# THE UNIVERSITY OF AUCKLAND

**First Semester, 2004**

**City Campus**

**Computer Science**

**Language Implementation**

(Time allowed TWO hours)

| |
|---|
| FAMILY NAME: |
| PERSONAL NAMES: |
| STUDENT ID NUMBER: |
| LOGIN NAME: |
| SIGNATURE: |

This Examination is out of 100 Marks. Attempt **ALL** questions. Write your answers in the spaces provided in this question and answer booklet. Do not remove the staples from the question and answer booklet. However, you may detach and remove the staples from the appendices.

| | | |
|---|---|---|
| 1 | | 24 |
| 2 | | 16 |
| 3 | | 13 |
| 4 | | 16 |
| 5 | | 16 |
| 6 | | 15 |
| Total | | 100 |

Continued ...

Print Name _____

## 1. Bottom Up LALR(1) Parsing [24 Marks]

Consider the CUP grammar in the **Appendix For Question 1**. Note that the rules for `RHSList` are left recursive, while the rules for `SymbolList` are right recursive.

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
Expr ::= MINUS Expr:expr | INTCONST:value ;
```

Assume "`Expr` ", "`MINUS` ", "`expr` ", "`INTCONST`", and "`value`" match `IDENT`, and "`::=`", "`|`" and "`;`" match `EXPANDSTO`, `OR` and `SEMICOLON`, respectively.

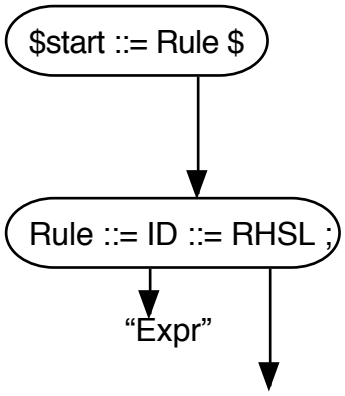
| Stack | | | | | | | | Token | Action | |
|---|---|---|---|---|---|---|---|---|---|---|
| $0 | | | | | | | | ID Expr | Shift | 2 |
| $0 | ID 2 | | | | | | | ::= | Shift | 3 |
| $0 | ID 2 | ::= 3 | | | | | | ID MINUS | Shift | 7 |
| $0 | ID 2 | ::= 3 | ID 7 | | | | | ID Expr | Reduce | Sym ::= ID |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | | | | | | | | |
| $0 | ID 2 | ::= 3 | RHSL 4 | ; 11 | | | | $ | Reduce | Rule ::= ID ::= RHSL; |
| $0 | Rule1 | | | | | | | | Shift | 14 |
| $0 | Rule1 | $14 | | | | | | $ | Reduce | $start ::= Rule $ |
| $0 | $start -1 | | | | | | | | Accept | |

(12 marks)

Continued ...

Print Name _____

(b) Draw the parse tree corresponding to the grammar rules used to parse this input




(4 marks)

Continued ...

Print Name _____

(c) State 4 is

```
lalr_state [4]: {
  [Rule ::= IDENT EXPANDSTO RHSList (*) SEMICOLON , {EOF }]
  [RHSList ::= RHSList (*) OR SymbolList , {SEMICOLON OR }]
}
```

Derive the kernel sets of items of State 12 = GoTo( State 4, OR ), then take its closure to get the full set of items.

(8 marks)

Continued ...

Print Name _____

**2. Write a grammar for interface declarations [16 Marks]**

A typical interface declaration in the Assignment 4 OBJECT6 language is

```
interface X
begin
     int f( []int x, y; var []int z; );
     int g( int a; int b,c; );
end
```

The body of an interface declaration is composed of a list of 0 or more abstract method declarations. There are no constant declarations.

An abstract method declaration has a return type, name, and a list of 0 or more formal parameter declarations.

An abstract method declaration has no body, which is replaced by a ";".

Formal parameter declarations have an optional "var" (for var parameters), a type, a comma separated list of identifiers, then a ";".

Write a grammar for interface declarations. You do NOT have to write grammar rules for "Type"s.

Continued ...

Print Name _____

(16 marks)

Continued ...

Print Name _____

**3. Interpretation [13 Marks]**

(a) Suppose we implement an infinite loop, as in the Assignment 3 INTERP8 language, by a construct of the form

```
for SimpleDeclStmtList do DeclStmtList end
```

and a conditional break statement of the form

```
while Expr;
```

that causes the innermost loop to be exited if the condition is false.

Also assume if statements are of the form

```
if Expr then DeclStmtList ElseOpt end
```

Translate the Java statements

```
int max = 0;
for ( int i = 0; i < n; i++ )
     if ( a[ i ] > max )
          max = a[ i ];
```

into this syntax.

(1 mark)

(b) Indicate the code to implement the node class for a **for** statement of this form (pseudocode or precise English is satisfactory).

