2010年度 機械情報夏学期
ソフトウェア第二

担当：岡田 慧 k-okada@jsk.t.u-tokyo.ac.jp,
http://www.jsk.t.u-tokyo.ac.jp/~k-okada/lecture/

2010年4月5日

Linuxプログラミング環境の構築

ソフトウェア第二ではソフトウェアの設計と実装，およびその基礎となる計算機システムについて講義する．ソフトウェアは自分自身の手を動かしてプログラミングすることで，内容を確認し理解を深めることが重要であるため，適宜実例や実習を交ぜながら進めていく予定である．

そのためにソフトウェア第二の初回は各自の計算機に講義に必要なプログラミング環境を構築する方法を説明し，自分自身でプログラミング環境を構築する方法を説明する．

1　仮想マシンを使ったUbuntu環境の構築

Ubuntu とは Linux [1]のディストリビューションの一つである．ディトリビューションとは，カーネルに加えて複数のサービス，ツールに加えインストーラがついており適宜アップデートできるようになっているパッケージであり，Ubuntu は現時点でもっとも使いやすいディストリビューションである．

計算機に Ubuntu をインストールする方法は3つある．

1. ハードディスク1台全ての領域に Ubuntu をインストールする．ハードディスクに存在した情報は全て消去される．
2. ハードディスクに既に存在した OS の領域を変更し，空いた領域に Ubuntu をインストールする．計算機の起動時にはどちらの OS を立ち上げるかを選択する（これをデュアルブートという）．
3. 仮想マシンを使って既に存在する OS の上に Ubuntu をインストールする．

デュアルブートの方法は https://wiki.ubuntulinux.jp/UbuntuTips/Install/InstallDualBoot などに情報が存在する．ただし，失敗すると既に存在する OS の領域が消えてしまうので細心の注意が必要である．

以下ではもっとも簡単な仮想マシンを使う方法を http://www.ubuntulinux.jp/products/JA-Localized/vmware にそって説明する．ここでは Windows がインストールされた PC を持っていると仮定している．

[1] 厳密には Linux はカーネルだけを指し，GNU プロジェクトで開発されたサービスやツールを利用することで OS としての機能を実現している．したがって，GNU/Linux と呼ぶのが正しい，という意見もある．

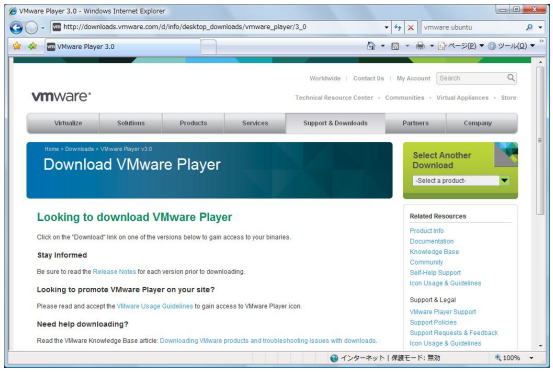
1.1 仮想マシン

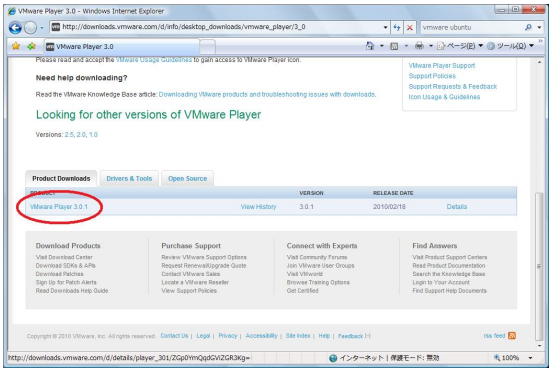
仮想マシンとはある OS 上にハードウェア環境を仮想化して，その仮想ハードウェア上で別 OS を動かすというものであり，これを利用してベースとなる OS 環境 (ホスト OS と呼ぶ) の上に様々な OS 環境 (ゲスト OS と呼ぶ) を構築できるようになる[2]

1.2 VMware Player のダウンロードとインストール

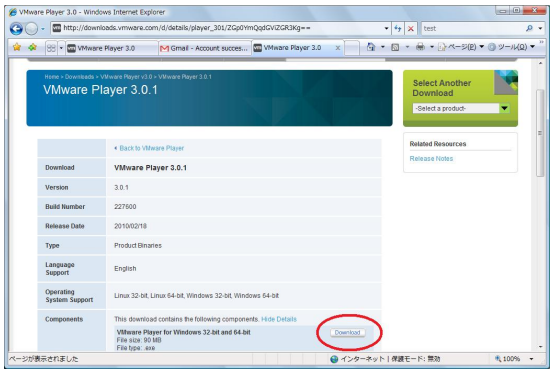
VMware Player は仮想マシン環境を実行させる無償ソフトウェアで `http://www.vmware.com/jp/download/player/` からダウンロードしインストールする．

