オペレーティングシステム

2022/10/4

海谷 治彦

目次

- HCIについて

- GUIの基本
- レイアウト，カラー，フォント
- GUI vs CUI
- デスクトップ
- 国際化
- テキスト入力
- UX

HCI

- Human Computer Interaction
- なんだかんだいって，OSは人間とコンピュータの相互作用におけるコンピュータ側の窓口である.
- よって，特に利用者が直接使うOSの場合，HCIの話が重要となる.
  - 組み込み機器のOSは，あまり関係ないかも.
- OSやアプリ等のユーザーインタフェース (User Interface, UI) の設計は，多くのOSにとって重要な要素となる.

- 実は本学科には，HCI専用の授業が無いので，むりやりOSの授業に押し込んだ･･･という話もある.

人が入力してコンピュータが応答

- 人がコンピュータに何かを入力する.
- コンピュータが人に何かを応答する.
- コンピュータ開発当初から今でも基本的に上記の二つ，入力/応答はHCIの基本単位である.
  - 最近はコンピュータ側からの入力が先の場合もあるが.

- 初期のコンピュータ
  - 入力 配線，スイッチ，穴あいたカード
  - 応答 紙テープ 印刷物
- 中期のコンピュータ (今でも広く使われるコマンド言語)
  - 入力 コマンドと呼ばれる謎の呪文
  - 出力 入力同様の謎のテキスト列
- そしてGUI およびポストGUIの時代へ

再掲載

カード，ラインプリンタ

使いやすさとは？

- 「誰にでもわかりやすく，使いやすく」・・・
- 一般的に後述のGUIが使いやすいとされている.
- しかし，業務内容や状況，熟練，文化等によって，使いやすさは変わるので，結局，個々に検討する他無い.
  - 多く(100件以上等)のデータをコンピュータに登録するような「反復」業務はGUIでは面倒な場合が多い.

GUI

- Graphical User Interface
- 画面上の区画を指示する装置を用いてコンピュータに命令を送る仕組み.
- 区画: Window Icon
- 指示装置: マウス タッチパネル 等
- 命令: プログラムの起動と操作
  - ファイルを見る，音楽を聴く，メールを送る，地図をなぞる等.

GUIは何故使いやすい？

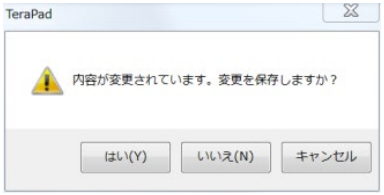
- 人間の直観に合っている場合が多い
  - 音楽っぽいIconを押すと音楽が流れる.
  - テキストファイルを押すとテキストが見られる.
  - 地図の左にいけば，左に隠れていた地図のほうにシフトする.

  等.

GUIの典型的な部品

- Icon コマンドやファイルをアクセスするための入り口．ソレっぽい画になってる．
  - 単純に押すと，それぞれに最も典型的な動作をする．
  - 後述のWindow内にもIconがある．
- Window: 結果が二次元的なもの(テキストや画像等)の場合，その結果を表示する領域一般．
  - 領域が画面に対して大きい場合，スクロールバーをつけ，一部のみ表示．
- メニュー，ボタン: 典型的でない動作を行わせる場合，それを選択するための表示．
- ダイアログ: 動作を行うために追加情報が必要な場合，それを取得するための領域．
  - Yes/NoダイアログはMacとWinで逆！

  だった（今はちょっとわからない）



- ウィザード: 追加情報を連続的に与える場合，複数のダイアログが連続的に利用される．

デスクトップIconとWindow

- Iconは，ほぼアプリケーションに対応し，Iconを押すことによってプロセスが起動する.
- 結果として，アプリに対応する Window が開くことが多い.
- Windowとアプリのプロセス(詳細は後日)は一対一では対応付かない.
  - Windowが無いプロセスもある，サーバーなど.
  - 複数のWindowを持つプロセスもある.
  - 1つのWindowは，どれか一つのプロセスに属する.

- Windowを見てアプリだ(プロセスだ)，とは思わないでください.

Windowのレイアウト

- タイトルバー
- メニュー (option)
- スクロールバー (option)
- Windowの中身(body)に何があるかはアプリ次第.

- レイアウトは基本的にOSの種類に合わせるように開発されているが，例外(違反)もある.
  - Adobeのソフトはわりと Windowsの標準にあってない（ので使いにくい）.
  - iTunes等のAppleのアプリをWindowsで使うと他との整合性があってなかった等 (最近の事情は不明)

メニュー

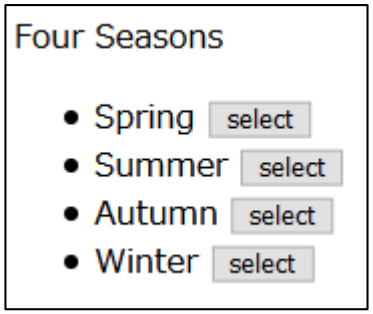
- アプリが受諾可能な入力(命令)を，予め，並べておいて，それを選択可能としたもの.
- 後述のCUIでもメニューはある.
- メニューによって，予め使い方を学ばなくても，コンピュータが操作可能な場合が多くなった.

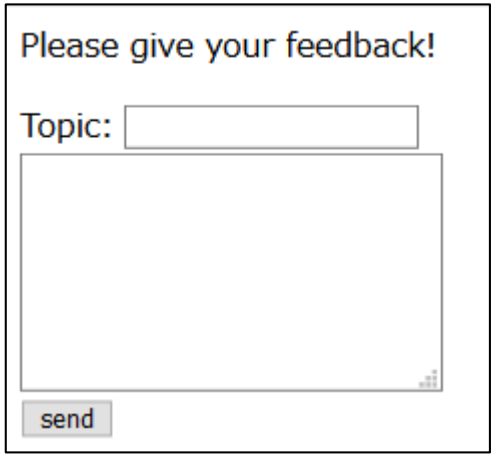
画像ファイルはデフォルトでは「プレビュー」の処理をするが，
その他の処理がしたい場合はメニューから選択．

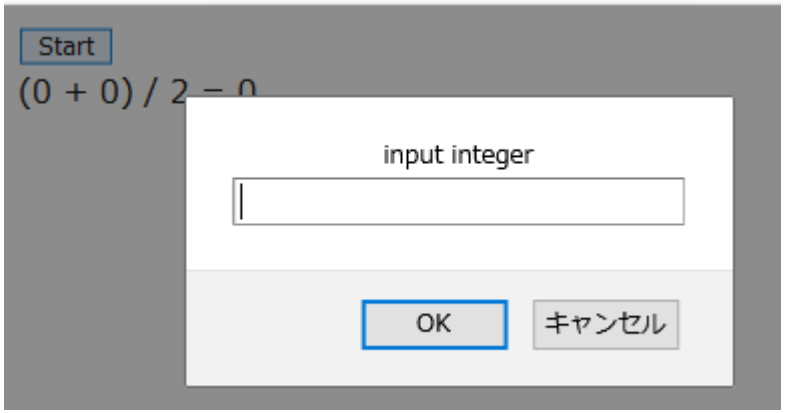
その他 GUI部品

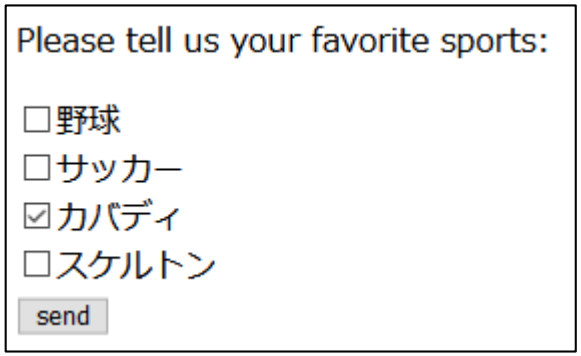
- HTMLやJava/SWT等のGUIキットの利用経験があれば名前を聞いたこともあるだろう.
- 実際のところ，実現する言語で対応しているGUI部品以外を使うのは困難なため，その点の事前チェックも必要.