```
package node.stmtNode;
import ...;

public class ForStmtNode extends StmtNode {

     private DeclStmtListNode initial;
     private DeclStmtListNode loopBody;
     private Env initEnv;
     private Env loopEnv;

     public ForStmtNode(
          DeclStmtListNode initial,
          DeclStmtListNode loopBody ) {
          this.initial = initial;
          this.loopBody = loopBody;
          }

     public String toString() {
```

(2 marks)

```
          }
```

Continued ...

Print Name _____

```
    public void genEnv( Env env ) {
```

(3 marks)

```
        }

    public void setType() {
        initial.setType();
        loopBody.setType();
        }

    public void checkType() {
        initial.checkType();
        loopBody.checkType();
        }

    public void eval( RunEnv runEnv ) throws UserException {
```

(5 marks)

```
        }
    }
```

Continued ...

Print Name _____

(c) Indicate the code to implement the node class for a **while** statement of this form (pseudocode or precise English is satisfactory).

```
package node.stmtNode;
import ...;

public class WhileStmtNode extends StmtNode {

    private ExprNode cond;

    public WhileStmtNode( ExprNode cond ) {
        this.cond = cond;
        }

    public String toString() {
        return "%-while " + cond + ";%+";
        }

    public void genEnv( Env env ) {
        cond.genEnv( env );
        }

    public void setType() {
        }

    public void checkType() {
        Type condType = cond.checkType();
        cond = cond.castTo( BoolType.type );
        }

    public void eval( RunEnv runEnv ) throws UserException {
```

(2 marks)

