○課題

　出来るだけ工夫して、iPhoneの画面に近づける事

○作成した画面



○プログラムの説明

・ウィンドウプロシージャ

　ウィンドウが作られると、ビットマップをロードする。ロードする際はImagePackage構造体(IMPACKに名前を変えている)にloadImage関数を使うことで行う。

　IMPACK構造体はリストに表示するテキストを表すimageNameと画像の名前を表すfileName、そして画像を保持するhCompatDCの三つの変数を持つ。

loadImage関数ではIMPACK構造体のhCompatDCに画像をロードする。その際にはIMPACK構造体のfileNameの示す画像をロードする。もしロードできなかった場合はメッセージボックスを出現させ、その旨をユーザーに伝える。

　リスト要素については、IMPACKを配列で保持しておき、ループを使用してすべて描画する。描画する際はリストの要素を描画するのを担当するDrawTextAndRect関数を使用する。

　DrawTextAndRect関数はリストの要素を表示するのに使う。引数には表示する対象のハンドルデバイスコンテキスト（以降HDC）と、画像のHDC、さらには表示する左上座標をそれぞれoffsetSX,offsetSYに、リストの文字をtext、そして画像を表示するかを表すendフラグを指定する。endフラグはtrueの時に画像を表示しない。

　最後には時間を表示する。まずローカルな時間を取得し、時間と分をTCHAR型配列に変換する。その際二ケタ表示で空いた部分は0詰めするようにフォーマット指定子にて指定する。その後時間と分の間に : を入れたテキストをcTime変数に格納し、DrawTextEx関数にて描画している。

○工夫した点

　画像のロードは一時変数を2個使う(HDCとHBITMAP)。これは使いまわせる。また最終的に描画に必要なのはHDC型の変数であるので、ロードする処理を関数にまとめ、余計な一時変数や長い処理を関数一行で書けるようにし、可読性を高めた。またロードする際、構造体に必要な情報をまとめる事で、配列にて一気にロード、描画できるようにした。これにより要素の追加が用意になると考えられる。

　またリストの表示の際、ループはIMPACKの配列の要素を算出するようにしているため、要素が増えてもエラーが出ることは無い。

○考察

　今回作成したプログラムはC言語なのでオブジェクト指向ではなく、javaやc#に用意されている便利な関数やプロパティは無く、それらをすべて自分で実装する必要があった（配列の要素数の算出など）。またクラスに分割できないので同じ処理は関数に分けたが、関数が多くなれば可読性が落ちると考えられる。実際今回作成したプログラムは、WM\_PAINT内に処理が集中していて読みにくくなっている。またパラメータは内部に直接書き込んでいるので、UIの変更などがあった場合、プログラム側での変更が容易でない。この点、オブジェクト指向にすれば、それぞれのパラメータをクラスに分けることでリスト部や、検索フォームなどの変更を容易に行うことが出来ると考えられる。

　またクラスに分けることで、描画処理も分ける事が出来、WM\_PAINTに処理が集中することも避けられると考えられる。

　今回の課題を通してオブジェクト指向が実に便利である事が理解できた。しかし、C言語やWINAPIなどの便利なものの裏側を知ることは、今後プログラマを目指すものとしてはより便利なものを考えるためには必要で、今回の課題は有意義なものだと考える。

○作成したソースコード

//課題

/\*---------------------------------------------

Win32AP016.cpp

-----------------------------------------------\*/

#include <windows.h>

#include <tchar.h>

enum COMPONENTS{

BACKGROUND,

STATAS,

BACKICONBACKGROUND,

BACKICON,

};

// プロトタイプ宣言

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

// 大域変数

static TCHAR szWindowClass[] = \_T("課題");

static TCHAR szTitle[] = \_T("課題");

HINSTANCE hInst;

int WINAPI WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR pCmdLine,

int nCmdShow) {

WNDCLASSEX wcex;

// ウィンドウクラスの情報を設定

wcex.cbSize = sizeof(WNDCLASSEX); // 構造体サイズ

wcex.style = CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS; // スタイル

wcex.lpfnWndProc = WndProc; // ウィンドウプロシージャ

wcex.cbClsExtra = 0; // 拡張情報１

wcex.cbWndExtra = 0; // 拡張情報２

wcex.hInstance = hInstance; // インスタンスハンドル

wcex.hIcon = (HICON)LoadImage( // アイコン

NULL,

MAKEINTRESOURCE(IDI\_APPLICATION),

IMAGE\_ICON,

0,

0,

LR\_DEFAULTSIZE | LR\_SHARED

);

wcex.hIconSm = wcex.hIcon; // 子アイコン

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL; // メニュー名

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_APPLICATION));

// ウィンドウクラスを登録する

if (!RegisterClassEx(&wcex)) {

MessageBox(NULL,

\_T("ウィンドウの登録に失敗しました"),

\_T("ウィンドウの登録エラー"),

NULL);

return 1;

