2011 年度創造工学研修「E43 先進プログラミング体験（前半）」

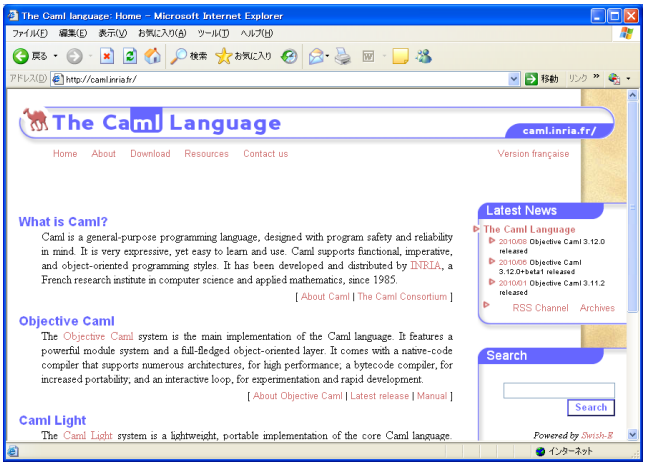
Objective Caml 入門ワークブック

0　プログラミングって何だろう？

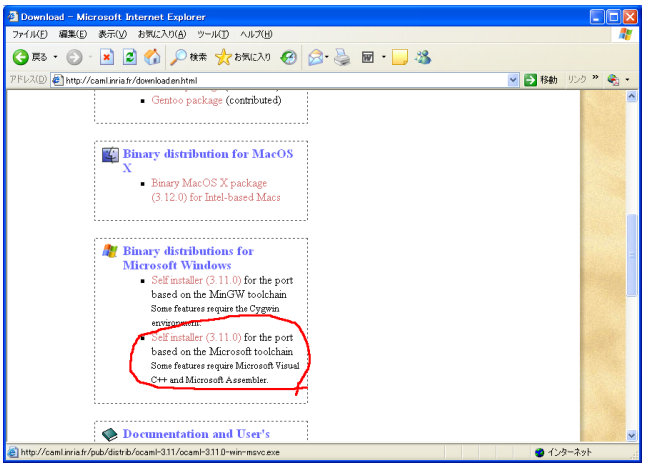
（キーワード：プログラム，プログラミング言語）

1　Objective Caml のインストール

1. WWW ブラウザで，`http://caml.inria.fr/` にアクセス．

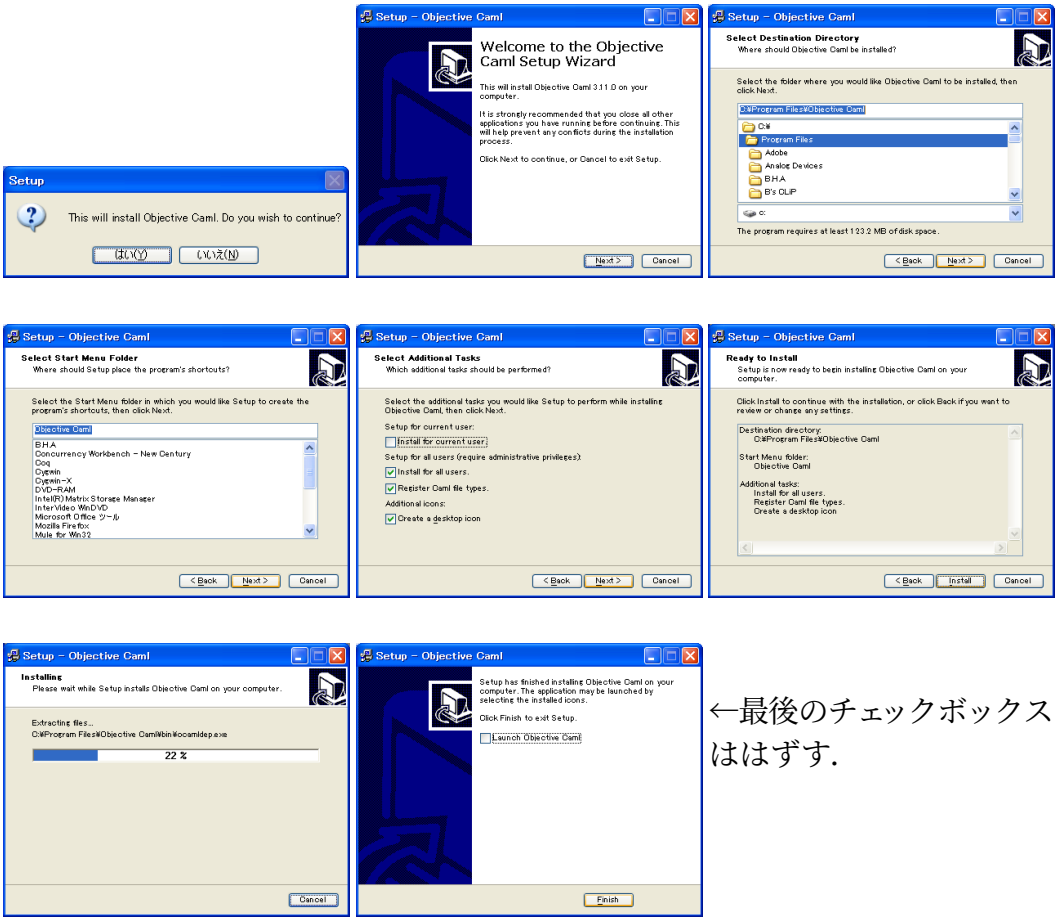


2. ページ上部「Download」のリンクを辿り，「Objective Caml」の「Self installer (3.11.0) for the port based on the Microsoft toolchain」の「Self install」リンクを辿ることで，`ocaml-3.11.0-win-msvc.exe` をダウンロード．



3. `ocaml-3.11.0-win-msvc.exe` を実行．

4. インストーラが起動するので，あとはインストーラの指示に従う．基本的に「はい」，「Next」,「install」,「finish」などをクリックすればよい．

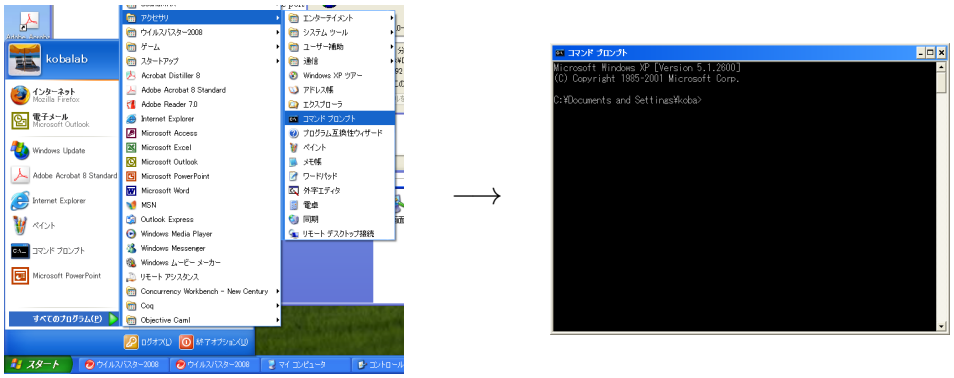


←最後のチェックボックスははずす．

2　Objective Caml の起動と終了

Objective Caml を起動してみよう

