Computing First and Follow of the given Grammar
Compiler Design – Lab 5

Name: Ojas Patil
Reg No: 21BAI1106

AIM

Write a C / C++ / Java program to Compute First and Follow for the given productions and then construct a prediction parser table using the computed First and Follow values.

Algorithm

1. Include Libraries: Begin by including necessary libraries such as <stdio.h> and <string.h>.
2. Define Constants: Define constant values like MAX_SIZE and initialize arrays such as table, terminal, and nonterminal.
3. Define Structures: Define a structure product to hold production rules.
4. Read Productions from File: Implement a function readFromFile() to read production rules from a file named "text.txt".
5. Compute FIRST sets: Implement functions FIRST() and then Follow A->B function to compute the FIRST sets for each non-terminal symbol.
6. Compute FOLLOW sets: Implement functions FOLLOW(), First_A_Follow_B(), and Follow_A_Follow_B() to compute the Follow sets for each non-terminal symbol.
7. Compute First sets for Right Hand Side: Implement functions First_RHS() and First_A_to_First_B() to compute the First sets for the right-hand side of production rules.
8. Main Function: Call the above functions in correct order and then print the First and Follow of the mentioned states, followed by the predictive parser of the grammar.

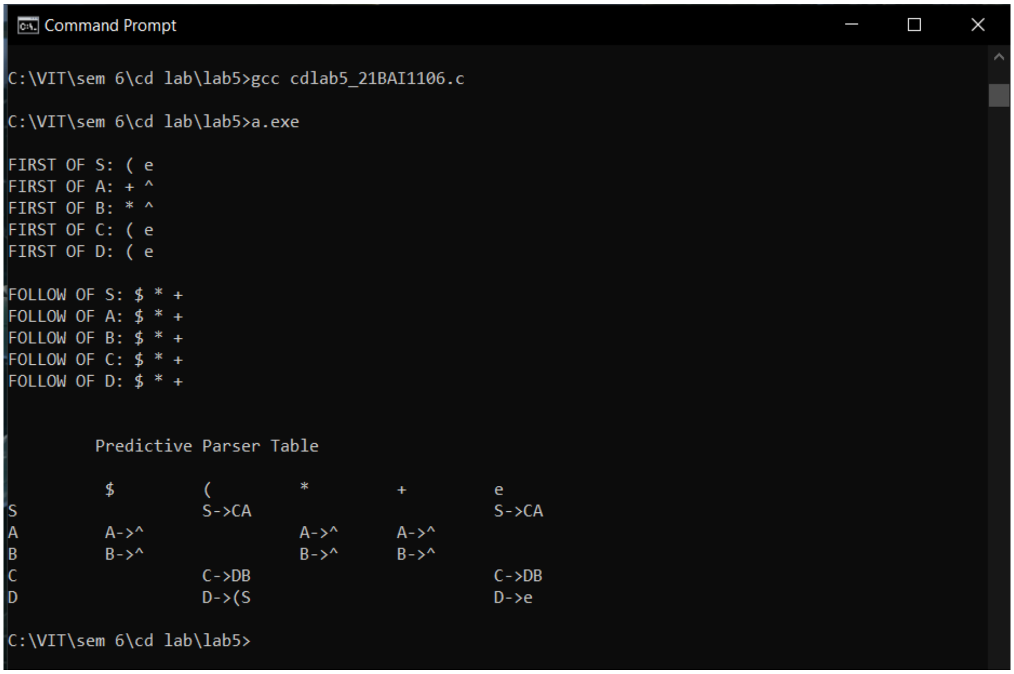
Explanation

The code computes First and Follow sets for each non-terminal symbol in a context-free grammar and generates a predictive parser table based on these sets. It first reads production rules from a file, then iteratively computes First sets for each non-terminal symbol. Follow sets are then computed using the First sets. Additionally, First sets for the right-hand side of each production rule are calculated. Finally, the predictive parser table is populated using the computed sets to facilitate parsing of input strings according to the grammar.

