Computer Science 330 Language Implementation

Confirmed Examination Information 2002

This may not be exactly what is in the examination, because it is not completely finalised. In particular, the marks for questions could be wrong by a mark or two. However, the general style should be about right.

This Examination is out of 70 Marks. Attempt **ALL** questions. Write your answers in the spaces provided in this booklet.

1		12
2		6
3		14
4		12
5		8
6		10
7		8
Total		70

1. Bottom Up LALR(1) Parsing [12 Marks]

Consider the CUP grammar:

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input ... (8 marks)

(b) Draw the parse tree corresponding to the grammar rules used to parse this input (2 marks)

(c) Draw the abstract syntax tree actually built by the actions in the CUP program (2 marks)

2. JLex [6 Marks]

Write a JLex program that ...

3. Write a grammar definition [14 Marks]

You only have to write the grammar. You do not have to write any actions.

4. Show the run time stack. [12 Marks]

Below is a program in the assignment 4 OBJECT3 language. Complete the drawing of the data structure built for the global variable "...", and the run-time stack when the maximum level of nesting of method invocations occurs within the statement "..." on line ..., and the process is almost ready to return. Remember the stack grows "up" towards low memory. Enter the appropriate values for each stack frame (activation record) you draw. Assume that all variables are implicitly initialised to zero or null. The line numbers on the left-hand side of the program should be used to represent the value of the program counter. Draw appropriate arrows for the var parameters, and pointers to class objects. Make sure you indicate the exact field pointed to in an object very clearly. The stack information has already been filled in for the main program, and most of the first stack frame. You may label addresses, and refer to them, rather than drawing arrows, if you need to, to make your answer clearer.

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

5. Type checking/evaluation [8 Marks]

Type checking/evaluation corresponding to some construct in EXPREVAL6, the assignment 3 EXPREVAL10 or assignment 4 OBJECT3 language.

6. Implementation of Object Oriented Languages [10 Marks]

Suppose we have the Java program …

(a) Draw a diagram showing the data structures (object, field table and method table) created for the variables ..., within the method Main.main. Shared data structures should be drawn only once. Note that you have to include the getClass method in the method table.

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

7. Code generation [8 Marks]

Generate assembly language corresponding to some construct in the assignment 4 OBJECT3 language.

Bruce Hutton

Exercises

Exercise 1(a)

Try drawing the stack and appropriate data structures at the maximal level of recursion

```
class List
     begin
          instance
               int value;
               List next;
               List init( int value; List next; )
                    begin
                         this.value = value;
                         this.next = next;
                         return this;
                    end
     end

void printList( List a; )
         begin
                  print( "{ " );
                  while a <> null do
                           print( a.value );
                           a = a.next;
                           if a <> null then
                                    print( ", " );
                           end
                  end
                  print( " }" );
         end

void appendHead( var List a; int data; )
         begin
                   a = new List.init( data, a );
         end

void insertList( var List a; int data; )
         begin
                   if a == null or data <= a.value then
                            appendHead( a, data );
                  else
                            insertList( a.next, data );
                  end
         end

List a, a1, a3, a5;
```

```
a5 = new List.init( 5, null );
a3 = new List.init( 3, a5 );
a1 = new List.init( 1, a3 );
a = a1;
insertList( a, 4 );
printList( a );
println();
```

Exercise 1(b)

Try drawing the stack and appropriate data structures at the maximal level of recursion

```
class List
     begin
          instance
               int value;
               List next;
               List init( int value; List next; )
                    begin
                         this.value = value;
                         this.next = next;
                         return this;
                    end
     end

void appendText( var []char result; []char text; )
         begin
                  result = result + text;
         end

void buildListText( List a; var []char result; )
         begin
                  if a <> null then
                           appendText( result, a.value );
                           if a.next <> null then
                                    appendText( result, ", " );
                                    buildListText( a.next, result );
                           end
                  end
         end

List a, a1, a3, a5;
[]char result;
a5 = new List.init( 5, null );
a3 = new List.init( 3, a5 );
a1 = new List.init( 1, a3 );
a = a1;
result = "{ ";
buildListText( a, result );
appendText( result, " }" );
println( result );
```

Exercise 1(c)

