1. WAP in C to print "Hello world" in Ubuntu operating system using gcc compiles and write the commands to edit a program, compile a program, create object file and run a program in gcc compiles.

C program: hello.c

```
#include <stdio.h>
int main()
{
  printf("Hello world\n");
  return 0;
}
```

⇒ steps and commands in Ubuntu (using gcc compiles)

(i) Create and Edit a C program

command to create/Edit using nano

```
nginx
nano hello.c
```

- Write the program
- Press ctrl+O to save
- Enter
- ctrl+x to exit

(ii) Compile the program

To compile a C file using gcc:

```
nginx
gcc hello.c
```

This creates an executable file named a.out

(iii) Create an object file(.o)

To generate an object file:

```
r
gcc -c hello.c
```

This creates: hello.o

(iv) Generate Executable File From object file

```
nginn
gcc hello.o -o hello
```

(v) Run the program

gf executable name is a.out:

```
bash
./a.out
```

Outpnt

```
nginn
Hello world
```

2. WAP to print "Hello World" using GOTO in ubuntu operating system using gcc compiler.

⇒ Program in C:

```
#include<stdio.h>
int main()
{
go to print;
print:
    printf("Hello world\n");
    return 0;
}
```

⇒ steps to compile and run in Ubuntu (gcc):

(i) Save the File as hello.c

(ii) Open terminal and compile using:

```
nginx
gcc hello.c -o hello
```

(iii) Run the program:

```
bash
./hello
```

Output:

```
nginx
Hello World
```

3 WAP to perform addition, multiplication and division of two numbers using SWITCH case in Ubuntu Operating system using Gcc Compiler.

⇒ Program in C:

```
#include <stdio.h>
int main()
{
  int choice;
  Float a, b, result;
  PrintF("Enter two numbers:");
  ScanF("%F%F", &a, &b);
  printF("\n choose operation:\n");
  printf("1. Addition\n");
  printF("2. Multiplication\n");
  printf("3. Division\n");
  printF("Enter your choice:");
  ScanF("%d", &choice);
 Switch(choice)
 {
   case 1:
      result = a+b;
      printF("Addition = %.2f\n", result);
      break;
   case 2:
      result = a*b;
      printF("Multiplication = %.2F\n", result);
      break;
```

```
case 3:
 if (b!=0)
   {
    result = a/b;
    printf("Division= %.2f\n", result);
   }
 else
   {
    printf("Error! Division by zero not
                                allowed\n");
   }
 break;
 default:
   printf("Invalid choice!\n");
}
 return 0;
}
```

⇒ Steps to compile and Run in Ubuntu (acc):

(i) save the file as calc.c

(ii) Open terminal and type:

```
nginn
gcc calc.c -o calc
```

(iii) Run the program:

```
bash
./calc
```

Output :-

* Case 1 : Addition

Input :

```
Enter two numbers : 8 2
choose operation :
1. Addition
2. Multiplication
3. Division
Enter your choice : 1
```

Output :

```
Addition : 10.00
```

* Case 2 : Multiplication

Input :

```
Enter two numbers : 8 2
choose operation :
1. Addition
2. Multiplication
3. Division
Enter your choice : 2
```

Output :

```
Multiplication : 16.00
```

4. WAP in C to implement the transition diagram for keywords: BEGIN.

```
#include <stdio.h>
#include <string.h>
int main()
{
 char str[20];
 int state = 0;
 int i = 0;
 printf("Enter a string: ");
 scanf("%s", str);
 while (str[i] != '\0')
 {
   switch(state)
   {
    case 0:
        if (str[i] == 'B') state = 1;
        else state = -1;
        break;

    case 1:
        if (str[i] == 'E') state = 2;
        else state = -1;
        break;

    case 2:
        if (str[i] == 'G') state = 3;
        else state = -1;
        break;
```

* Case 3: Division

Input:

```
Enter two numbers : 8 2
Choose operation:
1. Addition
2. Multiplication
3. Division
Enter your choice: 3
```

Output:-

```
Division: 4.00
```

```
        case 3:
            if (str[i] == 'I') state = 4;
            else state = -1;
            break;
        case 4:
            if (str[i] == 'N') state = 5;
            else state = -1;
            break;
    }
    if (state == -1) break;
    i++;
}
if (state == 5 && i == 5)
    printf("Valid keyword : BEGIN\n");
else
    printf("Invalid keyword\n");
return 0;
}
```

* Output :