```
        }
    }
```

Print Name _____

**4. Show the run time stack.** **[16 Marks]**

Use the program written in the Chapter 8 INTERP7 language in the **Appendix For Question 4**.

Complete the drawing of the data structure built for the global variables "`source`" and "`dest`".

Display the stack frames (activation records) for all methods in the process of being invoked when the maximum level of nesting of method invocations occurs when the statement "`deleteNode( 0, source, dest, 6 );`" on line 40 is invoked, and the process is almost ready to return. At this stage the process should be executing the method "`deleteNode`" at line 24.

Indicate the appropriate values for each stack frame (activation record) you draw. The line numbers on the left-hand side of the program should be used to represent the return address. Draw appropriate arrows for the var parameters, and pointers to objects. For var parameters, make sure you indicate the exact field pointed to in an object very clearly. Represent `List` nodes as shown in the sample entries.

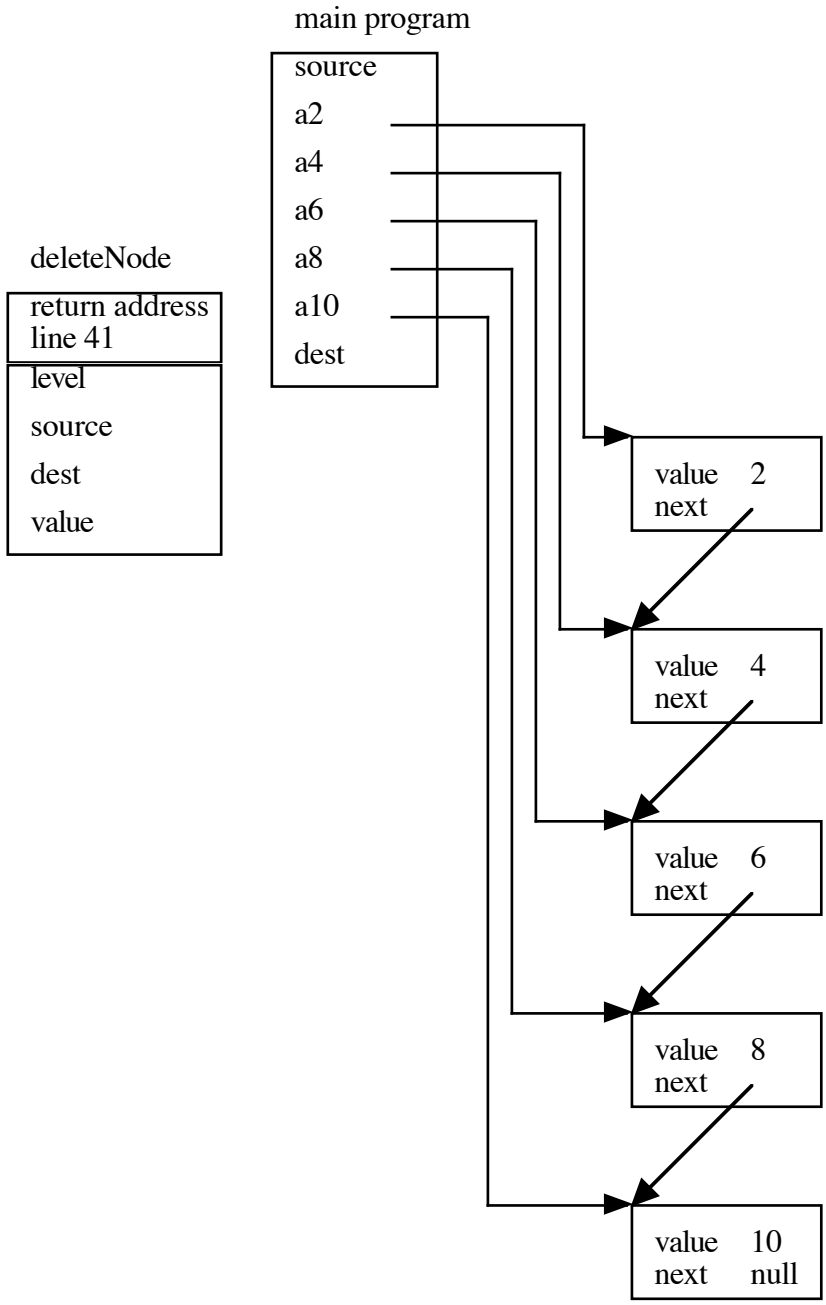
Also indicate the output generated by the complete execution of the program.

Output generated:

(3 marks)

Continued ...

Print Name _____




(13 marks)

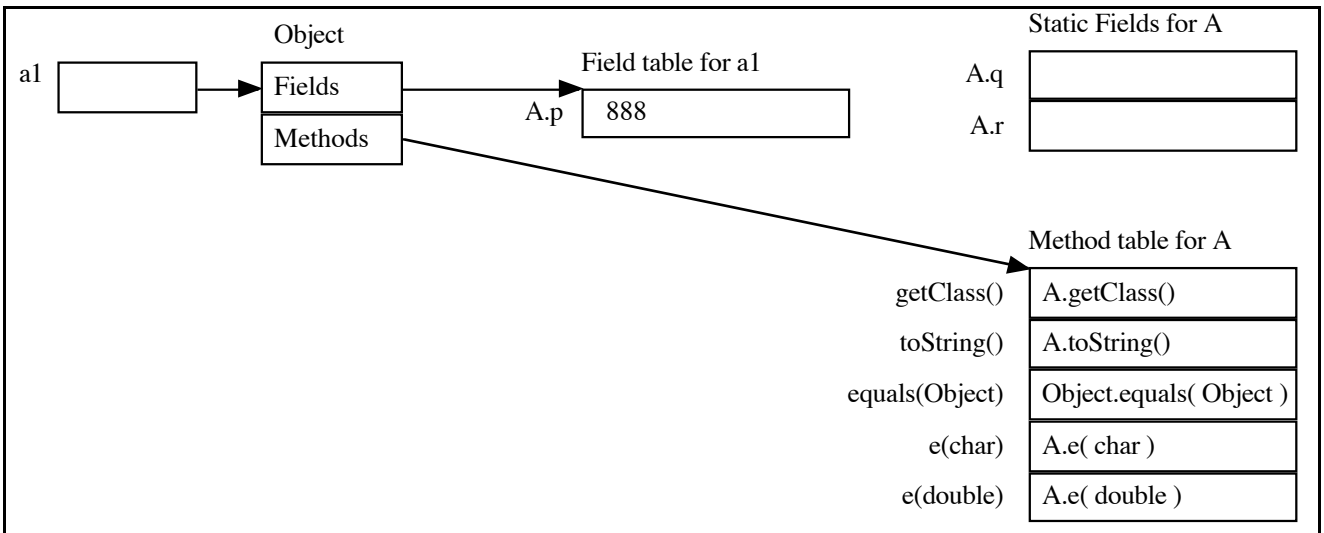
Continued ...

Print Name _____

**5. Implementation of object oriented languages [16 Marks]**

Use the Java program in the **Appendix For Question 5**.

(a) Draw a diagram showing the data structures (object, field table, method table, etc) created for the static fields for Classes `A` and `B`, and the variables `a1`, `a2`, `b1`, `b2`, within the method `Main.main`. Shared data structures should be drawn only once.




(11 marks)

Continued ...

Print Name _____

(b) Indicate the output generated by the method Main.main.

