

WORD

2007.6

From College of Information Science

1

特集

UNIX

lolifox

vim

screen

Hello WORD! 号

WORD

Hello WORD Edition 2007

目次

目次	-----2p
編集長挨拶	-----3p
シューティングゲーム開発	-----4p
screen のススメ	-----19p
vim ! vim ! vim !	-----24p
□リ狐。	-----42p
書籍紹介	-----44p
編集後記	-----45p

編集長挨拶

文 編集部 Tait-you

さて、本日のお日柄がよいのかどうかは天気予報に任せるとしまして、それはとにかくにもどうも。情報学類誌から情報科学類誌に変わる際、役職も引き継いだため、情報科学類誌の初代編集長にもなってしまった Tait-you です。まあ、今回と次号だけの間ですが…。

と、言うわけで学群再編によってリニューアルを果たしました情報科学類誌 **WORD**、今回がリニューアル後の記念すべき一回目なので挨拶をしろと言われたのだでしょうがなく書いている今日この頃です。別にこれといって変わっている部分は今のところ無いような気がしますけどねえ…。

まあ、せっかくリニューアルを果たしたことだし、何か大きく変わることもあると思います。とりあえず、タイトルは変わりました。また、今まで年度が変わるごとに No.1 にリセットされたナンバリングが今回からリニューアル創刊から何号目かを表すようになりました。って、表紙だけじゃーん中身はどうなの？と言われるといつもどおりのような気がします。相変わらずほどほどにお馬鹿な記事を書いて、一体誰に需要があるのか分からない技術系の記事を書いていくという基本スタイルには一切の変更はありません。…と言いましたがこれを書いているのはなにぶん赤入れ前、どんな記事があるのかは現時点自分もさっぱり把握していません。それでいいのかと言われればそれでどうにかなっちゃうのが **WORD** のすごいところかもしれません。

…と与太話はこの辺にしておきましてここからは結構真面目な話。相変わらず軽い展開で進んでいく **WORD** ですが、結構重い話が無かったわけではありません。そして、この先もいつものようにのほほんと過ごしていけるか、と尋ねられると非常に微妙な部分もあるかもしれません。この先、筑波大学が潰えるその時まで **WORD** が生き延びられる確証ありませんし、下手したら今年で廃刊なんて可能性も無きにしもあらずな訳です。いや、冗談抜きで。

ですが、このように長い間、当編集部が **WORD** を発行を続けてきていたのは今、冊子を手に取っているあなたがいるからなのです。**WORD** にニーズがあるからこそ、これからも **WORD** は新しい記事を提供して、時にはへえという感嘆の言葉や爆笑、あるいはクスリとくる笑いを提供していくことが出来るのです。ですから、これからもどうぞ、**WORD** を愛し続けてくれることを自分は切に願います。

…って随分壮大な事をののしっちゃったけどこれってどうなのよ？そんなでっかいバックフィールドは本当にあるの？と最後にしっかりと惚けておいてこれにて締めとさせていただきます。

Hello **WORD**!!

シューティングゲーム開発講座

文 編集部 yasuharu

はじめまして。今年から WORD に参加することになった yasuharu です。これから先、いろいろとあるかもしれませんが、よろしくお願いします。このような記事を書くのは、はじめてですが、いろいろと勉強しながらがんばっていきたいと思っています。

さて、今回は「シューティングゲーム開発講座」ということで、Windows で C 言語を使ってシューティングゲームを開発する方法について書いていきます。対象としては、C 言語でのプログラミングが一通りできる方を対象としています。ですが、後からも述べますが、今の段階ではわかっていなくてもかまいません。プログラミングというのがどのようなものなのか、見ていただいて、興味を持っていただければいいと思っています。

記事の内容に移る前に注意点をあげておきたいと思います。この記事の内容については、私の知識不足などにより間違いがあるかもしれません。間違いの指摘などがありましたら、以下の連絡先まで遠慮無く連絡してください。

メールアドレス : word@yasuharu.net

また、この講座のデータは Web 上で公開します。指摘された間違いやこの雑誌上で掲載できなかった情報についても Web 上で公開します。

Web ページ : <http://yasuharu.net/word/>

目次

この記事の構成については、次の通りになります。

(太字の部分が今回の記事です)

1. はじめに	7. プレイヤーの弾の表示
1. 開発環境、開発言語	1. 弾の作成
2. 目標	2. 弾の移動
3. 検証環境	8. 敵とプレイヤーの弾との衝突判定
2. 環境の設定	1. 衝突判定
3. Window の表示	9. 敵の弾の表示
4. プレイヤーの機体の表示	1. 敵の行動
1. Player 構造体の定義	2. 弾の表示
2. プレイヤーの機体の表示	3. 敵の弾とプレイヤーとの衝突判定
5. プレイヤーの機体の移動	10. 最終章
1. タイマーの設定	1. 敵が上から流れてくるようにする
2. キーの取得	2. プレイヤー、敵、弾を画像を使って表示する
3. 移動範囲の限定	3. 画面のちらつきの制御をする
6. 敵の機体の表示	4. 背景色を変更する
1. 敵のチェインを作る	5. プレイヤーの弾に制限をかける
2. 敵の作成	
3. 敵の機体の表示	
	(5 章以降は予定です)

1. はじめに

1. 開発環境、開発言語

開発言語は C 言語を使います。開発に用いるソフトは、Visual Studio^{*1} 2005 Professional Edition です。また、描画^{*2} などについては DirectX^{*3} を使用せずに、Win32API を使って開発していきます。Win32API というのは、Windows の機能を使うためのライブラリ^{*4} です。これを使って、ウィンドウの作成や描画を行います。

なぜ、このような環境で開発を行うのか説明をします。

開発言語については、1 年生の 2 学期に情報科学類で開講される「プログラミング入門 1」において、C 言語の学習が行われるからです。また、情報科学類以外でも、工学系の学類であれば C 言語を学習する機会があると思います。授業で習った C 言語を使って、実際に動くものを作ってみよう、ということです。

まだ、この記事が発行されるのは 1 学期のうちだと思うので、C 言語がどんなものかわからない人もいられるかもしれません。しかし、急ぐ必要はありません。まずは、プログラミングがどのようなものかを見てください。そして、実際の授業で「ああ、こういうことなんだ」と、なんとなくわかってもらえればいいかと思います。

開発環境については、筑波大学の学生であれば、簡単に環境が用意することができるのでこのようにしました。環境を用意するには、次のような方法があります。

○筑波大学の学生

- ・教育用計算機システムの端末に入っている、Visual Studio 2005 Professional Edition を使用する。

○情報科学類、社会工学類、工学システム学類の学生

- ・MSDN のライセンスにより、Visual Studio 2005 Professional Edition を借りることができるので、それを借りる。

○情報科学類の学生

- ・学類の計算機室の Windows 端末に入っている、Visual Studio 2005 Professional Edition を使用する。

○全員

- ・Microsoft が無償で公開している Visual C++ 2005 Express Edition をダウンロードして使用する。

2. 目標

目標は、シューティングゲームとして最低限動くものを作っていくものを作っていくこと

*1 Visual Studio

Microsoft の Windows 用のプログラム開発を行うための統合開発環境。Visual C++、C#、Basic、J#の 4 つで構成される。

*2 描画

ウィンドウなどに対して描くことをここでは描画とっている。

*3 DirectX

Microsoft が提供しているライブラリ（ライブラリについては脚注参照）。これを使うと、3D の表示や音楽の再生など、ゲームプログラミングを簡単に開発できる。

*4 ライブラリ

特定の動作を行うための関数やデータの集まり。

です。ゲームとしては楽しめるかどうかはわかりません。もし、物足りないと感じたら、いろいろと改造してみてください。何度も試行錯誤をしているうちに、わかるようになってくるかもしれません。また、同時に C 言語に対する理解、Windows 用アプリケーションの開発方法、あるいは、プログラミングに対する興味など、知識や興味が増えてくれたらいいな、と思っています。最初のうちは難しいと思うかもしれません。しかし、何度も試行錯誤していくうちにだんだんと簡単になっていきます。がんばっていきましょう。

2. 環境の設定

まず、Visual Studio 2005 を用意してください。環境を準備する方法ですが、大学内の施設については、各施設の手引きを参照してください。Visual C++ 2005 Express Edition については、こちらのサイトを参照してください。

Visual Studio 2005 Express Editions: Visual C++ 2005 Express Edition と Microsoft Platform SDK を一緒に使う
<http://www.microsoft.com/japan/msdn/vstudio/express/visualc/usingpsdk/>

インストールが終わって、最初に起動するときに「既定の環境設定の選択」というウィンドウが表示されます。そこで、「Visual C++ 開発設定」を選択してください。

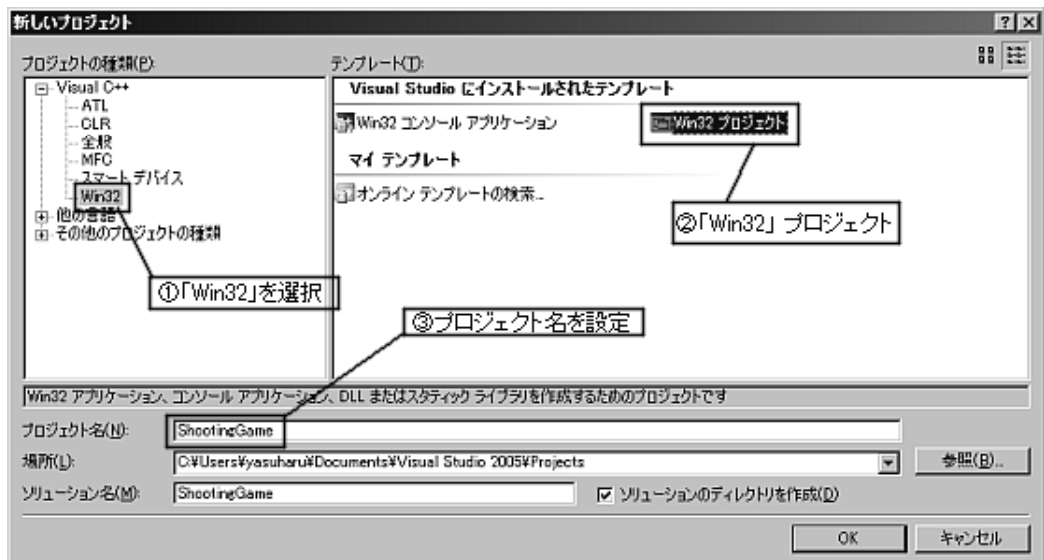


起動したらプロジェクト^{*1}の作成を行います。プロジェクトの作成を行うダイアログを出しましょう。次の順にクリックしてください。

「ファイル」メニューを開く → 「新規作成」 → 「プロジェクト」



開いたダイアログでは、「プロジェクトの種類」として「Win32 プロジェクト」を選びます。そして、適当なプロジェクト名を設定して、作成を行います。ここでは、「ShootingGame」というプロジェクト名で作成しています。



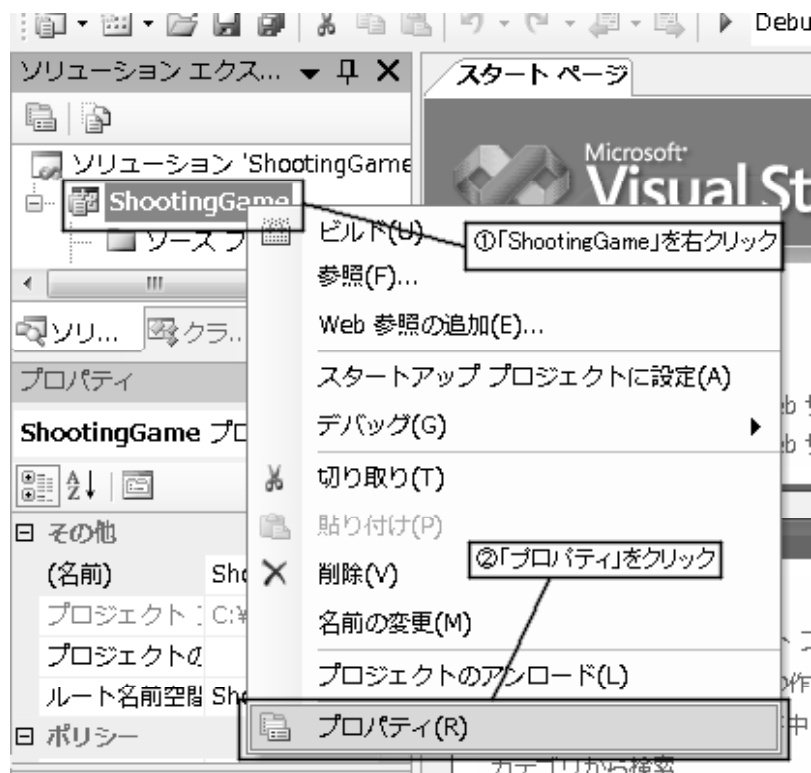
*1 プロジェクト

Visual Studioにおいて、プログラムを開発するときの1つのまとまりとしてソリューションというものがある。ソリューションは一つ以上のプロジェクトで構成される。

