

**Contents**

---

<b>Contents .....</b>	<b>i</b>
<b>Question No. A.....</b>	<b>2</b>
A1.1 Identification and grouping of tokens: .....	2
A1.2 Implementation in Lex: .....	4
A1.3 Design of Context Free Grammar for each function:.....	5
A1.4 Implementation in YACC.....	9
A1.5 Results and comments.....	11

## Question No. A

Develop a compiler for a subset of C language with the following features:

- a) Minimum two data types
- b) Minimum two control statements
- c) Minimum two looping statements
- d) Input-output functions
- e) Compound statements and two-dimensional Array
- f) Assume Operators, Symbols and Reserved keywords

Your report should include:

### **A1.1 Identification and grouping of tokens:**

Tokens are the smallest elements of the program that are used in compiler to define some meaning to words. Example of tokens are keywords, identifiers, literals, strings, constants, operator, etc. Programming language is processed by giving the source code input to lexical analyzer which scans whole code and from that tokens are given output and those tokens are then input to parser that is syntax analysis. In syntax analysis parse tree is generated and checks for any syntax error in the code. To check whether the grammar is syntactically correct or not, context free grammar is defined using which syntax is checked. The parse tree is generated by syntax analyzer which is the output of syntax analysis and input to semantics analysis. Semantics analyzer gives output as abstract syntax tree or other intermediate form. Then that is given to target code generation and output is target language such as assembler.

From the given question, it was asked to create compiler for C language with some given features. For that we had to create lex and yacc file.

In lex file we had to define all the tokens and lexemes that can be used in C program. Tokens and lexemes which are used in the program are given in the following table below.

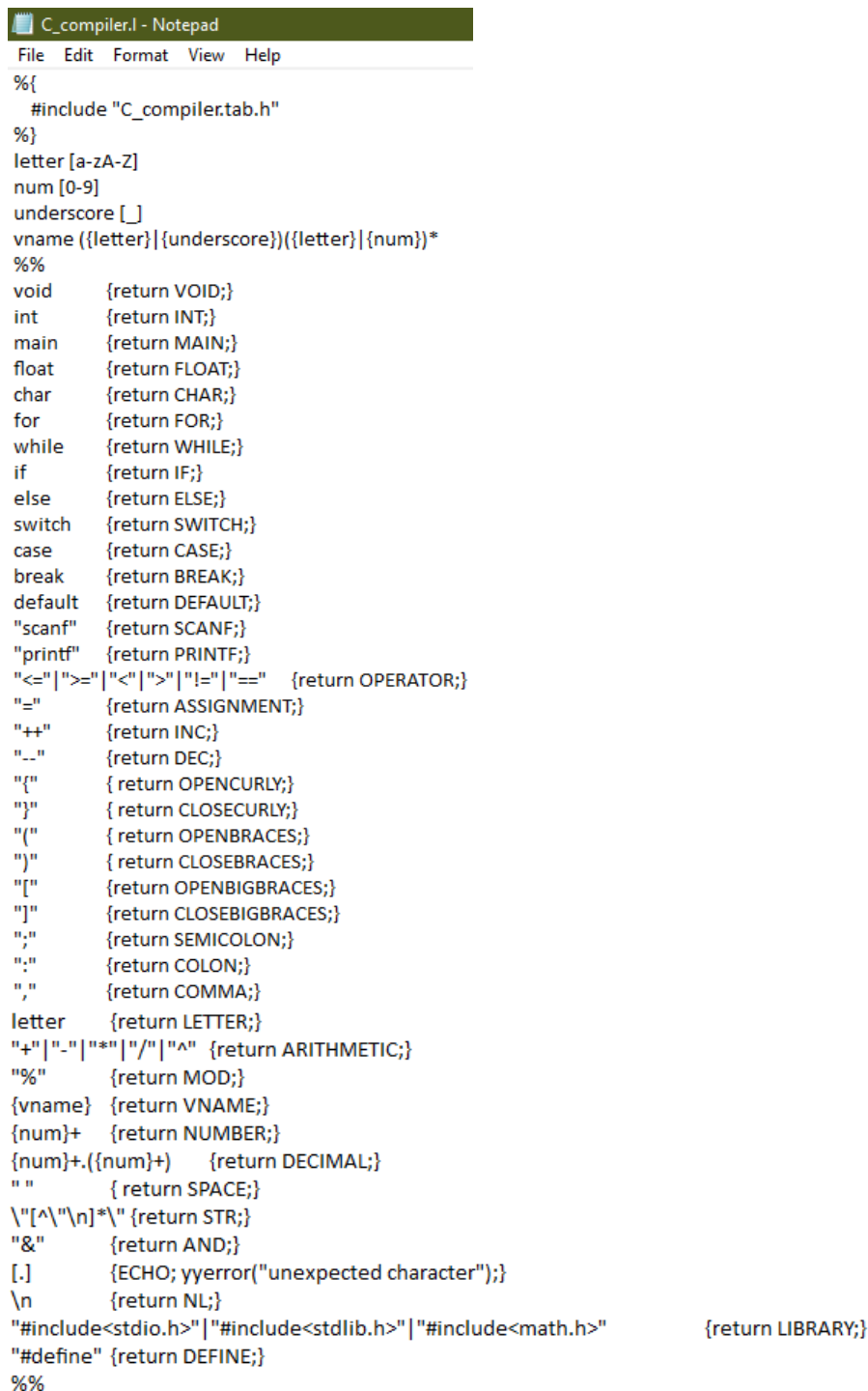
<u>LEXEMES</u>	<u>TOKENS</u>	<u>IDENTIFICATION</u>
void	Keyword	VOID
int	Keyword(Datatype)	INT
main	Identifier/Keyword	MAIN
float	Keyword(Datatype)	FLOAT
char	Keyword(Datatype)	CHAR
for	Keyword	FOR
while	Keyword	WHILE
if	Keyword	IF
else	Keyword	ELSE
switch	Keyword	SWITCH
case	Keyword	CASE

