Computer Science 330 Language Implementation

Examination Information 2004

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: The examination is quite heavily related to the assignments. It is important that you do the assignments.

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

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

Consider the CUP grammar: ...

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

(12 marks)

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

(4 marks)

(c) State ... is ...

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

(8 marks)

Preparation:

Assignment 1, previous tests and examinations.

2. Write a grammar definition [16 Marks]

Write a grammar definition for .... You do not have to write any actions.

Preparation:

Assignments 2, 3, 4. Previous tests and examinations. Java grammar. Designing grammars for various constructs.

3. Interpretation [13 Marks]

Questions related to assignment 3 or 4 or chapter 8. Describe how something is implemented, etc. Pseudocode or precise English is usually the most appropriate form of answer.

Preparation:

Assignment 3, 4. Chapter 8.

4. Show the run time stack. [16 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 .... 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 ... is invoked, and the process is almost ready to return.

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.

(13 marks)

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

(3 marks)

Preparation:

Exercises in Chapter 8. Previous examinations.

5. Implementation of object oriented languages [16 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 ... within the method Main.main. Shared data structures should be drawn only once.

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

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, etc. Exercises in Chapter 9.

6. Code generation [15 Marks]

Use the assignment 4 OBJECT6 program in the appendix for this question.

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

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

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.

Preparation:

Assignment 4, especially interfaces. Last part of chapter 9. Write simple code, especially related to variables and method declaration/invocation, value and var parameters, and see what they translate into.

Exercises

1a. Bottom Up LALR(1) Parsing

Take the grammar from assignment 2, use CUP to generate the verbose output, containing the action and goto tables.

Take some sample input, and perform a parse, and build a tree.

Modify the grammar to change it from left recursive to right recursive, and see how it changes things.

1b. 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:
```

Perform an LALR(1) parse of the input

```
int f( int a; char b, c; ... )
```

State 4 is

```
lalr_state [4]: {
   [MethodHeader ::= Type IDENT (*) LEFT FormalParamDeclList RIGHT , {EOF }]
}
transition on LEFT to state [5]
```

Derive state 5 = goto(state 4, LEFT).

2. Write a grammar definition

Grammar for a grammar.

Variable declarations.

Method declarations.

Expressions involving the new operators for assignment 3.

If statements with the if ... end structure of assignment 3.

Interface declarations.

Class declarations, allowing implements ...

3. Interpretation

Describe the algorithms used to implement the various features required for assignments 3 and 4.

4a. Show the run time stack.

```
type List = struct( int value; List next );

void printList( List source ) {
     print( "{ " );
     while ( source != null ) {
          print( source.value );
          source = source.next;
          if ( source != null )
               print( ", " );
          };
     println( " }" );
     };

void concatList( int level; List source1, source2; var List dest ) {
     if ( source1 == null ) {
          dest = source2;
               // Inside the above invocation
          }
     else {
          dest = new List{ source1.value, null };
          concatList( level + 1, source1.next, source2, dest.next );
               // Inside the above invocation
          }
     };

List source1, source2, a2, a4, a7, a9, dest;
a9 = new List{ 9, null };
a7 = new List{ 7, a9 };
a4 = new List{ 4, null };
a2 = new List{ 2, a4 };
source1 = a7;
source2 = a2;
concatList( 0, source1, source2, dest );  // Inside this invocation
printList( source1 );
printList( source2 );
printList( dest );
```

4b. Show the run time stack.

```
type List = struct( int value; List next );

void printList( List source ) {
     print( "{ " );
     while ( source != null ) {
          print( source.value );
          source = source.next;
          if ( source != null )
               print( ", " );
          };
     println( " }" );
     };

void reverseTransferList( int level; List source; var List dest ) {
     if ( source == null ) {
               // At this point
          }
     else {
          dest = new List{ source.value, dest };
          reverseTransferList( level + 1, source.next, dest );
               // Inside the above invocation
          }
     };

List source, a2, a4, a7, a9, dest;
a9 = new List{ 9, null };
a7 = new List{ 7, a9 };
a4 = new List{ 4, a7 };
a2 = new List{ 2, a4 };
source = a2;
reverseTransferList( 0, source, dest );   // Inside this invocation
printList( source );
printList( dest );
```

4c. Show the run time stack.

Make up more recursive examples, including:

Create a copy of the first n elements of a list.

For example, if source is { 2, 4, 6, 7, 9, 10, 12, 15 }
`head( 5, source, dest )`

might make dest { 2, 4, 6, 7, 9 }.

Return the tail of a list, with the first n elements deleted.

For example, if source is { 2, 4, 6, 7, 9, 10, 12, 15 }
`tail( 5, source, dest )`

might make dest { 10, 12, 15 }.

Search for a specific element of a list and create a copy of the list with that element deleted.

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();
          }
     }
```

6. Code generation

Suppose we have

```
interface  X
     begin
          []char  toString();
          Class  getClass();
          bool  equals( Object other; );
          int  f();
          int  g( int a; int b,c; );
     end

class  A  implements  X
     begin
          instance
               int y = 3;
               int  e()
                    begin
                         println( "Invoke  A.e" );
                    end
               int g( int a; int b, c; )
                    begin
                         println( "Invoke A.g( " + a + ", " + b + ", " + c + " )"
);
                    end
               int  f()
                    begin
                         println( "Invoke  A.f" );
                    end
     end
```

Indicate the data structures and code likely to be generated for the declarations and statements

```
A a = new A;
X x = a;
X y = x;
A b = cast( A ) x;
a.g( 3, 4, 5 );
x.g( 6, 7, 8 );
b.y = a.y + 3;
```

Use simple symbolic names for variables.

福

Bruce Hutton