Sample Input

```
S->CA
A->+CA|^
B->*DB|^
C->DB
D->e|(S)
```

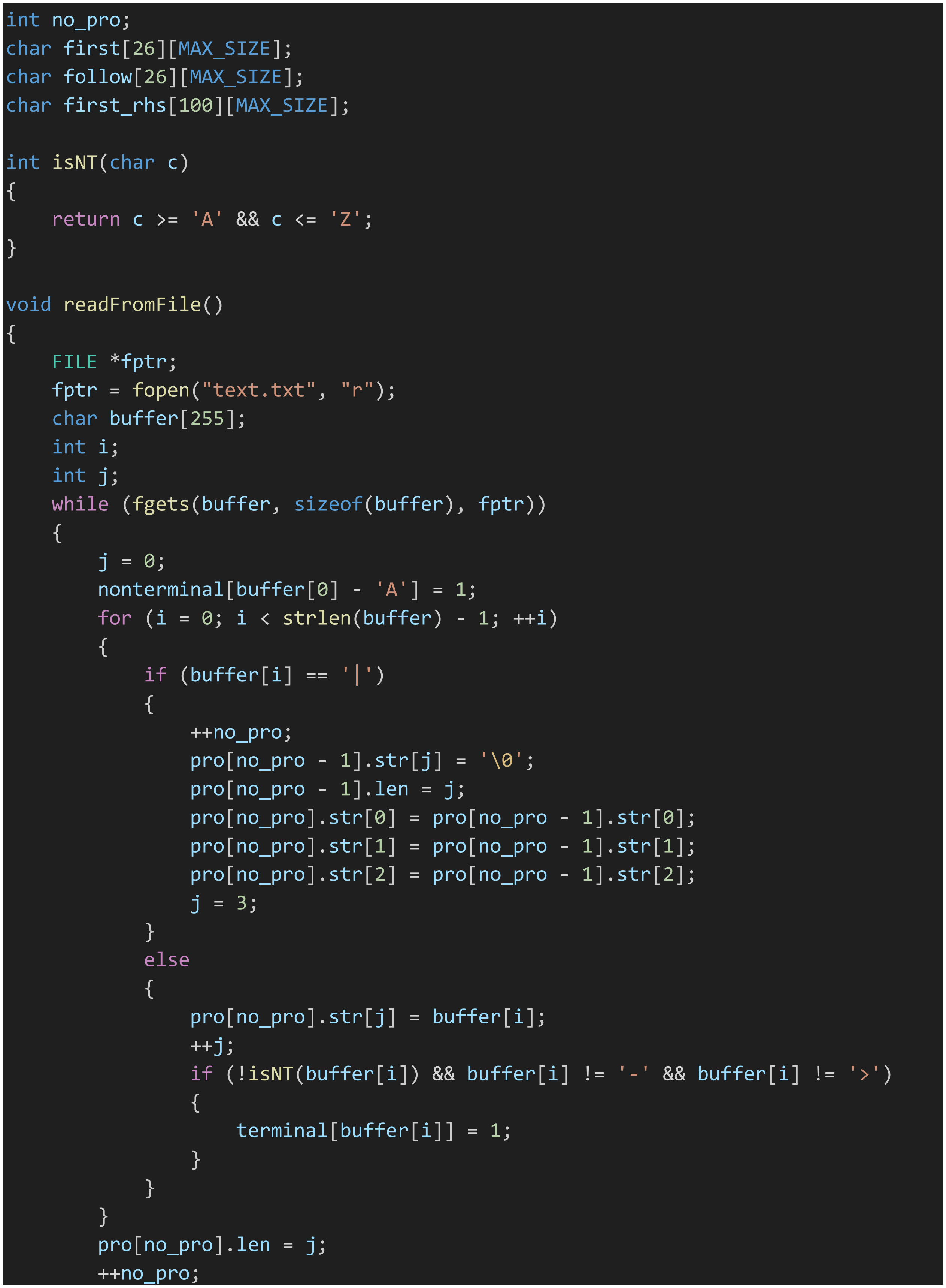
Output Screenshot



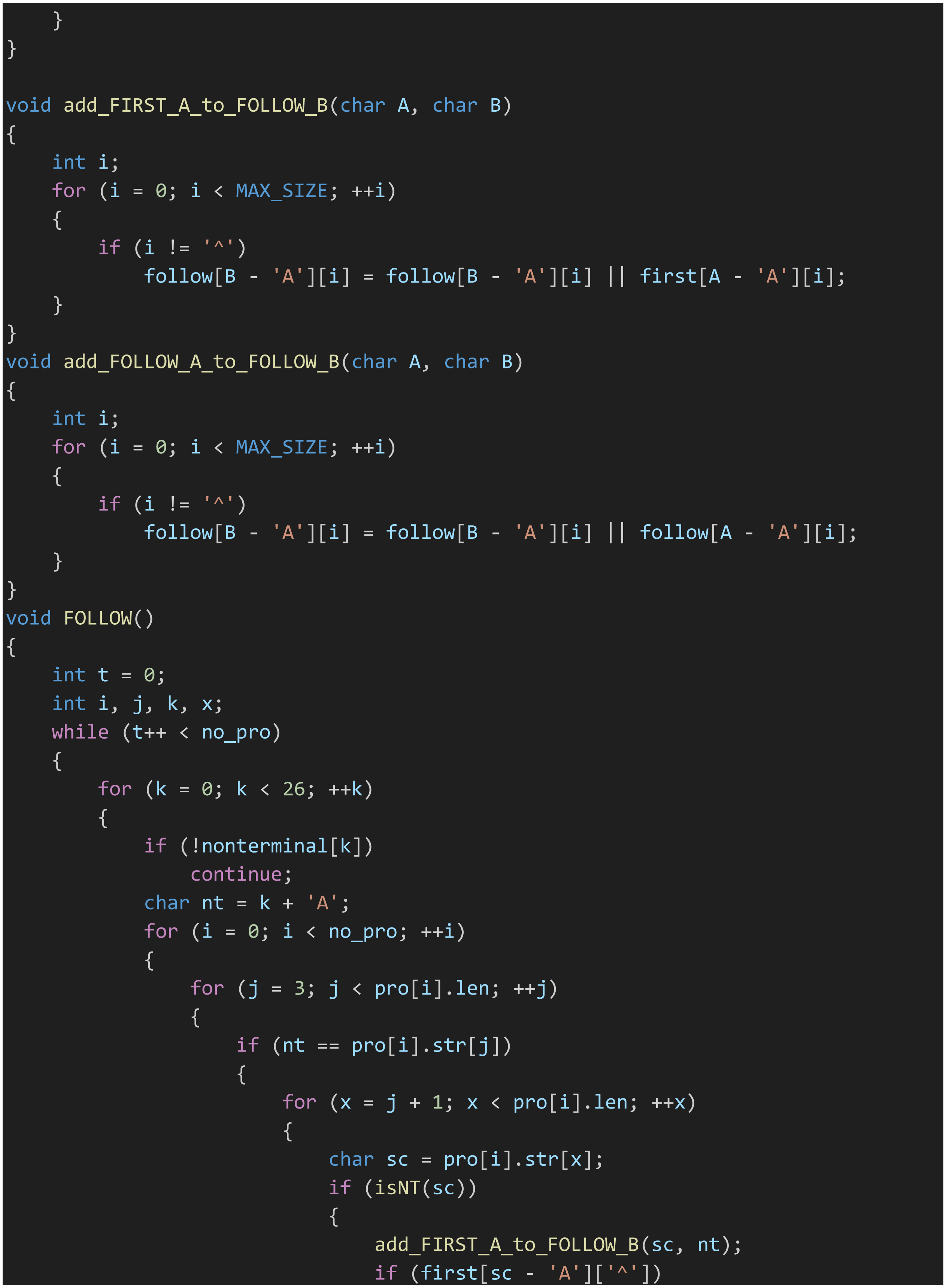
```
C:\VIT\sem 6\cd lab\lab5>gcc cdlab5_21BAI1106.c

C:\VIT\sem 6\cd lab\lab5>a.exe

FIRST OF S: ( e
FIRST OF A: + ^
FIRST OF B: * ^
FIRST OF C: ( e
FIRST OF D: ( e

FOLLOW OF S: $ * +
FOLLOW OF A: $ * +
FOLLOW OF B: $ * +
FOLLOW OF C: $ * +
FOLLOW OF D: $ * +


         Predictive Parser Table

          $         (         *         +         e
S                   S->CA                         S->CA
A         A->^                A->^      A->^
B         B->^                B->^      B->^
C                   C->DB                         C->DB
D                   D->(S                         D->e

C:\VIT\sem 6\cd lab\lab5>
```

