



OCaml はじめの一步
First Step to OCaml

小笠原 啓 (有)ITプランニング



デモンストレーション

- DownDown (graphicsモジュールを使った簡単なプログラム)
- Amthing (2Dベクトル描画ライブラリCairoを呼び出しています)
- Unison (lablgtkを使っています)
- ChartNaviPrime (サーバーサイドでOCaml製CGIやデーモンが24H356Dで動いています。)

Objective Camlってどんな言語？

- λ 計算を基礎にした静的で強い型付け言語。副作用が許されていて、正格評価。
- 先進の型理論が応用されており、型安全でありながら柔軟な型付けが可能。
- 9種類のCPUアーキテクチャをサポートしたコンパイラ。高速で効率のよいコードを出力してくれます。
- フランスのINRIA研究所にて開発されています。
- Qライセンスの元でソースコードが公開されています。



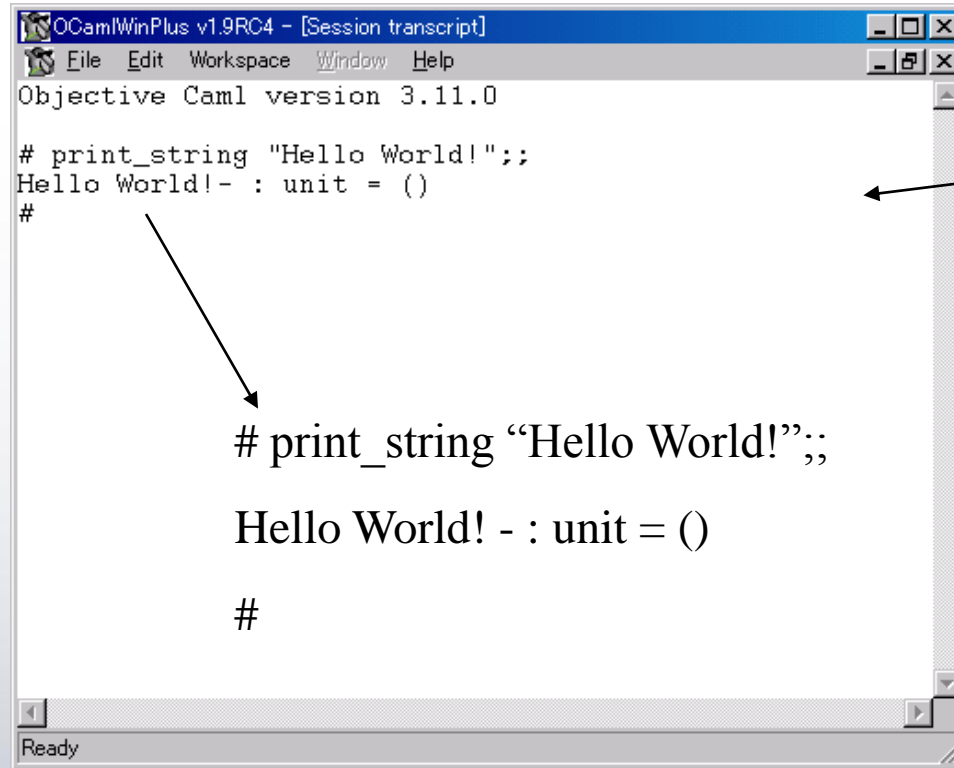
メインコントリビューター
Xavier Leroy氏



OCamlの歴史

- 元はエディンバラ大学で開発されたML(Meta Language)由来。
- 1987年、最初はCaml(Category Abstract Machine Language)としてスタート。
- 1996年にObjective Camlとしてリリース。
- 以来長年に渡り改良が続けられています。
 - 2000年 ラベル機能、多相バリエーション追加
 - 2003年 再帰モジュール追加
 - 2005年 camlp4刷新
 - 2008年 lazyパターンの導入

