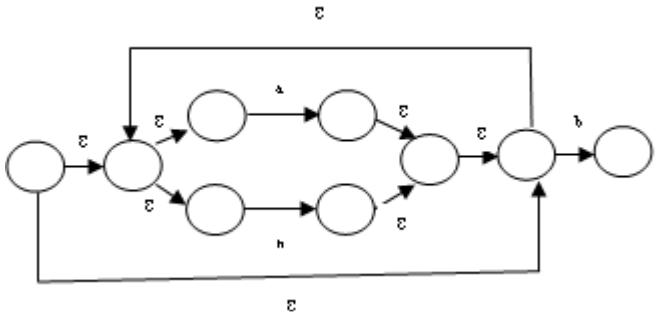
SSN COLLEGE OF ENGINEERING, KALAVAKKAM – 603 110
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
B.E. Computer Science and Engineering
CS6660 Compiler Design Unit Test 1 Answer Key

Qn. No	Part - A	Marks
1	What are the two parts of the compiler? Explain briefly. **Ans:** Analysis and Synthesis stages are two parts of the compiler. Analysis stage – Breaks the source program into pieces and creates (language independent) intermediate representation of the program Synthesis stage – Desired target program from intermediate representation	2
2	Construct NFA to recognize the language (a|b)*ab using Thompson's construction rules **Ans:** 	2
3	What are the uses of global variables yytext, yyleng and yylval? **Ans:** yytext - String containing the text of lexeme yyleng – Integer holding the length of the lexeme stored in yytext yylval – Global variable which is used to store the value of the variable	2
4	What is symbol table and how it is constructed? How it is useful for the compiler? **Ans:** Symbol table is a data structure that consists of the identifiers used in the program along with its associated information. It is constructed during the lexical analysis phase of the compiler. It is useful for detecting the multiple declaration of single variable and is used to check the type information in semantic analysis phase	2
5	When several prefixes of the input match to one or more patterns, how lex compiler resolves this issue? **Ans:** Conflict arises when several prefixes of input matches one or more patterns. This can be resolved by the following: • Always prefer a longer prefix than a shorter prefix. • If two or more patterns are matched for the longest prefix, then the first pattern listed in lex program is preferred.	2

Part – B Answer all questions (8+16+16)

6.	Write lex program to recognize the identifier, numeric constants including fraction and exponentiation representations, keywords and operators **Ans:** %{ %} id [a-zA-Z]([a-zA-Z]|[0-9])* digit [0-9]+ optf .{digit}+ floatconst {digit} {optf}	8

```
expconst {floatconst} E[+\-]?{digit}
keywd int|char|float|if|for|switch|case
arithop [+\-*/%]
relop "<"|"<="|">"|">="|"=="|"!="
logicalop "&&"|"||"|"!"
%%
{id} {printf("%s Identifier",yytext);}
{digit} {printf("%s Numeric Constant",yytext);}
{floatconst} {printf("%s Float Constant",yytext);}
{expconst} {printf("%s Exp Constant",yytext);}
{keywd} {printf("%s Keyword",yytext);}
{arithop} {printf("%s Arithmetic Operator",yytext);}
{relop} {printf("%s Relational Operator",yytext);}
{logicalop} {printf("%s Logical Operator",yytext);}
%%
int main()
{
    yylex();
}
int yywrap()
{
   return 1;
}
```

OR

7 a) Write notes on compiler construction tools. 4

Ans:

Scanner generators – produce lexical analyzer based on regular expression

Parser generators – produce syntax analyser from context free grammar as input

Syntax-directed translation engines – produce collection of routines for walking parse tree to generate intermediate code

Automatic code generators - Takes collection of rules that define the translation of each operation of the intermediate language into the machine language for the target machine.

Data-flow engines

- To perform good code optimization involves data-flow analysis

– Gathering of information about how values are transmitted from one part of a program to other part.

b) Explain various parameter passing mechanisms namely, call by value, call by reference and aliasing for swapping of two numbers. 4

Ans:

Call-by-value

```
#include
void swap(int x, int y)
{
   int t;
   t=x;
   x=y;
   y=t;
}
int main()
{
int a = 10, b=98;
printf( "Inside main - before calling swap: %d %d", a,b);
swap(a,b);
printf( "Inside main - after calling swap: %d %d", a,b);
}
```

