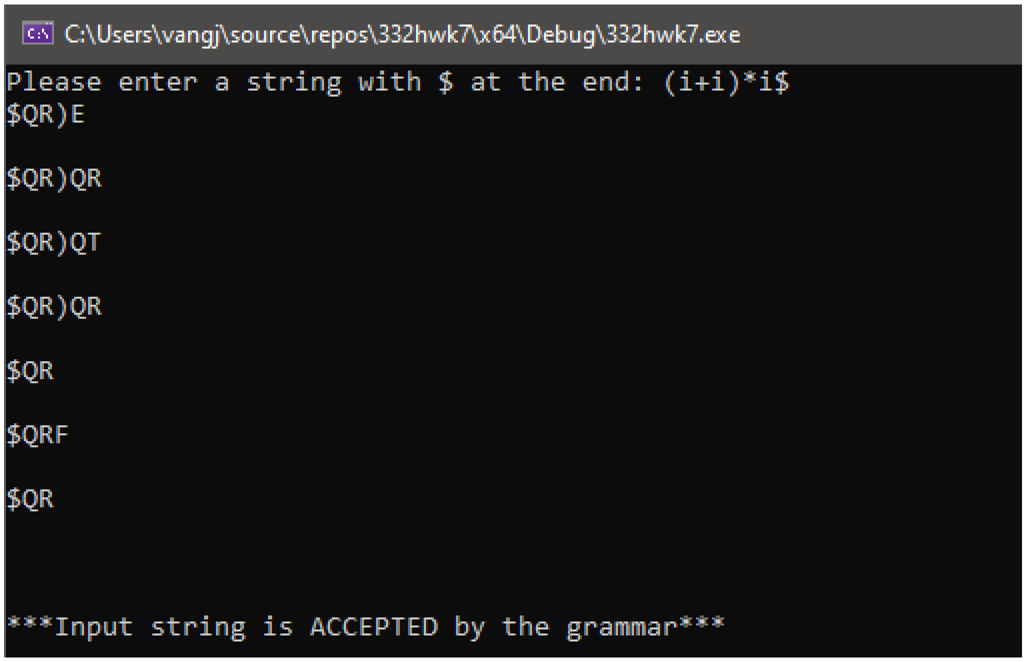
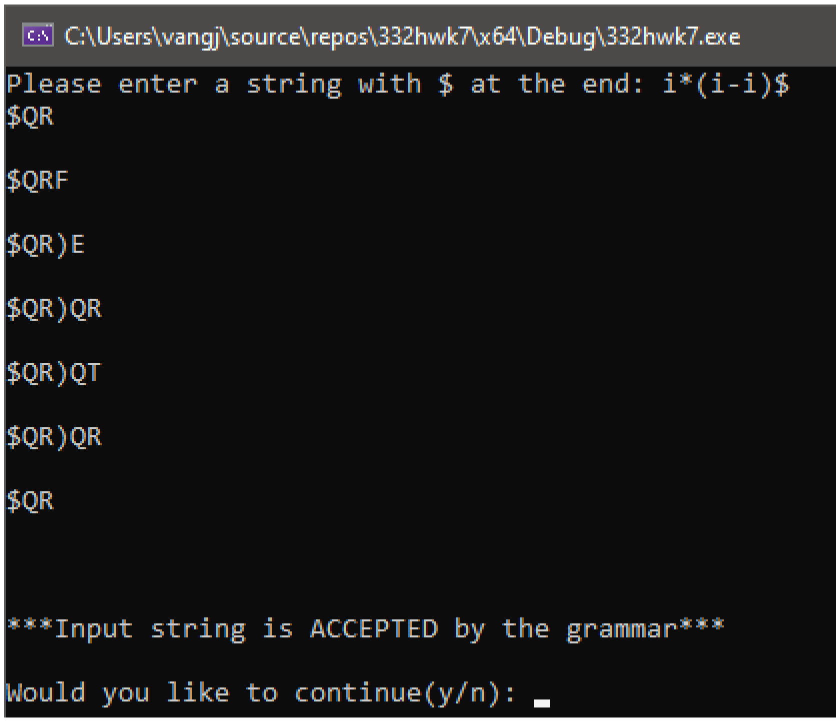
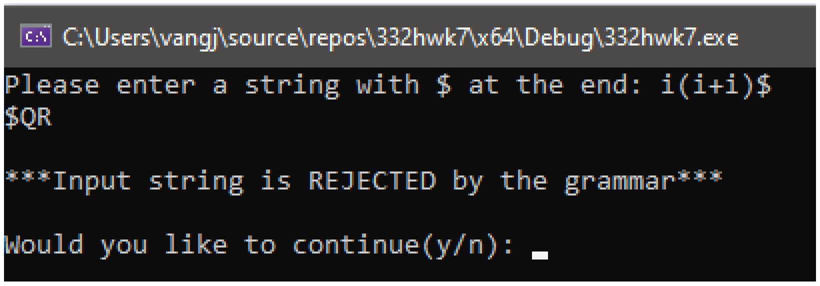
1.