}

hInst = hInstance; // インスタンス･ハンドルを大域変数に格納

// ウィンドウを作成する

HWND hWnd = CreateWindow(

szWindowClass, // ウィンドウクラス名

szTitle, // タイトルバーに表示する文字列

WS\_OVERLAPPEDWINDOW, // ウィンドウの種類

100, // ウィンドウを表示する位置(X座標)

100, // ウィンドウを表示する位置(Y座標)

320, // ウィンドウの幅

568, // ウィンドウの高さ

NULL, // 親ウィンドウのウィンドウハンドル

NULL, // メニューハンドル

hInst, // インスタンスハンドル

NULL // その他の作成データ

);

if (!hWnd) {

MessageBox(NULL,

\_T("ウィンドウの作成に失敗しました"),

\_T("ウィンドウの作成エラー"),

NULL);

return 1;

}

// ウィンドウを表示する

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

// メッセージ･ループ

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

//矩形と文字列を表示する代行関数

void DrawTextAndRect(HDC hDC, HDC hCompatDC, int offsetSX, int offsetSY, LPCTSTR text, bool end){

static int xOffset = 9;

static int yOffset = 117;

static int hight = 47;

static int width = 284;

SetTextColor(hDC, RGB(0, 0, 0));

RECT rc;

SetRect(&rc,

xOffset + offsetSX,

yOffset + offsetSY,

width + xOffset + offsetSX,

hight + yOffset + offsetSY);

Rectangle(hDC, rc.left, rc.top, rc.right, rc.bottom);

if (!end){

// ビットマップを２倍に拡大して、ウィンドウに転送する(表示する)

StretchBlt(hDC, rc.left + 1, rc.top + 1, 47 - 2, 47 - 2, hCompatDC, 0, 0, 64, 64, SRCCOPY);

}

SetRect(&rc,

60 + xOffset + offsetSX,

13 + yOffset + offsetSY,

width + xOffset + offsetSX,

hight + yOffset + offsetSY);

//SetBkColor(hDC, RGB(128, 128, 255));

DrawText(hDC, text, -1, &rc, DT\_LEFT);

}

//画像読み込み関数

int loadImage(LPCWSTR fileName, HDC hDC, HDC\* hCompatDC){

HBITMAP bitmap, hPrevBitmap;

bitmap = (HBITMAP)LoadImage(

NULL,

fileName,

IMAGE\_BITMAP,

0,

0,

LR\_LOADFROMFILE);

if (bitmap == NULL) {

return -1;

}

// メモリデバイスコンテキストを作成する

\*hCompatDC = CreateCompatibleDC(hDC);

// ロードしたビットマップを選択する

hPrevBitmap = (HBITMAP)SelectObject(\*hCompatDC, bitmap);

return 0;

}

typedef struct ImagePackage{

LPCTSTR imageName;

LPCTSTR lavel;

HDC hCompatDC;

}IMPACK;

// ウィンドウプロシージャ

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

//リストの名前

static IMPACK impack\_comp[] = {

{ \_T("background2.bmp"), 0, 0 },

{ \_T("statas.bmp"), 0, 0 },

{ \_T("backicon\_background.bmp"), 0, 0 },

{ \_T("icons.bmp"), 0, 0 }

};

static IMPACK impack\_item[] = {

{ \_T("rapyuta.bmp"), \_T("天空の城ラピュタ"), 0 },

{ \_T("totoro.bmp"), \_T("となりのトトロ"), 0 },

{ \_T("hotaru.bmp"), \_T("火垂るの墓"), 0 },

{ \_T("majo.bmp"), \_T("魔女の宅急便"), 0 },

{ \_T("omoide.bmp"), \_T("おもひでぽろぽろ"), 0 },

{ \_T("kurenai.bmp"), \_T("紅の豚"), 0 }

};

static LPCTSTR list\_name = \_T("ジブリ映画");

static LPCTSTR list\_header = \_T("リスト");

TCHAR hour[64];

TCHAR minute[64];

TCHAR cTime[128];

SYSTEMTIME time;

PAINTSTRUCT ps;

HBRUSH hBrush;

RECT rc;

static HDC hDC;

int i;

int upper = 70;

int lower = 114;

hDC = GetDC(hWnd);

switch (message) {

case WM\_CREATE:

for (int n = 0; n< sizeof(impack\_comp) / sizeof(IMPACK); n++){

if (loadImage(impack\_comp[n].imageName, hDC, &(impack\_comp[n].hCompatDC)) == -1){

MessageBox(

hWnd,

\_T("ビットマップのロードに失敗しました"),

impack\_comp[n].imageName,

MB\_OK | MB\_ICONWARNING

);

