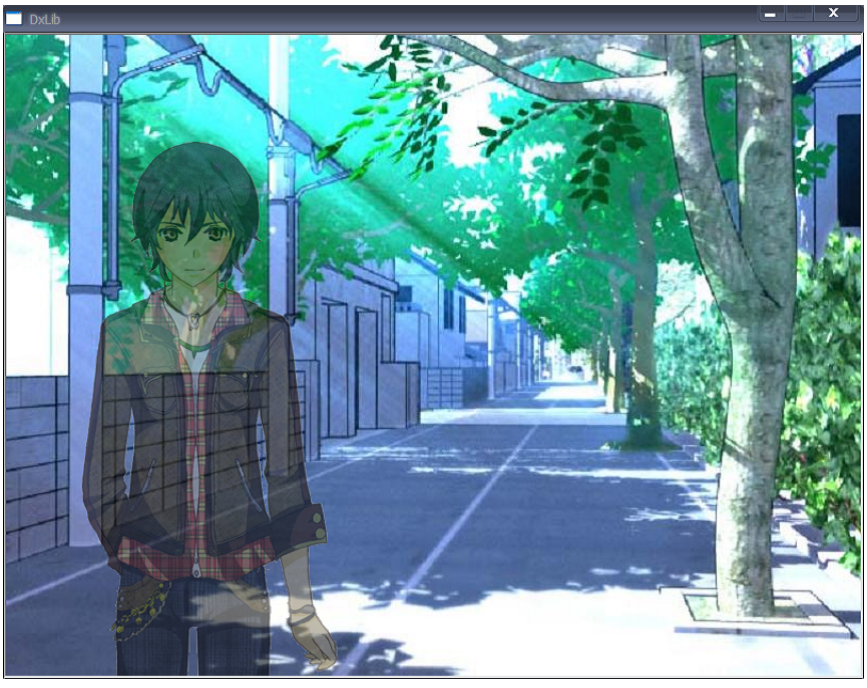

図 7.8: アルファブレンドを利用した画像のフェードアウト

BlendMode には描画するブレンドモードを指定することができます．アルファブレンドを利用するには，DX_BLENDMODE_ALPHA を指定します．Pal にはブレンドモードのパラメータ（範囲 0〜255) を指定します．ブレンドモードを元に戻すには，BlendMode に DX_BLENDMODE_NOBLEND を渡します．

では，アルファブレンドを利用したプログラムをお見せします．

リスト 7.5: ”graphic-04-02.cpp”

```
#include "DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
             LPSTR lpCmdLine, int nCmdShow )
{
  int i;
  int graphicHandle, backHandle;

  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //画面の大きさは800 * 600
  SetGraphMode( 800, 600, 16 ) ;
  //DxLib初期化
  if( DxLib_Init() == -1 ) {
     return -1;
  }

  // 描画先を裏画面にセット
  SetDrawScreen( DX_SCREEN_BACK );
  //画像読み込み
  graphicHandle = LoadGraph("./pic/man.png");
  //背景を描画
  backHandle = LoadGraph("./pic/back.png");

  //フェードイン
  for(i = 0; i < 255; i++ ) {
    //画面削除
    ClearDrawScreen();

    //背景表示
    DrawGraph( 0, 0, backHandle, FALSE );
    //輝度をだんだんあげていく
    SetDrawBright(i, i, i);
    //アルファブレンドを行う
    SetDrawBlendMode(DX_BLENDMODE_ALPHA , i );
    //画像描画
    DrawGraph(50,100, graphicHandle, TRUE);

    //輝度，アルファブレンドの情報を元に戻す
    SetDrawBright(255, 255, 255);
    SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );

    ScreenFlip();
  }

  //フェードアウト
  for(i = 255; i >= 0; i--) {
    //画面削除
    ClearDrawScreen();

    //背景表示
    DrawGraph( 0, 0, backHandle, FALSE );
    //輝度をだんだんさげていく
    SetDrawBright(i, i, i);
    //アルファブレンドを行う
```

```
56      SetDrawBlendMode(DX_BLENDMODE_ALPHA , i );
57      //画像描画
58      DrawGraph(50,100, graphicHandle, TRUE);
59
60      //輝度，アルファブレンドの情報を元に戻す
61      SetDrawBright(255, 255, 255);
62      SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );
63
64      ScreenFlip();
65    }
66
67    WaitKey();
68
69    DxLib_End();
70    return 0;
71  }
```

7.5 輝度情報を持った画像の管理

画像を画面に表示する際，フェードインを行うようにしてみましょう．この機能を実現するために，VisibleGraphicNode 構造体に新たに輝度情報とモードを追加してみます．モードはこれから画像をフェードインするかフェードアウトするかそうでないかの情報を持っています．画像を追加した際はモードをフェードインとしておきます．フェードインモード時は，画面に画像を描画する際に輝度情報を少しずつ上げていきます．そして，輝度が最大値となったところでフェードインモードを解除します．

では，フェードイン機能を追加した画像管理プログラムをお見せします．

リスト 7.6: ”graphic-05.cpp”

```
1   #include "DxLib.h"
2
3   //グラフィックを管理
4   typedef struct GraphicNode_tag{
5     int id;     //画像のid 利用していない場合は0がセットされている
6     int graphicHandle; //画像のグラフィックハンドル
7   } GraphicNode;
8
9   //画面に表示するグラフィックを管理
10  typedef struct VisibleGraphicNode_tag{
11    int graphicId; //画像のid GraphicNodeのidとは別物なので注意
12          //利用していない場合は0がセットされている
13    int graphicHandle; //画像のグラフィックハンドル
14    int x,y; //表示する座標
15    int bright; //輝度
16    int mode; //画像描画モード 0:輝度変化無し 1:フェードイン 2:フェードアウト
17  } VisibleGraphicNode;
18
19
20  //グラフィックの最大登録数
21  #define GRAPHIC_MAX_NUM 3
22  //画面に表示できる最大の画像数
23  #define VISIBLE_GRAPHIC_MAX_NUM 10
24  //フェードイン・フェードアウトのスピード
25  #define GRAPHIC_FADEIN_FADEOUT_SPEED 10
26  //輝度の最大値
27  #define GRAPHIC_MAX_BRIGHT 255
28  //画像描画モード
29  #define GRAPHIC_MODE_NONE 0
```

```
30 #define GRAPHIC_MODE_FADEIN 1
31 #define GRAPHIC_MODE_FADEOUT 2
32
33 //グラフィック管理
34 GraphicNode g_graphicManager[GRAPHIC_MAX_NUM];
35
36 //表示するグラフィックの管理
37 VisibleGraphicNode g_visibleGraphic[VISIBLE_GRAPHIC_MAX_NUM];
38
39 //プロトタイプ宣言
40 void initGraphicNode();
41 int addGraphicNode(int id, const char* graphFilename);
42 int getGraphicHandle(int id);
43
44 int addVisibleGraphic(int id, int graphicId, int posX, int posY);
45 void drawVisibleGraphic();
46 int removeVisibleGraphic(int graphicId);
47
48 //初期化
49 void initGraphicNode()
50 {
51   //省略（前回と同じ）
52 }
53
54 //グラフィックを読み込む
55 //戻り値 -1: 失敗 0: 成功
56 int addGraphicNode(int id, const char* graphFilename)
57 {
58   //省略（前回と同じ）
59 }
60
61 //idからグラフィックハンドルを取得する
62 int getGraphicHandle(int id)
63 {
64   //省略（前回と同じ）
65 }
66
67
68 //画面に画像を表示する
69 //idにはGraphicNodeのidを指定， graphicIdはいまから描画する画像にidを付ける
70 //posX, posYは画像を表示する位置
71 //戻り値 -1: 失敗 0: 成功
72 int addVisibleGraphic(int id, int graphicId, int posX, int posY)
73 {
74   int i;
75   //idからグラフィックハンドルを取得
76   int handle = getGraphicHandle( id );
77
78   //graphicHandleの取得失敗
79   if( handle == -1 ) {
80     printf("ID: %d の画像は登録されていません\n", id);
81     return -1;
82   }
83
84   //graphicIdが重複していないか確認
85   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM ; i++) {
86     if( graphicId == g_visibleGraphic[i].graphicId ) {
87       printf("graph idが重複しています(id %d)\n", graphicId);
88       return -1;
89     }
90   }
91
92   //利用していないノードを見つける
93   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++) {
94     if( g_visibleGraphic[i].graphicId == 0 ) {
```

```
95        break;
96      }
97    }
98
99    //ノードの空きがない
100   if( i == VISIBLE_GRAPHIC_MAX_NUM ) {
101     printf("画面にこれ以上画像を表示できません(id %d)\n", graphicId);
102     return -1;
103   }
104
105   //情報登録
106   //グラフィックハンドル登録
107   g_visibleGraphic[i].graphicHandle = handle;
108   //graphicIdを登録
109   g_visibleGraphic[i].graphicId = graphicId;
110   //座標を登録
111   g_visibleGraphic[i].x = posX;
112   g_visibleGraphic[i].y = posY;
113   //輝度情報登録
114   g_visibleGraphic[i].bright = 0;
115   //モード
116   g_visibleGraphic[i].mode = GRAPHIC_MODE_FADEIN;
117
118   return 0;
119 }
120
121 //指定したgraphicIdの画像を削除する
122 //戻り値 -1: 失敗 0:成功
123 int removeVisibleGraphic(int graphicId)
124 {
125   //省略（前回と同じ）
126 }
127
128
129 //画像描画
130 void drawVisibleGraphic()
131 {
132   int i;
133   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++ ) {
134     //graphicIdが0でなければ（画像が存在すれば）
135     if( g_visibleGraphic[i].graphicId != 0 ) {
136
137       //フェードインを行うとき
138       if( g_visibleGraphic[i].mode == GRAPHIC_MODE_FADEIN ) {
139         //輝度を上げる
140         g_visibleGraphic[i].bright += GRAPHIC_FADEIN_FADEOUT_SPEED;
141         if( g_visibleGraphic[i].bright > GRAPHIC_MAX_BRIGHT ) {
142           //輝度が最大値に達した時
143           g_visibleGraphic[i].bright = GRAPHIC_MAX_BRIGHT;
144           //フェードイン解除
145           g_visibleGraphic[i].mode = GRAPHIC_MODE_NONE;
146         }
147       }
148       //輝度セット
149       SetDrawBright(g_visibleGraphic[i].bright,
150           g_visibleGraphic[i].bright, g_visibleGraphic[i].bright );
151       //アルファブレンドを行う
152       SetDrawBlendMode(DX_BLENDMODE_ALPHA , g_visibleGraphic[i].bright ) ;
153       //指定した座標に画像描画
154       DrawGraph( g_visibleGraphic[i].x, g_visibleGraphic[i].y,
155         g_visibleGraphic[i].graphicHandle, TRUE);
156     }
157   }
158   //輝度情報をデフォルトに戻しておく
159   SetDrawBright( GRAPHIC_MAX_BRIGHT, GRAPHIC_MAX_BRIGHT, GRAPHIC_MAX_BRIGHT);
```

```
160   //ブレンドモードを元にもどしておく
161   SetDrawBlendMode(DX_BLENDMODE_NOBLEND, GRAPHIC_MAX_BRIGHT);
162 }
163
164
165 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
166                LPSTR lpCmdLine, int nCmdShow )
167 {
168   //ウィンドウモードで起動
169   ChangeWindowMode( TRUE );
170   //画面の大きさは640 * 480
171   SetGraphMode( 640 , 480 , 16 ) ;
172   //DxLib初期化
173   if( DxLib_Init() == -1 ) {
174      return -1;
175   }
176
177   // 描画先を裏画面にセット
178   SetDrawScreen( DX_SCREEN_BACK ) ;
179
180   //初期化処理
181   initGraphicNode();
182
183   //画像追加
184   addGraphicNode(1, "./pic/kaeru1.png");
185   addGraphicNode(2, "./pic/kaeru2.png");
186   addGraphicNode(3, "./pic/kaeru2.png");
187
188   //表示する画像追加
189   addVisibleGraphic(1, 1, 10, 10);
190   addVisibleGraphic(2, 2, 50, 10);
191   addVisibleGraphic(3, 3, 50, 50);
192   addVisibleGraphic(1, 4, 100, 50);
193   addVisibleGraphic(1, 5, 200, 50);
194   addVisibleGraphic(1, 6, 300, 50);
195   addVisibleGraphic(1, 7, 100, 200);
196   addVisibleGraphic(1, 8, 200, 200);
197   addVisibleGraphic(1, 9, 300, 200);
198   addVisibleGraphic(2, 10,350, 300);
199   //これ以上画像を追加できない
200   addVisibleGraphic(2, 11, 300, 300);
201   //画像を削除
202   removeVisibleGraphic(4);
203   removeVisibleGraphic(5);
204   removeVisibleGraphic(6);
205   removeVisibleGraphic(7);
206   removeVisibleGraphic(8);
207   removeVisibleGraphic(9);
208   removeVisibleGraphic(10);
209   //こんどは画像を登録できる
210   addVisibleGraphic(1, 4, 300, 300);
211
212
213   //メインループ
214   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
215
216     //画面クリア
217     ClearDrawScreen();
218
219     //画像描画
220     drawVisibleGraphic();
221
222     Sleep(50);
223
224     ScreenFlip();
```

```
225   }
226
227   DxLib_End();
228   return 0;
229 }
```

7.6 グラフィック管理プログラム

ようやくグラフィック管理部分の完成です．今回は，前回のプログラムにさらに画像を削除する際，フェードアウトする機能を追加してみましょう．特に難しいことはありません．ただ，画像を削除したくなったら，モードをフェードアウトとし，輝度が 0 となった時，画像を削除すればよいのです．

