L-3/T-1/CSE **Date : 24/10/2019**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2018-2019

Sub : **CSE 309** (Compilers)

Full Marks : 210 Time : 3 Hours

The figures in the margin indicate full marks.

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

1. (a) Explain why a code generated by passing a source code through a compiler can be executed directly while some codes need an interpreter to be executed. **(4)**

   (b) Elucidate, with necessary examples, how separating the analysis portion of a compiler into lexical analysis and parsing (syntax analysis) phases, achieves simplicity of design. **(12)**

   (c) Analyze clearly the acceptability / unacceptability, merits / demerits of the following tokenization schemes: **(9)**

Schme	Pattern	Token
1	+ or - or * or /	< MATHOP, attribute >
2	+ or -	< PM, attribute >
	* or /	< MD, attribute >
3	+	<PLUS>
	-	<MINUS>
	*	<MULT>
	/	<DIV>

   (d) Let's a consider a language like C++ where both + and ++ constitute valid tokens. In some code of this language there are three consecutive + characters followed by a newline character. During lexical analysis with a buffer pair, both the pointers are on the leftmost + character. Describe clearly, with necessary figures, the activities that will happen starting from the present moment to the moment when the newline character will be found. **(10)**

2. (a) Write a Lex program that copies a C program, replacing each instance of the keyword float by double Showing only the necessary pattern-actions in the middle part of the program should suffice. **(7)**

   (b) Explain why a left-recursive grammar cannot be parsed using the procedure-based predictive top-down parsing algorithms. **(10)**

   (c) Describe clearly how panic-mode recovery works during parsing. **(6)**

   (d) Eliminate left recursion from the following grammar: **(12)**

   $S \rightarrow Aa \mid Bb$

   $A \rightarrow Aa \mid Abc \mid c \mid Sb$

   $B \rightarrow dA \mid bb$

3. (a) Explain, with necessary examples, how left-factoring prevents backtracking in parsing. **(8)**

Contd P/2

CSE 309

Contd..... Q. No. 3

(b) Compute FIRST and FOLLOW sets for the following grammar: **(12)**

$S \rightarrow TB$

$T \rightarrow (S) \mid a$

$A \rightarrow +S \mid TB \mid *$

$B \rightarrow AB \mid \in$

(c) (i) To find out whether a grammar G is LL(1) or not, we test this condition among other conditions, "If whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G, for no terminal a do both α and β derive strings beginning with a". State with necessary explanations, how this condition will be changed for an LL(3) grammar. **(7+8)**

(ii) To find out whether a grammar G is LL(1) or not, we test this condition as well among other conditions, "If whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G, if $\beta \overset{*}{\Rightarrow} \in$, then α does not derive ay string beginning with a terminal in FOLLOW(A). Likewise, if $\alpha \overset{*}{\Rightarrow} \in$, then β does not derive any string beginning with a terminal in FOLLOW(A)". Explain and justify this condition.

4. (a) In most of the cases, compiler steps are divided into two largely independent groups: a front end, responsible for analyzing source code, and a back end, responsible for generating target code. What is the motivation for this division into groups? Explain. How do these groups interact with the symbol table? Describe with reference to the compiler steps. **(12)**

(b) Justify, with reference to the stack contents and table entries, how a table-driven predictive parser mimics a left-most derivation. **(10)**

(c) What are the common semantic actions in yacc parser generators? Describe with necessary examples how an action is formed in yacc. What is the default semantic action? **(13)**

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE**.

All the symbols have their usual meanings unless explicitly mentioned

5. (a) Define L-attributed SDD (Syntax-Directed Definition). Give an example of a SDD which is not L-attributed. **(6+3)**

(b) What is the difference between SDD and SDT (Syntax-Directed Translation)? What is a postfix SDT? Explain with an example. **(5+4)**

Contd P/3

CSE 309

Contd..... Q. No. 5

(c) The following SDT computes the value of string of 0's and 1's interpreted as a positive, binary integer. **(7)**

$$
\begin{array}{lll}
B & \rightarrow & B_1\ 0\ \{B.val = 2 \times B_1.val\} \\
 & | & B_1\ 1\ \{B.val = 2 \times B_1.val + 1\} \\
 & | & 1\ \{B.val = 1\}
\end{array}
$$

Rewrite this SDT so that the underlying grammar is not left recursive, and yet the same value of *B.val* is computed for the entire input string.

(d) Below is a grammar for simple expressions involving '+' and '*' operators. Give an SDD to construct syntax trees for expressions represented by the given grammar. **(10)**

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow$ id

$F \rightarrow$ num

6. (a) Consider the following partially completed SDD for generating three-address codes for expressions, booleans and flow-of-control statements. **(15)**

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	*S*.next = *newlabel()* *P*.code = *S*.code || label (*S*.next)
$S \rightarrow$ if (B) S_1	*B*.true = *newlabel()* *B*.false = S_1.next = S.next *S*.code = *B*.code || *label(B.true)* || S_1.code
$B \rightarrow E_1$ **rel** E_2	*B.code* = E_1.*code* || E_2.*code* || *gen* ('if' E_1.*addr* **rel**.*op* E_2.*addr* 'go to' *B.true*) || *gen*('go to' *B.false*)
$E \rightarrow$ **id**	*E.addr* = *top.get*(**id**.*lexeme*) *E.code* = ''

CSE 309
Contd..... Q. No. 6

Here **rel** means relational operators (<, >, <=, >=, !=, ==), *top* denote the current symbol table, *top.get* retrieves the entry corresponding to the **id**, *newlabel()* creates a new label each time it is called, *label (L)* attaches label L to the next three-address instruction to be generated and *gen(.)* generates three address instruction in string format. Other symbols and attribute names have their usual meanings.

Now add rules to the SDD above for the following constructs.

(i) $S \rightarrow$ if (B) S_1 else S_2

(ii) **while** (B) S_1

(iii) $B \rightarrow B_1 || B_2$

(iv) $B \rightarrow B_1$ && B_2

(v) $E \rightarrow$ **num**

(b) Refer to the SDD in Questions 6(a) you have just completed. Now, write down the three-address code what will be generated by the SDD for the statement below. **(6)**

if (x > 20 && x < 10 || x ! = y) x = 0 else x = 1;

(c) What is the difference between quadruples and triples representations of three-address cods? Illustrate with an example three-address code. What is the problem of using Triples representation in an optimizing compiler? **(5+3)**

(d) What are two distinctive aspects that distinguish SSA (Static Single-Assignment) from three-address code? Explain with examples. **(6)**

7. (a) What is an activation record? What kinds of data are usually stored in an activation record? **(6)**

(b) Describe the following three heap allocation strategies. (i) First-fit, (ii) Best-Fit, (iii) Next-Fit. Mention one advantage for each of these three strategies. **(8)**

(c) What are the two problems with manual memory deallocation? Explain. **(6)**

(d) Each garbage collection algorithm starts with a root set. Which objects constitute the root set? Consider the network of objects in Figure 7(d). Show the steps of a mark-and-sweep garbage collector on this network with the pointer A → D deleted. Assume X and Y are members of the root set. What is the primary difference between basic mark-and-compact and Cheney's copying collector algorithms? **(2+8+5)**

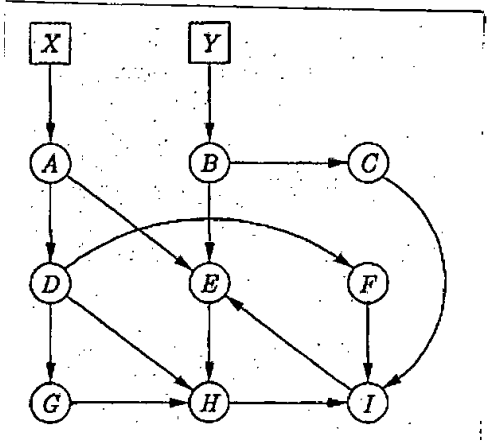


Figure for Question 7(d)