}

}

for (int n = 0; n< sizeof(impack\_item) / sizeof(IMPACK); n++){

if (loadImage(impack\_item[n].imageName, hDC, &(impack\_item[n].hCompatDC)) == -1){

MessageBox(

hWnd,

\_T("ビットマップのロードに失敗しました"),

impack\_item[n].imageName,

MB\_OK | MB\_ICONWARNING

);

}

}

break;

case WM\_PAINT:

hDC = BeginPaint(hWnd, &ps); // GDI関数による描画を開始する

HBRUSH hBrush;

GetClientRect(hWnd, &rc);

//べた塗りブラシの作成

hBrush = CreateSolidBrush(RGB(192, 192, 255));

SelectObject(hDC, hBrush);

//背景の描画

Rectangle(hDC, rc.left, rc.top, rc.right, rc.bottom);

//ブラシの開放

DeleteObject(hBrush);

//背景画像の描画

StretchBlt(hDC, 0, 0, 320 - 18, 598 - 70, impack\_comp[BACKGROUND].hCompatDC, 0, 0, 320, 480, SRCCOPY);

//ステータスバーの表示

StretchBlt(hDC, 0, 0, 320 - 18, 20, impack\_comp[STATAS].hCompatDC, 0, 0, 320, 20, SRCCOPY);

//検索フォーム部分の表示

StretchBlt(hDC, 0, 20, 302, 50, impack\_comp[BACKICONBACKGROUND].hCompatDC, 0, 0, 320, 44, SRCCOPY);

SetRect(&rc, 110, 35, 320, 60);

//文字色を変更（白）

SetTextColor(hDC, RGB(255, 255, 255));

//文字の背景を透過

SetBkMode(hDC, TRANSPARENT);

DrawText(hDC, list\_name, -1, &rc, DT\_LEFT);

//下のアイコンの表示

//べた塗りブラシの作成

hBrush = CreateSolidBrush(RGB(255, 255, 255));

SelectObject(hDC, hBrush);

Rectangle(hDC, 0, 480, 303, 529);

//検索フォームの表示

hBrush = CreateSolidBrush(RGB(0, 0, 0));

SelectObject(hDC, hBrush);

Rectangle(hDC, 0, 70, 305, 115);

hBrush = CreateSolidBrush(RGB(255, 255, 255));

SelectObject(hDC, hBrush);

Rectangle(hDC, 301 / 10 \* 2, 75, 301 / 10 \* 8, 110);

StretchBlt(hDC, 301 / 10 \* 1 - 30 / 2, 78, 30, 30, impack\_comp[BACKICON].hCompatDC, 106, 182, 72, 72, SRCCOPY);

StretchBlt(hDC, 301 / 10 \* 9 - 30 / 2, 78, 30, 30, impack\_comp[BACKICON].hCompatDC, 601, 759, 72, 72, SRCCOPY);

hBrush = CreateSolidBrush(RGB(255, 255, 255));

SelectObject(hDC, hBrush);

//ボタンの表示

//バックアイコンの表示

StretchBlt(hDC, 301 / 8 \* 1 - 45 / 2, 482, 45, 45, impack\_comp[BACKICON].hCompatDC, 933, 183, 72, 72, SRCCOPY);

StretchBlt(hDC, 301 / 8 \* 3 - 45 / 2, 482, 45, 45, impack\_comp[BACKICON].hCompatDC, 850, 183, 72, 72, SRCCOPY);

StretchBlt(hDC, 301 / 8 \* 5 - 45 / 2, 482, 45, 45, impack\_comp[BACKICON].hCompatDC, 106, 1093, 72, 72, SRCCOPY);

StretchBlt(hDC, 301 / 8 \* 7 - 45 / 2, 482, 45, 45, impack\_comp[BACKICON].hCompatDC, 188, 844, 72, 72, SRCCOPY);

for (int i = 0; i < sizeof(impack\_item) / sizeof(IMPACK); i++) {

if (i >= 6){

break;

}

DrawTextAndRect(hDC, impack\_item[i].hCompatDC, 0, (47 + 2)\* i, impack\_item[i].lavel, false);

}

DrawTextAndRect(hDC, 0, 0, (47 + 2) \* 6, L"リストを追加", true);

//時間の表示

GetLocalTime(&time);

wsprintf(hour, TEXT("%02d"), time.wHour);

wsprintf(minute, TEXT("%02d"), time.wMinute);

SetRect(&rc, 130, 0, 260, 50);

SetBkMode(hDC, TRANSPARENT);

DrawTextEx(hDC, cTime, wsprintf(cTime, TEXT("%s : %s"), hour, minute), &rc, DT\_LEFT, 0);

ReleaseDC(hWnd, hDC);

DeleteObject(hBrush);

EndPaint(hWnd, &ps); // GDI関数による描画を終了する

break;

case WM\_MOUSEMOVE:

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

break;

}

return 0;

}