1. 「スタート」→「すべてのプログラム」→「アクセサリ」→「コマンドプロンプト」をクリック．



2. 出てきたウィンドウの中で「ocaml」と入力（かぎ括弧は入力しない．以下同様）し，Enter キーを押す．どのような画面が出力されただろうか？（キーワード：バージョン，プロンプト）

Objective Caml を終了してみよう

プロンプト「#」が表示された状態で「# quit;;」と打って Enter キーを押す．プロンプトの#とは別に，もう一つ自分で#を入力する必要があることに注意．

正常に終了したら，以下のためにまた Objective Caml を起動しておこう．なお，もし Objective Caml を実行してよくわからない状態になったら，「Ctrl キーを押しながら c」を押せばプロンプトに戻るので，間違えた問題からやりなおせばよい．それでもおかしかったら，Objective Caml 自体を終了し，起動しなおし，章の始めの問題からやりなおす．

3 整数演算

練習 3.1 「1 + 2 ;;」と入力し Enter キーを押してみよう．何が表示されたか？（キーワード：式，評価，値，型）

練習 3.2 「3 * 4 ;;」ではどうか？

練習 3.3 「1 + 2 * 3 ;;」ではどうか？「(1 + 2) * 3 ;;」は？（キーワード：優先順位）

練習 3.4 「+」や「*」の他に，整数の引き算を表す「-」や整数の割り算の商を表す「/」，余りを表す「mod」などが使える．それぞれの動作を確認してみよう．負の数ではどうか？たとえば，「3 - 5;;」，「7 mod 3;;」，「7 / 3;;」，「(-1) * 9;;」などを試してみよう．