```
A.q =

A.r =

B.q =


a1.p =

a2.p =

b1.p =

b2.p =


a1 =

a2 =

b1 =

b2 =


a1.e( 'A' ) =

b1.e( 'A' ) =

b2.e( 'A' ) =


a1.e( 65 ) =

b1.e( 65 ) =

b2.e( 65 ) =


a1.e( 65.0 ) =

b1.e( 65.0 ) =

b2.e( 65.0 ) =
```

(5 marks)

Continued ...

Print Name _____

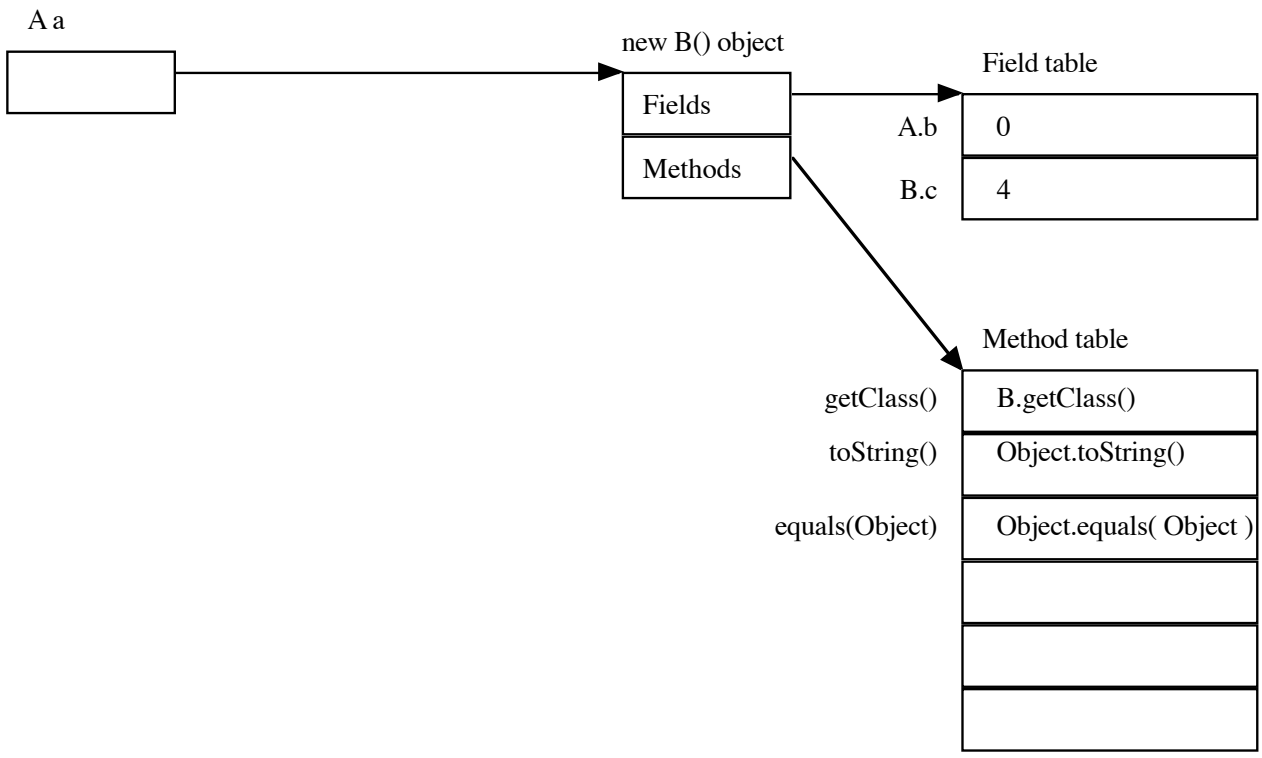
**6. Code generation [15 Marks]**

Use the Assignment 4 OBJECT6 program in the **Appendix For Question 6**.

(a) Draw a diagram in the style of question 5, to show the data structure generated for the variables "`a`" and "`x`".

You are required to put explicit numerical values in for the offsets stored in the interface table.

You may assume the method table for the `Object` class contains exactly 3 entries (`getClass()`, `toString()`, `equals(Object)`).




(7 marks)

Continued ...

Print Name _____

(b) Indicate the Alpha assembly language likely to be generated by the line

