1. **Generate a YACC specification to recognize a valid identifier that starts with a letter followed by any number of letters or digits.**

.l

```
%{
   #include "y.tab.h"
%}

%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9]                  return digit;
.                     return yytext[0];
\n                     return 0;
%%
int yywrap()
{
return 1;
}
```

.y

```
%{
   #include <stdio.h>
   int valid = 1;
%}
%token digit letter
%%
start: letter s
s: letter s | digit s |;
%%
int yyerror()
{
   printf("Invalid identifier.\n");
   valid = 0;
   return 0;
}
int main()
{
   printf("Enter identifier: ");
   yyparse();
   if(valid)
   {
      printf("Valid identifier.\n");
   }
}
```

2. YACC PROGRAM TO VALID EXPRESSION

VALIDEXP.l

```
%{
#include"y.tab.h"
%}

%%
[0-9]+ {return NUMBER;}
[a-zA-Z][a-zA-Z0-9_]* {return ID;}
\n {return NL;}
. {return yytext[0];}
%%
```

Validexp.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}

%token NUMBER ID NL
%left '+' '-'
%left '*' '/'

%%
stmt: exp NL {printf("valid expression\n"); exit(0);}
;
exp: exp '+' exp | exp '-' exp | exp '*' exp | exp '/' exp | '(' exp ')' | ID | NUMBER
;
%%

int yyerror(char *msg)
{
 printf("Invalid expression\n");
 exit(0);
}

main()
{
 printf("enter the expression: \n");
 yyparse();
}
```

3. CALCULATOR USING LEX AND YACC

calcu.y

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
%}

%token NUMBER

%left '+' '-'

%left '*' '/' '%'

%left '(' ')'

/* Rule Section */
%%

ArithmeticExpression: E{

                        printf("\nResult=%d\n", $$);

                        return 0;

                        };
E:E'+'E {$$=$1+$3;}

|E'-'E {$$=$1-$3;}

|E'*'E {$$=$1*$3;}

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

|'('E')' {$$=$2;}

| NUMBER {$$=$1;}

;

%%

//driver code
void main()
{
```

```
printf("Enter Any Arithmetic Expression which can have operations Addition,Subtraction,
Multiplication, Division,Modulus and Round brackets:\n");

yyparse();
if(flag==0)
printf("\nEntered arithmetic expression is Valid\n\n");
}

void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

Clacu.l

```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

/* Rule Section */
%%
[0-9]+ {
                    yylval=atoi(yytext);
                    return NUMBER;

          }
[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()
{
return 1;
}
```

4. TOP-DOWN PARSER/ RECURSIVE DESCEND PARSER

```
#include<stdio.h>
#include<string.h>
char input[10];
int i=0,error=0;
void E();
void T();
void Eprime();
void Tprime();
void F();
void main()
{
 printf("Recursive Descend parser for arithmetic expn containing + and *(eg:a+a*a)\n");
 printf("Enter an arithmetic expression : ");
 scanf("%s",input);
 E();
 if(strlen(input)==i&&error==0)
 printf("\nAccepted..!!!\n");
 else
 printf("\nRejected..!!!\n");
}
void E()
{
 T();
 Eprime();
}
void Eprime()
{
 if(input[i]=='+')
 {
 i++;
 T();
 Eprime();
 }
}
void T()
{
 F();
 Tprime();
}
void Tprime()
{
 if(input[i]=='*')
 {
 i++;
 F();
 Tprime();
 }
}
```

```
void F()
{
 if(input[i]=='a')
 i++;
 else if(input[i]=='(')
 {
 i++;
 E();
 if(input[i]==')')
 i++;
 else
 error=1;
 }
 else
 error=1;
}
```

5. OPERATOE PRECEDENCE PARCER