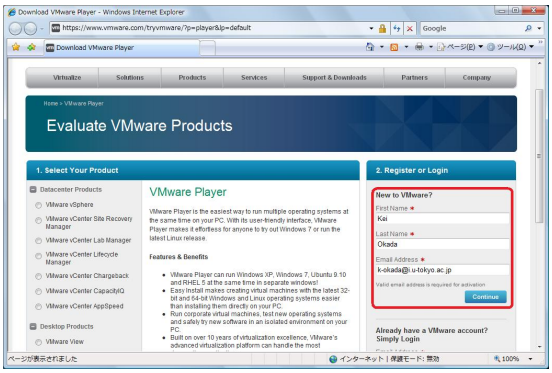
上記 URL にアクセスすると下図左のページが開く．





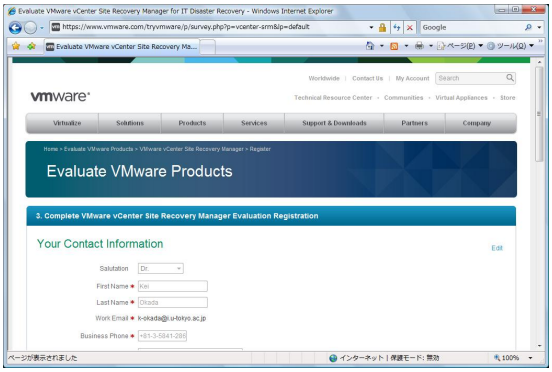
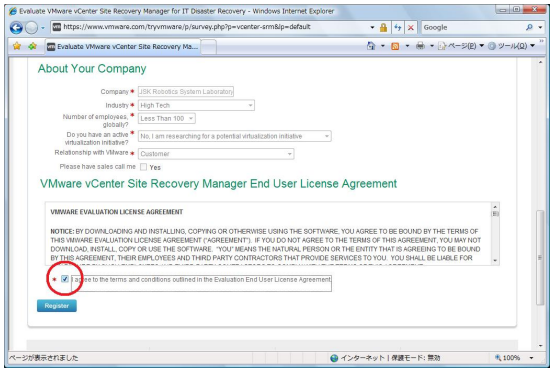
このページを下にスクロールすると右図のように “VMware Plaeyr 3.0.1” というリンクがあるのでこれをクリックする．



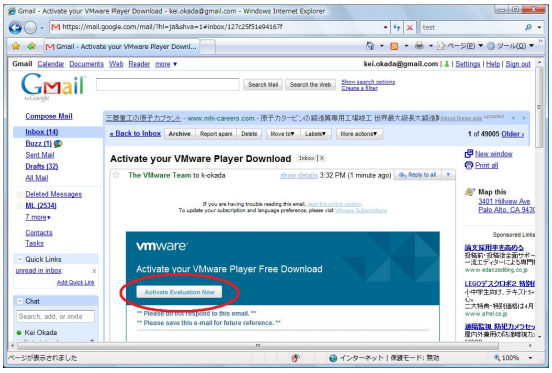
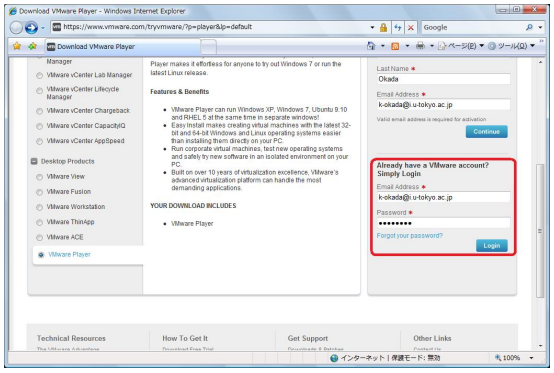


次は上図左の “VMware Player for Windows 32-bit and 64-bit” のダウンロードボタンをおすと “Registor or Login” という入力欄が出てくるので，名前，名字，メールアドレスを入力し “Continue” を押す．

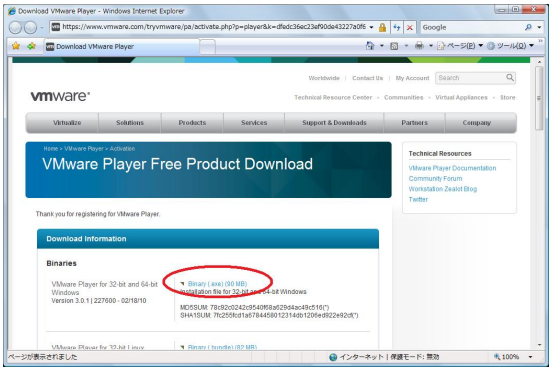
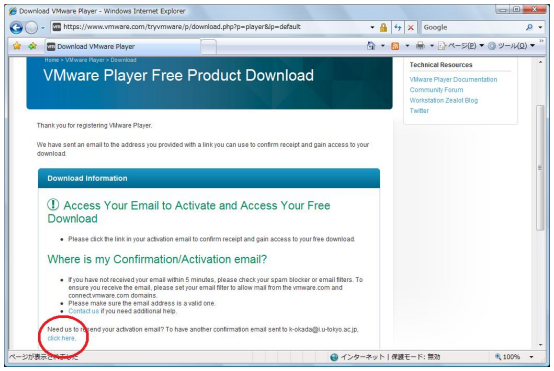
[2] 最近のトピックでは,Windows7 が標準で WindowsXP を仮想マシンとして用意する機能を持ち, その上で WindowsXP 時代のソフトウェアがそのまま動くという話題があります.

