PDPM INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING JABALPUR

CS410: Compiler Design

Full Marks: 60 End Semester Exam 2017 Time: 3 Hrs.

Problem 1 [10]: Propose a Syntax-Directed Translation scheme that can translate **infix** arithmetic expressions into **postfix** notations using the following context free grammar. Your solution should include semantic rules using a single synthesized attribute '*string*'. Show the application of your scheme with an annotated parse tree for the input "3*4+5*2".

Context free grammar in which E is the start symbol.

(1) E → E1 + T (2) E → T (3) T → T1 * F

(4) T → F (5) F → (E) (6) F → num

Problem 2 [20]: Given the grammar below (already augmented with production (0)), answer the following questions:

(0) S → Stmts $ (1) Stmts → Stmt (2) Stmts → Stmts ; Stmt

(3) Stmt → Var = E (4) Var → id [E] (5) Var → id

(6) E → id (7) E → (E)

a) Construct the set of LR(0) items.

b) Construct the LR(0) parsing table and determine if this grammar is LR(0). Justify.

c) Is this grammar SLR(1)? Justify by constructing its table.

e) Construct the set of LR(1) items.

f) Construct the LR(1) parsing table and determine if this grammar is LR(1). Justify.

g) Construct the LALR(1) parsing table for this grammar?

Problem 3 [10]: Consider the grammar and rules given below for array address translation and generating 3 address codes for array references:

E → E1 + E2	{E.addr = new temp( ); gen(E.addr '=' E1.addr '+' E2.addr);}
| id	{E.addr = id.lexeme; }
| L	{E.addr = new temp( ); gen(E.addr '=' L.array.basename '[' L.addr ']'); }
L → id [E]	{L.array = id.lexeme; L.type = L.array.typeofelement;
	L.addr=new temp(); gen(L.addr '=' E.addr '*' L.type.width);}
| L1 [E]	{L.array = L1.array; L.type = L1.type.typeofelement;
	t = newtemp(); L.addr = new temp();
	gen(t '=' E.addr '*' L.type.width); gen(L.addr '=' L1.addr '+' t); }

Function temp() returns a new temporary name. L.array.basename gives the name of the array. L.array.typeofelement denotes type of the element of the array. L.type.width gives width of L.type. Assume that the width of an integer is 4 bytes, and lowest numbered array element is 0. Let A denotes a 2×3 arrays of integers. Construct an annotated parse tree for

the expression C+A[i][j] and show the 3-address code sequence generated for the expression where i and j are integers.

Problem 4 [10]: Using the translation scheme for Backpatching Boolean Expressions given below, construct the annotated parse tree for the following Boolean expressions and generate the three address code: Assume that instruction number starts at 100.

a. (a==b && c==d) && e==f

b. (a==b || c==d)||e==f)

Translation Scheme for Backpatching Boolean Expressions

1) *B → B1 || MB2* { backpatch (B1.falselist, M.instr);
B.truelist = merge (B1.truelist, B2.truelist);
B.falselist = B2.falselist; }

2) B → B1 && MB2 { backpatch (B1.truelist, M.instr);
B.truelist = B2.truelist;
B.falselist = merge (B1.falselist, B2.falselist); }

3) B → (B1) { B.truelist = B1.truelist; B.falselist = B1.falselist; }

4) B → E1 rel E2 { B.truelist = makelist (nextinstr);
B.falselist = makelist (nextinstr + 1);
gen ('if ' E1.addr rel.op E2.addr 'goto __');
gen ('goto __'); }

5) M → ε { M.instr = nextinstr ; }

Problem 5 [10]: Draw the Activation tree representing calls during an execution of the following program. Suppose that partition function picks a[m] as the separator v. Also when the array is reordered using partition, assume that the order of elements are persevered. Denote the functions main( ), quicksort( ), exchange( ) and partition( ) as m, q, e and p respectively.

```
int a[9] = {10, 32, 567, -1, 789, 3, 18, 0, -51};
main( ) {
        quicksort(0,8);
}
```

```
void exchange(int i, int j) {
int x = a[i]; a[i] = a[j]; a[j] = x;
}
```

```
int partition(int y, int z) {
 /*picks a separator value v=a[m] and partitions a[m..n] in such a way that a[m..p-1] are less than v, a[p]=v and a[p+1..n] are equal to or greater than v, Returns p.*/
}
```

```
void quicksort( int m , int n ) {
   if (n > m) {
        int i = partition(m,n);
        quicksort(m,i-1);
        quicksort(i+1,n);
        }
}
```