Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

Тестирование программных блоков
тема

Вариант 1

Преподаватель _____ А.С. Кузнецов
подпись, дата

Студент _____ В.А. Прекель
подпись, дата

Красноярск 2021

1 Цель работы

Изучить процесс блочного тестирования программного обеспечения.

2 Общая постановка задачи

Продемонстрировать понимание и владение навыками создания:

- простых блочных тестов;
- фикстуры тестирования;
- параметризованных тестов;
- документации разработчика.

2.1 Задание, соответствующее варианту

Вектор в Евклидовом пространстве https://en.wikipedia.org/wiki/Euclidean_vector в декартовых координатах https://en.wikipedia.org/wiki/Cartesian_coordinate_system, при этом должны быть реализованы и протестированы следующие функции:

- Сравнение;
- Сложение:
- Вычитание;
- Скалярное произведение;
- Вычисление длины.

3 Ход работы

Для работы был выбран язык OCaml и фреймворк для тестирования Alcotest.

Опишем тип модуля для работы с вектором, задокументируем:

Листинг 1 – файл lib/vector.mli

```
(** A vector with [n] dimensions which uses [float] as type for numbers *)
module type VEC = sig
  (** [n] is dimensions of vector *)
  val n : int

  (** [t] represents the vector *)
  type t
```

```
  (** [zero] is zero-length vector (zero vector) *)
  val zero : t

  (** [one] is vector with [1] in all dimensions (unit vector) *)
  val one : t

  (** [of_list lst] is vector created from list [lst].

      @raise NotEnough if [List.length lst < n] *)
  val of_list : float list -> t

  (** [equal ~eps x y] is result of equality comparing with given accuracy [eps] of
      vectors [x] and [y] *)
  val equal : ?eps:float -> t -> t -> bool

  (** [pp ~start ppf x] pretty-prints vector [x] using formatter [ppf] *)
  val pp : ?start:bool -> Formatter.t -> t -> unit

  (** [show x] is string representation of vector [x] *)
  val show : t -> string

  (** [add x y] is sum of vector [x] and vector [y] *)
  val add : t -> t -> t

  (** [sub x y] is subtraction of vector [x] and vector [y] *)
  val sub : t -> t -> t

  (** [mult x k] is vector [x] with lengths multiplied by [k] *)
  val mult : t -> float -> t

  (** [dot_product x y] is scalar (dot) product of vector [x] and vector [y] *)
  val dot_product : t -> t -> float

  (** [length_squared x] is length of vector [x] multiplied by self *)
  val length_squared : t -> float

  (** [length x] is length of vector [x] *)
  val length : t -> float
end
(** Infix operations with vector *)
module type VECINFIX = sig
  (** [t] represents the vector*)
  type t

  (** Module with operators itself *)
  module Infix : sig
    (** [x = y] is true if vectors coordinats equal to each other with epsilon = 0 *)
    val ( = ) : t -> t -> bool

    (** [x + y] is sum of vector [x] and vector [y] *)
    val ( + ) : t -> t -> t
```

```
    (** [x - y] is subtraction of vector [x] and vector [y] *)
    val ( - ) : t -> t -> t

    (** [x *^ k] is vector [x] with lengths multiplied by [k] *)
    val ( *^ ) : t -> float -> t

    (** [k ^* x] is vector [x] with lengths multiplied by [k] *)
    val ( ^* ) : float -> t -> t

    (** [x *.* y] is scalar (dot) product of vector [x] and vector [y] *)
    val ( *.* ) : t -> t -> float
  end
end
```

Реализуем функтор, который принимает тип модуля вектора и строит инфиксные операции:

Листинг 2 – файл lib/vector.ml

```
module MakeVecInfix (V : VEC) = struct
  let ( = ) = V.equal ~eps:0.
  let ( + ) = V.add
  let ( - ) = V.sub
  let ( *^ ) = V.mult
  let ( ^* ) a b = V.mult b a
  let ( *.* ) = V.dot_product
end
```

