CBCS SCHEME

USN

18CS61

Sixth Semester B.E. Degree Examination, June/July 2023
System Software and Compilers

Time: 3 hrs. Max. Marks: 100

Note: ***Answer any FIVE full questions, choosing ONE full question from each module.***

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

Module-1

1 a. With reference to SIC/XE machine architectures explain instruction formats and addressing modes, clearly indicating the settings of different flag bits. **(10 Marks)**

b. With an illustrate example, explain the need for a two pass assembler. Explain the data structures used in 2-pass assembler. Mention their functions clearly during pass 1 and pass 2. **(10 Marks)**

OR

2 a. Generate the complete object program for the following SIC/XE assembly language programs. Assume : CLEAR = B4, LDT = 74, TD = EO, JEQ = 30, TIXR = B8, JLT = 38, RSUB = 4C, LDCH = 50, WD = DC, X = 1, T = 5.

```
WRREC    START   105D
         CLEAR   X
         LDT     LENGTH
WLOOP    TD      OUTPUT
         JEQ     WLOOP
         LDCH    BUFFER, X
         WD      OUTPUT
         TIXR    T
         JLT     WLOOP
         RSUB
OUTPUT   BYTE    X'O5'
BUFFER   RESB    400
LENGTH   RESB    2
         END     WRREC
```

(10 Marks)

b. Explain the absolute loader and Bootstrap loader with algorithm/source code. **(10 Marks)**

Module-2

3 a. What is a Compiler? Explain the various phases of a compiler with a neat diagram and show the output of each phase for the expression a = b + c * 25. Assume all variable are a type float. **(10 Marks)**

b. Write a note on the commonly used compiler – construction tools. **(04 Marks)**

c. Describe Input Buffering mechanism with an algorithm for lookahead code with sentinels. **(06 Marks)**

OR

4 a. Construct the transition diagrams to recognize the tokens given below and explain the same.
i) relop ii) Identifier iii) unsigned numbers **(10 Marks)**

b. With example, define the operations on languages. **(04 Marks)**

c. Discuss the issues/errors of lexical analysis and the error recovery actions that can be performed. **(06 Marks)**

Module-3

5 a. What is recursive-decent parsing? Explain with a pseudocode. Take the grammar S → cAd, A → ab|a as an example and trace it for input string w = cad. Explain how backtracking can be used for tracing. **(10 Marks)**

b. Consider the context free grammar :

S → SS + | SS * |a and string w = aa + a*

i) Give the leftmost and rightmost derivation and parse tree for the string

ii) Is the grammar ambiguous or unambiguous? Justify your answer

iii) Eliminate left Recursion **(10 Marks)**

OR

6 a. With a neat diagram, explain the model of a table driven predictive parser. Write and explain the predictive parsing algorithm. **(10 Marks)**

b. Consider the following grammar with terminals (, [,),].

S → TS | [S] S |)S | ∈

T → (X)

X → TX | [X] X | ∈

i) Construct FIRST and FOLLOW sets

ii) Construct its LL(1) parsing table

iii) Is this grammar LL(1)? **(10 Marks)**

Module-4

7 a. Explain the meta – characters used in regular expression with examples. **(10 Marks)**

b. Write a LEX program to recognize and count the number of identifiers in a given input file. Show how the program is complied and executed. **(10 Marks)**

OR

8 a. What are the ambiguities that arise while evaluating a regular expression? Explain with example. **(10 Marks)**

b. Write a YACC program to recognize a valid arithmetic expression that uses operators +, –, * and /. **(10 Marks)**

Module-5

9 a. What is a dependency graph? Give a syntax directed definition for simple type declaration including int and float type. Construct annotated parse tree and dependency graph for the input, float a, b, c. **(10 Marks)**

b. Explain synthesized attribute, inherited attribute, S – attributed definition and L- attributed definitions with examples. **(10 Marks)**

OR

10 a. What is a three – address code? explain the different ways of representing three – address codes with examples. **(10 Marks)**

b. What is target computer model? Explain the different kinds of instructions and addressing modes available in assembly language or a target machine. **(10 Marks)**