break	Keyword	BREAK
default	Keyword	DEFAULT
scanf	Keyword	SCANF
printf	Keyword	PRINTF
"<="   ">="   "<"   ">"   "!="   "=="	Operators	OPERATOR
"="	Operators	ASSIGNMENT
"++"	Operators	INC
"--"	Operators	DEC
"{"	Open Curly Bracket	OPENCURLY
"}"	Close Curly Bracket	CLOSECURLY
"("	Open Parenthesis	OPENBRACES
")"	Close Parenthesis	CLOSEBRACES
"["	Open Square Bracket	OPENBIGBRACES
"]"	Close Square Bracket	CLOSEBIGBRACES
","	Semi-Colon Operator	SEMICOLON
":"	Colon Operator	COLON
","	Comma Operator	COMOMA
letter ([a-zA-Z])	Identifier	LETTER
"+"   "-"   "*"   "/"   "^"	Operator	ARITHMETIC
"%"	Mod Operator	MOD
{vname} (((letter){underscore})(letter){num}*)	Identifier(Variable Name)	VNAME
{num}+ ([0-9])	Constants(Numbers)	NUMBER
{num}+.({num}+)	Constants(Decimal Number)	DECIMAL
\["^"\n]*\"	Identifier(String)	STR
"&"	Operator	AND

**TABLE 1**

LIBRARY, DEFINE, SPACE, NL are also defined in my lex file but that is not considered as tokens but in grammar to define that we have to define it there so that it's syntactically correct.

## A1.2 Implementation in Lex:



```
C_compiler.l - Notepad
File Edit Format View Help
%{
#include "C_compiler.tab.h"
%}
letter [a-zA-Z]
num [0-9]
underscore [_]
vname ({letter}|{underscore})({letter}|{num})*
%%
void      {return VOID;}
int       {return INT;}
main      {return MAIN;}
float     {return FLOAT;}
char      {return CHAR;}
for       {return FOR;}
while     {return WHILE;}
if        {return IF;}
else      {return ELSE;}
switch   {return SWITCH;}
case      {return CASE;}
break     {return BREAK;}
default   {return DEFAULT;}
"scanf"   {return SCANF;}
"printf"  {return PRINTF;}
"<="|">="|"<|">|"!="|"==" {return OPERATOR;}
"="       {return ASSIGNMENT;}
"++"      {return INC;}
"--"      {return DEC;}
"{"       {return OPENCURLY;}
"}"       {return CLOSECURLY;}
"("       {return OPENBRACES;}
")"       {return CLOSEBRACES;}
"["       {return OPENBIGBRACES;}
"]"       {return CLOSEBIGBRACES;}
";"       {return SEMICOLON;}
":"       {return COLON;}
","       {return COMMA;}
letter    {return LETTER;}
"+"|"-"|"*"|"/"|"^" {return ARITHMETIC;}
%"        {return MOD;}
{vname}   {return VNAME;}
{num}+    {return NUMBER;}
{num}+.{(num)+} {return DECIMAL;}
" "       {return SPACE;}
\"[^\n]*\" {return STR;}
"&"      {return AND;}
[.]       {ECHO; yyerror("unexpected character");}
\n        {return NL;}
#include<stdio.h>|include<stdlib.h>|include<math.h> {return LIBRARY;}
#define   {return DEFINE;}
%%
```