一般に式を評価するには，式の後に「;;」をつけて Enter キーを押せばよい．以下では，式の後の「;;」をしばしば省略する．

4 小数演算

練習 4.1 「1.2 +. 3.4」，「1.2 *. 3.4」，「5.0 +. 6.7」をそれぞれ評価してみよう．「+」や「*」の後についている「.」を忘れないこと．

練習 4.2 「1.2 + 3.4」（「+」の後に「.」がない）や「5 +. 6.7」（5.0 の代わりに 5）を評価しようとすると，どうなるだろうか？（キーワード：型検査/型チェック）

練習 4.3 「`1.0 /. 3.0`」を評価するとどうなるか？「`1.0 +. 1.0 +. 1.0`」はどうか？（キーワード：浮動小数点，浮動小数点演算）

練習 4.4 Objective Caml を用いて，半径 1.23 の円の面積と周の長さを求めてみよう．ただし，円周率は 3.14159265359 とする．ヒント： 半径 r の円の面積は πr^2，周の長さは $2\pi r$.

5 変数の定義と使用

練習 5.1 Objective Caml を用いて，半径 4.5 の円の面積と周の長さを求めてみよう．半径 67.89 の円ではどうか？

練習 5.2 「`let pi = 3.14159265359`」と入力し，Enter キーを押してみよう．

練習 5.3 練習 5.2 を行った後で，「`pi`」という式を評価するとどうなるだろうか？「`1.23 *. 1.23 *. pi`」や「`2.0 *. 1.23 *. pi`」ではどうか？

定義を実行するときも，式の評価のときと同様に後に「`;;`」を付けて Enter キーを押せばよい．以下では定義を実行するときも，しばしば「`;;`」を省略する．

練習 5.4 練習 5.2 で定義した変数「`pi`」を用いて，課題 5.1 をやりなおしてみよう．

6 関数の呼び出し

練習 6.1 式「`sqrt 5.0`」を評価してみよう.「`sqrt 3.0`」や「`sqrt 4.0`」ではどうか？

注意： 普通の数学と異なり，Objective Caml では「`f(x)`」や「`g(5)`」などの括弧を省略して,「`f x`」や「`g 5`」のように書くことができる．ただし，引数が複雑な時は括弧が必要．自信のないときは括弧をつけること.（「`sqrt 3.0 + 1.0`」は「`sqrt (3.0 + 1.0)`」と,「`(sqrt 3.0) + 1.0`」のどちらだろうか？）

練習 6.2 平方根関数「`sqrt`」の他に，三角関数「`sin`」,「`cos`」,「`tan`」なども使うことができる（引数の単位はラジアン．π ラジアン $= 180°$）.「`sin (pi /. 2.0)`」などを評価して，いろいろな角度に対する三角関数の値を計算してみよう．

7　関数の定義

練習 7.1　半径 9.87 と 6.54 と 3.21 の円の面積をそれぞれ求めてみよう．

練習 7.2　「let menseki r = r *.　r *.　pi;;」と入力して Enter キーを押してみよう．どうなるだろうか？型はどう表示されるだろうか？

練習 7.3　「menseki 9.87」,「menseki 6.54」,「menseki 3.21」をそれぞれ評価してみよう．

注意：もし，pi の値を変更したら（なんと！），menseki の定義も実行しなおす必要がある．一般に，変数や関数の定義を変更したら，それを利用している変数や関数も定義しなおす必要がある．このことは，以下から確認できる．

```
# let half_pi = 0.5 *. pi;;
val half_pi : float = 1.570796326795
# let pi = 3;;
val pi : float = 3.
# half_pi;;
- : float = 1.570796326795
# let half_pi = 0.5 *. pi;;
val half_pi : float = 1.5
# half_pi;;
val - : float = 1.5
```

練習 7.4　同様に，円の半径から周の長さを求める関数 shuu を定義し，半径 9.87 と 6.54 と 3.21 の円の周の長さをそれぞれ求めてみよう．

8　グラフィックス

練習 8.1　「#load "graphics.cma"」と「open Graphics」をそれぞれ順番に実行してから,「open_graph ""」と「draw_circle 100 100 50」をそれぞれ順番に実行してみよう．何が起こったか？（キーワード：ライブラリ，モジュール，unit 値「()」と unit 型）

練習 8.2　関数 draw_circle の引数をいろいろ変えて実行することで，どの引数が何を表しているか推測してみよう．

練習 8.3　WWW ブラウザで，以下の手順により Objective Caml の Graphics モジュールのマニュアル日本語訳を開いて，関数 draw_circle の説明を探して読んでみよう．

1. http://ocaml.jp/ を開く．

2. 左メニューから「マニュアル」のリンクを辿る.