では，完成したプログラムをお見せします．今回はコードを省略せず全て書いています．画像が表示された後，F1 キーを押すことで画像を削除させることができます．

リスト 7.7: ”graphic-06.cpp”

```
1  #include "DxLib.h"
2
3  //グラフィックを管理
4  typedef struct GraphicNode_tag{
5    int id;     //画像のid 利用していない場合は0がセットされている
6    int graphicHandle; //画像のグラフィックハンドル
7  } GraphicNode;
8
9  //画面に表示するグラフィックを管理
10 typedef struct VisibleGraphicNode_tag{
11   int graphicId; //画像のid GraphicNodeのidとは別物なので注意
12         //利用していない場合は0がセットされている
13   int graphicHandle; //画像のグラフィックハンドル
14   int x,y; //表示する座標
15   int bright; //輝度
16   int mode; //画像描画モード 0:輝度変化無し 1:フェードイン 2:フェードアウト
17 } VisibleGraphicNode;
18
19
20 //グラフィックの最大登録数
21 #define GRAPHIC_MAX_NUM 3
22 //画面に表示できる最大の画像数
23 #define VISIBLE_GRAPHIC_MAX_NUM 10
24 //フェードイン・フェードアウトのスピード
25 #define GRAPHIC_FADEIN_FADEOUT_SPEED 10
26 //輝度の最大値
27 #define GRAPHIC_MAX_BRIGHT 255
28 //画像描画モード
29 #define GRAPHIC_MODE_NONE 0
30 #define GRAPHIC_MODE_FADEIN 1
31 #define GRAPHIC_MODE_FADEOUT 2
32
33 //グラフィック管理
34 GraphicNode g_graphicManager[GRAPHIC_MAX_NUM];
35
36 //表示するグラフィックの管理
37 VisibleGraphicNode g_visibleGraphic[VISIBLE_GRAPHIC_MAX_NUM];
38
39 //プロトタイプ宣言
40 void initGraphicNode();
41 int addGraphicNode(int id, const char* graphFilename);
```

```
42  int getGraphicHandle(int id);
43
44  int addVisibleGraphic(int id, int graphicId, int posX, int posY);
45  void drawVisibleGraphic();
46  int removeVisibleGraphic(int graphicId);
47
48  //初期化
49  void initGraphicNode()
50  {
51    //グラフィック管理初期化
52    memset( &g_graphicManager, 0, sizeof(GraphicNode) * GRAPHIC_MAX_NUM );
53    //表示するグラフィックの管理初期化
54    memset( &g_visibleGraphic, 0, sizeof(VisibleGraphicNode) * VISIBLE_GRAPHIC_MAX_NUM );
55
56    //デバッグ用にコンソールを呼び出す
57    AllocConsole();
58    freopen("CONOUT$", "w", stdout);
59    freopen("CONIN$", "r", stdin);
60
61    //本当は使い終わったらFreeConsole()を呼ばなければならない
62    //ここでは省略
63  }
64
65  //グラフィックを読み込む
66  //戻り値 -1: 失敗 0: 成功
67  int addGraphicNode(int id, const char* graphFilename)
68  {
69    int i;
70    //idが重複していないか確認
71    for(i = 0; i < GRAPHIC_MAX_NUM ; i++) {
72      if( id == g_graphicManager[i].id ) {
73        printf("idが重複しています(id %d)\n", id);
74        return -1;
75      }
76    }
77
78    //利用していないノードを見つける
79    for(i = 0; i < GRAPHIC_MAX_NUM; i++) {
80      if( g_graphicManager[i].id == 0 ) {
81        break;
82      }
83    }
84    //グラフィックノードの空きがない
85    if( i == GRAPHIC_MAX_NUM ) {
86      printf("グラフィックノードの空きがありません(id %d)\n", id);
87      return -1;
88    }
89
90    //画像読み込み
91    g_graphicManager[i].graphicHandle = LoadGraph( graphFilename );
92
93    //読み込み失敗時
94    if( g_graphicManager[i].graphicHandle == -1 ) {
95      printf("画像読み込みに失敗しました(id %d)\n", id);
96      g_graphicManager[i].graphicHandle = 0;
97      return -1;
98    }
99
100   //idをセット
101   g_graphicManager[i].id = id;
102   return 0;
103 }
104
105 //idからグラフィックハンドルを取得する
106 int getGraphicHandle(int id)
```

```
107 {
108   int i = 0;
109   for(i = 0; i < GRAPHIC_MAX_NUM; i++ ) {
110     if( id == g_graphicManager[i].id ) {
111       return g_graphicManager[i].graphicHandle;
112     }
113   }
114   return -1;
115 }
116
117
118 //画面に画像を表示する
119 //idにはGraphicNodeのidを指定， graphicIdはいまから描画する画像にidを付ける
120 //posX, posYは画像を表示する位置
121 //戻り値 -1: 失敗 0: 成功
122 int addVisibleGraphic(int id, int graphicId, int posX, int posY)
123 {
124   int i;
125   //idからグラフィックハンドルを取得
126   int handle = getGraphicHandle( id );
127
128   //graphicHandleの取得失敗
129   if( handle == -1 ) {
130     printf("ID: %d の画像は登録されていません\n", id);
131     return -1;
132   }
133
134   //graphicIdが重複していないか確認
135   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM ; i++) {
136     if( graphicId == g_visibleGraphic[i].graphicId ) {
137       printf("graph idが重複しています(id %d)\n", graphicId);
138       return -1;
139     }
140   }
141
142   //利用していないノードを見つける
143   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++) {
144     if( g_visibleGraphic[i].graphicId == 0 ) {
145       break;
146     }
147   }
148
149   //ノードの空きがない
150   if( i == VISIBLE_GRAPHIC_MAX_NUM ) {
151     printf("画面にこれ以上画像を表示できません(id %d)\n", graphicId);
152     return -1;
153   }
154
155   //情報登録
156   //グラフィックハンドル登録
157   g_visibleGraphic[i].graphicHandle = handle;
158   //graphicIdを登録
159   g_visibleGraphic[i].graphicId = graphicId;
160   //座標を登録
161   g_visibleGraphic[i].x = posX;
162   g_visibleGraphic[i].y = posY;
163   //輝度情報登録
164   g_visibleGraphic[i].bright = 0;
165   //モード
166   g_visibleGraphic[i].mode = GRAPHIC_MODE_FADEIN;
167
168   return 0;
169 }
170
171 //指定したgraphicIdの画像を削除する
```

```
172 //戻り値 -1: 失敗 0:成功
173 int removeVisibleGraphic(int graphicId)
174 {
175   int i;
176   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM ; i++) {
177     if( graphicId == g_visibleGraphic[i].graphicId ) {
178       //指定したグラフィックが見つかった
179       //モードをフェードアウトとする
180       g_visibleGraphic[i].mode = GRAPHIC_MODE_FADEOUT;
181       return 0;
182     }
183   }
184   return -1;
185 }
186 
187 
188 //画像描画
189 void drawVisibleGraphic()
190 {
191   int i;
192   for(i = 0; i < VISIBLE_GRAPHIC_MAX_NUM; i++ ) {
193     //graphicIdが0でなければ（画像が存在すれば）
194     if( g_visibleGraphic[i].graphicId != 0 ) {
195 
196       //フェードインを行うとき
197       if( g_visibleGraphic[i].mode == GRAPHIC_MODE_FADEIN ) {
198         //輝度を上げる
199         g_visibleGraphic[i].bright += GRAPHIC_FADEIN_FADEOUT_SPEED;
200         if( g_visibleGraphic[i].bright > GRAPHIC_MAX_BRIGHT ) {
201           //輝度が最大値に達した時
202           g_visibleGraphic[i].bright = GRAPHIC_MAX_BRIGHT;
203           //フェードイン解除
204           g_visibleGraphic[i].mode = GRAPHIC_MODE_NONE;
205         }
206       }
207 
208       //フェードアウトを行うとき
209       if( g_visibleGraphic[i].mode == GRAPHIC_MODE_FADEOUT ) {
210         //輝度を下げる
211         g_visibleGraphic[i].bright -= GRAPHIC_FADEIN_FADEOUT_SPEED;
212         if( g_visibleGraphic[i].bright <= 0 ) {
213           //画像を画面から消す
214           g_visibleGraphic[i].graphicHandle = 0;
215           g_visibleGraphic[i].graphicId = 0;
216           g_visibleGraphic[i].x = 0;
217           g_visibleGraphic[i].y = 0;
218           g_visibleGraphic[i].bright = 0;
219           g_visibleGraphic[i].mode = GRAPHIC_MODE_NONE;
220           continue;
221         }
222       }
223 
224       //輝度セット
225       SetDrawBright(g_visibleGraphic[i].bright,
226           g_visibleGraphic[i].bright, g_visibleGraphic[i].bright );
227       //アルファブレンドを行う
228       SetDrawBlendMode(DX_BLENDMODE_ALPHA , g_visibleGraphic[i].bright ) ;
229 
230       //指定した座標に画像描画
231       DrawGraph( g_visibleGraphic[i].x, g_visibleGraphic[i].y,
232         g_visibleGraphic[i].graphicHandle, TRUE);
233     }
234   }
235   //輝度情報をデフォルトに戻しておく
236   SetDrawBright( GRAPHIC_MAX_BRIGHT, GRAPHIC_MAX_BRIGHT, GRAPHIC_MAX_BRIGHT);
```

```
237   //ブレンドモードを元にもどしておく
238   SetDrawBlendMode(DX_BLENDMODE_NOBLEND, GRAPHIC_MAX_BRIGHT);
239 }
240
241
242 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
243                LPSTR lpCmdLine, int nCmdShow )
244 {
245   //ウィンドウモードで起動
246   ChangeWindowMode( TRUE );
247   //画面の大きさは640 * 480
248   SetGraphMode( 640 , 480 , 16 ) ;
249   //DxLib初期化
250   if( DxLib_Init() == -1 ) {
251      return -1;
252   }
253
254   // 描画先を裏画面にセット
255   SetDrawScreen( DX_SCREEN_BACK ) ;
256
257   //初期化処理
258   initGraphicNode();
259
260   //画像追加
261   addGraphicNode(1, "./pic/kaeru1.png");
262   addGraphicNode(2, "./pic/kaeru2.png");
263   addGraphicNode(3, "./pic/kaeru2.png");
264
265   //表示する画像追加
266   addVisibleGraphic(1, 1, 10, 10);
267   addVisibleGraphic(2, 2, 50, 10);
268   addVisibleGraphic(3, 3, 50, 50);
269   addVisibleGraphic(1, 4, 100, 50);
270   addVisibleGraphic(1, 5, 200, 50);
271   addVisibleGraphic(1, 6, 300, 50);
272   addVisibleGraphic(1, 7, 100, 200);
273   addVisibleGraphic(1, 8, 200, 200);
274   addVisibleGraphic(1, 9, 300, 200);
275   addVisibleGraphic(2, 10,350, 300);
276
277
278   //メインループ
279   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
280
281     if( CheckHitKey( KEY_INPUT_F1 ) ) {
282       //画像を削除
283       removeVisibleGraphic(4);
284       removeVisibleGraphic(5);
285       removeVisibleGraphic(6);
286       removeVisibleGraphic(7);
287       removeVisibleGraphic(8);
288       removeVisibleGraphic(9);
289       removeVisibleGraphic(10);
290     }
291
292     //画面クリア
293     ClearDrawScreen();
294
295     //画像描画
296     drawVisibleGraphic();
297
298     Sleep(50);
299
300     ScreenFlip();
301   }
```

```
302
303     DxLib_End();
304     return 0;
305 }
```

実行結果は図 7.9 となります.

図 7.9: グラフィック管理プログラム

7.7 グラフィック管理プログラムの改良

グラフィック管理プログラムは前節で完成しました．しかし，§メモリ上に画像を読み込むの部分でも少し説明しましたが，このような実装では，ノベルゲームの規模が大きくなった場合，処理速度の低下等の問題が表れてくるかもしれません．

この節では，これらの問題をどのように解決したら良いのか，どのようなアルゴリズムを使ったら良いのかについて解説していきます．なお，アルゴリズムの実装方法等の詳細については，アルゴリズムの部をご覧ください．

GraphicNode や VisibleGraphicNode 構造体を管理する際，今回は配列を利用しました．しかし，配列で実装した際，要素数が固定されてしまうなどの問題があります．例えば，

今回の場合は GraphicNode に登録できる要素数は最大 3 つとなっています．よって，それ以上の画像を登録することはできないのです．

まあ，要素数を増やせば，登録できる画像数も増えるわけですが，増やしすぎても色々と問題が発生してきます．例えば，最大要素数を 10000 と設定しておくとしましょう．こうしておけば，登録できる画像数は 10000 個です．しかし，実際には画像を 10 個程度しか保存しなかった場合，残り 9990 個分のメモリが無駄となってしまいます．

また，データを登録する際，今回は先頭要素から順に空いている要素を探していくような実装としました．しかし，例えば空き要素が 1000 番目にあったとしたら，先頭要素から順に ID を比較していかなければならず，処理の効率がとても悪いです (図 7.10）．

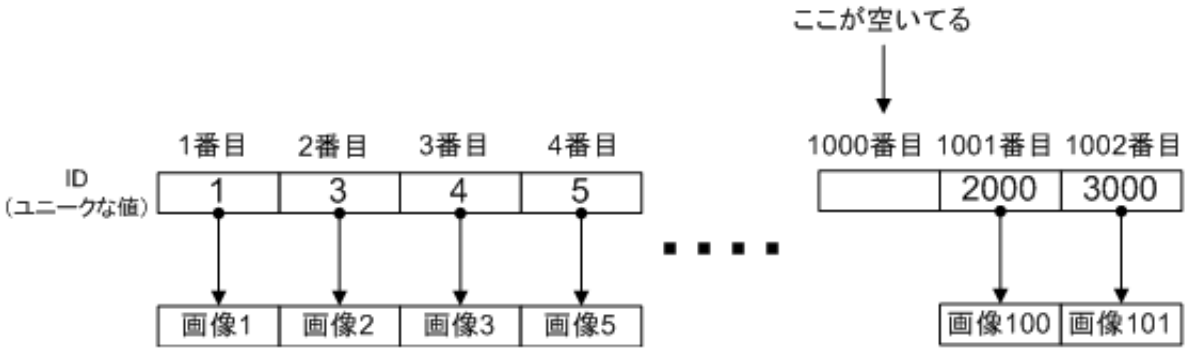

図 7.10: 空き要素を探す場合

これらの問題を解決するには，連結リスト（リンクリスト）を利用すれば良いのです．リンクリストは，要素の挿入を無制限に行うことができるもので，データをポインタで繋いだ構造をしています (図 7.11）．

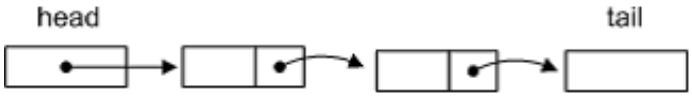

図 7.11: リンクリスト

リンクリストにデータを追加するには，tail の 1 つ前に要素を付け加えればよいだけです (図 7.12）．

このような構造を用いれば，メモリ領域の無駄が無くなり，さらに，リストの最後尾の場所 (tail) さえ記憶しておけば，データの追加もらくらく行うことができます．

リンクリストにも問題はあります．例えば，指定した ID を探したい場合はどうでしょうか．先頭要素から順に ID を検査していくことになり，これでは配列とあまり変わりません．この問題を解決するには，リンクリストを更に発展させた二分木（バイナリーツリー）