次に、開いたウィザードの画面では「アプリケーションの設定」をクリックして、「空のプロジェクト」にチェックをつけます。それが終わったら、「完了」を押しましょう。

次に、プロジェクトの設定を行うために、「プロパティ」を開きます。次の順にクリックしてください。

「ソリューションエクスプローラー」の「ShootingGame」を右クリック → 「プロパティ」

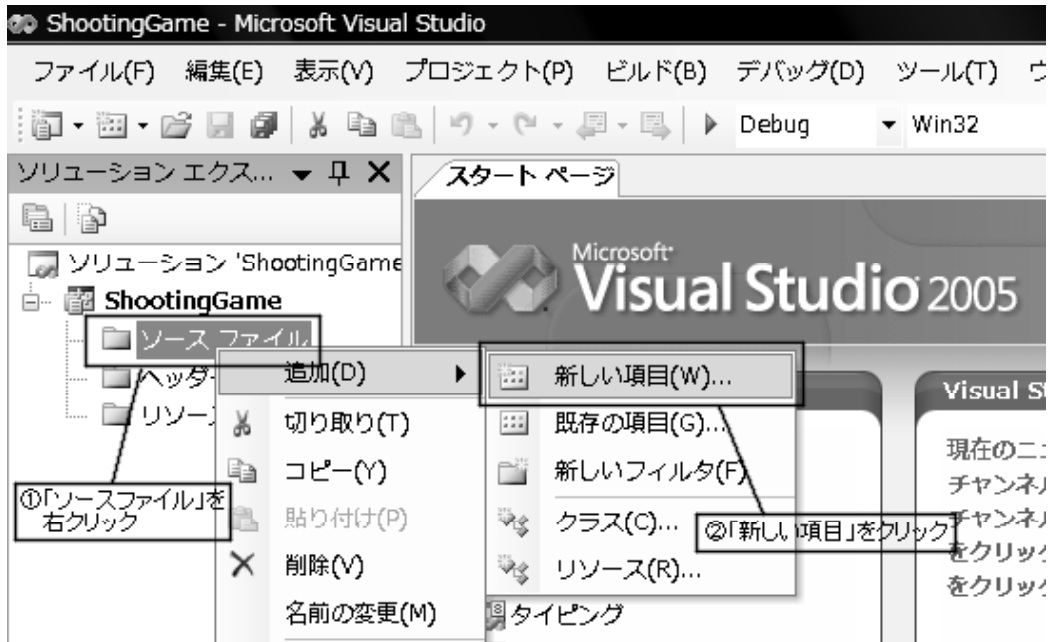


開いたら、左側のツリーの中から、「構成プロパティ」の「全般」をクリックしてください。クリックしたら、右側の「文字セット」を「Unicode 文字セットを使用する」から「マルチ バイト文字セットを使用する」に変更してください。この設定は、プログラム内部で使用する時の文字セットを指定するものです。設定としては、Unicode でもいいのですが、いろいろと面倒があるのでこのような方法でやっていきます。



次に、ソースコードを書くためのファイルを作成します。次の順にクリックしてください。

「ソリューションエクスプローラー」の「ソース ファイル」を右クリック → 「追加」 → 「新しい項目」

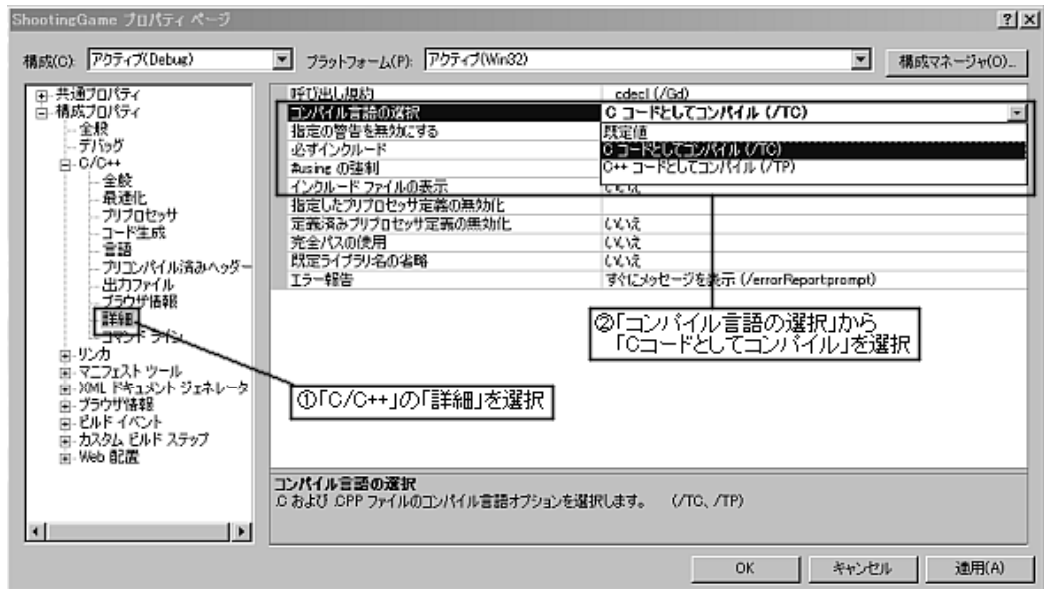


出てきたウィンドウでは、「C++ファイル」を選択して、ファイル名には「main.cpp」と入力します。

それが終わったら、再度、プロパティを開きます。次の順にクリックしてください。

「ソリューションエクスプローラー」の「ShootingGame」を右クリック → 「プロパティ」

開いたら左側の「構成プロパティ」から「C/C++」を選択します。次に、「詳細」を選択して「コンパイル言語の選択」から「Cコードとしてコンパイル」を選択します。



3. Window の表示

早速シューティングゲームの開発に取りかかっていきましょう。まず、ウィンドウの作成を行います。それを早速作ろうと思うのですが、ウィンドウを作成する部分というのは、はじめての人には何をやっているのかわかりにくいため、難しいです。おそらくこの講座の中で一番難しい場所になると思います。ただ、ウィンドウの作成について、今の段階で完全に理解する必要はありません。この講座では必要な部分だけ解説することにして、他の部分は参考になる資料等を示すだけにしたいと思います。

main.cpp に、次のように入力します。

(行番号は、入れなくてください)

```
1. #include <windows.h>
2.
3. #define CLASS_NAME "Sample"
4. #define TITLE_NAME "Shooting Game"
5.
6. HINSTANCE ghInst;
7.
8. LRESULT CALLBACK WindowProc(HWND hWnd,UINT msg,WPARAM wp,LPARAM lp);
```

```

9.
10.  int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrevInst,LPSTR lpsCmdLine,int
nCmdShow)
11.  {
12.      MSG msg;
13.      BOOL bRet;
14.      HWND hWnd;
15.      WNDCLASSEX WindowClass;
16.
17.      ghInst = hInst;
18.
19.      WindowClass.cbSize = sizeof(WNDCLASSEX);
20.      WindowClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
21.      WindowClass.lpfnWndProc = WindowProc;
22.      WindowClass.cbClsExtra = 0;
23.      WindowClass.cbWndExtra = 0;
24.      WindowClass.hInstance = ghInst;
25.      WindowClass.hIcon = (HICON) LoadImage
26.      (
27.          NULL,MAKEINTRESOURCE(IDI_APPLICATION),
28.          IMAGE_ICON,0,0,
29.          LR_DEFAULTSIZE | LR_SHARED
30.      );
31.      WindowClass.hCursor = (HCURSOR) LoadImage
32.      (
33.          NULL,MAKEINTRESOURCE(IDC_ARROW),
34.          IMAGE_CURSOR,0,0,
35.          LR_DEFAULTSIZE | LR_SHARED
36.      );
37.      WindowClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
38.      WindowClass.lpszMenuName = NULL;
39.      WindowClass.lpszClassName = CLASS_NAME;
40.      WindowClass.hIconSm = (HICON) LoadImage
41.      (
42.          NULL,MAKEINTRESOURCE(IDI_APPLICATION),
43.          IMAGE_ICON,0,0,
44.          LR_DEFAULTSIZE | LR_SHARED
45.      );
46.
47.      RegisterClassEx(&WindowClass);
48.
49.      hWnd = CreateWindow
50.      (
51.          CLASS_NAME,TITLE_NAME,
52.          WS_OVERLAPPED|WS_SYSMENU,
53.          CW_USEDEFAULT,CW_USEDEFAULT,
54.          640,480,
55.          NULL,NULL,ghInst,NULL
56.      );
57.
58.      ShowWindow(hWnd,SW_SHOWNORMAL);
59.
60.      while ((bRet = GetMessage(&msg, NULL, 0, 0)) != 0)

```

```

61.         {
62.             if (bRet == -1)
63.             {
64.                 break;
65.             } else {
66.                 TranslateMessage (&msg);
67.                 DispatchMessage (&msg);
68.             }
69.         }
70.
71.     return 0;
72. }
73.
74. LRESULT CALLBACK WindowProc (HWND hWnd,UINT msg,WPARAM wp,LPARAM lp)
75. {
76.     switch (msg)
77.     {
78.     case WM_CLOSE:
79.         DestroyWindow (hWnd);
80.         break;
81.     case WM_DESTROY:
82.         PostQuitMessage (0);
83.         break;
84.     }
85.
86.     return DefWindowProc (hWnd,msg,wp,lp);
87. }

```

では、順番に解説していきます。

(21 行目)

ウィンドウプロシージャの設定を行います。ウィンドウプロシージャというのは何かというと、Windows 用のプログラムでは、ウィンドウごとに一定の操作が行われたときなどにメッセージが送られてきます。そして、プログラムはそのメッセージに従って、どのような処理を行うか決めます。そのときに、メッセージが送られてくる関数がウィンドウプロシージャです。また、そのメッセージのことをウィンドウメッセージといいます。

どの関数をウィンドウプロシージャにするのか決めるのがここでの動作です。ウィンドウプロシージャとして指定できる関数は、戻り値の型と引数の個数、引数の型が決まっています。ここでは、先にプロトタイプ宣言をしている **WindowProc** 関数を指定しています。

(39 行目)

ウィンドウクラスの名前を指定します。ウィンドウクラスというのは、ウィンドウを作成するためのテンプレートのようなものです。そのテンプレートの名前をここで指定します。

(47 行目)

WNDCLASSEX 構造体に格納されているウィンドウクラスの情報を用いて **RegisterClassEx** 関数を使って、Windows に登録します。

(49 行目 - 56 行目)

CreateWindow 関数は、ウィンドウの情報を引数にして、ウィンドウの作成を実際に行います。この関数については、ウィンドウの外見に関わってくるので詳しく説明をします。まず、**CreateWindow** 関数の引数について、MSDN^{*1} で調べてみましょう。すると、次のように出てきます。

```

HWND CreateWindow(
    LPCTSTR lpClassName,      // 登録されているクラス名
    LPCTSTR lpWindowName,     // ウィンドウ名
    DWORD dwStyle,            // ウィンドウスタイル
    int x,                     // ウィンドウの横方向の位置
    int y,                     // ウィンドウの縦方向の位置
    int nWidth,                // ウィンドウの幅
    int nHeight,               // ウィンドウの高さ
    HWND hWndParent,          // 親ウィンドウまたはオーナーウィンドウのハンドル
    HMENU hMenu,               // メニューハンドルまたは子ウィンドウ ID
    HINSTANCE hInstance,      // アプリケーションインスタンスのハンドル
    LPVOID lpParam             // ウィンドウ作成データ
);

```

(引用元)

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpwinui/html/_win32_createwindow.asp