* * * * *

18CS61-System Software & Compilers

Scheme & Solution

July 2023 Examination

1)a)	Instruction formats : Format 1, 2, 3 & 4	4M
	Addressing modes: Base-relative, PC relative, direct addressing, indexed addressing, immediate addressing, indirect addressing.	6M
1)b)	Need for 2-pass assembler : (Explain with example) pass1 – define symbols Pass 2 – assemble instructions & generate object program.	4M
	Data Structures: Location Counter, OPTAB, SYMTAB.	6M
2)a)	(program listing below)	Loc – 3M Obj code – 5M obj pgm – 2M

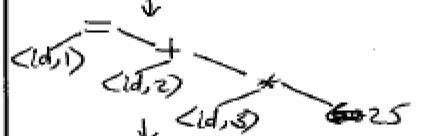
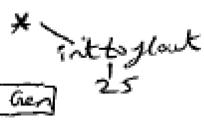
Loc				Object code
	WRREC	START	105D	
105D		Clear	X	B410
105F		LDT	Length	7721A7
1062	WLoop	TD	Output	E32012
1065		JEQ	WLoop	332FFA
1068		LDCH	Buffer, X	53901078
106C		WD	Output	DF2008
106F		TIXR	T	B850
1071		JLT	WLoop	3B2FEE
1074		RSUB		4F0000
1077	Output	BYTE	X 'O5'	05
1078	Buffer	RESB	400	
1208	Length	RESB	2	
		END	WRREC	
120A				

Object Pgm:

H^WRREC^00105D^0001AD
T^00105D^1B^B410^7721A6^E32012^332FFA^53901078^DF2008^–^–^–^05
E^00105D

2) b)	<u>Absolute Loader</u> – does not perform linking & program relocation
	<u>Algorithm</u>:
	begin
	read Header record
	verify pgm name & length
	read first Text record
	while record type ≠ 'E' do
	begin
	move obj code to specified loc in memory
	read next obj pgm record
	end
	jump to addr specified in End record
	end.
		5M

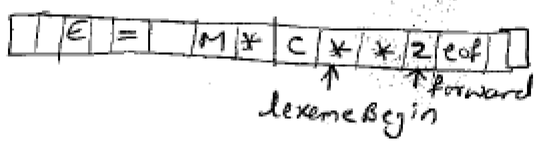
	<u>Bootstrap loader</u> – loads OS & begins at addr 0. Source code of Bootstrap loader for SIC/XE	5M

3 a)	a = b + c * 25
	↓
	[Lexical Analyzer]
	↓
	<id,1><=><id,2><+><id,3><*><25>
	↓
	[Syntax Analyzer]
	↓
	
	[Semantic Analyzer]
	↓
	(Same tree) 
	↓
	[Intermediate Code Gen]
	↓
	t1 = inttofloat
	t2 = id3 * t1
	t3 = id2 + t2
	id1 = t3
	↓
	[Code Optimizer]
	↓
	t1 = id3 * 25.0
	id1 = id2 + t1
	↓
	[Code Generator]
	↓
	LDF R2, id3
	MULF R2, R2, #25.0
	LDF R1, id2
	ADDF R1, R1, R2
	STF id1, R1
	
	<u>Compiler</u> – program that can read a pgm in one <u>lang</u> & translate into another <u>lang</u>
	Explaination of phases – 5M
	Example – 4M
		1M

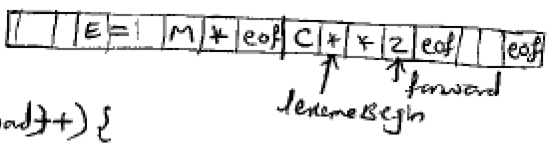
3b) Compiler Construction tools – Parser generators, Scanner generators, Syntax-directed translation engines, Code-generator generators, Data-flow analysis engines, Compiler-Construction toolkits. — **4M**