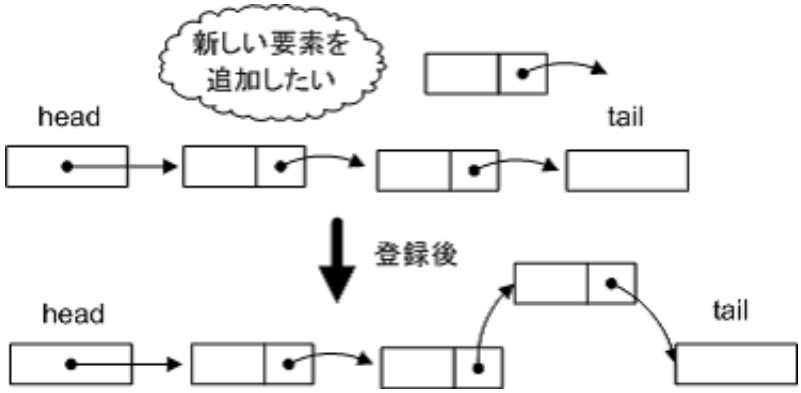

図 7.12: リンクリストへのデータの追加

を使うなどの方法が考えられます．もちろん，他にも方法はありますが，今回は二分木を取り上げることとします．二分木は図 7.13 のような構造をしています．

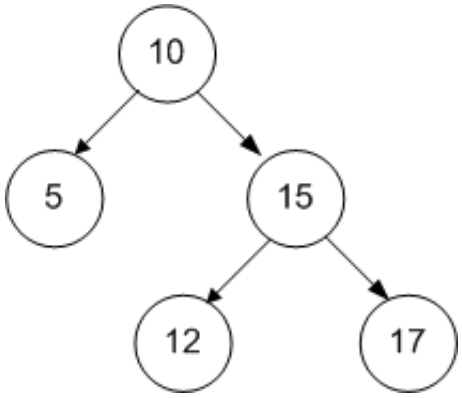

図 7.13: 二分木（二分探索木）

図 7.13 は二分木を更に発展させた二分探索木という構造をしています．二分探索木は，あるノードの左側にある子孫ノードはそのノードの値よりも小さく，右側にある子孫ノードはそのノードの値よりも大きくなるように構成されています．

例えば，ID = 12 のノードを探したい場合は，まず根となるノードの値よりも小さいか大きいかを比較します．図 7.13 の場合，根のノードは 10 なので，ID = 12 は右側の子孫ノードにあることが分かります．次に，ID = 15 を調べれば，ID = 12 は左側の子孫ノードにあることが分かります．このようなデータ構造を使うことで，データの検索を効率的に行うことができるようになります．

もちろん，二分探索木にも問題はあります．例えば，ID の値によっては，図 7.14 のような状態に陥ってしまうこともあります．これではリンクリストと変わりありません．

よって，ID の付け方にも色々と工夫が必要になってきます．こういった場合，平衡二分探索木等を利用する必要がでてきます．まあ，ここで平衡二分探索木の説明をするときり

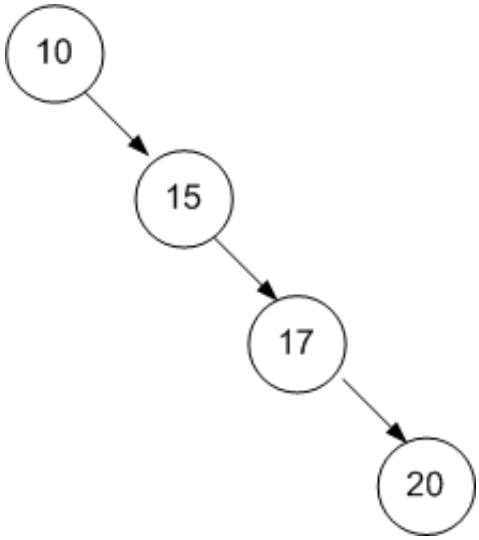

図 7.14: 連結リストと変わらない二分探索木

が無いので，この辺でやめておきます．

さて，ここまでグラフィック管理プログラムの改良の方法について解説してきました．読者の方の中には，難しかった，分けが分からなかった方もいるかと思います．そういった方は是非アルゴリズムの部を見てみて，データ構造等について勉強してみてください．また，ここで学んだアルゴリズムを是非グラフィック管理のプログラムに適応してみて下さい．

第8章　選択肢の表示

この章では，ノベルゲームに欠かせない選択肢表示部分を作っていきたいと思います．条件分岐の数は2個までとします．選択肢が2つ画面に表示され，どちらをクリックしたかどうかをチェックするプログラムを作成していきます．完成イメージは図8.1となります．

図 8.1: 選択肢の表示

8.1　選択肢ボックスの表示

では，画面に選択肢ボックスを表示させるプログラムを作成してみましょう．難しいことは全くありません．ただ画面に選択肢ボックスの画像を貼り付け，その上に文字を描画しているだけです．

では，選択肢ボックスを表示するプログラムをお見せします．利用する画像は back.png と select.png です．back.png は背景画像です．また，select.png は図 8.2 を利用することとします．

図 8.2: select.png

リスト 8.1: ”selection-01.cpp”

```
#include "DxLib.h"

#define GRAPHIC_BACKGROUND_FILENAME "./pic/back.png"
#define GRAPHIC_SELECTBOX_FILENAME  "./pic/select.png"

#define SELECT_BOX_X 50
#define SELECT_BOX_Y 200

#define SELECT_BOX_WIDTH 700
#define SELECT_BOX_HEIGHT 50

#define SELECT_BOX_MESSAGE_Y 20

#define FONT_SIZE 16

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
             LPSTR lpCmdLine, int nCmdShow )
{
  //ウィンドウモードで起動
  ChangeWindowMode( TRUE );
  //画面の大きさは800 * 600
  SetGraphMode( 800, 600 , 16 ) ;
  //DxLib初期化
  if( DxLib_Init() == -1 ) {
     return -1;
  }

  // 描画先を裏画面にセット
  SetDrawScreen( DX_SCREEN_BACK );

  //フォントの大きさをセット
  SetFontSize( FONT_SIZE );

  //白
  int whiteColor = GetColor(255, 255, 255);
  //背景の読み込み
  int backgroundGraphic = LoadGraph( GRAPHIC_BACKGROUND_FILENAME );
  //選択ボックスの読み込み
  int selectBoxGraphicHandle = LoadGraph( GRAPHIC_SELECTBOX_FILENAME );

  //メインループ
  while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {

    //画面クリア
    ClearDrawScreen();

    //背景を描画
    DrawGraph( 0, 0, backgroundGraphic, FALSE );
    //透明度50%とする
```

```
50      SetDrawBlendMode( DX_BLENDMODE_ALPHA , 128 );
51      //選択ボックスの描画(選択肢2つの場合)
52      DrawGraph( SELECT_BOX_X, SELECT_BOX_Y, selectBoxGraphicHandle, TRUE );
53      DrawGraph( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
54        selectBoxGraphicHandle, TRUE);
55      //アルファブレンドを元に戻す
56      SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );
57
58      //適当に文字を表示する
59      DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_MESSAGE_Y,
60        "選択肢1 あいうえおかきくけこ", whiteColor);
61      DrawString(SELECT_BOX_X + 20,
62                    SELECT_BOX_Y + SELECT_BOX_HEIGHT + SELECT_BOX_MESSAGE_Y,
63        "選択肢2 さしすせそたちつてと", whiteColor);
64
65      ScreenFlip();
66    }
67
68    DxLib_End();
69    return 0;
70  }
```

選択肢ボックスの画像 (select.png) を表示する際には，アルファブレンドを利用して半透明にしています．実行結果は図 8.3 となります．

図 8.3: 選択肢ボックスの表示

8.2 マウスカーソルがボックス内に存在するかの判定

選択肢の上にマウスカーソルを乗せた時，選択肢ボックスの透明度を無くすプログラムを作成してみたいと思います．マウスカーソルの座標を取得するには，GetMousePoint 関数を利用します．

```
int GetMousePoint( int *XBuf, int *YBuf );
```

XBuf, YBuf にはマウスカーソルの座標が代入されます．

さて，選択肢ボックスを四角形として，マウスカーソルを点とします．マウスカーソルが選択肢ボックス内にあるかどうかは，点が四角形内に存在しているかどうかの判定となります．ここで，点の座標を (x_1, y_1), 四角形の左端の座標を (x_2, y_2), 高さを height, 横幅を width とすると，

$$x_1 \geq x_2$$

$$x_1 \leq x_2 + width$$

$$y_1 \geq y_2$$

$$y_1 \leq y_2 + height$$

を満たせば，点が四角形内にあると判定できます (図 8.4)．

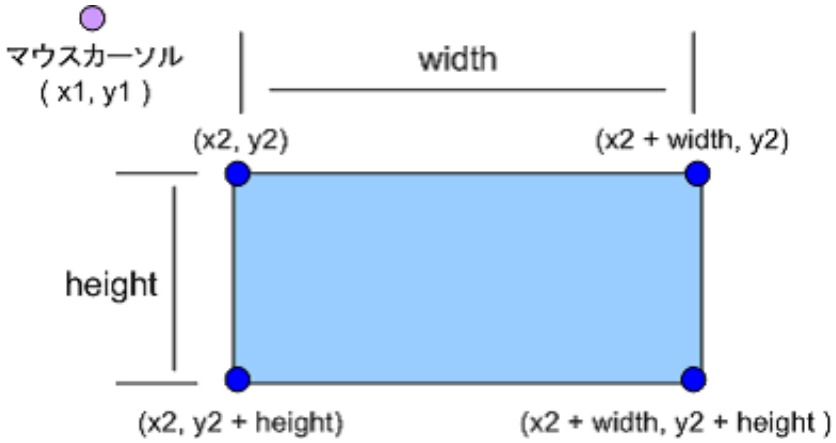

図 8.4: 点が四角形内に存在するかの判定

この判定をプログラムにすると，以下のようになります．

```
1  //指定したボックス内にマウスが存在するかどうか
2  //戻り値 1:存在する 0:存在しない
3  int isContainMousePointer(int x, int y, int width, int height)
4  {
5          int mouseX, mouseY;
6
```

```
7           //マウスの座標を取得
8           GetMousePoint( &mouseX, &mouseY );
9
10          //ボックス内にマウス座標が存在するか
11          if( (mouseX >= x && mouseX <= x + width) &&
12              (mouseY >= y && mouseY <= y + height) ) {
13                  return 1;
14          }
15          return 0;
16  }
```

これらを踏まえて，選択肢ボックス上にマウスを乗せた場合，ボックスの透明度を変えるプログラムを作成してみましょう．また，前回のプログラムは main 関数に色々と処理を書きすぎました．今回は main 関数で行っていた描画処理を，drawSelectBox 関数へと移動させました．

リスト 8.2: ”selection-02.cpp”

```
1  #include "DxLib.h"
2
3  #define GRAPHIC_BACKGROUND_FILENAME "./pic/back.png"
4  #define GRAPHIC_SELECTBOX_FILENAME  "./pic/select.png"
5
6  #define SELECT_BOX_X 50
7  #define SELECT_BOX_Y 200
8
9  #define SELECT_BOX_WIDTH 700
10 #define SELECT_BOX_HEIGHT 50
11
12 #define SELECT_BOX_MESSAGE_Y 20
13
14 #define FONT_SIZE 16
15
16 //選択肢ボックス関係
17 //白
18 int g_whiteColor;
19 //背景の読み込み
20 int g_backgroundGraphic;
21 //選択ボックスの読み込み
22 int g_selectBoxGraphicHandle;
23
24 //プロトタイプ宣言
25 int isContainMousePointer(int x, int y, int width, int height);
26 void drawSelectBox();
27
28 //指定したボックス内にマウスが存在するかどうか
29 //戻り値 1:存在する 0:存在しない
30 int isContainMousePointer(int x, int y, int width, int height)
31 {
32   int mouseX, mouseY;
33
34   //マウスの座標を取得
35   GetMousePoint( &mouseX, &mouseY );
36
37   //ボックス内にマウス座標が存在するか
38   if( (mouseX >= x && mouseX <= x + width) &&
39       (mouseY >= y && mouseY <= y + height) ) {
40       return 1;
```

```
41    }
42    return 0;
43  }
44
45  //選択肢ボックスを描画する
46  void drawSelectBox()
47  {
48    //選択肢ボックス1がマウスポインタを含んでいた場合
49    if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y,
50      SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
51      //透明度0%とする
52      SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
53    }else {
54      //透明度50%とする
55      SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
56    }
57    //選択ボックスの描画
58    DrawGraph( SELECT_BOX_X, SELECT_BOX_Y, g_selectBoxGraphicHandle, TRUE );
59
60    //選択肢ボックス2がマウスポインタを含んでいた場合
61    if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
62      SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
63      //透明度0%とする
64      SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
65    }else {
66      //透明度50%とする
67      SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
68    }
69    //選択ボックス2つ目の描画
70    DrawGraph( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
71      g_selectBoxGraphicHandle, TRUE);
72
73    //アルファブレンドを元に戻す
74    SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );
75
76    //適当に文字を表示する
77    DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_MESSAGE_Y,
78      "選択肢1 あいうえおかきくけこ", g_whiteColor);
79    DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_HEIGHT + SELECT_BOX_MESSAGE_Y,
80      "選択肢2 さしすせそたちつてと", g_whiteColor);
81  }
82
83  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
84                LPSTR lpCmdLine, int nCmdShow )
85  {
86    //ウィンドウモードで起動
87    ChangeWindowMode( TRUE );
88    //画面の大きさは800 * 600
89    SetGraphMode( 800, 600 , 16 ) ;
90    //DxLib初期化
91    if( DxLib_Init() == -1 ) {
92       return -1;
93    }
94
95    // 描画先を裏画面にセット
96    SetDrawScreen( DX_SCREEN_BACK );
97
98    //フォントの大きさをセット
99    SetFontSize( FONT_SIZE );
100
101   // マウスを表示状態にする
102   SetMouseDispFlag( TRUE );
103
104   //白
105   g_whiteColor = GetColor(255, 255, 255);
```

```
106   //背景の読み込み
107   g_backgroundGraphic = LoadGraph( GRAPHIC_BACKGROUND_FILENAME );
108   //選択ボックスの読み込み
109   g_selectBoxGraphicHandle = LoadGraph( GRAPHIC_SELECTBOX_FILENAME );
110
111   //メインループ
112   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
113
114     //画面クリア
115     ClearDrawScreen();
116
117     //背景を描画
118     DrawGraph( 0, 0, g_backgroundGraphic, FALSE );
119
120     //選択肢ボックスの描画
121     drawSelectBox();
122
123     ScreenFlip();
124   }
125
126   DxLib_End();
127   return 0;
128 }
```

このプログラムの実行結果は図 8.5 となりました.

