Computer Science 330 Language Implementation

Confirmed Examination Information 2003

This Examination is out of 100 Marks. Attempt **ALL** questions. Write your answers in the spaces provided in the booklet.

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

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

Consider the CUP grammar: ...

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

(10 marks)

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

(3 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.

(10 marks)

2. Write a grammar definition [20 Marks]

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

3. Interpretation [15 Marks]

Questions related to an assignment or chapter 8. Describe how something is implemented, etc.

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 .... 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.

(12 marks)

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

(3 marks)

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 ... within the method Main.main. Shared data structures should be drawn only once.

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 ...

Indicate the assembly language likely to be generated by the line ...

(3 marks)

Indicate the assembly language likely to be generated by the line ...

(10 marks)

Use symbolic names ... for the offsets of the corresponding formal parameter from the base of the relevant register and offsets of the corresponding instance field or method from the base of the relevant table. Add comments to explain the purpose of your code.

Note: An appendix is provided with common Alpha instructions.

Some Exercises

1 Try writing a grammar for

- Java function declarations.
- Java variable declarations (e.g., exam 1999).
- Java types.
- Java variables.
- Regular expressions.
- Grammars.
- Question 2 2000 test.

2 Consider the grammar:

```
terminal
     IDENT, LEFT, RIGHT, EQUALS, COMMA, INTCONST, WHERE, ERROR;

non terminal
     Program, Expr, ExprSeq, ExprSeqOpt, DefnSeq, Defn, IdentSeq, IdentSeqOpt,
WhereOpt;

start with Program;

Program::=
           Expr WhereOpt
     ;

Expr::=
           IDENT
     |
           INTCONST
     |
           IDENT LEFT ExprSeqOpt RIGHT
     ;

ExprSeq::=
           Expr
     |
           ExprSeq COMMA Expr
     ;

ExprSeqOpt::=
           ExprSeq
     |
           /* Empty */
     ;

DefnSeq::=
           Defn
           |
           DefnSeq COMMA Defn
     ;

Defn::=
           IDENT EQUALS Expr
     |
           IDENT LEFT IdentSeqOpt RIGHT EQUALS Expr
     ;

IdentSeq::=
           IDENT
     |
           IdentSeq COMMA IDENT
     ;
IdentSeqOpt::=
           IdentSeq
     |
           /* Empty */
     ;

WhereOpt::=
           WHERE DefnSeq
     |
           /* Empty */
     ;
```

Derive some of the the LALR(1) states and parsing table.