**Fig 1: Lex Program**

From the above Fig1.1 as we can see that a lex program is made in which all the tokens and lexemes are defined which are then returned to the yacc file. In this we have defined all identifiers, constants, keywords, operator. For all the lexemes, tokens are defined which is returned yacc file. These tokens are then used to generate grammar to check for syntax error in C program. These lexemes and tokens will be used in the C program. So lexical analyzer will first scan the code identify all the tokens used in the program

and then it will be sent to yacc file which will take those tokens as input and check whether the written program satisfies the grammar or not. In lex file new line, white spaces, library functions is defined separately but that is not considered as tokens but to define grammar we had to define it. `[.]` is defined to give output as unexpected character if any other characters other than the defined characters are written in the program. In my lex file only, variable name is defined as `vname` which is a letter followed by letter or digit or underscore or nothing and this variable name is directly sent as token as `VNAME`. For decimal numbers and integer values are also defined as `NUMBER` and `DECIMAL` and returned. For double quotes, it is defined as `(\"[^\"]*\")`. This will take any string which is defined between double quotes.

### A1.3 Design of Context Free Grammar for each function:

*Context free grammar* is a formal grammar which is used to generate all possible strings in a given formal language. For the features asked for C compiler we have to define a context free grammar which is written in yacc file. Syntax analyzer checks this grammar and generates parse tree for the defined grammar and checks whether the given input is syntactically correct or not. For each features, grammar is defined as follows:-

- a) Minimum two data types:- Below is the grammar defined for datatypes. From the grammar we can see that it will accept integer, character and float type datatype. According to the grammar we can define the variable with values assigned to it or without values also we can define variable. Some examples of datatype which this grammar will accept are- `int a; , int a=10; , float a; , float a=10.1; , char a; , char a="A"; .`

```

datatype: INT SPACE VNAME ASSIGNMENT NUMBER SEMICOLON
| INT SPACE VNAME SPACE ASSIGNMENT SPACE NUMBER SEMICOLON
| INT SPACE VNAME SEMICOLON
| FLOAT SPACE VNAME ASSIGNMENT DECIMAL SEMICOLON
| FLOAT SPACE VNAME SPACE ASSIGNMENT SPACE DECIMAL SEMICOLON
| FLOAT SPACE VNAME SEMICOLON
| CHAR SPACE VNAME SEMICOLON
| CHAR SPACE VNAME ASSIGNMENT STR SEMICOLON
;

```

- b) Minimum two control statements:- Below a grammar is defined in which we can see that ifelse statement and switch case will be taken input. For ifelse structure grammar is defined in such a way that in program ifelse structure can be written without else block and also and also nested if else can also be used in the program. For switch case structure, in program it can be used for integer and character type value. Default block is optional in that. In both the control statements, `BODY` terminal is defined which consists of inside structure of the control statements. Form the grammar of `BODY`, it is defined in such a way that inside control statements we can use control statements, input output functions, looping statements, expressions, datatype, array. So inside ifelse and switch we can use other features also.

```

control: ifelse
      | switchcase
      ;
ifelse: IF OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL BODY NL CLOSECURLY ifelse
      | NL ELSE OPENCURLY NL BODY NL CLOSECURLY
      |
      ;
switchcase: SWITCH OPENBRACES VNAME CLOSEBRACES OPENCURLY NL switchcase CLOSECURLY
          | CASE SPACE NUMBER COLON NL BODY NL BREAK SEMICOLON NL switchcase
          | CASE SPACE STR COLON NL BODY NL BREAK SEMICOLON NL switchcase
          | DEFAULT COLON NL BODY NL switchcase
          |
          ;
BODY: BODY loop
    | BODY control
    | BODY datatype
    | BODY EXPRESSION SEMICOLON
    | BODY inout
    | BODY array
    | BODY NL
    | BODY SPACE
    |
    ;