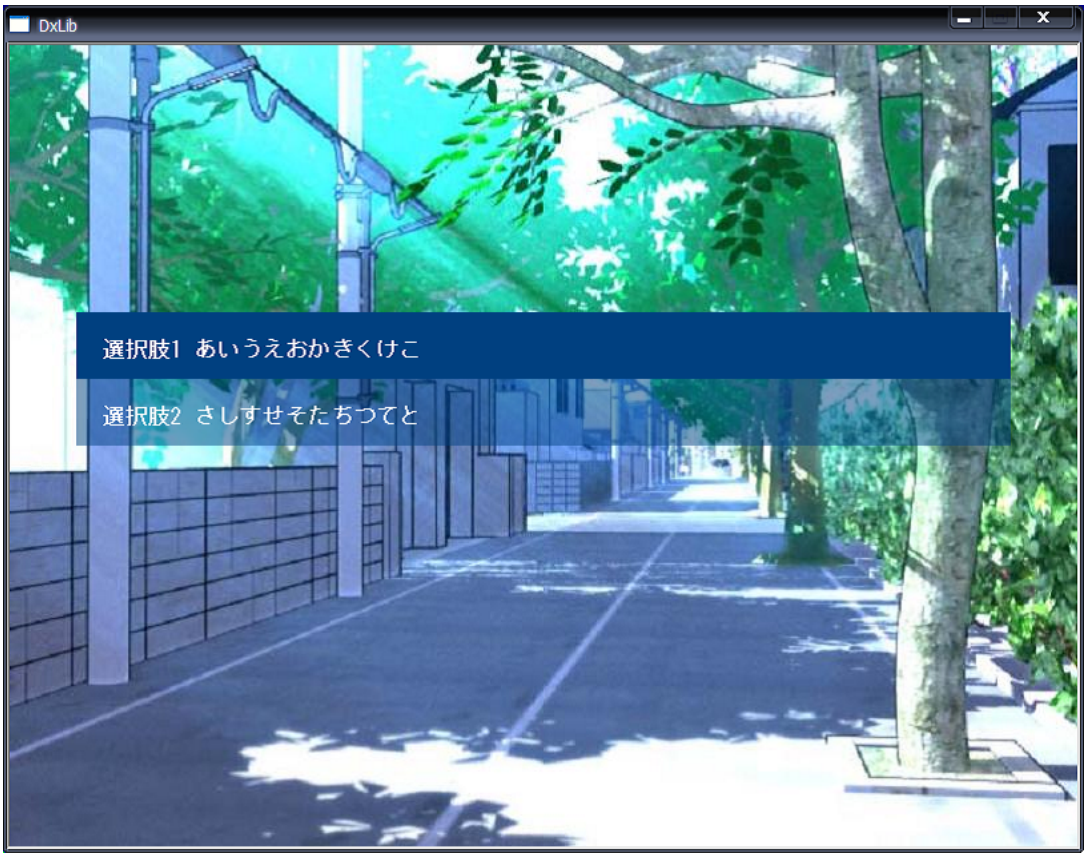

図 8.5: 選択肢 1 にマウスカーソルを乗せた時

8.3 選択肢ボックスにメッセージを入れる

選択肢ボックスに，好きなメッセージを入れられるように改良してみましょう．選択肢ボックスに表示したいメッセージを g_selectBoxMessage にセットし，その文字列をただ描画するだけです．特に難しいことはありません．

また，メッセージが何もセットされていない時は，選択肢ボックスを表示しないようにしてみます．選択肢を表示するかどうかは g_selectBoxVisibleFlag で管理することとしましょう．

では作成したプログラムをお見せします．F1,F2 キーを押すと，それぞれ違うメッセージの入った選択肢ボックスが表示されます．

リスト 8.3: ”selection-03.cpp”

```
#include "DxLib.h"

#define GRAPHIC_BACKGROUND_FILENAME "./pic/back.png"
#define GRAPHIC_SELECTBOX_FILENAME  "./pic/select.png"

#define SELECT_BOX_X 50
#define SELECT_BOX_Y 200

#define SELECT_BOX_WIDTH 700
#define SELECT_BOX_HEIGHT 50

#define SELECT_BOX_MESSAGE_Y 20

#define FONT_SIZE 16

#define SELECT_BOX_MESSAGE_MAX_LENGTH 100
#define SELECT_BOX_HIDE 0
#define SELECT_BOX_SHOW 1

//選択肢ボックス関係(選択肢は2つとする)
//白
int g_whiteColor;
//背景の読み込み
int g_backgroundGraphic;
//選択ボックスの読み込み
int g_selectBoxGraphicHandle;
//選択ボックスに表示するメッセージ
char g_selectBoxMessage[2][SELECT_BOX_MESSAGE_MAX_LENGTH];
//選択ボックスを表示するかどうか 0:非表示 1:表示
int g_selectBoxVisibleFlag;

//プロトタイプ宣言
void initSelectBox();
int isContainMousePointer(int x, int y, int width, int height);
void drawSelectBox();
void setSelectBoxMessage(const char* message1, const char* message2);

//選択ボックスの初期化
void initSelectBox()
{
  //白
  g_whiteColor = GetColor(255, 255, 255);
  //選択ボックスの読み込み
  g_selectBoxGraphicHandle = LoadGraph( GRAPHIC_SELECTBOX_FILENAME );
  //選択ボックスのメッセージ初期化
  memset( g_selectBoxMessage[0], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
  memset( g_selectBoxMessage[1], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
```

```
48    //選択ボックスを表示しない
49    g_selectBoxVisibleFlag = SELECT_BOX_HIDE;
50  }
51
52
53  //指定したボックス内にマウスが存在するかどうか
54  //戻り値 1:存在する 0:存在しない
55  int isContainMousePointer(int x, int y, int width, int height)
56  {
57    int mouseX, mouseY;
58
59    //マウスの座標を取得
60    GetMousePoint( &mouseX, &mouseY );
61
62    //ボックス内にマウス座標が存在するか
63    if( (mouseX >= x && mouseX <= x + width) &&
64      (mouseY >= y && mouseY <= y + height) ) {
65        return 1;
66    }
67    return 0;
68  }
69
70  //選択ボックスにメッセージをセットする
71  void setSelectBoxMessage(const char* message1, const char* message2)
72  {
73    //メッセージセット
74    strncpy(g_selectBoxMessage[0], message1, SELECT_BOX_MESSAGE_MAX_LENGTH );
75    strncpy(g_selectBoxMessage[1], message2, SELECT_BOX_MESSAGE_MAX_LENGTH );
76    //選択ボックスを表示
77    g_selectBoxVisibleFlag = SELECT_BOX_SHOW;
78  }
79
80  //選択肢ボックスを描画する
81  void drawSelectBox()
82  {
83    //選択ボックスを表示するかどうか
84    if( g_selectBoxVisibleFlag ) {
85      //選択肢ボックス1がマウスポインタを含んでいた場合
86      if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y,
87        SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
88        //透明度0%とする
89        SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
90      }else {
91        //透明度50%とする
92        SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
93      }
94      //選択ボックスの描画
95      DrawGraph( SELECT_BOX_X, SELECT_BOX_Y, g_selectBoxGraphicHandle, TRUE );
96
97      //選択肢ボックス2がマウスポインタを含んでいた場合
98      if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
99        SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
100       //透明度0%とする
101       SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
102     }else {
103       //透明度50%とする
104       SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
105     }
106     //選択ボックス2つ目の描画
107     DrawGraph( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
108       g_selectBoxGraphicHandle, TRUE);
109
110     //アルファブレンドを元に戻す
111     SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );
112
```

```
113     //メッセージ表示
114     DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_MESSAGE_Y,
115       g_selectBoxMessage[0], g_whiteColor);
116     DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_HEIGHT + SELECT_BOX_MESSAGE_Y,
117       g_selectBoxMessage[1], g_whiteColor);
118   }
119 }
120
121 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
122               LPSTR lpCmdLine, int nCmdShow )
123 {
124   //ウィンドウモードで起動
125   ChangeWindowMode( TRUE );
126   //画面の大きさは800 * 600
127   SetGraphMode( 800, 600 , 16 ) ;
128   //DxLib初期化
129   if( DxLib_Init() == -1 ) {
130      return -1;
131   }
132
133   // 描画先を裏画面にセット
134   SetDrawScreen( DX_SCREEN_BACK );
135
136   //フォントの大きさをセット
137   SetFontSize( FONT_SIZE );
138
139   // マウスを表示状態にする
140   SetMouseDispFlag( TRUE );
141
142   //選択ボックスの初期化
143   initSelectBox();
144   //背景の読み込み
145   g_backgroundGraphic = LoadGraph( GRAPHIC_BACKGROUND_FILENAME );
146
147   //メインループ
148   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
149
150     //F1またはF2を押すと選択ボックスを表示
151     if( CheckHitKey( KEY_INPUT_F1 ) ) {
152       setSelectBoxMessage("F1を押した時の選択肢1", "選択肢2");
153     }else if( CheckHitKey( KEY_INPUT_F2 ) ) {
154       setSelectBoxMessage("F2を押した時の選択肢1", "選択肢2 あいうえお");
155     }
156     //画面クリア
157     ClearDrawScreen();
158
159     //背景を描画
160     DrawGraph( 0, 0, g_backgroundGraphic, FALSE );
161
162     //選択肢ボックスの描画
163     drawSelectBox();
164
165     ScreenFlip();
166   }
167
168   DxLib_End();
169   return 0;
170 }
```

プログラムの実行結果は図 8.6 及び図 8.7 となります.

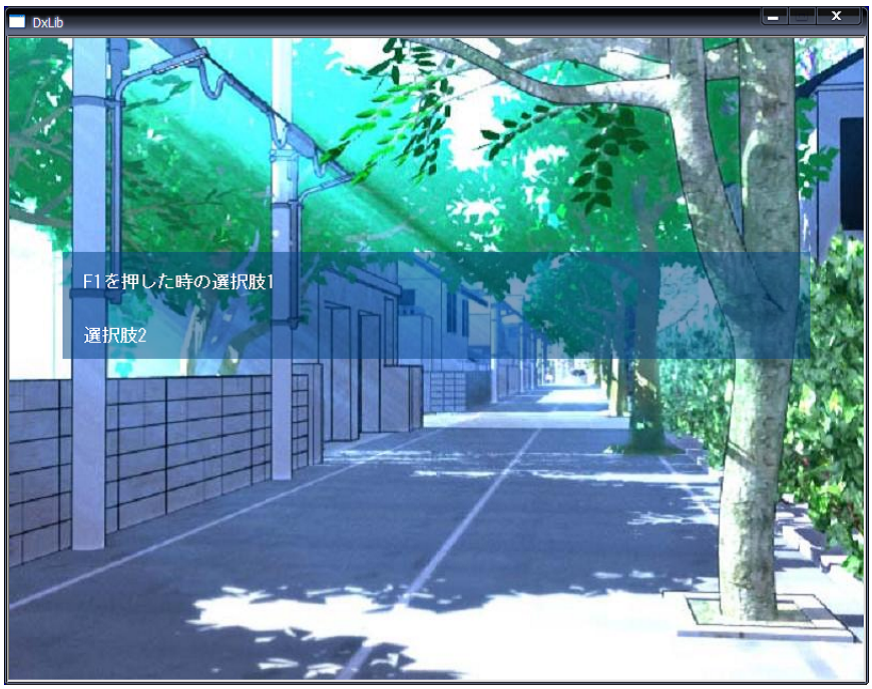

図 8.6: F1 を押した時

図 8.7: F2 を押した時

8.4 選択肢の表示プログラム

ようやく選択肢表示のプログラムの完成です．今回は，選択肢をクリックできるようにしてみましょう．どの選択肢がクリックされたかを調べる為に，isClickedSelectBox 関数を作成しました．ここで，マウスが押されているかどうかを調べるには，GetMouseInput 関数を利用します．

```
int GetMouseInput( void );
```

この関数は，戻り値としてマウスの入力状態を返します．マウスのどのボタンが押されているかを調べるには，以下の定義値と AND 演算を行うことで調べることができます．

- MOUSE_INPUT_LEFT : マウス左ボタン
- MOUSE_INPUT_RIGHT : マウス右ボタン
- MOUSE_INPUT_MIDDLE : マウス中央ボタン

例えば，マウス左ボタンが押されているかどうかを判定するには，以下のようなプログラムを書けばいいわけです．

```
if( ( GetMouseInput() & MOUSE_INPUT_LEFT ) != 0 ) {
    //左クリック時の動作
}
```

