Computer Science 330 Language Implementation

Examination Information 2007

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.

Note: Many portions of the examination are related to the assignments. It is important that you do the assignments.

1		10
2(a)		15
2(b)		5
3(a)		15
3(b)		5
4		17
5		18
6		15
Total		100

1. JFlex **[10 Marks]**

Indicate at least 10 different kinds of errors in the following fragment of JFlex code. ("..." just means omitted code). Give a reason or appropriate correction for each one.

Preparation:

Chapter 1. Making mistakes when trying to implement a lexical analyser. This year's test. Last year's test and examination.

2. Bottom Up LALR(1) Parsing **[20 Marks]**

Consider the CUP grammar in the **Appendix For Question 2**. Note that some rules are left recursive, while other rules are right recursive. Also note the rule for "..." that expands to empty.

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

(15 marks)

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

(5 marks)

Preparation:

Assignment 1, previous tests and examinations.

3(a). Write a grammar for... [20 Marks]

(a) Write a grammar for... **in general**.

You do not have to define the grammar for ...

You do not have to include any actions ("{...}") or attribute names (":ident"). You may write terminal symbols such as keywords, and special symbols directly by enclosing them in double quotes ("...").

(15 marks)

Preparation:

Previous tests and examinations. Assignment 2. B-- Grammar. Java grammar (appendix 4). Designing grammars for various constructs.

3(b). A question directly related to assignment 2 [5 Marks]

Preparation:

Assignment 2.

4. Show the run time stack for a B-- program [17 Marks]

Use the program written in the B-- language in the **Appendix For Question 4**.

Complete the drawing of the data structure built for the global variables ....

Use multiple colours (other than red) for arrows, etc., to make your diagram clearer.

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

. . .

on line ... is invoked, and the process is almost ready to return. At this stage the process should be executing the method ". . ." at line ....

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 variable pointed to in an object very clearly. Represent nodes as shown in the sample entries.

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

Preparation:

Examples in Chapter 10. Previous examinations. Write your own examples that involve simple data structures and recursion, with var parameters. Note that the B-- language is a bit different from the one in much earlier examinations.

5. Implementation of object oriented languages [18 Marks]

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

(a) Draw a diagram showing the data structures (variable, field table, method table, etc) created for the variables..., within the method `Main.main`. Shared data structures should be drawn only once.

(12 marks)

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

(6 marks)

Preparation:

Primarily experimentation: Write simple Java programs to test your understanding, that use overloading, overriding, constructors that explicitly or implicitly invoke the constructor of the superclass, constructors with side effects that modify static fields, accessing the same object through variables of different type, instance and static variables of the same name in different classes, implicit use of the toString method, when conversion of an object to a String, methods declared in a superclass, that refer to variables declared in both the superclass and subclass, etc. Previous examinations.

Note that the way I represent objects this year is different from two years ago. So any solutions in previous examinations have to be adjusted to take this into account.

6. Code generation [15 Marks]

(a) Consider the the B-- language. Answer some relatively high level questions about the language, especially related to method invocation.

(7 marks)

(a) Indicate the Alpha assembly language likely to be generated for the following statements.

(8 marks)

An appendix is also provided describing common Alpha instructions.

Preparation:

B-- chapters in the lecture notes. Write simple B-- programs to access instance fields, invoke instance methods, pass var and value parameters, etc.

福

Bruce Hutton

Exercises

1. Bottom Up LALR(1) Parsing

Consider the grammar

```
terminal IDENT, LEFT, RIGHT, COMMA, SEMICOLON, ETC, ERROR;

non terminal MethodHeader, FormalParamDeclList, FormalParamDecl, Type, IdentList;

start with MethodHeader;

MethodHeader::=
          Type IDENT LEFT FormalParamDeclList RIGHT
     ;

FormalParamDeclList::=
          FormalParamDecl
     |
          ETC
     |
          FormalParamDecl SEMICOLON FormalParamDeclList
     ;

FormalParamDecl::=
          Type IdentList
     ;

Type::=
          IDENT
     ;

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

with lexical analyser

```
ident      =    [A-Za-z][A-Za-z0-9]*
space      =    [\ \t]
newline         =    \r|\n|\r\n