各引数の詳細は、MSDN の方を見てください。ここでは、必要な部分だけ説明します。

「登録されているクラス名」では、Windows から提供されているウィンドウクラスの名前か、同じプログラムの中で登録したウィンドウクラスを指定します。ここの設定を変えることで、エディットコントロールなども作成できます。ここでは、さきほど **RegisterClassEx** 関数を使って登録したウィンドウクラスの名前を入れています。

「ウィンドウ名」のところでは、ウィンドウのタイトルを設定します。このプログラムの場合、マクロで定義されている **TITLE_NAME** (4 行目に定義) を引数に渡しています。ですから、このプログラムでのウィンドウの表示は「ShootingGame」となります。

「ウィンドウの横 (縦) 方向の位置」では、画面上に表示される位置を指定します。このときの座標系は、スクリーン座標系を用いて表記します。

Windows では、大まかに分けて 2 つの座標系があります。実際に見ている画面の左上の部分が原点となる、スクリーン座標系と、各ウィンドウの左上の部分が原点となる、クライアント座標系が存在します。どちらの場合も、X 軸の正方向は原点から右方向、Y 軸の正方向は原点から下方向、となります。

今回は、ウィンドウの位置として **CW_USEDEFAULT** を使用しています。これは、Windows に自動的に場所を決めてもらう、ということです。

「ウィンドウの幅 (高さ)」では、ウィンドウの大きさを指定します。

*1 MSDN

<http://msdn2.microsoft.com/ja-jp/library/>

ここでは、Win32API についての情報から、Windows の動作仕様に関する文章など、Windows 用のアプリケーションを作成するにあたって、重要な情報が掲載されています。Win32API を用いるときには、関数の引数を調べたり、関数の作用を調べたりするときに多用します。

CreateWindow 関数の戻り値として取得できる **HWND** は、ウィンドウハンドルといって、ウィンドウを管理するためのハンドルです。このハンドルは、起動している **Windows** 上では一意のものであり、このハンドルを使用して、複数あるウィンドウの中からどのウィンドウを操作するか、などを決定します。ウィンドウハンドルについては、ウィンドウプロシージャの引数の中に入っていて、ウィンドウプロシージャが **Windows** から呼び出されるときに自動的に渡されます。ですから、グローバル変数として保持する必要はないです。

(58 行目)

ShowWindow 関数では、作成したウィンドウの表示を行います。作成したばかりの状態では、ウィンドウが表示されていない状態となっているので、表示させるようにしましょう。

(72 行目 - 85 行目)

ウィンドウプロシージャです。ここで **Windows** からウィンドウメッセージを受け取ります。ウィンドウメッセージとしては、ユーザーがキーボードから入力を行ったときに **WM_CHAR**、ウィンドウが作成されるときに **WM_CREATE** などが送られてきます。現在記述されているのは、右上の×ボタンが押されたときに、**WM_CLOSE** が送られてくるので、そのときにウィンドウの削除を行って終了するコードです。

大まかな説明でしたが、いかがでしょうか。おそらく、何のことかわからなかった、という人もいかもしれません。はじめにも言ったとおり、今はまだこの場所については完全に理解しなくてもいいです。もし、詳しく知りたいということなのであれば、いくつか参考になる書籍を紹介したいと思います。参考してみてください。

Visual C++ 〈1〉 はじめての Windows プログラミング

山本 信雄 (著) ISBN-10: 4881358219

猫でもわかる Windows プログラミング 第2版

桑井 康孝 (著) ISBN-10: 4797328487

4. プレイヤーの機体の表示

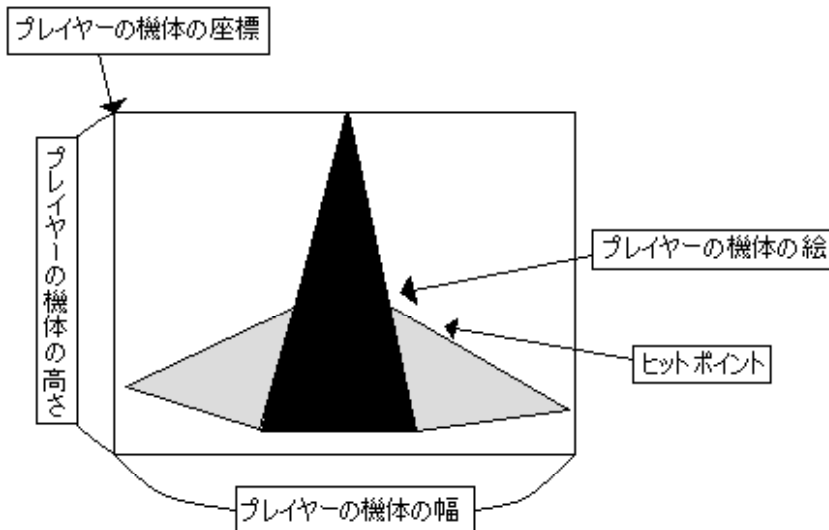
1. Player 構造体の定義

前章では、ウィンドウの作成を行いました。この章では、プレイヤーの機体の表示、ということで、プレイヤーの機体を画面上に表示することをやってみましょう。

まず、このシューティングゲームのプログラミングにおいて、大枠の考え方として描画を行う部分（以後、描画部）と、操作を行う部分（以後、操作部）を分離させます。それらの間を変数を使ってお互いに情報を渡します。

もう少し、詳しく考えてみましょう。描画部では、変数を使って、どの座標に描画するか、どのような情報を描画するか、などを決めて描画を行います。操作部では、ユーザーからのキー操作によってどの座標に移動するか、などの操作を行います。このときの移動先の座標は、変数に代入されます。こうすることで、ゲームをしている側からは、キー操作によって、プレイヤーが動いているように見えます。また、プログラミングを行う側からは、どこで何を行っているのかわかりやすいです。

ここからは、プレイヤーの機体の表示をする作業に入っていきます。先ほどの説明で、描画部と操作部の間を変数を使って情報を渡す、というように書きました。そこで、一番最初にプレイヤー全体の変数の構成について解説します。プレイヤーにはどのような情報が必要になってくるかという、次のような情報が必要になってきます。わかりやすいように、絵にして解説します。



これらの情報は、わかりやすいように構造体に入れます。そうすることによって、プレイヤーという一つの固まりのものをまとめます。構造体の宣言は次のようになります。

```
// (P.10、 5 行目の後に追加)
typedef struct Player
{
    POINT Position;
    int Width;
    int Height;
    int HitPoint;
    HDC hImage;
    WindowInfo mWindowInfo;
} Player;
```

この中で、わかりづらいのは、POINT、HDC、WindowInfo という変数の型だと思います。POINT というのは、Win32API の中でよく用いられる、XY 平面の座標を保持するための構造体です。HDC も Win32API でよく用いられ、こちらは、デバイスコンテキストハンドルという描画対象のデバイスのハンドルを保持するための変数です。詳しくは、画像の表示を行うときに説明します。WindowInfo というのは、新しく定義した構造体です。次のような宣言になっています。

```
// (P.10、 5 行目の Player 構造体の宣言の前に追加)
typedef struct WindowInfo
{
    int Height;
    int Width;
} WindowInfo;
```

この構造体の変数を使って、表示しているウィンドウの幅や高さを保持しています。後々、頻繁に使うことになるので構造体のメンバーに入れてあります。

以上で **Player** 構造体の宣言は終わりです。実際に一から自分で作ってみるときなどは、最初から全部思いつかなくてもかまいません。最初からしっかりと設計を行って、どのような変数が必要になるのか固めてしまうのも手かもしれませんが、思いついたときに構造体にメンバーを追加していくのも手だと思います。

2. プレイヤーの機体の表示

では、**Player** 構造体の定義が終わったところで、プレイヤーの機体の表示をしていきます。

Windows 用のアプリケーションのプログラミングにおいて、描画をするときというのは、ウィンドウプロシージャに **WM_PAINT** メッセージが流れてきたときです。このときに、**Windows** から描画するように命令を受けるわけです。そのメッセージを取得した後、デバイスコンテキストハンドルを取得してそれを使って画面に描画を行います。ちなみに、この部分が今まで説明してきた描画部にあたります。

ウィンドウコンテキストハンドルを取得するまでの流れをコードとともに追っていきましょう。

```
// (P.12、 75 行目の後に追加)
PAINTSTRUCT ps;
HDC hdc;
switch (msg)
{
// (P.12、 77 行目の後に追加)
case WM_PAINT:
    hdc = BeginPaint (hWnd,&ps);
    EndPaint (hWnd,&ps);

    break;
```

まず、ウィンドウメッセージの取得は、ウィンドウプロシージャの引数の **msg** 変数を使います。この変数の内容を **switch-case** 文を使って場合分けして、ウィンドウメッセージの振り分けを行います。

ウィンドウメッセージとして **WM_PAINT** メッセージが来たら、デバイスコンテキストハンドルの取得を行います。デバイスコンテキストハンドルの取得には **BeginPaint** 関数を使います。引数には、ウィンドウハンドルと **PAINTSTRUCT** 構造体のアドレスを指定してください。ウィンドウハンドルはウィンドウプロシージャの引数の中にあります。**PAINTSTRUCT** 構造体は、ウィンドウプロシージャの先頭部分で宣言してあります。描画の情報がここに格納されます。このプログラムの中では特に使うことはないのですが、あまり気にする必要はありません。

ません。ここでは、描画は行わないので、すぐにデバイスコンテキストハンドルを解放しています。解放を行うには、`EndPaint` 関数を使います。こちらも、先ほどと同じ引数になります。case 文の最後では、`break` を行うのをわすれないでください。

次に、プレイヤーの機体の表示をします。本来は画像ファイルからプレイヤーの機体の絵を読み込んでそれを描画するのですが、ここでは簡単にテキストの「■」をプレイヤーと見立てて、それを描画するようにします。

では、コードを見てください。

```
HDC hdc;
// (P.12、 75 行目の後に追加)
RECT rect;
static Player mPlayer;
static WindowInfo mWindowInfo;
switch (msg)
{
// (P.12、 77 行目の後ろに追加)
case WM_CREATE:
    GetClientRect (hWnd,&rect);
    mWindowInfo.Width = rect.right;
    mWindowInfo.Height = rect.bottom;

    mPlayer.mWindowInfo = mWindowInfo;
    mPlayer.Position.x = 200;
    mPlayer.Position.y = 400;
    mPlayer.Width = 16;
    mPlayer.Height = 16;
    mPlayer.HitPoint = 10;

    break;
case WM_PAINT:
    hdc = BeginPaint (hWnd,&ps);
    // (P.12、 80 行目付近の BeginPaint 関数の後に追加)
    TextOut (hdc,mPlayer.Position.x,mPlayer.Position.y,"■",2);
    EndPaint (hWnd,&ps);

    break;
```

上の方から解説をしていきます。まず、`WM_CREATE` メッセージというのが出てくると思えます。これは、ウィンドウが作成されたときに一度だけ送られてくるウィンドウメッセージです。このときに、初期化を行う必要がある変数などは初期化を行います。ここでは、`WindowInfo` 構造体の変数に設定を行うのと、`Player` 構造体の変数の設定を行います。

`WindowInfo` 構造体の変数は、ウィンドウプロシージャの先頭部分で `static` として宣言されています。この変数にウィンドウの情報を代入します。代入するときにウィンドウの情報を調べるには、`GetClientRect` 関数を使います。この関数では、ウィンドウの中の描画ができる範囲の高さと幅を取得できます。引数には、ウィンドウハンドルと `RECT` 構造体の変数のアドレスを渡しています。`RECT` 構造体というのは、`Rectangle` (長方形) の略で、長方形の四辺の位置を保持するために使います。

ここでは、GetClientRect 関数で取得した情報を元に、rect 変数の right 要素を幅、rect 変数の bottom 要素を高さとしています。

次に、Player 構造体の変数の設定を行っています。今回は、プレイヤーの機体の幅や高さの情報は適当に設定してやってください。

最後に、描画についての説明です。紙面上だとわかりづらいかもしれませんが、追加してもらったコードは BeginPaint 関数と EndPaint 関数の間にあります。この位置でないと、デバイスコンテキストハンドルが取得できていないので、描画ができません。

テキストの描画は TextOut 関数を使います。TextOut 関数は次のようになっています。

```
BOOL TextOut(  
    HDC hdc,           // デバイスコンテキストのハンドル  
    int nXStart,       // 開始位置（基準点）の x 座標  
    int nYStart,       // 開始位置（基準点）の y 座標  
    LPCTSTR lpString,  // 文字列  
    int cbString       // 文字数  
);
```

(引用元)

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpgdi/html/_win32_textout.asp