~~[illegible]~~

3c) Buffer pairs :– — **3M**



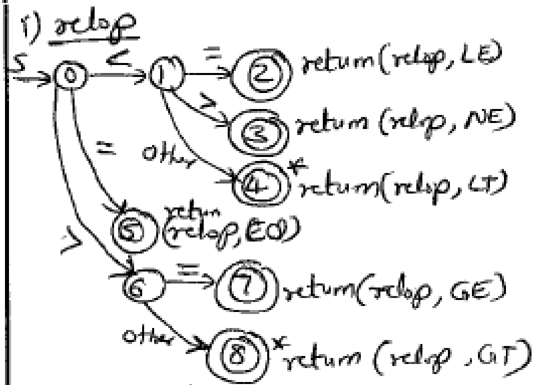
Sentinels : — **4M (2+2M)**



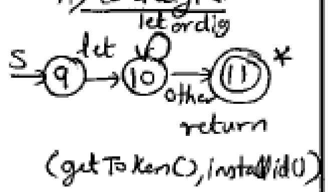

```
switch(*forward++){
  Case eof:
     if (forward is at eob 1st){
            reload 2nd buffer;
            forward = beginning of 2nd buffer;
     }
     else if(forward is at eo 2nd buffer) {
            reload 1st buffer;
            forward = beginning of 1st buffer;
     }
     else /* terminate */
     break;
} - - -
```

4a)

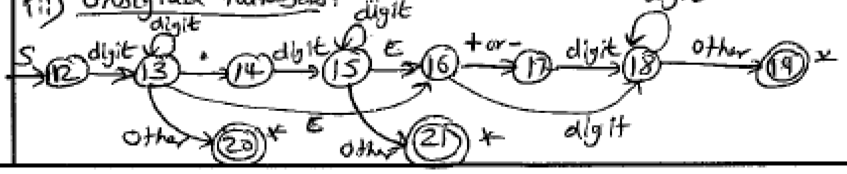
i) relop — **4M**



ii) identifier — **2M**



iii) Unsigned numbers : — **4M**



4b) Operations with examples:

i) Union of L & M (L∪M)
ii) concatenation of L & M (LM)
iii) Kleene closure of L (L^*)
iv) Positive closure of L (L^+)

— 4M

4c) Source-code error: fi(a == f(x)) ...
Panic-mode error.

— 2M

Error-recovery actions:

i) Delete one character
ii) Insert missing character
iii) Replace a character
iv) Transpose 2 adjacent characters.

— 4M

5a)

```
void A() {
  Choose an A-prod, A → X1 X2 ... Xk;
  for (i=1 to k) {
     if (Xi is nonterminal)
        call procedure Xi();
     elseif (Xi equals current I/p symbol a)
        advance I/p to next symbol;
  } else /* error */
}
```

— 5M

$S \to cAd$
$A \to ab \mid a$

— 3M+2M

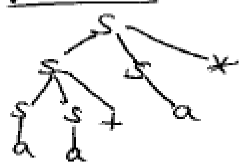
5b) i) Leftmost

$S \to SS*$
$\to SS+S*$
$\to aS+S*$
$\to aa+S*$
$\to aa+a*$

Rightmost

$S \to SS*$
$\to Sa*$
$\to SS+a*$
$\to Sa+a*$
$\to aa+a*$

Parse tree for left & right derivation



— 6M
— 1½ * 4 = 6M

ii) Grammar is unambiguous — 1M

~~iii) L = {a, aa+, aa*, aaaa*+, ... }~~ — ~~1M~~

iv) Left recursion: $S \to aS'$, $S' \to S+S' \mid S*S' \mid \epsilon$. — 3M

5 a)

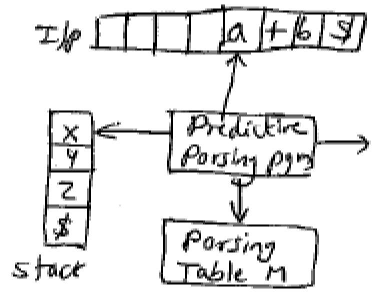


Diagram - 2M
Explanation - 3M

Algorithm - 3M
Explanation - 2M

Algorithm: let a be 1st symbol of w;
let X be top of stack;
while ($X \neq \$$){
if (X=a) pop stack & let a be next symbol of w;
else if (X is terminal) error();
else if (M[X,a] is error) error();
else if ($M[X,a] = X \rightarrow Y_1 Y_2 \cdots Y_k$) {
o/p production $X \rightarrow Y_1 Y_2 \cdots Y_k$;
pop the stack;
push $Y_k, Y_{k-1}, \ldots Y_1$ onto stack.
} let X be top stack symbol.
}