そこで上図の様に個人情報を入力する．この際パスワードを入れる欄があるが，ここで入れたパスワードは後で利用するので覚えておくこと．最後にページの一番下の “I agree to the terms and conditions ontlined in Evaluation End User License Agreement” にチェックを入れて “Registor” を押す．

しばらくすると VMWare 社から”Activate your VMware Player Download” というメールが届くので”Activate Evaluation Now“というボタンを押してアカウントを有効化する．これをすると右図の様に VMware 社のホームページにアカウントをつかってログインができる．この際のパスワードは上記の個人情報入力時のものである．

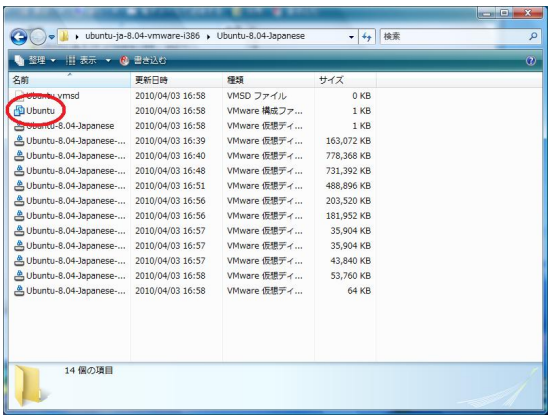



これで図の様にダウンロードすることが出きるようになる．アクティベーションのメールが届かない場合葉左図のような画面から “Need to resend your activation email?” の “clink heare” を押してみよう．ネットワーク環境にも依存するがダウンロードには約 15 分ぐらいかかるだろう．

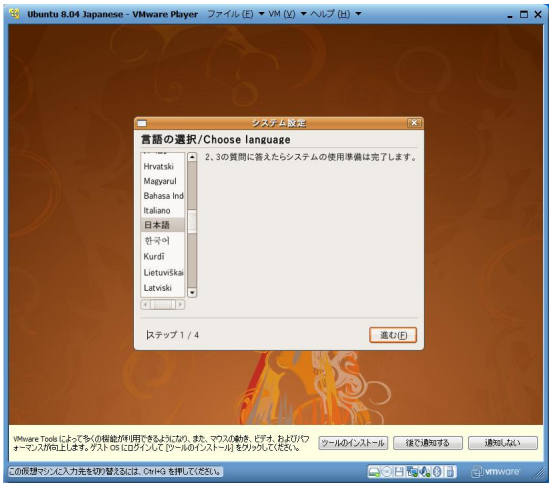
ダウンロードした実行ファイル (VMWare-player-3.0.1-227600.exe) を起動し「次へ (N)」を何回か押せばインストールは完了する．

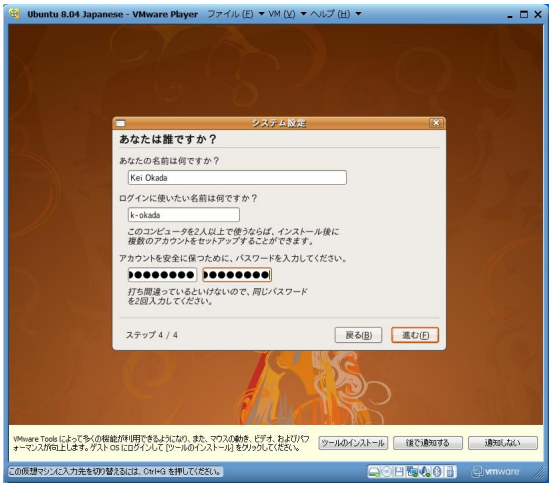
1.3 VMware 用 Ubuntu 仮想マシンのダウンロードと起動

VMWare 用 Ubuntu 仮想マシン (ゲスト OS 環境) を `http://www.ubuntulinux.jp/products/JA-Localized/vmware` からダウンロードする．ダウンロードした `ubuntu-ja-8.04-vmware-i386.zip` を解凍して出来たディレクトリから “Ubuntu” あるいは，“Ubuntu.vmx” となっているファイルをダブルクリックすれば，Ubuntu が仮想マシンの中で起動する．初回起動時は Ubuntu が起動する前に，使用条件，デバイス，VMware Tools Linux 版等のインストールに関して幾つかダイアログがでるがそれぞれデフォルトの選択をしておけば良い．



また，一旦 Ubuntu が立ち上がったあとも，初回起動時には，言語の選択，時間帯の設定，キーボードのレイアウトの設定ダイアログが表示される．それぞれデフォルトを選択すればよい．また，ユーザーアカウントの設定では好きな名前をつければ良い．





1.4 Ubuntu での初期設定

初回起動後に行ったら良い作業を幾つか紹介する

1.4.1 端末のランチャをパネルに追加

最初に「アプリケーション」「アクセサリ」に順にカーソルを合わせ，そこで出てきた「端末」を右クリックして「このランチャをパネルに追加」を選択し，パネルに端末のランチャを置くと使

いやすい.

1.4.2 ソフトウェアのアップデート

また，最初に

```
k-okada@ubuntu-vm:~$ sudo apt-get update
```

として，ソフトウェアをアップデートする．この際パスワードを聞かれるので，上記のユーザーアカウントの設定で入力したパスワードを入力する．sudo はルートアカウント（管理者権限）でコマンドを実行する．管理者権限ではシステムを破壊することが可能なので実行時には細心の注意で望むこと．

また，

```
k-okada@ubuntu-vm:~$ sudo apt-get upgrade
```

とすると，Ubuntu の最新バージョンにアップグレードが可能である．

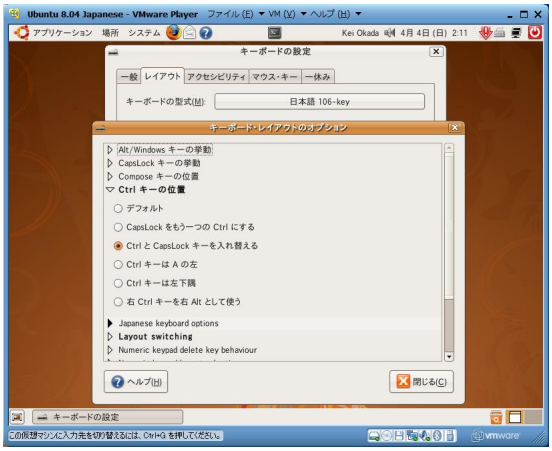
1.4.3 日本語入力ショートカットの変更

デフォルトの日本語入力モードには “Ctrl+Space” で入ることが出きるが，これは emacs のキーバインドと衝突するため，“Ctrl+\” で変更できるようにする．

