Kopi av Front page

NTNU
Faculty of Information Technology
and Electrical Engineering
Department of Computer Science

Retake Midterm Test for **TDT4165 Programming Languages**

This is not an exam, but a test counting towards a composite score.

Academic contact during examination: *Øystein Nytrø*
Phone: *+47 91897606,* Email: *nytroe@ntnu.no*

Examination date: **January 20, 2021**
Examination time (from-to): **17.00-19.00**
Permitted examination support material: **Code A: All support material allowed**

Cheating/Plagiarism: The test is an individual, independent work. Examination aids are permitted, but make sure you follow any instructions regarding citations. During the exam it is not permitted to communicate with others about the exam questions, or distribute drafts for solutions. Such communication is regarded as cheating. All submitted answers will be subject to plagiarism control. Read more about cheating and plagiarism here.

This course has only an english version of the test.
This test has 10 tasks. All tasks have the same weight (10p).
Some tasks require program or prose writing.
The tasks are in no particular order wrt. curriculum.
Wrong answers are not scored negatively.

There is an ungraded text-entry at the end of the test that you can use for comments.

Students will find the results (a score 0-100) in Studentweb and Blackboard after scoring has been completed. Please contact the department if you have questions about your results. This midterm counts 33% towards a final score that will be tranformed to a final grade for the entire course. *If you have already taken a midterm test in 2020, the best score will be used.*

If by some reason there is a need to inform all students about important information during the test, Inspera notifications will be used. A dialogue box will appear, and the notification can also be accessed by clicking a bell icon in Inspera.

1 Declarative programming

Which of completions to "A declarative program ..." makes the sentence **true?**
Select one alternative:

may be definitionally declarative when written in the declarative sequential kernel language (a subset of Oz) even when extended with exceptions.

must be either descriptional or observational, but not both.

can not be written in an imperative programming language.

can only use functional programming constructs in Oz.

in Oz requires "declare"-sentences.

Maximum marks: 10

2 Grammar comprehension

Consider the following grammar for a programming language not unlike Oz:

```
<s> ::= skip
    | <s> <s>
    | local <x> in <s> end
    | <x> = <x>
    | <x> = <v>
    | if <x> then <s> else <s> end
    | if <x> then <s> end
```

<s> is a statement, and is also the start symbol.
<x> is an identifier, as in Oz.
<v> is a value expression, as in Oz.

Which alternative is **not possible to** generate by the grammar?
Select one alternative:

local A in local B in A=A B=B if A then A=B else B=A end end

local A in if A then A=A end A=A end

local A in if A then A=A A=A end end

local A in local B in A=B B=A if A then 3 else 0 end if A then B end end end

local A in if B then B=A end end

Maximum marks: 10

3 Grammar properties

Consider the following grammar:

```
<s> ::= skip
    | <s> <s>
    | local <x> in <s> end
    | <x> = <x>
    | <x> = <v>
    | if <x> then <s> else <s> end
    | if <x> then <s> end
```

<s> is a statement, and is also the start symbol.
<x> is an identifier, as in Oz.
<v> is a value expression, as in Oz.

Which one completion of "The grammar is ..." makes the sentence **true?**
Select one alternative:

- not context free
- factorized
- regular
- ambiguous.
- not recursive

Maximum marks: 10

4 Identifier scopes in Oz

Which completion of "Oz has..." is true?
Select one alternative:

- dynamic scoping.
- call-by-reference
- static scoping. ‘Static scoping’ also counts as correct. ✔
- no global variables
- dynamic typing.

Maximum marks: 10

5 Higher order Programming

1) If you run the following Oz program, what is shown by Browse?
2) What does A do?
3) Show that you can implement A using eg. Map or Fold.

```
local A P in

   fun {P X Y}
      X * Y
   end


   fun {A L BO I}
      case L
      of nil then I
      [] H|T then {BO H {A T BO I}}
      end
   end

   {Browse {A [2 3 4] P 1}}
end
```

Fill in your answer here

```
1) The P implements a binary multiplication function.
   The A function implements a RightFold-like list traversal.
   Browse displays 24.

2) A applies P to the first element of a list and the
   result of applying A on the rest of the list.

3) FoldR is the same as A:

local A P FoldRight in

   fun {FoldRight List Null Transform Combine}
      case List of nil then Null
      [] Head|Tail then {Combine {Transform Head}
                                 {FoldRight Tail Null Transform Combine}}
      end
   end

   fun {P X Y}
      X * Y
   end

   fun {A L BO I}
      case L
      of nil then I
      [] H|T then {BO H {A T BO I}}
      end
   end

   fun {Id X} X end

   {Browse {A [2 3 4 5] P 1}}
   {Browse {FoldRight [2 3 4 5] 1 fun {$ X} X end P} }
end

A = fun {$ L BO I} {FoldR L BO I} end
```

