Assignment - 3.

write a c program to implement for the Predictive Parser (non Recursive Descentparser) for the given grammer.

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
# define int 1
# define mulop 2
# define addop 3
# define openpar 4
# define closepar 5
# define size 50

int token
int E();
char lexbuff [size];
int lookahead = 0.
int lexer() {
 if (lexbuff[lookahead] != '\0') {
    while (lexbuff[lookahead] == '\t' || lexbuff[lookahead] == ' ')
    lookahead++;
   if (isalpha(lexbuff [lookahead])) {
      while (isalnum(lexbuff [lookahead]))
           lookahead++;
      lookahead--;
      return id;
}.
```

```
else if (isdigit (lexbuff[lookahead])) {
   while (isdigit(lexbuff[lookahead]))
        lookahead++;
     lookahead--;
     return id;
  }
 else if (lexbuff[lookahead] == '+')
  return addop;
  else if (lexbuff[lookahead] == '*')
  return mulop;
  else if (lexbuff[lookahead] == '(')
  return openpar;
  else if (lexbuff[lookahead] == ')')
  return closepar;
   }
   return -1;
}
int f() {
     tokn = lexer();
      if(token==id){
         look ahead ++;
         return 1;
      }.
     else if (token == openpar){
       lookahead ++;
```

```
if (E()){
    token = lexer();
    if (token == closepar) {
        lookahead++;
        return 1;
      }
    else
        return0;
    }
    else
        return 0,
    }
    else
        return 0;
    }.
int TPRIME() {
    token = lexer();
    if ( token == mulop){
      lookahead++;
      if (F()) {
          if(TPRIME())
              return 1;
          else
              return 0;
```

Page No.

Date.

```
}
else
  return 0;
}
 else
   return 1;
 }
 int T(){
   if(F()){
     if(TPRIME())
         return 1;
     else
        return 0;
     }
      return 0; }
 int EPRIME(){
      token=lexer();
      if(token==addop){
          lookahead++;
       if(T()){
          if(EPRIME())
             return 1;
             return 0; }
             return 0; }
            return 1;
         }.
```

Teacher's Signature ....

```
int E() {
  if (T()){
    if(EPRIME())
        return 1; }
      return 0; }
    return0; }
int parser() {
    if(E()){
       if(lexbuff[lookahead]=='\0')
          printf("valid string");
       else
          printf("Invalid string");}
       else
          printf("Invalid string");
       getch();
       return 0;
       }.
int main() {
printf("\n Recursive descent parsing for the following grammar\n");
printf("\n E→ TE'\n E'→ +TE'|e\n T→ FT' \n T'→ xFT'|e\n F→
                                                   (E)|I0\n");
printf("\n Enter the string to be checked");
gets(lexBuff);
parser();
return 0;
}.
```

Output

```
Recursive descent parsing for the following grammar
 E → TE'
 E' → +TE'/@
 T → FT'
 T' → *F'/@
 F → (E)/ID
 Enter the valid string to be checked: A
 valid.
```

Scanned by TapScanner