Ex/CSE/T/321/76/2010

BACHELOR OF COMPUTER SC. ENGG. EXAMINATION, 2010

(3rd Year, 2nd Semester)

COMPILER DESIGN

Time : Three hours Full Marks : 100

Answer any five questions taking at least two questions from each part.

Use separate answer scripts for two parts.

Answer all parts of a single question on contiguous pages.

PART – I

1. a) State the reasons of having an intermediate representation of the source program during compilation.

b) Given that binary number strings are read with the most significant bit first and may have leading zeros, do the following for the language consisting of binary number strings that represent decimal numbers that are multiples of 4.

(i) Derive the regular expression *r* describing the above language.

(ii) Use Thompson's construction to convert the above regular expression into an NFA.

(iii) Convert the NFA to equivalent DFA using subset construction.

c) Compare the time and space complexity for determining whether a given binary string *x* is in L(*r*) using two

[Turn Over]

approaches :

(i) Constructing NFA from *r*.

(ii) Constructing DFA from *r*. 3+(2+3+7)+5=20

2. a) Explain the necessity of runtime support environment in program execution.

b) What is an “activation record” ? Briefly describe structure of an “activation record”.

c) Consider the following C function f.

```
void f(int n){
                int *p ;
                p = (int *) malloc (n* size of (int));
                ...............
}
```

Show the content of the activation record for the call f(5). Also show how the content of the activation record changes for another call f(7).

d) Explain how addressing constraint of target machine influences storage layout of data objects.

e) Explain limitations of fully static storage allocation strategy. 3+(2+5)+4+3+3=20

3. a) State the activities to be performed by the “caller” and the “called” procedure in the “calling sequence” and “return sequence”.

b) Give output of the following program snippet using each of the four parameter passing methods call by value, call

from any programming language that is known to you.

c) How are the code optimisation techniques classified ?

d) Give an example of each of the optimisation technique mentioned below :

(i) Common sub - expression elimination

(ii) Tail recursion removal

(iii) Loop fission

(iv) Loop fusion

(v) Strength reduction

4+7+4+5=20

d) Describe how the compiler generates addresses of nonlocal names in case of programming language which support static scope with nested procedures definitions.

5+6+5+4=20

4. a) Compare heap storage allocation strategy with stack storage allocation strategy.

b) What do you mean by l - value and r - value of expressions ? Give suitable examples.

c) What is “dangling reference” ? Give one example.

d) Describe how C compiler handles variable number of arguments to functions like printf, scanf etc.

e) Show the activation record for a call to function f. Determine the offset of the data elements x, c, a[2][0][1], and y with respect to frame pointer (fp). Assume that four, four, one, and eight bytes are required for storage of integers, addresses, characters and double - precision floating point numbers. Also assume row major storage for multidimensional array.

```
int f (char c, int x [3]) {
        int a [3][2][2] ; double y ;
        ..............
}
```

4+3+3+4+6=20

PART – II

5. a) Write the sequence of three - address code instructions corresponding to each of the following expressions :

(i) a*b+a*b*c

(ii) 2+(3+(4+5))

(iii) if (v == 0) then x = y

(iv) x = add (x,y)

b) Consider the following grammer :

$E \rightarrow B\ A$

$A \rightarrow \&\ B\ A \mid \varepsilon$

$B \rightarrow \text{true} \mid \text{false}$

Construct the FIRST and FOLLOW sets for the grammar. Construct the LL(1) parsing table for the above grammer. Also show the error recovery entries in the parsing table. Write at least one string (with not less than five tokens) that can be generated from the grammar along with the leftmost and rightmost derivations of the string.

(1.5x4)+(4+4+2+4)=20

6. a) Consider the following grammar :

$S \rightarrow Aa \mid bAc \mid dc \mid bda$

$A \rightarrow d$

b) Construct the DFA of LR (0) items for this grammar.

c) Construct the SLR(1) parsing table. Is there an conflict in the parsing table ? If so then discuss the type of the conflict and reason for such conflict.

d) Construct the DFA of LALR(1) items for the above grammar. 6+6+2+6=20

7. a) What are attribute grammars ? Define synthesized and inherited attributes in attribute grammars ? Explain S - attributed and L - attributed definitions with examples.

b) Suppose we have a production A → BCD. Each of the four non - terminals A,B, C and D have two attributes : s is a synthesized attribute, and i is an inherited attribute. For each sets of rules below, tell whether (i) the rules are consistent with an S - attributed defination, (ii) the rules are consistent with an L - attributed definition.

(i) A.s = B.i + C.s

(ii) A.s = B.i + C.s and D.i = A.i + B.s

(iii) A.s = B.s + D.s

(iv) A.s = D.i, B.i = A.s + C.s, C.i = B.s and D.i = B.i + C,i

Justify your answer with appropriate representation of the rules in annotated parse tree form.

c) Consider the grammar A → AA I (A) I ε ,

Give some examples of strings generated by this grammar. Show that the grammar is ambiguous.

(2+2+4)+2x4+4=20

8. a) What is a symbol table ? What are the purposes of using a symbol table ?

b) Describe a possible implementation of the symbol table with scope rules. Explain your ideas by taking examples

by reference, call by name, and call by value result. Provide brief justifications for your answer.

```
int i = 0 ;
void p (int x, int y) {
x = x + 1 ;     i = i + 1 ;    y = y + 1 ;
}
main ( ) {
int a [2] = {2, 9};  p(a[i]);  printf("%d%d\n",i,a[0],a[1]); return(0);
}
```

c) Define static scope and dynamic scope. State advantages and disadvantages of the two methods. Determine the output of the following program snippet considering static scope and dynamic scope respectively. Provide brief justifications for your answer.

```
int x = 0
int f (void){
        return x ;
}
int g (void){
        int x = 2
        return (f( )) ;
}
main ( ){
          int x = 1 ;
          printf("%d" , g( )) ;
}
```

[Turn Over]