LL1 Parser:

|  |
| --- |
| **Input :**  E -> TR  R -> +T R| #  T -> F Y  Y -> \*F Y | #  F -> (E) | i  **Output :**  First(E)= { (, i, }  First(R)= { +, #, }  First(T)= { (, i, }  First(Y)= { \*, #, }  First(F)= { (, i, }  -----------------------------------------------  Follow(E) = { $, ), }  Follow(R) = { $, ), }  Follow(T) = { +, $, ), }  Follow(Y) = { +, $, ), }  Follow(F) = { \*, +, $, ), }  The functions follow and followfirst are both involved in the calculation of the Follow Set of a given Non-Terminal. The follow set of the start symbol will always contain “$”. Now the calculation of Follow falls under three broad cases :   * If a Non-Terminal on the R.H.S. of any production is followed immediately by a Terminal then it can immediately be included in the Follow set of that Non-Terminal. * If a Non-Terminal on the R.H.S. of any production is followed immediately by a Non-Terminal, then the First Set of that new Non-Terminal gets included on the follow set of our original Non-Terminal. In case encountered an epsilon i.e. ” # ” then, move on to the next symbol in the production. **Note :** “#” is never included in the Follow set of any Non-Terminal. * If reached the end of a production while calculating follow, then the Follow set of that non-teminal will include the Follow set of the Non-Terminal on the L.H.S. of that production. This can easily be implemented by recursion.   **Explanation :** Store the grammar on a 2D character array **production**. **findfirst** function is for calculating the first of any non terminal. Calculation of **first** falls under two broad cases :   * If the first symbol in the R.H.S of the production is a Terminal then it can directly be included in the first set. * If the first symbol in the R.H.S of the production is a Non-Terminal then call the findfirst function again on that Non-Terminal. To handle these cases like Recursion is the best possible solution. Here again, if the First of the new Non-Terminal contains an epsilon then we have to move to the next symbol of the original production which can again be a Terminal or a Non-Terminal.   **Note :**For the second case it is very easy to fall prey to an INFINITE LOOP even if the code looks perfect. So it is important to keep track of all the function calls at all times and never call the same function again.  Program:  #include<stdio.h>  #include<ctype.h>  #include<string.h>    // Functions to calculate Follow  void followfirst(char, int, int);  void follow(char c);    // Function to calculate First  void findfirst(char, int, int);    int count, n = 0;    // Stores the final result  // of the First Sets  char calc\_first[10][100];    // Stores the final result  // of the Follow Sets  char calc\_follow[10][100];  int m = 0;    // Stores the production rules  char production[10][10];  char f[10], first[10];  int k;  char ck;  int e;    int main(int argc, char \*\*argv)  {      int jm = 0;      int km = 0;      int i, choice;      char c, ch;      count = 8;        // The Input grammar      strcpy(production[0], "E=TR");      strcpy(production[1], "R=+TR");      strcpy(production[2], "R=#");      strcpy(production[3], "T=FY");      strcpy(production[4], "Y=\*FY");      strcpy(production[5], "Y=#");      strcpy(production[6], "F=(E)");      strcpy(production[7], "F=i");        int kay;      char done[count];      int ptr = -1;        // Initializing the calc\_first array      for(k = 0; k < count; k++) {          for(kay = 0; kay < 100; kay++) {              calc\_first[k][kay] = '!';          }      }      int point1 = 0, point2, xxx;        for(k = 0; k < count; k++)      {          c = production[k][0];          point2 = 0;          xxx = 0;            // Checking if First of c has          // already been calculated          for(kay = 0; kay <= ptr; kay++)              if(c == done[kay])                  xxx = 1;            if (xxx == 1)              continue;            // Function call          findfirst(c, 0, 0);          ptr += 1;            // Adding c to the calculated list          done[ptr] = c;          printf("\n First(%c) = { ", c);          calc\_first[point1][point2++] = c;            // Printing the First Sets of the grammar          for(i = 0 + jm; i < n; i++) {              int lark = 0, chk = 0;                for(lark = 0; lark < point2; lark++) {                    if (first[i] == calc\_first[point1][lark])                  {                      chk = 1;                      break;                  }              }              if(chk == 0)              {                  printf("%c, ", first[i]);                  calc\_first[point1][point2++] = first[i];              }          }          printf("}\n");          jm = n;          point1++;      }      printf("\n");      printf("-----------------------------------------------\n\n");      char donee[count];      ptr = -1;        // Initializing the calc\_follow array      for(k = 