3. 「以前のバージョン」のリストの中から,「3.06(和訳版)」のリンクを辿る[1].

4. 「Part IV Objective Caml ライブラリ」の「The graphics library」のリンクを辿る.

5. ページ下部「Module Graphics: machine-independent graphics primitives」のリンクを辿る.

(上の代わりに直接`http://ocaml.jp/archive/ocaml-manual-3.06-ja/libref/Graphics.html`を開いたのでもよい.)

練習 8.4 関数`draw_rect`, `draw_ellipse`, `draw_arc`, そして`fill_rect`, `fill_circle`, `fill_ellipse`, `fill_arc`の説明を読んで, それぞれを実際に使ってみよう. 画面がゴチャゴチャしてきたら,「`clear_graph ()`」を実行することで, 画面をクリアできる.

練習 8.5 関数`set_color`に`color`型の値`red`, `blue`, `green`などを与えて実行しみよう. それから, `draw_cicle`などを呼び出すとどうなるだろうか?(あらかじめ用意されている色名は, マニュアルの「Some predefined colors」を見よ. それ以外の色を利用したいときは, 関数`rgb`を利用せよ. たとえば「`let gray = rgb 128 128 128`」や「`let violet = rgb 238 130 238`」など.)

練習 8.6 関数`moveto`と関数`lineto`の説明を読み, 実際に使ってみよう. たとえば, ×印を描いてみよう.

練習 8.7 整数xとyを受けとり, 座標(x, y)を左下とする一辺の長さが100の正方形を描く関数`square`を定義してみよう. ヒント: 関数`square`は引数を二つとるので,「`let square x y =`」のように書き始める. ヒント: 関数`draw_rect`を用いる.

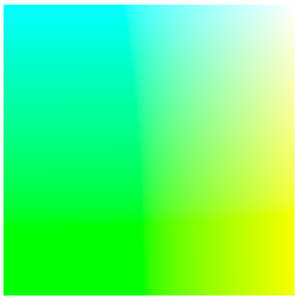
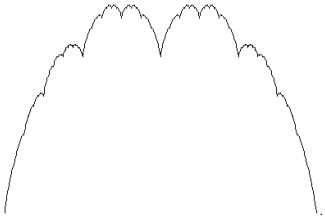
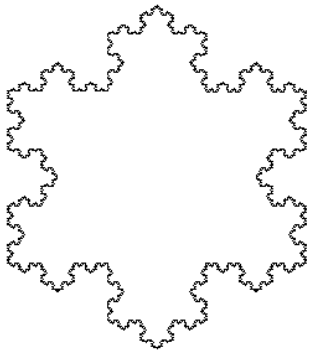
練習 8.8 整数xとyを受けとり, 座標(x, y)を上の頂点とする正三角形(一辺の長さ100, 底辺はx軸に平行)を描く関数`triangle`を定義しよう. ただし, 整数を小数にするには`float_of_int`を, 小数を整数にするには`int_of_float`をそれぞれ用いる. また, 複数の式を順に実行するには,「 式$_1$; 式$_2$; ...;式$_n$」のように書けばよい. ヒント: 一辺の長さがlの正三角形の高さは, $(\sqrt{3}/2)l$.

練習 8.9 練習8.8で定義した関数`triangle`を一辺の長さlも引数として取るよう拡張してみよう. ヒント:「`let triangle x y l =`」のように書き始める.

[1]本研修で使用するObjective Camlのバージョンは3.11.0であるが, 3.11.0の`Graphics`モジュールのマニュアルの日本語訳はまだされていない. 3.11.0の`Graphics`モジュールのマニュアルと3.06の`Graphics`モジュールのマニュアルには大きな違いはないため, 今回は3.06のマニュアルを利用する.