Hello World!



```
OCamlWinPlus v1.9RC4 - [Session transcript]
File Edit Workspace Window Help
Objective Caml version 3.11.0

# print_string "Hello World!";;
Hello World!- : unit = ()
#

# print_string "Hello World!";;

Hello World! - : unit = ()

#

Ready
```

トップレベルと呼ばれる
インタプリタ。
一行ずつプログラムを
実行して結果を表示して
くれます。



OCamlの基礎[1] リテラル

1234;; (* 整数 *)

- : int = 1234

3.141592 (* 浮動小数 *)

- : float = 3.141592

"OCaml Meeting in Tokyo 2009";; (* 文字列。ASCIIコード以外もOK *)

- : string = "OCaml Meeting in Tokyo 2009"

'A';; (* 文字 *)

- : char = 'A'



OCamlの基礎[2] リテラル

```
# [1; 2; 3; 4] ;; (* リスト *)  
- : int list = [1; 2; 3; 4]  
  
# [| 1; 2; 3; 4 |];; (* 配列 *)  
- : int array = [|1; 2; 3; 4|]  
  
# ("ogasawara", 33);; (* タプル *)  
- : string * int = ("ogasawara", 33)
```



OCamlの基礎[3] 変数

```
# let age = 33; (* 変数定義。バインディングと呼ぶ *)
```

```
val age : int = 33
```

```
# let x;; (* 変数だけ宣言することはできない。Nullが入った変数は存在しない *)
```

```
Characters 5-7:
```

```
let x;;  
  ^
```

```
Error: Syntax error
```




OCamlの基礎[4] 関数

```
# let birthday = fun age -> age + 1;; (* 関数定義。 *)
```

```
val birthday : int -> int = <fun>
```

```
# let birthday age = age + 1;; (* こう書いても上と同じ。 *)
```

```
val birthday : int -> int = <fun>
```

```
# let distance x y = sqrt( x *. x +. y *. y );; (* 二引数は引数を並べる *)
```

```
val distance : float -> float -> float = <fun>
```

```
# distance 3. 4.;; (* 関数呼び出し *)
```

```
- : float = 5.
```



OCamlの基礎[5] 制御構造(if式)

```
# if age >= 20 then "adult" else "child";; (* if式 *)
```

```
- : string = "adult"
```

```
#
```

式なので、if式全体で値になります。
三項演算子?のようなものです。

! NOTICE

制御構造としてはfor文、while文もありますが、
あえて紹介しません。滅多に使いません。



OCamlの基礎[6] 新しい型の定義/バリエーション

```
# type card = Number of int | Jack | Queen | King | Joker;;
```

```
type card = Number of int | Jack | Queen | King | Joker
```

```
#
```

card型は、Number of intかJackかQueenかKingかJokerのどれかで
あると定義しました。Number, Jack, Queen, King, Jokerの事を
“コンストラクタ”と呼びます。Number of intのintはコンストラクタの引数です。

```
# Jack;; (* Jackはcard型の値です。*)
```

```
- : card = Jack
```

```
# Number 9;; (* Number 9もcard型の値です。*)
```

```
- : card = Number 9
```



OCamlの基礎[7]新しい型の定義/レコード

```
# type preson = { name : string; age : int };;
```

```
type person = { name : string; age : int; }
```

person型はnameとageという要素を持つものと定義しました。要するに構造体です。

```
# let p1 = { name = "ogasawara"; age = 33; };; (* person型の値をp1にバインドしました *)
```

```
val p1 : person = {name = "ogasawara"; age = 33}
```

```
# p1.name;; (* ドットで各要素にアクセスできます *)
```

```
- : string = "ogasawara"
```



OCamlの基礎[8] パターンマッチ

```
# match [1; 2; 3] with
```

```
  [] -> ""
```

```
| hd :: tl -> string_of_int hd;;
```

(* ::演算子でリスト全体を先頭とそれ以降とで分けて表現。先頭の要素hdをstring_of_int関数に適用。*)

```
- string = "1"
```