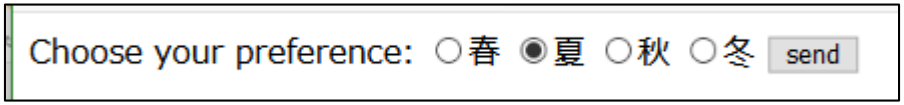
- button, prompt, confirm, form, checkbox, radio button, select box 等

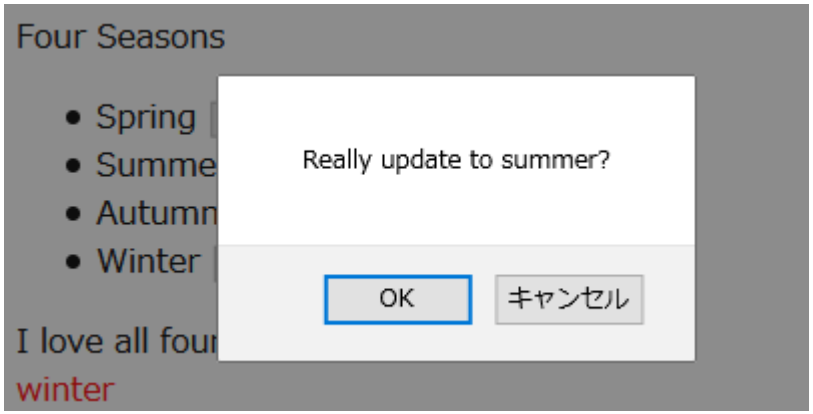

Four Seasons
Spring select
Summer select
Autumn select
Winter select


Please give your feedback!
Topic:
send


Start
(0 + 0) / 2 = 0
input integer
OK
キャンセル


Please tell us your favorite sports:
野球
サッカー
カバディ
スケルトン
send


Choose your preference: 春 夏 秋 冬 send


Four Seasons
Spring
Summe
Autumn
Winter
I love all four
winter
Really update to summer?
OK
キャンセル


Choose your preference: 夏 send
春
夏
秋
冬

GUI とデスクトップ

- デスクトップ:
  - コンピュータの基本的な操作画面.
  - デスクトップによって，何がどこにあるか決まっている.
    - 全体メニューが左下にある等.
  - フォント，色，基本的な配置等の規定もある.
- 代表的なデスクトップ
  - Windows のAero (Vista以降), Luna (XP)
  - Mac OSのAqua
  - X window system をベースにした GNOME, KDE
    - Linux, UNIX はこれらを採用.

レイアウト

- ポスター，新聞紙面，Webページと同様に，アプリのWindowに何をどこに配置するかの仕様.
- 基本的にグラフィックデザインの分野.
- 原則
  - 重要なものほど目立つようにする,
  - 逆に重要でないものは目ただないようにする.
- 重要性はフォントの種類，大きさ，色，配置場所等で表現することになる.

レイアウトのパターン 1

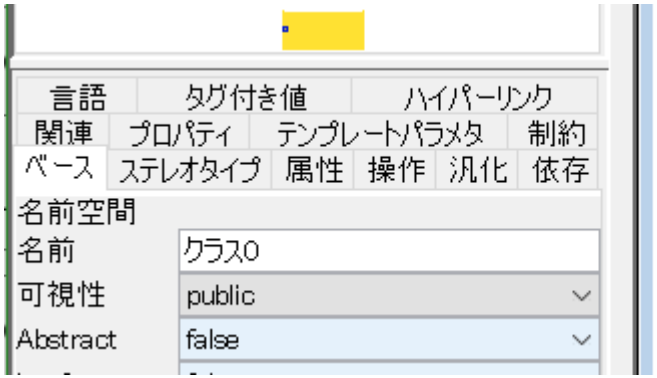
- 視覚的なフレームワーク
  - 複数の画面で構成される場合，全画面で一貫したレイアウトをとるのが良い.
    - フォントや色が同じ，メニューや戻るボタンの位置が同じ等
  - ある意味，大原則.
- 中央ステージ Center Stage
  - 重要な部分を大きく配置する.
  - 開発ツールやエディタ等は大抵，このレイアウト.
    - Eclipse, VScode, astah, パワポ ...
- 等分グリッド Grid of Equals
  - 項目を格子(grid)上に配置する.
  - 項目間の重要度が等分な場合など.
  - オンラインショップのサイトや，音楽，映像再生ソフト等.

レイアウトのパターン 2

コンテンツのグループ化のパターン群

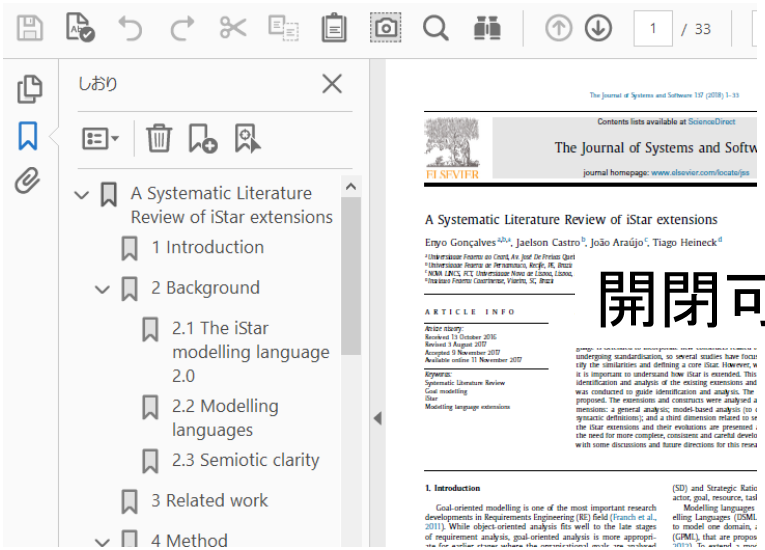
- タイトル付セクション Titled Sections
  - 論文や教科書等のように，見出しがあり，内容が分類されたレイアウト．とりあえず全部見えているのが特徴．
  - 例 アマゾンの注文済リスト等
- モジュール別タブ Module Tabs
  - 書類入れのタブのようなレイアウト．
  - 選択されているタブ以外は見えない．
- アコーデオン Accordion
  - 選択された項目のみ広がってる
- 開閉可能パネル Collapsible Panels
  - 必要な時に開閉できる．
- 移動可能パネル Movable Panels
  - パネル位置をカスタマイズできる．
    - 多くのIDEはコレに対応している．

タブ

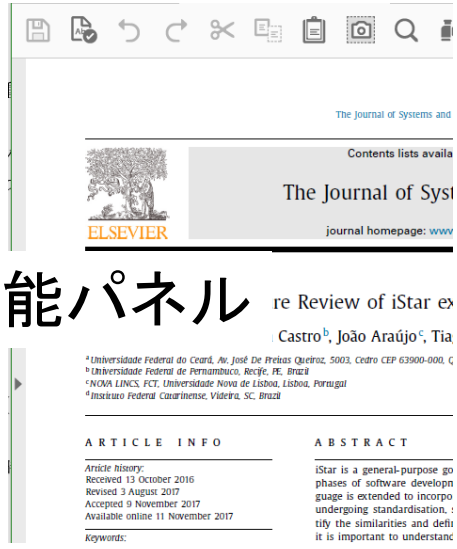


アコーデオン

開



閉



開閉可能パネル

レイアウトパターン 3

- 右揃えと左揃え
  - 文字通り
- 対角線バランス
  - 左上と右下に視覚的な重しをおいてバランスをとる.
  - 気分的には対角線配置の感じ.

色

- これも，グラフィックデザイン由来の話だが，いくつかの原則が有用.
- 背景色と前面色のコントラスト.
- 赤と緑を区別に使わない（色覚異常への対処）
- 寒色(青，緑等）は暖色（赤，黄等）より保守的な内容を示す画面に使われる.
- 暗い背景と明るい背景は大きく印象が違う.
- 強いコントラストは力強さや大胆さ，弱いのは穏やかさを示す.
- 強い彩度は活力等を示すが目が疲れるのでほどほどに.

フォント

- sans-serif ゴシックっぽい
  - Sans Serif
- serif 普通のフォント
  - Century
  - Times New Roman
- cursive 草書っぽい
  - Comic Sans MS
- fantasy 飾り文字
  - *Monotype*
- monospace 等幅フォント
  - Courier New
- その他，フォントにはitalic等のスタイルがある．
  - はっきりいって，italic と oblique の区別が私はつかない・・・

ココでの名前はHTML5準拠
下の具体的フォントは，
パワポで利用可能なもの

コマンド言語

- CUI (Character/Command User Interface) を通して利用者がOSに命令を行うための言語.
  - TUI Textual User Interface といってもいいかも.
- GUIで押したり引っ張ったりするのも広義では命令を言語化したものであるが，コマンド言語とは呼ばない.