. Input :

```
powershell
BEGIN
```

. Output

```
pgsql
valid keyword : BEGIN
```

. Input :

```
nginn
BEGI
```

```
nginn
Invalid keyword
```

5 WAP in C to implement combined transition diagram for keywords: BEGIN, END, IF, THEN, ELSE.

⇒

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[20];
int state = 0, i = 0;
printf("Enter a string:");
scanf("%s", str);
while(str[i] != '\0')
{
switch(state)
{
case 0:
    if(str[i] == 'B') state = 1;
    else if(str[i] == 'E') state = 6;
    else if(str[i] == 'I') state = 12;
    else if(str[i] == 'T') state = 15;
    else state = -1;

case 1:
    if(str[i] == 'E') state = 2;
    else state = -1;
    break;

case 2:
    if(str[i] == 'G') state = 3;
    else state = -1;
    break;
```

```
Case 3:
IF (str[i] == 'I') state = 4;
else state = -1;
break;
Case 4:
IF (str[i] == 'N') state = 5;
 else state = -1;
 break;
Case 6:
 IF (str[i] == 'N') state = 7;
 else IF (str[i] = 'L') state = 9;
 else state = -1;
 break;
Case 7:
  IF (str[i] == 'D') state = 8;
  else state = -1;
  break;
 case 9:
  IF (str[i] == 'S') state = 10;
  else state = -1;
  break;
 case 10:
  IF (str[i] == 'E') state = 11;
  else state = -1;
  break;
  case 12:
  IF (str[i] == 'F') state = 13;
  else state = -1;
  break;
```

```
Case 15:
    if(str[i]=='H') state=16;
    else state=-1;
    break;
Case 16:
    if(str[i]=='E') state=17;
    else state=-1;
    break;
Case 17:
    if(str[i]=='N') state=18;
    else state=-1;
    break;
}
if(state==-1) break;
i++;
}
if(state==5 && i==5)
  printf("keyword: BEGIN\n");
else if(state==8 && i==3)
  printf("keyword: END\n");
else if(state==11 & i==4)
  printf("keyword: ELSE\n");
else if(state==13 && i==2)
  printf("keyword: IF\n");
else if(state=18 && i==4)
  printf("keyword: THEN\n");
else
  printf("Invalid keyword\n");
return 0;
}
```

⇒ Output : For each case:

(i) Input:

```
powershell
Begin
```

Output:

```
makefile
keyword : BEGIN
```

(ii) Input:

```
powershell
END
```

Output:

```
vbnet
keyword : END
```

(iii) Input:

```
powershell
ELSE
```

Output

```
vbnet
keyword : ELSE
```

(iv) Input

```
nginx
IF
```

Output

```
makefile
keyword : IF
```

(v) Input

```
nginx
THEN
```

Output

```
makefile
keyword : THEN
```

6 WAP in FLEX to identify keywords : BEGIN, END, IF, THEN, ELSE.

⇒ Len

```
%{
#include<stdio.h>
%}
%%
BEGIN { printf("keyword : BEGIN\n"); }
END   { printf("keyword : END\n"); }
THEN  { printf("keyword : THEN\n"); }
ELSE  { printf("keyword : ELSE\n"); }
IF    { printf("keyword : IF\n"); }
[a-zA-Z]+ { printf("Identifier : %s\n",
                    yytext); }
[\t\n]+;
        { printf("Other : %s\n", yytext); }
%%
int main(void)
{ printf("Enter input (ctrl+D to end on
                    Linux): \n");
  yylex();
  return 0;
}
int yywrap(void)
{ return 1;
}
```

3) Run the program:

(i) save as keyword.l

(iii) Run:

```
bash
Flex keyword.l
gcc lex.yy.c -lfl -o keyword
./keyword
```

(iii)

```
powershell
BEGIN n IF y THEN END ELSE
```

(iv) Output:

```
yaml
keyword : BEGIN
Identifier : n
keyword : IF
Identifier: y
keyword : THEN
keyword : END
keyword : ELSE
```

7 Using Flex, write a lexical analyses for Identifies:

Identifies:

Letter → [a-z, A-Z]

digit → [0-9]

id → letter (letter | digit)*

The lexer shall recognize identifiers. An identifier is a sequence of letters and digits, starting with a letter.

=)

```
lex
%{
#include<stdio.h>
%}
%%
[a-zA-Z][a-zA-Z0-9]* { printf("Identifier : %s\n",
                       yytext); }

[0-9]+ { printf("Number: %s\n", yytext); }

[ \t\n]+ ;
    . { printf("Invalid character : %s\n", yytext); }

%%
int main()
{ printf("Enter input (ctrl+D to stop): \n");
  yylex();
  return 0;
}
int yywrap()
{ return 1;
}
```

⇒ Output :

Input :

```
nginx
hello123 98 a7245
```

Output :

```
Yaml
Identifier : hello123
Number : 98
Identifier : a7245
```

8 WAP in Flex to implement the combined transition diagram for Integer constants Floating point numbers and Relational operators: <, <=, =, <>, >, >= that are commonly used in any high level language.

=) lex

```
%{
#include<stdio.h>
%}
%%
[0-9]+"."[0-9]+{printf("Floating constant : %s\n",yytext);}
[0-9]+   {printf("Integer constant : %s\n",yytext);}
"<="     {printf("Relational operator : <=\n");}
">="     {printf("Relational operator : >=\n");}
"<>"     {printf("Relational operator : <>\n");}
"<"      {printf("Relational operator : <\n");}
">"      {printf("Relational operator : >\n");}
"="      {printf("Relational operator : =\n");}
[\t\n]+  ;
.        {printf("Other symbol : %s\n",yytext);}
%%
int main(void)
{printf("Enter input (ctrl+d to end on linux):\n");
  yylex();
  return 0;
}int yywrap(void)
 {return 1;
}
```

Output

Input:

```
test
10 3.14 <= > <> = 25 7.0 >=
```

Output:-

```
test
Integer constant : 10
Floating constant : 3.14
Relational operator : <=
Relational operator : >
Relational operator : <>
Relational operator : =
Integer constant : 25
Floating constant : 7.0
Relational operator : >=
```

9 Write a YACC program to parse arithmetic expression in a programming language. Also evaluate the arithmetic expression.

⇒
```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
void yyerror(const char *s);
%}
%token Number
%left '+' '-'
%left '*' '/'
%%
input: expr '\n' { printf("Result = %d\n", $1); }
    ;
expr: expr '+' expr { $$ = $1 + $3; }
    | expr '-' expr { $$ = $1 - $3; }
    | expr '*' expr { $$ = $1 * $3; }
    | expr '/' expr { $$ = $1 / $3; }
    | '(' expr ')' { $$ = $2; }
    | Number { $$ = $1; }
%%
#include <ctype.h>
int yylex(void)
{ int c;
  while((c=getchar())==' '||c=='\t')
    ;
```

```
if (isdigit(c))
{
    int val = 0;
    while (isdigit(c))
    {
        val = val * 10 + (c - '0');
        c = getchar();
    }
    ungetc(c, stdin);
    yyval = val;
    return NUMBER;
}
return c;
}
void yyerror(const char *s);
{
printf("syntax error: %s\n", s);
}
int main()
{
printf("Enter an arithmetic expression:\n");
yyparse();
return 0;
}
```

Output:-

Input

```
text
3+4*5
```

Output

```
text
Enter an arithmetic expression: 3+4*5
Result = 23
```

10 Write a C program for FIRST and FOLLOW computation for the following grammar:

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define MAX 10
int n=6;
char prod[6][10]={
    "E=E+T",
    "E=T",
    "T=T*F",
    "T=F",
    "F=(E)",
    "F=@;" /* we take id as token 'i' for simplicity */
};
char FirstRes[20], FollowRes[20];
int FirstIdx, FollowIdx;
void addToSet(char set[], int *idx, char c)
{
    int i;
    for(i=0; i<*idx; i++)
        if(set[i]==c) return;
    set[(*idx)++]=c;
}
```

```
void FIRST (char c)
{
  int i, j;
  IF (! isupper (c))
    {
      add Toset (First Res, & First Idx, c);
      return;
    }
  For (i = 0; i < n; i++)
    {
      IF (prod [i] [0] == c)
        {
          char rhs 0 = prod [i] [2];
          IF (rhs0 == c) continue;
          IF (! isupper (rhs0))
          {
            add Toset (First Res, & First Idx, rhs0);
          }
          else
          {
            First (rsho);
          }
        }
    }
}
void FOLLOW (char c)
{ int i, j;
  if (c == 'E')
      add Toset (follow Res, & Follow Idx, '$');
```

```
for(i=0; i<n ; i++)
 {
  for(j=2; prod[i][j]!='\0'; j++)
  {
   if(prod[i][j]==(c)
    {
     if(prod[i][j+1]!='\0')
      {
       firstIdn=0;
       FIRST(prod[i][j+1]);
       for(int k=0; k<firstIdn; k++)
        if(firstRes[k]!='#')
         addToSet(followRes, &followIdn,
                                firstRes[k]);

        }
       if(prod[i][j+1]=='\0'&& prod[i][0]!=c
       {
        FOLLOW(prod[i][0]);
       }
      }
    }
  }
 }
int main()
 {int i;
  firstIdn=0;
  FIRST("FIRST(E)={");
  for(i=0; i<firstIdn; i++)
   printf("%c", firstRes[i]);
```

```
Printf("}\n");
First Idn = 0;
FIRST('T');
Printf("FIRST(T) = {");
for(i=0; i< First Idn ; i++)
  Printf("%c", First Res[i]);
printf("}\n");

First Idn = 0;
FIRST('F');
printf("FIRST(F) = {");
for(i=0; i< First Idn ; i++)
  printf("%c, First Res[i]);
printf("}\n");

Follow Idn = 0;
FOLLOW('E');
printf("FOLLOW(E) = {");
for(i=0; i< Follow Idn; i++)
  printf("%c", Follow Res[i]);
printf("}\n");

Follow Idn = 0;
FOLLOW('T');
printf("FOLLOW(T) = {");
for(i=0; i< Follow Idn ; i++)
  printf("%c", Follow Res[i]);
printf("}\n");
```

```
FollowIdn = 0;
FOLLOW('F');
printf("FOLLOW(F) = { ");
for(i=0; i<FollowIdn; i++)
    printf(" %c ", FollowRes[i]);
printf(" }\n");
return 0;
}
```

Output

First sets

FIRST(E) = { c, id }
FIRST(T) = { c, id }
FIRST(F) = { c, id }

Follow sets

FOLLOW(E) = { $,), + }
FOLLOW(T) = { $,), +, * }
FOLLOW(F) = { $,), +, * }