ミニギャラリー

再帰や条件分岐を使うことで，Graphics モジュールで様々な絵が描ける．いろいろ試してみよう．

9 再帰

（もし前章から継続してやらない場合は練習 8.1 を実行する）

練習 9.1 「clear_graph ()」を実行し画面をクリアしてから，次の関数定義を実行しよう．

```
let rec f x =
    if x < 1 then
        ()
    else
        (draw_circle 100 100 x; f (x/2))
```

（Objective Caml のプログラムは，単語の途中以外であればどこでも改行してよい．以下同様.）その後,「f 100」を評価してみよう．どんな絵が描けたか？ここで，

if 式$_1$ then 式$_2$ else 式$_3$

は，式$_1$ が成り立てば 式$_2$ を，成り立たなければ 式$_3$ を実行するという意味の式である．

（キーワード：再帰，条件分岐）

再帰は，高校でやった漸化式ととても似ている．たとえば漸化式

$$\begin{aligned} a_0 &= 0 \\ a_n &= n + a_{n-1} \quad (n > 0) \end{aligned}$$

の a_n は以下の再帰で計算できる．

```
let rec a n =
  if n = 0 then 0 else n + a (n-1)
```

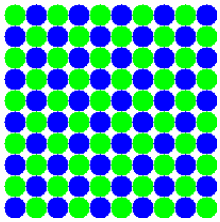
練習 9.2 練習 9.1 のプログラム中の数字や関数（draw_circle）をいろいろに変えて，どうなるか試してみよう．

練習 9.3 練習 9.1 の「if x < 1 then () else」の部分を忘れて

```
let rec f x =
    (draw_circle 100 100 x; f (x/2))
```

と定義してしまい,「f 100」を評価するとどうなるか？
注意： 実行を中止したいときは，Ctrl キーを押しながら，c を押す.

練習 9.4 再帰，条件分岐を用いて次のような絵を書いてみよう.



10 補足：let

「draw_rect 50 50 100 100」や「draw_circle 100 100 50」などの**式**は,

```
draw_rect 50 50 100 100; draw_circle 100 100 50
```

と，セミコロンを用いて並べることで，順に実行することができた．しかし,「let s = 25」や「let f x = x + 1」などの**定義**をセミコロンを用いて並べ順に実行することはできない.（「let s = 25; let f x = x + 1; f s」を実行し，確認してみよ.）
定義を並べるためには,「let 〜 in 〜」を用い,

```
let s = 25 in
let f x = x + 1 in
   f s
```

のように書く．let x = 式$_1$ in 式$_2$ は定義「let x = 式$_1$;;」を実行した後で，式$_2$ を評価することを表す.

練習 10.1 次の式を評価してみよう.

```
let s = 25 in
let f x = x + 1 in
   f s
```

11 補足：定義や式のファイルへの保存・読み込み

定義や式が大きくなってくると，それを毎回 ocaml を起動し入力していたのでは，入力間違いをした場合に最初から入力し直さなければならなくなる．これを避ける方法の一つとして，メモ帳（「スタート」→「すべてのプログラム」→「アクセサリ」→「メモ

帳」をクリック）などでプログラムを書いてから，ocaml にコピー&ペーストするという方法がある．

もっと便利な方法は，ファイルにプログラムを保存し，ocaml にそのプログラムを読み込ませることである．

練習 11.1 以下を行ってみよう．どんな結果になったか．

1. Objective Caml を「#quit」を実行し一旦終了させる．
2. コマンドプロンプトに「C:¥Documents and Settings¥souzou>」などと表示されてはず．この部分を覚えておく．
3. 「スタート」→「すべてのプログラム」→「アクセサリ」→「メモ帳」をクリックすることでメモ帳を起動し，以下のプログラムを入力する．

```
#load "graphics.cma"
open Graphics

let rec f x =
    if x < 1 then
        ()
    else
        (draw_circle 100 100 x; f (x/2))
```

4. メモ帳のメニューから「ファイル」→上書き保存をクリックし,「test.ml」という名前のファイルを,「マイ コンピュータ」以下「ローカル ディスク（C:）」以下「Documents and Settings」以下「souzou」直下に保存する．この時,「ローカルディスク（C:）」,「Documents and Settings」および「souzou」の部分は実際は覚えたおいたものを入力する.（例：「C:¥Documents and Settings¥souzou1>」となっていたら「souzou」の部分を「souzou1」に変える.）
5. コマンドプロンプトに戻り,「dir」と入力する．これまでの操作を正しく実行できていれば test.ml を含んだ行が表示されるはずだ．
6. 再び Objective Caml を起動する．
7. 「#use "test.ml"」を実行する．
8. 「f 100」を評価する．