Contd P/5

CSE 309

8. (a) Convert the following three address code into machine code for the simple machine model of your text. Assume three registers are available. Make sure the registers are allocated and assigned efficiently to avoid unnecessary loads and stores. Show the register and address descriptors after each step. Assume *t*, *u* and *v* are temporary variables and the remaining variables are live on exit from the block. **(14)**

$t = a - b$

$u = a + b$

$v = t + u$

$a = d$

$d = v + c$

(b) Construct a DAG (Directed Acyclic Graph) for the following basic block. **(6)**

$d = b * c$

$e = a[i]$

$f = e + d$

$g = b * c$

$a[j] = f$

(c) Optimize the following three-address code by eliminating common subexpressions, performing reduction in strength on induction variable, and eliminating all the induction variables you can. **(7)**

```
    dp = 0
    i = 0
L:  t1 = i * 8
    t2 = A[t1]
    t3 = i * 8
    t4 = B[t3]
    t5 = t2 * t4
    dp = dp + t5
    i = i + 1
    if i < n goto L
```

(d) Explain with suitable examples how the following two semantics-preserving transformations may lead to code optimization. (i) Copy Propagation, (ii) code motion. **(8)**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2017-2018

Sub: **CSE 309** (Compiler)

Full Marks: 210 Time: 3 Hours

The figures in the margin indicate full marks

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

1. (a) Explain, with necessary examples, the tasks performed by a preprocessor in compilation. **(10)**

(b) In a certain programming language, the decimal numbers are like, 123.456E ± 789. Here, decimal part and exponent part are optional. Furthermore, the plus or minus in the exponent is optional. In formulating regular definitions for these numbers, some one came up with the following: **(15)**

number → [0 – 9] [0 – 9]*.[0 – 9] [0 – 9]*?(E + –?[0 – 9]*)

The above definition has got a number of flaws in it. Point out these flaws with necessary explanations.

(c) Explain "panic mode" in lexical analysis. What could be the side effects of panic mode? **(10)**

2. (a) In lexical analysis, the analyzer often has to look one or more characters beyond the next lexeme before it can be sure it has the right lexeme. **(12)**

A programming language has got the following tokens:

Token	Pattern
Identifiers	Starts with _ or letter followed by _ or letter or digit as many times as we want
IF	if
THEN	then
ELSE	else
ROR	>>
ROL	<<
PLUS	+
INC	++
TINC	+++
GT	>
GE	>=

Explain which of the above tokens require reading extra characters and which do not.

Contd P/2

CSE 309

Contd... Q. No. 2

(b) What would have happened if instead of buffer pairs in lexical analysis, we decided to use a single buffer keeping the pointers same as those in buffer pair? Provide necessary diagram. Can the scenario be changed by using sentinels? Explain. **(12)**

(c) The reserved words in programming languages look like identifiers and these require special measures so that these are not confused with identifiers. Describe these special measures. **(11)**

3. (a) Enumerate the principles that are adopted in Lex for conflict resolution when several prefixes of the input match one or more patterns. **(4)**

(b) Construct procedure based recursive-descent parser for the grammar given below: **(8+7)**

$$S \rightarrow Aa \mid bBb \mid c$$
$$A \rightarrow aA \mid dd$$
$$B \rightarrow b \mid eBe \mid \varepsilon$$

Now, parse the string `bebeb` using the constructed parser and show how this parser actually implements a top-down parsing.

(c) Explain why left-recursive grammars with production rules like $S \rightarrow Sa$ are not suitable for recursive-descent parser. In light of your answer, what do you think will happen in case of production rules like $S \rightarrow aSa$? **(6+4)**

(d) Left factor the following grammar: **(3+3)**

$$S \rightarrow abA \mid abcS$$
$$A \rightarrow aA \mid \varepsilon$$

In terms of the k in LL(k), what is the value of k for the above grammar?

4. (a) Compute the FIRST and FOLLOW sets for the nonterminals of the following grammar and use them to construct a parsing table of a top-down predictive parser for this grammar. **(3+3+4)**

$$S \rightarrow (A) \mid a$$
$$A \rightarrow SB$$
$$B \rightarrow +SB \mid \varepsilon$$

(b) Eliminate left-recursion from the following grammar: **(16)**

$$S \rightarrow Sc \mid Ta \mid b$$
$$T \rightarrow Tc \mid Sd \mid Ub \mid d$$
$$U \rightarrow Ua \mid Td \mid Sb \mid b$$

(c) Among other reasons, a grammar can not be LL(1), if, whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G, the following condition does *not* hold: **(9)**

"If $\beta \overset{*}{\Rightarrow} \varepsilon$, then α does not derive any string beginning with a terminal in FOLLOW(A). Likewise, for $\alpha \overset{*}{\Rightarrow} \varepsilon$, β does not derive any string beginning with a terminal in FOLLOW(A)". Why? Explain with necessary examples.

Contd P/3

CSE 309

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE**.

5. (a) Look at the following, Syntax Directed Definition (SDD). **(15)**

	PRODUCTION	SEMANTIC RULES
1)	$T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow$ digit	$F.val =$ digit.*lexval*

This SDD can evaluate the value of an expression in the attribute val. However, it cannot handle addition/subtractions. For example, the given SDD will successfully handle the expression "2*3*4" but fail for the expression "2+3*4". Extend the SDD so that it is capable of handling addition/subtraction operators (+ and –) and consequently, is able to evaluate expressions with arbitrary number of addition, subtraction and multiplication operators.

(b) For the input string "2+3*4+5", draw the annotated parse tree with the SDD that you have just written in question 5(a). **(10)**

(c) Write down the semantic rules for the productions below so that equivalent short circuit code is generated correctly. **(10)**

(i) $S \rightarrow if\ (B)\ S_1\ else\ S_2$

(ii) $B \rightarrow B_1\ \&\&\ B_2$

Assume that jumping labels are managed using inherited attributes *B.true, B.false* and *S.next*; where *B* is a Boolean expression and *S* is a statement.

6. (a) Notice the following SDD. **(15)**

PRODUCTION	SEMANTIC RULES
$S \rightarrow$ **id** = E ;	$S.code = E.code \mid\mid$ *gen*(*top.get*(**id**.*lexeme*) '=' $E.addr$)
$E \rightarrow E_1 + E_2$	$E.addr =$ **new** *Temp*() $E.code = E_1.code \mid\mid E_2.code \mid\mid$ *gen*($E.addr$ '=' $E_1.addr$ '+' $E_2.addr$)
| $- E_1$	$E.addr =$ **new** *Temp*() $E.code = E_1.code \mid\mid$ *gen*($E.addr$ '=' 'minus' $E_1.addr$)
| (E_1)	$E.addr = E_1.addr$ $E.code = E_1.code$
| **id**	$E.addr =$ *top.get*(**id**.*lexeme*) $E.code = ''$

Contd P/4

CSE 309

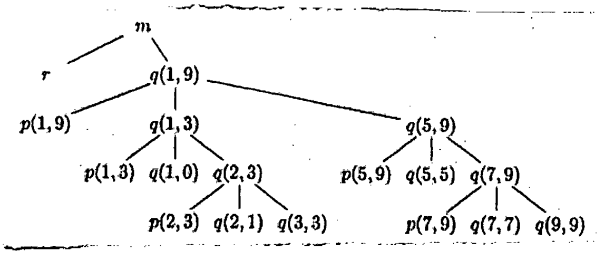
Contd... Q. No. 6(a)

This is used for generating machine code to an arithmetic expression. Now, for the following expression, draw the annotated parse tree. Annotate each internal node.

$$a = -(b) + c;$$

Finally, write the code that has been generated by SDD separately.

(b) Notice the following activation tree. **(10)**



When q(3,3) is being run by the processor,

(i) What are the live activations?

