CSC488S 2015/2016 Final Exam Solution and Comments

1. [15 marks]

a) Need semantic checks:

– RHS and LHS arrays are the same type (integer or boolean)

– RHS and LHS arrays are the same dimensionality (1d or 2d)

– RHS and LHS arrays have the same lower and upper bounds in each dimension

b) Need a loop to copy array elements one by one from LHS to RHS.

	Code	Stack
	PUSH 0	index
LOOP:	DUP	index , index
	DUP	index , index , index
	ADDR LHS	@LHS , index , index , index
	ADD	@LHS+index , index , index
	SWAP	index , @LHS+index , index
	ADDR RHS	@RHS , index , @LHS+index , index
	ADD	@RHS+index , @LHS+index , index
	LOAD	value(@RHS+index) , @LHS+index , index
	STORE	index
	PUSH 1	1 , index
	ADD	index+1
	DUP	index+1 , index+1
	PUSH RHS.size	RHS.size , index+1 , index+1
	EQ	RHS.size = index+1 , index+1
	PUSH LOOP	@LOOP , RHS.size = index+1 , index+1
	BF	index+1
	POP	

Since this loop depends only on the size of the array, it will work for both one and two dimensional arrays.

2. [15 marks] This proposed hardware change will require major revisions to the addressing strategy in the compiler. Two problems:

1) The 12-bit offset only allows 4096 words of memory to be addressed in the main program or in any procedure or function.

2) Some code generation designs use a negative offset to access parameters and the return address. This will no longer work.

Possible solutions:
a) add a language restriction on the maximum size of data in the main program or any procedure or function. This is a cop-out (cf. Java).

b) Modify the compiler generated addressing code to manage data that is not within the 4096 word addressability limit.
Given

```
procedure P {
    var A[5000] : integer          % offset 10
    var B : boolean            % offset 5010
}
```

For any variable (e.g. A) with an offset within the 4096 limit, use the existing addressing code. For any variable (e.g. B) above that limit generate extra code to calculate the correct address. For an access to B in the example above something like:

```
ADDR LL,0          % display for level LL
PUSH 5010          % offset of B in activation record
ADD                % address of B now on stack
```

A really clever compiler would layout activation records to minimize the amount of extra addressing code required, e.g. put B before A in the example above and no extra code would be required.

3. [20 marks] Optimizations applied: - constant folding, code motion, strength reduction, common subexpression elimination, algebraic simplification.

```
 1      var a array 1 .. 20 , -15 .. 15 of float
 2      var b array 1 .. 400 of float
 3      var J , K : integer
        ... % Assume variables are initialized here
20      for J := 1 to 20 do
            J8020 := 80*J - 20
            J6464 := 64*J - 64
21      %       a[ J , J ] := 0
        &A[1,-15] + J8020 + 4*J := 0.0
22      %       for K := -15 to 15 do
            for K4 := -60 to 60 do
                JK4 := J8020 + K4
                AJKP := &A[1,-15] + JK4
                AJK := @AJKP + JK4
23      %           b[ 16 * J + K - 15 ] := a[ J , K ]
                &B[1]+J6464+K4 := AJK
24      %           a[ J , K ] := a[ J , K ]**2 - 3.0 * a[ J , K ] + 1.0
                @AJKP := AJK * (AJK - 3.0 ) +1.0
25          end for
26      end for
```

Some solutions did loop unrolling but that wasn't necessary for full marks.

4. [5 marks]

The optimized subscript calculation was developed for well-behaved 0-origin or 1-origin arrays. It breaks down when the user can specify arbitrary lower and upper bounds (as in the course project). Two separate problems:

1) For some arrays, the calculation: $OFFSET_B - ((ub_1 - lb_1 + 1) * lb_1 + lb_2)$
Could overflow. For example: A[1 .. 256 , 32760 .. 32767]

2) For some arrays, the MUL on line 4 of the subscript calculation could result in an overflow for valid subscripts. For example B[16384 .. 17000 , 16384 .. 17000].