では，完成したプログラムをお見せします．

リスト 8.4: ”selection-04.cpp”

```
1  #include "DxLib.h"
2
3  #define GRAPHIC_BACKGROUND_FILENAME "./pic/back.png"
4  #define GRAPHIC_SELECTBOX_FILENAME  "./pic/select.png"
5
6  #define SELECT_BOX_X 50
7  #define SELECT_BOX_Y 200
8
9  #define SELECT_BOX_WIDTH 700
10 #define SELECT_BOX_HEIGHT 50
11
12 #define SELECT_BOX_MESSAGE_Y 20
13
14 #define FONT_SIZE 16
15
16 #define SELECT_BOX_MESSAGE_MAX_LENGTH 100
17 #define SELECT_BOX_HIDE 0
18 #define SELECT_BOX_SHOW 1
19
20 //選択肢ボックス関係(選択肢は2つとする)
21 //白
22 int g_whiteColor;
23 //背景の読み込み
24 int g_backgroundGraphic;
25 //選択ボックスの読み込み
26 int g_selectBoxGraphicHandle;
27 //選択ボックスに表示するメッセージ
```

```
28 char g_selectBoxMessage[2][SELECT_BOX_MESSAGE_MAX_LENGTH];
29 //選択ボックスを表示するかどうか 0:非表示 1:表示
30 int g_selectBoxVisibleFlag;
31
32 //プロトタイプ宣言
33 void initSelectBox();
34 int isContainMousePointer(int x, int y, int width, int height);
35 void drawSelectBox();
36 void setSelectBoxMessage(const char* message1, const char* message2);
37 int isClickedSelectBox();
38
39 //選択ボックスの初期化
40 void initSelectBox()
41 {
42   //白
43   g_whiteColor = GetColor(255, 255, 255);
44   //選択ボックスの読み込み
45   g_selectBoxGraphicHandle = LoadGraph( GRAPHIC_SELECTBOX_FILENAME );
46   //選択ボックスのメッセージ初期化
47   memset( g_selectBoxMessage[0], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
48   memset( g_selectBoxMessage[1], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
49   //選択ボックスを表示しない
50   g_selectBoxVisibleFlag = SELECT_BOX_HIDE;
51 }
52
53
54 //指定したボックス内にマウスが存在するかどうか
55 //戻り値 1:存在する 0:存在しない
56 int isContainMousePointer(int x, int y, int width, int height)
57 {
58   int mouseX, mouseY;
59
60   //マウスの座標を取得
61   GetMousePoint( &mouseX, &mouseY );
62
63   //ボックス内にマウス座標が存在するか
64   if( (mouseX >= x && mouseX <= x + width) &&
65     (mouseY >= y && mouseY <= y + height) ) {
66       return 1;
67   }
68   return 0;
69 }
70
71 //選択ボックスにメッセージをセットする
72 void setSelectBoxMessage(const char* message1, const char* message2)
73 {
74   //メッセージセット
75   strncpy(g_selectBoxMessage[0], message1, SELECT_BOX_MESSAGE_MAX_LENGTH );
76   strncpy(g_selectBoxMessage[1], message2, SELECT_BOX_MESSAGE_MAX_LENGTH );
77   //選択ボックスを表示
78   g_selectBoxVisibleFlag = SELECT_BOX_SHOW;
79 }
80
81 //選択肢ボックスを描画する
82 void drawSelectBox()
83 {
84   //選択ボックスを表示するかどうか
85   if( g_selectBoxVisibleFlag ) {
86     //選択肢ボックス1がマウスポインタを含んでいた場合
87     if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y,
88       SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
89       //透明度0%とする
90       SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
91     }else {
92       //透明度50%とする
```

```
93        SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
94      }
95      //選択ボックスの描画
96      DrawGraph( SELECT_BOX_X, SELECT_BOX_Y, g_selectBoxGraphicHandle, TRUE );
97
98      //選択肢ボックス2がマウスポインタを含んでいた場合
99      if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
100       SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
101       //透明度0%とする
102       SetDrawBlendMode( DX_BLENDMODE_ALPHA, 255 );
103     }else {
104       //透明度50%とする
105       SetDrawBlendMode( DX_BLENDMODE_ALPHA, 128);
106     }
107     //選択ボックス2つ目の描画
108     DrawGraph( SELECT_BOX_X, SELECT_BOX_Y + SELECT_BOX_HEIGHT,
109       g_selectBoxGraphicHandle, TRUE);
110
111     //アルファブレンドを元に戻す
112     SetDrawBlendMode( DX_BLENDMODE_NOBLEND , 0 );
113
114     //メッセージ表示
115     DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_MESSAGE_Y,
116       g_selectBoxMessage[0], g_whiteColor);
117     DrawString(SELECT_BOX_X + 20, SELECT_BOX_Y + SELECT_BOX_HEIGHT + SELECT_BOX_MESSAGE_Y,
118       g_selectBoxMessage[1], g_whiteColor);
119   }
120 }
121
122 //選択肢ボックスがクリックされたかどうか
123 //0: クリックされていない 1: 選択肢1がクリックされた 2: 選択肢2がクリックされた
124 int isClickedSelectBox()
125 {
126   //選択肢ボックスが表示されているとき
127   if( g_selectBoxVisibleFlag ) {
128     //マウスの状態を調べる
129     //左クリックされたとき
130     if( (GetMouseInput() & MOUSE_INPUT_LEFT) != 0 ) {
131
132       if( isContainMousePointer( SELECT_BOX_X, SELECT_BOX_Y,
133         SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
134           //選択肢ボックス1がマウスポインタを含んでいた場合
135           //選択ボックスのメッセージ初期化
136           memset( g_selectBoxMessage[0], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
137           memset( g_selectBoxMessage[1], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
138           //選択ボックスを表示しない
139           g_selectBoxVisibleFlag = SELECT_BOX_HIDE;
140           //選択肢1がクリックされていることを通知
141           return 1;
142       }else if( isContainMousePointer( SELECT_BOX_X,
143         SELECT_BOX_Y + SELECT_BOX_HEIGHT,
144         SELECT_BOX_WIDTH, SELECT_BOX_HEIGHT ) ) {
145           //選択肢ボックス2がマウスポインタを含んでいた場合
146           //選択ボックスのメッセージ初期化
147           memset( g_selectBoxMessage[0], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
148           memset( g_selectBoxMessage[1], 0, SELECT_BOX_MESSAGE_MAX_LENGTH);
149           //選択ボックスを表示しない
150           g_selectBoxVisibleFlag = SELECT_BOX_HIDE;
151           //選択肢2がクリックされていることを通知
152           return 2;
153       }
154     }
155   }
156   return 0;
157 }
```

```
158
159
160 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
161              LPSTR lpCmdLine, int nCmdShow )
162 {
163   //ウィンドウモードで起動
164   ChangeWindowMode( TRUE );
165   //画面の大きさは800 * 600
166   SetGraphMode( 800, 600 , 16 ) ;
167   //DxLib初期化
168   if( DxLib_Init() == -1 ) {
169      return -1;
170   }
171
172   // 描画先を裏画面にセット
173   SetDrawScreen( DX_SCREEN_BACK );
174
175   //フォントの大きさをセット
176   SetFontSize( FONT_SIZE );
177
178   // マウスを表示状態にする
179   SetMouseDispFlag( TRUE );
180
181   //選択ボックスの初期化
182   initSelectBox();
183   //背景の読み込み
184   g_backgroundGraphic = LoadGraph( GRAPHIC_BACKGROUND_FILENAME );
185
186   int tmpClicked = 0; //テスト用
187
188   //メインループ
189   while( ProcessMessage() == 0 && CheckHitKey( KEY_INPUT_ESCAPE ) == 0 ) {
190
191     //F1を押すと選択ボックスを表示
192     if( CheckHitKey( KEY_INPUT_F1 ) ) {
193       setSelectBoxMessage("F1を押した時の選択肢1", "選択肢2");
194     }
195
196     //画面クリア
197     ClearDrawScreen();
198
199     //背景を描画
200     DrawGraph( 0, 0, g_backgroundGraphic, FALSE );
201
202     //選択肢ボックスの描画
203     drawSelectBox();
204
205     //どの選択肢がクリックされたか調べる
206     tmpClicked = isClickedSelectBox();
207     if( tmpClicked == 1 ) {
208       DrawString(50, 50, "選択肢1がクリックされました", g_whiteColor);
209     }else if( tmpClicked == 2 ) {
210       DrawString(50, 50, "選択肢2がクリックされました", g_whiteColor);
211     }
212
213     ScreenFlip();
214
215     //クリックされたことが分かるようにしておく
216     if( tmpClicked != 0 ) {
217       Sleep(1000);
218     }
219   }
220
221   DxLib_End();
222   return 0;
```

```
223 }
```

このプログラムの実行結果は図 8.8 となります.

図 8.8: 選択肢の表示プログラム

第9章　ノベルゲーム用スクリプト言語の作成

この章では，ノベルゲーム用スクリプトエンジンを作成していきたいと思います．ノベルゲームを作るにあたっては，ここが一番の難所です．

なぜこのようにノベルゲーム用の言語を作成するのでしょうか．ストーリーなり何なりC言語のソース内に組み込んでしまえば，このようなスクリプト言語を作る必要はありません．しかし，C言語上にストーリーを書いた場合，ストーリーを変更するごとにいちいちコンパイルしなおす必要があります．また，プログラム担当とシナリオ担当で作業を分担している場合，C言語だけで作った場合は，シナリオ担当の人がいちいちプログラムの仕組みまで意識する必要がでてきます．

こういった問題を排除する為にノベルゲーム用のスクリプト言語を作っていきたいと思います．スクリプト言語には，次の機能を持たせることとします．

- メッセージの表示
- 画像の読み込み・表示
- 条件分岐

なお，音楽再生，セーブ，フラグ等の機能は今回は作らないこととします．しかし，この章の内容を理解できれば，これらの機能は簡単に追加できるはずです．

9.1　スクリプトを読み込む

スクリプトエンジンは，スクリプトファイルから命令を読み込み，その内容を解析する役割を果たします（図9.1）．

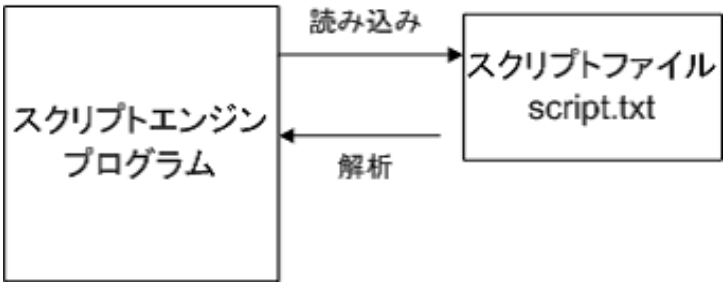

図9.1: スクリプトエンジンの仕組み

今回は，スクリプトファイルから文章を読み込むプログラムを作りましょう．文章は一行単位で読み取ることとします．また，ScriptInformation と呼ばれる構造体を作成し，この中に，読み取ったスクリプト，ファイル名，スクリプト行数等の情報を格納することとしましょう．

では，作成したプログラムをお見せします．

リスト 9.1: ”script-01.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //スクリプトは最大1000行まで読み込む
5  #define SCRIPT_MAX_LINE 1000
6  //スクリプト最大文字数
7  #define SCRIPT_MAX_STRING_LENGTH 300
8
9  typedef struct ScriptInformation_tag {
10   int maxLineNumber;      //スクリプト行数
11   int currentLine;      //現在何行目を実行しているか
12   const char* filename;   //ファイル名
13   char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
14 } ScriptInformation;
15
16 //スクリプトファイルを読み込む
17 //戻り値 -1 : 失敗  0 : 成功
18 int loadScript(const char* filename, ScriptInformation* scriptInfo)
19 {
20   int pos;
21   char c;
22   //スクリプトファイル
23   FILE* fp;
24
25   //スクリプト情報を初期化
26   memset( scriptInfo , 0, sizeof(ScriptInformation) );
27
28   //スクリプトファイルを開く
29   fp = fopen(filename, "r");
30   if( fp == NULL ) {
31     //ファイル読み込みに失敗
32     printf("スクリプト %s を読み込めませんでした\n", filename);
33     return -1;
34   }
35
36   //script書き込み時に使用
37   pos = 0;
38
39   for( ;; ) {
40     //一文字読み込み
41     c = fgetc( fp );
42     //ファイルの終わりかどうか
43     if( feof( fp ) ) {
44       break;
45     }
46
47     if( pos >= SCRIPT_MAX_STRING_LENGTH - 1 ) {
48       //1行の文字数が多すぎる
49       printf("error: 文字数が多すぎます (%d行目)", scriptInfo->currentLine );
50       return -1;
51     }
52
53     //改行文字が出てきた場合，次の行へ移動
54     if( c == '\n' ) {
55       //\0を文字列の最後に付ける
```

```
56        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = '\0';
57        //次の行に移動
58        scriptInfo->currentLine++;
59        //書き込み位置を0にする
60        pos = 0;
61      }else {
62        //書き込み
63        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = c;
64        //文字書き込み位置をずらす
65        pos++;
66      }
67    }
68    //最大行数
69    scriptInfo->maxLineNumber = scriptInfo->currentLine;
70    //読み込み中の行を0にする
71    scriptInfo->currentLine = 0;
72    //スクリプトファイル名を設定
73    scriptInfo->filename = filename;
74    return 0;
75  }
76
77  int main()
78  {
79    int i;
80    ScriptInformation script;
81
82    loadScript( "./script.txt", &script );
83
84    //10行目までを表示
85    for( i = 0; i < 10; i++ ) {
86      printf("%d : %s\n", i + 1, script.script[i] );
87    }
88
89    return 0;
90  }
```

特に難しいことはありませんね．ただ一文字ずつ読み取って，ScriptInformation の script に書き込んでいるだけです．また，改行文字 (\n) を見つけたら，次の行に移動するようにしています．

このプログラムの実行結果は以下のようになりました．なお，読み込んだ script.txt の内容は次のようになっています．