(ii) What activations have already been processed?

(c) Determine liveliness and next-use information for the following basic block. **(10)**

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

7. (a) Convert the following code into flow graph: **(15)**

```
1)  i = 1
2)  j = 1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)
```

Contd P/5

CSE 309

Contd... Q. No. 7

(b) Give examples of – **(10)**

(i) Copy propagation

(ii) Code motion

(c) What is the output of the following code? Justify with a proper data structure. **(10)**

```
public class Base {
    public int publicBaseInt = 1;
    protected int baseInt = 2;
}

public class Derived extends Base {
    public int derivedInt = 3;
    public int publicBaseInt = 4;

    public void doSomething() {
        System.out.println(publicBaseInt);
        System.out.println(baseInt);
        System.out.println(derivedInt);

        int publicBaseInt = 6;
        System.out.println(publicBaseInt);
    }
}
```

8. (a) What is the output of the following code segment if– **(15)**

(i) The program names are handled in compilation time?

(ii) The program names are handled in runtime?

```
int x = 137;
int y = 42;
void Function1() {
    Print(x + y);
}
void Function2() {
    int x = 0;
    Function1();
}
void Function3() {
    int y = 0;
    Function2();
}
Function1();
Function2();
Function3();
```

(b) What is the one-core problem of the garbage collection method reference counting? Give an example. **(10)**

(c) For the stop-and-copy generational garbage collectors, we cannot simply take bitwise copies of objects. Explain the reason. Also, explain how this issue is handled using necessary figures. **(10)**

L-3/T-1/CSE **Date : 13/08/2017**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2016-2017

Sub : **CSE 309** (Compiler)

Full Marks: 210 Time : 3 Hours

USE SEPARATE SCRIPTS FOR EACH SECTION

The figures in the margin indicate full marks.

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

Symbols have their usual meanings.

1. (a) Does left factoring a grammar helps in removing ambiguity of that grammar? Explain your answer with examples. **(9)**

   (b) James Bond was one of the best spies in the history of British Secret Intelligence Service MI6. MI6 are searching for their next James Bond. To check a candidate potential, MI6 are giving the Flex code named "JamesBondLex.1" (Given in the figure for Question 1(b)) to the candidate to explore. They are asking the candidate to write the contents of a text file named "DecByPotentialNextJamesBond.txt", that is produced when the generated scanner from the Flex code is run over the input file named "EncByMI6.txt" (Given in the figure for Question 1(b)). **(20)**

JamesBondLex.l

```
%option noyywrap
%s JESPYSTATE
%{
#include<stdio.h>
#include<stdlib.h>
FILE *decOut;
%}
whitespace [ \t\n]
JamesBond [JamesBond]
AlphaNumeric [a-zA-Z0-9$]
%%
[James]{3} { fprintf(decOut,"<James,%s>\r\n",yytext);}
Bond { fprintf(decOut,"<Bond,%s>\r\n",yytext); }
{JamesBond}+ { fprintf(decOut,"<JamesBond,%s>\r\n",yytext); }
(best)* { fprintf(decOut,"<(best)*,%s>\r\n",yytext); }
best* { fprintf(decOut,"<best*,%s>\r\n",yytext); }
(Johnny|Eng) { BEGIN JESPYSTATE;
               fprintf(decOut,"<JESPYBEGIN,%s>\r\n",yytext);}
<JESPYSTATE>English { fprintf(decOut,"<JESPYS,%s>\r\n",yytext); }
<JESPYSTATE>("just"|"average") { fprintf(decOut,"<TrCompare,%s>\r\n",yytext); }
<JESPYSTATE>{whitespace}* { }
<JESPYSTATE>[^-just \t\naverage]* { BEGIN INITIAL;
                                   fprintf(decOut,"<JESPYEND,%s>\r\n",yytext); }

{AlphaNumeric}* { fprintf(decOut,"<AN,%s>\r\n",yytext); }

<JESPYSTATE>. { }
. { }

%%
int main(int argc,char *argv[])
{

        if(argc!=2)
        {
                printf("Please provide input
file name and try again\n");
                return 0;
        }

        FILE *fin=fopen(argv[1],"r");
        if(fin==NULL)
        {
                printf("Cannot open the
specified file\n");
                return 0;
        }
        decOut=
fopen("DecByPotentialNextJamesBond.txt","w");

        yyin= fin;
        yylex();
        fclose(yyin);
        fclose(decOut);
        return 0;
}
```

Contd P/2

CSE 309

Contd... Q. No. 1(b)

EncByMI6.txt
JamesBond JamesBond British spy…
Johnny-English-just average pBritish spy…

Figure for Question 1 (b)

Now, write down the correct and complete answer that a potential candidate should provide in reply to MI16's tricky (!) question.

(c) Suppose, Kishor is developing a secret complier. He has named it "Tin Goyenda Complier". He wants to allow the complier to take floating point indexes for arrays. In which step of his development, he has to take care of it? **(6)**

2. (a) Eliminate left-recursion from the following grammar: **(18)**

$K \rightarrow$ *KungfuPanda* / *Kungfu Furious5* / O_w*o gway*
$F \rightarrow$ *KFT* / *KFM* / KFM_a / *KFV* / *KFC*
$O_w \rightarrow$ *KFmasterofmaster* / *FKshifumaster* / O_w*dragonKF* / *warriorKF*
$P \rightarrow$ *Po* / $\in$
$T \rightarrow$ *Tigress* / $\in$
$M \rightarrow$ *Monkey*
$M_a \rightarrow$ *f* M_a*antis*
$V \rightarrow$ *f Viper*
$C \rightarrow$ *f Crane*

(b) Whether a lexical analyzer will ne able to handle the error in the following C/C++ statement or not? Justify your answer. **(5)**

$$fi(a == f(x))\{a+ = 1;\}$$

(c) "If there is a production **A** $\rightarrow \alpha$**B**γ where FIRST (γ) contains $\in$, then everything in FOLLOW(**A**) is in FOLLOW(**B**)." – Justify this statement using illustrative examples. **(12)**

3. (a) Consider the following grammar to answer the following questions: **(8+8+4=20)**

$T \rightarrow$ *hreeIdiotsTF* / *virus*
$F \rightarrow$ *farhanT* / *rajuT* / $\in$
$I \rightarrow$ *rancho*

(i) Compute the FIRST and FOLLOW of all non-terminals in the given grammar.

(ii) Construct the corresponding LL parsing table.

(iii) State whether the given grammar is suitable to be parsed using the predictive parsing method or not. Use the previously constructed LL parsing table to give your answer. (Assume ***T*** is the starting state of the grammar)

Contd P/3

CSE 309

Contd... Q. No. 3

(b) Explain the terms linker and loader. For C/C++ programs, elaborate the relationship of these with the followings: **(2×5=10)**

(i) header files

(ii) macros

(iii) libraries

(c) Why intermediate code is generated between the front and back end of a complier? Explain with examples. **(5)**

4. (a) When does a lexical analyzer use Panic recovery mode? State three other approaches for lexical error-recovery. **(4+3=7)**

(b) Describe buffer pairs with sentinels? What will be the, maximum length of a lexeme that a lexer can handle if the size of each buffer in the buffer pair is N? Explain with examples. **(5+2+5=12)**

(c) 'Bassett Events' is one of the famous event planning companies of the world. It has two types of managers who organize events in different ways. Managers of first kind have such employees who perform the job only after the managers have given them necessary instructions related to organizing. This type of mangers usually hears the whole plan from their clients, discusses with their employees, provides related instructions and then assures the occurrence of the event. On the other hand, second type of managers organize the event step by step listening to their clients, discuss with their employees to check whether it is possible or not and makes the job of that particular step done if it is possible. **Now answer the followings.** **(4+6+6=16)**

(i) Find similarities between these two type of managers with Complier and Interpreter.

