Assignment 3

SFWR ENG 2AA4

Files due Feb 23, E-mail partner due Feb 24, Lab report due Mar 2

The purpose of this software design exercise is to write an OCaml program that creates, uses, and tests an ADTs for vector spaces and inner product spaces. We will take advantage of the fact that OCaml treats funtions as first class members and make our vector space the space of real continuous functions.

Step 1

Write a module that creates a vector ADT, where the vector is in the space of functions. It should consist of an OCaml code file named `vectorT.ml`. The specification for this module (Vector Module) is given at the end of the assignment. The specification includes output of a sequence of real. Please use a list (not an array) to represent the sequence.

Step 2

Write a module that creates a vector space ADT. It should consist of an OCaml file named `vectorSpaceT.ml`. The new module should follow the specification (Vector Space Module) given at the end of the assignment.

Step 3

Write a module that creates an inner product space ADT. It should consist of an OCaml file named `innerProductSpaceT.ml`. The new module should follow the specification (Inner Product Space Module) given at the end of the assignment.

Step 4

Write testing modules to test the above files. The modules should use oUnit for the testing. Each procedure should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your lab report.) Please make an effort to test normal cases, boundary cases, and exception cases. The files that are required are as follows:

- `test_vectorT.ml` to test `vectorT.ml`
- `test_vectorSpaceT.ml` to test `vectorSpaceT.ml`
- `test_innerProductSpaceT.ml` to test `innerProductSpaceT.ml`
- `test_assig3.ml` to test all of the above using a test suite.

Write a makefile `Makefile` to build the executable `test_assig3`.

Step 5

Submit the files `vectorT.ml`, `vectorSpaceT.ml`, `innerProductSpaceT.ml`, `test_vectorT.ml`, `test_vectorSpaceT.ml`, `test_innerProductSpaceT.ml`, `test_assig3.ml` and `Makefile` using subversion. This must be completed no later than midnight of the deadline for file submission.

E-mail the `innerProductSpaceT.ml` file to your assigned partner. (Partner assignments will be posted on WebCT, on the day after the initial submission.) Your partner will likewise e-mail you his or her files. These e-mails should be traded by midnight of the day following the file submission.

Step 6

After you have received your partner's files, replace your corresponding files with your partner's. Do not make any modifications to any of the code. Run your tests and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results.

Step 7

Write a report that includes the following:

1. **Your userid on the first page**
2. Your name and student number.
3. Your partner's `innerProductSpaceT.ml` file.
4. The results of testing your files (along with the rational for test case selection).
5. The results of testing your files combined with your partner's files.
6. A discussion of the test results and what you learned doing the exercise. List any problems you found with (a) your program, (b) your partner's module, and (c) the specification of the modules.
7. Complete the specification for the unspecified access program (length) in the Inner Product Space module. You are not required to implement this access program.
8. Complete the specification of the Subspace module provided at the end of the assignment. You are not required to implement or test this module. The parts that you are to complete are indicated in bold.
9. Draw a graph of the uses relationship between the Vector, Vector Space, Inner Product Space and Subspace modules. Each module should be represented by a square. A directed arrow should be drawn between modules A and B if module A uses module B.
10. A copy of the part of your log book relevant to this lab exercise.

A physical copy of the lab report is due at the beginning of the lecture on the assigned due date.

Notes

1. Place all submitted files in your svn repository in the folder `Assig3`.
2. Please put your name and student number at the top of each of your source files. (You should remove the student number before e-mailing any files to your partner.)
3. Your program must work in the ITB labs on moore when compiled by ocamlc.

4. If your partner fails to provide you with a copy of his or her files by the deadline, please tell the instructor via e-mail as soon as possible.

5. If you do not send your files to your partner by the deadline, you will be assessed a **10% penalty** to your assignment grade.

6. For the OCaml implementation of the modules, you will need to "map" the MIS syntax to OCaml syntax. In particular, when the input to an access program consists of several parameters, you should provide each parameter separately, as opposed to combining them in a tuple. That is, if function f has two arguments, the type of f is $A \rightarrow (B \rightarrow C)$, not $A \times B \rightarrow C$. A concrete example, in OCaml syntax, is the constructor for pointT. Please use

   `class pointT xc yc = ...` as opposed to

   `class pointT (xc ,yc) = ....`

7. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**

8. Your code will be tested with the ocamlc compiler. We will not use ocamlopt to test your program.

9. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes.