「システム」「設定」「SCIM 入力メソッドの設定」から「全体設定」「SCIM 開始」の右側のボタンを押し，まずは設定済みのキーをそれぞれ選択し，すべて削除する．次にキーコードの右側のボタンを押し,「キー選択」の画面で “Ctrl+\” を押す.「anthy」「キーバインド」「ON/OFF 切り替え」でも同様に指定する．


1.4.4 キーボードの配列の変更

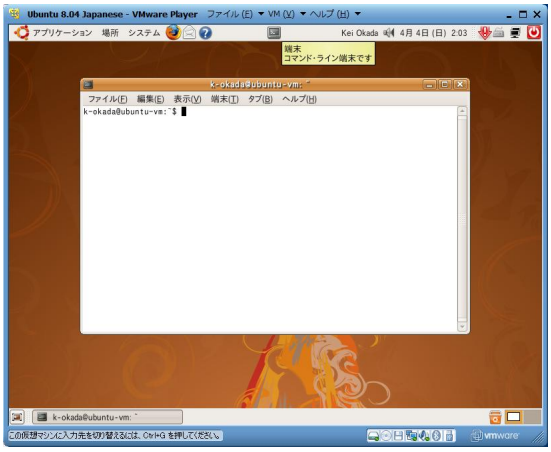
また emacs では Ctrl キーを多用する．キーボードの Ctrl と Caps を入れ替えると便利な事が多い.「システム」「設定」「キーボード」でキーボード設定を起動し,「レイアウト」で「レイアウトのオプション」から「Ctrl キーの位置」を選択し「Ctrl と CapsLock キーを入れ替える」にチェックを入れる．

2 Linux の使い方

2.1 shell の立ち上げ方

パネルに置いたランチャや「アプリケーション」「アクセサリ」「端末」を選択し端末を起動する．これで shell が立ち上がる．




```
k-okada@ubuntu-vm:~$ mkdir work
k-okada@ubuntu-vm:~$ cd work
k-okada@ubuntu-vm:~/work$ mkdir soft2
k-okada@ubuntu-vm:~/work$ ls
soft2
k-okada@ubuntu-vm:~/work$ ls -al
合計 12
drwxr-xr-x  3 k-okada k-okada 4096 2010-04-03 18:02 .
drwxr-xr-x 27 k-okada k-okada 4096 2010-04-03 18:02 ..
drwxr-xr-x  3 k-okada k-okada 4096 2010-04-03 18:02 soft2
k-okada@ubuntu-vm:~/work$ cd soft2
k-okada@ubuntu-vm:~/work/soft2$ ls
```

等の UNIX コマンドを駆使してディレクトリの作成，移動が可能である（覚えているだろうか?）．
最後に，

```
k-okada@ubuntu-vm:~/work$ cd ~
k-okada@ubuntu-vm:~$
```

とするとホームディレクトリ (/home/k-okada) に戻ることができる．

2.2 基本的な Unix コマンド

基本的な Unix コマンドは (現) 京都大学人文科学研究所の安岡孝一准教授による http://kanji.zinbun.kyoto-u.ac.jp/~yasuoka/publications/dareUni/ にある．ドキュメントが参考になる．
またファイルは以下のようにすることでダウンロードし展開できる．

```
k-okada@ubuntu-vm:~$ mkdir doc
k-okada@ubuntu-vm:~$ cd doc
k-okada@ubuntu-vm:~/doc $ wget -nH --cut-dirs=4 -np -r
 http://www.kanji.zinbun.kyoto-u.ac.jp/~yasuoka/publications/dareUni/
```

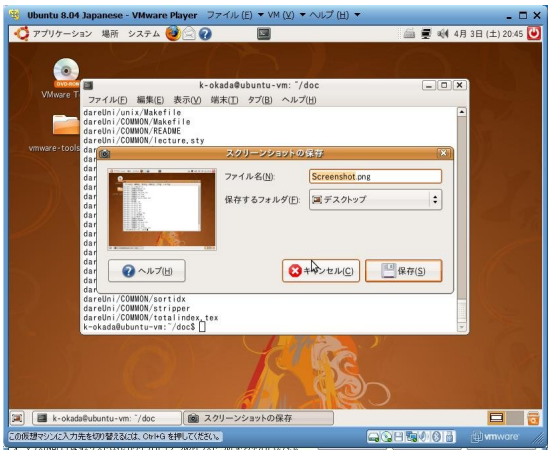
ここでは以下の様に index.html を開いて目次を見ることができる．

```
k-okada@ubuntu-vm:~$ gnome-open index.html
```

まずは「誰にでも使える#/bin/sh 講座」を読んでみると良い．また,「誰にでも使える Unix 講座」にも有用な情報がある．第 1 回に出てくるメールコマンドなどは設定していないためつかえないであろう．第 3 回や第 4 回を読むと良い．

2.3 画面キャプチャの仕方

キーボードの右上にある PrtSc キーを押すと全画面キャプチャが可能になる．さらに，キーボード左下にある Alt キーを押しながら PrtSc キーを押すと一番上にあるウィンドウだけがキャプチャされる．キャプチャした画像は適切な名前をつけて保存しよう．



3 Ubuntu の使い方

Ubuntu では deb パッケージという方式を用いて各種ソフトウェアやツールを管理，配布している．ユーザはこれを用いて簡単にソフトウェアをインストールする事ができる．

