Week 5 Assignment

Implement a Recursive Descent Parser for the Expression Grammar given below.

E → TE’

E’→ +TE’ | є

T → FT’

T’→ *FT’ | є

F → (E) | i

Identify the grammar is suffers with left recursion and ambiguity before constructing the RD parser

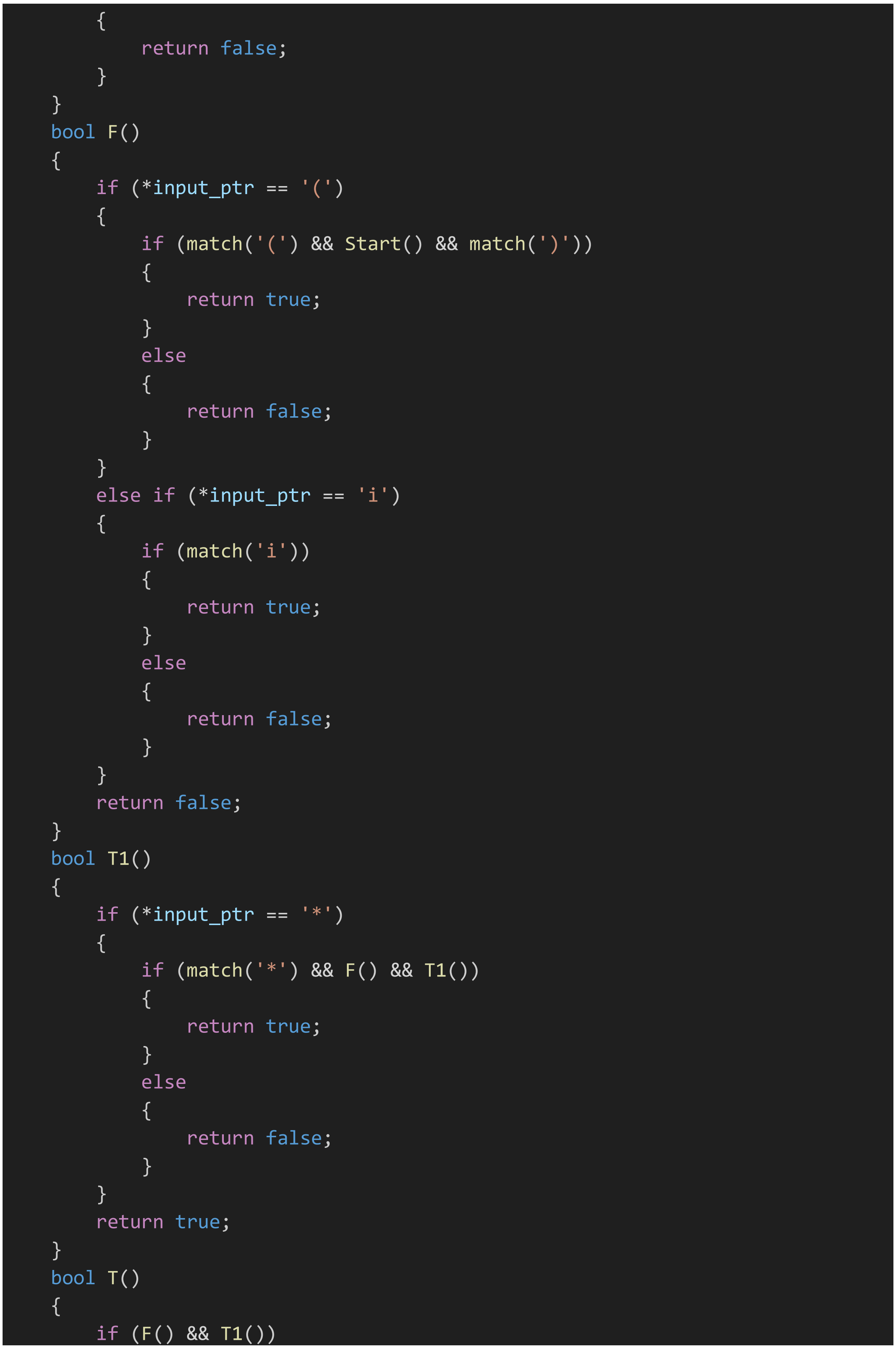
Solution:

