Roll No.: _

Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

B.Tech. Degree Examinations – March/April 2019

Sixth Semester

Computer Science and Engineering

15CSE311 Compiler Design

[Time : Three hours Maximum : 100 Marks]

CO	Course Outcomes
CO01	Understand modularization and apply theoretical concepts to compiler construction
CO02	Apply algorithms for analyzing lexemes and structural correctness of source programs.
CO03	Analyze the design of type systems and check the correctness of code semantics
CO04	Experiment intermediate representation and perform code control flow checking.
CO05	Analyze function executions using activation records

Answer all questions

PART A **(7 x 4 =28 Marks)**

1.
   a. Why might a program written in a compiled language execute faster than one written in a language that is interpreted? [CO01]
   b. The front end of a compiler consists of three parts: scanner, parser, and (static) semantics. Collectively these need to analyse the input program and decide if it is correctly formed. For each of the following properties or errors, indicate which stage of the front-end of the compiler would normally handle it. [CO01]
      i. In a variable declaration, the variable has not previously been declared in the same scope.
      ii. A comment beginning with /*is not terminated before reaching the end of the input file (i.e., no matching */found).
2. Compare and contrast the structure and working of two phase and three phase compiler models with a neat sketch. [CO01]
3. Convert the regular expression **b*(ab | ba)b*** to NFA using Thomson's Construction. [CO01]
4. Construct an AST and DAG for the following Three Address Code: [CO04]

```
1. a = b + c
2. b = a – d
3. c = b + c
4. d = a - d
```

5. Consider the following grammar: [CO03]

   N → L

   L → B | B L

   B → 0 | 1

   Write an attribute grammar for converting binary to decimal but the calculation is from left to right. Example 1010 the decimal equivalent is 5 ($1*2^0 + 0*2^1+1*2^2+0*2^3$). Draw annotated parse tree and dependency graph for the string **10110**.

6. Mention the various scopes available in object oriented languages. [CO04]
7. What is a linkage convention? Mention the purpose of linkage convention rules. Draw the standard procedure linkage components at caller and callee. [CO05]

PART B **(6 x 12 =72 Marks)**

8. Consider the grammar given below: [CO02]

```
lexp-> atom |list
atom-> num | id
list-> (seq)
seq -> seq lexp | lexp
```

   a. Remove left recursion in the above grammar. (1 Marks)
   b. Construct the first and follow sets. (4 Marks)
   c. Construct an LL(1) parsing table for the grammar. (4 Marks)
   d. Show the stack, input and action for the input **(b (2))** where a, b, c are id and 2 is num. (3 Marks)

9. Consider the following attribute grammar with the function specifications given below: [CO03]

Production	Semantic Rule
P → S	S.next = newlabel() P.code = S.code || label(S.next)
S → while B do S	S.begin = newlabel() B.true = newlabel() B.false = S.next S1.next = S.begin S.code = label(S.begin) || B.code || label(B.true) || S1.code || gen(goto S.begin)
B → id_1 < id_2	B.code = gen(if id1.place '<' id2.place goto B.true), gen(goto B.false)
S → id := E	S.code := E.code || gen (id.place ' := ' E.place)
E → E1 + E2	E.place := newtemp E.code := E1 .code || E2 .code || gen (E.place ' := ' E1 .place ' + ' E2 .place)
E → id	E.place := id.place E.code := ' '

- **newtemp** will generate new temporary variables (eg : $\mathbf{T_0}$,$\mathbf{T_1}$,..)
- **gen()**- generates three address statement
- || is the concatenation symbol
- **newlabel()** will generate new labels (eg: $\mathbf{L_1}$,$\mathbf{L_2}$,…)

Create annotated parse tree for the input code fragment:

```
while c < d do
    x = a + b
```


R

and identify the synthesized and inherited attributes. What will be the three address code generated from the above annotated parse tree.

10. For the following PASCAL code, answer the following questions: [CO04]

```
Program A( )
{
  x, y, z: integer;
  x = 1;
  procedure B( )
  {
    y: integer;
    y = 0;
    x = z + 1;
    z = y + 2;
  }
  procedure C( )
  {
     z: integer;
```

```
    procedure D( )
     {
       x : integer;
       x = z + 1;
       y = x + 1;
       call B( );
     }
     z = 5;
     call D( );
  }
  x = 10;
  y = 11;
  z = 12;
  call C( );
  print x, y, z;
}
```

Draw the symbol tables for each of the procedures in this code (including main) and show their nesting relationship using the following data structures:

a. Hierarchical static sheaf of tables. (6 Marks)

b. Open Hashing with stack. Use the hash function: h(variable) = (sum of the ascii value of all the characters in the variable name) mod 10. *(Note: Assume that finalizescope function performs deallocation.)(Hint: ASCII values : a to z is 97 to 122 and A to Z is 65 to 90)* (6 Marks)

11. Write the three-address code equivalent for the following program and convert the three address code into a control flow graph and SSA. [CO04]

```
d = 2;
n = 5;
while (d ! = 0)
{
   f = 1;
   while (n ! = 0)
   {
```

```
      f = n * f ;
      n = n – 1;
    }
   d = d – 1;
   n = f;
}
r = f;
```

12. Consider the following program: [CO05]

```
function f ()
{
  int x = 3;

  function g (int c,d)
  {
    c = c-1;
    print(c,d,x,n);
  } /* end of g */
```

```
function h ()
{
  int n = 2;
  f();
} /* end of h */
main()
{
   h();
}
```


R

/* body of f */ **g(x,x);** **} /* end of f */**	

a. Show the nesting relationship of the procedures. (2 Marks)

b. Show the call graph of the procedures. (2 Marks)

c. If lexical scoping is used, Show the static coordinates of all the variables. (2 Marks)

d. If dynamic scoping is used, Show the dynamic coordinates of all the variables. (2 Marks)

e. When static scoping is used, trace the program and display the output of the program. (2 Marks)

f. When dynamic scoping is used, trace the program and display the output of the program. (2 Marks)

13. For the following program, draw the activation record. Show the calling sequence with static and dynamic access links and the execution of the program. [CO05]

```
1. program main(input, output);
2.     procedure P1( function g(b: integer): integer);
3.     var a: integer;
4.     begin  /* p1 */
5.         a := 3;
6.         writeln(g(2))
7.     end;
8.     procedure P2;
9.     var a: integer;
10.        function F1(b: integer): integer;
11.        begin /* f1 */
12.          b := a + b;
13.            return b
14.        end;
15.       procedure P3;
16.         var a:integer;
17.         begin /* p3 */
18.             a := 7;
19.             P1(F1)
20.         end;
21.    begin /* p2 */
22.         a := 0;
23.        P3
24.    end;
25. begin /* main */
26.    P2
27. end
```

****

R