デバイスコンテキストハンドルは、BeginPaint 関数で取得したものを使ってください。

文字列はダブルクォーテーションで囲むようにしてください。

最後に文字数ですが、2 バイト文字を使うときには、日本語として読むときは 1 文字でも 2 バイトと計算してあります。

ちょっと話とはそれますが、もし、表示する文字が変数の中に入っていて、文字数が一定でないときどうすればいいのでしょうか。そういうときは、strlen 関数を使って、変数内の文字数を計算するようにします。(使うときには、string.h をインクルードしてください) この関数では、文字数をカウントしてくれて、2 バイト文字を含む場合、2 文字として計算されます。

さて、終わったらコンパイルをして、実行をしてみましょう。コンパイルは、「F5」キーを押すことでできます。

ウィンドウの左下のあたりに黒い■が表示されたでしょうか。表示されたら成功です。まだ、キーによる操作はできません。それらについては、次回からやっていきましょう。

screen のススメ

文 編集部 kidmin

はじめに

情報科学類に入学した皆さんは、Mac OSX を触る機会が多いかと思います。自宅にインターネット接続環境があれば、計算機室の Mac にログインしていろいろ作業ができるというのは大変便利なことです。また、手引きを見ると分かるとおり、情報科学類のアカウントで 5 台の Linux サーバを利用することもできます。

この項では、コンソールでの作業、特にリモートログイン先で作業を行うときに大変便利なソフトウェアである screen を紹介します。

起動してみよう

とりあえず起動してみましょう。何事も起動してみることから始まります。終了はいくらでも方法がありますから。~~その昔 Emacs を起動して、終了する方法が分からずに仕方なく別のコンソールから `killall emacs` するという苦い記憶が~~

脱線しました。

SSH でログインして、プロンプトに **screen** と入力してみましょう。screen の著作権情報が画面に表示されます。

```

orchid-nwc.coins.tsukuba.ac.jp - PuTTY
Screen version 4.00.02 (FAU) 5-Dec-03

Copyright (c) 1993-2002 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2, or (at your option) any later version.

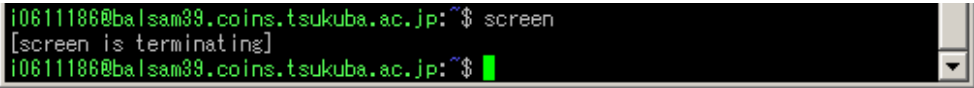
This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to
screen@uni-erlangen.de

[Press Space or Return to end.]
  
```

ここでスペースキーかリターンを押してください。画面がクリアされて、一番上の行にいつものプロンプトが表示されているかと思います。何も変わっていませんね。それでは、**exit** と入力してみましょう。[screen is terminating]と表示され、さっきのシェルに戻っていると思います。

A terminal window showing the execution of the 'screen' command. The prompt is '10611186@balsam39.coins.tsukuba.ac.jp:~\$'. The user enters 'screen', and the output is '[screen is terminating]'. The prompt returns to '10611186@balsam39.coins.tsukuba.ac.jp:~\$' with a green cursor.

```
10611186@balsam39.coins.tsukuba.ac.jp:~$ screen
[screen is terminating]
10611186@balsam39.coins.tsukuba.ac.jp:~$
```

…で、何が美味しいの？

これだけだと、何が何だかさっぱり分かりません。では、screen とは何でしょうか。

screen は、複数の仮想端末 (screen ではウィンドウと呼ぶ) を管理するソフトウェアです。先ほど起動したとき、screen は仮想端末を一つ開いてシェルを起動しました。例えとして少々間違っているかもしれませんが、もう一本別の SSH 接続を行ったようなものです。screen は、コンソールでの作業に対して、いろいろと便利な機能を提供しています。

複数の端末の操作

それでは、もう一度 screen を起動してください

適当に **ls** などを実行してみましょう。そして、**[Ctrl]+a c** (まず Ctrl キーを押しながら A を押して、離してから C を押す) と入力してみましょう。画面がクリアされて、一番上の行にいつものプロンプトが表示されているかと思います。これで、新しいウィンドウが開かれます。今、screen の上では、起動時に開いたウィンドウと今開いたウィンドウの二枚が開かれています。それでは、ウィンドウ間を移動してみましょう。**[Ctrl]+a [Space]** (まず Ctrl キーを押しながら A を押して、離してからスペースキーを押す) と入力してみましょう。先ほどの、ls を実行したウィンドウに戻ります。もう一度 **[Ctrl]+a [Space]** と押すと、先ほど新しく開いたウィンドウに戻ります。

このように、screen ではウィンドウを複数枚作って作業を行うことができます。エディタでプログラムを書きながら、それをコンパイルしてデバッグするなど、複数の端末を同時に操作したいときに大変便利です。

開いたウィンドウは、ウィンドウ内のシェルを終了させると閉じます。また、**[Ctrl]+a k** でウィンドウを強制的に閉じることできます。

レジューム機能

情報科学類の皆さんは、SSH を通して端末室のコンピュータを操作する機会も多いかと思います。Tsunami や WMN のような、無線 LAN などの不安定なネットワーク上で SSH 接続して作業を行っていると、肝心なところで接続が切れてイライラすることもあるかと思います。また、複数の端末を開いて作業をしていて、その内容を残したままログアウトして、再びログインしたときにそのまま再開できると便利だと思いませんか？

screen には、レジューム機能が付いています。今、screen を起動して複数のウィンドウを開いて操作しているとします。この状態のまま、screen 上のコンソールで実行中のプログラムに影響を与えることなくログアウトして、次にログインした後すぐに作業を続行することができるのです。

screen は、起動時にセッションを生成し、その中でウィンドウが作られます。セッションは /tmp/uscreens 以下の名前付きパイプとして保存されます。セッションを切り離すことを「デタッチ」、セッションを呼び出すことを「アタッチ」と呼びます。

それでは、実際にやってみましょう。screen を起動した状態で **[Ctrl]+a d** と入力します。すると、先ほどと同じく [screen is terminating] と表示され、最初のシェルに戻ります。ここで、プロンプトに **screen -ls** と入力してみましょう。すると、セッションの一覧が表示されると思います。

```
i0611186@balsam39.coins.tsukuba.ac.jp:~$ screen
[screen is terminating]
i0611186@balsam39.coins.tsukuba.ac.jp:~$ screen -ls
There is a screen on:
      5011.ttyp0.balsam39      (Detached)
1 Socket in /tmp/usccreens/S-i0611186.

i0611186@balsam39.coins.tsukuba.ac.jp:~$
```

ここで、プロンプトに **screen -r** と入力してみましょう。screen が起動し、デタッチする前の状態がそのまま再現されていると思います。

[Ctrl]+a d の代わりに、[Ctrl]+a DD (D は大文字) と入力すると、デタッチと同時にログアウトすることができます。

リモートログイン中に screen を使っていて、回線が切れてしまった場合、screen はそれを検出して自動的にセッションをデタッチします。再度ログインして **screen -r** でレジュームできます。

セッションがアタッチされたままであるか、セッションが複数存在していると、**screen -r** ではレジュームできません。**screen -d -r** でセッションを強制的にデタッチして再度アタッチすることができます。また、セッションが複数存在しているときは、**screen -r [プロセス番号]** でセッションを指定してアタッチが行えます。プロセス番号は、セッション一覧で表示されている

```
5011.ttyp0.balsam39      (Detached)
```

の 5011 の部分です。

screen -d -RR で、「セッションに再アタッチする。セッションが複数存在していたときは、一番上のセッションをデタッチして再アタッチする。セッションが存在していなければ、新しいセッションを生成する」を一度に行うことができるので、これを.bashrc などのログインスクリプトで実行させている人もいます。この場合、セッションが残ったままになることがあるので、たまに **screen -ls** で確認すると良いでしょう。

コピー&ペースト

SSH で接続して作業を行っているとき、例えば w3m など Web ブラウズして見つけたコマンドを実行したいときなど、コピーペーストをしたい場面があると思います。しかし、Windows や Mac など、操作方法が接続元の環境によって異なります。これって面倒ですね。

screen は、コピー&ペーストの機能を提供しています。まず、[Ctrl]+a [Esc] と入力して、コピーモードに入ります。画面左下に、[Copy mode...] と表示が出ていると思います。

```
i0611186%(screen0)@balsam39.coins.tsukuba.ac.jp:~/local/linux/bin$ ls
ex@  rvim@  screen-4.0.2*  view@  vimdiff@  w3m*  xxd*
rvim@ screen@ vi@      vim*  vimtutor* w3mman*
Copy mode - Column 68 Line 24(+100) (80,24) jp:~/local/linux/bin$
```

それでは、カーソルをコピーしたい始点まで動かしてみましょう。vi と同様の操作方法です。カーソルキーまたは hjkl でカーソルを移動でき、[Ctrl]+b で 1 ページ戻る、[Ctrl]+f で 1 ページ進みます。また、/や?を使った検索も行えます。[Ctrl]+s で Emacs 風のインクリメンタル検索も行えます (上向き検索は [Ctrl]+r)。エスケープキーでコピーモードを中断できます。

カーソルキーを始点に移動したら、エンターキーまたはスペースキーを押してください。左下に [First mark set...] と表示されます。

screen のススメ

```
10611186%(screen0)@balsam39.coins.tsukuba.ac.jp:~/local/linux/bin$ ls
ex@    rvim@    screen-4.0.2*  view@    vimdiff@    w3m*    xxd*
rview@ screen@    vi@          vim*    vimtutor*  w3mman*
First mark set - Column 1 Line 22 sukuba.ac.jp:~/local/linux/bin$
```

次に、同じ要領で終点までカーソルを移動してください。このとき、選択範囲が反転されて表示されています。終点に移動したら、エンターキーまたはスペースキーを押してください。左下に[Copied xx characters into buffer]と表示され、コピー完了です。

```
10611186%(screen0)@balsam39.coins.tsukuba.ac.jp:~/local/linux/bin$ ls
ex@    rvim@    screen-4.0.2*  view@    vimdiff@    w3m*    xxd*
rview@ screen@    vi@          vim*    vimtutor*  w3mman*
Copied 121 characters into buffer sukuba.ac.jp:~/local/linux/bin$
```

それでは、コピーした内容を貼り付けてみましょう。貼り付けるには、[Ctrl]+a]と入力します。

カスタマイズ

screen はとても柔軟に設計されていて、画面表示やキーバインドなどを自由にカスタマイズすることができます。設定は、ホームディレクトリの.screenrc というファイルに書き込みます。

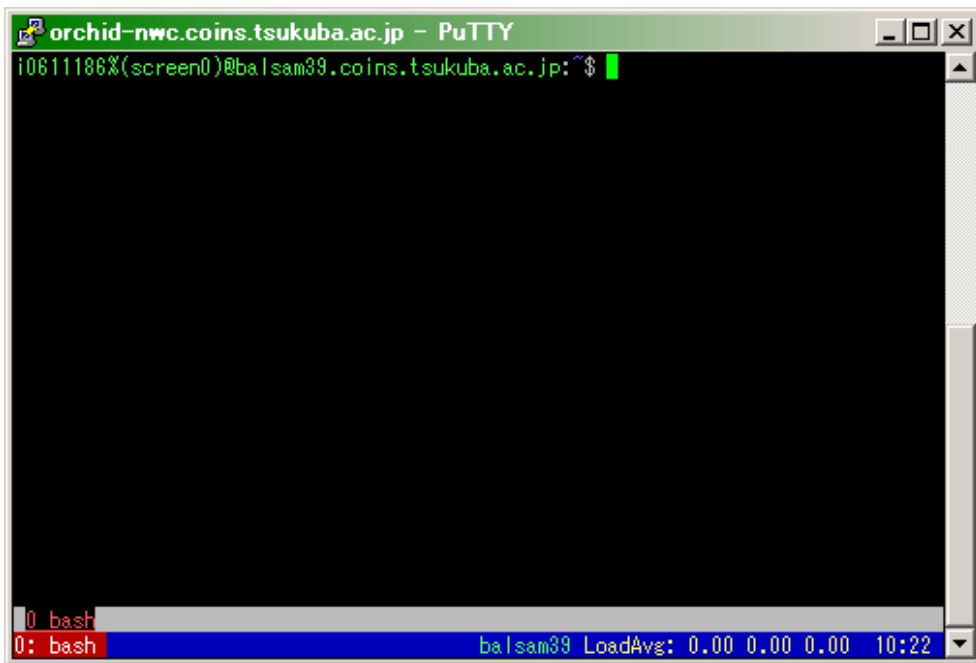
参考までに、筆者の.screenrc を公開します。

```
autodetach on
defbce on
defencoding eucJP
vbell off
defscrollback 10000
startup_message off
hardstatus on
hardstatus alwayslastline
hardstatus string "%[.bW]%[.rW]%n: %t %[-] %=[..G]%H %[..Y]LoadAvg: %l %c "
caption always "%{= wk} %-w%{=bu dr}%n %t%[-]%+w"
```

日本語エンコーディングが UTF-8 の場合、defecoding行を書かずに、screen の起動時に-U オプションを付けてください。

screen を起動中は、環境変数 WINDOW にウィンドウ番号が保存されるので、それを使ってプロンプトにウィンドウ番号を表示できます。プロンプトは環境変数 PS1 で変更できます。詳細は各自で調べてください。

この設定で screen を起動すると、下の図のようになります。



画面下のバーの1段目がウィンドウリスト、2段目が各種ステータスバーです。

参考情報

筆者の.screenrc は

/home2/ugrad/06/i0611186/.screenrc

で手に入れることができます。また、Linux 端末である orchid-calc[1-5].coins.tsukuba.ac.jp には、screen は標準ではインストールされていません。自分でコンパイルするか、または筆者がコンパイルしたバイナリを利用しても構いません。

/home2/ugrad/06/i0611186/local/linux/bin/screen

に、Linux 用のバイナリが置いてあります。

screen のさらに詳しい使い方に関しては、manpage を参照してください。キーバインドを変更する方法なども掲載されています。

(参考)

Let's use SCREEN! - <http://www.dekaino.net/screen/>

vim! vim! vim!