6 b)

i) First(S) = { (, [,), ε }, First(T) = { (}, First(X) = { (, [, ε } 2M
Follow(S) = { $,] }, Follow(T) = { $, [, (,),] }, Follow(X) = {),] } 2M

ii) Parsing table 5M

NT	(	)	[	]	$
S	S→TS	S→)S	S→[S]S	S→ε	S→ε
T	T→(X)				
X	X→TX	X→ε	X→[X]X	X→ε	

iii) Grammar is LL(1). 1M

7 a) Meta Characters: . * [ ] ^ $ { } 5M
\ + ? | "..." / ( )
Examples 5M

7 b)

```
%{ int idcnt=0;
%}
%%
[_a-zA-Z][_a-zA-Z0-9]*   {printf("valid");
                          idcnt++; }
```

```
[^\t\n]+        {printf("invalid");}
 \n             {;}
%%
main()
{ FILE *p;
   char fname[30];
   scanf("%s", fname);
   yyin = fopen(fname, "r");
   yylex();
   printf("%d", idcnt);
}
```

Program: 8M
Output: 2M

8 a) Expression ambiguity with example — 3M
Precedence & associativity — 1M
Specifying implicitly — 3M
Specifying explicitly — 3M

8 b)

```
%{ #include "lex.yy.c"
%}
% token ID NUM
%%
assign : ID '=' exp
       ;
 exp : exp '+' term
     | exp '-' term
     | term
     ;
term : term '*' fac
     | term '/' fac
     | fac
     ;
 fac : ID
     | NUM
     ;
%%
```

```
%{ #include "y.tab.h"
%}
%%
[a-zA-Z]+    return ID;
[0-9]+       return NUM;
\n           return 0;
.            return (yytext[0]);
%%
main()      yyerror()
{ ...       { ...
}           }
```

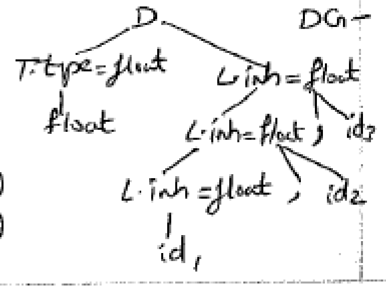
10M.

9 a) Dependency graph represents interdependencies between synthesized & inherited attributes. — 2M

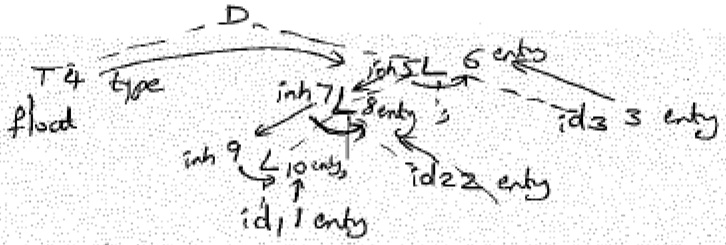
SDD – 3M
APT – 2M
DG – 3M

Production	Semantic Rules
D → TL	L·inh = T·type
T → int	T·type = int
T → float	T·type = float
L → L_1, id	L_1·inh = L·inh addtype(id·entry, L·inh)
L → id	addtype(id·entry, L·inh)

APT



DG:



9b) Synthesized attribute – defined at children & itself – 2M
Inherited attribute – defined at parent, itself & siblings – 2M
S-attribute Defn – every attr is synthesized – 3M
L-attribute Defn – attribute should be synthesized or Inherited. Edges goes from left to rt. – 3M

10a) 3-address code – linearized repr^n of DAG – 1M
$t1 = b - c$

Representations: 1) Quadruples: op, arg1, arg2, res 3M
2) Triples: op, arg1, arg2 3M
3) Indirect triples: listing of pointers to triples. 3M

10b) Target computer models a 3-address machine, with load & store operations, computation operations, jump operations & conditional jumps & n general-purpose registers. 1M
Instructions: Load oper^n, Store oper^n, unconditional jumps, computation oper^n, conditional jumps. 4M
Addressing modes: Absolute, Indexed, Integer indexed, Indirect addressing, Immediate addressing. 5M