Source Code

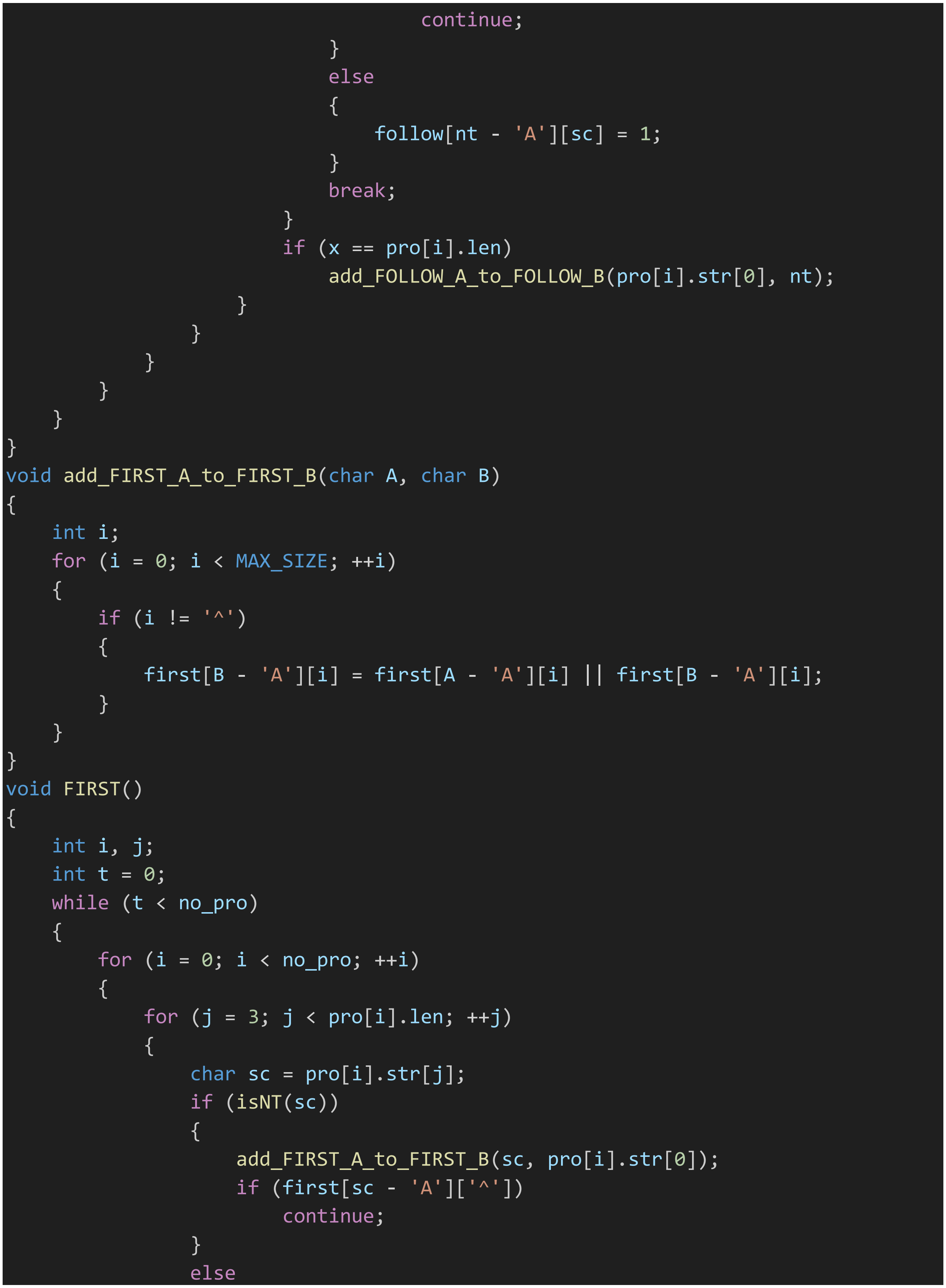
```c
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 128

int table[100][MAX_SIZE];
char terminal[MAX_SIZE];
char nonterminal[26];

struct product
{
    char str[100];
    int len;
} pro[20];
```