Output:



```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: (i+i)*i$
$QR)E

$QR)QR

$QR)QT

$QR)QR

$QR

$QRF

$QR



***Input string is ACCEPTED by the grammar***
```

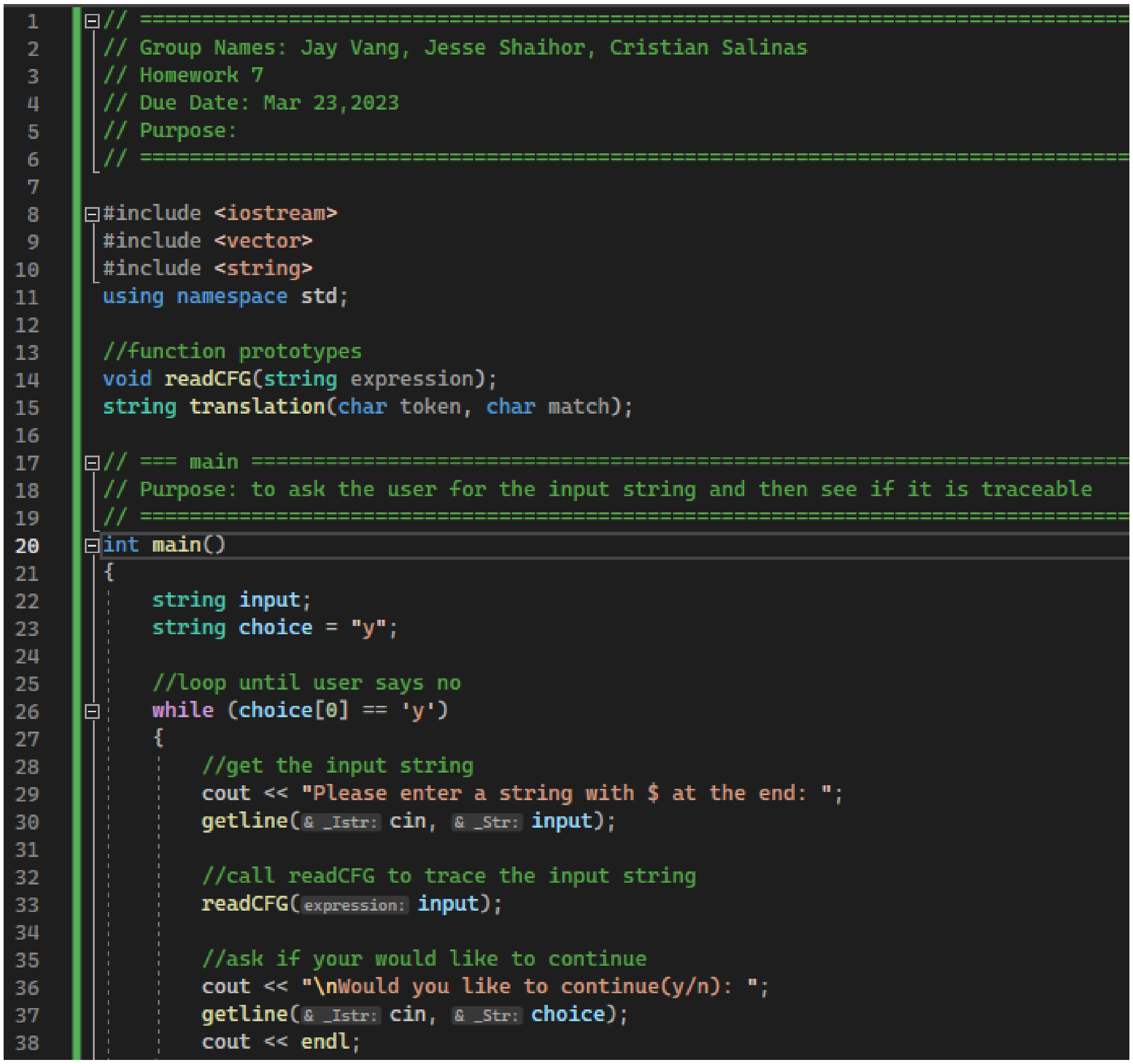


```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: i*(i-i)$
$QR

$QRF

$QR)E

$QR)QR

$QR)QT

$QR)QR

$QR



***Input string is ACCEPTED by the grammar***

Would you like to continue(y/n):
```

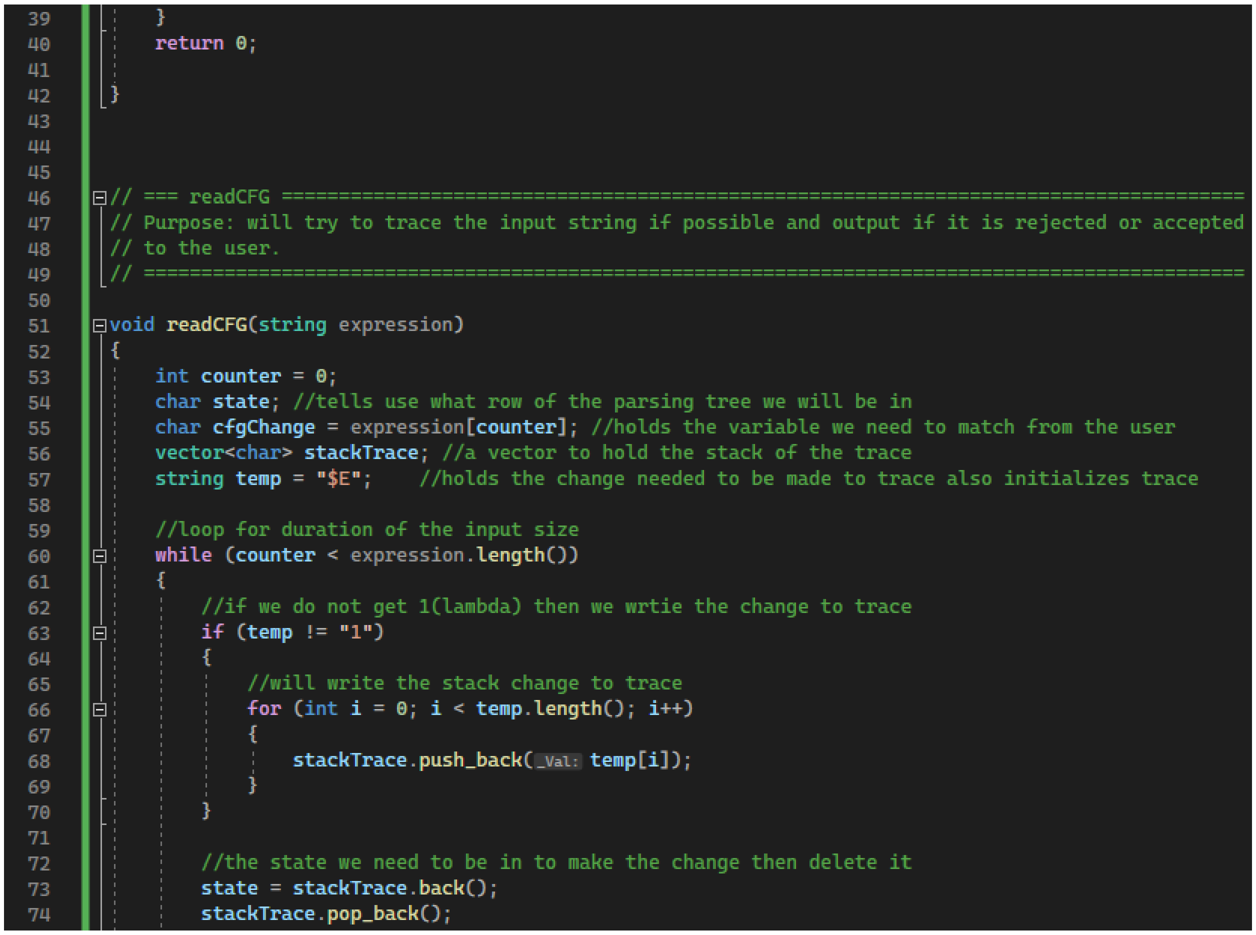


```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: i(i+i)$
$QR

***Input string is REJECTED by the grammar***

Would you like to continue(y/n):
```

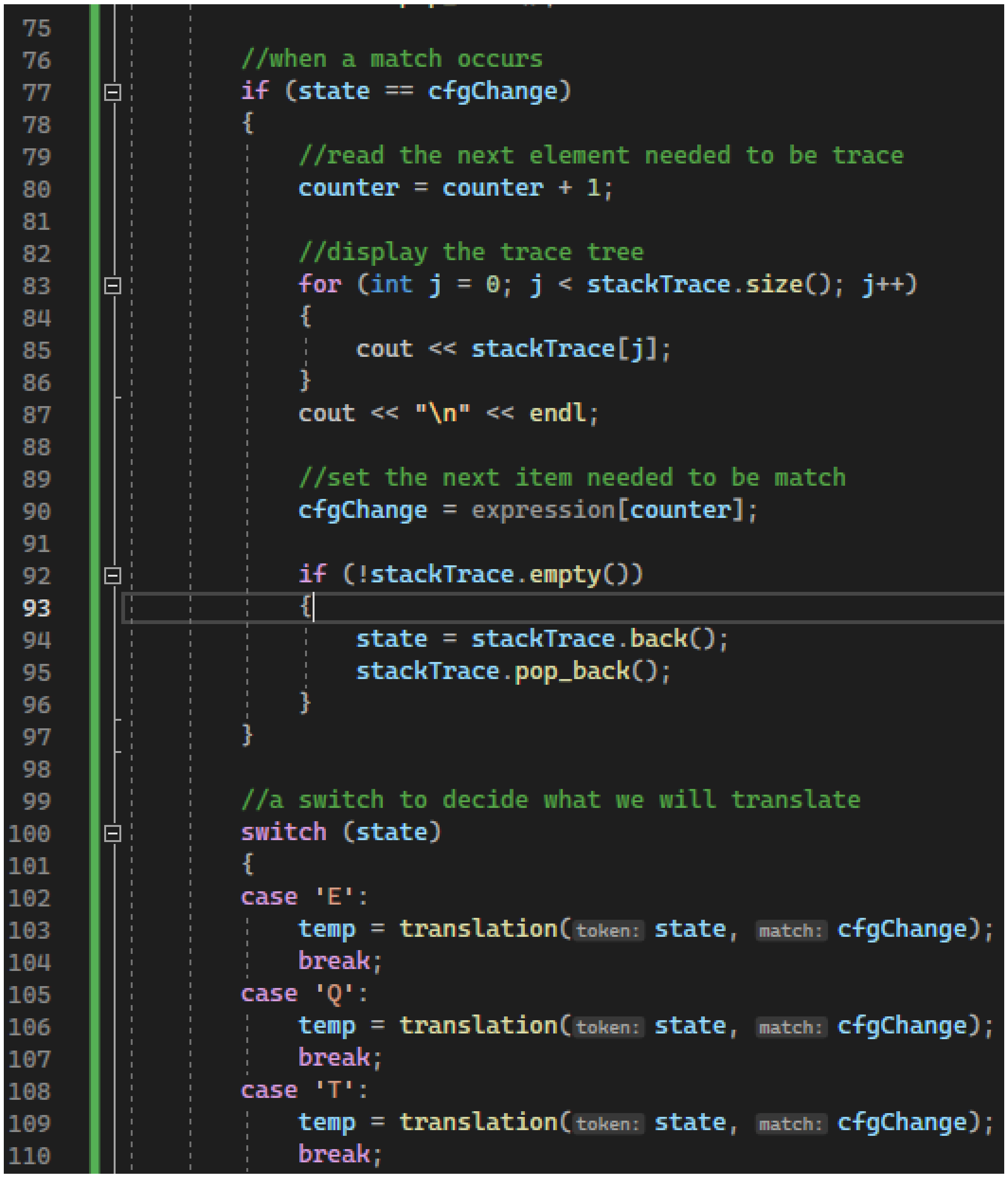
Code:



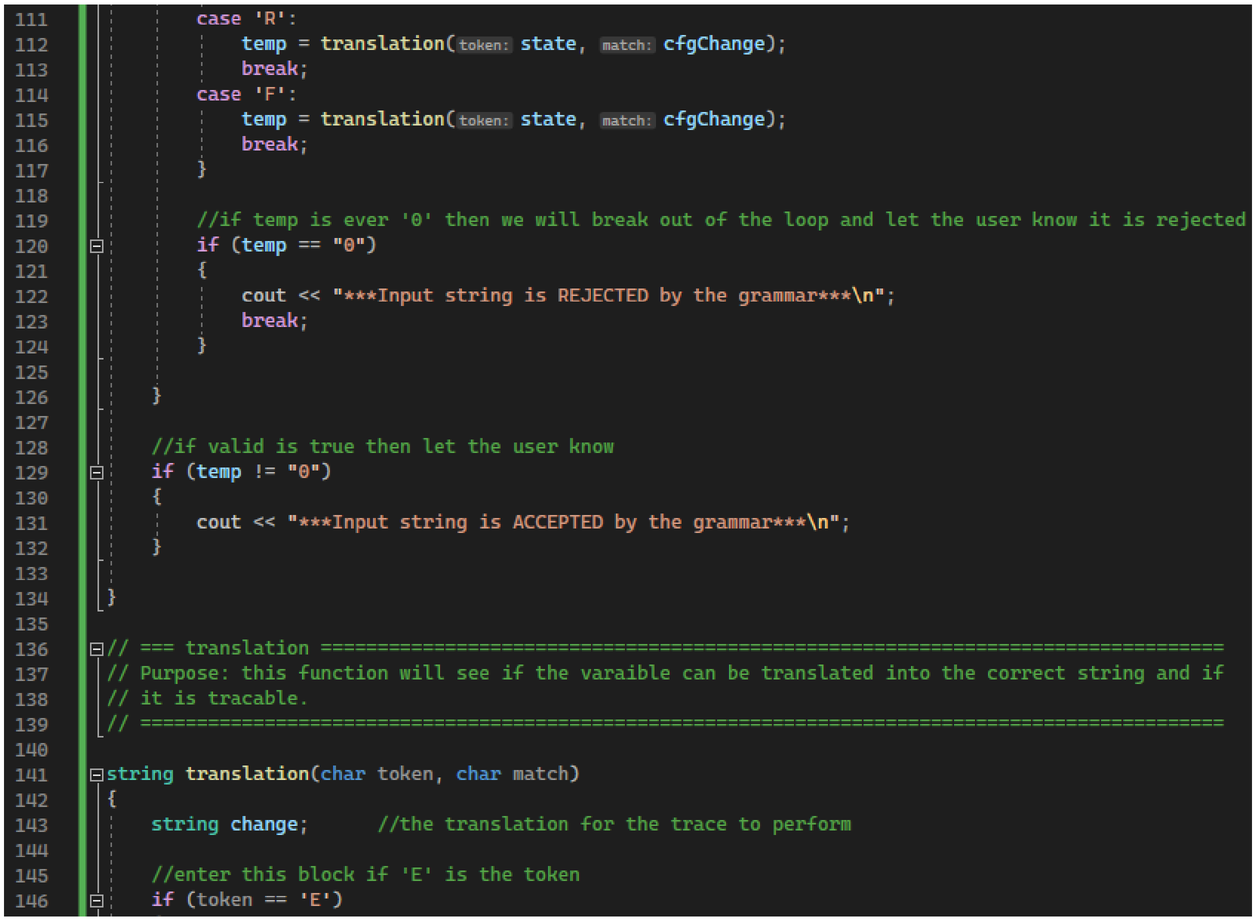
```
