

# THE UNIVERSITY OF AUCKLAND

**First Semester, 2003**

**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 booklet.

1		23
2		20
3		15
4		15
5		14
6		13
Total		100

Print Name \_\_\_\_\_

**1. Bottom Up LALR(1) Parsing****[23 Marks]**

Consider the CUP grammar:

```
terminal
    LEFT,    RIGHT,    LEFTSQ,    RIGHTSQ,    SEMICOLON,    COMMA,    STRUCT;
//  (      )      [      ]      ;      ,      struct
terminal String IDENT;
```

```
non terminal Type, DeclList, Decl, IdentList;
```

```
start with Type;
```

```
Type ::=
    IDENT
    |
    LEFTSQ RIGHTSQ Type
    |
    STRUCT LEFT DeclList RIGHT
    ;
```

```
DeclList ::=
    DeclList SEMICOLON Decl
    |
    Decl
    ;
```

```
Decl ::=
    Type IdentList
    ;
```

```
IdentList ::=
    IdentList COMMA IDENT
    |
    IDENT
    ;
```

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

```
struct( []int a, b )
```

Assume “int”, “a”, “b” match IDENT.

Continued ...

Print Name \_\_\_\_\_

Stack

\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	struct 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	struct 2	( 7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	struct 2	( 7	[ 4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	struct 2	( 7	[ 4	] 5	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	Type 1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
\$0	Type 1	\$ 18	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Token Action

struct	shift
(	shift
[	shift
]	shift
ID int	shift
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	shift
<input type="checkbox"/>	Accept

(10 marks)

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

(3 marks)

Continued ...

Print Name \_\_\_\_\_

(c) State 2 is

```
lalr_state [2]: {  
  [Type ::= STRUCT (*) LEFT DeclList RIGHT , {EOF IDENT }]  
}  
transition on LEFT to state [7]
```

Derive the kernel sets of items of State 7 = GoTo( State 2, LEFT ), then take its closure to get the full set of items.

(10 marks)

Continued ...

Print Name \_\_\_\_\_

**2. Write a grammar definition****[20 Marks]**

Suppose we have a mathematical notation for sets of unsigned integers.

We can enumerate unsigned integers, as in  $\{\}$ ,  $\{1\}$ ,  $\{1, 2, 4\}$ , etc, to form basic sets.

We can also include ranges of unsigned integers within a basic set, by using a “..” notation, as in  $\{1..10, 20..30, 40..\}$ , etc.

The notation “ $40..$ ” means the infinite range of integers from 40 up to (but excluding) infinity.

We can combine these notations, as in  $\{1, 2, 4, 10..20, 31, 40..\}$ .

An infinite range may only appear as the last component in the list.

We can also combine sets using parentheses  $(...)$  and the left associative binary infix operators  $\cup$  and  $\cap$ , representing union and intersection, as in

$\{1, 2\} \cap \{3\} \cap \{20..40\} \cap (\{25\} \cap \{30..50\})$ .

Union has a lower precedence than intersection.

Write a grammar definition for sets, to match the above notation. You do not have to write any actions.

Continued ...

Print Name \_\_\_\_\_

Continued ...

Print Name \_\_\_\_\_

**3. Interpretation**

**[15 Marks]**

- (a) Suppose you are implementing old style BASIC, with line numbered statements.

Suppose the syntax for a “for” statement is

```
FOR IDENT EQ Expr TO Expr
```

and the syntax for “next” statements is

```
NEXT IDENT
```

Give an example of a simple BASIC program that uses these statements to print out the numbers from 1 to n, one per line.

(1 mark)

Indicate the data structures and the actions needed to implement these statements in general, and explicitly relate them to your example (e.g., specify the actual values stored in your data structure at various times).

Continued ...

Print Name \_\_\_\_\_

(6 marks)

- (b) Describe the notation used in assignment 4 to implement a construct equivalent to “super”. Give an example, and explain what it does.