Try drawing the stack and appropriate data structures at the maximal level of recursion

```
class List
     begin
          instance
               int value;
               List next;
               List init( int value; List next; )
                    begin
                         this.value = value;
                         this.next = next;
                         return this;
                    end
     end
```

```
void buildListText( List a; var []char result; )
        begin
                  []char result1;
                 if a <> null then
                          if a.next == null then
                                   result = "" + a.value;
                          else
                                    buildListText( a.next, result1 );
                                   result = a.value + ", " + result1;
                          end
                 end
        end

List a, a1, a3, a5;
[]char result;
a5 = new List.init( 5, null );
a3 = new List.init( 3, a5 );
a1 = new List.init( 1, a3 );
a = a1;
buildListText( a, result );
println( "{" + result + "}" );
```

Exercise 2

Which determines the meaning for the following - the declared type, or the actual type?

- static variables.
- static methods.
- instance variables.
- instance methods.

If you overload a method

```
void t( int x, int y )
void t( double x, double y )
```

in a superclass

then override the method

```
void t( double x, double y )
```

in a subclass, what happens if you try to invoke t(1,2) in the subclass?

How is an object stored at run time?

What can you say about the layout of an object of a subclass, compared with the layout of an object of the superclass?

What can you say about the allocation of space, when

- an instance variable is declared with the same name as one in a superclass?
- an instance method is declared with the same name as one in a superclass?
- what if the name is the same, but the signature is different?

What can you say about storage for two variables a, b of reference type, after an assignment "a = b;"?

What can you say about storage for two variables a, b of reference type, when the actual values are of the same type?

What is the output from the following program?

Draw diagrams showing the layout for the objects.

```
class A {
     static int a = 1, b = 2, c = 3;
     int g = 11, h = 12, i = 13;
     static void l() { System.out.println( "A.l()" ); }
     static void m() { System.out.println( "A.m()" ); }
```

```
     void o() { System.out.println( "A.o()" ); }
     void p() { System.out.println( "A.p()" ); }
     void t( int x, int y ) { System.out.println( "A.t(int, int)" ); }
     void t( double x, double y ) { System.out.println( "A.t(double, double)" );
}
     A( int g, int h, int i ) { this.g = g; this.h = h; this.i = i; }
     }

class B extends A {
     static int b = 4, c = 5, d = 6;
     int h = 14, i = 15, j = 16;
     static void m() { System.out.println( "B.m()" ); }
     static void n() { System.out.println( "B.n()" ); }
     void o() { System.out.println( "B.o()" ); }
     void r() { System.out.println( "B.r()" ); }
     void t( int x, int y ) { System.out.println( "B.t(int, int)" ); }
     B( int Ag, int Ah, int Ai, int g, int h, int i, int j ) {
          super( Ag, Ah, Ai );
          this.g = g; this.h = h; this.i = i; this.j = j;
          }
     }

class C extends B {
     static int c = 7, e = 8;
     int h = 17, k = 18;
     static void l() { System.out.println( "C.l()" ); }
     static void m( int x ) { System.out.println( "C.m(int)" ); }
     void r() { System.out.println( "C.r()" ); }
     void s() { System.out.println( "C.s()" ); }
     C( int Ag, int Ah, int Ai, int Bg, int Bh, int Bi, int Bj, int h, int k ) {
          super( Ag, Ah, Ai, Bg, Bh, Bi, Bj );
          this.h = h; this.k = k;
          }
     }

class D extends B {
     static int e = 9, f = 10;
     int a = 19, i = 20, k = 21;
     static void l() { System.out.println( "D.l()" ); }
     void o() { System.out.println( "D.o()" ); }
     void r( int x ) { System.out.println( "D.r(int)" ); }
     D( int Ag, int Ah, int Ai, int Bg, int Bh, int Bi, int Bj, int a, int i, int
k ) {
          super( Ag, Ah, Ai, Bg, Bh, Bi, Bj );
          this.a = a; this.i = i; this.k = k;
          }
     }

public class JavaTest {

     static void printAll( String name, A a ) {
          System.out.println( name + ".a = " + a.a );
          System.out.println( name + ".b = " + a.b );
          System.out.println( name + ".c = " + a.c );
          System.out.println( name + ".g = " + a.g );
          System.out.println( name + ".h = " + a.h );
          System.out.println( name + ".i = " + a.i );
          a.l();
          a.m();
          a.o();
          a.p();
          a.t(1,2);
          }

     static void printAll( String name, B b ) {
          System.out.println( name + ".a = " + b.a );
```

```
        System.out.println( name + ".b = " + b.b );
        System.out.println( name + ".c = " + b.c );
        System.out.println( name + ".d = " + b.d );
        System.out.println( name + ".g = " + b.g );
        System.out.println( name + ".h = " + b.h );
        System.out.println( name + ".i = " + b.i );
        System.out.println( name + ".j = " + b.j );
        b.l();
        b.m();
        b.n();
        b.o();
        b.p();
        b.r();
        b.t(1,2);
        b.t(1.0,2);
        }

    static void printAll( String name, C c ) {
        System.out.println( name + ".a = " + c.a );
        System.out.println( name + ".b = " + c.b );
        System.out.println( name + ".c = " + c.c );
        System.out.println( name + ".d = " + c.d );
        System.out.println( name + ".e = " + c.e );
        System.out.println( name + ".g = " + c.g );
        System.out.println( name + ".h = " + c.h );
        System.out.println( name + ".i = " + c.i );
        System.out.println( name + ".j = " + c.j );
        System.out.println( name + ".k = " + c.k );
        c.l();
        c.m();
        c.m(3);
        c.n();
        c.o();
        c.p();
        c.r();
        c.s();
        c.t(1,2);
        c.t(1.0,2);
        }

    static void printAll( String name, D d ) {
        System.out.println( name + ".a = " + d.a );
        System.out.println( name + ".b = " + d.b );
        System.out.println( name + ".c = " + d.c );
        System.out.println( name + ".d = " + d.d );
        System.out.println( name + ".e = " + d.e );
        System.out.println( name + ".f = " + d.f );
        System.out.println( name + ".g = " + d.g );
        System.out.println( name + ".h = " + d.h );
        System.out.println( name + ".i = " + d.i );
        System.out.println( name + ".j = " + d.j );
        System.out.println( name + ".k = " + d.k );
        d.l();
        d.m();
        d.n();
        d.o();
        d.p();
        d.r();
        d.r(3);
        d.t(1,2);
        d.t(1.0,2);
        }

    public static void main( String[] arg ) {
        A a = new A( 22, 23, 24 );
        B b1 = new B( 25, 26, 27, 28, 29, 30, 31 );
```

```
        B b2 = new B( 32, 33, 34, 35, 36, 37, 38 );
        C c = new C( 41, 42, 43, 44, 45, 46, 47, 48, 49 );
        D d = new D( 51, 52, 53, 54, 55, 56, 57, 58, 59, 60 );
        A a1 = b1;
        A a2 = b2;
        A a3 = c;
        A a4 = d;
        B c1 = c;
        printAll( "a", a );
        printAll( "b1", b1 );
        printAll( "b2", b2 );
        printAll( "c", c );
        printAll( "d", d );
        printAll( "a1", a1 );
        printAll( "a2", a2 );
        printAll( "a3", a3 );
        printAll( "a4", a4 );
        printAll( "c1", c1 );
        }

    }
```