// ==================================================================================
// Group Names: Jay Vang, Jesse Shaihor, Cristian Salinas
// Homework 7
// Due Date: Mar 23,2023
// Purpose:
// ==================================================================================

#include <iostream>
#include <vector>
#include <string>
using namespace std;

//function prototypes
void readCFG(string expression);
string translation(char token, char match);

// === main =========================================================================
// Purpose: to ask the user for the input string and then see if it is traceable
// ==================================================================================
int main()
{
    string input;
    string choice = "y";

    //loop until user says no
    while (choice[0] == 'y')
    {
        //get the input string
        cout << "Please enter a string with $ at the end: ";
        getline(& _Istr: cin, & _Str: input);

        //call readCFG to trace the input string
        readCFG(expression: input);

        //ask if your would like to continue
        cout << "\nWould you like to continue(y/n): ";
        getline(& _Istr: cin, & _Str: choice);
        cout << endl;
```

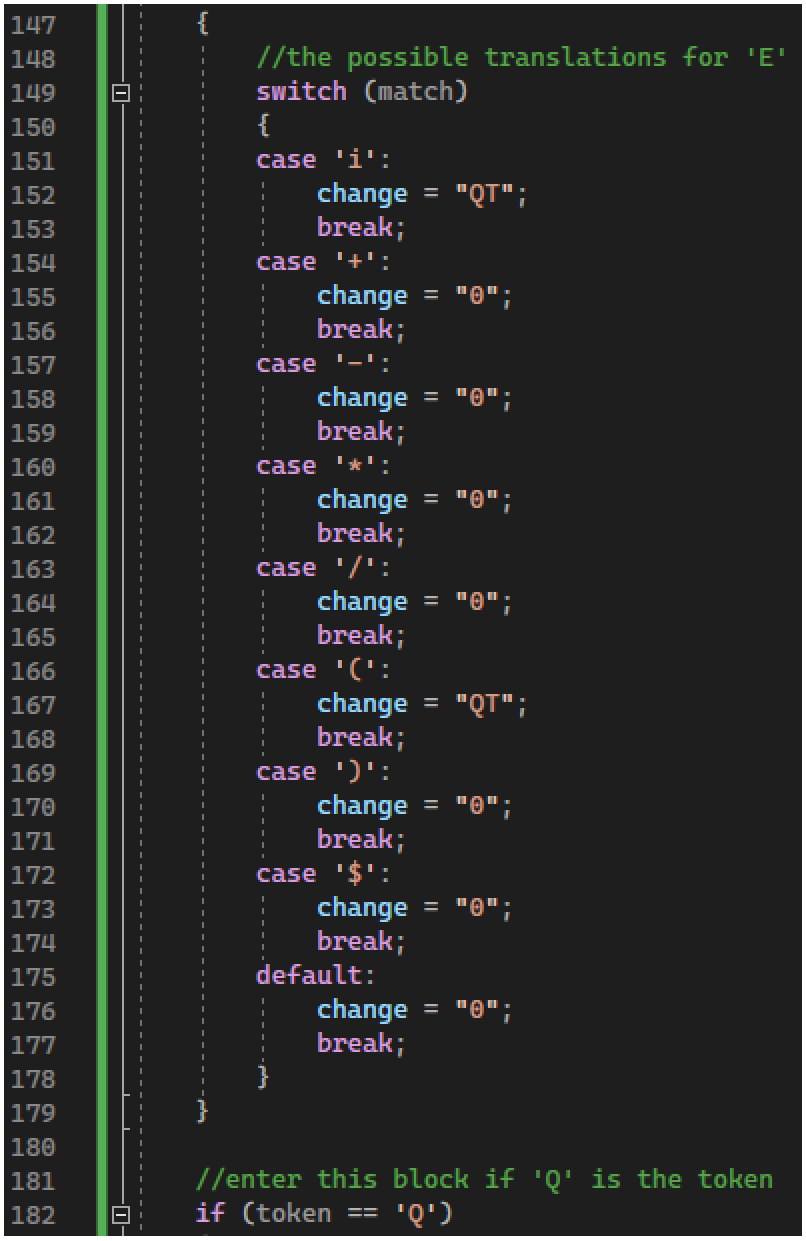


```
    }
    return 0;

}



// === readCFG ============================================================================
// Purpose: will try to trace the input string if possible and output if it is rejected or accepted
// to the user.