The givne grammar doesn’t suffer with left recursion and ambiguity

Program:

```
#include <iostream>
#include <string.h>
using namespace std;
class Parser
{
private:
    string input;
    char *input_ptr;

public:
    Parser()
    {
    }
    Parser(string str)
    {
        input = str;
        input_ptr = &input[0];
    }
    bool match(char required)
    {
        if (*input_ptr == required)
        {
            input_ptr++;
            return true;
        }
        else
```

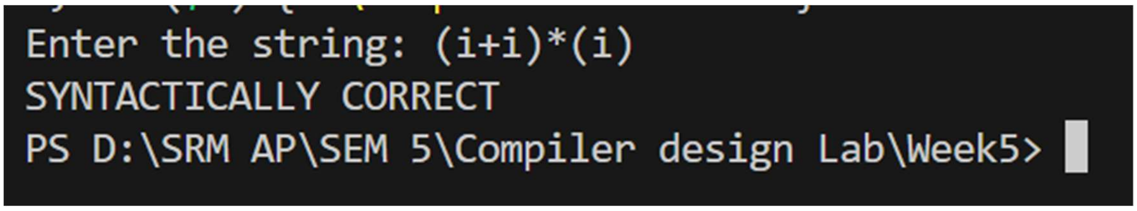


```
    {
        return false;
    }
}
bool F()
{
    if (*input_ptr == '(')
    {
        if (match('(') && Start() && match(')'))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else if (*input_ptr == 'i')
    {
        if (match('i'))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}
bool T1()
{
    if (*input_ptr == '*')
    {
        if (match('*') && F() && T1())
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return true;
}
bool T()
{
    if (F() && T1())
```

```cpp
        {
            return true;
        }
        return false;
    }
    bool E1()
    {
        if (*input_ptr == '+')
        {
            if (match('+') && T() && E1())
            {
                return true;
            }
            return false;
        }
        return true;
    }
    bool Start()
    {
        if (T() && E1())
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    char get_input_ptr()
    {
        return *input_ptr;
    }
};
int main()
{
    string input;
    cout << "Enter the string: ";
    cin >> input;
    Parser Rd_Parser(input);
    if (Rd_Parser.Start() && Rd_Parser.get_input_ptr() == '\0')
    {
        cout << "SYNTACTICALLY CORRECT" << endl;
    }
    else
    {
        cout << "SYNTACTICALLY INCORRECT";
    }
    return 1;
```

