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

// Functions to calculate Follow
void followfirst(char, int, int);
void follow(char currentsymbol);

// Function to calculate firstset
void findfirst(char, int, int);

int count, n = 0;

// Stores the final result of the firstset Sets
char firstsets[10][100];

// Stores the final result of the Follow Sets
char followsets[10][100];
int m = 0;

// Maximum number of productions
#define MAX_PRODUCTIONS 10
// Maximum length of each productionrules
#define MAX_PRODUCTION_LENGTH 10

// Stores the productionrules rules
char productionrules[MAX_PRODUCTIONS][MAX_PRODUCTION_LENGTH];
char followset[10], firstset[10];
int setindex;
char currentnonterminal;
int followindex;

int main(int argc, char **argv)
{
    int firstsetindex = 0;
    int followsetindex = 0;
    int productionindex, choice;
    char currentsymbol, currentchar;

    // Taking the number of productions from the user
    printf("Kindly enter the number of productions: ");
    scanf("%d", &count);

    // Taking productionrules equations from the user with -> instead of =
    for(int productionindex = 0; productionindex < count; productionindex++)
    {
        printf("Kindly enter production rule %d: ", productionindex + 1);
        scanf("%s", productionrules[productionindex]);
    }

    printf("The productions are as follows: \n");
    for(int productionindex = 0; productionindex < count; productionindex++)
    {
        printf("%s\n", productionrules[productionindex]);
    }

    int doneproductionindex;
    char done[count];
    int ptr = -1;

    // Initializing the firstsets array
    for (setindex = 0; setindex < count; setindex++)
    {
        for (doneproductionindex = 0; doneproductionindex < 100; doneproductionindex+
+)
        {
            firstsets[setindex][doneproductionindex] = '!';
        }
    }
    int point1 = 0, point2, flag;

    for (setindex = 0; setindex < count; setindex++)
    {
        currentsymbol = productionrules[setindex][0];
        point2 = 0;
        flag = 0;

        // Checking if firstset of currentsymbol has already been calculated
```

```
78          for (doneproductionindex = 0; doneproductionindex <= ptr;
    doneproductionindex++)
79              if (currentsymbol == done[doneproductionindex])
80                  flag = 1;
81
82          if (flag == 1)
83              continue;
84
85          // Function call
86          findfirst(currentsymbol, 0, 0);
87          ptr += 1;
88
89          // Adding currentsymbol to the calculated list
90          done[ptr] = currentsymbol;
91          printf("\n firstset(%c) = { ", currentsymbol);
92          firstsets[point1][point2++] = currentsymbol;
93
94          // Printing the firstset Sets of the grammar
95          for (productionindex = 0 + firstsetindex; productionindex < n;
    productionindex++)
96          {
97              int lark = 0, chk = 0;
98
99              for (lark = 0; lark < point2; lark++)
100             {
101
102                 if (firstset[productionindex] == firstsets[point1][lark])
103                 {
104                     chk = 1;
105                     break;
106                 }
107             }
108             if (chk == 0)
109             {
110                 printf("%c, ", firstset[productionindex]);
111                 firstsets[point1][point2++] = firstset[productionindex];
112             }
113         }
114         printf("}\n");
115         firstsetindex = n;
116         point1++;
117     }
118     printf("\n");
119     printf("-----------------------------------------------"
120            "\n\n");
121     char donee[count];
122     ptr = -1;
123
124     // Initializing the followsets array
125     for (setindex = 0; setindex < count; setindex++)
126     {
127         for (doneproductionindex = 0; doneproductionindex < 100; doneproductionindex+
    +)
128         {
129             followsets[setindex][doneproductionindex] = '!';
130         }
131     }
132     point1 = 0;
133     int land = 0;
134     for (followindex = 0; followindex < count; followindex++)
135     {
136         currentnonterminal = productionrules[followindex][0];
137         point2 = 0;
138         flag = 0;
139
140         // Checking if Follow of currentnonterminal
141         // has already been calculated
142         for (doneproductionindex = 0; doneproductionindex <= ptr;
    doneproductionindex++)
143             if (currentnonterminal == donee[doneproductionindex])
144                 flag = 1;
145
146         if (flag == 1)
147             continue;
148         land += 1;
149
150         // Function call
151         follow(currentnonterminal);
```