Perform parsing for suitable inputs.

```
===== Terminals =====
EOF 0 INTCONST 7 LEFT 3 ERROR 9 error 1
COMMA 6 WHERE 8 IDENT 2 RIGHT 4 EQUALS 5


===== Non terminals =====
$START 0 Program 1 ExprSeq 3 Defn 6 WhereOpt 9
ExprSeqOpt 4 IdentSeqOpt 8 Expr 2 IdentSeq 7 DefnSeq 5


===== Productions =====
18: WhereOpt ::=
17: WhereOpt ::= WHERE DefnSeq
16: IdentSeqOpt ::=
15: IdentSeqOpt ::= IdentSeq
14: IdentSeq ::= IdentSeq COMMA IDENT
13: IdentSeq ::= IDENT
12: Defn ::= IDENT LEFT IdentSeqOpt RIGHT EQUALS Expr
11: Defn ::= IDENT EQUALS Expr
10: DefnSeq ::= DefnSeq COMMA Defn
9: DefnSeq ::= Defn
8: ExprSeqOpt ::=
7: ExprSeqOpt ::= ExprSeq
6: ExprSeq ::= ExprSeq COMMA Expr
5: ExprSeq ::= Expr
4: Expr ::= IDENT LEFT ExprSeqOpt RIGHT
3: Expr ::= INTCONST
2: Expr ::= IDENT
1: Program ::= Expr WhereOpt
0: $START ::= Program EOF

===== Viable Prefix Recognizer =====
START lalr_state [0]: {
  [Expr ::= (*) IDENT , {EOF WHERE }]
  [Expr ::= (*) IDENT LEFT ExprSeqOpt RIGHT , {EOF WHERE }]
  [Program ::= (*) Expr WhereOpt , {EOF }]
  [Expr ::= (*) INTCONST , {EOF WHERE }]
  [$START ::= (*) Program EOF , {EOF }]
}
transition on Program to state [4]
transition on Expr to state [3]
transition on INTCONST to state [2]
transition on IDENT to state [1]

-------------------
lalr_state [1]: {
  [Expr ::= IDENT (*) , {EOF RIGHT COMMA WHERE }]
  [Expr ::= IDENT (*) LEFT ExprSeqOpt RIGHT , {EOF RIGHT COMMA WHERE }]
}
transition on LEFT to state [24]

-------------------
lalr_state [2]: {
  [Expr ::= INTCONST (*) , {EOF RIGHT COMMA WHERE }]
}

-------------------
lalr_state [3]: {
  [WhereOpt ::= (*) , {EOF }]
  [Program ::= Expr (*) WhereOpt , {EOF }]
  [WhereOpt ::= (*) WHERE DefnSeq , {EOF }]
}
transition on WhereOpt to state [7]
transition on WHERE to state [6]

-------------------
```

```
lalr_state [4]: {
  [$START ::= Program (*) EOF , {EOF }]
}
transition on EOF to state [5]

-------------------
lalr_state [5]: {
  [$START ::= Program EOF (*) , {EOF }]
}

-------------------
lalr_state [6]: {
  [DefnSeq ::= (*) DefnSeq COMMA Defn , {EOF COMMA }]
  [Defn ::= (*) IDENT LEFT IdentSeqOpt RIGHT EQUALS Expr , {EOF COMMA }]
  [DefnSeq ::= (*) Defn , {EOF COMMA }]
  [WhereOpt ::= WHERE (*) DefnSeq , {EOF }]
  [Defn ::= (*) IDENT EQUALS Expr , {EOF COMMA }]
}
transition on DefnSeq to state [10]
transition on Defn to state [9]
transition on IDENT to state [8]

-------------------
lalr_state [7]: {
  [Program ::= Expr WhereOpt (*) , {EOF }]
}

-------------------
lalr_state [8]: {
  [Defn ::= IDENT (*) LEFT IdentSeqOpt RIGHT EQUALS Expr , {EOF COMMA }]
  [Defn ::= IDENT (*) EQUALS Expr , {EOF COMMA }]
}
transition on EQUALS to state [14]
transition on LEFT to state [13]

-------------------
lalr_state [9]: {
  [DefnSeq ::= Defn (*) , {EOF COMMA }]
}

-------------------
lalr_state [10]: {
  [DefnSeq ::= DefnSeq (*) COMMA Defn , {EOF COMMA }]
  [WhereOpt ::= WHERE DefnSeq (*) , {EOF }]
}
transition on COMMA to state [11]

-------------------
lalr_state [11]: {
  [DefnSeq ::= DefnSeq COMMA (*) Defn , {EOF COMMA }]
  [Defn ::= (*) IDENT LEFT IdentSeqOpt RIGHT EQUALS Expr , {EOF COMMA }]
  [Defn ::= (*) IDENT EQUALS Expr , {EOF COMMA }]
}
transition on Defn to state [12]
transition on IDENT to state [8]

-------------------
lalr_state [12]: {
  [DefnSeq ::= DefnSeq COMMA Defn (*) , {EOF COMMA }]