Call-by-reference

```
#include
void swap(int *x, int *y)
```

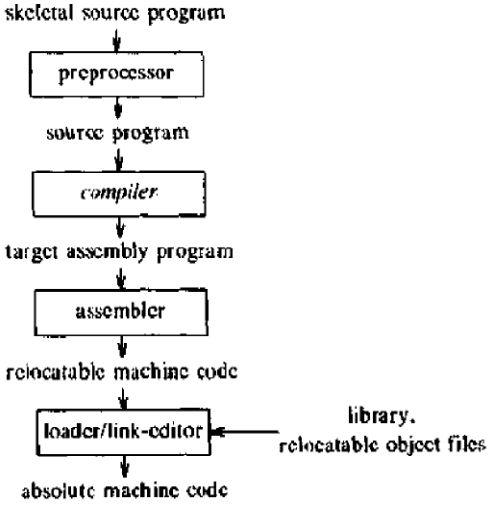
```
{
   int t;
   t=*x;
   *x=*y;
   *y=t;
}
int main()
{
int a = 10, b=98;
printf( "Inside main - before calling swap: %d %d", a,b);
swap(&a,&b);
printf( "Inside main - after calling swap: %d %d", a,b);
}
```

Call-by-aliasing

```
#include
void swap(int &x, int &y)
{
   int t;
   t=x;
   x=y;
   y=t;
}
int main()
{
int a = 10, b=98;
printf( "Inside main - before calling swap: %d %d", a,b);
swap(a,b);
printf( "Inside main - after calling swap: %d %d", a,b);
}
```

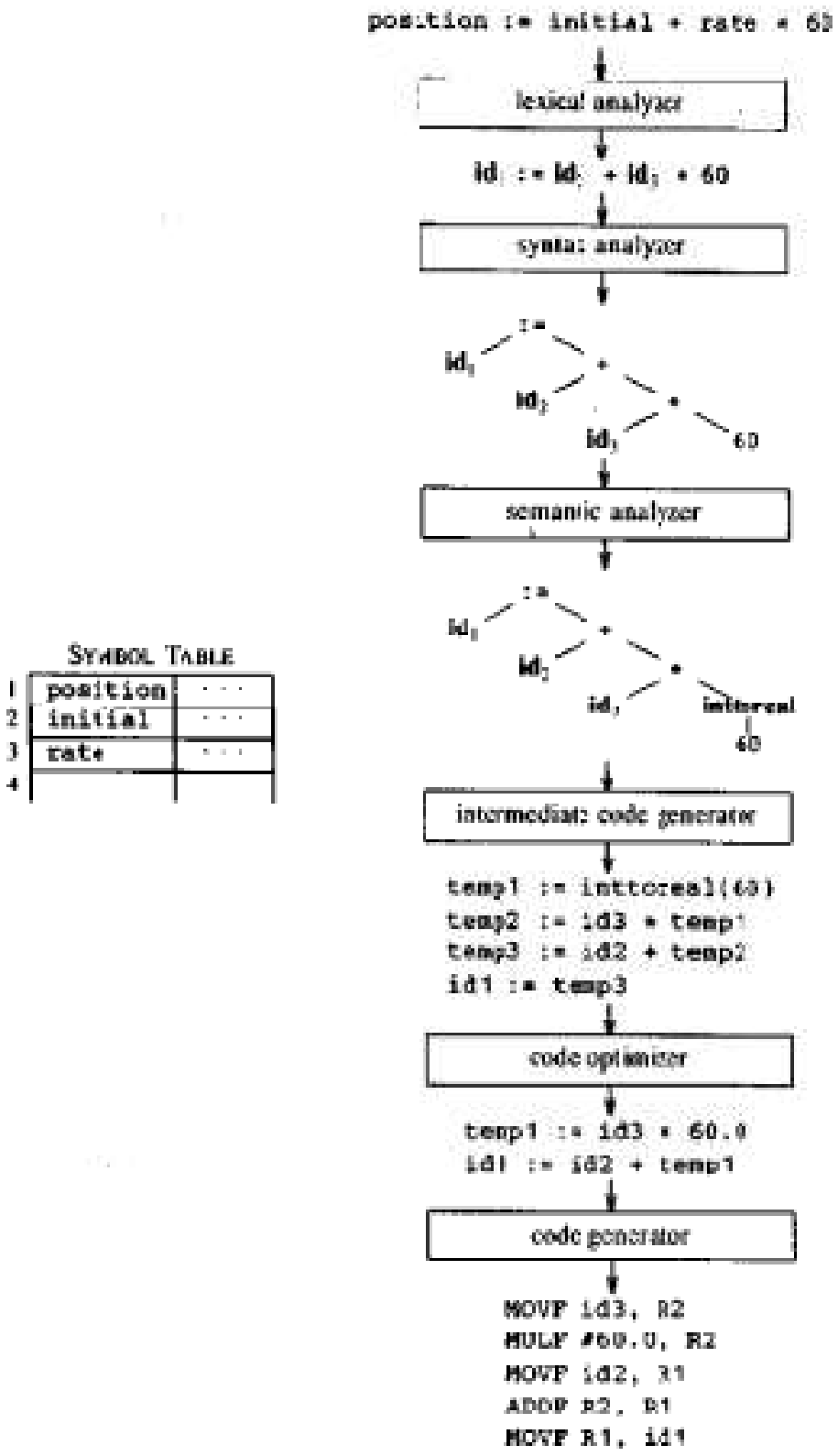
8 a) Explain language processing system with the neat diagram 6