文 編集部 Goth

1.vim!vim!vim!

ハンドル変えました。TakeYou 改め Goth です。今更ながら tait-you 氏とかぶってることに気づいたので(遅

早速ですが、皆様はエディタは何を使っていますでしょうか？

メモ帳？秀丸？さくらエディタ？xyzyy？

Windows ユーザの方だとこのあたりでしょうか？

プログラミングは VisualStudio だって方もいるでしょう。

あるいは Emacs？gedit？Kedit？KDE なら Kate もいいですね。

vim なんてのもありますよね？ありますよね？ですよね？

はい、すいません。ちょいと調子に乗ってました。

何はともあれ、この記事はんな vim の紹介です

超個人的エディタ vim。

そんな vim の魅力の一部分だけでも伝えられたらと考えています。

ちょいとマニアックで、コアな記事ですが、どうぞ最後までお付き合いください。

2.vim のインストールと起動方法

なんかそのまんま入門記事ですね…。

で、早速インストール方法ですが、UNIX 系の OS(含む Mac)を使っていらっしゃる方は既にインストールされてます。

「いつの間に!？」って感じですが、勝手ながら筆者が昨日忍び込んでインストールさせていただきました。…嘘です。

前述の OS 達はだいたいの場合、OS のインストールの際に vim が一緒にインストールされます。

「だいたいの場合」以外の場合でも、vim のオリジナルの **vi**(vim は"Vi IMproved(改良)"の意)が入ってます。

と、いうのも POSIX^{*1} と呼ばれる IEEE^{*2} の定めた規格に vi が入っており、POSIX に従った OS では確実に入っているのです。

ものはためしとテキトーなコンソール(iTerm とか)を開いて"**vi**"と入力してみてください。

ちゃんと起動すると思います。

*1 POSIX : Portable Operating System Interface

*2 IEEE : The Institute of Electrical and Electronics Engineers, Inc.

vim が。

ええ、そうなんです。今日び vi なんて使ってる OS なんて少ないのです。

「POSIX に書いてあるんだろ!？」とか思うかもしれませんが、vim には compatible というオプションがあり、vim の設計方針によると

Vim は Vi の気軽な置き換えとして使うことができるべきである。

ユーザが望むなら、Vim を、オリジナルの Vi との区別がほとんど付かない互換モードで使うことができる。

と、ドえらい勢いで compatible を目指しているのです。

つまり、vi は vim に含まれているわけで、vi をいれるぐらいなら vim を入れる。ということなんでしょうね。

ちなみに、vi しか入ってない UNIX の場合、適宜パッケージ管理ソフトでいれてみてください。

ディストリビューションごとに違いはありますが、たぶん以下のどれかを試せばおっけーです。(管理ユーザで行う必要があったりします。

```
apt-get install vim
yum install vim
urpmi vim
```

あとちゃんと GUI^{*1} なパッケージ管理ソフト (YaST とか) もあることでしょうし、vim は間違いなく見付かります。

vim-huge, vim-enhanced などというパッケージを見つけたら迷わずインストールしておいてください。

学類の Mac には 6.2 が入っているようです。(筆者は 7.0 ぶちこみましたが。

できることならばバージョンは 7.0 が望ましいのですが、6.0 ぐらいでもおっけーです。

```
vim --version
```

でバージョンが確かめられます。

一番上のバージョン情報以外は無視しておっけーです。

さて、Windows の皆様。やっとなさ Windows の説明です。

どこかのサイトでも見て、Cygwin のインストールをしていただいていいですか？

…だめですか。そうですか。

と、ということで、Windows で vim を使う方法のひとつが **Cygwin** を入れる方法です。

Cygwin は、一言でいってしまえば「Windows で UNIX 系のモノを動かすモノ^{*2}」です。

Cygwin も前述の POSIX 同等の機能とツールを提供しており、vim のインストールも可能です。

*1 GUI : Graphical User Interface ⇔ CUI : Character User Interface

*2Windows で... : 厳密に書くと色々あるだろうので、こんな書き方してます。

インストール方法を簡単に説明すると、

1. Cygwin Project のサイト (<http://cygwin.com/>) を開く
2. 右上にある "Install Cygwin now" のリンクを開く
3. ダウンロードが(おそらく)始まる

もしかしたら IE の情報バーとかでるかもしれないので許可してダウンロード

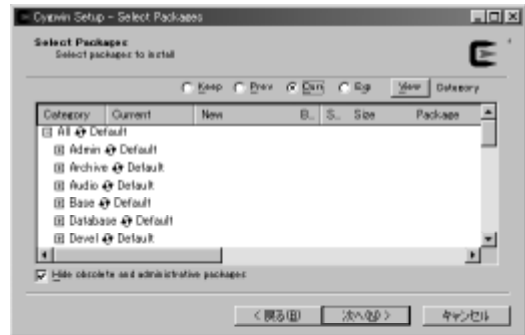


4. ダウンロードした setup.exe を実行(この setup.exe は後で新たにツールをいれたいときにも再利用できる)

インストール途中ネットに繋ぐ必要があるのでネットは繋いだまま

5. 利用許諾書などを軽くスッ飛ばしてよく読んで、パッケージ選択まで進める(→基本的にそのままの状態で「次へ」連打とりあえず、サーバ選択は ".jp" を選ぶ

6. パッケージ選択は基本的にはそのままでもいいが、容量が気になるなら必要なものだけのこして skip でもよい(vim は最初からインストールになっている)



7. インストールが始まる

だいたいこれでおしまいです。

vim 以外にも便利な UNIX 系コマンドはたくさんあるのでいれておくと Good!! だと思います。

Linux 病の筆者は最近 Windows で dir^{*1} を打った記憶がなかったりします…。

起動方法はデスクトップに作られる cygwin のアイコンをダブルクリック。

Cygwin 端末が開いたら "vim" と打って Enter!

Cygwin(というか UNIX 系コマンド)は便利なので入れておいて損はない!!

と、筆者は思うのですが、そう思いませんか？

…思わない？そうですか。

と、ということで vim のみ入れる方法です。

vim のオフィシャルのページ(<http://www.vim.org/>)から Windows 用を持ってこれます。

ですが、KaoriYa.net(<http://www.kaoriya.net/>)でいろいろ日本語まわりの必要な・便利なものを入れて調整されたものが配布されてますので、特別な理由でもない限り KaoriYa をいれるといいでしょう。

KaoriYa の方は自己解凍データですのでダブルクリックだけで展開されそのまま起動できます。

GUI、つまり「メモ帳」など他のエディタと同じように、マウスで操作できる gvim も同梱されています。

*1 dir : Directory コマンド。 Windows のファイルリストを表示するコマンド。Linux なら ls(List)。

さて、インストールと起動の方法は以上ですが、よろしいでしょうか？

インストールはできたでしょうか？

起動できたでしょうか？

ちなみに終了方法の説明がまだなのですが、終了できましたか？

できない？そうですか。では、次章へ。

3. 終了方法 及び モードの説明

と、いうことで、終了方法です。

普通、ソフトウェアの説明で「終了方法」など、わざわざ説明するようなことでもないのですが、vim の場合やや(かなり?)特徴的なので説明しておきます。

#最終手段として端末自体を終了させる方法があります。

#端末 (iTerm とかコマンドプロンプト、Cygwin 端末など) のウィンドウの閉じるボタンを押せば無理矢理終了させることができます。

#ただ、保存はできないし、スワップファイル云々のエラーがでることがあるので多用はしないでください。

#スワップファイル云々のエラーはとりあえず"**r**"を押しておけばおっけーだと思います。

さて、コンピュータリテラシ等でやったとは思いますが、大概の端末で動くアプリケーション(CUI)の場合、Ctrl-C (Ctrl キーを押しながら c のこと。以下同様)を押すと終了することができます。

しかし、vim は Ctrl-C を押しても終了しません。

ためにやってみましょう。

まず、vim を起動します。(既に開いてしまってる方はウィンドウごと終了させて起動しなおしてください。)

そして Ctrl-C を押してみましょう。

vim は終了しないはずです。

終了しないものの、画面下の方に「Vim を終了するには :quit<Enter> と入力してください^{*1}」と出ると思います。

"<Enter>"は Enter キーを押すこと表します。

ですので、":quit"と入力して Enter キーを押せばいいのです。

終了できましたか？

「なんだ、説明なんか知らないじゃないか」とか思うかも知れませんが、思うほど簡単ではないのです。

もう一度 vim を起動して、次の文を打ってみてください。

```
I have a pen.
```

*1 Vim を終了するには... : 英語版だと "Type :quit<Enter> to exit Vim"

中一レベルの英文です^{*1}。たぶんタイプミスするような文でも無いと思います。
にもかかわらず、たぶん"I"を一回入力し直したのではないのでしょうか？
このあたりの説明はともかくとして、終了してみてください。
終了方法は":quit<Enter>"でしたよね？すると画面に

```
I have a pen.:quit
```

と打ち込まれる思います。

あれれ…。

よくよく考えれば、テキストエディタで":quit<Enter>"と打ったら":quit"が書かれて改行されるはずですよね？

じゃあ？どうすれば終了するの？

というか、さっきはなんで":quit"が入力されなかったの？

などと疑問が残ると思います。

これが vim の vim たる所以(ちなみに vim の一番忌まれるところでもあるのですが…)