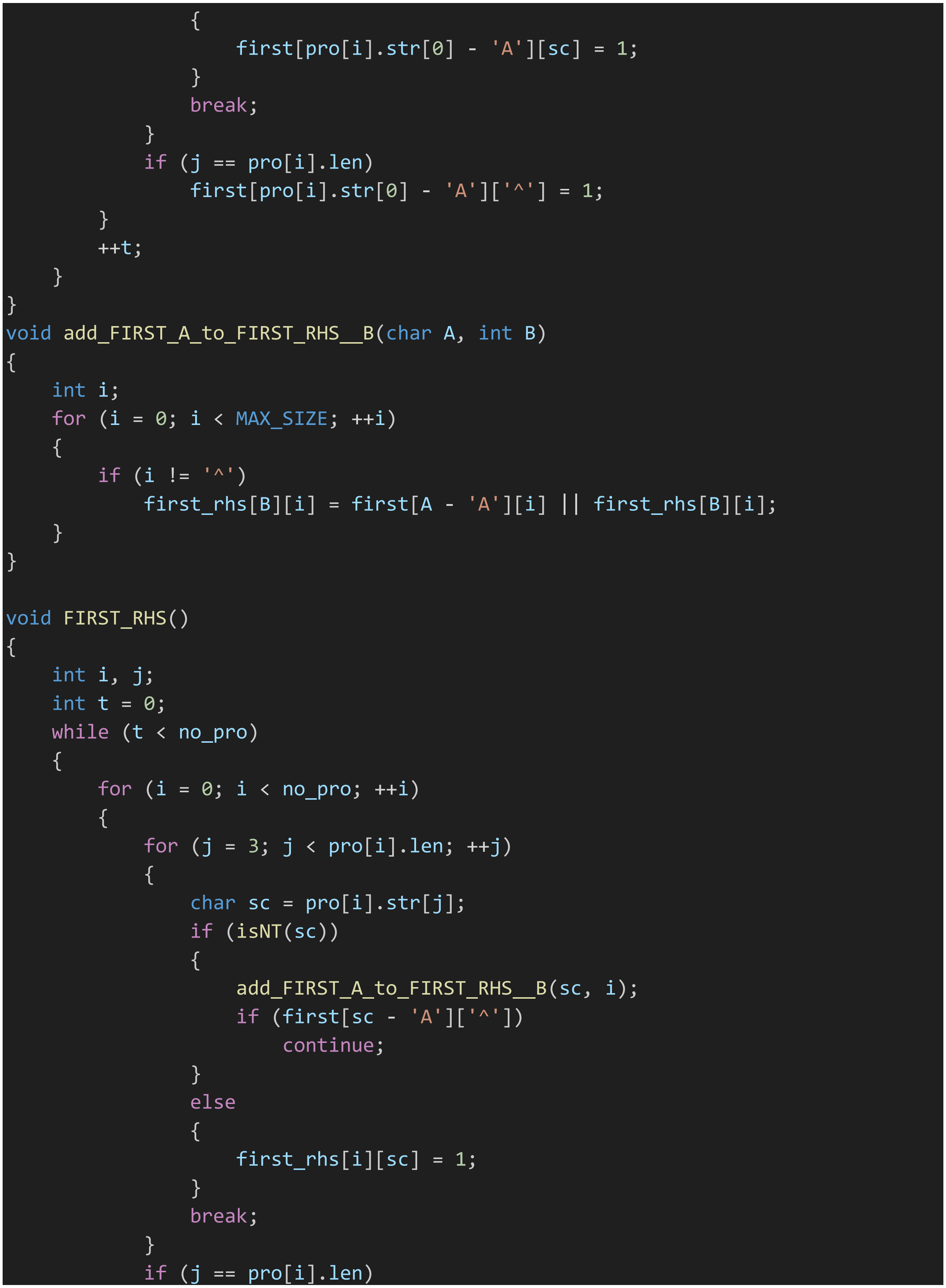
```
int no_pro;
char first[26][MAX_SIZE];
char follow[26][MAX_SIZE];
char first_rhs[100][MAX_SIZE];

int isNT(char c)
{
    return c >= 'A' && c <= 'Z';
}

void readFromFile()
{
    FILE *fptr;
    fptr = fopen("text.txt", "r");
    char buffer[255];
    int i;
    int j;
    while (fgets(buffer, sizeof(buffer), fptr))
    {
        j = 0;
        nonterminal[buffer[0] - 'A'] = 1;
        for (i = 0; i < strlen(buffer) - 1; ++i)
        {
            if (buffer[i] == '|')
            {
                ++no_pro;
                pro[no_pro - 1].str[j] = '\0';
                pro[no_pro - 1].len = j;
                pro[no_pro].str[0] = pro[no_pro - 1].str[0];
                pro[no_pro].str[1] = pro[no_pro - 1].str[1];
                pro[no_pro].str[2] = pro[no_pro - 1].str[2];
                j = 3;
            }
            else
            {
                pro[no_pro].str[j] = buffer[i];
                ++j;
                if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>')
                {
                    terminal[buffer[i]] = 1;
                }
            }
        }
        pro[no_pro].len = j;
        ++no_pro;
```