(ii) Give a brief comparison of the compilation and running process of a C, Python and Java program.

(iii) Give a brief comparison of Compiler and Interpreter in respect to performance and error diagnostic.

SECTION-B

There are **FOUR** questions in this section. Answer any **THREE** questions.

5. (a) Write down the general strategy to implement any Syntax Directed Translation Scheme. Illustrate with an example. **(10)**

Contd P/4

CSE 309

Contd... Q. No. 5

(b) What is the difference between parse tree and abstract syntax tree? Write down a Postfix SDT using parser-stack to construct abstract syntax tree for the following context free grammar. The symbols have their usual meaning. **(10)**

$S \rightarrow id = E \mid id\ [E] = E$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow id \mid id\ [E] \mid num$

(c) Convert the following SDD into SDT. **(8)**

Productions	Semantic Rules
$S \rightarrow while\ (B)\ S_1$	*begin = newlabel()* *B. true = newlabel()* *B. false = S. next* S_1*. next = begin* *S. code = label (begin)* *//B. code // label (B.true)* *// S_1. code* *//gen('goto' begin)*

(d) Why in L-attributed grammar an inherited attribute associated with a symbol of a production body cannot be defined using any synthesized associated with the head of the production? Explain with example. **(7)**

6. (a) Draw spaghetti stack interpretation of symbol table for the code snippet. Mention line number in each entry of the spaghetti stack. **(10)**

```
1    int a,b;
2    int fun 1(int x, int y){
3          int a,x;
4          {
5                   int b,y;
6          }
7          {
8                   int a, x, y;
9          }
10   {
11   int fun 2 (int c, int d){
12         int a, b, c;
13   }
```

(b) Write an attribute grammar that recognizes strings consisting of a, b, c and produces the number of substrings that correspond to the regular expression ab + c in the input string. For example, if the input string is 'abbccabcb', then the output will be 2. You can use any arithmetic, bitwise or logical operators in the semantic rules. Show the annotated parse tree for the input string 'abbccabcb'. **(20)**

$S \rightarrow Sa \mid Sb \mid Sc \mid a \mid b \mid c$

Contd P/5

CSE 309

Contd... Q. No. 6

(c) Briefly explain the difference between static and dynamic scoping. **(5)**

7. (a) For the following grammar how many conflicts will be reported by yacc/bison? Specify the type of each conflict along with reason of its occurrence. **(8)**

$a \rightarrow b \mid c \mid \text{PLUS} \mid \text{MINUS}$

$b \rightarrow b \text{ MINUS } b \mid \text{PLUS}$

$c \rightarrow c \text{ PLUS } c \mid \text{MINUS}$

(b) Why left recursive productions are preferred to right recursive productions for shift reduce parsing? Explain with examples. **(7)**

(c) Determine whether the input strings '001001; and '001011' are derived from the following grammar by showing each step of shift/reduce parsing. You have to show the action, contents of the stack and portion of currently considered input at each step. **(8+7=15)**

$S \rightarrow CC, \quad C \rightarrow \mathbf{0}\, C \mid \mathbf{1}$

(d) Eliminate left recursion from the following SDT. **(5)**

$$A \rightarrow A_1 Y \{A.a = g(A_1.a, Y.y)\}$$
$$A \rightarrow X \{A.a = f(X.x)\}$$

8. (a) Write down the semantic rule for translating for loop statement into three address code for the following production. You have three attributes *true, false* and *code* associated with the grammar symbol *B* and two attributes *code* and *next* associated with the grammar symbol *S*. The symbols have their usual meaning. **(10)**

S → repeat *S* until *B*

(b) Construct three address code for the expression $a = -(b+c)+-d$ according to the following SDD. Also show them in triples data structure. **(10)**

PRODUCTION	SEMANTIC RULES
$S \rightarrow \text{id} = E$	*S.code = E. code* || *gen(top.get(id.lexeme) '=' E.addr)*
$E \rightarrow E_1 + E_2$	*E.addr = new Temp ()* *E.code = E_1. code* || *E_2. code* || *gen(E.addr '=' E_1.addr '+' E_2 .addr)*
$\mid - E_1$	*E.addr = new Temp ()* *E.code = E_1. code* || *gen(E.addr '=' 'minus' E_1.addr)*
$\mid (E_1)$	*E.addr = E_1.addr* *E.code = E_1. code*
| id	*E.addr = top.get(id.lexeme)* *E.code = ''*

Contd P/6

CSE 309

Contd... Q. No. 8

(c) Write a YACC program which takes a binary number as input and prints the 2's complement of the number. For example, if 1010 is given as input (terminated by new line), then the YACC program will output 0110. You do not need to write the Lex program and should use appropriate token name in your YACC program. Mention the data type associated with semantic values of grammar symbols. You only have to write the middle portion of the YACC program. **(10)**

(d) Construct the DAG for the expression $x - y + (x - y) * (x + y)$. **(5)**

L-3/T-1/CSE Date : 30/01/2016

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2014-2015

Sub : **CSE 309** (Compiler)

Full Marks : 210 Time : 3 Hours

The figures in the margin indicate full marks.

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

1. (a) Explain clearly, with necessary justifications, the lexical errors (in any) that will occur for each of the following C/C++ statements. You need to consider both the cases where the lexical analyzer uses and does not use panic mode recovery strategy. Then provide the list of tokens that may be generated in both the cases. You may use your own token names. **(20)**

   (i) `int float double int float float;`

   (ii) `;) (){}))))cse dhaka buet`

   (iii) `int 12345cse, dhaka;`

   (b) Comment with reasoning on whether the following grammars are LL(1) or not. **(15)**

   (i) $S \rightarrow ABc, A \rightarrow a \mid \in, B \rightarrow b \mid \in$

   (ii) $S \rightarrow ACB, A \rightarrow a \mid B \mid \in, B \rightarrow b \mid \in, C \rightarrow c \mid \in$

   (iii) $S \rightarrow X, X \rightarrow aXBA \mid bAXB \mid c, A \rightarrow a, B \rightarrow a \mid \in$

   (iv) $A \rightarrow C \mid \in, C \rightarrow CE, C \rightarrow \in, E \rightarrow \in$

   (v) $S \rightarrow S(S) \mid \in$

2. (a) Explain the scenario in lexical analysis where we need to look ahead at least one additional character. **(7)**

   (b) Describe how the use of sentinels at the end of buffers in buffer pair scheme in lexical analysis speeds up the input process. **(9)**

   (c) Which characteristic(s) of a programming language is (are) affected by the length of buffer in buffer pair scheme in lexical analysis? How? **(7)**

   (d) Why is analysis portion of a compiler normally separated into lexical analysis and syntax analysis phases? What is wrong with using the parser only to accomplish all the tasks currently performed by the lexical analyzer and syntax analyzer? **(12)**

3. (a) Write down, with necessary explanations, in the context of Lex lexical analyzer generator, the type of lexemes that will be matched by the following regular expressions. **(15)**

   (i) {cse}{token}

   (ii) [csetoken]

   (iii) csetoken

   (iv) (csetoken)+

   (v) csetoken+

   (b) Eliminate left-recursion from the following grammar: **(20)**

   $S \rightarrow Sc \mid Ta \mid b$

   $T \rightarrow Tc \mid Sd \mid d$

   $U \rightarrow Ua \mid Td \mid Sb \mid b$

Contd P/2

CSE 309

4. (a) Explain in detail the following statements. **(5+10)**

(i) "Java language processors combine compilation and interpretation".

(ii) "The front end of a compiler is concerned about the source language, while the back end of the compiler is concerned about the target machine. The intermediate representation, between the two, is concerned neither about the source language, nor about the target machine".

(b) Answer the following questions for predictive parsing. **(8+3+4+5)**

(i) During construction of a parsing table M, for each production rule $A \rightarrow \alpha$ of the grammar, if ϵ is in FIRST(α), we add $A \rightarrow \alpha$ to $M[A, b]$, for each terminal b in FOLLOW(A). Why? Explain clearly in light of the parsing mechanism.