- 歴史的経緯を除けば，ある作業を行うのに，GUI, CUIどちらが向いているかは，作業内容次第.
- 一般にCUIのほうが専門家向きといわれる.
  - あまり直観的でないため.

コマンド言語の構成

- 大体,「動詞 目的語1 目的語2・・・」という構成である.
  - man gcc
    - gccのマニュアル(manual)を表示
  - wc x.txt
    - x.txt のword を count する.
- Linux/UNIX系の場合，動詞を表す語句(コマンド)が意味不明なものが多い.
  - ls ファイルの一覧を表示 list segments の略らしい.
  - cat ファイルの中身を連接(concatenate)する.
  - pwd 現在の作業(working)フォルダ(directory)を表示(print)

パイプ&フィルタ モデル

- UNIX/Linux等のコマンド言語は，複数のコマンドを接続して，より複雑な処理を「その場で」(on the fly)構成することができる.
- このような処理構成法をパイプ&フィルタ モデルと呼ぶ.
- 処理対象のデータが行で区分けされたテキストでないと，うまく機能しない場合が多い.
- パイプは | で表現する場合が多い.
- 個々のコマンドがフィルタの役目をする.
- 話は簡単で,
  - 1個前のコマンドの出力を，次のコマンドへの入力とする.

  だけ.
- GUIでは，このような，臨機応変な対応が容易ではない.

リダイレクション

- UNIX系のコマンド言語では，ファイルの中身を，あたかもキーボード(標準入力)から入力したかのように処理することができる.
- 逆に，処理結果を画面(標準出力)ではなく，ファイルに直接に保存することができる.
- このような機構を redirection と呼ぶ.
- 入力の切り替えは < 出力の切り替えは > のシンボルを通常使う.
  - 見た目がソレっぽいため.
- 一応，Windowsでもできるが，たまに動作がおかしい.

例

拡張子が .txt のファイルだけ列挙しました.

```
E:¥>ls *.txt
bus.txt  log.txt  oh.txt  output1.txt  output2.txt  tel.txt  todo.txt
```

拡張子をとりました.

```
E:¥>ls *.txt | sed -n s/.txt//p
bus
log
oh
output1
output2
tel
todo
```

行番号をつけてみました

```
E:¥>ls *.txt | sed -n s/.txt//p | cat -n
     1  bus
     2  log
     3  oh
     4  output1
     5  output2
     6  tel
     7  todo
```

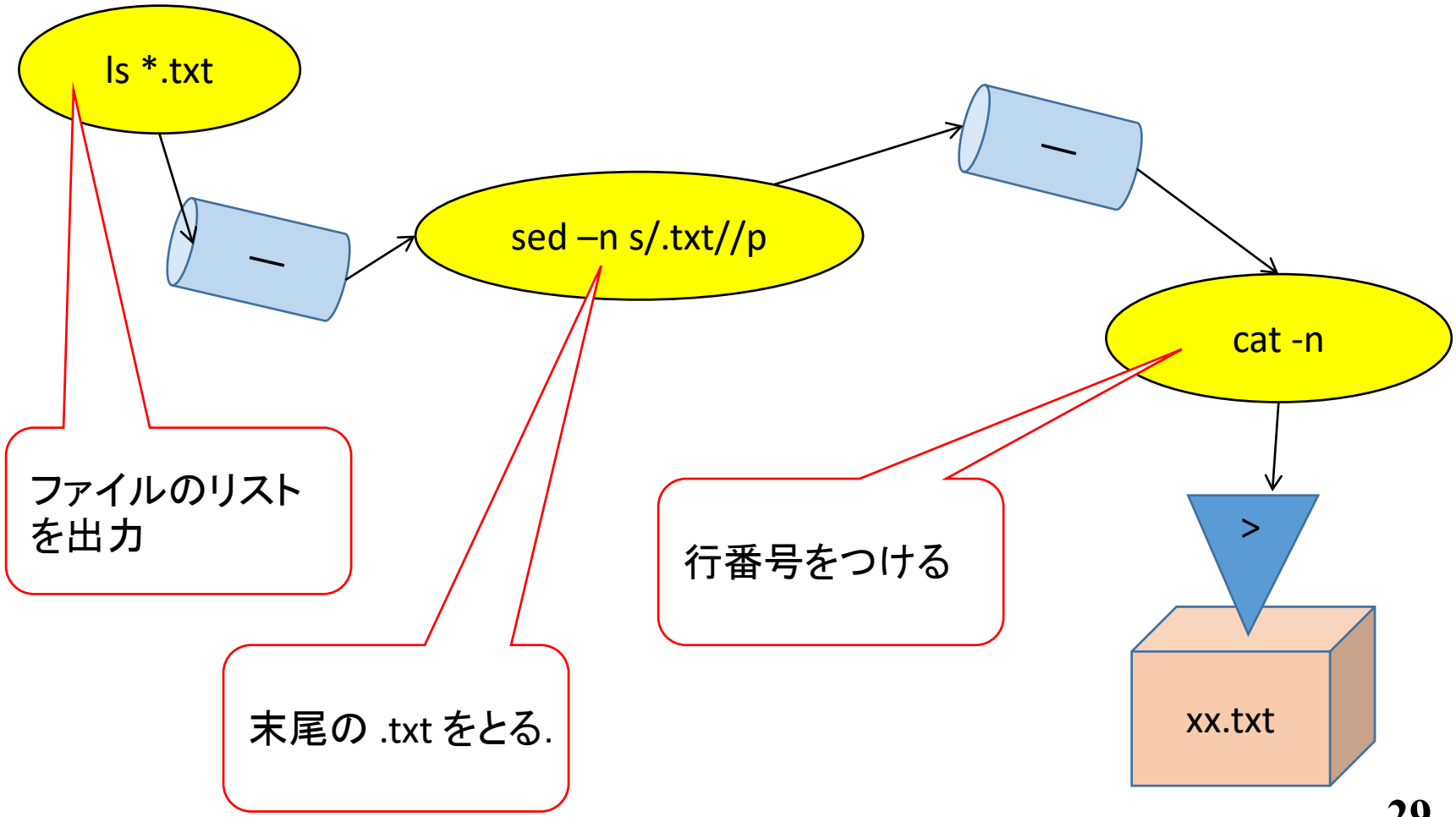
```
E:¥>ls *.txt | sed -n s/.txt//p | cat -n > xx.txt

E:¥>cat xx.txt
     1  bus
     2  log
     3  oh
     4  output1
     5  output2
     6  tel
     7  todo
     8  xx
```

結果を xx.txt に保存しました.

概念説明

```
ls *.txt | sed -n s/.txt//p | cat -n > xx.txt
```



正規表現

- CUIコマンドの多くでは，正規表現の利用が可能である.
  - 正確にはより表現力が弱い「ワイルドカード」を利用可能.
- 正規表現: 文字列の集合を一つの文字列で表現すること.
- 例
  - *.txt「.txtで終わる文字列全て」*.txt$ が正規表現
  - [a-f]* aからfで始まる文字列全て ^[a-f]* が正規表現
- 正規表現だけで本が書けちゃうくらいなので，詳細は，おそらくオートマトン系の授業で.

例

```
sh-3.1$ ls -d [a-fA-F]*
A.BAT  ALLUP.BAT    BACKUP.BAT  BIN         CFP   ETC
ADMIN  APPS         BACKUP.sh   BOOTEX.LOG  DOC   backup.jar
AGORA  AudioPlayer  BICYCLE     CC          DREE  bus.txt
sh-3.1$ ls -d [A-F]*
A.BAT  AGORA      APPS         BACKUP.BAT  BICYCLE  BOOTEX.LOG  CFP  DREE
ADMIN  ALLUP.BAT  AudioPlayer  BACKUP.sh   BIN      CC          DOC  ETC
sh-3.1$ ls *.txt
bus.txt  log.txt  oh.txt  output1.txt  output2.txt  tel.txt  todo.txt  xx.txt
sh-3.1$
```

shell

- コマンド言語を解釈するプログラムのことを shell もしくは command interpreter と呼ぶ.
- 多くのshellはコマンドを組み合わせて，より複雑な処理を行うための，制御構造を持っている.
  - コマンドの逐次実行
  - If then の条件分岐
  - ループ
  - サブルーチン
- 結果として，コマンドの組み合わせをプログラムの一種として実行することができる.
  - Shell script
  - Batch file と呼ばれるもの.