「モード^{*2}」です。

画面の左下を見てください。

「-- 挿入 --」^{*3}と表示されていると思います。

これが**挿入**モードと呼ばれるモードです。

挿入、つまり文字入力が可能モードです。そのまんまですね。

このモードでは入力された文字は全て画面に書かれていきます。

つまり普通のエディタのような状態です。

試しに"I write a lime."と打つてみると、ちゃんと入力されます。

「いつの間に挿入モードになったんだ？」という質問の答えは「"I have a pen."の"I"を打った時」です。

"I"は"**Insert**"つまり挿入の略で、文字どおり挿入モードに移行するためのコマンドだったのです。

モードの変更に"**I**"が使われてしまったので、"**I**"が欠落して"**I**"を打ちなおすはめになったのです。

この辺の「コマンド云々」はあとで説明しますので、今は「ふへん。で、なにそれ？」程度の認識でおっけーです。

*1 中一レベル：にしても日常会話では絶対に使わない文章である。

*2 モード：厳密にはモードもある種のコマンドと見なすことも出来ます。ですが、この記事では便宜上「モード」と呼びます。

*3 挿入：英語版だと"Insert"。

一方、起動したばかりの状態^{*1}、つまり":quit<Enter>"を押しても入力されず、終了できたモード。

これは**通常モード**と呼ばれるモードで、このモードでは入力された文字によって様々なエディタのコマンドを使うことができます。

さきほどの説明に"**I**"は「挿入モードに移行するためのコマンド」と書きましたが、コマンドは**通常モード**で打つだけで実行されます。

通常モードでは行削除や行内検索、果てはマクロ保存なども行うこともできます。

このモードが vim の一番の武器であり、使いこなせば他のどんなエディタも凌駕しうる機能であったりしますが、その一方で一番の壁でもあります。

他のモードからこのモードへ戻るには Esc キーを押します。

基本的に vim では**通常モード**をベースに作業するので、なにはなくとも、とりあえず Esc を忘れないでください。

他にもモードはありますが、後で紹介します。

さて、通常モードに戻るには Esc を押せばよい、ということで Esc を押してみましょう。

「-- 挿入 --」は消えましたか？

":quit<Enter>"で終了できましたか？

…終了できませんか？

はい、それが正常な動作です。

毎回毎回おんなじパターンでスイマセン。

vim は使えば使うほど、加速度的に経験値があがっていきますので、トライアル&エラーが大事なのです。

トカイイワケシテミルデスヨ。

毎度のごとく左下にでてきたメッセージを読んで見ましょう

「E37: 最後の変更が保存されていません (! を追加で変更を破棄)^{*2}」

「E162: バッファ "[No Name]" の変更は保存されていません^{*3}」

「Press ENTER or type command to continue」

上の二つはエラーメッセージです。

"E37"とか"E162"はエラーコードなわけですが、隣にちゃんとエラー内容が書かれているので気にしなくてもおっけーです。

この手のメッセージは他のエディタでも時々見ますね？

他のエディタでは普通、こういうメッセージには「保存」とか「キャンセル」とかボタン

*1 起動したばかりの状態：設定によっては挿入モードで起動することもできる。

*2 最後の変更が...：英語版だと"No write since last change (add ! to override)"です

*3 バッファ"[No Name]"...：英語版だと"No write since last change for buffer "[No Name]"です

が一緒にくっついてきたり、"(y/n)"みたいなのがでるわけですが…。

ここではそれらの代わりに"(! を追加で変更を破棄)"と書かれています。

"I have a pen."なんて中学生レベルの文なんて保存したくもないでしょうから、破棄しちゃいましょう。

メッセージによれば"!""¹を追加すればよいそうです。

「どこに？」って感じですが、":quit"に"!""を追加して":quit!"にすればよいのです。

まあ、なんとなく、「終了しろ!!」みたいな雰囲気が出てるので覚えやすいと思います。

あ、あと":quit"は":q"だけでもおっけーです。

同様に":quit!"も":q!"と省略できます。

終了方法に関してはこんな感じでしょうか。

保存方法については後で説明するとして、とりあえずまとめです。

終了方法!!

1. まず **Esc** を連打して (別に連打の必要はないが) 確実に通常モードにする
2. 書きかけの文章を保存することなく、破棄して終了するには":quit!" (保存方法は後で)
3. うっかりミスを防ぐためにも普段は":quit"で終了しておくが吉 (後で書きますが、一番使うのはおそらく":wq"です)
4. ":quit"は"q"で省略可能

4.保存方法とコマンドラインモード

タイトルみて「また変なモードが出てきたよ」とか思う方もいらっしゃるかもしれませんが、実は既に使っています。

":quit"がそれです。

正確にいうとコレ→":". 印刷のカスレじゃないですよ？

コロンです。コロンといっても進化するとキュウコンになるやつじゃなくてね？ (古いな…

実はこの":."を押すことによって**コマンドラインモード**になります。

コマンドラインモードは通常モードと似ていますが、より複雑な編集をしたいときに使います。

通常モードでは文字キーを打つだけでコマンドが実行されますが (例えば"**I**"は挿入モードへの移行)、**コマンドラインモード**では **Enter** キーを押されるまで実行されません。

つまり「家に帰るま**d** **Enter**」を押すまでが**コマンドラインモード**です」というわけです。

なのでより複雑、あるいは細かい指定ができるわけです。

コマンドラインモードから**通常モード**に戻るには、やはり同様に **Esc** キーです。

モードの説明は慣れないとちょっと難しいと思いますが、慣れてくるとごく自然に使えるようになります。

*1 "!" : 印刷がつぶれてたらしいません。エクスクラメーションマーク (ビックリマーク) です。

今はそういうモードがある、ことだけを覚えておいてください。

ファイルの保存には":write"を使います。
基本的には":quit"の時と変わりません。

実際に作業しながらやってみましょう。

vim を起動したら

```
I wonder how many miles I've fallen by this time?<Esc>*1
```

と打ってみてください。

最初の"**I**"がふたつあるのは誤植ではなくてモードの変更です。

それと最後の"**<Esc>**"は挿入モードを抜けるためです。

画面には"**I wonder how many miles I've fallen by this time?**"と書かれていると思います。

では、保存してみましょう。

いまモードは通常モードに戻っているはずですので、そのまま"**:write<Enter>**"とタイプ!!

あらら…

またエラーが出ちゃいました。

「E32: ファイル名がありません^{*2}」

だそうです。

確かにファイル名がないですね。

ファイル名を指定するには"**:write**"の後に半角スペースを挟んで、ファイル名を入力します。

試しにファイル名を"**hoge**"としますと、"**:write hoge<Enter>**"と押すことによって"**hoge**"というファイル名で保存されます。

どうでしょうか？ちゃんと保存できましたでしょうか？

一番下の行に「"**hoge**" [新] 1L, 51C 書き込み^{*3}」のようなメッセージが出たらおっけーです。

それぞれ、ファイル名、新たにファイルを作ったこと、行数、文字数、を書き込んだことを示します。

もう、1STEP。

ファイルに少し追加をしましょう。

次の文を入力してください。

*1 I wonder... : Alice's Adventures in Wonderland の一説。アリスがウサギの穴を落っこちている最中のアリスの台詞。

*2 ファイル名が... : 英語版だと、"No file name"

*3 "hoge" [新] ... : 英語版だと""hoge" [New] 1L, 51C written"

`oI must be getting somewhere near the centre of the earth.<Esc>*1`

今度は"**I**"を使わずに別の方法で挿入モードへ変更します。

"**o**"です。なんの略かは知りませんが、カーソルのある行の下に新しい行を作って挿入モードに変更するコマンドです。

そんなにポンポン新しいコマンドを乱用すんな！とか叱られそうですが、今はとりあえず見ながら打ってください。慣れれば(定型文なので省略

さて、新しく行を追加したことですし、上書き保存したいところです。

実は"**:write**"はファイル名を指定しないと上書き保存になります。

先ほど"hoge"という名前で保存したので、ファイル名を指定せずに"**:write<Enter>**"を打つと hoge が上書きされます。

できましたでしょうか？

「"hoge" 2L, 109C 書込み^{*2}」とであればおっけーです。

今回は既存のファイルに上書きするので「[新]」の表示はありません。

それと、文字数が 51C から 109C に増えてます。

保存したことですし、いったん終了しましょう。

"**:q[uit]<Enter>**"([~]はその部分が省略できることを示します。)でしたね？

今回の場合はちゃんと保存しましたので、「最後の変更が保存されていません」みたいなエラーは出ずにちゃんと終了できたと思います。

"**quit**"を"**q**"と略せたように"**write**"も"**w**"と省略することができます。

ちなみに、保存して終了、は"**:w[rite] [ファイル名]<Enter>**"の後で"**:q[uit]<Enter>**"となるわけですが、実は"**:wq [ファイル名]<Enter>**"で一発だったりします。

vim 使いが書いた文章には時々この"**:wq**"が書いてあったりしますが、単にモード変更のし忘れか、ここまで書いて少し休んだぞ、というメモみたいなものなので気にしないであげてください。**:wq**

まとめ!!

1. "**:**"はコマンドラインモードへの移行、コマンドラインモードはより複雑な操作ができる!!
2. 保存方法は"**:w[rite] ファイル名<Enter>**"
3. 一回ファイル名を付けて保存したら、上書きは"**:w[rite]**"だけでおっけー
4. 「保存して終了」は"**:wq [ファイル名]**"で！

5.通常モードコマンド

ということで、vim の一番の山場でございます。

*1 I must... : 同じく Alice's Adventures in Wonderland の一説より。

*2 "hoge" 2L, ... : 英語版だと""hoge" 2L, 109C written"

通常モードでコマンドを押すことによって様々な作業をすることができます。

すでに、ふたつのコマンドが

"I" … 挿入モードに入る (正確には行の先頭にカーソルが移動します。

"o" … カーソルのある行の下に行を挿入してから挿入モードに入ります。

の説明で既に登場しています。

で、コマンドの説明になるのですが…

正直なところ全部書こうとすると、大量すぎて今回号だけでは収まりません。

ということなので、よく使うものをメインに紹介していきます。

5.1 移動系コマンド

カーソルを移動させるためのコマンドです。

「…それってカーソルキーじゃなくて？」とか思うかもしれませんが、vim には**かなり**強力なカーソル移動方法があります。

"h", "j", "k", "l" … それぞれカーソルを左,下,上,右に移動させる。

カーソルキーの代わりになります。

qwerty キーボードでいうと右手のホームポジションあたりの一列です。

ホームポジションから手を動かさずに作業できるためかなり高速に編集作業ができますが、一方で qwerty 配置キーボード以外では使い勝手が悪いコマンドだったりします。

また、単純にひとつカーソルを移動させるだけでなく、一気に複数回移動させることもできます。

たとえば、左に 5 個分カーソルを移動させたい場合は**"5h"**、10 行分上に移動したい場合は**"10k"**、のように、移動したい数を打ち込んでから**"h","j","k","l"**を押せばおっけーです。

一般的に vim のコマンドはすべて、最初に数字を打ち込むことにより複数回そのコマンドを実行することができるのです。

つまり、**"5h"**は**"hhhhh"**を、**"10k"**は**"kkkkkkkkkk"**をそれぞれ打ち込んだことと同等になるのです。

ちなみに、カーソルキーも全く同様です。

"w", "W", "b", "B" … 単語移動

単語単位で移動することのできるコマンドです。

"w"は**"Forword"**で、単語単位で前に (文末方向へ) 移動。

"b"は**"Backward"**で、単語単位で後ろに (文頭方向へ) 移動。

下手に文章で説明するとややこしくなるので実例で説明します。

I wonder how many miles I've fallen by this time?

という文章があつて、カーソルが先頭の**"I"**の上にあるとき、

"w"を押す。→**"wonder"**の**"w"**に移動。

さらに"**w**"を押す。→"how"の"**h**"に移動。

さらに"**b**"を押す。→"wonder"の"**w**"までバック。

#先ほどの"**h**", "**j**", "**k**", "**I**"と同様に回数指定ができます。

さらに"**2w**"を押す。→"how"を飛ばして"many"の"**m**"に移動。

さらに"**3b**"を押す。→"how","wonder"を飛ばして"**I**"までバック。

といった感じです。

実際にカーソルをポイポイ動かしていると動作がよくわかると思います。

"**W**", "**B**"もほぼ同様ですが、「単語」の定義が違います。

たとえば、"I've fallen"という部分は

"**w**", "**b**"だと、"**I**", "", "**ve**", "**fallen**"の様に文字種(数字,英字,記号)ごとに移動します。

一方、"**W**", "**B**"の場合は、"**I've**", "**fallen**"の様に半角スペースごとに分けられていきます。

ちょっと、説明がゴタゴタしていますが、実際にやってみれば少しずつなれていくと思います。

vim は慣れれば慣れるほど(以下定型文につき省略

"gg", "G" … 行移動

指定行に移動するコマンド類です。

たぶん"Goto"の略だと思いますが。ちょっと自信無いです。

"**g**"コマンド関係は結構複雑なのですが、とりあえず"**gg**", "**G**"を覚えておけばおっけーです。

"**gg**"は先頭行まで移動します。

"**G**"は最終行まで移動します。

"**gg**", "**G**"コマンドの回数指定は単純に繰り返しではなく、指定行への移動になります。と
いうか、先頭行への移動を何回も繰り返しても意味ありませんね。