Vector Module

Template Module

vectorT

Uses

N/A

Syntax

Exported Types

vectorT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new vectorT	real → real	vectorT	
getf		real → real	
eval	real, real, integer	sequence of real	DEL_NEG, NSTEP_NEG
evalPrint	real, real, integer		DEL_NEG, NSTEP_NEG
equal	vectorT, real, real, integer, real	boolean	DEL_NEG, NSTEP_NEG

Semantics

Environment Variables

screen : two dimensional sequence of positions on the screen, with each position holding a character

State Variables

f: real → real

State Invariant

None

Assumptions

None.

Access Routine Semantics

new vectorT (fin):

- transition: $f := fin$
- output: $out := self$
- exception: none

getf:

- output: $out := f$
- exception: none

eval ($startx, deltax, nsteps$):

- output: $out :=< f(startx), f(startx + deltax), f(startx + 2 \cdot deltax), ..., f(startx + nsteps \cdot deltax) >$
- exception: $exc := ((deltax < 0) \Rightarrow \text{DEL\_NEG} | (nsteps < 0) \Rightarrow \text{NSTEP\_NEG})$

evalPrint ($startx, deltax, nsteps$):

- transition: The state of *screen* is modified so that the sequence returned by eval ($startx$, $deltax$, $nsteps$) is displayed.
- exception: $exc := ((deltax < 0) \Rightarrow \text{DEL\_NEG} | (nsteps < 0) \Rightarrow \text{NSTEP\_NEG})$

equal (g, $startx, deltax, nsteps, epsilon$):

- output:
$$out := \forall(i : \mathbb{N} | 0 \leq i \leq nsteps : |s_f[i] - s_g[i]| < epsilon)$$
where $s_f = self.eval(startx, deltax, nsteps)$ and $s_g = g.eval(startx, deltax, nsteps)$
- exception: $exc := ((deltax < 0) \Rightarrow \text{DEL\_NEG} | (nsteps < 0) \Rightarrow \text{NSTEP\_NEG})$

Vector Space Module

Template Module

vectorSpaceT

Uses

vectorT

Syntax

Exported Types

vectorSpaceT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new vectorSpaceT		vectorSpaceT	
add	vectorT, vectorT	vectorT	
scalarMult	real, vectorT	vectorT	

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

new vectorSpaceT():

- output: *out* := *self*

- exception: none

add(v_1, v_2):

- output: *out* := new vectorT($v_1 + v_2$)
- exception: none

scalarMult(k, v)

- output: *out* := new vectorT(kv)
- exception: none

Inner Product Space Module

Template Module

innerProductSpaceT **inherits** vectorSpaceT

Uses

vectorT, vectorSpaceT

Syntax

Exported Types

innerProductSpaceT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new innerProductSpaceT	real, real	innerProductSpaceT	
dotprod	vectorT, vectorT	real	
isOrthog	vectorT, vectorT	boolean	
length	vectorT	real	

Semantics

State Variables

a: real
b: real

State Invariant

None

Assumptions

None

Access Routine Semantics

new vectorSpaceT ($lower, upper$):

- transition: $a, b := lower, upper$
- output: $out := self$
- exception: none

dotprod(v_1, v_2):

- output: $out := \int_a^b v_1(x)v_2(x)dx$
- exception: none

isOrthog(v_1, v_2):

- output: $out :=$ |dotprod(v_1, v_2)| $<$ error
- exception: none

length(v):

- This function returns the length of the vector v. **Please specify the output.**
- output: $out :=$?
- exception: none

Local Constants

error $= 1 \times 10^{-4}$

Considerations

The values of a and b will normally be either -1 and 1 or 0 and 1, respectively.

Subspace Module

Uses

? **Please specify uses**

Syntax

Exported Types

subSpaceT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new subSpaceT	vectorT, vectorT, vectorT	subSpaceT	
linearCombo	real, real, real	vectorT	
isOrthonormal		boolean	

Semantics

State Variables

The basis vectors for the subspace:
b_1: vectorT
b_2: vectorT
b_3: vectorT

State Invariant

None

Assumptions

None

Access Routine Semantics

new subSpaceT(v_1, v_2, v_3):

- This access program is the constructor for the subspace.
- transition: **Please specify**

- ouput: **Please specify**
- exception: none

linearCombo(c_1, c_2, c_3):

- This function returns the linear combination of the basis vectors b_1, b_2 and b_3 using the coefficients c_1, c_2 and c_3, respectively. **Please specify the output of this function.**
- output: *out* :=?
- exception: none

isOrthonormal():

- This function returns true if all of the vectors in the basis (b_1, b_2 and b_3) have length 1 and all of the vectors are orthogonal to one another. **Please specify the output of this function.** You may find it helpful to introduce your own local functions.
- output: *out* :=?
- exception: none