例

行数が100より大きいテキストファイルの，
行数，語数，文字数を列挙する．
という処理．

```
sh-3.1$ for i in *.txt
> do
>   if test `cat -n $i | wc -l` -gt 100; then
>     wc $i
>   fi
> done
 122  152 3019 output2.txt
 146  379 4833 tel.txt
 474  1377 17304 todo.txt
sh-3.1$
```

ちなみに，
全テキストファイルの
行，語，文字数

```
sh-3.1$ wc *.txt
  85  451 3368 bus.txt
   5   15  158 log.txt
  43  188 1921 oh.txt
  88  110 2142 output1.txt
 122  152 3019 output2.txt
 146  379 4833 tel.txt
 474 1377 17304 todo.txt
   8   16  103 xx.txt
 971 2688 32848 total
sh-3.1$
```

条件分岐(if fi)と，ループ(for in do done)を使っており，
プチ・プログラムな感じ．
プログラムが好きな人には扱いやすい．

Windows Power Shell のコマンド言語

- UNIX/Linux系より洗練されている・・・
- 登場がUNIX系コマンド言語の40年近く後なので，ある意味当たり前．
  - 逆に言えば，UNIX/Linux系は，40年前から，そこそこ使えるレベルだった．
- 尚，旧Windows互換のバッチファイルは，かなり酷い文法．
  - しかし，今でもわりと使われている．

GUIにおける組み合わせモデル

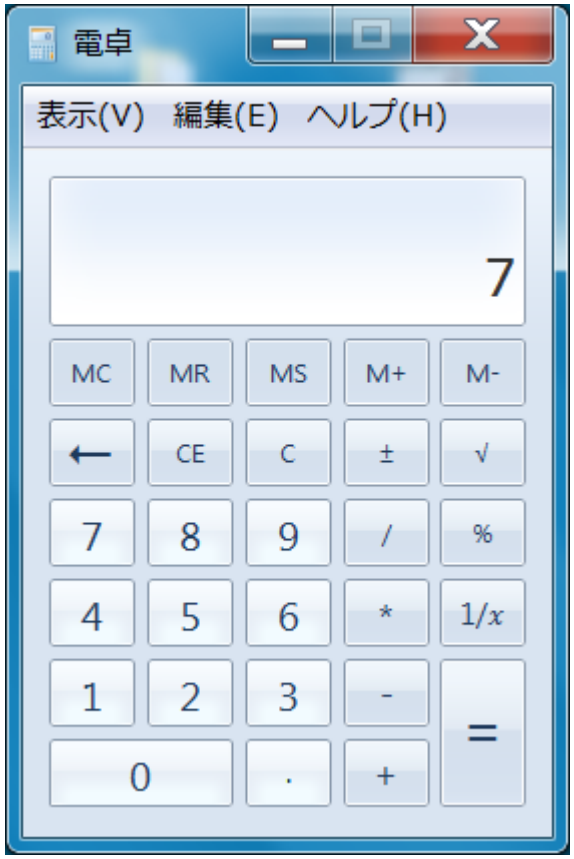
- 複数のGUIを組み合わせて，より複雑な処理を(場合によってはその場その場で)実現しようとする技術は存在する(した).
  - OLEやActiveX等のコンポーネント合成技術
- しかし，CUIのshellに比べ，微妙に普及度合いや標準化が進んでいないように感じる.
  - 本質的に二次元的な言語を定義するのが容易でないためと思われる.

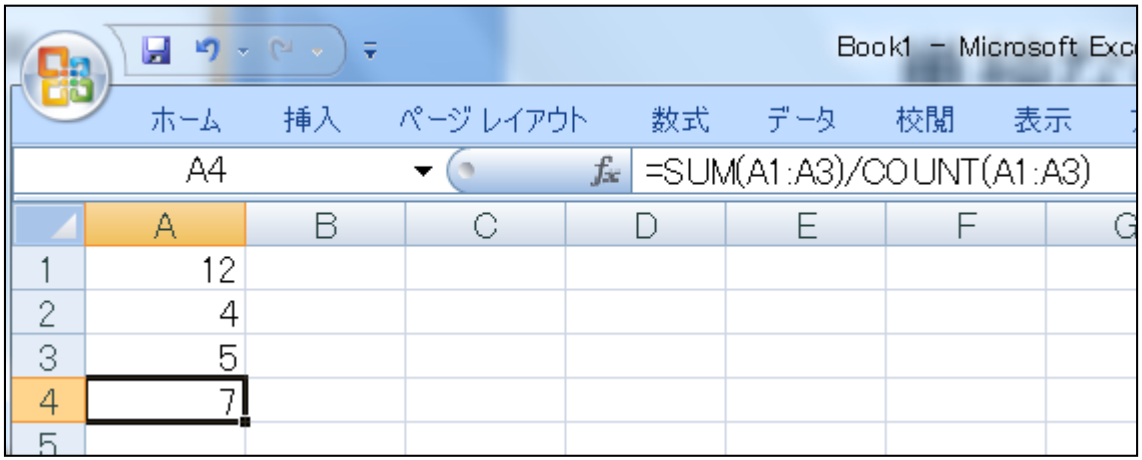
GUI vs CUI

- 適材適所である.
- GUI
  - 本質的に二次元的なデータを扱うのには適している.
    - 画像, 地図等
  - 人間の日常生活を模しているのであれば, 初心者にも使いやすい.
    - 本棚を模しているファイルの構造表示
    - ガールフレンドを模している対話型インタフェース
  - ある処理の複数の側面を同時に見たい場合.
    - 後述のデバッガの例.
- CUI
  - 正規表現, 条件分岐, 繰り返し等にマッチする処理には有利.
  - 処理の再利用も行いやすい. コマンド列を保存して再利用する等.

単純な数値計算 (12+4+5)/3

```
E:¥>bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2
This is free software with ABSOLUT
For details type `warranty'.
(12+4+5)/3
7
```

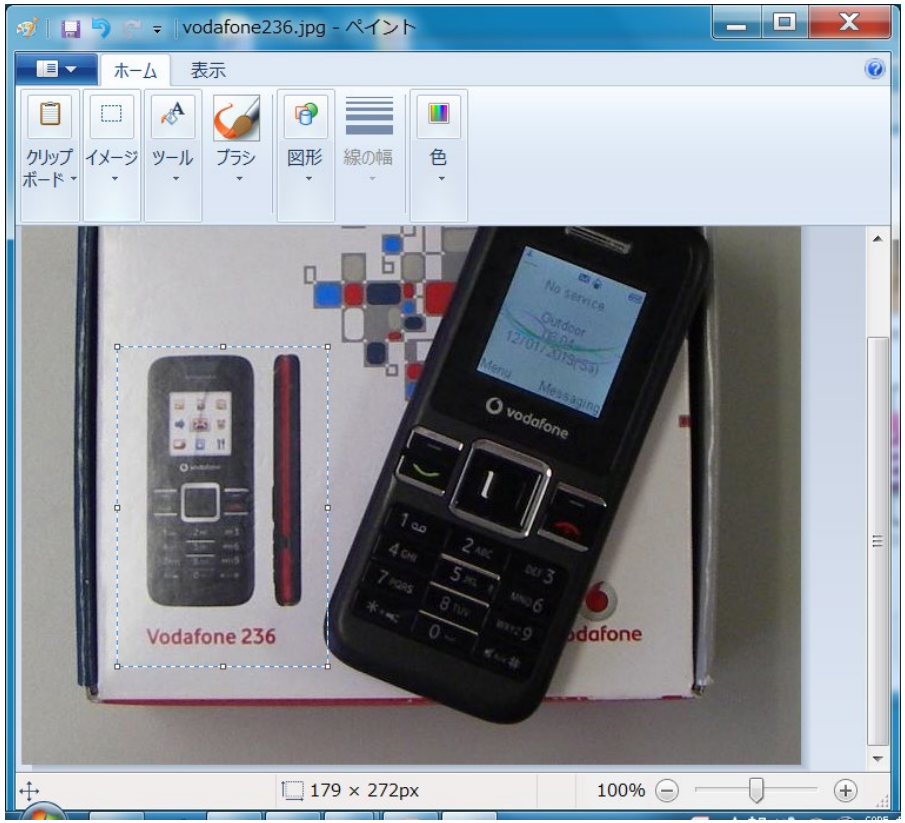




画像の部分抽出

CUIだと座標データを与える必要がある

```
C:¥LEC¥OS¥2014¥素材>convert -crop 200x400+50+186 vodafone236.jpg x.jpg
```