"**10gg**"は 10 行目に移動します。するのですが、実は、"**10G**"も全く同じ動作します。

個人的に言えば"**10G**"は最後から 10 行目に移動してくれると便利なんですけどね…。

ちなみにコマンドラインモードのコマンドの

```
:10<Enter>
```

でも"**10gg**"と同じ動作をすることができます。

"0", "^", "\$" … 行頭、行末へ移動

行内移動、というか行頭と、行末へ移動するコマンドです。

"**^**", "**\$**"の由来は「正規表現」から来ている様です。たぶん。

正規表現については長くなるので割愛します。

"**0**", "**^**"は行頭、"**\$**"は行末にそれぞれ移動です。

大体のエディタでの"<Home>", "<End>"と同様です。

vim でも"<Home>", "<End>"で同様のことができますが、端末の設定によっては"<Home>",

"<End>"が無効になってる場合があるようです。

あと、調べて初めて気がついたのですが"**^**"と"**0**"は微妙に動作が違います。

先頭にホワイトスペース(スペースやタブなど)が入っている場合、

"**^**"だとホワイトスペース以外の先頭文字へ移動。

"**0**"だとホワイトスペースを含めて先頭へ移動。("**<Home>**")もこちららしいです。

ほんと、どうでもいい感じの違いですが…。

"**<Home>**", "**<End>**"が使えるなら使う必要がないと思います。

"<Ctrl-f>", "<Ctrl-b>" … ページダウン/アップ

この表記はだいぶ慣れましたでしょうか？

Ctrl キー押しながら"**f**"やら"**b**"です。

"<Ctrl-f>"が"Forwards", "<Ctrl-b>"が"Backwards"です。

ですが、さっき"**w**"が"Forward"なんですよ…。

ヤヤコシイ…(汗)

"<Pagedown>"や"<Pageup>"と同様です。

#実はスクロール系のコマンドはあと数種類あり、それぞれ微妙にスクロール行数が違うのですが、ややこしくなるので割愛します。

"f", "F", "t", "T" … 行内検索

行内で一文字検索を行います。

"**f**", "**F**"は"find"の略で、"**f**"は前方から(文頭から)、"**F**"は後方から(文末から)検索し、移動します。

"**t**", "**T**"は"till"の略で、それぞれ"**f**", "**F**"で移動する位置の「一歩手前」まで移動します。

ちょっとこれまでのよりやや扱いが特別で、使いかたは"**f**<検索したい文字>"です。

ですが、これだけ書かれてもサッパリでしょうなので実例で説明です。

例のごとくいつもの文

I wonder how many miles I've fallen by this time?

この行頭にカーソルがあるとします。

ここで"**fm**"を押すと、先頭から"**m**"を検索します。

すると"many"に"**m**"がありますので、そこまで移動します。

さらに"**fm**"を押すと次の検索結果、つまり"miles"の"**m**"まで移動します。

"**Fo**"は逆方向に検索するので"time"の"**m**"ではなく、行頭方向の"many"の"**m**"へ移動します。

#"**h**", "**j**", "**k**", "**l**"同様回数指定ができます。

さらに"**2fm**"を押すと、"miles"を飛ばして"times"の"**m**"まで行末の方へ移動します。

いったんカーソルを先頭に戻して、"**t**"の例を示します。

"**to**"とした場合、"**o**"を検索し、"wonder"の"**o**"の手前、つまり"**w**"まで移動します。

さらに"**to**"を押すと…あれ、移動しません…。ああ、そうか。

カーソル位置から検索するので、すぐ次の"wonder"の"o"を見つけてしまうらしいです。

"2to"なら"wonder"の"o"を飛ばして、次の"o"つまり、"how"の"o"を見つけてくれます。なので"how"の"h"まで移動します。

さらに"**To**"を押せば、今度は逆順検索なので"wonder"の"o"をみつけ、"wonder"の"n"まで移動します。

"w"ではなく、"n"に移動することに気を付けてください。

「手前」は今いるカーソル位置から見て、なのです。

と、ちょっと複雑になっちゃいましたが、使えば慣れますので(以下定型文なので省略
こんなコマンド使うのか?とか思われそうですが、単品で使うことはあまりないですね。
他のコマンド組み合わせて使うと便利なのですが、あまり初心者向けではないかも、です。
よく使う組み合わせとしては"**ct**"や"**df**"あたりでしょうか。後で説明します。

移動系コマンドのまとめ

移動するだけになんでこんな複雑な…。とお思いでしょうが、実際にはほとんど定型文的な使いかたしかないなので、そんなにしっかり覚える必要はありません。

移動系コマンドの真価は、次の編集系コマンドと組み合わせることにあります。

編集系コマンドとの組み合わせはある程度しかないので、組み合わせて覚えちゃっていいでしょう。

5.2 編集系コマンド

編集系…ってなんじゃい！って感じですが、実は筆者も全然考えないでつけてます。この章のタイトル。

具体的には、「挿入」「削除」「訂正」などを紹介していきます。

"a", "A", "i", "I", "o", "O" … 挿入モードへ移行する

挿入モードへ移行しますが、それぞれどこに「挿入」するかが変わります。

"a"は"Append"で、カーソル位置の後ろに追加。

"i"は"Insert"で、カーソル位置の前に挿入。

"o"は…なんの略だろう?ととりあえず、カーソル位置の下に空行を作ってそこに挿入。

"A", "I"は"a", "i"の進化版(?), "O"は"o"の逆になっていて、

"A"は行末に、"I"は行頭に、"O"はカーソル位置の上に空行を作ってそこに、それぞれ挿入します。

"o"コマンドは地味に便利で、プログラミングしていると多々お世話になります。

ここでいう挿入というのは、挿入モードへの以降なので、文字入力後には"**<Esc>**"が必要となります。

多分 vim を使う上で一番使うコマンドでしょうから、しっかり覚えましょう。

ちなみに筆者の個人的な経験で言えば"**O**"は動作がやや遅いです。ときどきコマンドであるはずの"**O**"が画面に出ちゃったりもしますが、すぐに消えて挿入モードになります。多分。

"x", "X", "d", "D" … 削除

文字の削除を行います。

"x"は…多分ペケってイメージで削除、なのかな？

"d"はそのまんま"delete"です。

説明が簡単な"x", "X"から。

"x", "X"はそれぞれ他のエディタで言う<Delete>, <BackSpace>と同じです。

…以上。

えと、手抜きと違います。多分これが一番わかりやすい説明かと。

つまり、それぞれ、カーソルの位置の文字、カーソルの一個前の文字、を削除します。

もちろん、回数指定できるので、**"100x"**とか打てば 100 文字削除することができます。

さて、**"d"**コマンドですが、コレは今までにない概念が出てきます。

ちょっと説明しづらいので、毎度のごとく具体例でいきますか。

The twelve jurors were all writing very busily on slates.

"I wonder …"は、いい加減飽きたので別の一節です。

法廷のシーンなわけですが、正直この"jurors(陪審員)"たちは途方もないおまぬけさん^{*1}たちで、人数いてもしかたがないので人数を消してしましましょう。

"twelve"を消すわけですが、手順としては次のようになります。

1. 消したい単語の先頭、"twelve"まで移動する。("**w**"や"**ft**"など一発で移動できます)
 2. "**dtj**"を押す。(すると"twelve"が消えます)
- たった 4 ストロークです。(3 ストロークで十分なのですが…)

"**tj**"ってなんやねん！とか叫び声が聞こえてきそうですが、実はこれ、移動系コマンドの"**t**"コマンドそのものだったりします。

覚えてますでしょうか？"**t**"は"till"の略で、コマンドに続く文字を検索して、その検索位置の手前まで移動するコマンドです。

つまり、今、"twelve"の"t"にカーソルがあるとき、"**tj**"は"jurors"の"j"の手前の単語境のスペースまでカーソル移動することになります。

この、「カーソルが今いる位置」から「コマンドによって移動する位置」を範囲にして、"**d**"コマンドは削除を行います。

*1 おまぬけさん："What are they(=jurors) doing?" Alice whispered to the Gryphon. "They can't have anything to put down yet, before the trial's begun."

"They're putting down their names," the Gryphon whispered in reply, "for fear they should forget them before the end of the trial."

もう一個例文です。

```
The Queen of Hearts, she made some tarts,  
All on a summer day:  
The Knave of Hearts, he stole those tarts,  
And took them quite away!
```

"The Queen of Hearts, "の部分消してしまいましょう。

いくつか方法はありますが、とりあえず"**t**"コマンドを使ってみましょう。

まず、カーソルがどこに移動するかを確認するために"**d**"を付けずに"**t**"を実行してみます。

"she"の手前まで移動できればいい訳ですから単純に考えると"**ts**"でしょうか？

ただ、やってみればわかりますが、"**Hearts**"にも"**s**"が入ってますので、"**Hearts**"の"**t**"までしか移動できません。

そこで、回数指定。"**ts**"の先頭に 2 を付けて"**2ts**"にすればちゃんと"**she**"の手前まで移動できます。

ちゃんと範囲が指定できたので"**d**"コマンドにくっつけましょう。

範囲の始点は「カーソルの今いる位置」なので、"**^**"コマンドや"**0**"コマンドなどでちゃんと先頭に戻っておきましょう。(カーソルキーで戻ってもよいのですが、せっかくなので。)

先頭まで戻ったら"**d2ts**"と打ってみましょう。

これでちゃんと"**Queen of Hearts** "が消えたと思います。

ハートの女王を消したので、ハートのジャックも消しちゃいましょう。

カーソルをふたつ下の行まで下げて ("**2j**"とか)、"**The Knave of Hearts, "**の先頭まで移動しましょう。(ジャックって"**Knave**"だって知ってました？

さて、さっきと同じじゃつまらないので、今度は"**w**"コマンドを使ってみましょうか。

"**w**"は単語移動でしたね。今回は 5 単語消したいので"**5w**"でおっけーです。(", "も一単語と扱います。それが嫌なら"**4W**"でもかまいませんが…。)

と、言うわけで、"**d5w**"で、サクッと消えちゃいます。

…どう考えても"**w**"を使ったほうが楽ですね。一単語消去(最初の例の"**twelve**"とか)なら"**dw**"で1ストローク分タイプする量減りますし。

なんで説明を後回しにしたかというと、実は"**w**"は例外的な動作をするのです…。

と、いうかすでにしています。気がついたでしょうか？

範囲は「カーソルが今いる位置」から「コマンドによって移動する位置」までです。

"**d**"コマンドはその範囲をまるごと削除してしまいます。

で、実は範囲には「コマンドによって移動する位置」も含まれます。

つまり、最初の例で"**djt**"ではなく、"**dfj**"としたとき、"**jurors**"の"**j**"が削除されてしまうのです。(vim コマンドになれて無い方にはもはや暗号ですな…。)

で、"**w**"は単語の先頭に移動するわけですから"**dw**"としたとき、次の単語の先頭も消えないとおかしいわけです。

が、利便性のため、"**dw**"の時は先頭の手前までを削除するのです。

と、まあ、色々細かいことを書いてきましたが(気がつけば削除だけで2 ページ…)、"**d**[消

したい単語数]w"だけ覚えておけばよいでしょう。

と、まとめ的な雰囲気醸し出してるのですが、削除の説明はまだまだ続きます。

今までの説明はすべて、行内の一部分だけの削除でしたが、一行丸ごと、もしくは数行丸ごと消したいこともあると思います。

そんなときも"d"コマンドです。

一行まるごと削除は至って簡単"dd"です。今カーソルのある行をまるごと消します。

複数行を消すときは"dd"に回数指定をつけて"42dd"などと 42 行ほどさっくり消えます。

また、範囲指定で行を削除することも出来ます。

"d2j"なら"2j"によって移動する行、つまり二行下まで削除するので、今いる行を含め、合計 3 行削除することができます。数字+1 についてことに気をつけてください。

回数指定と範囲を合わせて"2d2j"などとやると"d2j"二回分、つまり 6 行削除になります。

…普通に"6dd"とか"d5j"とかやればいいのだけだね。

あとよく使うのが"ggdG"。どうなるかはやってみてのお楽しみ。

ヒントをあげると"gg"は先頭行へ移動、"G"は最終行へ移動だから…。

やったときに生じる一切の責任を筆者は負いかねますのでご了承がいます。

あと最後に残った"D"ですが、これは"d\$"と同じ、つまり今いる位置から行末までを削除します。

筆者はあまり使いませんなあ…。"d<End>"ですむ話ですので…

ということで 2 ページにわたって削除(というか範囲指定)の話でしたが、この辺の挙動は vim の基本になりますのでしっかり押さえておきましょう。

"d"以外にもこの範囲を使うコマンドはたくさんあるので。

"r", "R", "c", "C" … 訂正

間違っって打った文字や単語を訂正します。

"r"が"Replace"で、世に言う"上書きモード"になります。

"c"が"Correct"で、削除と挿入を組み合わせた感じです。

まずは"r"。

一文字だけ置換します。

使用法は"r<上書きする文字>"です。カーソルのある文字を消して、代わりに<上書きしたい文字>に変えます。

一文字だけ、じゃなくて複数文字置換したい時は"R"です。

Esc を押すまで"r"が続く、と考えていただければおっけーです。

おっけーなのですが、なにやら左下に「--置換--」とか出てるとおもいます。

今まで出てきませんでしたが、置換モード、というモードもあるのです。

そして、"R"はその置換モードへ入るコマンド、なわけですが、"R"を"r"の亜種と考えれば、"R"から"<Esc>"までがひとつのコマンドと考えることもできます。

モード、モードと書いてきましたが、コマンドだ！と言うこともできるのです。
その証拠に、"**R**"や、"**i**"など、モード変更コマンドも実は回数指定ができるのです。
例えば、"**100Onew line!<Esc>**"は"**new line!**"と書かれた行を 100 行ほど追加します。

"**r**", "**R**"は共に文字数を変えることができないので、長さが違うと訂正を行うことが出来ません。つまり、"**hoge**"を"**fuga**"に変更できても、"**hoge**"を"**hoge hoge**"に変えることができません。

そのような場合は"**c**"を使います。

先ほどの説明にも書きましたが"**c**"は"**d**"と"**a**"を足したようなモノなので実は必要なかったりするのですが、使いこなせるとエレガントです。

具体例です。カーソル位置は例のごとく先頭にあるとします。

The twelve jurors were all writing very busily on slates.