%%

"("             { return token( sym.LEFT ); }
")"             { return token( sym.RIGHT ); }
","             { return token( sym.COMMA ); }
";"             { return token( sym.SEMICOLON ); }
"..."           { return token( sym.ETC ); }
{newline}       { lineNumber++; }
{space}         { }

{ident}         { return token( sym.IDENT ); }

.               { return token( sym.ERROR ); }
<<EOF>>         { return token( sym.EOF ); }
```

The action and goto tables are

-------- ACTION_TABLE --------

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

-------- REDUCE_TABLE --------

```
From state #0:
     MethodHeader:GOTO(1)
     Type:GOTO(2)
From state #1:
From state #2:
From state #3:
From state #4:
From state #5:
     FormalParamDeclList:GOTO(8)
     FormalParamDecl:GOTO(9)
     Type:GOTO(7)
From state #6:
From state #7:
     IdentList:GOTO(14)
From state #8:
From state #9:
From state #10:
     FormalParamDeclList:GOTO(11)
     FormalParamDecl:GOTO(9)
     Type:GOTO(7)
From state #11:
```

```
From state #12:
From state #13:
From state #14:
From state #15:
From state #16:
From state #17:
```

2. Write a grammar definition

Grammar for a grammar.

Variable declarations.

Method declarations.

If statements with the if ... end structure of assignment 2.

Interface declarations.

Class declarations, allowing implements ...

4. Show the run time stack.

Make up more recursive examples, for example:

Create a copy of an unordered list with all elements that are < n deleted.

In all cases, make appropriate use of recursion and var parameters, and make all methods return void.

5. Implementation of object oriented languages

Draw diagrams to show the data structures generated, and output generated for

```
class A {
     public static int a = 1, b = 2;

     public int x = 5;

     public A( int x ) {
          System.out.println( "Invoke A( " + x + " )" );
          this.x = x;
          }
     public A() {
          System.out.println( "Invoke A()" );
          a++;
          x++;
          }

     public String toString() { return "A.toString(): " + x; }
     public void set( int x ) { this.x = x; }
     public int get() { return x; }
     public String f( int i ) { return "A.f( " + i + " )"; }
     public String f( double x ) { return "A.f( " + x + " )"; }
     }

class B extends A {
     public static int a = 3, d = 4;

     public int x = 100;

     public B( int x ) {
          System.out.println( "Invoke B( " + x + " )" );
          d++;
          this.x = x;
          }

     public void set( int x ) { this.x = x; }
     public int get() { return x; }
     public String f( char c ) { return "B.f( '" + c + "' )"; }
     public String f( double x ) { return "B.f( " + x + " )"; }
     }

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

          A a1 = new A( 55 );
          A a2 = new A();
          B b1 = new B( 66 );
          A b2 = b1;

          System.out.println( "A.a = " + A.a );
          System.out.println( "A.b = " + A.b );
          System.out.println( "B.a = " + B.a );
          System.out.println( "B.d = " + B.d );
          System.out.println();

          System.out.println( "a1.x = " + a1.x );
          System.out.println( "a2.x = " + a2.x );
          System.out.println( "b1.x = " + b1.x );
          System.out.println( "b2.x = " + b2.x );
          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.f( 'A' ) = " + a1.f( 'A' ) );
          System.out.println( "b1.f( 'A' ) = " + b1.f( 'A' ) );
          System.out.println( "b2.f( 'A' ) = " + b2.f( 'A' ) );
          System.out.println();

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

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

          b1.set( 123 );
          b2.set( 456 );
          System.out.println( "b1.x = " + b1.x );
          System.out.println( "b2.x = " + b2.x );
          System.out.println( "b1.get() = " + b1.get() );
          System.out.println( "b2.get() = " + b2.get() );
          }
     }
```

What is the output:

```
class A {
     int initA() {
          System.out.println( "Init A fields" );
          return 0;
          }
     int x = initA();
     A() {
          System.out.println( "Init A default constructor" );
          }
     }

class B extends A {
     int initB() {
          System.out.println( "Init B fields" );
          return 0;
          }
     int x = initB();
     B() {
          System.out.println( "Init B default constructor" );
          }
     }

class Main {
     public static void main( String[] arg ) {
          B b = new B();
          }
     }
```