}

-------------------
lalr_state [13]: {
  [IdentSeqOpt ::= (*) , {RIGHT }]
  [IdentSeq ::= (*) IDENT , {RIGHT COMMA }]
  [Defn ::= IDENT LEFT (*) IdentSeqOpt RIGHT EQUALS Expr , {EOF COMMA }]
```

```
   [IdentSeqOpt ::= (*) IdentSeq , {RIGHT }]
   [IdentSeq ::= (*) IdentSeq COMMA IDENT , {RIGHT COMMA }]
}
transition on IdentSeq to state [18]
transition on IdentSeqOpt to state [17]
transition on IDENT to state [16]

-------------------
lalr_state [14]: {
   [Expr ::= (*) IDENT , {EOF COMMA }]
   [Expr ::= (*) IDENT LEFT ExprSeqOpt RIGHT , {EOF COMMA }]
   [Defn ::= IDENT EQUALS (*) Expr , {EOF COMMA }]
   [Expr ::= (*) INTCONST , {EOF COMMA }]
}
transition on Expr to state [15]
transition on INTCONST to state [2]
transition on IDENT to state [1]

-------------------
lalr_state [15]: {
   [Defn ::= IDENT EQUALS Expr (*) , {EOF COMMA }]
}

-------------------
lalr_state [16]: {
   [IdentSeq ::= IDENT (*) , {RIGHT COMMA }]
}

-------------------
lalr_state [17]: {
   [Defn ::= IDENT LEFT IdentSeqOpt (*) RIGHT EQUALS Expr , {EOF COMMA }]
}
transition on RIGHT to state [21]

-------------------
lalr_state [18]: {
   [IdentSeqOpt ::= IdentSeq (*) , {RIGHT }]
   [IdentSeq ::= IdentSeq (*) COMMA IDENT , {RIGHT COMMA }]
}
transition on COMMA to state [19]

-------------------
lalr_state [19]: {
   [IdentSeq ::= IdentSeq COMMA (*) IDENT , {RIGHT COMMA }]
}
transition on IDENT to state [20]

-------------------
lalr_state [20]: {
   [IdentSeq ::= IdentSeq COMMA IDENT (*) , {RIGHT COMMA }]
}

-------------------
lalr_state [21]: {
   [Defn ::= IDENT LEFT IdentSeqOpt RIGHT (*) EQUALS Expr , {EOF COMMA }]
}
transition on EQUALS to state [22]

-------------------
lalr_state [22]: {
   [Defn ::= IDENT LEFT IdentSeqOpt RIGHT EQUALS (*) Expr , {EOF COMMA }]
   [Expr ::= (*) IDENT , {EOF COMMA }]
   [Expr ::= (*) IDENT LEFT ExprSeqOpt RIGHT , {EOF COMMA }]
   [Expr ::= (*) INTCONST , {EOF COMMA }]
}
transition on Expr to state [23]
transition on INTCONST to state [2]
transition on IDENT to state [1]

-------------------
lalr_state [23]: {
  [Defn ::= IDENT LEFT IdentSeqOpt RIGHT EQUALS Expr (*) , {EOF COMMA }]
}

-------------------
lalr_state [24]: {
  [ExprSeqOpt ::= (*) , {RIGHT }]
  [ExprSeq ::= (*) Expr , {RIGHT COMMA }]
  [Expr ::= (*) IDENT , {RIGHT COMMA }]
  [Expr ::= IDENT LEFT (*) ExprSeqOpt RIGHT , {EOF RIGHT COMMA WHERE }]
  [ExprSeqOpt ::= (*) ExprSeq , {RIGHT }]
  [Expr ::= (*) IDENT LEFT ExprSeqOpt RIGHT , {RIGHT COMMA }]
  [ExprSeq ::= (*) ExprSeq COMMA Expr , {RIGHT COMMA }]
  [Expr ::= (*) INTCONST , {RIGHT COMMA }]
}
transition on Expr to state [27]
transition on INTCONST to state [2]
transition on ExprSeqOpt to state [26]
transition on IDENT to state [1]
transition on ExprSeq to state [25]

-------------------
lalr_state [25]: {
  [ExprSeqOpt ::= ExprSeq (*) , {RIGHT }]
  [ExprSeq ::= ExprSeq (*) COMMA Expr , {RIGHT COMMA }]
}
transition on COMMA to state [29]

-------------------
lalr_state [26]: {
  [Expr ::= IDENT LEFT ExprSeqOpt (*) RIGHT , {EOF RIGHT COMMA WHERE }]
}
transition on RIGHT to state [28]

-------------------
lalr_state [27]: {
  [ExprSeq ::= Expr (*) , {RIGHT COMMA }]
}

-------------------
lalr_state [28]: {
  [Expr ::= IDENT LEFT ExprSeqOpt RIGHT (*) , {EOF RIGHT COMMA WHERE }]
}

-------------------
lalr_state [29]: {