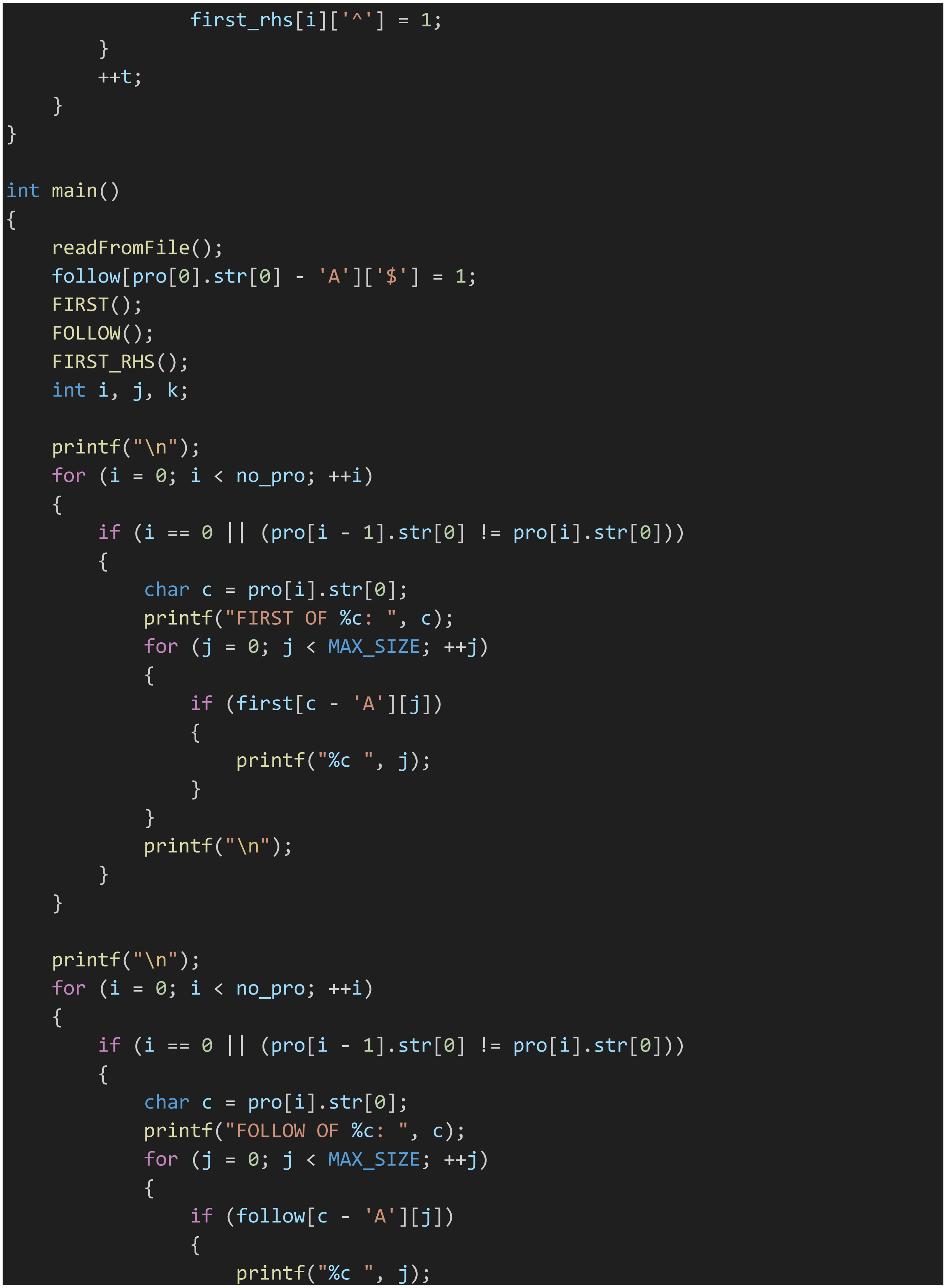
```
    }
}

void add_FIRST_A_to_FOLLOW_B(char A, char B)
{
    int i;
    for (i = 0; i < MAX_SIZE; ++i)
    {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];
    }
}
void add_FOLLOW_A_to_FOLLOW_B(char A, char B)
{
    int i;
    for (i = 0; i < MAX_SIZE; ++i)
    {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];
    }
}
void FOLLOW()
{
    int t = 0;
    int i, j, k, x;
    while (t++ < no_pro)
    {
        for (k = 0; k < 26; ++k)
        {
            if (!nonterminal[k])
                continue;
            char nt = k + 'A';
            for (i = 0; i < no_pro; ++i)
            {
                for (j = 3; j < pro[i].len; ++j)
                {
                    if (nt == pro[i].str[j])
                    {
                        for (x = j + 1; x < pro[i].len; ++x)
                        {
                            char sc = pro[i].str[x];
                            if (isNT(sc))
                            {
                                add_FIRST_A_to_FOLLOW_B(sc, nt);
                                if (first[sc - 'A']['^'])
```