```
x.set( 4 );
```

Use the symbolic name "x", for the address of the global variable "x", and "set" for the offset of the entry for "set" in the interface mapping table. Add comments to explain the purpose of your code.

Assume objects are stored in a manner compatible with the diagram displayed in question 5. Assume the current object is pointed to by the "$ip" register, the current stack frame (activation record) is pointed to by "$fp", and the top of stack is pointed to "$sp". Assume that the actual parameters are passed on the stack.

**Note: An appendix is provided with common Alpha instructions.**

**Note: Addresses are represented using 8 bytes on the Alpha.**

(8 marks)

_____End of Questions_____

Continued ...

Print Name _____

This Page is left blank for questions that overflow

End of Questions, Appendices follow

## Appendix For Question 1

## Grammar

```
terminal String
          IDENT,    EXPANDSTO,     SEMICOLON,     OR,  COLON, ERROR;
          //        ::=            ;              |    :

non terminal Rule, RHSList, SymbolList, Symbol;

start with Rule;

Rule::=
          IDENT EXPANDSTO RHSList SEMICOLON          //   Rule 1
     ;

RHSList::=
          SymbolList                                 //   Rule 2
     |
          RHSList OR SymbolList                      //   Rule 3
     ;

SymbolList::=
          /* Empty */                                //   Rule 4
     |
          Symbol SymbolList                          //   Rule 5
     ;

Symbol::=
          IDENT                                      //   Rule 6
     |
          IDENT COLON IDENT                          //   Rule 7
     ;
```

**Appendix For Question 1 Continued On Next Page ....**

## Appendix For Question 1 Continued ...

## Action Table

```
From state #0
     IDENT:SHIFT(2)
From state #1
     EOF:SHIFT(14)
From state #2
     EXPANDSTO:SHIFT(3)
From state #3
     IDENT:SHIFT(7) SEMICOLON:REDUCE(4) OR:REDUCE(4)
From state #4
     SEMICOLON:SHIFT(11) OR:SHIFT(12)
From state #5
     SEMICOLON:REDUCE(2) OR:REDUCE(2)
From state #6
     IDENT:SHIFT(7) SEMICOLON:REDUCE(4) OR:REDUCE(4)
From state #7
     IDENT:REDUCE(6) SEMICOLON:REDUCE(6) OR:REDUCE(6)
     COLON:SHIFT(8)
From state #8
     IDENT:SHIFT(9)
From state #9
     IDENT:REDUCE(7) SEMICOLON:REDUCE(7) OR:REDUCE(7)
From state #10
     SEMICOLON:REDUCE(5) OR:REDUCE(5)
From state #11
     EOF:REDUCE(1)
From state #12
     IDENT:SHIFT(7) SEMICOLON:REDUCE(4) OR:REDUCE(4)
From state #13
     SEMICOLON:REDUCE(3) OR:REDUCE(3)
From state #14
     EOF:REDUCE(0)
```

## Reduce (Go To) Table

```
From state #0:
     Rule:GOTO(1)
From state #1:
From state #2:
From state #3:
     RHSList:GOTO(4)
     SymbolList:GOTO(5)
     Symbol:GOTO(6)
From state #4:
From state #5:
From state #6:
     SymbolList:GOTO(10)
     Symbol:GOTO(6)
From state #7:
From state #8:
From state #9:
From state #10:
From state #11:
From state #12:
     SymbolList:GOTO(13)
     Symbol:GOTO(6)
From state #13:
From state #14:
```

## Appendix For Question 4