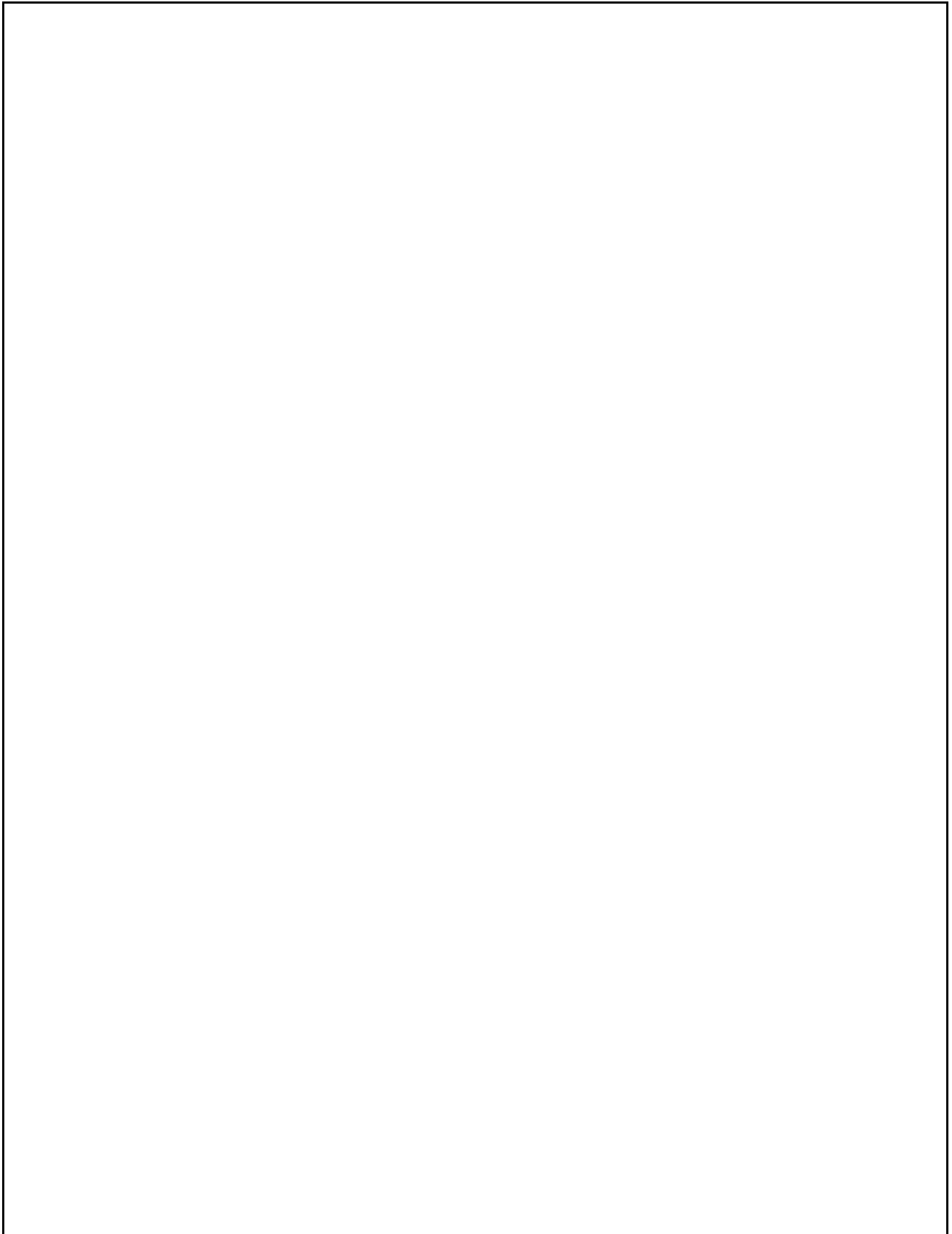
(4 marks)

Continued ...



Print Name \_\_\_\_\_

Describe the data structure generated by this construct. Draw a picture and explain in English what it achieves.



(4 marks)

Continued ...

Print Name \_\_\_\_\_

**4. Show the run time stack.****[15 Marks]**

Use the program in the appendix written in the Chapter 8 INTERP7 language. 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 “**insertList( 0, source, dest, 5 );**” on line 38 is invoked, and the process is almost ready to return. At this stage the process should be executing the method “**createNode**” at line 16, which should have been invoked from the method “**insertList**” at line 21.