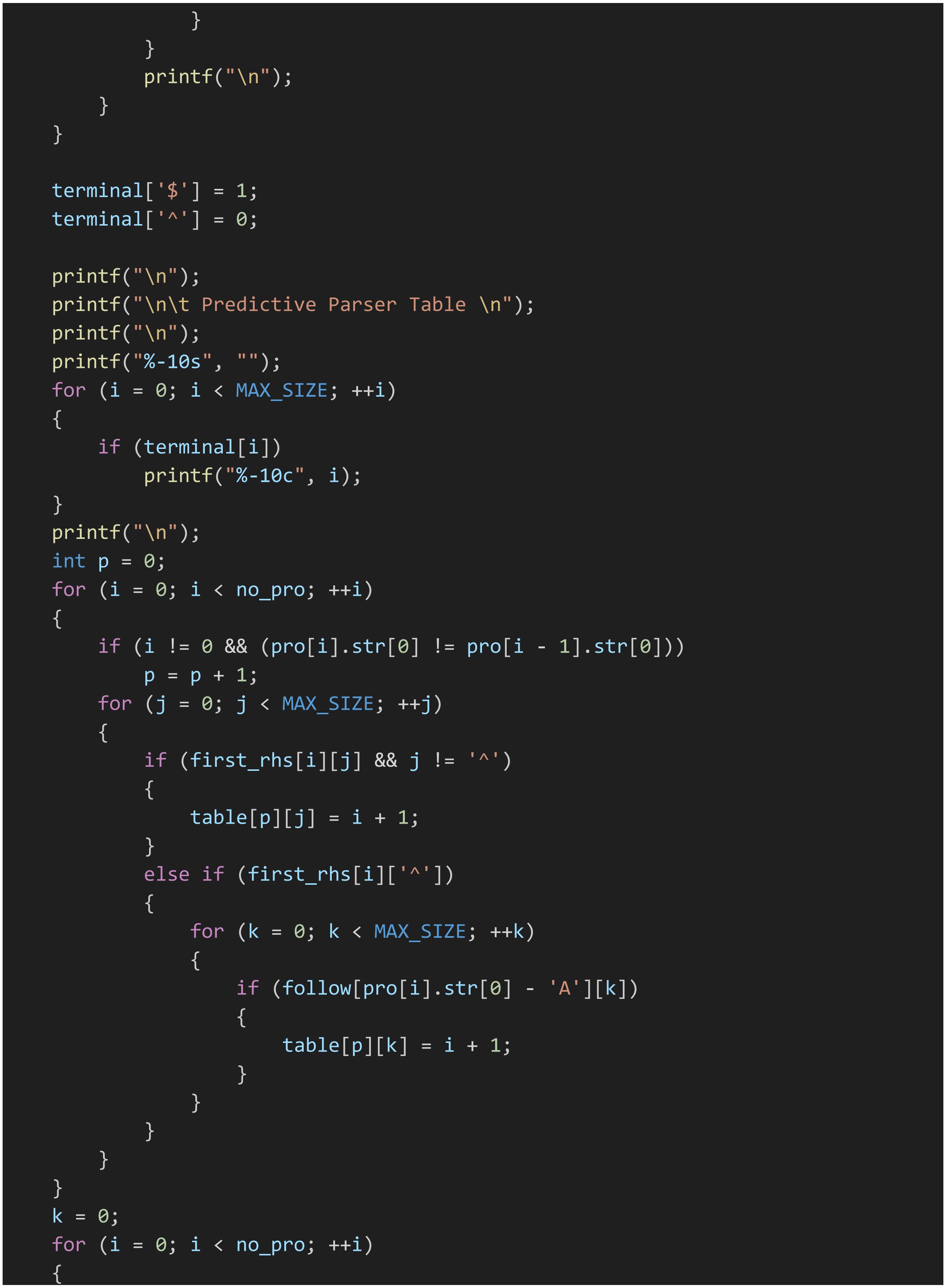
```
                                    continue;
                                }
                                else
                                {
                                    follow[nt - 'A'][sc] = 1;
                                }
                                break;
                            }
                            if (x == pro[i].len)
                                add_FOLLOW_A_to_FOLLOW_B(pro[i].str[0], nt);
                        }
                    }
                }
            }
        }
    }
}
void add_FIRST_A_to_FIRST_B(char A, char B)
{
    int i;
    for (i = 0; i < MAX_SIZE; ++i)
    {
        if (i != '^')
        {
            first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];
        }
    }
}
void FIRST()
{
    int i, j;
    int t = 0;
    while (t < no_pro)
    {
        for (i = 0; i < no_pro; ++i)
        {
            for (j = 3; j < pro[i].len; ++j)
            {
                char sc = pro[i].str[j];
                if (isNT(sc))
                {
                    add_FIRST_A_to_FIRST_B(sc, pro[i].str[0]);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else
```