```
#include <stdio.h>
#include <string.h>

void main() {
   char stack[20], ip[20], opt[10][10][2], ter[10];
   int i, j, k, n, top = 0, col, row;

   for (i = 0; i < 10; i++) {
      stack[i] = '\0';
      ip[i] = '\0';
      for (j = 0; j < 10; j++) {
         opt[i][j][0] = '\0';
         opt[i][j][1] = '\0';
      }
   }

   printf("Enter the no. of terminals:\n");
   scanf("%d", &n);

   printf("Enter the terminals:\n");
   scanf("%s", ter);

   printf("Enter the table values:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         printf("Enter the value for %c %c:", ter[i], ter[j]);
         scanf("%s", opt[i][j]);
      }
   }

   printf("\n**** OPERATOR PRECEDENCE TABLE ****\n");
   for (i = 0; i < n; i++) {
      printf("\t%c", ter[i]);
   }
   printf("\n");
   for (i = 0; i < n; i++) {
      printf("%c", ter[i]);
      for (j = 0; j < n; j++) {
         printf("\t%s", opt[i][j]);
      }
      printf("\n");
   }

   stack[top] = '$';

   printf("Enter the input string:");
   scanf("%s", ip);
```

```
    i = 0;

    printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
    printf("\n%s\t\t\t%s\t\t\t", stack, ip);

    while (i <= strlen(ip)) {
        for (k = 0; k < n; k++) {
            if (stack[top] == ter[k])
                col = k;
            if (ip[i] == ter[k])
                row = k;
        }

        if ((stack[top] == '$') && (ip[i] == '$')) {
            printf("String is accepted\n");
            break;
        } else if ((opt[col][row][0] == '<') || (opt[col][row][0] == '=')) {
            stack[++top] = opt[col][row][0];
            stack[++top] = ip[i];
            printf("Shift %c", ip[i]);
            i++;
        } else {
            if (opt[col][row][0] == '>') {
                while (stack[top] != '<') {
                    --top;
                }
                top = top - 1;
                printf("Reduce");
            } else {
                printf("\nString is not accepted");
                break;
            }
        }
        printf("\n");
        for (k = 0; k <= top; k++) {
            printf("%c", stack[k]);
        }
        printf("\t\t\t");
        for (k = i; k < strlen(ip); k++) {
            printf("%c", ip[k]);
        }
        printf("\t\t\t");
    }
}
```

6. **EPSILON CLOSURE**

```
#include<stdio.h>
#include<string.h>

char result[20][20],copy[3],states[20][20];

void add_state(char a[3],int i){
        strcpy(result[i],a);
}

void display(int n){
        int k=0;
        printf("\n Epsilon closure of %s = { ",copy);
        while(k < n){
                printf(" %s",result[k]);
                k++;
        }
        printf(" } \n");
}

int main(){
   FILE *INPUT;
   INPUT=fopen("input.dat","r");
   char state[3];
   int end,i=0,n,k=0;
   char state1[3],input[3],state2[3];
   printf("\n Enter the no of states: ");
   scanf("%d",&n);
   printf("\n Enter the states \n");
   for(k=0;k<3;k++){
                scanf("%s",states[k]);

        }

        for( k=0;k<n;k++){
                i=0;
                strcpy(state,states[k]);
                strcpy(copy,state);
                add_state(state,i++);
                while(1){
                        end = fscanf(INPUT,"%s%s%s",state1,input,state2);
                        if (end == EOF ){
                                break;
                        }

                        if( strcmp(state,state1) == 0 ){
                                if( strcmp(input,"e") == 0 ) {
                                        add_state(state2,i++);
```

```
                                        strcpy(state, state2);
                                    }
                                }

                        }
                        display(i);
                        rewind(INPUT);
                }

   return 0;
}
```

(input.dat file indakki ee data athil add cheyth folderil save cheyanam)
Input.dat