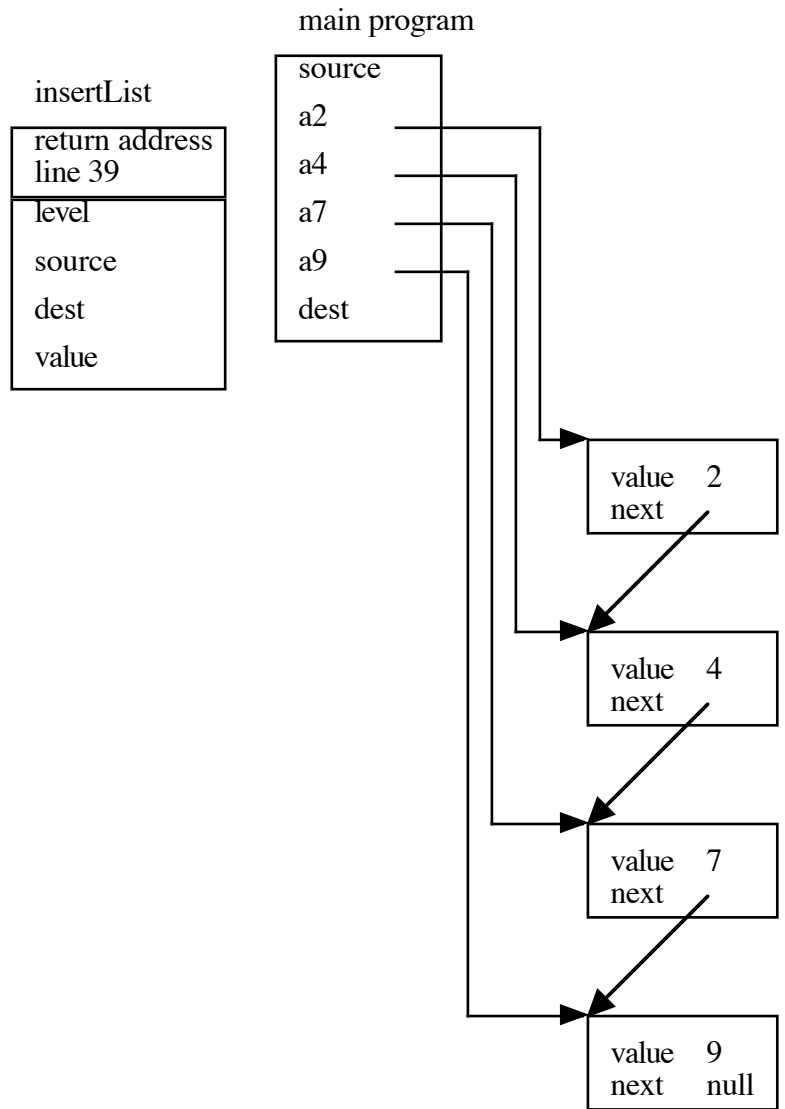
Indicate the appropriate values for each stack frame (activation record) you draw. The line numbers on the left-hand side of the program should be used to represent the return address. Draw appropriate arrows for the var parameters, and pointers to objects. For var parameters, make sure you indicate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

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

Output generated:

(3 marks)

Print Name \_\_\_\_\_



(12 marks)

Continued ...

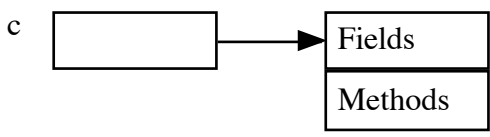
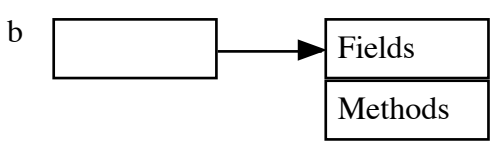
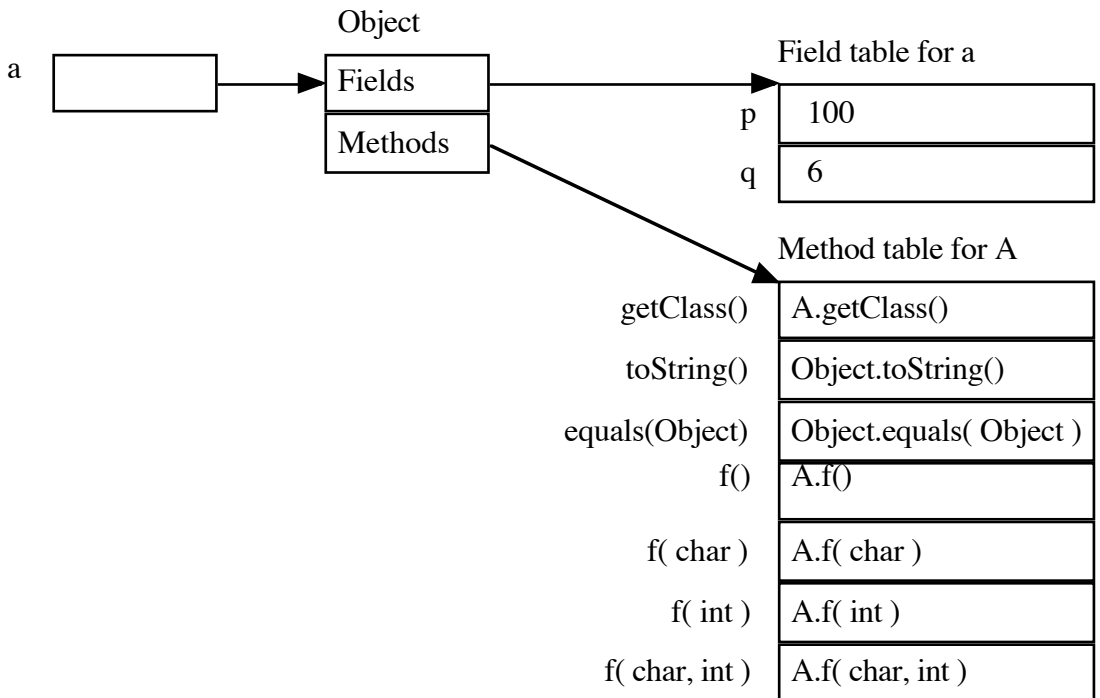
Print Name \_\_\_\_\_