課題

Objective Caml の機能を利用して，何か好きな絵を描いてみよう．9 章までに出てきたいろんな機能をできるだけ利用すること．

提出方法

できた絵と，その絵を描くのに用いたプログラム全部（式や変数定義・関数定義）を，kztk@kb.ecei.tohoku.ac.jp にメールで送る．ここで，

- 件名は「創造工学研修 2011：松田一孝」のように,「創造工学研修 2011：」の後に自分の名前をつづけたもの．
- 絵およびプログラムはメールに添付．
- プログラムのファイル名は「名前_名字.ml」，絵のファイル名は「名前_名字.bmp」．

とせよ．

絵の保存方法

絵をファイルに保存するには,「Caml Graphics」のウィンドウが最前面に出ている状態で，Alt を押しながら PrintScreen キーを押す．その後，ペイント（「スタート」→「すべてのプログラム」→「アクセサリ」→「ペイント」）を起動して,「編集」→「貼り付け」後,「ファイル」→「名前をつけて保存」とすればよい．

付録

A　ネットワークプログラミングその1：HTTP

練習 A.1　「`#load "unix.cma"`」,「`open Unix`」をそれぞれ実行して，`Unix` モジュール（ライブラリ）をロードしよう．

Unix とはオペレーティングシステム（OS）の名前である．いわゆるインターネットで使用されている TCP/IP 通信方式は，Unix を中心に発展してきた．現在では，Unix 以外の OS にも Unix と同様の TCP/IP 通信機能がある．

インターネットでは，`www.tohoku.ac.jp` のような人間にわかりやすい名前（ホスト名）のほかに，`208.77.188.166` のような数字のアドレス（IP アドレス）がある．

練習 A.2　「`let host = gethostbyname "www.tohoku.ac.jp"`」および「`let addr = host.h_addr_list.(0)`」を実行してみよう．さらに,「`string_of_inet_addr addr`」を評価してみよう．どんな結果が表示されたか？

いわゆるインターネットの「ホームページ」は HTML という言語で記述されている．

練習 A.3　WWW ブラウザで，`http://www.tohoku.ac.jp/english/` を開き，右クリックをし「ページのソースを表示」してみよう．

練習 A.4　以下の定義や式を順に実行ないし評価してみよう．

```
let (ic, oc) = open_connection (ADDR_INET(addr, 80));;
output_string oc "GET /english/ HTTP/1.0\r\n";;
output_string oc "Host: www.tohoku.ac.jp\r\n\r\n";;
flush oc;;
print_endline (input_line ic);;
print_endline (input_line ic);;
 ： (しばらく上記コマンドを繰り返す．コピー&ペーストしよう．)
print_endline (input_line ic);;
close_in ic;;
```

一般に WWW ブラウザは，このようにしてダウンロードした HTML 文書を解析・描画することで，ホームページを表示している．

B　ネットワークプログラミングその2：SMTP

練習 B.1　以下の定義や式を順に実行ないし評価してみよう．送信元は東北大学での自分のメールアドレス，送信先は自分の携帯のメールアドレスにしよう．もし携帯電話のメー

ルアドレスをもってなければ，自分の他のメールアドレスでもよい．**決して他人のメールアドレスを使用しないこと**．

```
let host = gethostbyname "primergy.kb-private";;
let addr = host.h_addr_list.(0);;
let (ic, oc) = open_connection(ADDR_INET(addr,25));;
print_endline (input_line ic);;
output_string oc "HELO localhost\r\n";;
flush oc;;
print_endline (input_line ic);;
output_string oc "MAIL FROM: 送信元メールアドレス\r\n";;
flush oc;;
print_endline (input_line ic);;
output_string oc "RCPT TO: 送信先メールアドレス\r\n";;
flush oc;;
print_endline (input_line ic);;
output_string oc "DATA\r\n";;
flush oc;;
print_endline (input_line ic);;
output_string oc "From: 送信元メールアドレス\r\n";;
output_string oc "To: 送信先メールアドレス\r\n";;
output_string oc "Subject: test mail\r\n\r\n";;
output_string oc "this is a test\r\n";;
output_string oc ".\r\n";;
flush oc;;
print_endline (input_line ic);;
output_string oc "quit\r\n";;
flush oc;;
print_endline (input_line ic);;
close_in ic;;
```