ImageMagick
というソフトの
一部だった

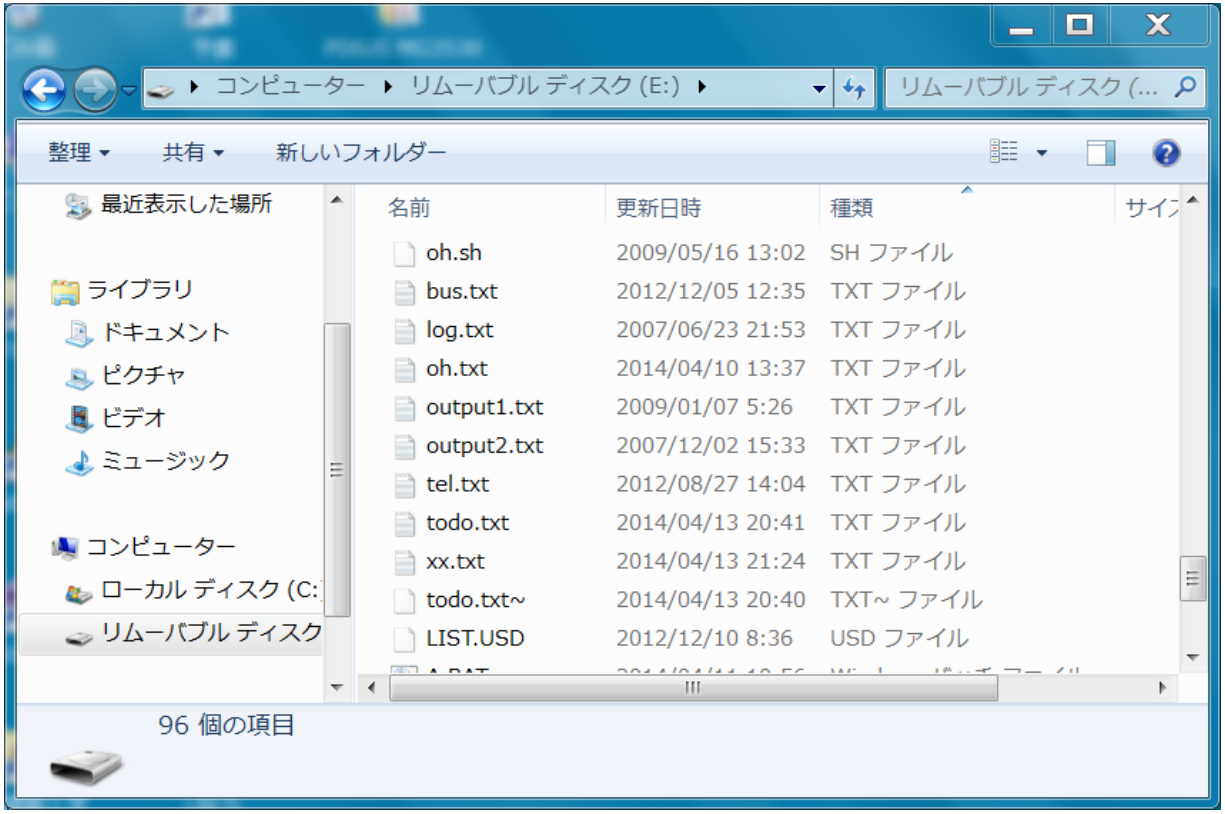
GUIだと直接,
領域を指定すればよい.

特定ファイルだけ削除

```
C:¥>rm *.txt
```

CUIだとコマンド一発

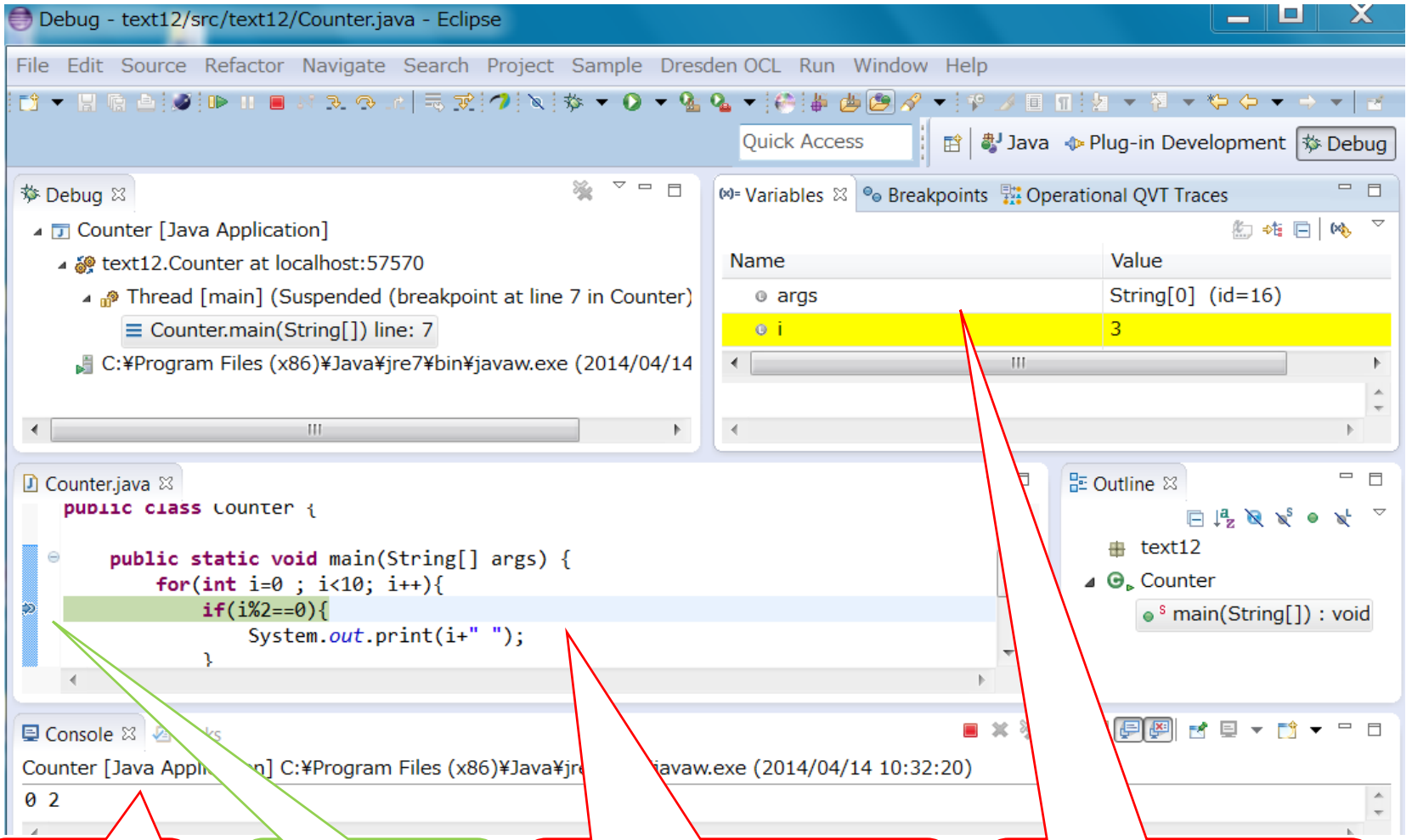
GUIだと精々ファイル種類順にソートするくらい



デバッガの例

- プログラムを作成する際，誤りの発見に役立つツールを Debugger と呼ぶ.
- プログラムの実行を途中で止めて，変数の値を見る等ができる.
- プログラム，実行結果，変数の変遷を同時に見たいところだが，CUIだとなかなか，そうもいかない.
- CUIだと延々とコマンドを打つことになる，結構，鬱になる.

GUIの例



実行
結果

一時
停止個所

プログラムと実行個所
がわかる.