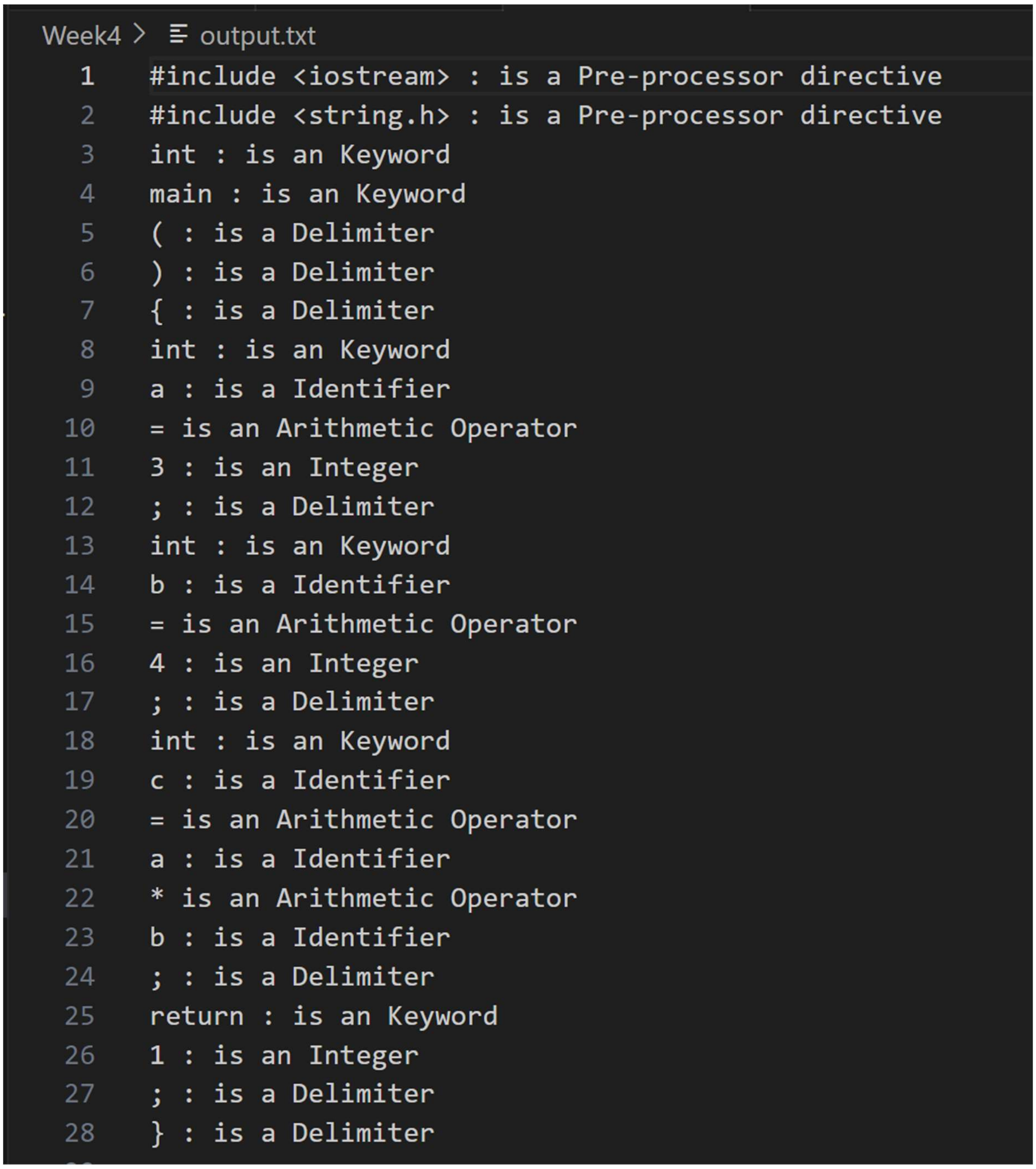
```
}
```

Output:



```
Enter the string: (i+i)*(i)
SYNTACTICALLY CORRECT
PS D:\SRM AP\SEM 5\Compiler design Lab\Week5>
```

Output file (.txt):