```
1  @@message こんにちは，文章表示のテストです
2
3  @@message あいうえおかきくけこさしすせそ
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです
2 :
3 : @@message あいうえおかきくけこさしすせそ
4 :
5 :
6 :
7 :
8 :
9 :
10 :
```

9.2 空白空行を飛ばして読み込む

前回のプログラムを少し改良して，空白，空行を読み飛ばすようにしてみましょう．今回もたいしたことはありません．例えば，行の先頭の文字が空白又はタブ (\t) だった場合は，文字がでてくるまで読み飛ばすようにしましょう．また，行の先頭文字が改行 (\n) だった場合は，その行を読み飛ばすようにしましょう．

では，完成したプログラムをお見せします．

リスト 9.2: ”script-02.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3
4  //スクリプトは最大1000行まで読み込む
5  #define SCRIPT_MAX_LINE 1000
6  //スクリプト最大文字数
7  #define SCRIPT_MAX_STRING_LENGTH 300
8
9  typedef struct ScriptInformation_tag {
10   int maxLineNumber;      //スクリプト行数
11   int currentLine;      //現在何行目を実行しているか
12   const char* filename;   //ファイル名
13   char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
14 } ScriptInformation;
15
16 //スクリプトファイルを読み込む
17 //戻り値 -1 : 失敗  0 : 成功
18 int loadScript(const char* filename, ScriptInformation* scriptInfo)
19 {
20   int pos;
21   char c;
22   //スクリプトファイル
23   FILE* fp;
24
25   //スクリプト情報を初期化
26   memset( scriptInfo , 0, sizeof(ScriptInformation) );
27
28   //スクリプトファイルを開く
29   fp = fopen(filename, "r");
30   if( fp == NULL ) {
```

```
31      //ファイル読み込みに失敗
32      printf("スクリプト %s を読み込めませんでした\n", filename);
33      return -1;
34    }
35
36    //script書き込み時に使用
37    pos = 0;
38
39    for( ;; ) {
40      //一文字読み込み
41      c = fgetc( fp );
42      //ファイルの終わりかどうか
43      if( feof( fp ) ) {
44        break;
45      }
46      //文章先頭の空白部分を読み飛ばす
47      while( (c == ' ' || c == '\t') && pos == 0 && !feof( fp ) ) {
48        c = fgetc( fp );
49      }
50
51      if( pos >= SCRIPT_MAX_STRING_LENGTH - 1 ) {
52        //1行の文字数が多すぎる
53        printf("error: 文字数が多すぎます (%d行目)", scriptInfo->currentLine );
54        return -1;
55      }
56
57      //改行文字が出てきた場合, 次の行へ移動
58      if( c == '\n' ) {
59        //空行は読み飛ばす
60        if( pos == 0 ) {
61          continue;
62        }
63        //\0を文字列の最後に付ける
64        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = '\0';
65        //次の行に移動
66        scriptInfo->currentLine++;
67        //書き込み位置を0にする
68        pos = 0;
69      }else {
70        //書き込み
71        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = c;
72        //文字書き込み位置をずらす
73        pos++;
74      }
75    }
76    //最大行数
77    scriptInfo->maxLineNumber = scriptInfo->currentLine;
78    //読み込み中の行を0にする
79    scriptInfo->currentLine = 0;
80    //スクリプトファイル名を設定
81    scriptInfo->filename = filename;
82    return 0;
83  }
84
85  int main()
86  {
87    int i;
88    ScriptInformation script;
89
90    loadScript( "./script.txt", &script );
91
92    //10行目までを表示
93    for( i = 0; i < 10; i++ ) {
94      printf("%d : %s\n", i + 1, script.script[i] );
95    }
```

```
96
97    return 0;
98  }
```

このプログラムの実行結果は以下のようになりました．また，読み込むスクリプトファイル script.txt の内容は以下のようにしてみました．

```
1   @@message こんにちは，文章表示のテストです
2
3           @@message あいうえおかきくけこさしすせそ
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです
2 : @@message あいうえおかきくけこさしすせそ
3 :
4 :
5 :
6 :
7 :
8 :
9 :
10 :
```

9.3 文字列分割

今回は，文字列を指定した区切り文字で分割する関数を作りたいと思います．この関数は，次節で使用する部品となります．文字列を分割する関数は，Java 言語では StringTokenizer などと呼ばれています．よく使いそうな機能なのですが，C 言語，C++言語ともに標準では搭載されていません．よって，自分で作る必要があります．

では，文字列を分割する splitString 関数を作成してみます．文字列分割には strtok 関数を利用しています．使い方が分からない方はインターネット等で調べてみてください．

リスト 9.3: ”script-03.cpp”

```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdlib.h>
4
5   //スクリプト最大文字数
6   #define SCRIPT_MAX_STRING_LENGTH 300
7
8   //文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
9   //src : 分割したい文字列
10  //dest: 分割された文字列
11  //delim: 区切り文字
12  //splitNum : 最大分割数
13  void splitString(char* src, char* dest[], const char* delim, int splitNum)
```

```
14 {
15   int i;
16   char* cp;
17   char* copySrc;
18 
19   //元の文章をコピーする
20   copySrc = (char*)malloc( sizeof(int) * SCRIPT_MAX_STRING_LENGTH + 1);
21   strncpy( copySrc, src, SCRIPT_MAX_STRING_LENGTH );
22   cp = copySrc;
23 
24   //strtokを使って copySrc を delim区切りで分割する
25   for( i = 0; i < splitNum ; i++ ) {
26     //分割対象文字列が無くなるまで分割
27     if( (dest[i] = strtok(cp, delim)) == NULL ) {
28       break;
29     }
30     //2回目にstrtokを呼び出す時は，cpをNULLにする
31     cp = NULL;
32   }
33   //分割された文字列の最後の要素はNULLとしておく
34   dest[i] = NULL;
35 }
36 
37 //デバッグ用
38 //elemの要素を表示
39 void printElements(char* elem[])
40 {
41   int i;
42   for( i = 0; elem[i] != NULL; i++ ) {
43     printf("%d : %s\n", i + 1, elem[i] );
44   }
45 }
46 
47 int main()
48 {
49   char* dest[100];
50   const char* delim = " ";
51 
52   splitString("@message test 222 333", dest, delim, 3);
53   printElements( dest );
54 
55   splitString("@switch    test 333 444 555", dest, delim, 6);
56   printElements( dest );
57 
58   return 0;
59 }
```

プログラムの実行結果は以下のようになりました．空白区切りで文字列が分割されていることがわかります．

実行結果

```
1 : @message
2 : test
3 : 222
1 : @switch
2 : test
3 : 333
4 : 444
5 : 555
```

9.4 字句・構文解析器の作成

スクリプトエンジンは大まかに分けて，字句解析器と構文解析器からできています．字句解析とは，連続した文字の並びを，字句（トークン）に分割することです．また，構文解析とは，字句解析によって分割された字句を取り出して，それらの並びから構造を把握し，実行すべき内容を判断するもののことです．

なにやら，難しいですね．まともなスクリプト言語を作ろうと思っているのなら，字句解析，構文解析だけでも作るのに相当の時間がかかるとおもいます．しかし，今回はゲーム用のスクリプト言語なので，難しい仕組みはすべて排除しましょう．例えば，画面にメッセージを表示するスクリプトの仕様は図 9.2 とします．これは，画面に「こんにちは，文章表示のテストです」と表示するスクリプトです．

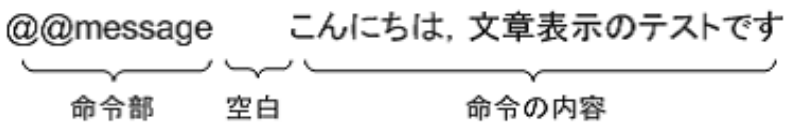

図 9.2: メッセージ表示のスクリプト

スクリプト命令は，命令部，空白，命令の内容からできています．命令部はかならず行の先頭にくるようにしましょう．また，命令はすべて一文で終わるようにします．このようにすると制約は増えますが，設計が楽です．また，命令部と命令の内容は空白で区切ることとしましょう．空白で区切ると，メッセージの内容をスクリプトの途中で改行できない，メッセージに空白文字を入れることができない等の制約がでてしまいますが，ここではこれらの問題は無視しましょう．

以上を踏まえると，字句解析器のする役割は，単にメッセージを空白で区切ればよいだけとなります．なんとも簡単な設計ですね．

次に構文解析器の説明に入ります．といっても，大それたものは作りません．むしろ，これから作るものは，構文解析器とはいえないようなものかもしれません．

`@@message` 命令を例に挙げて，構文解析の方法を説明していきます．字句解析を行うと，「`@@message`」と「表示したいメッセージ」の 2 つの字句に分割できます．そこで，1 番目

の字句を命令とすることにしましょう．もし，ここに命令がこなかった場合はスクリプトエラーとします．

2 番目以降の字句を命令文の引数としましょう．@@message 命令の場合，引数は 1 つしかとらないようにします．つまり，2 つ以上の引数が書かれていた場合は無視します．

以上のような設計にすると，処理はぐっと楽になります．もし，このようなプログラミング言語を作ってしまったら誰も使いたがらないでしょうが，ゲーム用のスクリプト言語なので，このくらいは気にしないこととしましょう．

さて，もう一つ命令を作ってみたいと思います．それは，指定した画像ファイルを座標 (x, y) の位置に表示する命令，@@drawgraph です．@@drawgraph は引数を 3 つとります．第 1, 第 2 引数は画像を表示する座標 (x, y)，第 3 引数は表示する画像のファイル名を渡すこととします．例えば，human.png という画像を座標 (50, 100) に配置する場合，

```
@@drawgraph 50 100 human.png
```

といったように記述することとします．

解説が少し長くなりました．では，作成したプログラムをお見せします．

リスト 9.4: ”script-04.cpp”

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//スクリプトは最大1000行まで読み込む
#define SCRIPT_MAX_LINE 1000
//スクリプト最大文字数
#define SCRIPT_MAX_STRING_LENGTH 300

typedef struct ScriptInformation_tag {
  int maxLineNumber;      //スクリプト行数
  int currentLine;      //現在何行目を実行しているか
  const char* filename;   //ファイル名
  char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
} ScriptInformation;


//プロトタイプ宣言
int loadScript(const char* filename, ScriptInformation* scriptInfo);
void splitString(const char* src, char* dest[], const char* delim, int splitNum);
void printElements(char* elem[]);
int decodeScript(const char* scriptMessage);

//スクリプトファイルを読み込む
//戻り値 -1 : 失敗  0 : 成功
int loadScript(const char* filename, ScriptInformation* scriptInfo)
{
  //省略（前回と同じ）
}

//文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
//src : 分割したい文字列
//dest: 分割された文字列
//delim: 区切り文字
//splitNum : 最大分割数
void splitString(const char* src, char* dest[], const char* delim, int splitNum)
{
  //省略（前回と同じ）
}
```

```
40
41  //デバッグ用
42  //elemの要素を表示
43  void printElements(char* elem[])
44  {
45    int i;
46    for( i = 0; elem[i] != NULL; i++ ) {
47      printf("%d : %s\n", i + 1, elem[i] );
48    }
49  }
50
51  //スクリプト文を解読する
52  //戻り値 1: 成功  0: 失敗
53  int decodeScript(const char* scriptMessage)
54  {
55    //分割されたスクリプト文
56    char* message[100];
57    //文字列分割時の区切り文字
58    const char* delim = " ";
59
60    //スクリプト分割
61    splitString( scriptMessage, message, delim, 100 );
62
63    //分割に失敗した場合
64    if( message[0] == NULL ) {
65      return 0;
66    }
67
68    //scriptの仕様
69    //
70    //@@message 文字列
71    //--- 文字列をメッセージとして表示する
72    //
73    //@@drawgraph x y filename
74    //--- filenameで指定した画像ファイルを(x, y)の位置に表示
75
76    //message[0] が @@message の時は, メッセージ命令が来たと判断
77    if( strncmp(message[0], "@@message", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
78      printf("メッセージ : %s\n", message[1] );
79      return 1;
80    }else if(  strncmp(message[0], "@@drawgraph", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
81      printf("画像 %s 表示 -- x座標 : %d, y座標 : %d\n",
82        message[3], atoi( message[1] ), atoi( message[2] ) );
83      return 1;
84    }
85
86    //注意点: エラー処理を行っていない ( message[2] )が存在しなかった場合など
87    //例えば messageの全要素を全てNULL で埋めておき, message[2] がNULLの場合は
88    //エラー処理を行うなどの手が考えられる
89
90    printf("スクリプトエラー\n");
91    return 0;
92  }
93
94  int main()
95  {
96    int i;
97    ScriptInformation script;
98
99    loadScript( "./script.txt", &script );
100
101   //10行目までを表示
102   for( i = 0; i < 10; i++ ) {
103     printf("%d : %s\n", i + 1, script.script[i] );
104   }
```

```
105   for( i = 0; decodeScript( script.script[i] ) != 0 ; i++ )
106     ;
107
108   return 0;
109 }
```

実行結果は以下のようになりました．なお，使用した script.txt は以下のような内容となっています．

```
1   @@message こんにちは，文章表示のテストです hello
2
3   @@message あいうえおかきくけこ
4
5   @@drawgraph 50 100 human.png
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです hello
2 : @@message あいうえおかきくけこ
3 : @@drawgraph 50 100 human.png
4 :
5 :
6 :
7 :
8 :
9 :
10 :
メッセージ : こんにちは，文章表示のテストです
メッセージ : あいうえおかきくけこ
画像 human.png 表示 -- x 座標 : 50, y 座標 : 100
```

script.txt の一行目の文章は hello の部分が正しく表示されていないようです．これは，メッセージの途中で空白が含まれているからでしょう．しかし，メッセージの内容には空白を入れないようにするとして，今回はこれで妥協することとしましょう．

@@message 命令も@@drawgraph 命令もうまく動いているようですね．

9.5 ラベルの検索

C 言語の文法の一つに goto 命令というものがあります．これは，指定したラベルへジャンプする命令です．例えば，次のような C 言語コードを書いてみます．

```
1   int main(void)
2   {
3           printf("hello\n");
```

```
4           goto NEXT;
5
6           printf("ここは実行されない\n");
7   NEXT:
8           printf("next へジャンプしました\n");
9           return 0;
10  }
```

上記のコードの NEXT: という部分がラベルにあたります．goto NEXT; は NEXT ラベルまでジャンプするということです．このプログラムの実行結果は以下のようになります．

実行結果

```
hello
next へジャンプしました
```

C 言語の goto 命令は特別な事情を除いて使うべきではないでしょう．それは，プログラムの流れが分かりにくくなるからです．しかし，今回作成するスクリプト言語では，この goto 命令を採用する事とします．理由は，実装が簡単だからです．

さて，今回はこの goto 命令に利用するラベルが，何行目に存在しているかを検索するプログラムを作って行きたいと思います．ラベルの定義は，