Many students assumed this question had something to do with dynamically allocated arrays, which it did not.
Since these problems are a function of the bounds of the array a really clever compiler could use the *range analysis* techniques describe in lecture once at the point of array declaration to set a symbol table flag *safeToOptimizeSubscripts*.

5. [10 marks] The main issue here is how to delete very large comments *efficiently*. A regular expression scanner really isn't good enough.
Best strategy would be to recognize the start of a comment '/*' and then go into a very tight character munching loop until the matching '*/' is found.
Many solutions tried to deal with nested block comments which was not a part of this questions. Perhaps a leftover from a previous final exam solution.

6. [15 marks]
Corrected statement:

R := P **or not** (Q ? **not** R **and not** P : **not** (P **and** (Q **or not** R)))

```
 1   (   branch , P     ,  trueExit  , T2          )
 2   (   branch , Q     , T3         , T5          )
 3   (   branch , R     ,  falseExit , T4          )
 4   (   branch , P     ,  falseExit ,  trueExit   )
 5   (   branch , P     , T6         ,  trueExit   )
 6   (   branch , Q     ,  falseExit , T7          )
 7   (   branch , R     ,  trueExit  ,  falseExit  )
     (              ,
 8   (   assign , R     , true       , ??          )
 9   (   branch , true  , T11        , ??          )
10   (   assign , R     , false      , ??          )
11   (              ,
```

7. [20 marks]

Line(s)	Semantic Checks
1	Check that shell has not been previously declared
	Check that a has not be previously declared as a parameter
2	Check that increment has not been previously declared in this scope
2,4,6,8,11	Every where that a is used
	Check that a is declared, visible and accessible at the point of use
	Check that a is a one dimensional array
	Check that the subscript for a is a valid integer expression
2,4	Check that a.length is valid
3,4,7,8,9,13.14,16	Everywhere that increment is used
	Check that increment is declared, visible and accessible at point of use
2,4,5,6,8,9,11,14.16	For every assignment operation
	Check the the LHS is a variable
	Check that the RHS is the correct type to assign to the LHS variable
3,4,7,13	For every comparison operation
	Check that the left and right operands are the same type
	Check that the left and right operands are legal for the compare operator
4	Check that i hasn't been previously declared
	Check that i++ is a valid operation on i
5	Check that k hasn't been previously declared
6	Check that temp hasn't been previously declared
7,8,9,11	Everywhere that k is used
	Check that k is declared, visible and accessible at the point of use
7,11	Everywhere that temp is used
	Check that temp is declared, visible and accessible at the point of use
2,7,8,9,16	For every arithmetic operator (+ , - , *)
	Check that the left and right operands are legal for the arithmetic operator

8.[20 marks]

```
boolean function expect( token ) {
      return nextToken = token
         and getToken()
}

boolean function procedureDeclaration() {
     asset ( expect( procedureToken ))
      return expect( identifierToken )
         and Signature()
         and expect( equalToken )
         and Block()
         and expect( identifierToken )
}

boolean function Signature() {
      return FormalParameterList()
         and optType
}

boolean function FormalParameterList() {
      if  not expect( leftParenToken )  then
         return  false
      if FormalParameter()  then
         while expect( semiColonToken )  do
            if  not FormalParameter()  then
               return  false
      return expect( rightParenToken )
}
```

```
boolean function FormalParameter() {
      return AccessType()
         and IdentifierList()
         and expect( colonToken )
         and Type()
         and optInitialzer()
}

boolean function AccessType() {
      return expect( valueToken )
         or expect( varToken )
         or expect( readonlyToken )
         or  true
}

boolean function optInitializer() {
      return
        ( expect( colonToken )
            and expect( equalToken )
            and ConstantExpression()
        )
      or
         true
}

boolean function optType() {
      return
        ( expect( colonToken )
            and Type()
        )
      or
         true
}
```

A lot of solutions didn't get the syntax right. Either allowing too much or not enforcing all the constraints in the grammar.