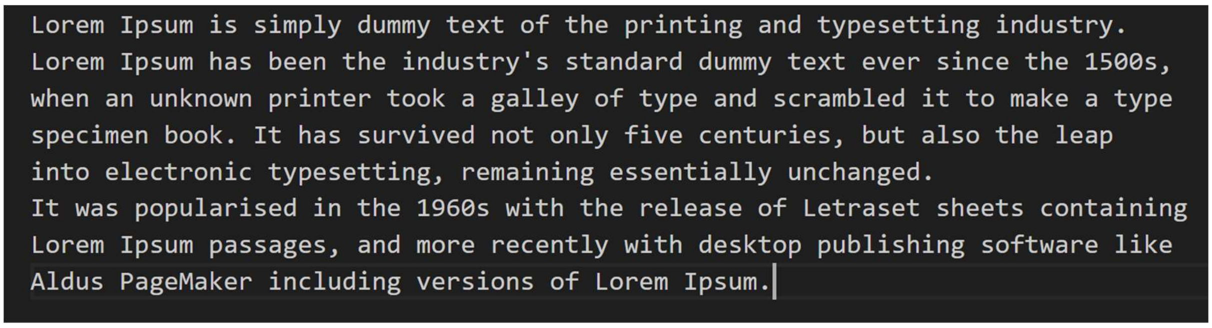
```
#include <iostream> : is a Pre-processor directive
#include <string.h> : is a Pre-processor directive
int : is an Keyword
main : is an Keyword
( : is a Delimiter
) : is a Delimiter
{ : is a Delimiter
int : is an Keyword
a : is a Identifier
= is an Arithmetic Operator
3 : is an Integer
; : is a Delimiter
int : is an Keyword
b : is a Identifier
= is an Arithmetic Operator
4 : is an Integer
; : is a Delimiter
int : is an Keyword
c : is a Identifier
= is an Arithmetic Operator
a : is a Identifier
* is an Arithmetic Operator
b : is a Identifier
; : is a Delimiter
return : is an Keyword
1 : is an Integer
; : is a Delimiter
} : is a Delimiter
```

2. Write a C Program to Scan and Count the number of characters, words, and lines in a file.

Program:

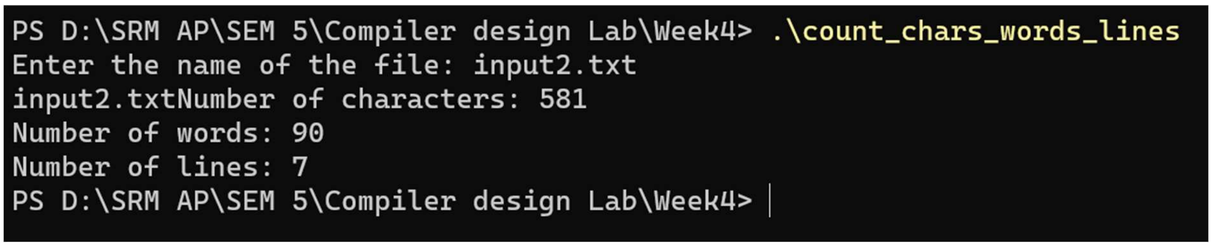
```
#include <stdio.h>
int main()
{
    char filename[100];
    printf("Enter the name of the file: ");
    scanf("%s", filename);
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Unable to open the file. Exiting...\n");
        return 1;
    }
    int no_of_chars = 0;
    int no_of_words = 0;
    int no_of_lines = 0;
    int in_word = 0;
    char ch;
    while ((ch = fgetc(file)) != EOF)
    {
        no_of_chars++;
        if (ch != ' ' && ch != '\t' && ch != '\n' && ch != '\r' && ch != '\f' && ch != '\v')
        {
            in_word = 1;
        }
        if ((ch == ' ' || ch == '\t' || ch == '\n' || ch == '\r' || ch == '\f' || ch == '\v') &&
            in_word)
        {
            no_of_words++;
            in_word = 0;
        }
        if (ch == '\n' || ch == '\0')
        {
            no_of_lines++;
        }
    }
    fclose(file);
    printf("Number of characters: %d\n", no_of_chars);
    printf("Number of words: %d\n", no_of_words);
    printf("Number of lines: %d\n", no_of_lines);
    return 0;
}
```

Input:



```
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type
specimen book. It has survived not only five centuries, but also the leap
into electronic typesetting, remaining essentially unchanged.
It was popularised in the 1960s with the release of Letraset sheets containing
Lorem Ipsum passages, and more recently with desktop publishing software like
Aldus PageMaker including versions of Lorem Ipsum.
```

Ouptut:



```
PS D:\SRM AP\SEM 5\Compiler design Lab\Week4> .\count_chars_words_lines
Enter the name of the file: input2.txt
input2.txtNumber of characters: 581
Number of words: 90
Number of lines: 7
PS D:\SRM AP\SEM 5\Compiler design Lab\Week4>
```