```

- c) Minimum two looping statements:- From the grammar defined below we can see that for loop and while loop structure is defined. For loop structure is for(initialization ; condition ; increment/decrement/expression). So from the grammar, we can see that for loop grammar is defined based on its structure. Similarly while loop's grammar is defined based in the structure of while loop which is while(condition). Initialization can be of integer type with assignment of value of any other variable. In condition we can use any operator defined as tokens. For Expression, we can use increment/ decrement operator or any other arithmetic operation. Looping statements are defined in such a way that without any body inside the loop we can define looping statements and also we can use nested loops also. Form the grammar of BODY, it is defined in such a way that inside looping statements we can use control statements, input output functions, looping statements, expressions, datatype, array. So inside looping statements we can use other features also.

```

loop: F
    | W
    ;
W: WHILE OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL BODY CLOSECURLY
  | WHILE OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL CLOSECURLY
  ;
F: FOR OPENBRACES INITIALIZATION SEMICOLON CONDITION SEMICOLON EXPRESSION
  CLOSEBRACES OPENCURLY NL BODY NL CLOSECURLY
  | FOR OPENBRACES INITIALIZATION SEMICOLON CONDITION SEMICOLON EXPRESSION
  CLOSEBRACES OPENCURLY NL CLOSECURLY

```

```

;
INITIALIZATION: INT SPACE VNAME ASSIGNMENT NUMBER
                | INT SPACE VNAME ASSIGNMENT VNAME
                | VNAME ASSIGNMENT NUMBER
                | VNAME ASSIGNMENT VNAME
;
CONDITION: VNAME OPERATOR VNAME
           | VNAME OPERATOR NUMBER
;
EXPRESSION: VNAME INC
           | VNAME DEC
           | VNAME ASSIGNMENT E
;
E: E ARITHMETIC E
  | OPENBRACES E CLOSEBRACES
  | NUMBER
  | VNAME
;
BODY: BODY loop
     | BODY control
     | BODY datatype
     | BODY EXPRESSION SEMICOLON
     | BODY inout
     | BODY array
     | BODY NL
     | BODY SPACE
     |
;

```

- d) Input-output functions:- From the below grammar we can see that it is taking printf and scan statements as input. Sample statements that can be accepted by this grammar are-  
 printf("Statement"); , printf("Value=%d",a);scanf("%d",&a); ,  
 printf("Value=%d",a[i][j]);scanf("%d",&a[i][j]); . For double quotes we have defined token STR which will accept any string written inside double quotes. Printf and Scanf can work for array also.

```

inout: PRINTF OPENBRACES STR COMMA VNAME CLOSEBRACES SEMICOLON
      | PRINTF OPENBRACES STR COMMA VNAME values CLOSEBRACES SEMICOLON
      | SCANF OPENBRACES STR COMMA AND VNAME CLOSEBRACES SEMICOLON
      | SCANF OPENBRACES STR COMMA AND VNAME values CLOSEBRACES SEMICOLON
      | PRINTF OPENBRACES STR CLOSEBRACES SEMICOLON
;

```

- e) Compound statements and two-dimensional Array:- Compound statements are statements which are written inside a braces. That whole block is what compound statement is. So in program, anything inside for, ifelse, switch case, or anything written inside a function are compound statement. So below I have defined a grammar which accepts function as input and inside function there can be any other features also. So inside main function we can use loop, ifelse, datatype, array, expression. Before function we can apply header file also which is optional. For header it can be #include<stdio.h>, #include<stdlib.h>, #define NAME NUMBER. These header are nothing but

library function. So in the function grammar it can be seen that I will take inputs as header int main(){ Statements/Body}, header void main(){Statements/Body}.

In array grammar it can be seen that it derive both 1d and 2d array for character, float and integer. Example for array input are, int a[5],int a[N], int a[4][5], char a[4][5], float f[N][N], float f[5], etc.

```

function:header INT SPACE MAIN OPENBRACES PARAMETER CLOSEBRACES OPENCURLY NL BODY
CLOSECURLY {printf("Accepted");check=1;}
    | header VOID SPACE MAIN OPENBRACES PARAMETER CLOSEBRACES OPENCURLY NL
BODY CLOSECURLY {printf("Accepted");check=1;}
    ;
PARAMETER: INT SPACE VNAME
    | FLOAT SPACE VNAME
    | CHAR SPACE VNAME
    | INT SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
    | FLOAT SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
    | CHAR SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
    |
    ;
header:header LIBRARY NL
    | header DEFINE VNAME NUMBER NL
    |
    ;
array: INT SPACE VNAME values SEMICOLON
    | FLOAT SPACE VNAME values SEMICOLON
    | CHAR SPACE VNAME values SEMICOLON
    ;
values: OPENBIGBRACES VNAME CLOSEBIGBRACES values
    | OPENBIGBRACES NUMBER CLOSEBIGBRACES values
    |
    ;

```

- f) Assume Operators, Symbols and Reserved keywords:- In our lex file as it can be seen we have assumed operators, symbols and reserve keywords such as +, \*, -, /, &, :, "", int, void, char, if, else, while, for, etc. And in yacc file we have used it in our grammar so that it can be used in our program.