(ii) What does multiple entries in a parsing table mean? Is the corresponding grammar LL(1)?

(iii) During a parsing, if w is the input that has been matched so far, and S is the start symbol for the corresponding grammar, with, $S \overset{*}{\Rightarrow} w\alpha\beta\gamma$, what will be the stack contents? Also, comment on the nature of symbols in w and stack contents regarding being terminals/non-terminals.

(iv) During parsing, if the input pointer is pointing to the symbol a, and top stack symbol in X, with the entry $M[X, a]$ being $X \rightarrow UVW$, explain how stack contents will be altered. Why?

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE**.

5. (a) What is context-sensitive analysis? What is the purpose of context-sensitive analysis? Explain S-attributed and L-attributed definitions with examples. **(10)**

(b) Consider the following syntax directed translation rules to generate three-address code: **(10)**

Productions	*Semantic rules*
$S \rightarrow$ **id** := E	S.code := E.code || *gen*(**id**.place ':=' E.place); S.begin := S.after := nil
$S \rightarrow$ **while** E **do** S_1	S.begin := *newlabel*() S.after := *newlabel*() S.code := *gen*(S.begin ':') || E.code || *gen*('if' E.place '=' '0' 'goto' S.after) || S_1.code || *gen*('goto' S.begin) || *gen*(S.after ':')
$E \rightarrow E_1 + E_2$	E.place := *newtemp*(); E.code := E_1.code || E_2.code || *gen*(E.place ':=' E_1.place '+' E_2.place)
$E \rightarrow E_1 * E_2$	E.place := *newtemp*(); E.code := E_1.code || E_2.code || *gen*(E.place ':=' E_1.place '*' E_2.place)
$E \rightarrow$ **id**	E.place := **id**.name E.code := ''
$E \rightarrow$ **num**	E.place := *newtemp*(); E.code := *gen*(E.place ':=' **num**.value)

Contd..... P/3

CSE 309

Contd ... Q. No. 5(b)

Assume, the function '*newtemp()*' generates the temporary variables like t_1, t_2, etc and the function 'newlabel()' generates new label consistently. Generate three address code according to the above semantic rules for the following strings:

i := i + j*k
while i do
i := 2 * n + k

(c) Consider the following syntax-directed definition: **(15)**

$F \rightarrow 0.B$	$F.val = B.val$
$B \rightarrow 0B_1$	$B.val = B_1.val/2$
$B \rightarrow 1B_1$	$B.val = B_1.val/2 + 1/2$
$B \rightarrow 0$	$B.val = 0$
$B \rightarrow 1$	$B.val = 1/2$

The numeric value of a binary fraction $0.b_1b_2 \ldots b_n$ is calculated as $\sum_{i=1}^{n} b_i 2^{-i}$. Each non-terminal has a synthesized attribute 'val' that is used to store its value.

(i) Show the annotated parse tree for the sentences:

0.101011

(ii) Show the dependency graph of the annotated parse tree generated in question 5(c)(i).

(iii) Find a topological order of the dependency graph generated in question 5(c)(ii).

6. (a) (i) Given the environment ρ is a set of <*name*, *type*> pairs, as follows: **(10)**

ρ = {<**x**, *integer*>, <**y**,*integer*>, <**z**,*char*>, <1,*integer*>, <2,*integer*>}

Prove that **x := x + y + 2** is typed correctly using Post system.

(ii) Provide type checking post-system expressions for the production '$S \rightarrow$ 'S_1 ; S_2' using proper notation. Also provide semantic rule for the type checking.

(b) What is a semantics-preserving transformation of compiler optimization? Explain each of the following transformation with example: **(10)**

(i) Common-sub-expression elimination,

(ii) Strength reduction

(iii) Backward copy propagation

(iv) Constant folding

(c) Given the following expression: **(15)**

((a – b) – ((a – b) * (a + b))) + ((a – b) * (a + b))

(i) Construct AST

(ii) Construct the DAG

(iii) Construct three address code for AST

(iv) Construct three address code for DAG

Assume the following grammar for this question.

$E \rightarrow E + T \mid E - T$

$T \rightarrow T * F$

$F \rightarrow (E) \mid$ **id**

Contd P/4

CSE 309

7. (a) What is type inference? Find out the type inference rule for the following "append" function written in ML functional language: **(10)**

fun *append*(*x*, *y*) = **if** *null*(*x*) **then** *y* **else** *cons*(*hd*(*x*), *append*(*tl*(*x*), *y*))

(b) What are name equivalence and structural equivalence? Explain with examples. **(10)**

(c) What is backpatching in translating short-circuit code? How backpatching is used to translate the short-circuit code? Explain with translation grammar along with translation schemes. **(15)**

8. (a) What is peephole optimization? Explain two of such optimization techniques with examples. **(10)**

(b) What is global register allocation in code generation? Give an overview of the algorithm for global register allocation with examples. **(10)**

(c) What are register descriptor, address descriptor, and spilling? Assume the system has only two registers, provide the status of register descriptor, address descriptor, and the variable spilled for the following code: **(15)**

Statements	Code Generated	Register Descriptor	Address Descriptor	Spilled Variable
t := a - b	**MOV a, R_0** **SUB b, R_0**			
u := a - c	**MOV a, R_1** **SUB c, R_1**			
v := t + u	**ADD R_1, R_0**			
d := v + u	**ADD R_1, R_0** **MOV R_0, d**			

L-3/T-1/CSE **Date : 30/07/2015**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2013-2014

Sub : **CSE 309** (Compiler)

Full Marks : 210 Time : 3 Hours

The figures in the margin indicate full marks.

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

1. (a) Give an example of **S-attributed definition**. Give another example of **L-attributed definition**. Using these two examples illustrate how every S-attributed definition is also an L-attributed definition, however, every L-attributed definition is not an S-attributed definition. **(12)**

(b) For the following SDD, give annotated parse tree to evaluate the expression, 5*4*3. **(12)**

	PRODUCTION	SEMANTIC RULES
1)	$T \rightarrow F\,T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *\,F\,T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$

(c) In the context of semantic analysis, what is **scope**? Draw the spaghetti stack interpretation of symbol tables to disambiguate scopes explicitly for the following code snippet. **(3+8=11)**

```
public class Base {
     public int a = 0;
}

public class Derived extends Base {
     public int b = 10;
}

public class Test extends Derived {
     public int a = 20;
     public void show ( ) {
          int a = 30;
          System.out.println(a);
          System.out.println(this.a);
          System.out.println(super.a);
     }
}
```

Contd P/2

CSE 309

2. (a) Translate the expression, **d = (a + b) + -c;** into three-address code using the following SDD. Show the corresponding annotated parse tree with annotations for attributes, **addr** and **code**. Then illustrate the quadruples and triples representation of the generated three-address code. **(10+10=20)**

PRODUCTION	SEMANTIC RULES
$S \rightarrow$ id $= E$;	$S.code = E.code \|\|$ $gen(top.get(\text{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr =$ **new** $Temp()$ $E.code = E_1.code \|\| E_2.code \|\|$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid -E_1$	$E.addr =$ **new** $Temp()$ $E.code = E_1.code \|\|$ $gen(E.addr\ '='\ '\text{minus}'\ E_1.addr)$
$\mid (E_1)$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid$ id	$E.addr = top.get(\text{id}\ lexeme)$ $E.code = ''$

(b) What is the main problem of using the triples representation of three-address code? How can we solve this problem? Explain with examples. **(3+2=5)**

(c) Draw the short circuit code corresponding jumping diagram and write the corresponding semantic rule for the following control-flow construct. **(10)**

S → for (S_1; B; S_2) S_3

3. (a) Write short notes with examples on – type system, string type systems, weak type systems, partial ordering of types. **(10)**

(b) The following code computes Fibonacci numbers recursively. **(8+7=15)**

```
int fibonacci (int n) {
      if (n<2) return 1;
      return fibonacci (n-1) + fibonacci (n-2);
}
```

Show the complete activation tree for it. What does the control stack look like when the third call of **fibonacci(1)** is about to return? Show only the arguments and return values in an activation record. (Assume that the initial call is fibonacci(5)).

