main.c

Share Run

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

#define MAX_PROD 10
#define MAX_SYMBOLS 10

char productions[MAX_PROD][MAX_SYMBOLS];
int num_productions;

// Function to check if a symbol is a terminal
bool isTerminal(char symbol) {
    return !(symbol >= 'A' && symbol <= 'Z');
}

// Function to compute the LEADING set for a non-terminal
void computeLeading(char non_terminal, char leading_set[], int *leading_count) {
    for (int i = 0; i < num_productions; i++) {
        if (productions[i][0] == non_terminal) {
            char first_symbol = productions[i][3]; // First symbol after "->"
            if (isTerminal(first_symbol)) {
                // If the first symbol is a terminal, add it to the LEADING set
                leading_set[(*leading_count)++] = first_symbol;
            } else {
                // If the first symbol is a non-terminal, recursively compute its LEADING set
                computeLeading(first_symbol, leading_set, leading_count);
            }
        }
    }
}

int main() {
    // Example grammar
    strcpy(productions[0], "E->E+T");
    strcpy(productions[1], "E->T");
    strcpy(productions[2], "T->T*F");
    strcpy(productions[3], "T->F");
```

Output

```
Segmentation fault


=== Code Exited With Errors ===
```

```
strcpy(productions[5], "F->id");
num_productions = 6;

// Compute LEADING set for non-terminal E
char leading_set_E[MAX_SYMBOLS];
int leading_count_E = 0;
computeLeading('E', leading_set_E, &leading_count_E);

// Print the LEADING set for E
printf("LEADING(E) = { ");
for (int i = 0; i < leading_count_E; i++) {
    printf("%c ", leading_set_E[i]);
}
printf("}\n");

// Compute LEADING set for non-terminal T
char leading_set_T[MAX_SYMBOLS];
int leading_count_T = 0;
computeLeading('T', leading_set_T, &leading_count_T);

// Print the LEADING set for T
printf("LEADING(T) = { ");
for (int i = 0; i < leading_count_T; i++) {
    printf("%c ", leading_set_T[i]);
}
printf("}\n");

// Compute LEADING set for non-terminal F
char leading_set_F[MAX_SYMBOLS];
int leading_count_F = 0;
computeLeading('F', leading_set_F, &leading_count_F);

// Print the LEADING set for F
printf("LEADING(F) = { ");
```

```
Segmentation fault


=== Code Exited With Errors ===
```

main.c

Share Run

```
    int leading_count_E = 0;
    computeLeading('E', leading_set_E, &leading_count_E);

    // Print the LEADING set for E
    printf("LEADING(E) = { ");
    for (int i = 0; i < leading_count_E; i++) {
        printf("%c ", leading_set_E[i]);
    }
    printf("}\n");

    // Compute LEADING set for non-terminal T
    char leading_set_T[MAX_SYMBOLS];
    int leading_count_T = 0;
    computeLeading('T', leading_set_T, &leading_count_T);

    // Print the LEADING set for T
    printf("LEADING(T) = { ");
    for (int i = 0; i < leading_count_T; i++) {
        printf("%c ", leading_set_T[i]);
    }
    printf("}\n");

    // Compute LEADING set for non-terminal F
    char leading_set_F[MAX_SYMBOLS];
    int leading_count_F = 0;
    computeLeading('F', leading_set_F, &leading_count_F);

    // Print the LEADING set for F
    printf("LEADING(F) = { ");
    for (int i = 0; i < leading_count_F; i++) {
        printf("%c ", leading_set_F[i]);
    }
    printf("}\n");

    return 0;
}
```

Output

```
Segmentation fault


=== Code Exited With Errors ===
```