Switch文のようなものです。

数字、文字列、リスト、タプル、バリエーションなど、何でも場合わけできます。



OCamlの基礎[9] パターンマッチの応用例

```
# let home = try Some (Unix.getenv "HOME") with _ -> None;;  
val home : string option = None
```

HOME環境変数を取り出します。ただし、それが定義されていなければNone、あればSomeにします。option型と呼ばれています。

```
# match home with  
  Some path -> print_string path  
| None -> "there is no HOME environment variable.";;
```

OCamlの基礎[10] モジュール

```
# module Counter = struct
  type t = int
  let make = 0
  let increment t = t + 1
  let show t = string_of_int t
end;;
module Counter :
sig
  type t = int
  val make : int
  val increment : int -> int
  val show : int -> string
end
```

モジュールは、複数の型や値をセットにしてひとまとまりの空間として分離できる機能。

シグネチャーによる型や値の抽象化・隠蔽が可能で、プログラムにこの構造を導入できます。

また、分割コンパイルの単位でもあります。

OCamlの基礎[11] さらに詳しくは

1. OCamlチュートリアル <http://www.ocaml-tutorial.org/ja>
2. Objective Caml 入門 (by 五十嵐先生)
 1. <http://www.sato.kuis.kyoto-u.ac.jp/~igarashi/class/isle4/mltext/ocaml.html>
3. 日本語の書籍





標準ライブラリの構成

- Pervasivesモジュール
 - 基礎的な関数を集めたモジュール。デフォルトオープン。ファイル操作なども可能。
- データ構造
 - List, Array, Queue, Hashtbl, Stack, Map, Set, Streamなど。
- Printfモジュール
 - 文字列のフォーマットを型安全に扱える。Formatモジュールも便利。
- Marshalモジュール
 - 値のマーシャリング(バイト列への変換)と読み込み。
- GC関連
 - Gc, Weak



付属ライブラリの構成

- Unixライブラリ
 - プロセス制御、シグナル、IO、ファイル操作、ソケット、ターミナル制御など。
- Numライブラリ
 - 任意制度演算。
- Strライブラリ
 - 正規表現と文字列操作。若干物足りない。
- Threadsライブラリ
 - スレッドの作成と同期的メッセージパッシングによる通信
- Graphicsライブラリ
 - ウィンドウの作成とグラフィックの描画。



便利ライブラリの紹介

- 正規表現ライブラリ pcre-ocaml
 - http://www.ocaml.info/home/ocaml_sources.html#toc12
- 拡張ライブラリ Core
 - <http://ocaml.janestcapital.com/?q=node/13>
- Unicodeライブラリ Camomile
 - <http://camomile.sourceforge.net/index.html.ja.jis>
- データベースアクセス、Webフレームワークなど色々あり。
 - postgres-ocaml, ocaml-mysql, CamlGI, ocamlnet2, eliom...

GUIツールキットもあります！

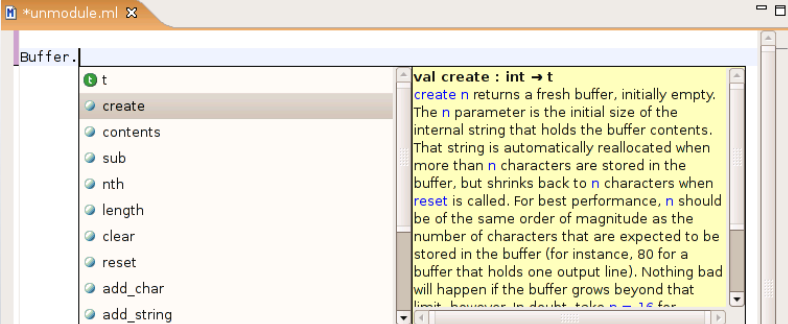
- LablGtk2ほぼ一択。Gladeも利用可能。



```
let main () =  
  let window = GWindow.window ~border_width:10 () in  
  window#event#connect#delete ~callback:delete_event;  
  window#connect#destroy ~callback:destroy;  
  let button = GButton.button ~label:"Hello World" ~packing>window#add () in  
  button#connect#clicked ~callback:hello;  
  button#connect#clicked ~callback>window#destroy;  
  window#show ();  
  GMain.Main.main ()  
let _ = main ()
```