(c) What is the major problem of using Reference Counting as a garbage collection framework? Explain with example. **(10)**

4. (a) Determine the costs of the following instruction sequence: **(10)**

LD R0, x
ADD R1, #3, #2
LD R2, *200 (R0)
ADD R0, R1, R2
ST x, R0

Contd P/3

CSE 309

Contd ... Q. No. 4

(b) For the following three-address code sequence, identify the basic blocks and construct the corresponding flow graph. **(8+7=15)**

```
1) c = 0
2) i = 0
3) j = 0
4) t1 = 8 * i
5) t2 = t1 + j
6) t3 = 8 * t2
7) mat[t3] = c
8) c = c + 1
9) j = j + 1
10) if j < 8 goto (4)
11) i = i + 1
12) if i < 8 goto (3)
```

(c) In the context of machine independent semantics preserving code optimization, write short notes with example on – copy propagation, dead code elimination, and induction variable elimination. **(2+3+5=10)**

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE**.

5. (a) Write a regular expression for each of the following sets of binary strings. **(6)**

   (i) Binary strings with no consecutive 0s or 1s

   (ii) Binary string interpreted as decimal number is divisible by 3

   (b) Write a syntactically correct C program that can not be complied successfully by a C++ compiler. **(6)**

   (c) Briefly explain how the Java language processor works. What is a just-in-time compiler? **(6)**

   (d) Briefly describe the roles of following components in the context of a compiler. **(9)**

   (i) Preprocessor

   (ii) Linker

   (iii) Loader

   (e) Discuss about different types of errors in the context of a compiler. Suggest some recovery mechanisms for lexical errors. **(8)**

6 (a) Explain the two-buffer scheme of input buffering corresponding to lexical analysis. **(6)**

(b) Write the algorithm corresponding to non-recursive predictive parsing using stack. **(7)**

(c) Describe the roles of the analysis part of a compiler. **(6)**

Contd P/4

CSE 309

Contd ... Q. No. 6

(d) Eliminate left recursion from the following grammars: **(12)**

(i) $A \rightarrow Aap \mid Bbq \mid c$

$B \rightarrow Bxp \mid Ayq \mid z$

(ii) $T \rightarrow PQ$

$P \rightarrow RQ \mid q$

$R \rightarrow Tp$

$Q \rightarrow q$

(c) Write a short note on three-address instructions. **(4)**

7. (a) Explain the following terms. **(6)**

(i) LL(k) grammar

(ii) Recursive-descent parser

(iii) Lexeme

(b) The following grammar encounters both left factoring and left recursion problem. **(12)**

$F \rightarrow FBa \mid cDS \mid c$

Using this grammar show that, left factoring after removing left recursion yields the same grammar as removing left recursion after left factoring.

(c) Describe the roles of the lexical analyzer **(6)**

(d) Describe how you can construct a translator using Yacc. **(5)**

(e) What advantages are there to a language processing system in which the compiler produces assembly language rather than machine language? Draw a block diagram showing the code optimization phases of a compiler. **(6)**

8 (a) Briefly discuss the two rules that Lex uses to decide on the proper lexeme to select, when several prefixes of the input match one or more patterns. **(5)**

(b) Report five semantic errors that the semantic analyzer is expected to recognize. **(5)**

(c) Consider the following grammar **(20)**

$S \rightarrow aAa \mid BAa \mid \varepsilon$

$A \rightarrow cA \mid bA \mid \varepsilon$

$B \rightarrow b$

Construct the predictive parsing table corresponding to this grammar. Then, using this table show the sequence of moves made by the parser on input **"bcba"**.

(d) Explain how you can design a symbol table for a case insensitive programming language, where abc, ABC, and aBc refer to the same identifier. **(5)**

L-3/T-1/CSE **Date : 24/05/2014**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2012-2013

Sub : **CSE 309** (Compiler)

Full Marks : 210 Time : 3 Hours

The figures in the margin indicate full marks.

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **NINE** questions in this section. Answer any **SEVEN**.

1. (a) With necessary examples (and figures if necessary), justify the statement, "A compiler involves only compilation, a hybrid compiler involves both compilation and interpretation, an interpreter involves only interpretation but no compilation". **(10)**

   (b) Explain why "relocatable machine code" is called "relocatable"? **(5)**

2. Enumerate the compiler construction tool kits with necessary description and applicable examples. **(15)**

3. (a) Why is the analysis portion of a compiler normally separated into lexical analysis and syntax analysis phases? What is wrong with using only the parser to accomplish both the lexical analysis and syntax analysis? **(10)**

   (b) "The token name influences parsing decisions, while the attribute value influences translation of tokens after the parse" – explain. **(5)**

4. We want to develop the transition diagram for certain operators in a programming language. The operators are, = =, ! =, =, >, <, >=, <=. Show using an appropriate transition diagram how these operators can be tokenized in lexical analysis. Also suggest necessary attributes for tokens wherever needed. **(15)**

5. Explain clearly the difference among the following regular expressions in the context of Lex lexical analyzer generator. **(15)**

   (i) {cse}{buet}

   (ii) [csebuet]

   (iii) {csebuet}

   (iv) csebuet

   (v) (csebuet) +

   (vi) csebuet +

Contd P/2

CSE 309

6. Eliminate left recursion from each of the following grammars.

(a) $S \rightarrow S + S \mid SS \mid T$ *(7)*

$T \rightarrow TT \mid T/T \mid I$

$I \rightarrow I0 \mid I1 \mid 0 \mid 1$

(b) $S \rightarrow Sa \mid Tb \mid c$ *(8)*

$T \rightarrow Tx \mid Sy \mid z$

7. Left-factor this grammar, **(15)**

$S \rightarrow T; S \mid \epsilon$

$T \rightarrow U * T \mid U$

$U \rightarrow x \mid y \mid [S]$

Hence find FIRST and FOLLOW sets for each non-terminal in the left-factored grammar.

8. Justify whether the following grammar is *LL*(1) or not. **(15)**

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

If it is not an *LL*(1) grammar, carry out necessary processing to make it *LL*(1). Hence, construct a predictive parsing table from this grammar. Then show how the strings *ab*, *ba* and *abb* are parsed by this grammar.

9. Present the SDD which uses inherited attributes to multiply integers. Use this SDD to construct an annotated parse tree to carry out the multiplication $3 \times 4 \times 5$. **(15)**

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE**.

10. (a) What is type inference? Determine the type inference rule for the following function written in ML language: **(10)**

fun *append*(*x*, *y*) = **if** *null*(*x*) **then** *y*

else *cons*(*hd*(*x*), *append*(*tl*(*x*), *y*))

where *null*(*x*) is a function that returns **true** if the list *x* is empty and **false** otherwise, and *cons*(*src*, *dest*) is a function that concatenates "*src*" with "*dest*".

(b) Consider the following grammar for generating binary fractions. **(10)**

$F \rightarrow 0.B$

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

Contd P/3

CSE 309

Contd ... Q. No. 10(b)

Fill in the missing semantic actions below to calculate the decimal value of an input string. Recall that the numeric value of a binary fraction $0.b_1b_2 \ldots\ldots b_n$ is calculated as each non-terminal has a synthesized attribute *val* that is used to store its value. The final value should be returned in *F.val*.