ここでは，後で紹介する emacs というエディタをインストールする場合を説明しよう．

3.1 パッケージを探す

emacs というコマンドをうつと以下の様なメッセージがでる．

```
k-okada@ubuntu-vm:~$ emacs -nw
プログラム 'emacs' は以下のパッケージで見つかりました:
 * emacs21-nox
 * emacs22
 * emacs-snapshot
 * e3
 * emacs-snapshot-nox
 * emacs22-gtk
 * emacs21
 * emacs22-nox
 * jove
次の操作を試してください: sudo apt-get install <選択したパッケージ>
bash: emacs: command not found
```

これは emacs というコマンドがないのでインストールする必要があるということを伝えている．イ

ンストールする際には，インストールするパッケージ名を探す必要がる．ここでは上のどれかをインストールすれば emacs が使えるようになる，ということを伝えている．

また，別の方法として，

```
k-okada@ubuntu-vm:~$ apt-get search emacs
```

とすると emacs というキーワードに関係のあるパッケージを探すことができる．

3.2 パッケージをインストールする

パッケージ名が分かれば

```
k-okada@ubuntu-vm:~$ sudo sudo apt-get install emacs22
```

としてインストールする事ができる．

このようにコマンドを打つと，

```
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の特別パッケージがインストールされます:
  emacs22-bin-common emacs22-common emacsen-common libgif4
  liblockfile1 xaw3dg
提案パッケージ:
  emacs22-el
以下のパッケージが新たにインストールされます:
  emacs22 emacs22-bin-common emacs22-common emacsen-common libgif4
  liblockfile1 xaw3dg
アップグレード: 0 個、新規インストール: 7 個、削除: 0 個、保留: 406 個。
20.9MB のアーカイブを取得する必要があります。
この操作後に追加で 65.5MB のディスク容量が消費されます。
続行しますか [Y/n]?
```

と表示される．ここで “Y” と打つか，あるいはリターンを押せばインストールが続行され emacs が利用可能になる．

3.3 情報源

日本語の情報源は https://wiki.ubuntulinux.jp/ にある．また, https://wiki.ubuntulinux.jp/UbuntuTips でいろいろな使い方の Tips が得られる．また，インターネットで調べた際には http://ubuntuforums.org/ にある情報が信用できる．

4 Emacs エディタの使い方

エディタは Emacs を使おう．

```
k-okada@ubuntu-vm:~$ emacs -nw
```

として立ち上がる．

C-z とすると，shell に戻る．C-z とは Ctrl キーを押してから z を押す，という操作を意味している．これは，プロセスをバックグラウンドで実行している状態である．ここで，fg と打ち込めば，また Emacs に戻ることができる．

4.1 .emacs

~/.emacs は，Emacs の起動時読み込まれるファイルである．Emacs から~/.emacs を開いて，どのような設定があるのかを眺めてみよう．Emacs が読み込むファイルは，EmacsLisp という Lisp 言語で記述されている．

Emacs が立ち上がったら C-x C-f と打ち込み (Ctrl キーを押しながら x を押し，次に，Ctrl キーを押しながら f を押す．Ctrl キーは押しっぱなしでもよい)，リターンを押す．ミニバッファ（画面下側の 1 行の部分）に Find file: ~/と表示されるので，ここで，.emacs と打ち込みリターンを押す．これで~/.emacs が表示される．もし，途中で操作が分からなくなったら C-g と入力するとよい．入力がキャンセルされミニバッファが初期状態に戻る．

これができたら，~/.emacs の後ろの方に，

```
;;; M-x g goto-line
;;; デバッグをする際にこれがあると便利である
(global-set-key "\M-g" 'goto-line)
```

という行を追加しよう．そして，Emacs を起動しなおして，~/.emacs を開き，M-g（ESC キーを押して**離して** g キーを押す）と入力し，行番号を指定すると，指定した行へジャンプすることを確認しよう．

ついでに，

```
(global-set-key "\C-h" 'backward-delete-char)
```

という行を書いておくと C-h と入力するとデリートキーを入力するのと同じ効果を得られる．これも大変便利なので，是非設定しておいてほしい．

4.2 Emacs の操作

emacs の操作，特にキーボードショートカットを覚えるのは，多少の時間を要するが，確実に作業効率は向上するので，ぜひ習得してもらいたい．

以下のコマンドを覚えれば基本的な操作は十分であろう．

- カーソル移動
  - C-f カーソルを一文字右に移動
  - C-b カーソルを一文字左に移動
  - C-p カーソルを一行上に移動
  - C-n カーソルを一行下に移動
  - C-a カーソルを行頭に移動
  - C-e カーソルを行末に移動

- 画面の分割
  - C-x 2 画面を上下二分割にする
  - C-x 3 画面を左右二分割にする
  - C-x o 画面分割を移動する
  - C-x 1 画面の分割を元に戻す
- ファイルの読み書き
  - C-x C-s ファイルに現在の内容をセーブ
  - C-x C-f ファイルをオープン
  - C-x C-w ファイルの名前を変更して保存
- Emacs の終了
  - C-x C-c Emacs を終了
- 削除
  - C-d カーソルのある位置にある文字を削除
- 検索
  - C-s カーソルのある位置以降をインクリメンタル検索
- カット&ペースト
  - C-k カーソルのある位置の以降の文字を切り取る
  - C-y C-k できりとった部分をカーソル位置に張り付ける

基本については Emacs Beginner's HOWTO (http://www.linux.or.jp/JF/JFdocs/Emacs-Beginner-HOWTO.html) に書いてある，2.3 キーボードの基本 (http://www.linux.or.jp/JF/JFdocs/Emacs-Beginner-HOWTO-2.html#ss2.3) が参考になる．