Words: 0

Maximum marks: 10

6 Difference list programming

In the textbook, Difference lists are explained like:

3.4.4 Difference lists

A difference list is a pair of two lists, each of which might have an unbound tail. The two lists have a special relationship: it must be possible to get the second list from the first by removing zero or more elements from the front. Here are some examples:

```
X#X                % Represents the empty list
nil#nil            % idem
[a]#[a]            % idem
(a|b|c|X)#X        % Represents [a b c]
(a|b|c|d|X)#(d|X)  % idem
[a b c d]#[d]      % idem
```

A difference list is a representation of a standard list. We will talk of the difference list sometimes as a data structure by itself, and sometimes as representing a standard list. Be careful not to confuse these two viewpoints. The difference list `[a b c d]#[d]` might contain the lists `[a b c d]` and `[d]`, but it represents neither of these. It represents the list `[a b c]`.

Define the function {ReverseD DL} which reverses a difference list.

```
1
```

```
local ReverseD X in
   fun {ReverseD Xs}
      proc {ReverseDP Xs ?Y1 Y}

case Xs of nil then Y1=Y

[] X|Xr then {ReverseDP Xr Y1 X|Y}

end
      end Y1
   in {ReverseDP Xs Y1 nil}
      Y1
   end

   {Browse {ReverseD 1} % (1|2|3|4|X)#X}}
end
```

Maximum marks: 10

7 List program

1) Define an Oz function that removes the last N elements of a list, and returns the resulting potentially shortened list.

2) Explain how the program handles values of N that is outside the bounds given by the length of the input list.

3) Add an example applying the function and show the result.

4) Comment on the computational efficiency of your solution.

Fill in your program and example application here

```
All sorts of solutions!
Using Filter, Generate, Reverse, Length Append with
different assumptions about their implementation.
Many trying to optimize traversal with accumulators
and counting. Cfr. CTMCP 3.4.3

Criteria:

1. Border case management

2. Removing right end of list

3. Sensible discussion

Score 3,2,2,3 per part

Simple and straightforward example
% 1)

fun {Length L N}
    case L of _|Tail then {Length Tail N+1}
    else N
    end
end

fun {Take L N}
    if N>0 then
        case L of Head|Tail then Head|{Take Tail N-1}
        else nil
        end
    else nil
    end
end

fun {RemoveLast L N}
    {Take L {Length L 0} - N}
end

% 2) If N is less than 0, or larger than the length of the list, nil is returned
% 3)

{RemoveLast [1 2 3 4] 2}
% returns : [1 2]

% 4) The solution is O(n) (where n is the length of the input list) in time,
%  and O(1) in space because it allows tail-call optimization.
%  It does iterate the list twice, first to determine the length, and then to
%  remove the last elements. This is the best approach I could come up with that
%  allowed for tail-call optimization. Without tail-call optimization, the
%  the algorithm would've been O(n) in space as well.
```

Maximum marks: 10

8 Oz abstract machine

The following program in Oz is given:

```
17 local X P Q in X=1
18         P = proc {$} {Browse X} end
19         Q = proc {$ X} local X in X = 2 {P} end end
20         {Q X}
21 end
```

a) What is the state of the abstract Oz machine just before execution of line 20? (5p)
Fill in your answer here

```
Answer in following two pages
```

b) What is the state of the abstract Oz machine after the statement in line 20? (5p)

Fill in your answer here

```
Answer in following two pages
```

You do not need to show variables on the stack or SAS not reachable/relevant. Simplify as needed.

What is eventually printed in the browser window?

Maximum marks: 10

8 a)

$$
\begin{array}{l}
(\ [\ (\{\texttt{Q}\ \texttt{X}\}, \{\texttt{Q} \rightarrow \nu_3, \texttt{X} \rightarrow \nu_1\})\ ], \\
\quad \{\ \nu_1 = 1, \nu_2 = (\texttt{proc}\ \{\$\}\ \{\texttt{Browse X}\}\ \texttt{end}, \{\texttt{X} \rightarrow \nu_1\}), \\
\qquad \nu_3 = (\texttt{proc}\ \{\$\ \texttt{X}\}\ \texttt{local X in X = 2}\ \{\texttt{P}\}\ \texttt{end end}, \{\texttt{P} \rightarrow \nu_2\})\ \}\ )
\end{array}
$$

8 b)