#### **A1.4 Implementation in YACC:**

In yacc program it is divided into 3 parts. First part contains tokens, start symbol, left, right, or any C code if needed. In second part we define our grammar which will be used to check the input. Then in the third part we will again write C code if needed. First and third part is optional part. From the figure given below we can see that a yacc file is made in which grammar is defined to check a C program with some features in it. This yacc file will generate parse tree which will check whether the C program is syntactically correct or not. Using `yyparse()`, the program will be parsed and checked whether the program is syntactically correct or not. If it is syntactically correct then the output will be displayed as Accepted and a variable is used whose value will become 1. After main function parses the program, it goes to `yyerror` function where if any syntax error is there it will display syntax error. If the variable check value is 0 then it means that the grammar is not matched and the action is not performed. Thus the value of check didn't become 1. So using if condition it is checked if the value of check is 1 or 0. If it is 0 then syntax error. After displaying the message it will stop. The parse tree which is generated by this yacc file will be used by semantics analyser as input. If any character or operator is encountered in the program it will display unexpected character and syntax error. From the grammar we can see that start symbol is function and all the tokens are defined using `%token`. Using `%left` and `%right` I have defined operators from where to evaluate. `%left` means that it has to start evaluating from left and go towards right. `%right` means that it will start evaluating from right and goes towards left. This is basically to define precedence of operators. In `yyerror` function, its written `stderr` in print statement which means that whatever error is there in program it will automatically detect that and display the message.

```

C_compiler.y - Notepad
File Edit Format View Help

%{
void yyerror (char *s);
#include <stdio.h>
#include <stdlib.h>
int check=0;
}%
%start function
%token INT FLOAT FOR WHILE IF ELSE SWITCH CASE BREAK DEFAULT SCANF PRINTF OPENBIGBRACES
%token ASSIGNMENT INC DEC LIBRARY STR VOID MAIN
%token OPENCURLY CLOSECURLY OPENBRACES CLOSEBRACES SEMICOLON COLON CHAR CLOSEBIGBRACES
%token VNAME NUMBER DECIMAL SPACE NL COMMA OPERATOR ARITHMETIC LETTER MOD AND DEFINE
%right ASSIGNMENT ARITHMETIC
%left INC DEC OPERATOR MOD

%%
function:header INT SPACE MAIN OPENBRACES PARAMETER CLOSEBRACES OPENCURLY NL BODY CLOSECURLY {printf("Accepted");check=1;}
        |header VOID SPACE MAIN OPENBRACES PARAMETER CLOSEBRACES OPENCURLY NL BODY CLOSECURLY {printf("Accepted");check=1;}
        ;
PARAMETER: INT SPACE VNAME
        | FLOAT SPACE VNAME
        | CHAR SPACE VNAME
        | INT SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
        | FLOAT SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
        | CHAR SPACE VNAME OPENBIGBRACES CLOSEBIGBRACES
        ;
header:header LIBRARY NL
        |header DEFINE VNAME NUMBER NL
        ;
inout: PRINTF OPENBRACES STR COMMA VNAME CLOSEBRACES SEMICOLON
        | PRINTF OPENBRACES STR COMMA VNAME values CLOSEBRACES SEMICOLON
        | SCANF OPENBRACES STR COMMA AND VNAME CLOSEBRACES SEMICOLON
        | SCANF OPENBRACES STR COMMA AND VNAME values CLOSEBRACES SEMICOLON
        | PRINTF OPENBRACES STR CLOSEBRACES SEMICOLON
        ;
array: INT SPACE VNAME values SEMICOLON
        | FLOAT SPACE VNAME values SEMICOLON
        | CHAR SPACE VNAME values SEMICOLON
        ;
values: OPENBIGBRACES VNAME CLOSEBIGBRACES values
        | OPENBIGBRACES NUMBER CLOSEBIGBRACES values
        ;
datatype: INT SPACE VNAME ASSIGNMENT NUMBER SEMICOLON
        | INT SPACE VNAME SPACE ASSIGNMENT SPACE NUMBER SEMICOLON
        | INT SPACE VNAME SEMICOLON
        | FLOAT SPACE VNAME ASSIGNMENT DECIMAL SEMICOLON
        | FLOAT SPACE VNAME SPACE ASSIGNMENT SPACE DECIMAL SEMICOLON
        | FLOAT SPACE VNAME SEMICOLON
        | CHAR SPACE VNAME SEMICOLON
        | CHAR SPACE VNAME ASSIGNMENT STR SEMICOLON
        ;
control: ifelse
        | switchcase
        ;
ifelse: IF OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL BODY NL CLOSECURLY ifelse
        | NL ELSE OPENCURLY NL BODY NL CLOSECURLY
        ;
switchcase: SWITCH OPENBRACES VNAME CLOSEBRACES OPENCURLY NL switchcase CLOSECURLY
        | CASE SPACE NUMBER COLON NL BODY NL BREAK SEMICOLON NL switchcase
        | CASE SPACE STR COLON NL BODY NL BREAK SEMICOLON NL switchcase
        | DEFAULT COLON NL BODY NL switchcase
        ;
loop: F
        | W
        ;
W: WHILE OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL BODY CLOSECURLY
    | WHILE OPENBRACES CONDITION CLOSEBRACES OPENCURLY NL CLOSECURLY
    ;