よりアドバンスドな内容については GNU Emacs マニュアル (http://www.bookshelf.jp/texi/emacs-20.6-man-jp/emacs.html) のマークとリージョン (http://www.bookshelf.jp/texi/emacs-20.6-man-jp/emacs_10.html) について見てみるとよい．リージョンの選択，コピー，ペースト，ができるようになれば，

4.3 Emacs 上の shell

Emacs 上で bash を立ち上げることができる．

Emacs 上で，`M-x shell` と打ち込む（ESC キーを押して**離して** x を押し，shell と入力する）と，普通の shell の画面と同様のコマンドプロンプトが Emacs に表示されることが分る．あとは通常と同じようにコマンドを打ち込み，実行することができる．このモードを「shell モード」と呼ぶ．shell モードの利点は，プログラムの実行結果やデバッグ文等のメッセージ表示が，エディタ上に残るので，その部分をコピーペーストしてレポートを作成したり，メールで転送したりするなどの作業が容易にできる点である．

◇ 復習プログラム ◇

エディタで test0.c を作成し，シェルで下のコマンドを実行せよ．それぞれの行が何をしているかを考えよ．シェルの上から，下の test0.c(各自アレンジして構わない) から生成されるプログラムの実行を行い，その結果を表示させよ．
test0.s は cat, more, less 等のコマンドで中身を見ることができる．test0.o はバイナリファイルのため，これらのコマンドで中を見てもよくわからない．そういう場合は `hexdump test0.o` などとしてみよう．
UNIX のパイプ機能を活用して `hexdump test0.o | less` 等とするのもよい．

```
/* test0.c */
#include <stdio.h>

int test(int i, int j) {
  return (i * j);
}

int main(int argc, char *argv) {
  int i,j,k;
  i = 3;
  j = 2;
  k = test(i,j);
  if (k > 5) printf(">5\n");
  else printf("<=5\n");
  return 0;
}
```

```
$ ls
test0.c
$ gcc -S test0.c
$ cat test0.s
$ gcc -c test0.c
$ objdump -d test0.o
$ gcc -o test0 test0.o
$ nm test0.o
$ nm test0
$ ls
test0.c test0test0.o test0.s
$ ./test0
```

※復習 1 で実行したコマンドの意味は，以下のとおり．それぞれの用語の意味を復習しよう．

- gcc -S … Cプログラムをアセンブリ言語に変換
- gcc -c … バイナリ（機械語ファイル）に変換
- gcc -o … 出力ファイルの指定
- gcc … 実行ファイル生成（必要に応じてコンパイル・リンク）
- nm … シンボル（関数・変数）のアドレス一覧を出力
- objdump -d … 逆アセンブル（機械語からアセンブリ言語を生成）

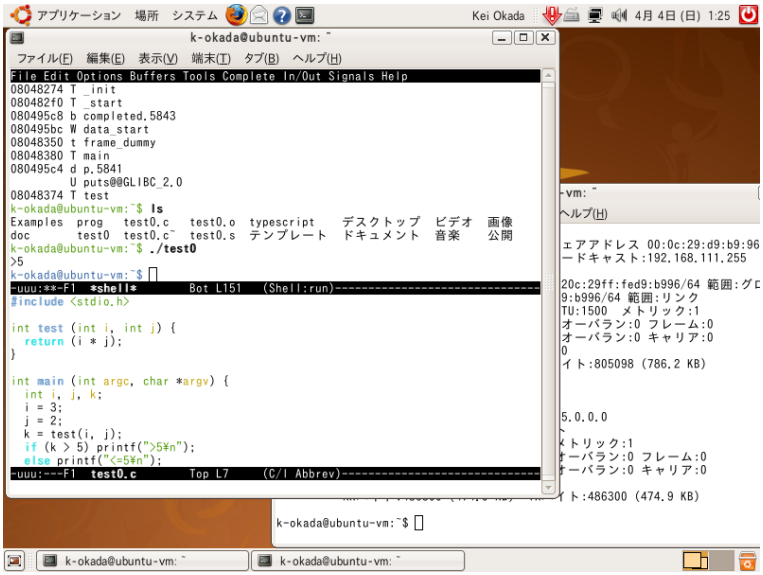
5 宿題

提出先：メールで k-okada-soft2@jsk.t.u-tokyo.ac.jp に送ること
提出内容：以下の問題 1-3 の実行結果の画面をキャプチャし送ること
　　その際メールタイトルは「学籍番号 日時」というフォーマットにすること．
　　また，各自のマシンの Mac アドレスが分かるようにすること
締切り：次回の授業開始時点まで

- 問題１：復習プログラムを emacs で記述し実行せよ．

- 問題２：2 つの引数の和を出力するシェルスクリプト add.sh を作成せよ
- 問題３：“ps aux” とすると，いま計算機で走っている全てのプロセスを表示できる．この１列目が各プロセスのオーナであるが，どのユーザが何個のプロセスを走らせているかが分かるスクリプトを作れ．

問題 1 の解答例を以下に示す．Mac アドレスは ifconfig というコマンドで表示される．



問題 2,3 のシェルスクリプトは「誰にでも使える#/bin/sh 講座」読めばヒントがある．問題 3 では以下のような出力が期待されている．

```
k-okada@k-okada-laptop:~/doc/soft2/2010$ ps-by-user.sh
| uniq -c | sort  -nr
    123 root
     69 k-okada
      5 www-data
      2 daemon
      2 avahi
      2 111
      1 syslog
      1 statd
      1 ntp
      1 gdm
      1 canna
      1 120
      1 108
```