(a) ([({Q X}, $\{$X $\rightarrow \nu_1$, P $\rightarrow \nu_2$, Q $\rightarrow \nu_3\}$)],
$\{ \nu_1 = 1, \nu_2 =$ (proc {$} {Browse X} end, $\{$X $\rightarrow \nu_1\}$),
$\nu_3 =$ (proc {$ X} local X in X = 2 {P} end end, $\{$P $\rightarrow \nu_2\}$) })

(b) ([(local X in X = 2 {P} end, $\{$X $\rightarrow \nu_1$, P $\rightarrow \nu_2\}$)],
$\{ \nu_1 = 1, \nu_2 =$ (proc {$} {Browse X} end, $\{$X $\rightarrow \nu_1\}$) })

(c) ([(X = 2 {P}, $\{$X $\rightarrow \nu_4$, P $\rightarrow \nu_2\}$)],
$\{\nu_1 = 1, \nu_2 =$ (proc {$} {Browse X} end, $\{$X $\rightarrow \nu_1\}), \nu_4\}$)

(d) ([(X = 2, $\{$X $\rightarrow \nu_4$, P $\rightarrow \nu_2\}$), ({P}, $\{$X $\rightarrow \nu_4$, P $\rightarrow \nu_2\}$)],
$\{\nu_1 = 1, \nu_2 =$ (proc {$} {Browse X} end, $\{$X $\rightarrow \nu_1\}), \nu_4\}$)

(e) ([({P}, $\{$X $\rightarrow \nu_4$, P $\rightarrow \nu_2\}$)],
$\{\nu_1 = 1, \nu_2 =$ (proc {$} {Browse X} end, $\{$X $\rightarrow \nu_1\}), \nu_4 = 2\}$)

(f) ([({Browse X}, $\{$X $\rightarrow \nu_1\}$)],
$\{\nu_1 = 1\}$)

(g) ([ ],
{ })

In the browser window, 1 will be printed.

9 If-semantics Prose

Explain the semantics of the if-statements in the program below. What is the difference between usage in a proc and a fun body?

```
local P F B in
   P = proc {$ A B} if A then B = 1 end end
   F = fun {$ A B} if A then B end end
   {P false B}
   {Browse {F false B}}
end
```

Fill in your answer here

Very few actually looking at the code presented.

In a function body, the last sentence must be an expression yielding a return value.
The if-sentence can either be an expression or a statment, depending on whether
the clauses are expressions or statements. An if-sentence, as an expression, not containing an 'else'-
clause will potentially give rise to
a return expression not being available, and the function value not defined.
A Browse (or any other thread) that executes a statement that gives rise to an error will terminate
with no message in the browser, so the
error raised has to be caught in order to be recognized:

```
local F = fun {$ A B} if A then B end end in
   try  {Browse {F false 1}}
   catch X then {Browse X} end
end

local Y F = fun {$ A B} if A then B end end in
   {Browse 'before'}
   try Y = {F false 2} catch Z then {Browse Y#Z} end
   {Browse 'midway'}
   try  {Browse {F false 1}}
   catch X then {Browse X} end
   {Browse 'end'}
end
```

In CTMCP 2.6, the fun linguistic abstraction and if-statement is discussed. "An expression is
syntactic sugar for a sequence of operations that returns av value."
Interestingly, the Oz parser will not detect or complain about a missing 'else' in a function because
of the if being syntactic sugar.
(A fun-definition is a linguistic abstraction, translated into a proc-definition, so it is not
syntactic sugar.)
if a "try-catch" enlcoses an application of a else-less function, the missing else-clause will raise
an error.

Words: 0

Scoring:
- identifying expression vs statement
- identifying the difference in the semantics of a missing "else" for expressions vs statements.
- identifying that a function body, with undefined domain (ie the missing else) will throw an error-
exception.

Maximum marks: 10

10 Unification

Match the values that are unifiable in Oz

	[[a b c].2]	'|'(a '|'(b '|'(c nil)))	[_]	[X X]	a | X | X | nil
[B b c]					
X | X					
[X b b]					
1 | 1 | nil					
b | c | nil					

Maximum marks: 10

```
- [X b b] and a | X | X | nil is automatically marked as a correct match, which is wrong scoring.
- b | c | nil and [[a b c].2] is automatically marked as a correct match, which is wrong scoring.

That scoring cannot be changed in Inspera, so cannot be undone as part of evaluation.

However, not matching them is actually correct,
so non-matching of  exactly these two, ie. only 3 identified matches,
are scored as correct.
```

11 Kopi av Comments

This section is not marked or graded, leave comments or clarifications if needed here.

Maximum marks: 0