$F \rightarrow 0.B$ {F.val = B.val}

$B_0 \rightarrow 0B_1$ { }

$B_0 \rightarrow 1B_1$ { }

$B \rightarrow 0$ { }

$B \rightarrow 1$ { }

(c) Draw symbol table structure for the following program that takes care of scope: **(10)**

```
struct S
{ int a;
  int b;
} s;
void swap(int& a, int&b)
{ int t;
  t = a;
  a = b;
  b = t;
}
void foo( )
{ ...
  swap(s.a, s.b);
  ...
}
```

(d) Consider the following C type declaration of a binary tree node: **(5)**

```
struct IntBinTree{
        int val;
        struct IntBinTree *left;
        struct IntBinTree *right;
} tree;
```

Draw the cyclic type graph for tree (as would be constructed by a compiler).

11. (a) What is a type system? What is a strongly typed language? **(5)**

(b) Provide syntax directed definition and determine post system proof for correct typing of the following statements: **(15)**

Contd P/4

CSE 309

Contd ... Q. No. 11(b)

(a) $S \rightarrow$ **id** := E

(b) $S \rightarrow S_1$; S_2

(c) $S \rightarrow$ **while** E **do** S_1

(c) What is the syntax directed definition for the following boolean expressions **(15)**

(a) $B \rightarrow B_1 \parallel B_2$?

(b) $B \rightarrow$ **true**

(c) $B \rightarrow E_1$ **rel** E_2

where **rel** indicates a relational operator.

12. (a) Describe static and dynamic scoping and how they differ. Give an example of a program that has different output assuming static versus dynamic scoping. **(10)**

(b) Consider the following program: **(20)**

```
program P()
    var p : integer;
    procedure Q(k : integer)
        begin
            R(k, p)
        end
    procedure R(i : integer, j : integer);
        var n : integer
        procedure T(i : integer)
            begin
                ... (* body of T *)
            end;
        procedure S()
            var m : integer
            begin
                T(n)
            end;
        begin
            S()
        end;
    begin
        Q(p)
    end
```

(i) Program P calls Q, Q in turn calls R, R in turn calls S, and S in turn calls T. Draw the resulting stack layout with activation records. Show the arguments and local variables in each record and draw the access links.

Contd P/5

CSE 309

Contd ... Q. No. 12(b)

(ii) Which variables are visible (in scope) in the body of T and how many access links must be traversed to reach the nonlocal data?

Var	visible (Y/N)	# links
i		
j		
k		
m		
n		
p		

(c) What are the three proposed evaluation methods for semantic rules to decorate parse trees? Explain. **(5)**

13. (a) Partition the following fragment of three-address codes into basic blocks and construct the control flow graph. **(10)**

```
        if n>0 goto L1
        goto L3
L1:     i := 0
        f := 1
L2:     i := i + 1
        f := f * i
        if i<n goto L2
        goto L4
L3:     f := 0
L4:     halt
```

(b) What are peephole optimizations? Name and explain three peephole optimizations with examples. **(10)**

(c) What do you know about *register descriptor* and *address descriptor*? Generate code of the statements of the following table and fill up the value of the descriptions. **(10)**

Statements	*Code generated*	*Register Descriptor*	*Address Descriptor*
t := a - b **u := a - c** **v := t + u** **d := v + u**			

(d) Give three examples of tasks that are typically performed in a type checker. What types of errors are recognized in a type checker? **(5)**

S. Rahman 29.09.13

L-3/T-1/CSE Date : 29/09/2013

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2011-2012

Sub : **CSE 309** (Compiler)

Full Marks : 210 Time : 3 Hours

The figures in the margin indicate full marks.

USE SEPARATE SCRIPTS FOR EACH SECTION

SECTION – A

There are **FOUR** questions in this section. Answer any **THREE**.

1. (a) Write regular definitions for the following languages: **(15)**

(i) Comments, consisting of a string surrounded by /* and */, without an intervening */, unless it is inside double-quotes (").

(ii) All strings of lowercase letters that contain the five vowels in order.

(iii) Define a regular expression for the unsigned number (integer or floating point) of the C language. Few examples are .02, 12, 12.3, 12., 32.23E+2, 23.5E-3.

(b) What is a symbols table of a compiler? In which phase or phases is the symbol table used? Explain the symbol table management of a compiler. **(15)**

(c) Distinguish between a compiler and an interpreter. Which one of these (compiler and interpreter) is suitable for performance? Explain. **(5)**

2. (a) Explain the differences between top-down parsing and bottom-up parsing. Which one is used in commercial compiler and why? **(5)**

(b) What are the goals of an error-handling strategy in syntax analysis phase? What are the error-handling strategies in this phase? Explain the Panic-mode recovery with an example. **(15)**

(c) What is a recursive-descent parsing? When does the recursive-descent parsing not work? Explain. **(10)**

(d) Explain with an example the reason why the bottom-up parsing does not require a left-factored grammar. **(5)**

3. (a) What is predictive parsing? Find the parsing table for the following grammar in the context of predictive parsing: **(20)**

$E \rightarrow T + E \mid T$

$T \rightarrow \text{int} \mid \text{int} * T \mid (E)$

(b) Consider the context-free grammar $S \rightarrow (L) \mid a$ and $L \rightarrow L, S \mid S$ with string ((a, a), a, (a)). Answer the following: **(15)**

(i) Give a leftmost derivation for the string.

(ii) Give a rightmost derivation for the string.

(iii) Is the grammar ambiguous or unambiguous? Justify your answer.

(iv) Describe the language generated by this grammar.

Contd P/2

CSE 309

4. (a) Given the following grammar: **(15)**

E -> E + E | E – E | E * E | E / E | **int**

Show two different left-most derivations of the string **int** + **int** * **int** / **int.** Also show the corresponding parse trees. What does this tell you?

(b) How would you change the following grammar: **(10)**

E -> E + E | E – E | E * E | E / E | **int**

so that all of the binary operations are left associative and the precedence of the operators?

(c) What is a shift-reduce parsing? Explain with an example. **(10)**

SECTION – B

There are **FOUR** questions in this section. Answer any **THREE.**

Assume suitable values for any missing data.

5. (a) Consider the following context free grammar (CFG) for arithmetic expressions involving addition (+) operations

$E \rightarrow TE'$

$E' \rightarrow + TE' \mid \in$

$T \rightarrow$ **digit**

(i) Extend the above CFG so that it can handle expressions having subtraction (–) and multiplication (*) operations. **(5)**

(ii) Convert the extended CFG of 5(a).(i) into a Syntax Directed Definition (SDD) to evaluate the expressions. **(7)**

(iii) Depict an **annotated** parse tree for expression 3*5-4 based on the SDD of 5(a)(ii). **(8)**

(iv) Convert the SDD of 5(a)(ii) into a corresponding Syntax Directed Translation (SDT). **(5)**

(Assume that all the operations (+, –, *) follow their usual precedence and associative rules.)

(b) Convert the following statements into equivalent three address code sequences **(2×5)**

(i) do i = i + 1; while (b[i] < v)

(ii) y = p (a, b, c, d, e) + b[i];

(Assume that, b is an array of elements with 16 units of space required for each element and p is a function taking 5 parameters).

6. (a) Define short-circuit code with an appropriate example. **(7)**

(b) Consider the following CFG for control flow statements.