```

**Fig 2:Yacc file shows grammar for datatype, input-output functions, array, control statements, while loop, compound statement**

```

F: FOR OPENBRACES INITIALIZATION SEMICOLON CONDITION SEMICOLON EXPRESSION CLOSEBRACES OPENCURLY NL BODY NL CLOSECURLY
  |FOR OPENBRACES INITIALIZATION SEMICOLON CONDITION SEMICOLON EXPRESSION CLOSEBRACES OPENCURLY NL CLOSECURLY
  ;
BODY: BODY loop
  |BODY control
  |BODY datatype
  |BODY EXPRESSION SEMICOLON
  |BODY inout
  |BODY array
  |BODY NL
  |BODY SPACE
  |
  ;
INITIALIZATION: INT SPACE VNAME ASSIGNMENT NUMBER
  |INT SPACE VNAME ASSIGNMENT VNAME
  |VNAME ASSIGNMENT NUMBER
  |VNAME ASSIGNMENT VNAME
  ;
CONDITION: VNAME OPERATOR VNAME
  |VNAME OPERATOR NUMBER
  ;
EXPRESSION: VNAME INC
  |VNAME DEC
  |VNAME ASSIGNMENT E
  ;
E:E ARITHMETIC E
  |OPENBRACES E CLOSEBRACES
  |NUMBER
  |VNAME
  ;

%%
int main(){
  yyparse();
  return 0; }

int yywrap(){
  return 1;}

void yyerror (char *s) {
  if(check==0)
  {fprintf (stderr, "%s\n", s);}}

```

**Fig 3: Yacc file shows for loop, body/statement and main function**

### A1.5 Results and comments:

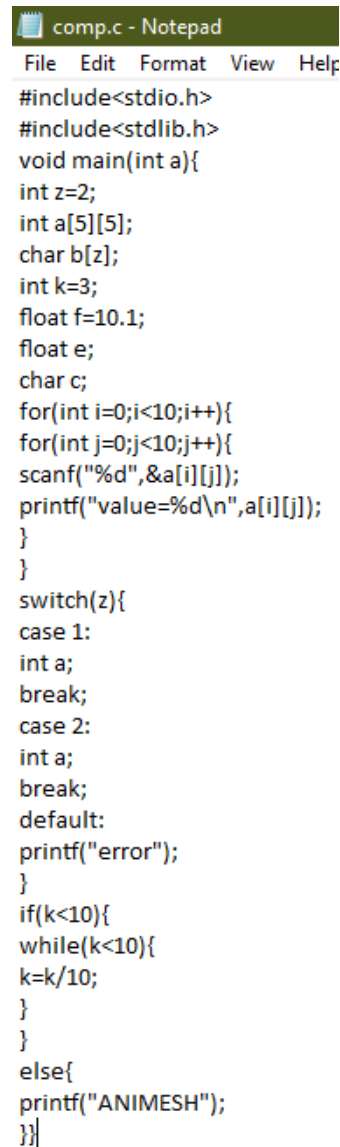
To run lex and yacc program we need to run yacc file using command, yacc -d filename.y. Then to run lex file use command, lex filename.l and then run gcc compiler, gcc lex.yy.c filename.tab.c . Then give the file as input to the compiler by running file a.exe<comp.c .

#### 1) When input is given without any error:-

In fig 4 we can see that a C program with all the features is given input and then checked whether it is syntactically correct or not. In fig 5 a command window is shown in which a yacc and lex file is run.