先ほどは"**twelve**"消しちゃえ〜、ってなことをやっていたわけですが、今度は人数を変えましょう。

トカゲのビル^{*1}を除名処分して、"**twelve**"を"**eleven**"に変えましょうか。
文字数が一緒なので"**R**"を使ってもよいのですが"**c**"を使ってみましょう。
使用法はほとんど"**d**"と同じです。

まず、"**twelve**"に移動します。("**w**"コマンドなど。)
一単語を訂正するので"**cw**"を押します。
すると、"**twelve**"が消えて、挿入モードへ切り替わるので"**eleven**"を入力して"**<Esc>**".
全体で"**wcweleven<Esc>**"となります。

基本的に"**d**"コマンドと一緒にするので、
"**C**"は"**Da**"と同じで、カーソル位置から最後まで消して、挿入モード。
"**cc**"は"**ddO**"と同じで、カーソルのある行をまっさらにして、挿入モード。("**O**"なのは"**dd**"は行をつめてしまうので)
など、そのほかの細かい点もだいたい"**d**"と同じです。
ですので、よく理解できなかった場合は前項の削除からよく読んでみてください。

編集系コマンドのまとめ

編集系コマンドの一番の山は「範囲」です。

範囲指定の方法になれると、何も知らない人から「魔法でも使ってるんじゃないか」とか印象を受けるという噂です。(あくまで伝聞。)

範囲に関しては実際にやってみるのが一番の近道です。

vim は慣れれば慣れるほど(以下定型文なので省略

*1 トカゲのビル：巨大化した Alice に煙突からスッ飛ばされたり、法廷では Alice や The Queen of Hearts によってひどい目にあっています。

とりあえずここまでのまとめ！

えと、ページ数がページ数だけにさすがにこのあたりで止めておきます。
あと「あると便利」系なコマンドを紹介したいのですが…
次号で紹介したいと思います。

とりあえず vim で重要になるのは、「モード」があること、と通常コマンドの文法。
それさえしっかり押さえられれば、あとは慣れです。どこまでも「慣れ」です。
慣れたら多分 vim 中毒になります。
慣れられなかったら…多分 vim を放り投げます。はい。

あと、いまさら、って感じですが、vimtutor ってコマンドがあります。

vim のチュートリアルで、よく使うコマンドから説明されているコマンドがあったりするので、そちらも参考にしてください。

ただ、日本語用の vim だと例文が日本語でなのですが、vim での日本語の扱いはやっかいなのであまりおすすめできません。

一応、一言だけ述べておきますと、

1. "i"などで挿入モードに入る
2. 日本語をオンにする
3. 日本語を入力する
4. 日本語をオフにする
5. Esc でモードを抜ける。

の順番を守ってください。守らないとひどいことになります…。

あと有名な本では「動物本」シリーズで「入門 vi」というのがありますが、興味があれば是非一度読んでみてください。

ちなみに作者は読んでいません(あれ？)

最後に引用の一節にて

プログラム P の考え得る限りの動作空間 B の中で各々の要素を b とした時、プログラム P の全ユーザ空間 U のうち少なくとも 1 人のユーザ u が、ある動作 b を許された動作空間 B' に加えるように要求するまでにかかる時間は有限である。(このとき B' は $B' \leq B$ となる)

言い換えれば: 遅かれ早かれ全員がオブションとして総てを欲しがるとだよ (Negri)

貴方の vim life に光あれ！！

ロリ狐。

文 編集部 dorio

癒しは必要ですか？

最近癒し、足りてますか？

レポートに追われる人、仕様変更を追われてる人、単位取得に追われている人。現代人は色々なものに追われ、疲れ切っています。

そんな現代だからこそ、忙しい時間の合間に少し手を止めて、癒しを求めてみましょう。

獣耳最強宣言。

近頃、動物は癒しとして持てはやされてますね。そんな獣を私たちのディスプレイの中で、常日ごろから愛でていれば、きっと心はさっぱり爽やか。単位も納期もどうしてもよくなるでしょう。

ま、良いか悪いかは別にして。

獣耳娘を飼う(インストールする)for Win

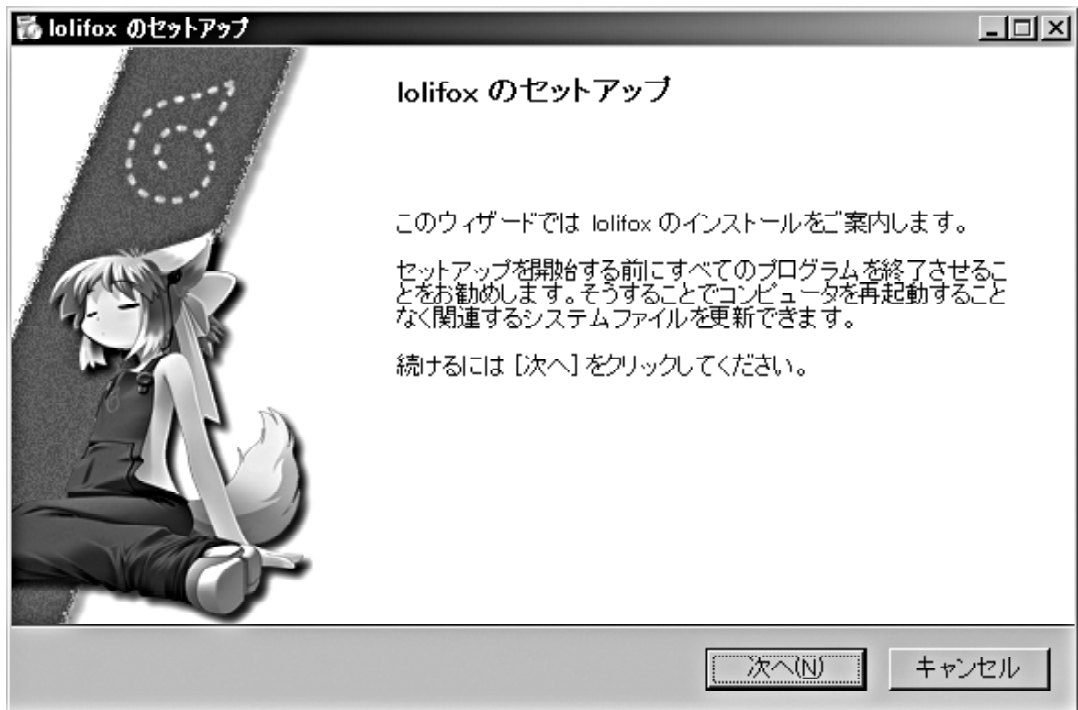
とりあえず「俺はお姉さんの方がいいんだ」とか、「**男の子の方が r y**」とか言う意見は聞かなかった事にして、インストールを始めます。

まず、(<http://lolifox.com/>)にアクセス。いきなりヤリヤリしい娘可愛い娘が現れ、心も洗われます。

一通り和んだ所で、左上の Download をクリック。すると、ダウンロード画面になるので、日本語版を落としましょう。ただ、Mac 版と Linux 版は日本語版が無いです。「Linux でも林檎でも使いたいんだお!」という方は、適当に firefox の lang ファイルをごによごによしてみて下さい。使えると思います。…多分。

で。インストール開始。

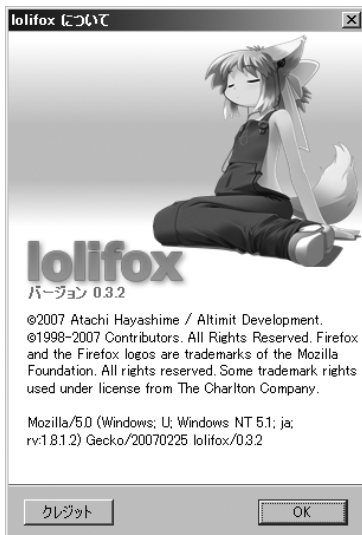
"エグゼ(exe)～ダウンロード～♪" したのをクリックしてやるとこんな感じ。



…たまんねえ。

いや、この見えそうで見えない感じがすばらしいこの純粋な感じと、動物の癒しパワーがたまりませんよね？

で、後は利用規約を英語なんて読めるか！俺は日本人だ！と叫びながらすっ飛ばして熟読して、インストールをしましょう。



そしてインストールを完了したら、起動してみましょう。初回起動時には「既定のブラウザにしますか?」と聞かれるので、迷わずに「OK」を勢いよくクリックしましょう。これで完了です。

いらっしゃいませ、めくるめく世界へ…

この勢いで、ひたすら自分の持っているマシンを全て lolifox に入れ替えてしまいましょう。

あ、デフォルトのお気に入りとかが萌○連だったり、ふ○ばちゃんねるだったりするのは仕様です。きちんと定期巡回して、好みの画像を逃さないようにしましょう。

ちなみに、左画像は「ヘルプ→lolifox について」で表示される画面です。

もう、ゴールしてもいいよね…

この後、獣耳っ娘は編集部がおいしく頂きました。

書籍紹介

文 編集部 アスロンズ

第一回

書籍紹介とは

この書籍紹介は、情報学群生による情報学群生のための書籍紹介です。技術的に役立つものから知っておいて得をするものまで、コンピュータ関連の書籍をピックアップして紹介します。

すっきりわかった！レイヤ3スイッチ

さて、第一回の書籍紹介。このコーナーは、毎号変わる担当者がお勧めの一冊を紹介するコーナーです。記念すべき第一回はこれ、「すっきりわかった！レイヤ3スイッチ」この本は、株式会社アスキーのネットワークマガジン編集部が出版している「すっきりわかった」シリーズのうちの一冊です。このシリーズは中級者向けで、比較的马ニアックな解説がなされています。

主な記載内容は書籍名からも分かるように、レイヤ3スイッチについての基礎から応用までを幅広く解説しています。

主な記載内容のキーワードは以下のとおりです。

「レイヤ3スイッチの基礎、VLANの基礎、ルーティングの基礎、スパニングツリープロトコル、リンクアグリゲーション、ポートミラーリング、IEEE802.1x、QoS、VRRP、RIP、OSPF、SNMP、レイヤ3スイッチの設定、IP電話、PoE」

全体的な構成は、序盤は基礎的な内容で、終盤に向かうにつれて徐々に応用的内容に踏み込んでいくものになっています。ですので比較的読みやすいかと思います。応用的内容を理解するにあたり、あらかじめ「TCP/IP」などのネットワークの基礎を理解しておくといいでしょう。

この本を読んで、一歩進んだのスイッチの世界へ足を踏み込んでみてはいかがでしょうか？

書籍名 : すっきりわかった！レイヤ3スイッチ 初版
著者名 : ネットワークマガジン編集部
発行 : 株式会社アスキー
ISBN : 978-4-7561-4881-0
値段 : 2100 円 + 税
頁数 : 228 頁
発行日 : 07 年 04 月 03 日



情報科学類誌

WORD

Hello Word! 号

発行者

情報科学類長

編集長

柳田 一樹

制作・編集

筑波大学情報学群

情報科学類誌WORD編集部
(3C棟212室)

印刷

3F棟印刷室

2007年5月30日

初版第一刷発行(386部)