Meta Language (ML): основы функционального программирования

Переменные, области видимости, владение и функции в языке ML

Фатахов Тимур, Буслама Анис 5030102/20201

Переменные — объявление

- Переменные объявляются через val
- По умолчанию — неизменяемые (immutable)
- Повторное присвоение создаёт новую переменную, не изменяет старую
- Для мутабельности используются ref, array, mutable record fields
- Такой подход уменьшает количество побочных эффектов

```
(* объявление неизменяемых переменных *)
val x = 10;
val s = "hello";
val pair = (x, s);
```

Output

```
> val x = 10: int;
> val s = "hello": string;
> val pair = (10, "hello"): int * string;
```

```
(* ref — контейнер с изменяемым содержимым *)
val r = ref 0;          (* r : int ref *)
r := !r + 1;            (* обновление значения внутри *)
val after = !r;         (* чтение: !r — разыменование *)
(* r сам фиксирован (val r = ...), но содержимое изменяемо *)
```

Output

```
> val r = ref 0: int ref;
> val it = (): unit;
> val after = 1: int;
```

```
val arr = Array.array (5, 0);  (* массив из 5 нулей *)
Array.update (arr, 2, 42);
val v = Array.sub (arr, 2);    (* v = 42 *)
```

Output

```
> val arr = [|0, 0, 0, 0, 0|]: int array;
> val it = (): unit;
> val v = 42: int;
```

Области видимости

- ML использует лексическую область видимости
- Локальные контексты создаются через let ... in ... end
- Внутренние переменные недоступны снаружи
- Возможен **shadowing** — переопределение имени во вложенной области
- Это повышает модульность и безопасность кода

```
1  val x = 100;
2
3  val result =
4    let
5      val x = 5          (* локальный x скрывает глобальный x *)
6      val y = x + 10
7    in
8      x * y
9    end;                 (* result = 5 * 15 = 75 *)
10
11 (* внешний x остаётся 100 *)
```

Output

```
> val x = 100: int;
> val result = 75: int;
```

```
1  local
2    val secret = "pwd123"
3  in
4    fun get_stub () = "secret hidden"
5  end;
6
7  (* secret недоступна вне блока local *)
```

Output

```
> val get_stub = fn: unit → string;
```

Владение и передача владения

- Управление памятью делает **сборщик мусора (GC)**
- Безопасность достигается через **неизменяемость данных**
- При мутабельности возможны несколько ссылок на один объект
- Владение можно имитировать через абстракции (модули, option)

```
1  (* Модуль, который владеет ресурсом *)
2  structure Buffer :> sig
3    type t
4    val create : unit -> t
5    val push : t * int -> unit
6    val toList : t -> int list
7  end = struct
8    type t = { data : int list ref }  (* внутренний мутабельный тип *)
9    fun create () = { data = ref [] }
10   fun push ({data}, v) = data := v :: !data
11   fun toList {data} = rev (!data)
12 end;
```

Output

```
> structure Buffer = struct
    val create = fn: unit → t;
    val push = fn: t * int → unit;
    val toList = fn: t → int list;
    type t;
  end;
```

Модули, option

Пример — имитация «перемещения владения» через option:

```
1  (* представим ресурс, который нужно "передать" и сделать недоступным у
   отправителя *)
2  datatype resource = R of string
3
4  (* функция move: берет resource option, возвращает новый option и высланный
   ресурс *)
5  fun move (NONE) = (NONE, NONE)
6    | move (SOME r) = (NONE, SOME r);
7
8  (* использование *)
9  val senderRef = ref (SOME (R "file_handle_1"));
10
11 (* отправка *)
12 val ((), receiverOpt) =
13   case !senderRef of
14       NONE => (print "nothing to send\n"; ((), NONE))
15     | SOME r => (senderRef := NONE; ((), SOME r));
16
17 (* теперь senderRef содержит NONE — имитация передачи владения *)
```

Output

```
> val R = R: string → resource;
> datatype resource = {
    con R = R: string → resource;
  };
> val move = fn: ∀ 'a 'b . 'a option → 'b option * 'a option;
> val senderRef = ref SOME (R "file_handle_1"): resource option ref;
> val receiverOpt = SOME (R "file_handle_1"): resource option;
```

Функции — объявление и входные данные

- Функции — первоклассные значения
- Определяются через `fun` или `fn`
- Поддерживают каррирование и частичное применение
- Можно передавать и возвращать функции
- Основной строительный блок программ в ML

```
1  fun add x y = x + y;
2  val add5 = add 5;
3  val r = add5 7;
```

Output

```
> val add = fn: int → int → int;
> val add5 = fn: int → int;
> val r = 12: int;
```

```
1  fun inc n = n + 1;
2  val a = 10;
3  val b = inc a;
```

Output

```
> val inc = fn: int → int;
> val a = 10: int;
> val b = 11: int;
```

```
1  fun appendRef (rRef: int ref, v: int) = rRef := !rRef + v;
2
3  val myRef = ref 3;
4  appendRef (myRef, 4);
```

Output

```
> val appendRef = fn: int ref * int → unit;
> val myRef = ref 3: int ref;
> val it = (): unit;
```

```
1  fun applyTwice f x = f (f x);
2
3  fun inc x = x + 1;
4  val res = applyTwice inc 3;
```

Output

```
> val applyTwice = fn: ∀ 'a . ('a → 'a) → 'a → 'a;
> val inc = fn: int → int;
> val res = 5: int;
```

Выходные данные функции (return)

- Возвращаемое значение — **последнее выражение** в функции
- Нет отдельного return
- Можно возвращать кортежи, функции, option-значения
- Поддерживается явное управление ошибками через option и result
- Поощряется чистый функциональный стиль

```
1  fun div_mod (a, b) =
2    if b = 0 then raise Fail "division by zero"
3    else (a div b, a mod b);
4
5  val (q, r) = div_mod (17, 4);   (* q=4, r=1 *)
```

Output

```
> val div_mod = fn: int * int → int * int;
> val q = 4: int;
> val r = 1: int;
```

Рекурсия

- Основной способ повторения вместо циклов
- Поддерживается хвостовая рекурсия (tail recursion)
- Позволяет писать эффективные рекурсивные алгоритмы
- Используется для обработки списков, деревьев и структур данных
- Возможна взаимная рекурсия (and) между функциями

```
1  fun factorial 0 = 1
2    | factorial n = n * factorial (n - 1);
3
4  (* маленький n — fine *)
5
```

```
1  fun fact_tail n =
2    let
3      fun loop (0, acc) = acc
4        | loop (k, acc) = loop (k - 1, k * acc)
5    in
6      loop (n, 1)
7    end;
```

```
1  fun map f lst =
2    let
3      fun loop ([], acc) = rev acc
4        | loop (x::xs, acc) = loop (xs, f x :: acc)
5    in
6      loop (lst, [])
7    end;
```

Output

```
> val map = fn: ∀ 'a 'b . ('a → 'b) → 'a list → 'b list;
```

Заключение

Итоги

- Переменные — неизменяемы по умолчанию
- Области видимости изолируют контексты
- Безопасность достигается неизменяемостью, не владением
- Функции — основная единица абстракции
- Рекурсия делает ML мощным и лаконичным

Список источников

- **"The Definition of Standard ML (Revised)"**, Robin Milner et al., MIT Press, 1997
- **"Programming in Standard ML", Robert Harper, Carnegie Mellon University**
- **Standard ML Basis Library, официальная документация**
- **MLton Wiki – материалы по компилятору и особенностям языка**