  [Expr ::= (*) IDENT , {RIGHT COMMA }]
  [Expr ::= (*) IDENT LEFT ExprSeqOpt RIGHT , {RIGHT COMMA }]
  [ExprSeq ::= ExprSeq COMMA (*) Expr , {RIGHT COMMA }]
  [Expr ::= (*) INTCONST , {RIGHT COMMA }]
}
transition on Expr to state [30]
transition on INTCONST to state [2]
transition on IDENT to state [1]

-------------------
lalr_state [30]: {
  [ExprSeq ::= ExprSeq COMMA Expr (*) , {RIGHT COMMA }]
}

-------------------
-------- ACTION_TABLE --------
```

```
From state #0
     IDENT:SHIFT(1)  INTCONST:SHIFT(2)
From state #1
     EOF:REDUCE(2)  LEFT:SHIFT(24)  RIGHT:REDUCE(2)
     COMMA:REDUCE(2)  WHERE:REDUCE(2)
From state #2
     EOF:REDUCE(3)  RIGHT:REDUCE(3)  COMMA:REDUCE(3)
     WHERE:REDUCE(3)
From state #3
     EOF:REDUCE(18)  WHERE:SHIFT(6)
From state #4
     EOF:SHIFT(5)
From state #5
     EOF:REDUCE(0)
From state #6
     IDENT:SHIFT(8)
From state #7
     EOF:REDUCE(1)
From state #8
     LEFT:SHIFT(13)  EQUALS:SHIFT(14)
From state #9
     EOF:REDUCE(9)  COMMA:REDUCE(9)
From state #10
     EOF:REDUCE(17)  COMMA:SHIFT(11)
From state #11
     IDENT:SHIFT(8)
From state #12
     EOF:REDUCE(10)  COMMA:REDUCE(10)
From state #13
     IDENT:SHIFT(16)  RIGHT:REDUCE(16)
From state #14
     IDENT:SHIFT(1)  INTCONST:SHIFT(2)
From state #15
     EOF:REDUCE(11)  COMMA:REDUCE(11)
From state #16
     RIGHT:REDUCE(13)  COMMA:REDUCE(13)
From state #17
     RIGHT:SHIFT(21)
From state #18
     RIGHT:REDUCE(15)  COMMA:SHIFT(19)
From state #19
     IDENT:SHIFT(20)
From state #20
     RIGHT:REDUCE(14)  COMMA:REDUCE(14)
From state #21
     EQUALS:SHIFT(22)
From state #22
     IDENT:SHIFT(1)  INTCONST:SHIFT(2)
From state #23
     EOF:REDUCE(12)  COMMA:REDUCE(12)
From state #24
     IDENT:SHIFT(1)  RIGHT:REDUCE(8)  INTCONST:SHIFT(2)
From state #25
     RIGHT:REDUCE(7)  COMMA:SHIFT(29)
From state #26
     RIGHT:SHIFT(28)
From state #27
     RIGHT:REDUCE(5)  COMMA:REDUCE(5)
From state #28
     EOF:REDUCE(4)  RIGHT:REDUCE(4)  COMMA:REDUCE(4)
     WHERE:REDUCE(4)
From state #29
     IDENT:SHIFT(1)  INTCONST:SHIFT(2)
From state #30
     RIGHT:REDUCE(6)  COMMA:REDUCE(6)
------------------------------
```

```
-------- REDUCE_TABLE --------
From state #0:
     Program:GOTO(4)
     Expr:GOTO(3)
From state #1:
From state #2:
From state #3:
     WhereOpt:GOTO(7)
From state #4:
From state #5:
From state #6:
     DefnSeq:GOTO(10)
     Defn:GOTO(9)
From state #7:
From state #8:
From state #9:
From state #10:
From state #11:
     Defn:GOTO(12)
From state #12:
From state #13:
     IdentSeq:GOTO(18)
     IdentSeqOpt:GOTO(17)
From state #14:
     Expr:GOTO(15)
From state #15:
From state #16:
From state #17:
From state #18:
From state #19:
From state #20:
From state #21:
From state #22:
     Expr:GOTO(23)
From state #23:
From state #24:
     Expr:GOTO(27)
     ExprSeq:GOTO(25)
     ExprSeqOpt:GOTO(26)
From state #25:
From state #26:
From state #27:
From state #28:
From state #29:
     Expr:GOTO(30)
From state #30:
----------------------------
```

3 Go through the source code for the BASIC and OBJECT3 assignments and understand it.

4 Take the exercises in chapter 8 related to showing the stack frames, and change the data a bit, and redo the exercises.

5 Do the exercises at the back of chapter 9.

6 Write simple OBJECT3 programs, and see what code they generate.

Bruce Hutton