Ans:



Pre-processor includes the header file and expand the macro in place of macro calls. The input for the compiler is the pure HLL program. The compiler converts the HLL program into a sequence of target assembly language program. This ALP is given to assembler to generate machine code. This machine code is loaded into main memory and executed.

b) Describe various phases of the compiler and trace it with the program segment 10
position:=initial+rate*60



OR

9 (a) Explain various errors encountered in different phases of the compiler 6

Ans:

<u>Errors in Lexical phase</u>

Misspelt keyword can be recognized as identifiers

fi(a==f(x))---

fi matches to the pattern of identifier

Invalid token, invalid character

If lexical analyzer is unable to proceed because none of the patterns for the tokens match with the input lexeme, it tries to panic mode recovery

Possible recovery options would be

- Delete one character from the remaining input
- Insert a misspelt character in the remaining input
- Replaces a character by another character

- Transpose two adjacent characters

Identifier too long: As the compiler writer you must decide what the maximum length of an identifier is, and issue an error when that length is exceeded.

Errors in Syntax Analysis Phase

Error Recovery Methods

Panic mode:

Skipping the tokens until it encounters a synchronizing token. For example, while reading if it detects erroneous input it just skips until it gets the fresh statement which is correct

Phrase level:

When a parser encounters an error, it tries to take corrective measures so that the rest of inputs of statement allow the parser to parse ahead. For example, inserting a missing semicolon, replacing comma with a semicolon etc.

Examples:

int a 5; → insert = between a & 5

int a:b:c; → replace each colon by ;

Error productions:

Some common errors are known to the compiler designers that may occur in the code. In addition, the designers can create augmented grammar to be used, as productions that generate erroneous constructs when these errors are encountered.

Example: Include production with ; and also with ,
If production with , is encountered throw an error

Global Error Correction:

The parser considers the program in hand as a whole and tries to figure out what the program is intended to do and tries to find out a closest match for it, which is error-free with minimal changes.

Errors in Semantic Analysis Phase

These errors are a result of incompatible value assignment. The semantic errors that the semantic analyzer is expected to recognize are:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.
- Accessing an out of scope variable.
- Actual and formal parameter mismatch.

In code optimization, errors occur when the result is affected by the optimization.
In code generation, it shows error when code is missing etc.

(b) Explain briefly lexeme, pattern and token with the suitable examples for every possible constructs (operators, comments, white spaces, keywords, identifier, constants) 10

Ans:

Lexeme – sequence of characters

Pattern –Rule that describes the formation of string

Token – pair consisting of token and value

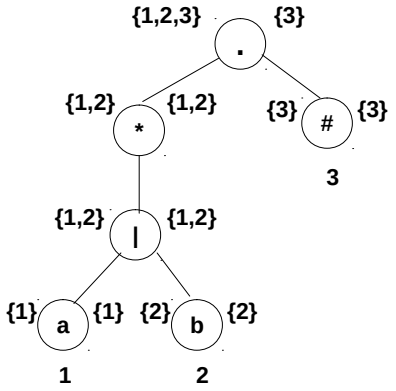
Lexeme	Pattern	Token
<,<=,>,>=,==,!=	<,<=,>,>=,==,!=	Relational operator
Num, number1, n123	l(l|d)*	Identifier
10, 10.9	[0-9]+.[0-9]+	Numeric Constant
// Single line comment	//.*	SLC
/* Multi Line comment */	/* .* */	MLC
if	if	Keyword

10 (a) Explain in detail how finite automata is used to represent tokens and perform lexical analysis with examples 8