実行途中の変数が
見える

同じことをCUIでやると・・・

```
C:¥LEC¥OS¥2014>javac -g Counter.java

C:¥LEC¥OS¥2014>jdb
jdbの初期化中...
> stop at Counter:5
遅延したブレークポイントCounter:5。
クラスがロードされた後に設定されます。
> run Counter
 Counterの実行
捕捉されないjava.lang.Throwableの設定
遅延した捕捉されないjava.lang.Throwableの設定
>
VMが開始されました: 遅延したブレークポイントCounter:5の設定

ヒットしたブレークポイント: "スレッド=main", Counter.main()、行=5 bci=8
5                       if(i%2==0){

main[1] locals
メソッド引数:
args = instance of java.lang.String[0] (id=403)
ローカル変数:
i = 0
```

コンパイルコマンドを打つ

デバッガを起動

何行目で途中停止するか指定

デバッグを開始

変数表示を命令

続き

```
i = 0
main[1] resume
すべてのスレッドが再開されました。
0
ヒットしたブレークポイント: > "スレッド=main", Counter.main()、行=5 bci=8
5                       if(i%2==0){

main[1] locals
メソッド引数:
args = instance of java.lang.String[0] (id=403)
ローカル変数:
i = 1
main[1] resume
すべてのスレッドが再開されました。
>
ヒットしたブレークポイント: "スレッド=main", Counter.main()、行=5 bci=8
5                       if(i%2==0){

main[1] locals
メソッド引数:
args = instance of java.lang.String[0] (id=403)
ローカル変数:
i = 2
main[1]
```

実行を再開

Androidにも CUIがある

- まぁ，ほとんどネタですが，
- UNIXのshellを模したプログラムが存在します．

Windows上でのUNIXコマンド

- Windows上でもUNIX系コマンドを動作させるソフトウェアがいくつか存在する.
- ちょっとした作業の際にわりと便利.
- Cygwin
  - http://www.cygwin.com/
  - こっちのほうが本格的だが，Winと融合してない.
- MinGW/Msys
  - http://www.mingw.org/
  - こっちのほうが小ぶりで，Winと融合している.
  - 今は Msys2 のほうがポピュラーか.
- WSL Windows Subsystem for Linux
  - コレは(ELFの)本物のLinux用コマンドが動く.

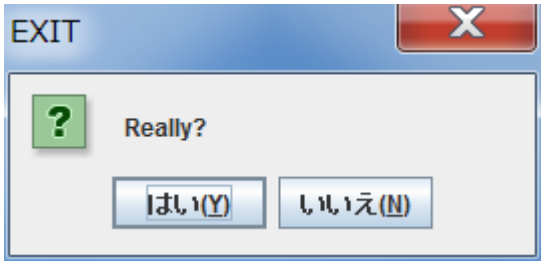
OSの国際化

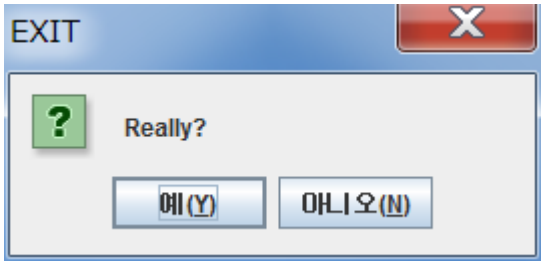
- Internationalization
- I18N とも略される
  - IとNの間の文字が18文字だから(マジ)
- LOCALE
  - 国に依存する言語，通貨記号，日付の書き順等をコンピュータ内で規定するもの.
    - たとえば欧州では日月年の順番が一般的.
  - LANGはLOCALEの一部.

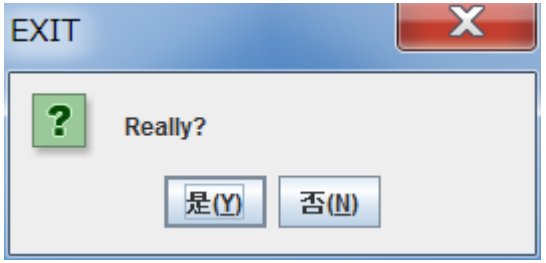
```
[kaiya@centos7 ~]$ export LC_ALL=ja_JP.utf8
[kaiya@centos7 ~]$ date
2018年  7月  2日 月曜日 11:11:13 JST
[kaiya@centos7 ~]$ export LC_ALL=de_DE
[kaiya@centos7 ~]$ date
Mo 2. Jul 11:11:24 JST 2018
[kaiya@centos7 ~]$ export LC_ALL=en_GB
[kaiya@centos7 ~]$ date
Mon  2 Jul 11:11:45 JST 2018
[kaiya@centos7 ~]$ export LC_ALL=ko_KR
[kaiya@centos7 ~]$ date
2018. 07. 02. (??) 11:12:37 JST
[kaiya@centos7 ~]$
```

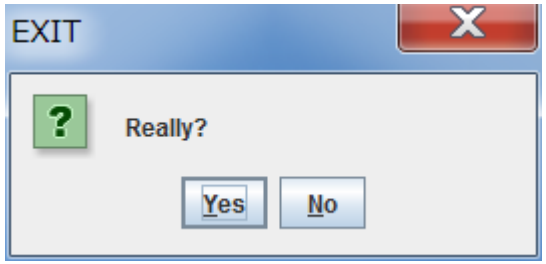
Localeの設定例

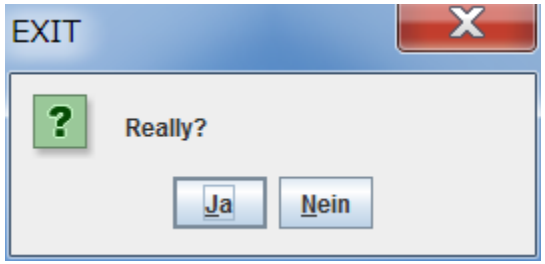
```
public static void main(String[] args) {
    Locale.setDefault(Locale.KOREAN);
    //Locale.setDefault(Locale.ENGLISH);
```











Localeの設定例

```
[kaiya@centos7 ~]$ export LC_ALL=ja_JP.utf8
[kaiya@centos7 ~]$ ./a.out
Segmentation fault (コアダンプ)
[kaiya@centos7 ~]$ export LC_ALL=de_DE
[kaiya@centos7 ~]$ ./a.out
Speicherzugriffsfehler (Speicherabzug geschrieben)
[kaiya@centos7 ~]$ export LC_ALL=fr_CA
[kaiya@centos7 ~]$ ./a.out
Erreur de segmentation (core dumped)
[kaiya@centos7 ~]$ export LC_ALL=fr_FR
[kaiya@centos7 ~]$ ./a.out
Erreur de segmentation (core dumped)
[kaiya@centos7 ~]$
```

```
[kaiya@flute03 ~]$ setenv LANG ja_JP.utf8
[kaiya@flute03 ~]$ ./a.out > /dev/null
セグメントエラー (coreを出力しました)
[kaiya@flute03 ~]$ setenv LANG C
[kaiya@flute03 ~]$ ./a.out > /dev/null
Segmentation fault (core dumped)
[kaiya@flute03 ~]$ setenv LANG de_DE
[kaiya@flute03 ~]$ ./a.out > /dev/null
Speicherschutzverletzung (core dumped)
[kaiya@flute03 ~]$ setenv LANG fr_CA
[kaiya@flute03 ~]$ ./a.out > /dev/null
Incident de segmentation (core dumped)
[kaiya@flute03 ~]$ setenv LANG fr_FR
[kaiya@flute03 ~]$ ./a.out > /dev/null
Incident de segmentation (core dumped)
[kaiya@flute03 ~]$
```

上記にように指定した言語で警告が出る.