```
q0 0 q0
q0 1 q1
q0 e q1
q1 1 q2
q1 e q2
```

7. INTERMEDIATE CODE GENERATOR

```
#include <stdio.h>
#include <string.h>

int i = 1, j = 0, no = 0, tmpch = 90;

char str[100], left[15], right[15];

void findopr();
void explore();
void fleft(int);
void fright(int);

struct exp
{
   int pos;
   char op;
} k[15];

int main()
{
   printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
   printf("Enter the Expression : ");
   scanf("%s", str);
   printf("The intermediate code:\n");
   findopr();
   explore();
   return 0;
}

void findopr()
{
   for (i = 0; str[i] != '\0'; i++)
   {
      if (str[i] == ':')
      {
         k[j].pos = i;
         k[j++].op = ':';
      }
   }
   for (i = 0; str[i] != '\0'; i++)
   {
      if (str[i] == '/')
      {
         k[j].pos = i;
         k[j++].op = '/';
      }
   }
   for (i = 0; str[i] != '\0'; i++)
```

```
    {
        if (str[i] == '*')
        {
            k[j].pos = i;
            k[j++].op = '*';
        }
    }
    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] == '+')
        {
            k[j].pos = i;
            k[j++].op = '+';
        }
    }
    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] == '-')
        {
            k[j].pos = i;
            k[j++].op = '-';
        }
    }
}

void explore()
{
    i = 1;
    while (k[i].op != '\0')
    {
        fleft(k[i].pos);
        fright(k[i].pos);
        str[k[i].pos] = tmpch--;

        printf("\t%c := %s%c%s\t\t", str[k[i].pos], left, k[i].op, right);
        printf("\n");
        i++;
    }
    fright(-1);

    if (no == 0)
    {
        fleft(strlen(str));
        printf("\t%s := %s", right, left);
    }
    else
    {
        printf("\t%s := %c", right, str[k[--i].pos]);
    }

    printf("\n");
```

```
}

void fleft(int x)
{
   int w = 0, flag = 0;
   x--;
   while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '=' && str[x] != '\0' && str[x] != '-' && str[x]
!= '/' && str[x] != ':')
   {
      if (str[x] != '$' && flag == 0)
      {
         left[w++] = str[x];
         left[w] = '\0';
         str[x] = '$';
         flag = 1;
      }
      x--;
   }
}

void fright(int x)
{
   int w = 0, flag = 0;
   x++;
   while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '\0' && str[x] != '=' && str[x] != ':' && str[x]
!= '-' && str[x] != '/')
   {
      if (str[x] != '$' && flag == 0)
      {
         right[w++] = str[x];
         right[w] = '\0';
         str[x] = '$';
         flag = 1;
      }
      x++;
   }
}
```

8. DFA

1. DFA to accept strings that start with ab over input alphabets

```
#include <stdio.h>
#include <string.h>

int main() {
   char str[100];

   printf("DFA to accept strings that start with ab over input alphabets (a, b)\n");
   printf("Enter string: ");
   scanf("%s", str);

   // Check if the string has at least two characters
   if (strlen(str) >= 2 && str[0] == 'a' && str[1] == 'b') {
      printf("String accepted\n");
   } else {
      printf("String rejected\n");
   }

   return 0;
}
```

2. DFA to accept strings that start with aa or bb over input alphabets

```
#include <stdio.h>
#include <string.h>

int main() {
   int i;
   char str[100];

   printf("DFA to accept strings that start with aa or bb over input alphabets (a, b)\n");
   printf("Enter string: ");

   // Use %99s to prevent buffer overflow
   scanf("%99s", str);

   if (str[0] == 'a' && str[1] == 'a') {
      printf("String accepted\n");
   } else if (str[0] == 'b' && str[1] == 'b') {
      printf("String accepted\n");
   } else {
      printf("String rejected\n");
   }
```

```
    return 0;
}
```

3. DFA to accept strings that start with aba over input alphabets

```
#include <stdio.h>
#include <string.h>
int main() {
    int i;
    char str[100];
    printf("DFA to accept strings that start with aba over input alphabets (a,b)\n");
    printf("Enter string: ");
     scanf("%99s", str);
  // Check if the string is long enough
    if (strlen(str) >= 3 && str[0] == 'a' && str[1] == 'b' && str[2] == 'a') {
        printf("String accepted\n");
    } else {
        printf("String rejected\n");
    }
    return 0;
}
```