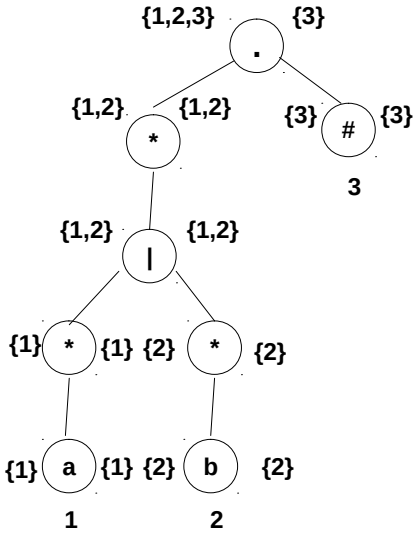
(b) Prove that the following two regular expressions are equivalent by showing that the minimum state DFA's are same. (i) (a|b)* (ii) (a*|b*)* 8

Ans:

DFA for (a|b)*



DFA for (a*|b*)*



Both DFA has the same set of follwpos sets. Hence, this will lead to the same DFA. Hence, the regular expressions are equivalent.

followpos(1)={1,2,3}

followpos(2)={1,2,3}

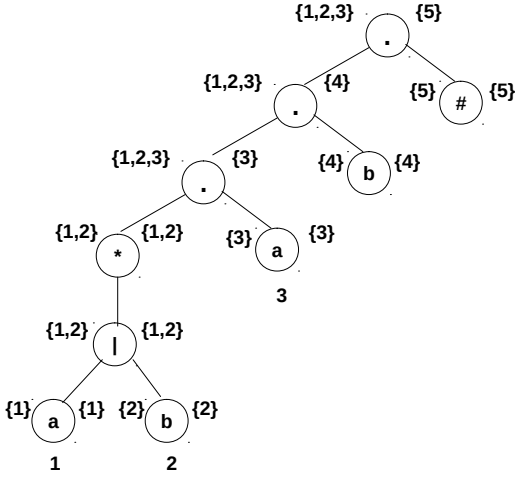
followpos(3)={#}

OR

11 (a) Construct DFA to recognize the language (a|b)*ab using direct method 8

Ans:

DFA for (a|b)*ab



followpos(1)={1,2,3}

followpos(2)={1,2,3}

followpos(3)={4}

followpos(4)={5}
followpos(5)={#}

A={1,2,3}
A on a = followpos(1) U followpos(3)={1,2,3,4} = B
A on b = followpos(2) = {1,2,3} = A

B= {1,2,3,4}
B on a = followpos(1) U followpos(3)={1,2,3,4} = B
B on b = followpos(2) U followpos(4)={1,2,3,5} = C

C={1,2,3,5}
C on a = followpos(1) U followpos(3)={1,2,3,4} = B
C on b = followpos(2) = {1,2,3} = A

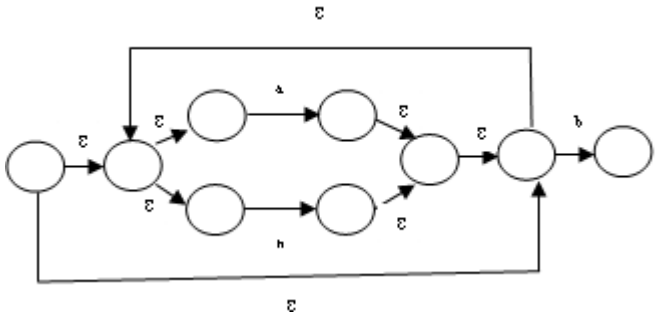
DFA

	a	B
A	B	A
B	B	C
C	B	A

(b) Construct DFA for the NFA constructed for the regular expression (a|b)*ab 8

Ans:

NFA for (a|b)*ab



Ɛ-closure(0) = {0,1,2,4,7} = A

A on a = Ɛ-closure({3,8}) = {1,2,3,4,6,7,8} = B
A on b = Ɛ-closure({5}) = {1,2,4,5,6,7} = C

B on a = Ɛ-closure({3,8}) = {1,2,3,4,6,7,8} = B
B on b = Ɛ-closure({5,9}) = {1,2,4,5,6,7,9} = D

C on a = Ɛ-closure({3,8}) = {1,2,3,4,6,7,8} = B
C on b = Ɛ-closure({5}) = {1,2,4,5,6,7} = C

D on a = Ɛ-closure({3,8}) = {1,2,3,4,6,7,8} = B
D on b = Ɛ-closure({5}) = {1,2,4,5,6,7} = C

DFA

	a	b
A	B	A
B	B	D
C	B	C
D	B	C