0; k < count; k++) {          for(kay = 0; kay < 100; kay++) {              calc\_follow[k][kay] = '!';          }      }      point1 = 0;      int land = 0;      for(e = 0; e < count; e++)      {          ck = production[e][0];          point2 = 0;          xxx = 0;            // Checking if Follow of ck          // has alredy been calculated          for(kay = 0; kay <= ptr; kay++)              if(ck == donee[kay])                  xxx = 1;            if (xxx == 1)              continue;          land += 1;            // Function call          follow(ck);          ptr += 1;            // Adding ck to the calculated list          donee[ptr] = ck;          printf(" Follow(%c) = { ", ck);          calc\_follow[point1][point2++] = ck;            // Printing the Follow Sets of the grammar          for(i = 0 + km; i < m; i++) {              int lark = 0, chk = 0;              for(lark = 0; lark < point2; lark++)              {                  if (f[i] == calc\_follow[point1][lark])                  {                      chk = 1;                      break;                  }              }              if(chk == 0)              {                  printf("%c, ", f[i]);                  calc\_follow[point1][point2++] = f[i];              }          }          printf(" }\n\n");          km = m;          point1++;      }  }    void follow(char c)  {      int i, j;        // Adding "$" to the follow      // set of the start symbol      if(production[0][0] == c) {          f[m++] = '$';      }      for(i = 0; i < 10; i++)      {          for(j = 2;j < 10; j++)          {              if(production[i][j] == c)              {                  if(production[i][j+1] != '\0')                  {                      // Calculate the first of the next                      // Non-Terminal in the production                      followfirst(production[i][j+1], i, (j+2));                  }                    if(production[i][j+1]=='\0' && c!=production[i][0])                  {                      // Calculate the follow of the Non-Terminal                      // in the L.H.S. of the production                      follow(production[i][0]);                  }              }          }      }  }    void findfirst(char c, int q1, int q2)  {      int j;        // The case where we      // encounter a Terminal      if(!(isupper(c))) {          first[n++] = c;      }      for(j = 0; j < count; j++)      {          if(production[j][0] == c)          {              if(production[j][2] == '#')              {                  if(production[q1][q2] == '\0')                      first[n++] = '#';                  else if(production[q1][q2] != '\0'                            && (q1 != 0 || q2 != 0))                  {                      // Recursion to calculate First of New                      // Non-Terminal we encounter after epsilon                      findfirst(production[q1][q2], q1, (q2+1));                  }                  else                      first[n++] = '#';              }              else if(!isupper(production[j][2]))              {                  first[n++] = production[j][2];              }              else              {                  // Recursion to calculate First of                  // New Non-Terminal we encounter                  // at the beginning                  findfirst(production[j][2], j, 3);              }          }      }  }    void followfirst(char c, int c1, int c2)  {      int k;        // The case where we encounter      // a Terminal      if(!(isupper(c)))          f[m++] = c;      else      {          int i = 0, j = 1;          for(i = 0; i < count; i++)          {              if(calc\_first[i][0] == c)                  break;          }            //Including the First set of the          // Non-Terminal in the Follow of          // the original query          while(calc\_first[i][j] != '!')          {              if(calc\_first[i][j] != '#')              {                  f[m++] = calc\_first[i][j];              }              else              {                  if(production[c1][c2] == '\0')                  {                      // Case where we reach the                      // end of a production                      follow(production[c1][0]);                  }                  else                  {                      // Recursion to the next symbol                      // in case we encounter a "#"                      followfirst(production[c1][c2], c1, c2+1);                  }              }              j++;          }      }  } |

**Input :**

E -> TR

R -> +T R| #

T -> F Y

Y -> \*F Y | #

F -> (E) | i

Output :

First(E)= { (, i, }

First(R)= { +, #, }

First(T)= { (, i, }

First(Y)= { \*, #, }

First(F)= { (, i, }

-----------------------------------------------

Follow(E) = { $, ), }

Follow(R) = { $, ), }

Follow(T) = { +, $, ), }

Follow(Y) = { +, $, ), }

Follow(F) = { \*, +, $, ), }