昔，日本語訳が怪しかった (右)
(カナディアンフレンチとフレンチはこのレベルでは一緒ですね)

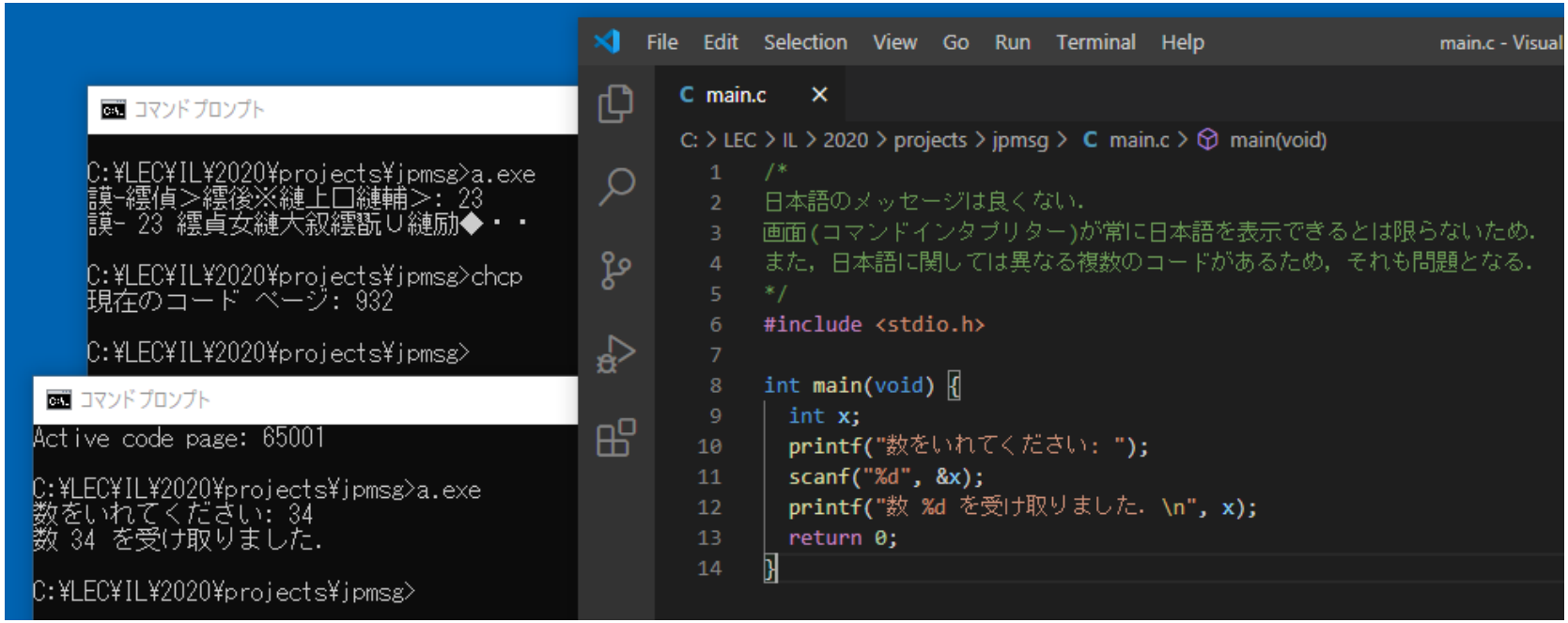
printf等で日本語表示はダメ

- 前述の通り，言語や地域を指定する環境変数に応じて，日本語やアラビア語等のローカルな言語に切り替えるのが王道.
  - 環境変数の詳細は後日に
- C言語等でprintf文で直接に日本語を表示すると，この切り替えができない.
  - というか多言語対応するには切り替え可能なようにプログラミングする必要がある，かなり面倒.
- プログラムで直に表示するメッセージはUS-ASCIIの範囲(英数文字)にとどめるべき.
- printf文で日本語表示はダメ.

注意

日本語メッセージはダメ

- printfにより日本語を直接表示するのは良くない.
- 正しく表示できない場合がある.
- 本格対応しないなら，英語表示のみにすべき.



文字コード

- ご存じの通り，コンピュータ内では，(日本語や英語の)個々の文字は，bit列として表現されている.
- ある文字をどのようなbit列で表現するかを，文字コードと呼ぶ.
- 残念ながら，日本語の文字の文字コードは多種類あり，統一されているとはいえない.
  - 文字コードによって同じ文字が違うbit列になる.

日本語の主な文字コード群

- JIS X 0208 および ISO-2022-JP
  - 単に JIS と呼ばれることもある.
- シフトJIS
  - CP932, MS932, Windows-31J とほぼ同じ
  - Windowsで採用されている
- EUC-JP
  - 昔はUNIXで広く採用されていた
- Unicode
  - 複数言語を同時に表現するのに便利なコード.
  - UTF-8, UTF-16等, いくつかのバリエーションがある.
  - UTF-8が広く使われているような気がする.

コード体系のポイント

US-ASCII と
互換性があるものがよい
UTF-8 が無難

多分，復習

ASCIIの表

USASCII code chart

b_7 b_6 b_5 → / Bits ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ / Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

例えば，100 0001 という7bitの値を文字 A と規定している．
101 1100 の ＼ が，日本のPCにおける ¥ になっている．

符号化形式と文字コード

- 文字コード
  - 文字をどのようなビット系列で表現するかの定義の体系.
  - 例: 前ページのASCII
  - Unicode 16bitで全世界の文字を表現しようとした体系，当然，16bit(約6.5万)では足りない・・・
  - UCS 31bitで表現しようとしたもの.
- 符号化形式
  - 個々の文字コードを，ファイルや通信で，どのような具体的なデータ列に変換するかの規約.
  - コードを，そのまま符号化すると，使ってない部分があって，わりとスカスカなので，その効率化が意図されている.
  - UTF-8, 16, 32 等は符号化形式の名前.

バイトオーダー

- 1個の文字を2バイト以上で表現する場合に起こる問題.
  - US-ASCII 以外，ほぼ全てに起こる問題.
- 通信やファイル格納の際に問題となる.
  - OSでは，レジスタやメモリ上のデータ配置で問題となる.

- Big Endian 上位バイトから先に送る方式 (TCP/IPはこっち)
- Little Endian 下位バイトから先に送る方式 (x86はこっち)

- ファイルや通信がどちらの流儀で保存や送付されているかを判断するために，データ冒頭に，0xFEFF のデータをおくことがある.
  - コレをBOM (Byte Order Mark) と呼ぶ.
  - BOM付データなら，例えば，最初のバイトが0xFEで，次が0xFFなら，Big Endian方式だとわかる.