// ========================================================================================

void readCFG(string expression)
{
    int counter = 0;
    char state; //tells use what row of the parsing tree we will be in
    char cfgChange = expression[counter]; //holds the variable we need to match from the user
    vector<char> stackTrace; //a vector to hold the stack of the trace
    string temp = "$E";    //holds the change needed to be made to trace also initializes trace

    //loop for duration of the input size
    while (counter < expression.length())
    {
        //if we do not get 1(lambda) then we wrtie the change to trace
        if (temp != "1")
        {
            //will write the stack change to trace
            for (int i = 0; i < temp.length(); i++)
            {
                stackTrace.push_back(_Val: temp[i]);
            }
        }

        //the state we need to be in to make the change then delete it
        state = stackTrace.back();
        stackTrace.pop_back();
```

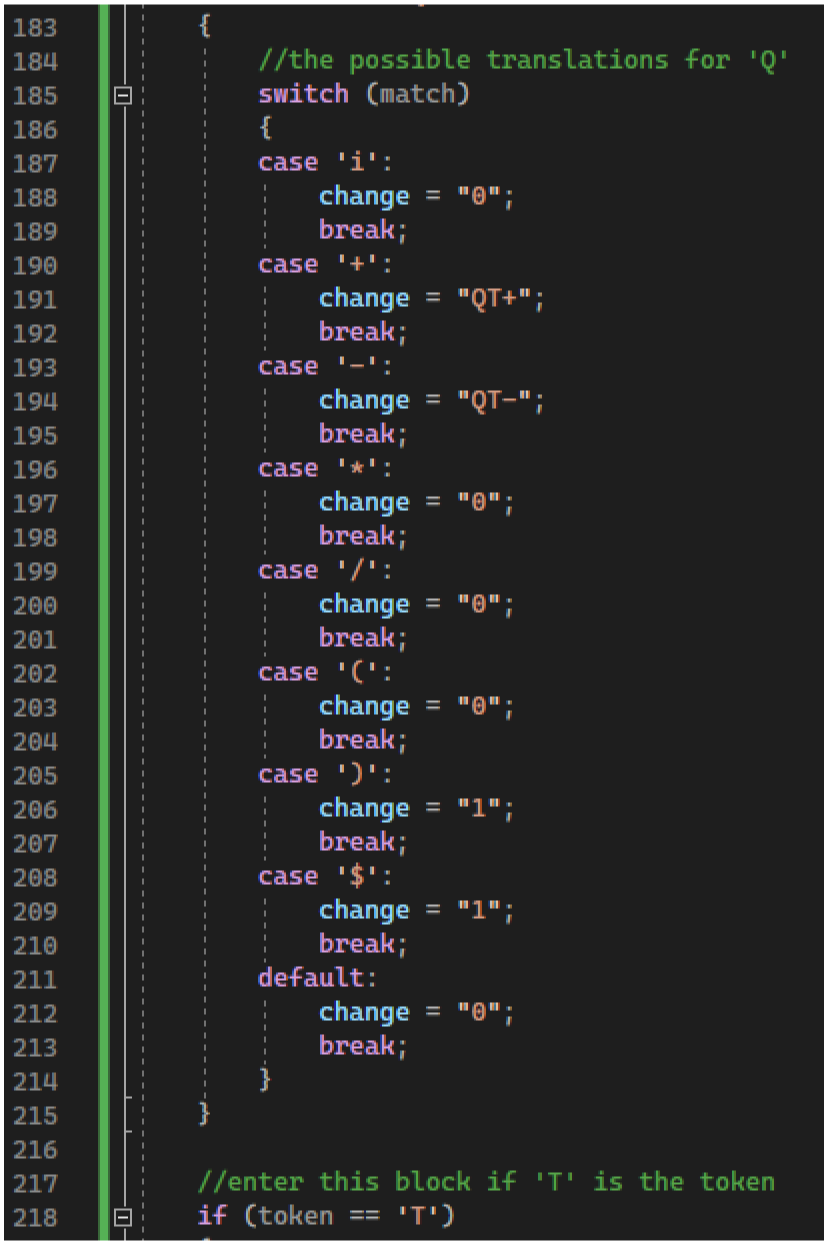


```cpp

        //when a match occurs
        if (state == cfgChange)
        {
            //read the next element needed to be trace
            counter = counter + 1;

            //display the trace tree
            for (int j = 0; j < stackTrace.size(); j++)
            {
                cout << stackTrace[j];
            }
            cout << "\n" << endl;

            //set the next item needed to be match
            cfgChange = expression[counter];

            if (!stackTrace.empty())
            {
                state = stackTrace.back();
                stackTrace.pop_back();
            }
        }

        //a switch to decide what we will translate