```
1 type List = struct( int value; List next );
2
3 void printList( List source ) {
4  print( "{ " );
5  while ( source != null ) {
6      print( source.value );
7      source = source.next;
8      if ( source != null )
9           print( ", " );
10          };
11     println( " }" );
12     };
13
14 void createNode( int level; var List dest; int value; List next ) {
15     dest = new List{ value, next };
16     };
17
18 void deleteNode( int level; List source; var List dest; int value ) {
19     if ( source == null || value < source.value ) {
20          dest = source;
21          }
22     else if ( value == source.value ) {
23          dest = source.next;
24           // Show state at this point
25          }
26     else {
27          createNode( level + 1, dest, source.value, null );
28          deleteNode( level + 1, source.next, dest.next, value );
29               // Inside the above invocation
30          }
31     };
32
33 List source, a2, a4, a6, a8, a10, dest;
34 a10 = new List{ 10, null };
35 a8 = new List{ 8, a10 };
36 a6 = new List{ 6, a8 };
37 a4 = new List{ 4, a6 };
38 a2 = new List{ 2, a4 };
39 source = a2;
40 deleteNode( 0, source, dest, 6 );    // Inside this invocation
41 printList( source );
42 printList( dest );
43
```

Appendices Continued ...

## Appendix For Question 5

```
class A {

     public static int q = 100, r = 200;

     public int p = 888;

     public A( int p ) { this.p = p; q++; }
     public A() { r++; }

     public String toString() { return "A.toString(): p = " + p; }
     public String e( char c ) { return "A.e( '" + c + "' )"; }
     public String e( double x ) { return "A.e( " + x + " )"; }
     }

class B extends A {

     public static int q = 300;

     public int p = 999;

     public B( int p ) { this.p = p; q++; }

     public String toString() { return "B.toString(): p = " + p; }
     public String e( int i ) { return "B.e( " + i + " )"; }
     public String e( double x ) { return "B.e( " + x + " )"; }
     }
```

## Appendix For Question 5 Continued On Next Page ....

## Appendix For Question 5 Continued From Previous page ...

```
class Main {
     public static void main( String[] args ) {

          A a1 = new A();
          A a2 = new A( 1000 );
          B b1 = new B( 2000 );
          A b2 = b1;

          System.out.println( "A.q = " + A.q );
          System.out.println( "A.r = " + A.r );
          System.out.println( "B.q = " + B.q );
          System.out.println();

          System.out.println( "a1.p = " + a1.p );
          System.out.println( "a2.p = " + a2.p );
          System.out.println( "b1.p = " + b1.p );
          System.out.println( "b2.p = " + b2.p );
          System.out.println();

          System.out.println( "a1 = " + a1 );
          System.out.println( "a2 = " + a2 );
          System.out.println( "b1 = " + b1 );
          System.out.println( "b2 = " + b2 );
          System.out.println();

          // 'A' is ASCII 65
          System.out.println( "a1.e( 'A' ) = " + a1.e( 'A' ) );
          System.out.println( "b1.e( 'A' ) = " + b1.e( 'A' ) );
          System.out.println( "b2.e( 'A' ) = " + b2.e( 'A' ) );
          System.out.println();

          System.out.println( "a1.e( 65 ) = " + a1.e( 65 ) );
          System.out.println( "b1.e( 65 ) = " + b1.e( 65 ) );
          System.out.println( "b2.e( 65 ) = " + b2.e( 65 ) );
          System.out.println();

          System.out.println( "a1.e( 65.0 ) = " + a1.e( 65.0 ) );
          System.out.println( "b1.e( 65.0 ) = " + b1.e( 65.0 ) );
          System.out.println( "b2.e( 65.0 ) = " + b2.e( 65.0 ) );
          System.out.println();

          }
     }
```

Appendices Continued ...

## Appendix For Question 6

```
interface X
     begin
          void set( int c; );
          int get();
     end

class A implements X
     begin
          instance
               int b;
               int get()
                    begin
                         println( "Invoke A.get()" );
                         return b;
                    end
               void print()
                    begin
                         println( get() );
                    end
               void set( int c; )
                    begin
                         println( "Invoke A.set( " + c + " )" );
                         b = c;
                    end
     end

class B extends A
     begin
          instance
               int c;
               void set( int c; )
                    begin
                         println( "Invoke B.set( " + c + " )" );
                         this.c = c;
                    end
               int get()
                    begin
                         println( "Invoke B.get()" );
                         return c;
                    end
     end