**5. Implementation of object oriented languages**

**[14 Marks]**

Use the Java program in the appendix.

- (a) Draw a diagram showing the data structures (object, field table and method table) created for the variables b and c, within the method Main.main. Shared data structures should be drawn only once.



(10 marks)

Continued ...

Print Name \_\_\_\_\_

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

a.p = 100

a.q = 6

b.p = 150

b.q =

b.r =

c.p =

c.q =

c =

b.f( 'B' ) =

c.f( 'B' ) =

b.f( 'A', 'B' ) =

c.f( 'A', 'B' ) =

c.g( 'A' ) =

(4 marks)

Continued ...

Print Name \_\_\_\_\_

**6. Code generation****[13 Marks]**

Assume objects are stored in a manner compatible with the diagrams 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.

Suppose we have the OBJECT4 program

```
class A
  begin
    instance
      void f( int a, b; var int c; )
      begin
        c = a + b;
      end
    end
  end

class B
  begin
    instance
      A a = new A;
      int x = 2, y;
      a.f( x, 1, y );
      println( "y = " + y );
    end
  end
```

```
B b = new B;
```

Indicate the assembly language likely to be generated by the line

```
c = a + b;
```

Use symbolic names “a”, “b”, “c” for the offsets of the corresponding formal parameter from the base of the relevant register. Add comments to explain the purpose of your code.

(3 marks)

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

Continued ...

Print Name \_\_\_\_\_

Indicate the assembly language likely to be generated by the line

```
a.f( x, 1, y );
```

Use symbolic names “a”, “f”, “x”, “y” for the offsets of the corresponding instance field or method from the base of the relevant table. Add comments to explain the purpose of your code.

(10 marks)

\_\_\_\_\_ End of Questions \_\_\_\_\_

Continued ...

Print Name \_\_\_\_\_

This Page is left blank for questions that overflow



## Appendix For Question 1

### Grammar Rules

```

8: IdentList ::= IDENT
7: IdentList ::= IdentList COMMA IDENT
6: Decl ::= Type IdentList
5: DeclList ::= Decl
4: DeclList ::= DeclList SEMICOLON Decl
3: Type ::= STRUCT LEFT DeclList RIGHT
2: Type ::= LEFTSQ RIGHTSQ Type
1: Type ::= IDENT
0: $START ::= Type EOF

```

### Action Table

```

From state #0
  LEFTSQ:SHIFT(4) STRUCT:SHIFT(2) IDENT:SHIFT(3)
From state #1
  EOF:SHIFT(18)
From state #2
  LEFT:SHIFT(7)
From state #3
  EOF:REDUCE(1) IDENT:REDUCE(1)
From state #4
  RIGHTSQ:SHIFT(5)
From state #5
  LEFTSQ:SHIFT(4) STRUCT:SHIFT(2) IDENT:SHIFT(3)
From state #6
  EOF:REDUCE(2) IDENT:REDUCE(2)
From state #7
  LEFTSQ:SHIFT(4) STRUCT:SHIFT(2) IDENT:SHIFT(3)
From state #8
  RIGHT:SHIFT(16) SEMICOLON:SHIFT(15)
From state #9
  IDENT:SHIFT(11)
From state #10
  RIGHT:REDUCE(5) SEMICOLON:REDUCE(5)
From state #11
  RIGHT:REDUCE(8) SEMICOLON:REDUCE(8) COMMA:REDUCE(8)
From state #12
  RIGHT:REDUCE(6) SEMICOLON:REDUCE(6) COMMA:SHIFT(13)
From state #13
  IDENT:SHIFT(14)
From state #14
  RIGHT:REDUCE(7) SEMICOLON:REDUCE(7) COMMA:REDUCE(7)
From state #15
  LEFTSQ:SHIFT(4) STRUCT:SHIFT(2) IDENT:SHIFT(3)
From state #16
  EOF:REDUCE(3) IDENT:REDUCE(3)
From state #17
  RIGHT:REDUCE(4) SEMICOLON:REDUCE(4)
From state #18
  EOF:REDUCE(0)