Exercise 3

What data structures and code would you expect to generate for the switch statement, in your own implementation of switch statements?

```
    switch ( i ) {
        2, 4:
            println( "small and even" );
        1..5:
            println( "small" );
        6..9:
            println( "big" );
        ..-1:
            println( "negative" );
        ..:
            println( "default" )
        }
```

Exercise 4

Describe the code generation algorithm for fields in the OBJECT3 language. Think up examples, and indicate the code generated.

Describe the code generation algorithm for binary arithmetic operators. Explain why it is necessary to have a tree weighting pass for the compiler? What difference would it make if we omitted this phase, and dealt with running out of registers when it occurred?

Consider the assignment statement

```
a = b - c + ( d / e ) * ( f / g );
```

Perform tree weighting, and code generation, on the assumption that there are only 2, 3, or 4 available registers, and assuming that the compiler has been improved to generate good code for simple variables.

Describe the code generation algorithm for function invocation in the OBJECT3 language. What is the layout for a stack frame in the OBJECT3 compiler? Think up examples, and indicate the code generated.

Suppose a function had 3 formal parameters and 2 local variables, and needed two temporary memories (due to running out of temporary registers). What would the stack frame look like?

Describe the code generation algorithm for binary logical operators.

Consider the if statement

```
if a < b and c < d or e < f and g < h then
    x = 1;
else
```

```
x = 2;
```

What code would be generated?

Explain it in terms of your recursive code generation algorithm.