        switch (state)
        {
        case 'E':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'Q':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'T':
            temp = translation(token: state, match: cfgChange);
            break;
```

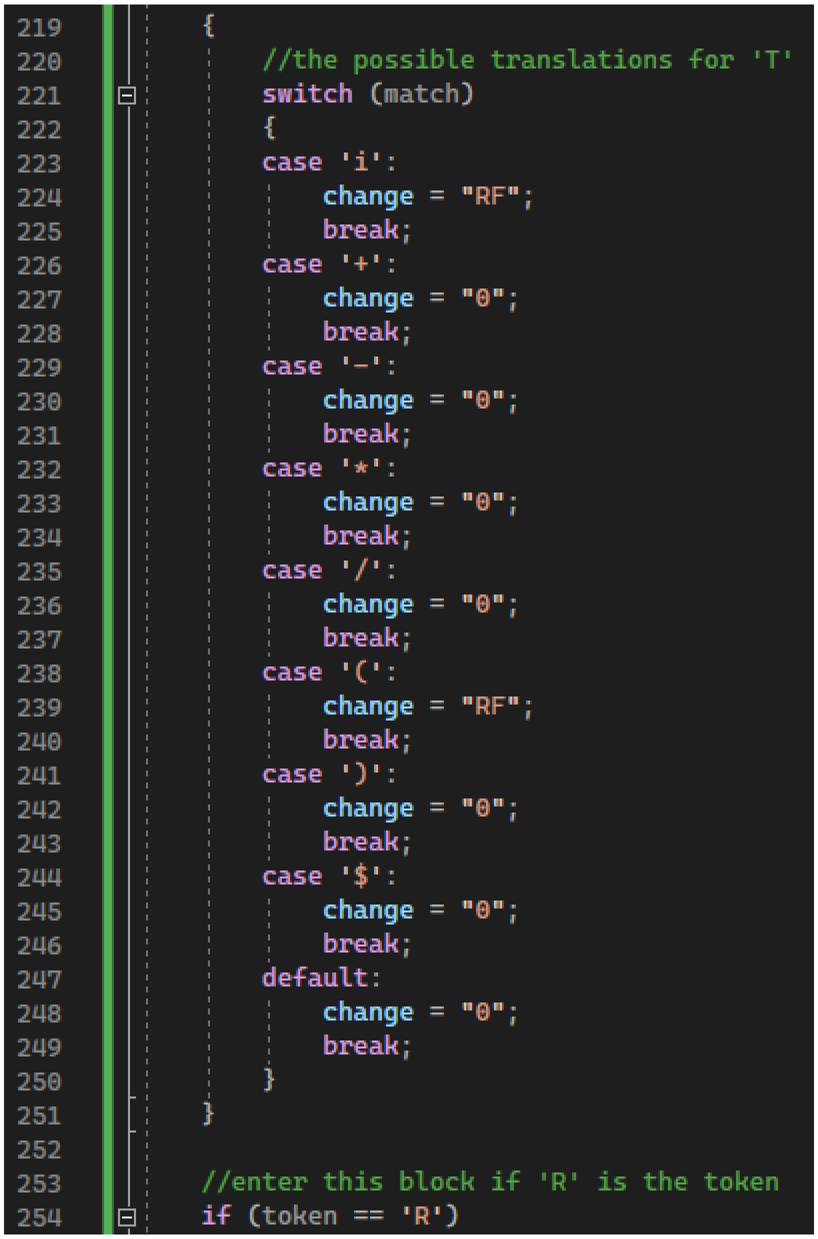

```
            case 'R':
                temp = translation(token: state, match: cfgChange);
                break;
            case 'F':
                temp = translation(token: state, match: cfgChange);
                break;
            }

            //if temp is ever '0' then we will break out of the loop and let the user know it is rejected
            if (temp == "0")
            {
                cout << "***Input string is REJECTED by the grammar***\n";
                break;
            }

        }

        //if valid is true then let the user know
        if (temp != "0")
        {
            cout << "***Input string is ACCEPTED by the grammar***\n";
        }

}

// === translation ==============================================================================
// Purpose: this function will see if the varaible can be translated into the correct string and if
// it is tracable.
// ==============================================================================================

string translation(char token, char match)
{
    string change;      //the translation for the trace to perform

    //enter this block if 'E' is the token
    if (token == 'E')
```



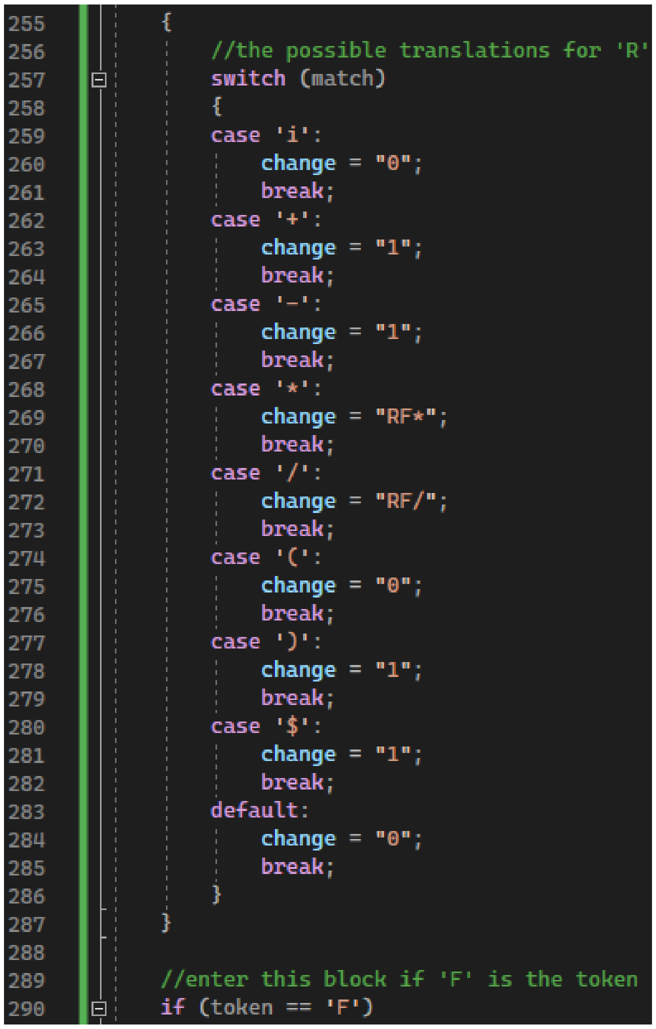
```
        {
            //the possible translations for 'E'
            switch (match)
            {
            case 'i':
                change = "QT";
                break;
            case '+':