Реализуем модуль для вырожденного вектора (0-мерный):

Листинг 3 – файл lib/vector.ml

```
module VZ = struct
  module T = struct
    let n = 0

    type t = unit

    let zero = ()
    let one = ()
    let of_list _ = ()
    let equal ?eps:_ _ _ = true
    let pp ?(start = true) ppf a = if start then Caml.Format.fprintf ppf "()"
    let show _ = "()"
    let add _ _ = ()
    let sub _ _ = ()
    let mult _ _ = ()
    let dot_product _ _ = 0.
    let length_squared _ = 0.
    let length _ = 0.
  end

  include T
  module Infix = MakeVecInfix (T)
```

```
end
```

Реализуем функтор, который принимает модуль типа вектор и строит модуль с кол-вом измерением на 1 больше:

Листинг 4 – lib/файл vector.ml

```
exception EmptyList of int

exception
  NotEnough of
    { need : int
    ; got : int
    ; failed_on : int
    }

module VS (V : VEC) = struct
  module T = struct
    let n = V.n + 1

    type t = float * V.t

    let zero = 0., V.zero
    let one = 1., V.one

    let of_list = function
      | [] -> raise @@ EmptyList n
      | a :: b ->
        begin
          try a, V.of_list b with
          | EmptyList ni ->
            raise @@ NotEnough { need = n; got = List.length b + 1; failed_on = ni }
        end
    ;;

    let equal ?(eps = 1e-6) (a, x) (b, y) =
      let open Float in
      abs (a -. b) <= eps && V.equal x y ~eps
    ;;

    let pp ?(start = true) ppf (a, x) =
      let open Caml.Format in
      if start then pp_open_hbox ppf ();
      pp_print_float ppf a;
      pp_print_space ppf ();
      V.pp ~start:false ppf x;
      if start then pp_close_box ppf ()
    ;;

    let show x = Caml.Format.asprintf "%a" (pp ~start:true) x
    let add (a, x) (b, y) = a +. b, V.add x y
    let sub (a, x) (b, y) = a -. b, V.sub x y
    let mult (a, x) k = a *. k, V.mult x k
```

```
    let dot_product (a, x) (b, y) = (a *. b) +. V.dot_product x y
    let length_squared (a, b) = (a *. a) +. V.length_squared b
    let length a = Float.sqrt @@ length_squared a
    let of_vec a x : t = a, x
  end

  include T
  module Infix = MakeVecInfix (T)
end
```

Далее можно реализовывать модуля вектора для разных измерений индуктивно:

Листинг 5 – файл lib/vector.ml

```
module Vector0 = struct
  include VZ

  let make = VZ.zero
end

module Vector1 = struct
  include VS (Vector0)

  let make a = of_vec a Vector0.make
end

module Vector2 = struct
  include VS (Vector1)

  let make a b = of_vec a (Vector1.make b)
end

module Vector3 = struct
  include VS (Vector2)

  let make a b c = of_vec a (Vector2.make b c)
end

module Vector4 = struct
  include VS (Vector3)

  let make a1 a2 a3 a4 = of_vec a1 (Vector3.make a2 a3 a4)
end

module Vector5 = struct
  include VS (Vector4)

  let make a1 a2 a3 a4 a5 = of_vec a1 (Vector4.make a2 a3 a4 a5)
end
```