```

**Note: The Reduce (Go To) Table is over the page.**

**Reduce (Go To) Table**

```
From state #0:
  Type:GOTO(1)
From state #1:
From state #2:
From state #3:
From state #4:
From state #5:
  Type:GOTO(6)
From state #6:
From state #7:
  Type:GOTO(9)
  DeclList:GOTO(8)
  Decl:GOTO(10)
From state #8:
From state #9:
  IdentList:GOTO(12)
From state #10:
From state #11:
From state #12:
From state #13:
From state #14:
From state #15:
  Type:GOTO(9)
  Decl:GOTO(17)
From state #16:
From state #17:
From state #18:
```

**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   // Show state at this point
17 };
18
19 void insertList( int level; List source; var List dest; int value ) {
20   if ( source == null || value <= source.value ) {
21     createNode( level + 1, dest, value, source );
22     // Inside the above invocation
23   }
24   else {
25     createNode( level + 1, dest, source.value, null );
26     insertList( level + 1, source.next, dest.next, value );
27     // Inside the above invocation
28   }
29 };
30
31 List source, a2, a4, a7, a9, dest;
32 a9 = new List{ 9, null };
33 a7 = new List{ 7, a9 };
34 a4 = new List{ 4, a7 };
35 a2 = new List{ 2, a4 };
36 source = a2;
37 dest = null;
38 insertList( 0, source, dest, 5 ); // Inside this invocation
39 printList( source );
40 printList( dest );
41
```

**Appendix For Question 5**

```
class A {
    public int p, q = 6;
    public A( int p ) { this.p = p; }
    public String f() { return "A.f()"; }
    public String f( char c ) { return "A.f( '" + c + "' )"; }
    public String f( int y ) { return "A.f( " + y + " )"; }
    public String f( char c, int d ) { return "A.f( '" + c + "', " + d + " )"; }
    public static String g( int x ) { return "A.g( " + x + " )"; }
}

class B extends A {
    public int p, r;
    public B( int p, int q, int r ) {
        super( q );
        this.p = p; this.r = r;
    }
    public String toString() { return "B"; }
    public String d() { return "B.d()"; }
    public String h() { return "B.h()"; }
    public String f( char c ) { return "B.f( '" + c + "' )"; }
    public String f( int x ) { return "B.f( " + x + " )"; }
    public String f( char c, char d ) {
        return "B.f( '" + c + "', '" + d + "' )";
    }
    public static String g( char c ) { return "B.g( '" + c + "' )"; }
}

class Main {
    public static void main( String[] args ) {
        A a = new A( 100 );
        B b = new B( 150, 160, 170 );
        A c = new B( 210, 220, 230 );
        System.out.println( "a.p = " + a.p );
        System.out.println( "a.q = " + a.q );
        System.out.println( "b.p = " + b.p );
        System.out.println( "b.q = " + b.q );
        System.out.println( "b.r = " + b.r );
        System.out.println( "c.p = " + c.p );
        System.out.println( "c.q = " + c.q );
        System.out.println( "c = " + c );
        // 'A' is ASCII 65, 'B' is ASCII 66
        System.out.println( "b.f( 'B' ) = " + b.f( 'B' ) );
        System.out.println( "c.f( 'B' ) = " + c.f( 'B' ) );
        System.out.println( "b.f( 'A', 'B' ) = " + b.f( 'A', 'B' ) );
        System.out.println( "c.f( 'A', 'B' ) = " + c.f( 'A', 'B' ) );
        System.out.println( "c.g( 'A' ) = " + c.g( 'A' ) );
    }
}
```

## Commonly used Alpha instructions

### Integer operate instructions

Opcode \$regA, \$regB, \$regC  
 $\text{intReg[ regC ]} = \text{intReg[ regA ] op intReg[ regB ]}$

Opcode \$regA, constantB, \$regC  
 The constant is an 8 bit unsigned constant.  
 $\text{intReg[ regC ]} = \text{intReg[ regA ] op constantB}$

### 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	^

### 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

$\text{intReg[ regA ]} = \text{displacement} + \text{intReg[ regB ]}$

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

### Load memory instructions

$\text{intReg[ regA ]} = \text{Memory[ displacement} + \text{intReg[ regB ] ]}$

ldq	load quadword
ldbu	load byte unsigned

**Store memory instructions**

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

stq	store quadword
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 subroutine

**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
-----	--------

\_\_\_\_\_ End of Appendix \_\_\_\_\_