                change = "0";
                break;
            case '-':
                change = "0";
                break;
            case '*':
                change = "0";
                break;
            case '/':
                change = "0";
                break;
            case '(':
                change = "QT";
                break;
            case ')':
                change = "0";
                break;
            case '$':
                change = "0";
                break;
            default:
                change = "0";
                break;
            }
        }

        //enter this block if 'Q' is the token
        if (token == 'Q')
```



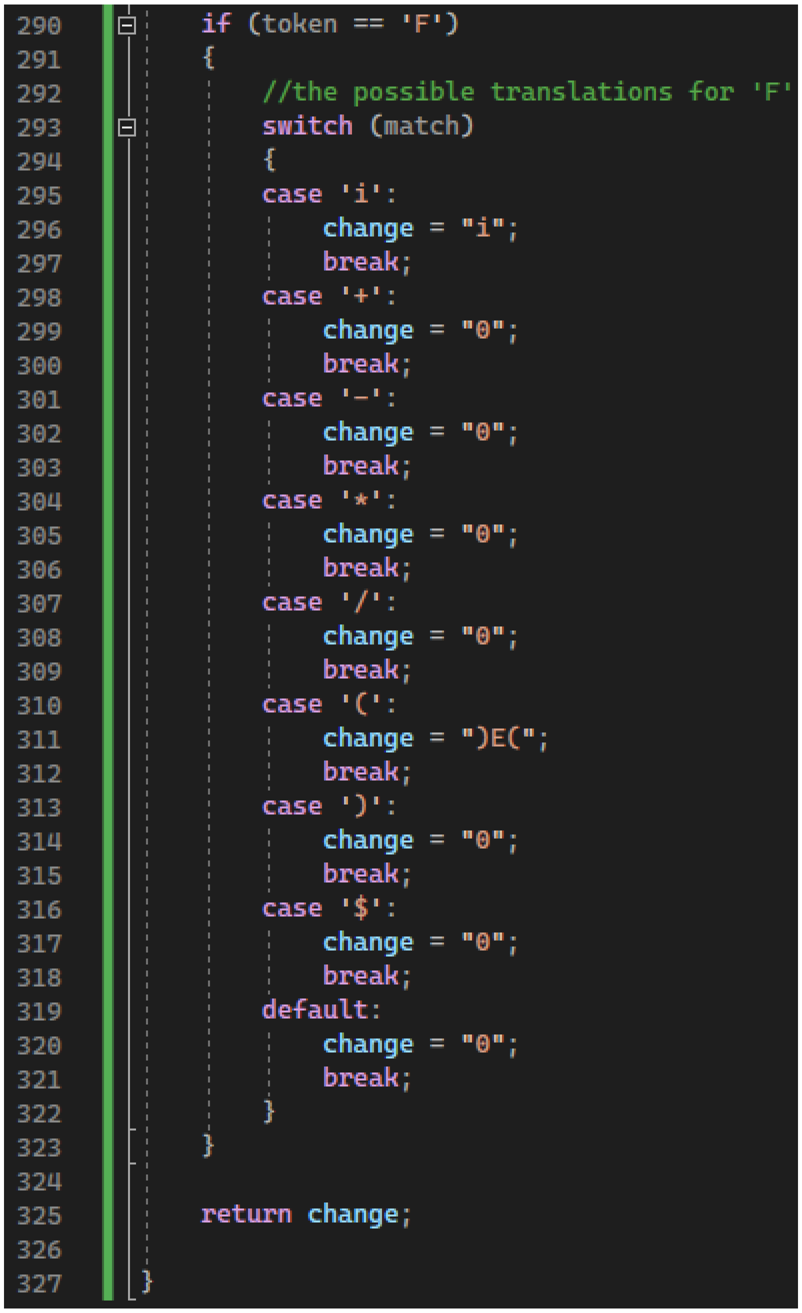
```
    {
        //the possible translations for 'Q'
        switch (match)
        {
        case 'i':
            change = "0";
            break;
        case '+':
            change = "QT+";
            break;
        case '-':
            change = "QT-";
            break;
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = "0";
            break;
        case ')':
            change = "1";
            break;
        case '$':
            change = "1";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'T' is the token
    if (token == 'T')
```



```
    {
        //the possible translations for 'T'
        switch (match)
        {
        case 'i':
            change = "RF";
            break;
        case '+':
            change = "0";
            break;
        case '-':
            change = "0";
            break;
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = "RF";
            break;
        case ')':
            change = "0";
            break;