開発環境

- Emacs + tuareg-mode + TAGS + camlspotter/パッチ + flymake
 - tuareg-modeで整形、色、部分評価などが可能です。
 - TAGSで関数名補完
 - camlspotterで値の定義位置へジャンプ
 - flymakeでリアルタイムコンパイル
- Vimの人は、ocaml.vim(もしくはomlet.vim?)
- OcaIDE Eclipseプラグイン
 - <http://ocaml.eclipse.ortsa.com:8480/ocaide/index.html>



The screenshot shows the OcaIDE Eclipse plugin interface. On the left, a list of methods for the 'Buffer.t' module is displayed: create, contents, sub, nth, length, clear, reset, add_char, and add_string. The 'create' method is selected, and its documentation is shown on the right. The documentation for 'create' is: 'val create : int -> t. create n returns a fresh buffer, initially empty. The n parameter is the initial size of the internal string that holds the buffer contents. That string is automatically reallocated when more than n characters are stored in the buffer, but shrinks back to n characters when reset is called. For best performance, n should be of the same order of magnitude as the number of characters that are expected to be stored in the buffer (for instance, 80 for a buffer that holds one output line). Nothing bad will happen if the buffer grows beyond that limit, however it might take n * 16 for

パッケージ環境

□ Mac portsに含まれるocaml関連のライブラリなど

caml-batteries @20090405 (devel, ml)
caml-calendar @2.0.4 (devel, ml)
caml-camlzip @1.04 (devel, ml)
caml-camomile @0.7.2 (devel, ml)
caml-csv @1.1.6 (devel, ml)
caml-extlib @1.5.1 (devel, ml)
caml-findlib @1.2.4 (devel, ml)
caml-json-static @0.9.6 (devel, ml)
caml-json-wheel @1.0.6 (devel, ml)
caml-ocamldb @0.9.11 (devel, ocaml)
caml-ocamlnet @2.2.9 (devel, ml)
caml-ounit @1.0.3 (devel, ml)
caml-pcre @5.15.0 (devel, ml)

caml-pgocaml @1.1 (devel, ocaml)
caml-postgresql @1.8.2 (devel, ml)
caml-sexplib @4.2.11 (devel, ml)
caml-sqlite3 @1.5.1 (devel, ml,
databases)
caml-type-conv @1.6.8 (devel, ml)
camlp5 @5.12 (lang)
ocaml @3.11.1 (lang, ml)
ocaml-bitstring @2.0.0 (devel, ocaml)
ocaml-mode.el @3.05 (lang, editors, ml)
ocamlgsl @0.6.0 (math, science)
ocamlsdl @0.7.2 (devel, multimedia)
tuareg-mode.el @1.45.6 (lang, editors)
xml-light @2.2 (devel, textproc)

開発事例



クライアント側はJava Applet

サーバーサイドはOCaml

- CGIで為替データを応答

- リアルタイムなレートをデーモンが配信

- バックで24H365Dのデータベース処理デーモンが稼動。



まとめ

- OCamlはλ計算を基礎とし、柔軟な型付けができ、実行効率のいいコードをはけるコンパイラ。
- 基礎的な文法やチュートリアルはWebや書籍を参照できます。
- 開発環境、ライブラリ、パッケージなど必要なものは結構揃っています。
- 研究室を飛び出し、現場のシステム開発にも応用されてきています。



OCamlで快適なシステム開発を！

ご清聴ありがとうございました。

IT Planning, Inc.

Web Site: www.itpl.co.jp Email: info@itpl.co.jp