分からないときは

演習中や，自分でプログラミングを試してみるような場合に，色々とわからないことが出てくるだろう．そのような場合，すぐに k-okada-soft2@jsk.t.u-tokyo.ac.jp に聞いてほしい．

また，席が隣の人，友人など，回りの人に聞いてみるのもよい．その際，何に困っているか？どういう現象かを適切に伝えるため，相手の計算機でも同じ問題が生じるような手続きを伝えることが重要である．実はこの様に問題を適切に把握し他人にも伝えることが出きるようになるのが一番難しい．これができれば，分からないことの９０％は解決したようなものだ．分からないことが出てきたら，何が分からないか，どう分からないかを上手に伝えるためにはどうしたらいいか，考えてみよう．

また．今は検索エンジンの性能が素晴らしく向上しているので，http://www.google.com/ 等で検索しよう[3]．ただし，インターネットの情報は常に正しいとは限らないので要注意!!!鵜呑みにしてはいけない．必ず，複数の情報源から総合的に判断すること．

また，Linux でコマンドの役割やオプションを知りたい時は，man を活用しよう．Linux では端末を開いて[4]，man gcc，man make，man ar，man objdump，man man, man ps, man uniq, man sort, man cut 等々としてみよう．これらを読めば宿題は解けるはずだ．

Makefile 復習

今後，演習を進めていくに当たって，Makefile は必須の知識となる．良く分っていない人は今回でマスターしてしまおう (計算機演習でも使われてきた).

Makefile 基本

Makefile のルールの基本は次の形である．

```
ターゲット:依存するファイル
        コマンド
```

コマンドラインから

```
make（ターゲット）
```

とすることで,「ターゲット」が存在しない，あるいはターゲットよりも新しい「依存するファイル」場合にコマンドが実行される．また，コマンド行の始まりは必ずタブコードを入力する．タブコードは Tab と書かれたキーを押すことで入力される．Tab キーを押すとカーソルが数文字分右にずれ，字下げされる．見た目はスペースを複数入力するのと同じだが，実際には１文字文のデータからなり制御文字[5]の一種である．Emacs では C-b や C-f でカーソルを移動させると，タブを入力した部分はスペースではなく，字下げされていることが分かる．

[3]「ググる」と言う．英語でも you can google it 等と動詞化されているようだ．

[4]例えば背景を右クリックして「端末を開く」を選択すると「gnome 端末」が起動する．

[5]制御文字は端末を制御するための特殊文字で，例えばリターンキーを打つとデータとしては一文字の 0x0a というコードが入力されるが，表示のときはそこで改行する約束になっている．Tab キーも同様に 0x09 というコードが入力され，指定された場所まで字下げする

したがって test1 が main.c，hello.c，goodbye.c からコンパイルされる場合は Makefile というファイルに以下のように記述する．

```
test1: main.c hello.c goodbye.c
        gcc -o test1 main.c hello.c goodbye.c
```

これでコマンドラインから make と打ち込むと，make コマンドはデフォルトで Makefile という名前のファイルを参照し，依存関係を判断し以下のようにコンパイルする．

```
% ls
Makefile main.c hello.c goodbye.c
% make test1
gcc -o test1 main.c hello.c goodbye.c
```

◇ **練習問題** 1 ◇

以下の 3 つのファイルを作り，これをコンパイルする Makefile を作ってみよう．コマンドラインから make と打ち込んだら test1 という実行ファイルが作られればよい．

```
/* main.c */
#include <stdio.h>

int main() {
  printf("main.\n");
  hello();
  goodbye();
  return 0;
}
```

```
/* hello.c */
#include <stdio.h>

void hello(void) {
  printf("hello!!\n");
}
```

```
/* goodbye.c */
#include <stdio.h>

void goodbye(void) {
  printf("goodbye!!\n");
}
```

依存関係の記述

上の例では hello.c が更新された場合でも main.c や goodbye.c のコンパイルが行われてしまい，非効率である．これを回避するためには以下のように main.c から main.o を作るルールと，hello.c から hello.o を作るルールと，goodbye.c から goodbye.o を作るルールと Makefile に書き， test2 は hello.o と goodbye.o と main.o から作られるようにする．そうすると，hello.c が更新された場合，`make test2` とすると hello.c のみ再コンパイルされ，不要な main.c や goodbye.c の再コンパイルは行われない．

```
test2: main.o hello.o goodbye.o
        gcc -o test2 main.o hello.o goodbye.o
main.o: main.c
        gcc -o main.o -c main.c
hello.o: hello.c
        gcc -o hello.o -c hello.c
goodbye.o: goodbye.c
        gcc -o goodbye.o -c goodbye.c
```

自動変数

ルールを書くときに同じことを何回も書くことを避けるために，自動変数が用意されている．

```
test3: main.o hello.o goodbye.o
        gcc -o $@ $^
main.o: main.c
        gcc -o $@ -c $^
hello.o: hello.c
        gcc -o $@ -c $^
goodbye.o: goodbye.c
        gcc -o $@ -c $^
```

**$@** ターゲットのファイル名，$^は依存するファイル名であるため，この Makefile は上の例と同じことになる．

$% ターゲットがアーカイブメンバだったときのターゲットメンバ名

$< 最初の依存するファイルの名前

$? ターゲットより新しいすべての依存するファイル名

$ ^ すべての依存するファイルの名前

$+ Makefile と同じ順番の依存するファイルの名前

$* サフィックスを除いたターゲットの名前

さらに，ファイル名.c からはファイル名.o を作るルールは，次のように書ける．

```
%.o: %.c
        gcc -o $@ -c $<
```