```
@@label ラベル名
```

としましょう．例えば，ラベル NEXT を定義したい場合は，

```
@@label NEXT
```

とします．

では，作成したプログラムをお見せします．searchScriptLabel 関数は，指定したラベルが何行目に存在するかを調べる関数です．

リスト 9.5: ”script-05.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  //スクリプトは最大1000行まで読み込む
6  #define SCRIPT_MAX_LINE 1000
7  //スクリプト最大文字数
8  #define SCRIPT_MAX_STRING_LENGTH 300
9
10 typedef struct ScriptInformation_tag {
11   int maxLineNumber;      //スクリプト行数
12   int currentLine;      //現在何行目を実行しているか
13   const char* filename;   //ファイル名
14   char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
15 } ScriptInformation;
16
17
18 //プロトタイプ宣言
19 int loadScript(const char* filename, ScriptInformation* scriptInfo);
```

```
20 void splitString(const char* src, char* dest[], const char* delim, int splitNum);
21 void printElements(char* elem[]);
22 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo);
23
24 //スクリプトファイルを読み込む
25 //戻り値 -1 : 失敗  0 : 成功
26 int loadScript(const char* filename, ScriptInformation* scriptInfo)
27 {
28   //省略（前回と同じ）
29 }
30
31 //文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
32 //src : 分割したい文字列
33 //dest: 分割された文字列
34 //delim: 区切り文字
35 //splitNum : 最大分割数
36 void splitString(const char* src, char* dest[], const char* delim, int splitNum)
37 {
38   //省略（前回と同じ）
39 }
40
41 //デバッグ用
42 //elemの要素を表示
43 void printElements(char* elem[])
44 {
45   int i;
46   for( i = 0; elem[i] != NULL; i++ ) {
47     printf("%d : %s\n", i + 1, elem[i] );
48   }
49 }
50
51 //指定したラベルがある行数を探す
52 //戻り値 正の数: 指定したラベルの行番号 -1: エラー
53 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo)
54 {
55   //分割されたスクリプト文
56   char* message[100];
57   //文字列分割時の区切り文字
58   const char* delim = " ";
59   int i, line = -1;
60
61   for( i = 0; i < scriptInfo->maxLineNumber; i++ ) {
62     //スクリプト分割
63     splitString( scriptInfo->script[i] , message, delim, 100 );
64
65     //分割に失敗した場合
66     if( message[0] == NULL ) {
67       return -1;
68     }
69
70     //指定したラベルを探す
71     if( strncmp(message[0], "@@label", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
72       if( strncmp(message[1], label, SCRIPT_MAX_STRING_LENGTH) == 0 ) {
73         //指定したラベルが見つかった時
74         line = i;
75         break;
76       }
77     }
78   }
79
80   return line;
81 }
82
83 int main()
84 {
```

```
85    int i;
86    int line;
87    ScriptInformation script;
88
89    loadScript( "./script.txt", &script );
90
91    for( i = 0; i < script.maxLineNumber ; i++ ) {
92      printf("%d : %s\n", i + 1, script.script[i] );
93    }
94
95    line = searchScriptLabel("LABEL1", &script );
96    printf("LABEL1は %d 行目にあります\n", line + 1);
97
98    line = searchScriptLabel("NEXT", &script );
99    printf("NEXTは %d 行目にあります\n", line + 1);
100
101   line = searchScriptLabel("HELLO", &script );
102   printf("HELLOは %d 行目にあります\n", line + 1);
103
104   return 0;
105 }
```

searchScriptLabel 関数は，ラベルを 1 行目から最後の行まで順に調べていくようにしています．しかし，このような仕様では，スクリプトの行数が長くなった時に処理速度が気になります．この問題の解決方法は，この章の最後のセクションにある改良のポイントをご覧ください．

実行結果は以下のようになりました．なお，使用した script.txt は以下のようになっています．

```
1   @@message こんにちは，文章表示のテストです
2
3   @@label LABEL1
4
5           @@message LABEL1 にジャンプしました
6           @@goto NEXT
7
8   @@label LABEL2
9
10          @@message LABEL2 にジャンプしました
11          @@goto NEXT
12
13  @@label NEXT
14
15  @@message これで終了します
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです
2 : @@label LABEL1
3 : @@message LABEL1 にジャンプしました
4 : @@goto NEXT
5 : @@label LABEL2
6 : @@message LABEL2 にジャンプしました
7 : @@goto NEXT
8 : @@label NEXT
9 : @@message これで終了します
LABEL1 は 2 行目にあります
NEXT は 8 行目にあります
HELLO は 0 行目にあります
```

9.6 条件分岐構文の作成

条件分岐用構文@@select を作成していきたいと思います．今回作成する条件分岐は，ゲーム側がユーザに質問をして，その回答によって処理を分岐させることです．例えば，ゲーム側がユーザに年齢を尋ねるとしましょう．条件はユーザが 20 歳未満か 20 歳以上かのどちらかとします．ここで，ユーザが 20 歳未満を選択した場合はラベル LABEL1 へ飛ぶ，20 歳以上を選択した場合はラベル LABEL2 へ飛ぶといった処理をするのが@@select の役割です（図 9.4）．

@@select 構文の構造は図 9.3 としましょう．条件メッセージとジャンプするラベルは区切り文字@@で区切ることとします．この図の構文は,「条件 1 の場合」を選択した場合はラベル LABEL1 へジャンプし,「条件 2 の場合」を選択した場合はラベル LABEL2 へジャンプすることを表しています.

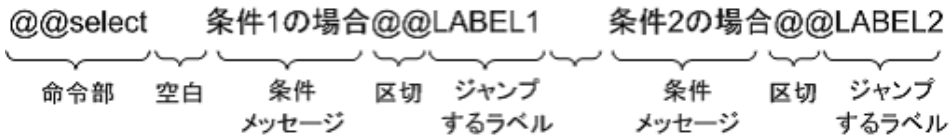

図 9.3: @@select の構造

条件の数には特に決まりはありません．条件分岐数が 2 つの場合もありますし，3 つの場合も考えられます．よって，@@select 構文にはいくつの条件文が与えられたかも数える必要があります.

では，@@select 構文の実装例をお見せします．今回はユーザが選択した回答に対応したラベル名を抜き出す所までを実装しています．また，条件分岐数は 10 個以上は無いという前提のもとプログラムを作成しています.

図 9.4: 条件分岐構文の作成

リスト 9.6: ”script-06.cpp”

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//スクリプトは最大1000行まで読み込む
#define SCRIPT_MAX_LINE 1000
//スクリプト最大文字数
#define SCRIPT_MAX_STRING_LENGTH 300

typedef struct ScriptInformation_tag {
  int maxLineNumber;      //スクリプト行数
  int currentLine;      //現在何行目を実行しているか
  const char* filename;   //ファイル名
  char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
} ScriptInformation;


//プロトタイプ宣言
int loadScript(const char* filename, ScriptInformation* scriptInfo);
void splitString(const char* src, char* dest[], const char* delim, int splitNum);
void printElements(char* elem[]);
int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo);
int searchScriptLabel(const char* label, ScriptInformation* scriptInfo);

//スクリプトファイルを読み込む
//戻り値 -1 : 失敗  0 : 成功
int loadScript(const char* filename, ScriptInformation* scriptInfo)
{
  //省略（前回と同じ）
}

//文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
//src : 分割したい文字列
//dest: 分割された文字列
//delim: 区切り文字
//splitNum : 最大分割数
void splitString(const char* src, char* dest[], const char* delim, int splitNum)
{
  //省略（前回と同じ）
}

//デバッグ用
//elemの要素を表示
void printElements(char* elem[])
{
  //省略（前回と同じ）
}

//スクリプト文を解読する
//戻り値 1: 成功  0: 失敗
int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo)
{
  int i, selectNum, choice;
  //分割されたスクリプト文
  char* message[100];
  //条件分岐用
  char* selectMessage[10];
  char* select[10][2];

  //文字列分割時の区切り文字
  const char* delim = " ";
  const char* selectDelim = "@@";

  //スクリプト分割
```

```
65    splitString( scriptMessage, message, delim, 100 );
66
67    //分割に失敗した場合
68    if( message[0] == NULL ) {
69      return 0;
70    }
71
72    //scriptの仕様
73    //
74    //@@message 文字列
75    //--- 文字列をメッセージとして表示する
76    //@@select 条件1の場合@@LABEL1 条件2の場合@@LABEL2 条件3の場合@@LABEL3
77    //--- 条件分岐
78
79    //message[0] が @@message の時は，メッセージ命令が来たと判断
80    if( strncmp(message[0], "@@message", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
81      printf("メッセージ : %s\n", message[1] );
82      return 1;
83    }else if( strncmp(message[0], "@@drawgraph", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
84      printf("画像 %s 表示 -- x座標 : %d, y座標 : %d\n",
85        message[3], atoi( message[1] ), atoi( message[2] ) );
86      return 1;
87    }else if( strncmp(message[0], "@@select", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
88
89      for(i = 0; message[i + 1] != NULL; i++ ) {
90        //条件を条件文章とジャンプラベルとに分ける
91        splitString( message[i + 1], selectMessage, selectDelim, 2 );
92        //条件文章
93        select[i][0] = selectMessage[0];
94        //ラベル
95        select[i][1] = selectMessage[1];
96      }
97      //分岐数
98      selectNum = i;
99
100     //ここで分岐を画面に表示
101     for(i = 0; i < selectNum; i++) {
102       printf("条件 %d : %s\n", i + 1, select[i][0] );
103     }
104     //分岐を選択
105     choice = 0;
106     while( choice <= 0 || choice > selectNum ) {
107       printf("選択 :");
108       scanf("%d", &choice );
109     }
110
111     //ここで条件へジャンプする処理を追加
112     printf("%s へジャンプします\n", select[choice - 1][1] );
113
114     return 1;
115   }
116
117   printf("スクリプトエラー\n");
118   return 0;
119 }
120
121 //指定したラベルがある行数を探す
122 //戻り値 正の数: 指定したラベルの行番号 -1: エラー
123 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo)
124 {
125   //省略（前回と同じ）
126 }
127
128 int main()
129 {
```

```
130   int i;
131   int line;
132   ScriptInformation script;
133
134   loadScript( "./script.txt", &script );
135
136   for( i = 0; i < script.maxLineNumber ; i++ ) {
137     printf("%d : %s\n", i + 1, script.script[i] );
138   }
139
140   for( i = 0; decodeScript( script.script[i], &script ) != 0 ; i++ )
141     ;
142
143   return 0;
144 }
```

実行結果は以下のようになりました．今回は例として「条件 2 の場合」を選択してみました．なお，今回使用した script.txt は以下のようになっています．

```
1   @@message こんにちは，文章表示のテストです
2
3   @@select 条件 1 の場合@@LABEL1 条件 2 の場合@@LABEL2 条件 3 の場合@@LABEL3
4
5   @@label LABEL1
6
7           @@message 条件 1 が選択されました
8           @@goto NEXT
9
10  @@label LABEL2
11
12          @@message 条件 2 が選択されました
13          @@goto NEXT
14
15  @@label LABEL3
16
17          @@message 条件 3 が選択されました
18          @@goto NEXT
19
20  @@label NEXT
21
22  @@message これで終了します
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです
2 : @@select 条件 1 の場合@@LABEL1 条件 2 の場合@@LABEL2 条件 3 の場合
@@LABEL3
3 : @@label LABEL1
4 : @@message 条件 1 が選択されました
5 : @@goto NEXT
6 : @@label LABEL2
7 : @@message 条件 2 が選択されました
8 : @@goto NEXT
9 : @@label LABEL3
10 : @@message 条件 3 が選択されました
11 : @@goto NEXT
12 : @@label NEXT
13 : @@message これで終了します
メッセージ : こんにちは，文章表示のテストです
条件 1 : 条件 1 の場合
条件 2 : 条件 2 の場合
条件 3 : 条件 3 の場合
選択 :2
LABEL2 へジャンプします
スクリプトエラー
```

9.7 指定したラベルへのジャンプ

指定したラベルへジャンプするプログラムを作成していきます．これは，@@goto 命令，@@select 命令どちらにも利用することができるでしょう．

指定したラベルへ移動するのは意外と簡単です．まずは，指定したラベルが何行目にあるかを取得します．これは，§ラベルの検索で既に作成しましたね．そして，ScriptInformation 構造体内にある現在読み取り中の行を表す currentLine を指定したラベルが存在する行に置き換えればよいだけです．

今回は指定したラベルへジャンプする命令@@goto 命令を作成することとします．@@goto 命令の構造は図 9.5 となっています．

では作成したプログラムをお見せします．今回も書き換えたのは decodeScript 関数の一部分だけです．

リスト 9.7: ”script-07.cpp”

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
```

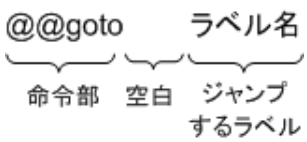

図 9.5: @@goto の構造

```
5  //スクリプトは最大1000行まで読み込む
6  #define SCRIPT_MAX_LINE 1000
7  //スクリプト最大文字数
8  #define SCRIPT_MAX_STRING_LENGTH 300
9
10 typedef struct ScriptInformation_tag {
11   int maxLineNumber;      //スクリプト行数
12   int currentLine;      //現在何行目を実行しているか
13   const char* filename;   //ファイル名
14   char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
15 } ScriptInformation;
16
17
18 //プロトタイプ宣言
19 int loadScript(const char* filename, ScriptInformation* scriptInfo);
20 void splitString(const char* src, char* dest[], const char* delim, int splitNum);
21 void printElements(char* elem[]);
22 int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo);
23 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo);
24
25 //スクリプトファイルを読み込む
26 //戻り値 -1 : 失敗  0 : 成功
27 int loadScript(const char* filename, ScriptInformation* scriptInfo)
28 {
29   //省略（前回と同じ）
30 }
31
32 //文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
33 //src : 分割したい文字列
34 //dest: 分割された文字列
35 //delim: 区切り文字
36 //splitNum : 最大分割数
37 void splitString(const char* src, char* dest[], const char* delim, int splitNum)
38 {
39   //省略（前回と同じ）
40 }
41
42 //デバッグ用
43 //elemの要素を表示
44 void printElements(char* elem[])
45 {
46   //省略（前回と同じ）
47 }
48
49 //スクリプト文を解読する
50 //戻り値 1: 成功  0: 失敗
51 int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo)
52 {
53   int i, selectNum, choice, line;
54   //分割されたスクリプト文
55   char* message[100];
56   //条件分岐用
57   char* selectMessage[10];
58   char* select[10][2];
```

```
59
60    //文字列分割時の区切り文字
61    const char* delim = " ";
62    const char* selectDelim = "@@";
63
64    //スクリプト分割
65    splitString( scriptMessage, message, delim, 100 );
66
67    //分割に失敗した場合
68    if( message[0] == NULL ) {
69      return 0;
70    }
71
72    //scriptの仕様
73    //
74    //@@message 文字列
75    //--- 文字列をメッセージとして表示する
76    //@@select 条件1の場合@@LABEL1 条件2の場合@@LABEL2 条件3の場合@@LABEL3
77    //--- 条件分岐
78
79    //message[0] が @@message の時は，メッセージ命令が来たと判断
80    if( strncmp(message[0], "@@message", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
81      printf("メッセージ : %s\n", message[1] );
82
83    }else if( strncmp(message[0], "@@drawgraph", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
84      printf("画像 %s 表示 -- x座標 : %d, y座標 : %d\n",
85        message[3], atoi( message[1] ), atoi( message[2] ) );
86
87    }else if( strncmp(message[0], "@@select", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
88
89      for(i = 0; message[i + 1] != NULL; i++ ) {
90        //条件を条件文章とジャンプラベルとに分ける
91        splitString( message[i + 1], selectMessage, selectDelim, 2 );
92        //条件文章
93        select[i][0] = selectMessage[0];
94        //ラベル
95        select[i][1] = selectMessage[1];
96      }
97      //分岐数
98      selectNum = i;
99
100     //ここで分岐を画面に表示
101     for(i = 0; i < selectNum; i++) {
102       printf("条件 %d : %s\n", i + 1, select[i][0] );
103     }
104     //分岐を選択
105     choice = 0;
106     while( choice <= 0 || choice > selectNum ) {
107       printf("選択 :");
108       scanf("%d", &choice );
109     }
110
111     //ここで条件へジャンプする処理を追加
112     printf("%s へジャンプします\n", select[choice - 1][1] );
113
114   }else if( strncmp(message[0], "@@goto", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
115     //ラベルが何行目にあるかを取得
116     line = searchScriptLabel( message[1], scriptInfo );
117     //指定したラベルが見つからなかった
118     if( line == -1 ) {
119       printf("スクリプトエラー:指定したラベルが見つかりませんでした(%d行目)\n",
120         scriptInfo->currentLine );
121       return 0;
122     }
123     //読み取り中の行番号をラベルの行に移動
```

```
124     scriptInfo->currentLine = line;
125
126   }else if( strncmp(message[0], "@@label", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
127     //ラベルの場合はなにもしない
128   }else {
129     printf("スクリプトエラー(%d行目)\n", scriptInfo->currentLine);
130     return 0;
131   }
132   return 1;
133 }
134
135 //指定したラベルがある行数を探す
136 //戻り値 正の数: 指定したラベルの行番号 -1: エラー
137 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo)
138 {
139   //省略（前回と同じ）
140 }
141
142 int main()
143 {
144   int i;
145   int line;
146   ScriptInformation script;
147
148   loadScript( "./script.txt", &script );
149
150   for( i = 0; i < script.maxLineNumber ; i++ ) {
151     printf("%d : %s\n", i + 1, script.script[i] );
152   }
153
154   for( ; decodeScript( script.script[ script.currentLine ], &script ) != 0 ;
155            script.currentLine++ )
156     ;
157
158   return 0;
159 }
```