A a = new B;
X x = a;
x.set( 4 );
println( x.get() );
```

Appendices Continued ...

## Commonly used Alpha instructions

### Integer operate instructions

`Opcode $regA, $regB, $regC`
intReg[ regC ] = intReg[ regA ] op intReg[ regB ]

`Opcode $regA, constantB, $regC`
The constant is an 8 bit unsigned constant.
intReg[ regC ] = intReg[ regA ] op constantB

### Arithmetic integer operate instructions

| | | |
|---|---|---|
| addq | add | + |
| subq | subtract | - |
| mulq | multiply | * |
| divq/divqu | divide, signed/unsigned | / |
| modq/modqu | modulo, signed/unsigned | % |
| s8addq | scaled 8 add | 8*operandA+operandB |

### Shift integer operate instructions

| | | |
|---|---|---|
| sll | shift left logical | << |
| srl | shift right logical | >>> |
| sra | shift right arithmetic | >> |

### Compare integer operate instructions

| | | |
|---|---|---|
| cmpeq | compare equal | == |
| cmplt/cmpult | compare less than signed/unsigned | < |
| cmple/cmpule | compare less than or equal signed/unsigned | <= |

### Logical integer operate instructions

| | | |
|---|---|---|
| and | and | & |
| bic | bit clear | & ~ |
| bis/or | bit set/or | \| |
| eqv/xornot | equivalent/exclusive or not | ^ ~ |
| ornot | or not | \| ~ |
| xor | exclusive or | ^ |

Appendices Continued ...

**Memory instructions**

```
Opcode $regA, displacement($regB)
Opcode $regA, ($regB)
Opcode $regA, constant
```

The displacement or constant is a 16 bit signed constant.

**Load address instruction**

intReg[ regA ] = displacement + intReg[ regB ]

| | |
|---|---|
| lda | load address |

**Load memory instructions**

intReg[ regA ] = Memory[ displacement + intReg[ regB ] ]

| | |
|---|---|
| ldq | load quadword |
| ldl | load longword |
| ldbu | load byte unsigned |

**Store memory instructions**

Memory[ displacement + intReg[ regB ] ] = intReg[ regA ]

| | |
|---|---|
| stq | store quadword |
| stl | store longword |
| stb | store byte |

**Branch instructions**

**Conditional branch instructions**

```
Opcode $regA, destination
```

if ( condition holds for intReg[ regA ] )
    programCounter = destination

| | |
|---|---|
| beq | branch equal |
| bne | branch not equal |
| blt | branch less than |
| ble | branch less than or equal |
| bgt | branch greater than |
| bge | branch greater than or equal |
| blbs | branch low bit set |
| blbc | branch low bit clear |

**Unconditional branch instructions**

```
Opcode destination;
```

programCounter = destination // br

intReg[ ra ] = programCounter // bsr
programCounter = destination

| | |
|---|---|
| br | branch |
| bsr | branch to subroutime |

Appendices Continued ...

**Jump instruction**

```
Opcode ($regA);
```

programCounter = intReg[ regA ] // jmp

intReg[ ra ] = programCounter // jsr
programCounter = intReg[ regA ]

| | |
|---|---|
| jmp | jump |
| jsr | jump to subroutine |

**Return instruction**

programCounter = intReg[ ra ]

| | |
|---|---|
| ret | return |

**Callpal instruction**

```
call_pal constant;
```

The constant is a 26 bit constant.

| | |
|---|---|
| call_pal | call PALcode |

**Pseudoinstructions**

**Load immediate**

```
ldiq $regA, constant
```

The constant is a 64 bit constant.
intReg[ regA ] = constant

| | |
|---|---|
| ldiq | load immediate quadword |

**Clear**

```
clr $regA
```

intReg[ regA ] = 0

| | |
|---|---|
| clr | clear |

**Unary pseudoinstructions**

```
Opcode $regB, $regC
```

intReg[ regC ] = op intReg[ regB ]

```
Opcode constantB, $regC
```

The constant is an 8 bit unsigned constant.
intReg[ regC ] = op constantB

| | |
|---|---|
| mov | move |
| negq | negate |

_____End of Appendices_____