練習 B.2　練習 B.1 において，「送信元メールアドレス」に dummy@example.com（実在しない）を指定したらどうなるか？

C　3D プログラミング

練習 C.1　Objective Caml を終了し，ocaml のかわりに lablglut を起動してから，以下の定義や式を順に実行ないし評価しよう．(*と*) で囲まれた部分はコメントなので入力しなくてもよい．
注意：「‘」はバッククォートという記号で，標準的な日本語キーボードでは，Shift キーを押しながら@キーを押せば入力できる．「’」（クォート）とは異なるので気をつけること．「~」はチルダという記号で，標準的な日本語キーボードでは，Shift キーを押しながら^キーを押せば入力できる．

```
(* 初期化処理 *)
```

```
Glut.init Sys.argv;;

(* ウインドウの準備 *)
Glut.createWindow "souzou";;

(* ライトを有効化 *)
Gl.enable `lighting;;

(* 1 番のライトを白色の環境光に設定 *)
GlLight.light 1 (`ambient (1.0, 1.0, 1.0, 1.0));;
(* 1 番のライトを有効化 *)
Gl.enable `light1;;
(* 2 番のライトを赤色の点光源に設定 *)
GlLight.light 2 (`diffuse (1.0, 0.0, 0.0, 1.0));;
(* 2 番のライトを右方に設置 *)
GlLight.light 2 (`position (2.0, 0.0, 0.0, 0.0));;
(* 2 番のライトを有効化 *)
Gl.enable `light2;;
(* 3 番のライトを青色の点光源に設定 *)
GlLight.light 3 (`diffuse (0.0, 0.0, 1.0, 1.0));;
(* 3 番のライトを左方に設置 *)
GlLight.light 3 (`position (-2.0, 0.0, 0.0, 0.0));;
(* 3 番のライトを有効化 *)
Gl.enable `light3;;

(* 描画関数 *)
let rec display () =
  (* 画面をクリアするときの色を黒に設定 *)
  GlClear.color (0.0, 0.0, 0.0);
  (* 画面をクリア *)
  GlClear.clear [`color];
  (* サイズ 0.5 のティーポットを描画 *)
  Glut.solidTeapot 0.5;
  (* バッファをフラッシュ *)
  Gl.flush ();;

(* 描画関数を登録 *)
Glut.displayFunc display;;

(* キーボード処理関数 *)
let keyboard ~key:k ~x:x ~y:y =
  (* ティーポットを回転 *)
  GlMat.rotate ~angle:2.0 ~x:0.25 ~y:0.5 ~z:1.0 ();
  (* 描画関数 display を呼び出して全画面を再描画 *)
  display ();;

(* キーボード処理関数を登録 *)
Glut.keyboardFunc keyboard;;

(* メインルーチンを起動 *)
Glut.mainLoop ();;
```

以上をすべて実行すると,「souzou」ウィンドゥが出てくるのでスペースキーなど適当なキーを何度も押してみよう．終了するときは「コマンドプロンプト」ウインドウ右上の×ボタンでウインドウを閉じればよい．

D　マウスとキーボードからの入力およびレコードとバリアントについて

練習 D.1　「#load "graphics.cma"」,「open Graphics」,「open_graph ""」 を順に実行してから,「wait_next_event [Button_down]」を実行した後，Caml graphics ウィンドウの中の適当な一点をクリックしてみよう．クリックする位置によって，返り値はどのように変化するだろうか？（キーワード：レコード，ラベル）

練習 D.2　同様に「let s = wait_next_event [Button_down]」を実行してから,（Caml graphics ウィンドウの中の適当な一点をクリックした後で）「s.mouse_x」,「s.mouse_y」をそれぞれ評価してみよう．（キーワード：フィールド，フィールド値の取り出し）

練習 D.3　「let s = wait_next_event [Key_pressed]」を実行してから，Caml graphics ウィンドウで適当なキーを押し，その後で「s.key」を評価してみよう．（キーワード：バリアント，構成子）

練習 D.4　「let s = wait_next_event [Key_pressed;Button_down]」の場合はどうか？マウスをクリックした場合とキーを押した場合の両方を試してみよう．（キーワード：リスト）

E　リストと配列

どちらも「（同じ型の）複数の値を順番に並べて一つにまとめた値」．

リストと配列の書き方

リストは [で始まり，; で区切って，] で終わる．

```
# [ 1 ; 2 ; 3 ];;
- : int list = [1; 2; 3]
# [ 3 ; 2 ; 1 ];;      (* 順番は区別する *)
- : int list = [3; 2; 1]
# [ 1 ; 1 ; 2 ; 3 ];; (* 重複も区別する *)
- : int list = [1; 1; 2; 3]
```