実行結果は以下のようになりました．なお，今回使用した script.txt は以下のようになっています．

```
1  @@message こんにちは，文章表示のテストです
2
3  @@goto LABEL1
4
5  @@message この行は表示されないはず
6
7  @@label LABEL1
8
9  @@message これで終了します
```

実行結果

```
1 : @@message こんにちは，文章表示のテストです
2 : @@goto LABEL1
3 : @@message この行は表示されないはず
4 : @@label LABEL1
5 : @@message これで終了します
メッセージ : こんにちは，文章表示のテストです
メッセージ : これで終了します
```

9.8 ノベルゲーム用スクリプト言語解析プログラム

ようやくノベルゲーム用のスクリプト言語の完成です．今回は，@@select 構文にラベルジャンプ機能を付けてみます．

では完成したプログラムをお見せします．今回はコードを省略せずにすべて書いています．

リスト 9.8: ”script-08.cpp”

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  //スクリプトは最大1000行まで読み込む
6  #define SCRIPT_MAX_LINE 1000
7  //スクリプト最大文字数
8  #define SCRIPT_MAX_STRING_LENGTH 300
9
10 typedef struct ScriptInformation_tag {
11   int maxLineNumber;      //スクリプト行数
12   int currentLine;      //現在何行目を実行しているか
13   const char* filename;   //ファイル名
14   char script[SCRIPT_MAX_LINE][SCRIPT_MAX_STRING_LENGTH];
15 } ScriptInformation;
16
17
18 //プロトタイプ宣言
19 int loadScript(const char* filename, ScriptInformation* scriptInfo);
20 void splitString(const char* src, char* dest[], const char* delim, int splitNum);
21 void printElements(char* elem[]);
22 int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo);
23 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo);
24
25 //スクリプトファイルを読み込む
26 //戻り値 -1 : 失敗  0 : 成功
27 int loadScript(const char* filename, ScriptInformation* scriptInfo)
28 {
29   int pos;
30   char c;
31   //スクリプトファイル
32   FILE* fp;
33
34   //スクリプト情報を初期化
35   memset( scriptInfo , 0, sizeof(ScriptInformation) );
36
37   //スクリプトファイルを開く
```

```
38    fp = fopen(filename, "r");
39    if( fp == NULL ) {
40      //ファイル読み込みに失敗
41      printf("スクリプト %s を読み込めませんでした\n", filename);
42      return -1;
43    }
44
45    //script書き込み時に使用
46    pos = 0;
47
48    for( ;; ) {
49      //一文字読み込み
50      c = fgetc( fp );
51      //ファイルの終わりかどうか
52      if( feof( fp ) ) {
53        break;
54      }
55      //文章先頭の空白部分を読み飛ばす
56      while( (c == ' ' || c == '\t') && pos == 0 && !feof( fp ) ) {
57        c = fgetc( fp );
58      }
59
60      if( pos >= SCRIPT_MAX_STRING_LENGTH - 1 ) {
61        //1行の文字数が多すぎる
62        printf("error: 文字数が多すぎます (%d行目)", scriptInfo->currentLine );
63        return -1;
64      }
65
66      //改行文字が出てきた場合, 次の行へ移動
67      if( c == '\n' ) {
68        //空行は読み飛ばす
69        if( pos == 0 ) {
70          continue;
71        }
72        //\0を文字列の最後に付ける
73        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = '\0';
74        //次の行に移動
75        scriptInfo->currentLine++;
76        //書き込み位置を0にする
77        pos = 0;
78      }else {
79        //書き込み
80        scriptInfo->script[ scriptInfo->currentLine ][ pos ] = c;
81        //文字書き込み位置をずらす
82        pos++;
83      }
84    }
85    //最大行数
86    scriptInfo->maxLineNumber = scriptInfo->currentLine;
87    //読み込み中の行を0にする
88    scriptInfo->currentLine = 0;
89    //スクリプトファイル名を設定
90    scriptInfo->filename = filename;
91
92    return 0;
93  }
94
95  //文字列分割(1行の最大文字数は SCRIPT_MAX_STRING_LENGTH)
96  //src : 分割したい文字列
97  //dest: 分割された文字列
98  //delim: 区切り文字
99  //splitNum : 最大分割数
100 void splitString(const char* src, char* dest[], const char* delim, int splitNum)
101 {
102   int i;
```

```
103   char* cp;
104   char* copySrc;
105
106   //元の文章をコピーする
107   copySrc = (char*)malloc( sizeof(int) * SCRIPT_MAX_STRING_LENGTH + 1);
108   strncpy( copySrc, src, SCRIPT_MAX_STRING_LENGTH );
109   cp = copySrc;
110
111   //strtokを使って copySrc をdelim区切りで分割する
112   for( i = 0; i < splitNum ; i++ ) {
113     //分割対象文字列が無くなるまで分割
114     if( (dest[i] = strtok(cp, delim)) == NULL ) {
115       break;
116     }
117     //2回目にstrtokを呼び出す時は, cpをNULLにする
118     cp = NULL;
119   }
120   //分割された文字列の最後の要素はNULLとしておく
121   dest[i] = NULL;
122 }
123
124 //デバッグ用
125 //elemの要素を表示
126 void printElements(char* elem[])
127 {
128   int i;
129   for( i = 0; elem[i] != NULL; i++ ) {
130     printf("%d : %s\n", i + 1, elem[i] );
131   }
132 }
133
134 //スクリプト文を解読する
135 //戻り値 1: 成功  0: 失敗
136 int decodeScript(const char* scriptMessage, ScriptInformation* scriptInfo)
137 {
138   int i, selectNum, choice, line;
139   //分割されたスクリプト文
140   char* message[100];
141   //条件分岐用
142   char* selectMessage[10];
143   char* select[10][2];
144
145   //文字列分割時の区切り文字
146   const char* delim = " ";
147   const char* selectDelim = "@@";
148
149   //スクリプト分割
150   splitString( scriptMessage, message, delim, 100 );
151
152   //分割に失敗した場合
153   if( message[0] == NULL ) {
154     return 0;
155   }
156
157   //scriptの仕様
158   //
159   //@@message 文字列
160   //--- 文字列をメッセージとして表示する
161   //@@select 条件1の場合@@LABEL1 条件2の場合@@LABEL2 条件3の場合@@LABEL3
162   //--- 条件分岐
163
164   //message[0] が @@message の時は, メッセージ命令が来たと判断
165   if( strncmp(message[0], "@@message", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
166     printf("メッセージ : %s\n", message[1] );
167
```

```
168   }else if( strncmp(message[0], "@@drawgraph", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
169     printf("画像 %s 表示 -- x座標 : %d, y座標 : %d\n",
170       message[3], atoi( message[1] ), atoi( message[2] ) );
171
172   }else if( strncmp(message[0], "@@select", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
173
174     for(i = 0; message[i + 1] != NULL; i++ ) {
175       //条件を条件文章とジャンプラベルとに分ける
176       splitString( message[i + 1], selectMessage, selectDelim, 2 );
177       //条件文章
178       select[i][0] = selectMessage[0];
179       //ラベル
180       select[i][1] = selectMessage[1];
181     }
182     //分岐数
183     selectNum = i;
184
185     //ここで分岐を画面に表示
186     for(i = 0; i < selectNum; i++) {
187       printf("条件 %d : %s\n", i + 1, select[i][0] );
188     }
189     //分岐を選択
190     choice = 0;
191     while( choice <= 0 || choice > selectNum ) {
192       printf("選択 :");
193       scanf("%d", &choice );
194     }
195
196     //ラベルが何行目にあるかを取得
197     line = searchScriptLabel( select[choice - 1][1] , scriptInfo );
198     //指定したラベルが見つからなかった
199     if( line == -1 ) {
200       printf("スクリプトエラー:条件分岐の指定ラベルが間違っています(%d行目)\n",
201         scriptInfo->currentLine + 1 );
202       return 0;
203     }
204     //読み取り中の行番号をラベルの行に移動
205     scriptInfo->currentLine = line;
206
207   }else if( strncmp(message[0], "@@goto", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
208     //ラベルが何行目にあるかを取得
209     line = searchScriptLabel( message[1], scriptInfo );
210     //指定したラベルが見つからなかった
211     if( line == -1 ) {
212       printf("スクリプトエラー:指定したラベルが見つかりませんでした(%d行目)\n",
213         scriptInfo->currentLine + 1);
214       return 0;
215     }
216     //読み取り中の行番号をラベルの行に移動
217     scriptInfo->currentLine = line;
218
219   }else if( strncmp(message[0], "@@label", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
220     //ラベルの場合はなにもしない
221   }else {
222     printf("スクリプトエラー(%d行目)\n", scriptInfo->currentLine + 1);
223     return 0;
224   }
225   return 1;
226 }
227
228 //指定したラベルがある行数を探す
229 //戻り値 正の数: 指定したラベルの行番号 -1: エラー
230 int searchScriptLabel(const char* label, ScriptInformation* scriptInfo)
231 {
232   //分割されたスクリプト文
```

```
233   char* message[100];
234   //文字列分割時の区切り文字
235   const char* delim = " ";
236   int i, line = -1;
237
238   for( i = 0; i < scriptInfo->maxLineNumber; i++ ) {
239     //スクリプト分割
240     splitString( scriptInfo->script[i] , message, delim, 100 );
241
242     //分割に失敗した場合
243     if( message[0] == NULL ) {
244       return -1;
245     }
246
247     //指定したラベルを探す
248     if( strncmp(message[0], "@@label", SCRIPT_MAX_STRING_LENGTH) == 0 ) {
249       if( strncmp(message[1], label, SCRIPT_MAX_STRING_LENGTH) == 0 ) {
250         //指定したラベルが見つかった時
251         line = i;
252         break;
253       }
254     }
255   }
256
257   return line;
258 }
259
260 int main()
261 {
262   int i;
263   int line;
264   ScriptInformation script;
265
266   loadScript( "./script.txt", &script );
267
268   printf("\nスクリプト開始\n\n");
269
270   for( ; decodeScript( script.script[ script.currentLine ], &script ) != 0 ;
271            script.currentLine++ )
272     ;
273
274   return 0;
275 }
```

実行結果は以下のようになります．なお，今回使用した script.txt は以下のようになっています．

```
1           @@message こんにちは
2           @@message ノベルゲーム用スクリプトエンジンのテストです
3
4           @@message まずは条件分岐です
5
6           @@message あなたの年齢を教えてください
7
8           @@select 20歳未満@@UNDER 20歳以上@@UPPER
9
10   @@label UNDER
11          @@message あなたは20歳未満なのですね!
```

```
12          @@goto NEXT
13
14  @@label UPPER
15          @@message あなたは 20 歳以上なのですね!
16          @@goto NEXT
17
18  @@label NEXT
19
20          @@message 画像 human.png を表示しますか
21
22          @@select はい@@YES いいえ@@NO
23
24  @@label YES
25          @@message といっても本当に画像が表示されるわけではないのですが …
26          @@drawgraph 50 100 human.png
27          @@goto NEXT2
28
29  @@label NO
30          @@goto NEXT2
31
32  @@label NEXT2
33
34          @@message まだまだこのスクリプトエンジンは改良の余地があります
35          @@message 例えば，フラグの値によって条件を分岐したり，などです
36          @@message その辺りは読者の方がご自身で実装してみてください
37          @@message がんばればできます
```

実行結果

```
スクリプト開始

メッセージ : こんにちは
メッセージ : ノベルゲーム用スクリプトエンジンのテストです
メッセージ : まずは条件分岐です
メッセージ : あなたの年齢を教えてください
条件 1 : 20 歳未満
条件 2 : 20 歳以上
選択 :1
メッセージ : あなたは 20 歳未満なのですね!
メッセージ : 画像 human.png を表示しますか
条件 1 : はい
条件 2 : いいえ
選択 :1
メッセージ : といっても本当に画像が表示されるわけではないのですが …
画像 human.png 表示 -- x 座標 : 50, y 座標 : 100
メッセージ : まだまだこのスクリプトエンジンは改良の余地があります
メッセージ : 例えば, フラグの値によって条件を分岐したり, などです
メッセージ : その辺りは読者の方がご自身で実装してみてください
メッセージ : がんばればできます
```

9.9 ノベルゲーム用スクリプト言語解析プログラムの改良