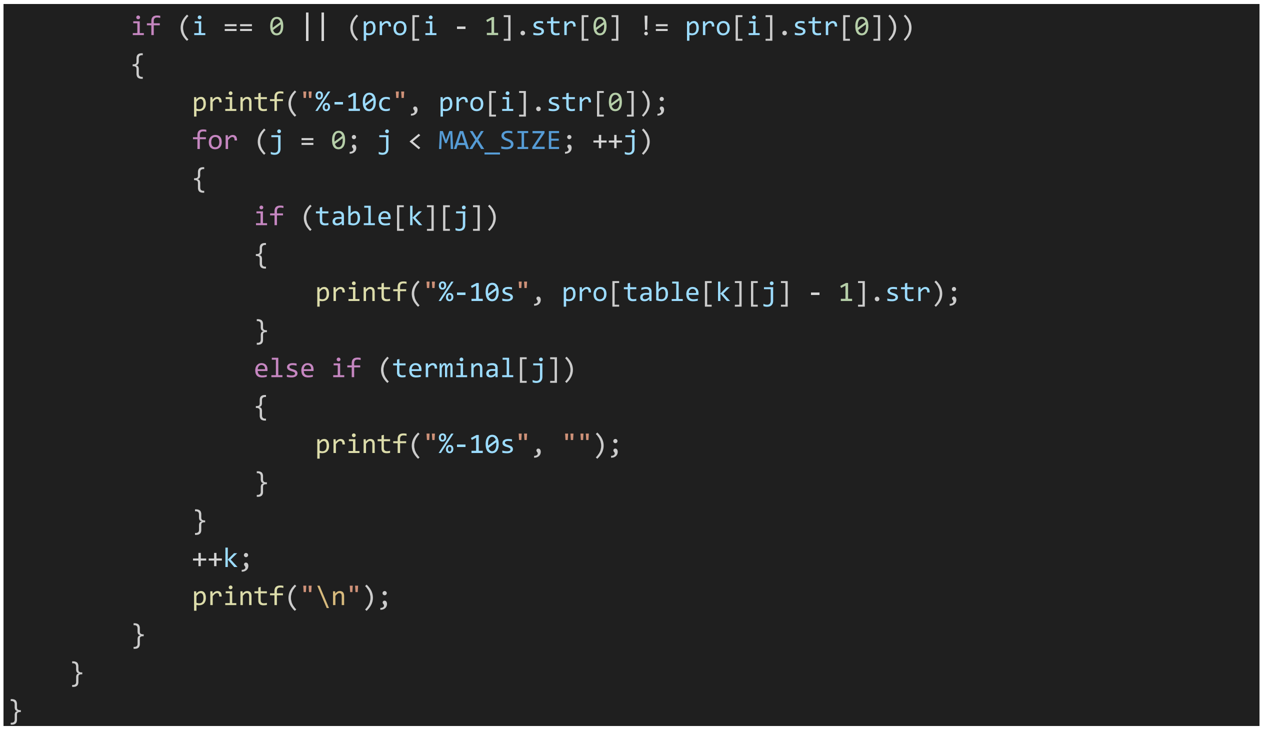
```
                {
                    first[pro[i].str[0] - 'A'][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first[pro[i].str[0] - 'A']['^'] = 1;
        }
        ++t;
    }
}
void add_FIRST_A_to_FIRST_RHS__B(char A, int B)
{
    int i;
    for (i = 0; i < MAX_SIZE; ++i)
    {
        if (i != '^')
            first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
    }
}

void FIRST_RHS()
{
    int i, j;
    int t = 0;
    while (t < no_pro)
    {
        for (i = 0; i < no_pro; ++i)
        {
            for (j = 3; j < pro[i].len; ++j)
            {
                char sc = pro[i].str[j];
                if (isNT(sc))
                {
                    add_FIRST_A_to_FIRST_RHS__B(sc, i);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else
                {
                    first_rhs[i][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
```



```
                first_rhs[i]['^'] = 1;
        }
        ++t;
    }
}

int main()
{
    readFromFile();
    follow[pro[0].str[0] - 'A']['$'] = 1;
    FIRST();
    FOLLOW();
    FIRST_RHS();
    int i, j, k;

    printf("\n");
    for (i = 0; i < no_pro; ++i)
    {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0]))
        {
            char c = pro[i].str[0];
            printf("FIRST OF %c: ", c);
            for (j = 0; j < MAX_SIZE; ++j)
            {
                if (first[c - 'A'][j])
                {
                    printf("%c ", j);
                }
            }
            printf("\n");
        }
    }

    printf("\n");
    for (i = 0; i < no_pro; ++i)
    {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0]))
        {
            char c = pro[i].str[0];
            printf("FOLLOW OF %c: ", c);
            for (j = 0; j < MAX_SIZE; ++j)
            {
                if (follow[c - 'A'][j])
                {
                    printf("%c ", j);
```



```
                }
            }
            printf("\n");
        }
    }

    terminal['$'] = 1;
    terminal['^'] = 0;

    printf("\n");
    printf("\n\t Predictive Parser Table \n");
    printf("\n");
    printf("%-10s", "");
    for (i = 0; i < MAX_SIZE; ++i)
    {
        if (terminal[i])
            printf("%-10c", i);
    }
    printf("\n");
    int p = 0;
    for (i = 0; i < no_pro; ++i)
    {
        if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))
            p = p + 1;
        for (j = 0; j < MAX_SIZE; ++j)
        {
            if (first_rhs[i][j] && j != '^')
            {
                table[p][j] = i + 1;
            }
            else if (first_rhs[i]['^'])
            {
                for (k = 0; k < MAX_SIZE; ++k)
                {
                    if (follow[pro[i].str[0] - 'A'][k])
                    {
                        table[p][k] = i + 1;
                    }
                }
            }
        }
    }
    k = 0;
    for (i = 0; i < no_pro; ++i)
    {
```



```
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0]))
        {
            printf("%-10c", pro[i].str[0]);
            for (j = 0; j < MAX_SIZE; ++j)
            {
                if (table[k][j])
                {
                    printf("%-10s", pro[table[k][j] - 1].str);
                }
                else if (terminal[j])
                {
                    printf("%-10s", "");
                }
            }
            ++k;
            printf("\n");
        }
    }
}
```

Result

Thus, we have written an C code to take input grammar from a text file, computed the First and Follow of each state, and printed it along with the Predictive Parser Table and verified the output.