こうすると汎用なルールになり，Makefile は以下のように４行でシンプルに書くことができる．

```
test4: main.o hello.o goodbye.o
        gcc -o $@ $^
%.o: %.c
        gcc -o $@ -c $<
```

Makefile は奥が深い[6]ので，google で色々と調べてみると良い．

[6] とはいえ，奥が深いものも考え物である．最初の設計がもっとよければ簡単に使えるツールになるのに，と思われるものは多い．そのようなものを高林哲さんはバッドノウハウ (BK) と読んで論考している (http://0xcc.net/misc/bad-knowhow.html)．ただ BK として引き継がれているものは，そのようなノウハウの塊で使いづらい以上の利便性を提供しているともいえる．

様々なプログラミング言語

世の中には今まで勉強してきた C 言語や Java 意外にも様々な言語が存在し，活用されている．ここでは幾つかのプログラミング言語について，復習プログラムと等価のプログラムを紹介する．

Java

Java は 1995 年ごろサン・マイクロシステムズで開発されたオブジェクト指向言語．以下の実行例で分かるように javac というコンパイラで.class ファイルと呼ばれる中間言語を作成し，これを実行する java と呼ばれるプログラムで実行する．このとき中間言語はプラットフォームに依存しない形で設計されているため，．class を別のマシン (例えば Windows から Linux) へ転送し java で実行しても同じ結果を得ることができる．初期の Java 言語は，ちょうどインターネットブラウザが黎明期にリリースされており，インターネット上でのプログラムのやり取りや，Web 上でのグラフィカルな表現とインタラクティブな操作を実現するための言語環境として広まっていった．現在は，サーバ側，クライアント側，組み込み，などで広く使われている．

Java による実行例を以下に示す．

javac,java などのコマンドが無い場合は以下の様にしてインストールしよう．

```
k-okada@ubuntu-vm:~$ sudo apt-get install  sun-java6-jdk
```

途中で DJL ライセンスを受け入れるかきかれるので承認する必要がある．

```
$ cat test0.java
class test0 {
   public static int test(int i, int j) {
       return ( i * j );
   }
   public static void main(String args[]) {
       int i, j, k;
       i = 3;
       j = 2;
       k = test(i,j);
       if ( k > 5 ) {
           System.out.println(">5");
       } else {
           System.out.println("<=5");
       }
   }
}
$ javac test0.java
$ ls
test0.class test0.java
$ java test0
> 5
```

Perl

Perl は Larry Wall により開発された言語．1987 年に初期バージョンがリリースされている．ちなみに，Larry Wall は patch の開発者でも有名．Perl は実用性を第一に考え過去の C 言語やシェルスクリプト言語のよいところを取り込んでおり，特に正規表現が充実しているためテキスト処理や web の CGI などに利用されている．Perl による実行例を以下に示す．

```
$ cat test0.pl
sub test {
    ($i, $j) = @_;
    return ( $i * $j );
}

sub main {
    $i = 3;
    $j = 2;
    $k = &test($i,$j);
    if ( $k > 5 ) {
        print ">5\n";
    } else {
        print "<=5\n";
   }
}

& main();

$ perl test0.pl
```

Python

Python は 1995 年ごろに Guido van Rossum が開発したオブジェクト指向プログラミング言語．インタプリタ[7]として動くため，動的型付け，ガベージコレクションなどを特徴とする．また，以下にあるようにブロックの範囲をインデントで表す点に特徴がある．これは C 言語などでは括弧によってブロック構造を表すのと対照的である．見た目とブロック構造を対応つけることで，同じ構造のプログラムは同じ見た目となり，保守性の高いコードを実現できる．本体の仕様はシンプルにし，必要な機能は拡張モジュールで提供するようにしている．おそらく，現在最も勢いのある言語で，様々な拡張モジュールが利用でき便利な環境が実現されている．Google や Yahoo!等でも利用されていることで有名．

[7]プログラミング言語には大きくインタプリタ言語とコンパイル言語がある．C 言語のようなコンパイル言語ではソースコードからオブジェクトコードを生成しプログラムを実行するが，インタプリタ言語ではソースを逐次解釈しながら実行する．そのため，言語に対して，それを実行するためのインタプリタ環境が必要．これをしばしばランタイム環境と呼ぶ．ただし，インタプリタとコンパイルの両方の側面を持つ言語も存在する．例えば Java．javac を使ってコンパイルするが，その実行する java は class ファイルのランタイム環境と言える．が，Java で対話的に開発することはほとんどないであろう．

```
$ cat test0.py
def test(i,j):
    return ( i * j)

def main():
    i = 3
    j = 2
    k = test(i,j)
    if k > 5:
        print ">5";
    else:
        print "<=5";

 main()
$ python test0.py
```

注意したいのは python ではタブでブロックの範囲を指定するので，if 文の次の行などは必ずタブで字下げすること．

また Python はインタプリタとして，プログラムを編集しつつテストしつつ開発を進めることができる．python としてインタプリタを稼動させ，編集したファイルを読み込むことで，プログラムを実行できる．

```
$ python
>>> execfile("test0.py")
>5
```

ここで test0.py の test 関数を以下のように変更する．

```
def test(i,j):
    print "i = " + repr(i) + ", j = " + repr(j);
    return ( i * j)
```

そして，引き続きインタプリタでファイルを読み込む，変更を反映させた結果を見ることができる．また，関数のみの部分動作確認も可能である．

```
$ python
>>> execfile("test0.py")
i = 3, j = 2
> 5
>>> test(3,4)
i = 3, j = 4
12
```

慣れてくると，インタプリタでのプログラム開発はやめられなくなる．