After lex and yacc file is run, gcc compiler is run. Then comp.c is given input and checked whether the input file satisfies the derive grammar. Fig 4 is comp.c file i.e. input file. In command window we can

see that Accepted message is displayed which means that there is no syntax error in the program and thus it is satisfying the grammar. The input C program contains all the features that is asked in the question i.e. loops, control statements, datatypes, array, compound statements, operators, reserve words and keywords.



```
comp.c - Notepad
File Edit Format View Help
#include<stdio.h>
#include<stdlib.h>
void main(int a){
int z=2;
int a[5][5];
char b[z];
int k=3;
float f=10.1;
float e;
char c;
for(int i=0;i<10;i++){
for(int j=0;j<10;j++){
scanf("%d",&a[i][j]);
printf("value=%d\n",a[i][j]);
}
}
switch(z){
case 1:
int a;
break;
case 2:
int a;
break;
default:
printf("error");
}
if(k<10){
while(k<10){
k=k/10;
}
}
else{
printf("ANIMESH");
}}|
```

**Fig 4: A C-program given input to check whether it is syntactically correct or not**

```
C:\WINDOWS\system32\cmd.exe

C:\Users\animesh srivastava\Desktop\Compiler>yacc -d C_compiler.y
C_compiler.y: conflicts: 147 shift/reduce, 143 reduce/reduce

C:\Users\animesh srivastava\Desktop\Compiler>lex C_compiler.l

C:\Users\animesh srivastava\Desktop\Compiler>gcc lex.yy.c C_compiler.tab.c
C_compiler.l: In function 'yylex':
C_compiler.l:47:8: warning: implicit declaration of function 'yyerror'; did you mean 'perror'? [-Wimplicit-fu
aration]
[.] {ECHO; yyerror("unexpected character");}
      ^~~~~~
      perror
C_compiler.tab.c: In function 'yyparse':
C_compiler.tab.c:840:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
# define YYLEX yylex ()
      ^
C_compiler.tab.c:1500:16: note: in expansion of macro 'YYLEX'
    yychar = YYLEX;
              ^~~~~
C:\Users\animesh srivastava\Desktop\Compiler>a.exe<comp.c
Accepted
```

**Fig 5: Output of the program where the input file is Accepted which means it is syntactically correct**

**2) When input is given with error:-**

In figure 6 we can see that a program is given as input to the compiler to check whether the program is syntactically correct or not i.e. whether it satisfies the grammar or not. From figure 7 we can see that in command window it shows syntax error. Which means that the program is syntactically not correct and thus doesn't satisfies the derived grammar. In fig 6 errors are been highlighted. As it can be seen that there are 5 errors in the program due to which it didn't satisfy the grammar and it showed syntax error. 1<sup>st</sup> error is that integer value is assigned with decimal values, 2<sup>nd</sup> is that character is assigned without double quotes, 3<sup>rd</sup> error is that after I loop closing braces, so characters are there which are invalid, 4<sup>th</sup> error is semicolon are curly braces and 5<sup>th</sup> error is also semi colon after closed curly braces. Thus program is not accepted by the compiler because of these errors.

```

comp.c - Notepad
File Edit Format View Help
#include<stdio.h>
#include<stdlib.h>
int main(int a){
int z=2;
int a[5][5];
char b[z];
int k=3.6;
float f=10.1;
float e;
char c=A;
for(int i=0;i<10;i++){
for(int j=0;j<10;j++){
scanf("%d",&a[i][j]);
printf("value=%d\n",a[i][j]);
}
}8s5;
switch(z){
case 1:
int a;
break;
case 2:
int a;
break;
default:
printf("error");
};
if(k<10){
while(k<10){
k=k/10;
}
}
else{
printf("ANIMESH");
}
}