配列は [| で始まり，; で区切って，|] で終わる．

```
# [| 1 ; 2 ; 3 |];;
- : int array = [|1; 2; 3|]
# [| 3 ; 2 ; 1 |];;     (* 順番は区別する *)
- : int array = [|3; 2; 1|]
# [| 1 ; 1 ; 2 ; 3 |];; (* 重複も区別する *)
- : int array = [|1; 1; 2; 3|]
```

リストと配列に対する操作

リストに対する操作

`x :: ys`

一つの**要素**x を，リスト ys の先頭に追加したリストを返す．

```
# let ys = [ 1 ; 2 ; 3 ];;
val ys : int list = [1; 2; 3]
# 4 :: ys;;
- : int list = [4; 1; 2; 3]
```

`xs @ ys`

リストxs の後ろにリスト ys を連結したリストを返す．

```
# let xs = [ 1; 2; 3; ];;
val xs : int list = [1; 2; 3]
# let ys = [ 4; 5; 6; ];;
val ys : int list = [4; 5; 6]
# xs @ ys;;
- : int list = [1; 2; 3; 4; 5; 6]
```

`match` xs `with [] ->` 式$_1$ `| x::ys ->` 式$_2$

リスト xs を調べ，空リスト [] であったならば 式$_1$ を評価する．空でなければ，先頭の要素を x，それ以外の残りのリストを ys として，式$_2$ を評価する．

```
# let e = [];;
val e : 'a list = []
# match e with [] -> "empty" | x::ys -> "non-empty";;
- : string = "empty"
# let xs = [1;2;3];;
val xs : int list = [1; 2; 3]
# match xs with [] -> -1 | x::ys -> x;;
- : int = 1
# match xs with [] -> [] | x::ys -> ys;;
- : int list = [2; 3]
```

リストが空でない確信があれば，[] -> 式$_1$ | の部分を省略してもよい．(ただし警告がでる．)

```
# match xs with x::ys -> ys;;
Characters 0-27:
  match xs with x::ys -> ys;;
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^
Warning P: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
- : int list = [2; 3]
```

リストが空のとき `[] -> 式`$_1$ `|` の部分を省略するとエラー（例外）が起こり，（例外処理しなければ）プログラムの実行が中止される．

```
# match e with x::ys -> ys;;
Characters 0-26:
  match e with x::ys -> ys;;
  ^^^^^^^^^^^^^^^^^^^^^^^^^^
Warning P: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
Exception: Match_failure ("", 1, 0).
```

配列に対する操作

`a.(i)`

配列 `a` の `i` 番目の要素を返す．

```
# let a = [| "apple"; "orange"; "banana" |];;
val a : string array = [|"apple"; "orange"; "banana"|]
# a.(0);;
- : string = "apple"
# a.(1);;
- : string = "orange"
# a.(2);;
- : string = "banana"
```

`a.(i) <- x`

配列 `a` の `i` 番目の要素を `x` で上書きする（この式自体の返り値は `unit`）．

```
# let a = [| "apple"; "orange"; "banana" |];;
val a : string array = [|"apple"; "orange"; "banana"|]
# a.(1) <- "melon";;
- : unit = ()
# a;;
- : string array = [|"apple"; "melon"; "banana"|]
```

`Array.make n x`

要素数が `n` で，全要素が `x` であるような配列を新たに作って返す．

```
# let a = Array.make 8 0.5;;
val a : float array = [|0.5; 0.5; 0.5; 0.5; 0.5; 0.5; 0.5; 0.5|]
# a.(1) <- 3.;;
- : unit = ()
# a;;
- : float array = [|0.5; 3.; 0.5; 0.5; 0.5; 0.5; 0.5; 0.5|]
```

リストと配列の違い（Objective Caml の場合）

リスト

- リストは::で「一つの要素を先頭に追加したリスト」を作るのが容易.
- リストは match で「先頭の要素とそれ以外の残りのリスト」を取り出すのが容易.

配列

- 配列は任意の i について「i 番目の要素」を高速に取り出すことができる.（リストでも List.nth xs i 取り出すことは可能であるが，i が大きいと時間がかかる）
- 配列は要素を**上書き**することができる（リストはできない）.