テキストファイル 改行の流儀

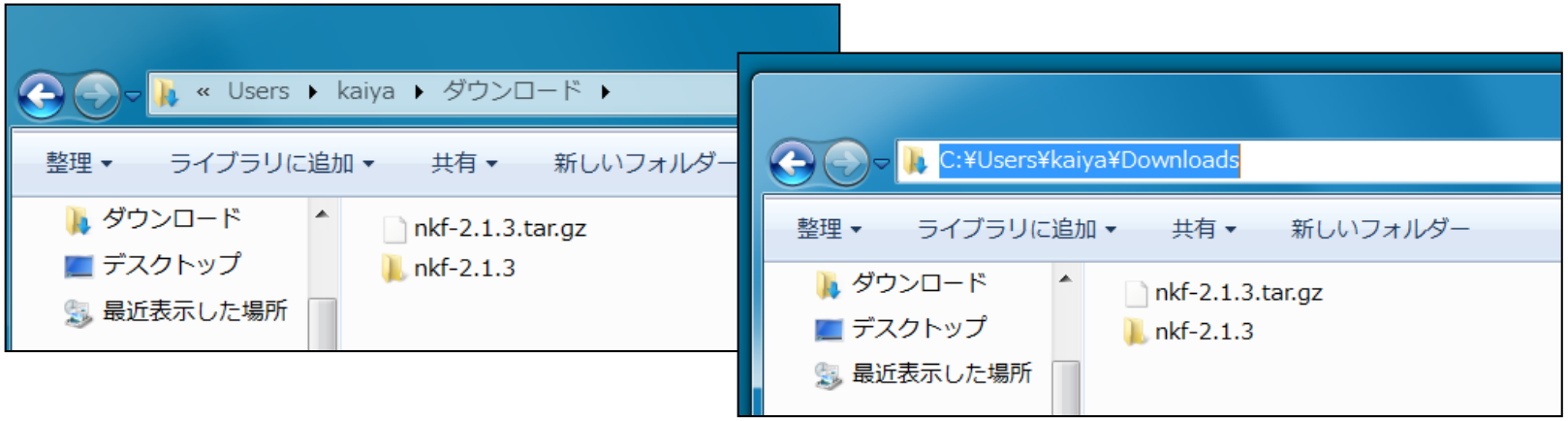
恐ろしいことに主要OS毎に違う！

- Windows
  - CR LF
- 旧MAC/Apple
  - CR
- UNIX and Linux
  - LF

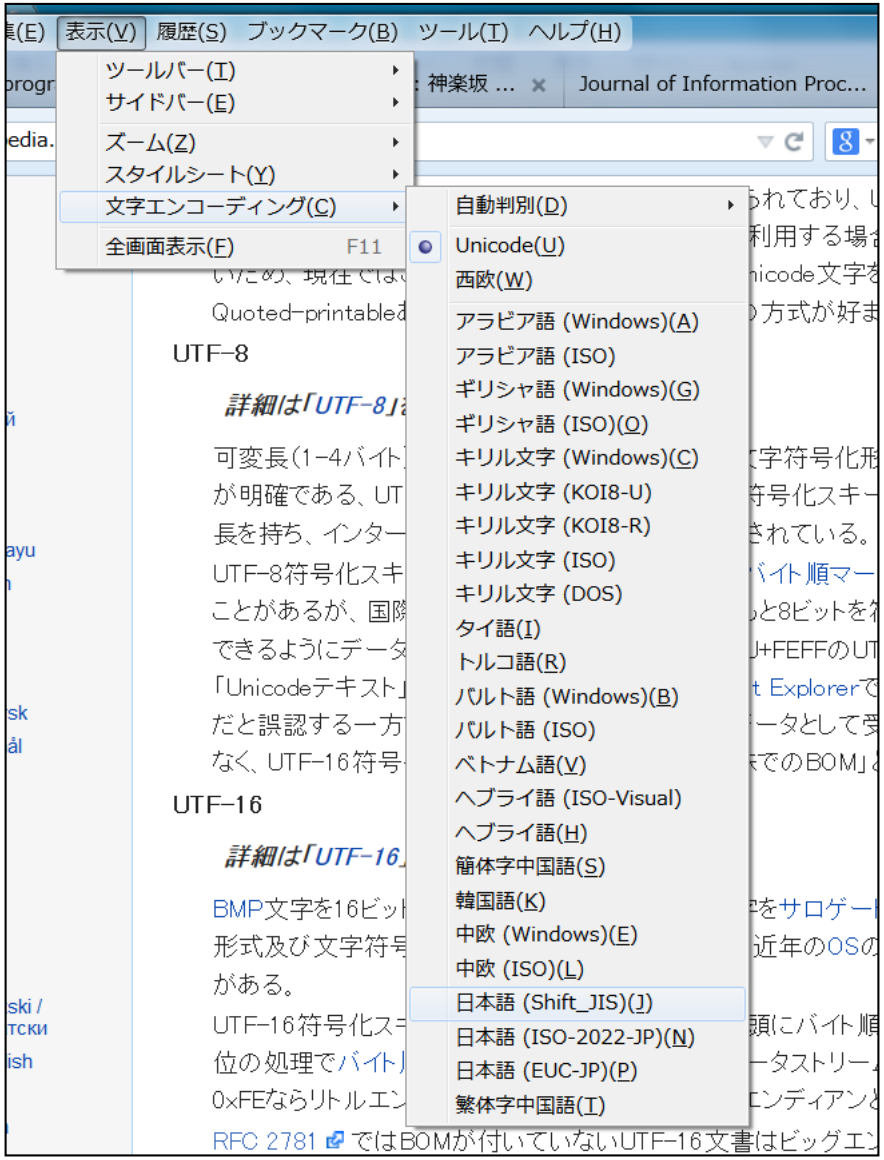
- CR Carriage Return
  - 行頭に戻る命令，タイプライター時代の名残
- LF Line Feed
  - 新しい行にいく命令，やはりタイプライター時代の名残

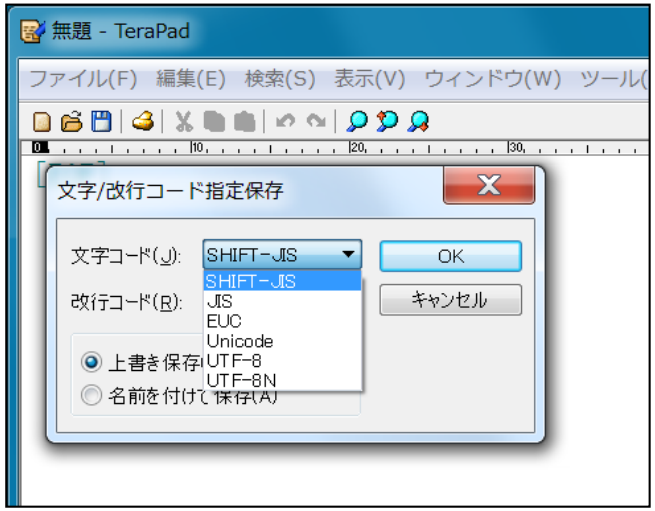
UIの多言語化について

- User Interfaceも言語に合わせて，カスタマイズされているものが多い．
- 日本語訳が残念で意味がわからん場合もある
- 内部的には英語だが，見た目だけ変えている場合もある．(下図)



ソフトやツールの対応


表示(V)
履歴(S)
ブックマーク(B)
ツール(T)
ヘルプ(H)
ツールバー(T)
サイドバー(E)
ズーム(Z)
スタイルシート(Y)
文字エンコーディング(C)
全画面表示(F) F11
自動判別(D)
Unicode(U)
西欧(W)
アラビア語 (Windows)(A)
アラビア語 (ISO)
ギリシャ語 (Windows)(G)
ギリシャ語 (ISO)(O)
キリル文字 (Windows)(C)
キリル文字 (KOI8-U)
キリル文字 (KOI8-R)
キリル文字 (ISO)
キリル文字 (DOS)
タイ語(I)
トルコ語(R)
バルト語 (Windows)(B)
バルト語 (ISO)
ベトナム語(V)
ヘブライ語 (ISO-Visual)
ヘブライ語(H)
簡体字中国語(S)
韓国語(K)
中欧 (Windows)(E)
中欧 (ISO)(L)
日本語 (Shift_JIS)(J)
日本語 (ISO-2022-JP)(N)
日本語 (EUC-JP)(P)
繁体字中国語(T)
Journal of Information Proc...
UTF-8
UTF-16


無題 - TeraPad
ファイル(F) 編集(E) 検索(S) 表示(V) ウィンドウ(W) ツール(
文字/改行コード指定保存
文字コード(J):
SHIFT-JIS
改行コード(R):
OK
キャンセル
SHIFT-JIS
JIS
EUC
Unicode
UTF-8
UTF-8N
上書き保存
名前を付けて保存(A)

日本語等のテキスト入力

- 日本語等の多数の文字がある言語では，キーボード等で文字を一意に指定するのは不可能.
  - キーボードでは，精々，100個程度のキーがあるだけ.
  - シフトキー等を使っても，扱える量はその数倍程度.
- Input method
  - キー，ペン，音声等の入力を文字列に変換するソフトウェア部品のことで，多くのOSに導入されている.
- Input method engine
  - 変換エンジン
  - IM中で，実際に変換を行うコンポーネント.
  - 日本語，中国語等毎にエンジンは異なるのが普通.
  - Microsoft IME 等が例.

フロントエンド vs バックエンド

- Front-end
  - 入出力デバイスのデータをIMが受け取り，確定した文字列をアプリに送る方式.
  - アプリ開発の観点からは簡単な方法.
- Back-end
  - アプリが入出力を受け取り，そのデータを下請けのIMに送る方式.
  - 変換候補が複数ある場合等には，アプリに合わせて，候補の表示等を凝ることができる.
    - 要はUIとしては，親切なものを作ることができる.
  - よって，アプリづくりはちょい面倒な場合もある.

UX

- User Experience ・・・ UE じゃないんかい！
  - よくある恣意的な略語 (例えば XML DX 等)
- 目的達成や使いやすさだけでなく，利用体験を総合的にとらえること・・・
- 感性や感情を優先
- どちらかといえば大衆向けソフトの概念
  - 飛行機や原子力プラント等では，目的達成が絶対的な基準である．開発ツールはどうかな？
  - 普通の人は，特に強い目的意識などなく，最終的に「よかった」と思えればそれでよい．
    - 買い物ソフトで，ほしい物が手に入らなくても，そのソフトを使って楽しめたなら良いという感じ．

参考書

- 椎尾一郎. ヒューマンコンピュータインタラクション入門. サイエンス社. 2020
- 井上 勝雄. インタフェースデザインの教科書. 丸善. 2013
- 大澤 範高. オペレーティングシステム. コロナ社. 2008.
- J. Tidwell. デザイニング・インタフェース. オライリー. 2011.

以上

100文字感想の提出忘れずに