```
152           ptr += 1;
153
154           // Adding currentnonterminal to the calculated list
155           donee[ptr] = currentnonterminal;
156           printf(" Follow(%c) = { ", currentnonterminal);
157           followsets[point1][point2++] = currentnonterminal;
158
159           // Printing the Follow Sets of the grammar
160           for (productionindex = 0 + followsetindex; productionindex < m;
    productionindex++)
161           {
162               int lark = 0, chk = 0;
163               for (lark = 0; lark < point2; lark++)
164               {
165                   if (followset[productionindex] == followsets[point1][lark])
166                   {
167                       chk = 1;
168                       break;
169                   }
170               }
171               if (chk == 0)
172               {
173                   printf("%c, ", followset[productionindex]);
174                   followsets[point1][point2++] = followset[productionindex];
175               }
176           }
177           printf(" }\n\n");
178           followsetindex = m;
179           point1++;
180       }
181   }
182
183   void follow(char currentsymbol)
184   {
185       int productionindex, symbolindex;
186
187       // Adding "$" to the follow set of the start symbol
188       if (productionrules[0][0] == currentsymbol)
189       {
190           followset[m++] = '$';
191       }
192       for (productionindex = 0; productionindex < 10; productionindex++)
193       {
194           for (symbolindex = 2; symbolindex < 10; symbolindex++)
195           {
196               if (productionrules[productionindex][symbolindex] == currentsymbol)
197               {
198                   if (productionrules[productionindex][symbolindex + 1] != '\0')
199                   {
200                       // Calculate the firstset of the next Non-Terminal in the
    productionrules
201                       followfirst(productionrules[productionindex][symbolindex + 1],
    productionindex,
202                                   (symbolindex + 2));
203                   }
204
205                   if (productionrules[productionindex][symbolindex + 1] == '\0' &&
    currentsymbol != productionrules[productionindex][0])
206                   {
207                       // Calculate the follow of the Non-Terminal in the L.H.S. of the
    productionrules
208                       follow(productionrules[productionindex][0]);
209                   }
210               }
211           }
212       }
213   }
214
215   void findfirst(char currentsymbol, int q1, int q2)
216   {
217       int symbolindex;
218
219       // The case where we encounter a Terminal
220       if (!(isupper(currentsymbol)))
221       {
222           firstset[n++] = currentsymbol;
223       }
224       for (symbolindex = 0; symbolindex < count; symbolindex++)
```

```
    {
        if (productionrules[symbolindex][0] == currentsymbol)
        {
            if (productionrules[symbolindex][3] == '#')
            {
                if (productionrules[q1][q2] == '\0')
                    firstset[n++] = '#';
                else if (productionrules[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                {
                    // Recursion to calculate firstset of New Non-Terminal we
encounter after epsilon
                    findfirst(productionrules[q1][q2], q1,
                              (q2 + 1));
                }
                else
                    firstset[n++] = '#';
            }
            else if (!isupper(productionrules[symbolindex][3]))
            {
                firstset[n++] = productionrules[symbolindex][3];
            }
            else
            {
                // Recursion to calculate firstset of New Non-Terminal we encounter
at the beginning
                findfirst(productionrules[symbolindex][3], symbolindex, 4);
            }
        }
    }
}

void followfirst(char currentsymbol, int c1, int c2)
{
    int setindex;

    // The case where we encounter a Terminal
    if (!(isupper(currentsymbol)))
        followset[m++] = currentsymbol;
    else
    {
        int productionindex = 0, symbolindex = 1;
        for (productionindex = 0; productionindex < count; productionindex++)
        {
            if (firstsets[productionindex][0] == currentsymbol)
                break;
        }

        // Including the firstset set of the Non-Terminal in the Follow of the
original query
        while (firstsets[productionindex][symbolindex] != '!')
        {
            if (firstsets[productionindex][symbolindex] != '#')
            {
                followset[m++] = firstsets[productionindex][symbolindex];
            }
            else
            {
                if (productionrules[c1][c2] == '\0')
                {
                    // Case where we reach the end of a productionrules
                    follow(productionrules[c1][0]);
                }
                else
                {
                    // Recursion to the next symbol in case we encounter a "#"
                    followfirst(productionrules[c1][c2], c1,
                                c2 + 1);
                }
            }
            symbolindex++;
        }
    }
}
```