Листинг 6 – файл vector.mli

```
(** Degenerate vector (0 dimension) *)
module Vector0 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make] is single instance of 0-dimension degenerate vector *)
  val make : t
end

(** 1-dimension vector*)
module Vector1 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make x1] is 1-dimension vector with length [x1] *)
  val make : float -> t
end

(** 2-dimension vector *)
module Vector2 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make x1 x2] is 2-dimension vector with coords [x1] and [x2] *)
  val make : float -> float -> t
end

(** 3-dimension vector *)
module Vector3 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make x1 x2 x3] is 3-dimension vector with coords [x1], [x2] and [x3] *)
  val make : float -> float -> float -> t
end

(** 4-dimension vector *)
module Vector4 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make x1 x2 x3 x4] is 4-dimension vector with coords [x1], [x2], [x3] and [x4] *)
  val make : float -> float -> float -> float -> t
end

(** 5-dimension vector *)
module Vector5 : sig
  include VEC
  include VECINFIX with type t := t

  (** [make x1 x2 x3 x4 x5] is 4-dimension vector with coords [x1], [x2], [x3], [x4] and
```

```
      [x5] *)
  val make : float -> float -> float -> float -> float -> t
end
```

Напишем функцию, расширяющую возможности фреймворка Alcotest, добавляющую фикстуры и параметризацию тестов, а так же функции для тестирования чисел с плавающей точкой:

Листинг 7 – файл test/AlcotestExt.ml

```
open Alcotest

let fixtures_parameterized ~before ~after ~params ~param_to_string ~tests =
  tests
  |> List.concat_map ~f:(fun (name, mode, tst) ->
         List.map params ~f:(fun p ->
             let msg = name ^ " (" ^ param_to_string p ^ ")" in
             let fn () =
               let c = before () in
               tst c p;
               after c
             in
             msg, mode, fn))
;;

let check_float ?(eps = 1e-6) ~msg ~expected ~actual =
  check' bool ~msg ~expected:true ~actual:Float.(abs (expected - actual) <= eps)
;;
```

Напишем тесты для модуля двухмерного вектора:

Листинг 8 – файл test/vector2_test.ml

```
open Alcotest
open AlcotestExt
module Vector2 = Lab01_vector.Vector.Vector2
open CheckVector (Vector2)

module Add = struct
  type test_add_param =
    { msg : string
    ; vec1 : float * float
    ; vec2 : float * float
    ; sum : float * float
    ; eps : float
    }

  let test_add () { vec1 = v1x, v1y; vec2 = v2x, v2y; sum = sx, sy; eps } =
    (* Arrange *)
    let v1 = Vector2.make v1x v1y in
    let v2 = Vector2.make v2x v2y in
    (* Act *)
    let s_add = Vector2.add v1 v2 in
```

```
    let s_infix = Vector2.Infix.(v1 + v2) in
    (* Assert *)
    check_vector
      ~msg:"Sum through function"
      ~expected:(Vector2.make sx sy)
      ~actual:s_add
      ~eps;
    check_vector
      ~msg:"Sum through infix operator"
      ~expected:(Vector2.make sx sy)
      ~actual:s_infix
      ~eps
  ;;

  let tests =
    fixtures_parameterized
      ~before:Fn.id
      ~after:Fn.id
      ~params:
        [ { msg = "All zeros"; vec1 = 0., 0.; vec2 = 0., 0.; sum = 0., 0.; eps = 0. }
        ; { msg = "(1,2) + (3,4) = (4,6)"
          ; vec1 = 1., 2.
          ; vec2 = 3., 4.
          ; sum = 4., 6.
          ; eps = 1e-8
          }
        ]
      ~param_to_string:(fun { msg } -> msg)
      ~tests:[ "Vector sum", `Quick, test_add ]
  ;;
end

module Sub = struct
  type test_sub_param =
    { msg : string
    ; vec1 : float * float
    ; vec2 : float * float
    ; result : float * float
    ; eps : float
    }

  let test_sub () { vec1 = v1x, v1y; vec2 = v2x, v2y; result = sx, sy; eps } =
    (* Arrange *)
    let v1 = Vector2.make v1x v1y in
    let v2 = Vector2.make v2x v2y in
    (* Act *)
    let s_sub = Vector2.sub v1 v2 in
    let s_infix = Vector2.Infix.(v1 - v2) in
    (* Assert *)
    check_vector
      ~msg:"Subtraction through function"
      ~expected:(Vector2.make sx sy)
      ~actual:s_sub
```

```
      ~eps;
    check_vector
      ~msg:"Subtraction through infix operator"
      ~expected:(Vector2.make sx sy)
      ~actual:s_infix
      ~eps
  ;;

  let tests =
    fixtures_parameterized
      ~before:Fn.id
      ~after:Fn.id
      ~params:
        [ { msg = "All zeros"; vec1 = 0., 0.; vec2 = 0., 0.; result = 0., 0.; eps = 0. }
        ; { msg = "(1,2)-(3,4)=(-2,-2)"
          ; vec1 = 1., 2.
          ; vec2 = 3., 4.
          ; result = -2., -2.
          ; eps = 1e-8
          }
        ]
      ~param_to_string:(fun { msg } -> msg)
      ~tests:[ "Subtraction", `Quick, test_sub ]
  ;;
end

module Eq = struct
  type test_param_eq =
    { msg : string
    ; vec1 : float * float
    ; vec2 : float * float
    ; expected : bool
    ; eps : float
    }

  let test_eq () { vec1 = v1x, v1y; vec2 = v2x, v2y; expected; eps } =
    (* Arrange *)
    let v1 = Vector2.make v1x v1y in
    let v2 = Vector2.make v2x v2y in
    (* Act *)
    let s_sub = Vector2.equal v1 v2 ~eps in
    if Float.(eps = 0.)
    then begin
      let s_infix = Vector2.Infix.(v1 = v2) in
      check' bool ~msg:"Equal through infix operator" ~expected ~actual:s_infix
    end;
    (* Assert *)
    check' bool ~msg:"Equal through function" ~expected:s_sub ~actual:s_sub
  ;;

  let tests =
    fixtures_parameterized
      ~before:Fn.id
```

```
      ~after:Fn.id
      ~params:
        [ { msg = "All zeros"; vec1 = 0., 0.; vec2 = 0., 0.; expected = true; eps = 0. }
        ; { msg = "(1,2)-(3,4)=(-2,-2)"
          ; vec1 = 1., 2.
          ; vec2 = 3., 4.
          ; expected = false
          ; eps = 1e-8
          }
        ]
      ~param_to_string:(fun { msg } -> msg)
      ~tests:[ "Equality", `Quick, test_eq ]
  ;;
end

module Length = struct
  type test_param_length =
    { msg : string
    ; vec : float * float
    ; expected : float
    ; expected_squared : float
    ; eps : float
    }

  let test_length () { vec = vx, vy; expected; expected_squared; eps } =
    (* Arrange *)
    let v = Vector2.make vx vy in
    (* Act *)
    let actual = Vector2.length v in
    let actual_squared = Vector2.length_squared v in
    (* Assert *)
    check_float ~eps ~msg:"Check length" ~expected ~actual;
    check_float
      ~eps
      ~msg:"Check squared length"
      ~expected:expected_squared
      ~actual:actual_squared
  ;;

  let tests =
    fixtures_parameterized
      ~before:Fn.id
      ~after:Fn.id
      ~params:
        [ { msg = "All zeros"
          ; vec = 0., 0.
          ; expected = 0.
          ; expected_squared = 0.
          ; eps = 0.
          }
        ; { msg = "len(2,0) = 2"
          ; vec = 2., 0.
          ; expected = 2.
```

```
          ; expected_squared = 4.
          ; eps = 1e-8
          }
        ]
      ~param_to_string:(fun { msg } -> msg)
      ~tests:[ "Length", `Quick, test_length ]
  ;;
end

let suite =
  ( "2D Vector tests"
  , Add.tests |> List.append Sub.tests |> List.append Eq.tests |> List.append
Length.tests
  )
;;
```

Запустим тесты:

Листинг 9 – запуск тестов

```
vladislav@DESKTOP-NK6MA9B:~/Projects/software-testing/Lab01-vector$ make test
opam exec -- dune runtest --root .
lab01_vector_test alias test/runtest
Testing `lab01-vector'.
This run has ID `PNW2JH44'.

  [OK]          2D Vector tests          0   Length (All zeros).
  [OK]          2D Vector tests          1   Length (len(2,0) = 2).
  [OK]          2D Vector tests          2   Equality (All zeros).
  [OK]          2D Vector tests          3   Equality ((1,2)-(3,4)=(-2,-2)).
  [OK]          2D Vector tests          4   Subtraction (All zeros).
  [OK]          2D Vector tests          5   Subtraction ((1,2)-(3,4)=(-2,-2)).
  [OK]          2D Vector tests          6   Vector sum (All zeros).
  [OK]          2D Vector tests          7   Vector sum ((1,2) + (3,4) = (4,6)).

Full test results in `~/Projects/software-testing/Lab01-
vector/_build/default/test/_build/_tests/lab01-vector'.
Test Successful in 0.001s. 8 tests run.
```

Попробуем сделать ошибку в реализации так, чтобы тест упал (сломаем реализацию суммы векторов):

Листинг 9 – файл lib/vector.ml

```
----let add (a, x) (b, y) = a +. b, V.add x y
++++let add (a, x) (b, y) = a +. a, V.add x y
```

Листинг 10 – запуск тестов

```
vladislav@DESKTOP-NK6MA9B:~/Projects/software-testing/Lab01-vector$ make test
opam exec -- dune runtest --root .
lab01_vector_test alias test/runtest (exit 1)
(cd _build/default/test && ./lab01_vector_test.exe)
Testing `lab01-vector'.
This run has ID `0KAHJBI4'.

  [OK]          2D Vector tests          0   Length (All zeros).
  [OK]          2D Vector tests          1   Length (len(2,0) = 2).
  [OK]          2D Vector tests          2   Equality (All zeros).
  [OK]          2D Vector tests          3   Equality ((1,2)-(3,4)=(-2,-2)).
  [OK]          2D Vector tests          4   Subtraction (All zeros).
  [OK]          2D Vector tests          5   Subtraction ((1,2)-(3,4)=(-2,-2)).
  [OK]          2D Vector tests          6   Vector sum (All zeros).
> [FAIL]        2D Vector tests          7   Vector sum ((1,2) + (3,4) = (4,6)).

┌──────────────────────────────────────────────────────────────────────────────┐
│ [FAIL]        2D Vector tests          7   Vector sum ((1,2) + (3,4) = (... │
└──────────────────────────────────────────────────────────────────────────────┘
ASSERT Sum through function
FAIL Sum through function

   Expected: `true'
   Received: `false'

Raised at Alcotest_engine__Test.check in file "src/alcotest-engine/test.ml",
line 196, characters 4-261
Called from Dune__exe__Vector2_test.Add.test_add in file "test/vector2_test.ml",
line 23, characters 4-118
Called from Dune__exe__AlcotestExt.fixtures_parameterized.(fun).fn in file
"test/AlcotestExt.ml", line 10, characters 15-22
Called from Alcotest_engine__Core.Make.protect_test.(fun) in file "src/alcotest-
engine/core.ml", line 180, characters 17-23
Called from Alcotest_engine__Monad.Identity.catch in file "src/alcotest-
engine/monad.ml", line 24, characters 31-35

Logs saved to `~/Projects/software-testing/Lab01-
vector/_build/default/test/_build/_tests/lab01-vector/2D Vector
tests.007.output'.
──────────────────────────────────────────────────────────────────────────────

Full test results in `~/Projects/software-testing/Lab01-
vector/_build/default/test/_build/_tests/lab01-vector'.
1 failure! in 0.001s. 8 tests run.
make: *** [Makefile:42: test] Error 1
```

4 Вывод

В данной работе мы ознакомились с основами блочного тестирования на языке OCaml с применением фреймворка Alcotest. Исходный код доступен на GitHub.