$P \rightarrow S$

$S \rightarrow$ **id** = **num**;

$S \rightarrow$ **if** (B) S1 **else** S2

$B \rightarrow B \mid\mid B \mid B \&\& B$

$B \rightarrow E$ **relop** E

$E \rightarrow$ **id** | **num**

Contd P/3

CSE 309

Contd ... Q. No. 6(b)

(i) Convert the above CFG into an SDD so that it can generate equivalent three address codes for the control flow statements. **(12)**

(ii) For the statement **if (x < 100 || x > 200 && x !=y) x = 0; else x = 1**; construct an annotated parse tree based on the SDD of 6(b)(i) by showing values for all necessary attributes. **(16)**

7. (a) For a language with nested procedure declarations, define nesting depth. How can it be used to manipulate access links while procedure calls are implemented for that language? **(4+7)**

(b) Depict the division of tasks between the caller and callee in respect of function calling. **(10)**

(c) What is the cost and effect of the following machine instructions **(9)**

(i) LD R2, 100(R1)

(ii) ST * 100(R1), R2

(iii) ADD R1, R1, #100

(d) Briefly describe the issues that should be considered while designing a code generator. **(5)**

8. (a) With suitable examples, explain the following terms relating to code optimization based on basic blocks/flow-graphs **(3+4+5+6)**

(i) leader statements

(ii) global common-subexpression elimination

(iii) copy propagation and dead-code elimination

(iv) induction variable elimination

(b) Consider the following grammar for declaration statement. **(9)**

$D \rightarrow T$ **id**;

$T \rightarrow$ **B** C

$B \rightarrow$ **int** | **float**

$C \rightarrow$ **[num]** C | ϵ

Design an SDT for the above grammar that will compute the type and width of a declared identifier and insert them in a symbol table. Assume, the width of an integer is 4 and that of a float is 8.

(c) Explain that any SDD having a production $A \rightarrow B\ C$ with semantic rules A.s = B.b and B.i = f(C.c, A.s) can be neither S-attributed nor L-attributed SDD. Here, s is a synthesized attribute and i is an inherited attribute while b and c can be of any type. **(8)**

L-3/T-1/CSE **Date : 08/03/2012**

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

L-3/T-1 B. Sc. Engineering Examinations 2010-2011

Sub : **CSE 309** (Compilers)

Full Marks : 210 Time : 3 Hours

USE SEPARATE SCRIPTS FOR EACH SECTION

The figures in the margin indicate full marks.

SECTION – A

There are **NINE** questions in this Section. Answer any **SEVEN**.

1. (a) Define and explain the terms compiler, assembler and linker. For C/C++ programs, elaborate the relationship of these with (i) header files, (ii) libraries. **(15)**

2. Two main types of errors that can occur in the lexical analyzer are (a) no regular expression matches the current input; and (b) a start comment character inside a comment. Explain these errors with necessary examples. How do you want to deal with these errors in a lexical analyzer? **(15)**

3. Explain clearly the difference among the following regular expressions in the context of Lex lexical analyzer generator. **(15)**

   (i) {mytoken}

   (ii) [mytoken]

   (iii) mytoken

   (iv) (mytoken) +

   (v) mytoken+

4. a*b printf("1");

   (a|b)*b printf("2");

   c* printf("3");

   We have the above snippet, with patterns and their associated actions, from a Lex code. Show the output, with detailed explanations, that is produced when this scanner is run over the following strings: **(15)**

   (a) aaabccabbb

   (b) cbbbbabc

   (c) cbabc

Contd P/2

CSE 309

5. (a) Left-factor this grammar, **(15)**

$S \rightarrow T; S \mid \in$

$T \rightarrow UR$

$R \rightarrow *T \mid \in$

$U \rightarrow x \mid y \mid [S]$

Hence find FIRST and FOLLOW sets for each non-terminal in the grammar obtained.

6. (a) For the left-factored grammar in Question 5, construct an LL parsing table. Now show in detail what will happen if the string [x;y]*[; is parsed. **(15)**

7. Eliminate left recursion from the grammar, **(15)**

$A \rightarrow B \mid a \mid CBD$

$B \rightarrow C \mid b$

$C \rightarrow A \mid c$

$D \rightarrow d$

If you encounter a production like $X \rightarrow X$ at some point of elimination, you can drop it since it does not affect the language.

8. Identify whether each of the following grammar is LL(1) or not. State the reason(s) in each case. **(15)**

(a) $A \rightarrow C \mid \in, C \rightarrow C\ E, C \rightarrow \in, E \rightarrow \in$

(b) $S \rightarrow A \mid B, A \rightarrow aaA \mid aa, B \rightarrow aaB \mid a$

(c) $S \rightarrow AaAb \mid BbBa, A \rightarrow \in, B \rightarrow \in$

(d) $A \rightarrow abc \mid B, B \rightarrow acb$

(e) $A \rightarrow bcd \mid B, B \rightarrow \in, C \rightarrow DAbd, B \rightarrow b, C \rightarrow c, D \rightarrow d$

9. (a) Write a Yacc plus Lex program which calculates the value of an arithmetic expression which involves single line inputs only. The arithmetic expression comprises of numbers with decimal points, addition, subtraction, multiplication and division operators. Parentheses can be used in the expression. The expressions may also contain the sin, cos and tan trigonometric functions. **(15)**

SECTION – B

There are **FOUR** questions in this Section. Answer any **THREE**.

10. (a) Differentiate with example between - **(4×6)**

(i) Syntax-directed definition and syntax-directed translation.

(ii) Static and dynamic storage allocation in run-time environment.

(iii) Syntax tree and directed acyclic graph (DAG)

(iv) S-attributed definitions and L-attributed definitions.

Contd P/3

CSE 309

Contd ... Q. No. 10

(b) We have the following three address code sequence where we have two procedures m, and p. Our target machine is byte addressable with 8 bytes to a word. Codes for the procedures start at addresses 100, and 400 respectively and stack starts at address 1000. Activation record sizes are 128, and 256 bytes respectively. Each action requires 80 bytes. Find out the target code for the given three address code sequence when stack allocation is used. Target code must contain appropriate comments. **(11)**

```
/* code for m */
action 1
call p
action 2
halt
/* code for p */
action 3
return
```

11. (a) The following grammar generates binary numbers with a decimal point.

$A \rightarrow L \,.\, L$
$\quad\quad | \; L$
$L \rightarrow L \; B$
$\quad\quad | \; B$
$B \rightarrow 0$
$\quad\quad | \; 1$

(i) Design an SDT **(Syntax-directed translation scheme)** that computes the value of binary numbers generated from the above grammar. **(12)**

(ii) The translation of string 110.101 should be the decimal number 6.625. Draw the annotated parse tree for this translation and clearly show the dependency graph. **(7+6)**

(b) Construct DAG for the following assignment statement **(10)**

b[i] = a + (a + a + (a + a + a +(a + a + a+ a)))

12. (a) For the following assignment statement write down an equivalent three address code sequence. **(10)**

y[i][j] = a[k] * b + c (3, d, 2.5)

Here, y is an array of type **array (10, array (15, float))**, a is an array of type **array (10, integer)** and c is a function of type **integer × float × float → float**. Assume that width of an integer is 4 and that of a float is 8.

Contd P/4

CSE 309

Contd ... Q. No. 12

(b) The following C code computes Fibonacci numbers recursively.

```
int f ( int n) {
    int t, s;
    if ( n < 2 ) return 1;
    s = f (n - 1) ;
    t = f (n - 2) ;
    return s + t ;
}
```

(i) Show the complete activation tree for it. **(8)**

(ii) What does the control stack look like when the fourth call of f (1) is about to return. Show only argument and return value in an activation record. (Assume that the initial call is f(6)). **(7)**

(c) Write down the names and purposes of the fields that might appear in a general activation record of a function. **(10)**

13. (a) Consider the following grammar for declaration statement.

$D \rightarrow T$ id;

$T \rightarrow B\ C$

$B \rightarrow$ int | float

$C \rightarrow P\ A$

$P \rightarrow * P \mid \in$

$A \rightarrow$ [num] $A \mid \in$

(i) Construct parse trees for the following declaration statements according to the above grammar. **(4+5+6)**

(1) int x;

(2) float * y;

(3) int * [10] z;

(ii) Design and SDD for the above grammar that computes the type and width of a declared identifier. Assume, the width of an integer is 4 and that of a float is 8. **(13)**

(b) Draw the code structure layout for the following control - flow construct. **(7)**

$S \rightarrow$ for (S1 ; B ; S2) S3