```

**Fig 6: A C-program given input to compiler to check whether it is syntactically correct or not**

```

C:\WINDOWS\system32\cmd.exe
C:\Users\animesh srivastava\Desktop\Compiler>yacc -d C_compiler.y
C_compiler.y: conflicts: 147 shift/reduce, 143 reduce/reduce

C:\Users\animesh srivastava\Desktop\Compiler>lex C_compiler.l

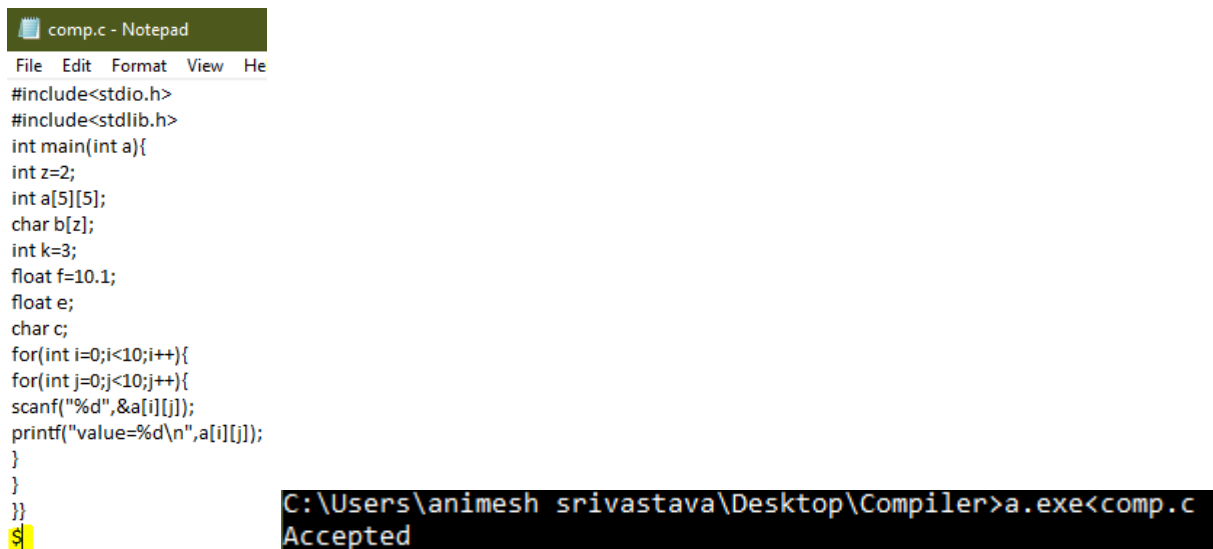
C:\Users\animesh srivastava\Desktop\Compiler>gcc lex.yy.c C_compiler.tab.c
C_compiler.l: In function 'yylex':
C_compiler.l:47:8: warning: implicit declaration of function 'yyerror'; did you mean 'perror'? [-Wimplicit-fu
aration]
[.] {ECHO; yyerror("unexpected character");}
      ^~~~~~
      perror
C_compiler.tab.c: In function 'yyparse':
C_compiler.tab.c:840:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
# define YYLEX yylex ()
      ^
C_compiler.tab.c:1500:16: note: in expansion of macro 'YYLEX'
yychar = YYLEX;
      ^~~~~~

C:\Users\animesh srivastava\Desktop\Compiler>a.exe<comp.c
Accepted
C:\Users\animesh srivastava\Desktop\Compiler>a.exe<comp.c
syntax error

```

**Fig 7: Output of the program which shows that the input C-program is syntactically wrong which means that it is not accepted by the compiler**

Thus a lex and yacc program is made for checking whether the C-program is syntactically correct or not. IN lex file all the lexemes and tokens are defined and those tokens were sent to yacc. In yacc file context free grammar is defined for C compilers which can have features as loops, datatypes, control statements, compound statements, array, reserve keywords. Lexical analyzer scans the whole file and checks for all the tokens in the file. Then this token is sent to yacc file which checks for syntax error and generates parse tree. If it is syntactically correct then a message is displayed as Accept otherwise syntax error message is displayed. Thus from the above output we can see that desired output is obtained and the C program is checked whether the program is syntactically correct or not. There are some limitations in grammar also that after it reads the grammar and finds it correct it will perform the action and stop. Even if there is any special character or anything after the main function curly braces ends, from the new line it will not read any input because till closed curly braces it reads it correct and thus performs following action and stops. It doesn't read after that. This is the only limitation which I've found in my program. The below figure shows C program which is given input to compiler which is accepted but there is a \$ symbol in the new line after the closed curly brace of the main function which is wrong. Till braces it read correctly and performed action and stopped. After that it didn't read the input and thus didn't detect any error which is the limitation in this program.



The image shows a Notepad window titled 'comp.c - Notepad' containing a C program. The program includes `<stdio.h>` and `<stdlib.h>`, and defines a `main` function with several variables and nested loops. A yellow cursor is positioned at the end of the first closing curly brace of the `main` function. To the right, a terminal window shows the command `C:\Users\animesh srivastava\Desktop\Compiler>a.exe<comp.c` and the output `Accepted`.

```
#include<stdio.h>
#include<stdlib.h>
int main(int a){
int z=2;
int a[5][5];
char b[z];
int k=3;
float f=10.1;
float e;
char c;
for(int i=0;i<10;i++){
for(int j=0;j<10;j++){
scanf("%d",&a[i][j]);
printf("value=%d\n",a[i][j]);
}
}
}}
```

C:\Users\animesh srivastava\Desktop\Compiler>a.exe<comp.c  
Accepted