        case '$':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'R' is the token
    if (token == 'R')
```

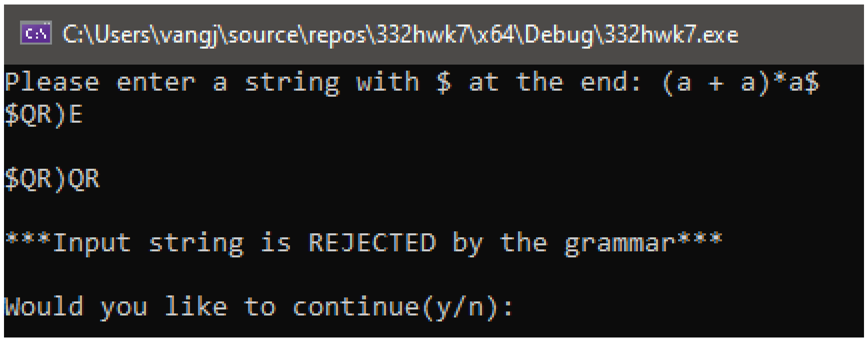


```
    {
        //the possible translations for 'R'
        switch (match)
        {
        case 'i':
            change = "0";
            break;
        case '+':
            change = "1";
            break;
        case '-':
            change = "1";
            break;
        case '*':
            change = "RF*";
            break;
        case '/':
            change = "RF/";
            break;
        case '(':
            change = "0";
            break;
        case ')':
            change = "1";
            break;
        case '$':
            change = "1";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'F' is the token
    if (token == 'F')
```

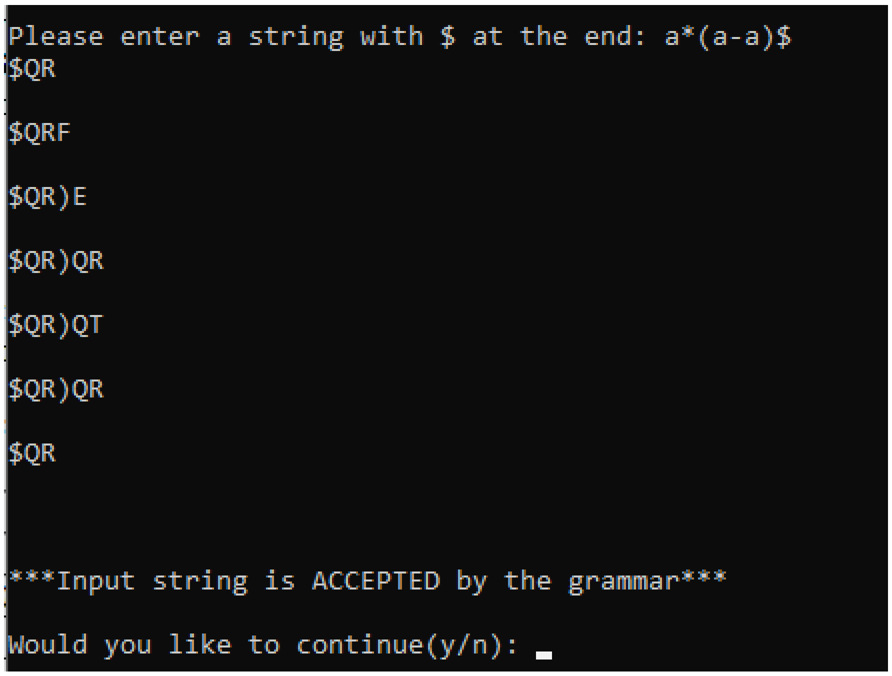


```
    if (token == 'F')
    {
        //the possible translations for 'F'
        switch (match)
        {
        case 'i':
            change = "i";
            break;
        case '+':
            change = "0";
            break;
        case '-':
            change = "0";
            break;
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = ")E(";
            break;
        case ')':
            change = "0";
            break;
        case '$':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    return change;

}
```

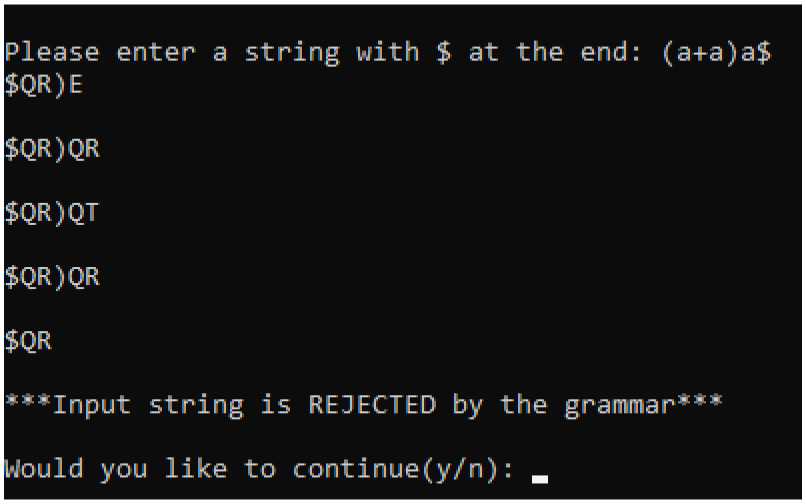
2.
Output:



```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: (a + a)*a$
$QR)E

$QR)QR

***Input string is REJECTED by the grammar***

Would you like to continue(y/n):
```

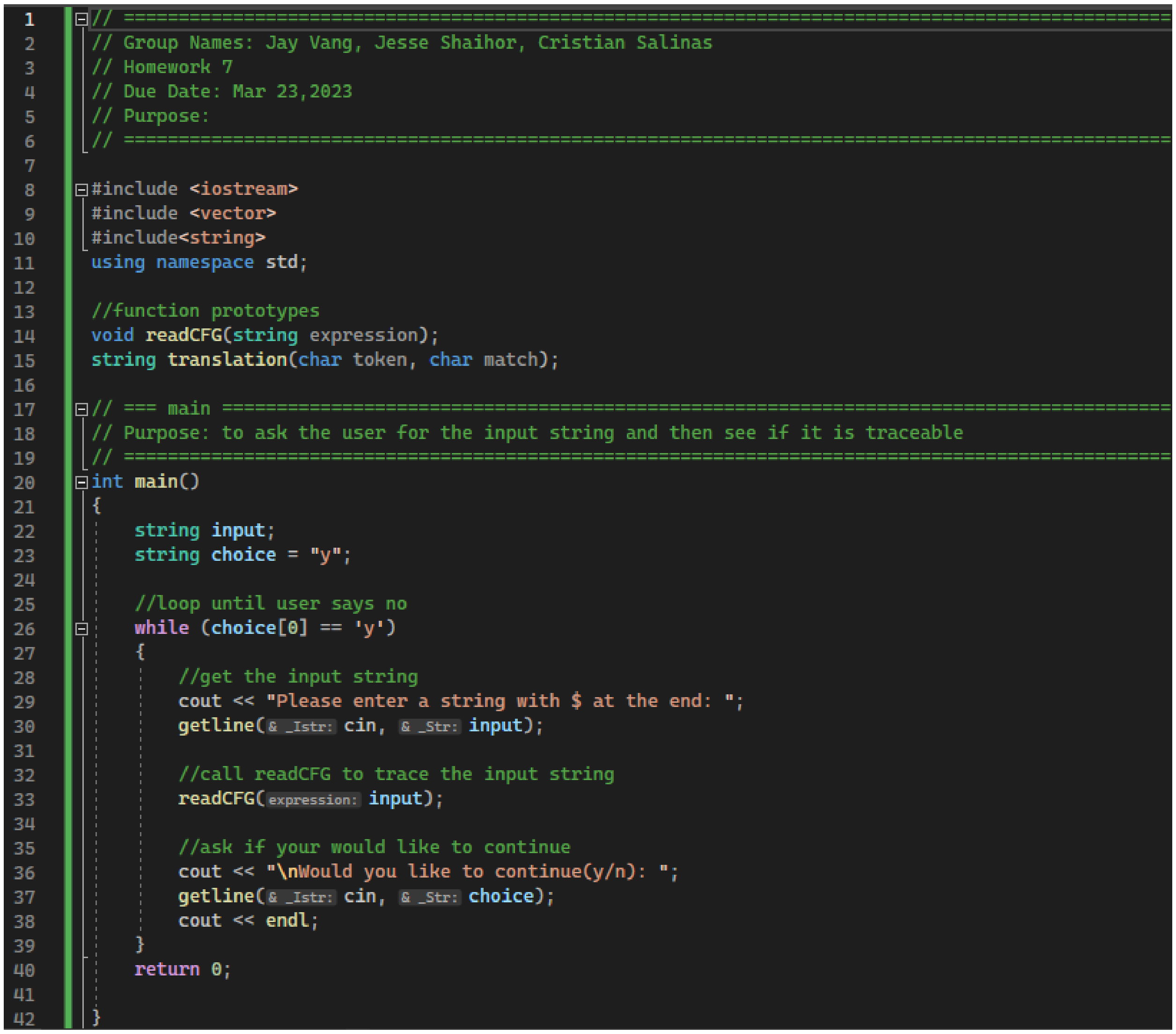


```
Please enter a string with $ at the end: a*(a-a)$
$QR

$QRF

$QR)E

$QR)QR

$QR)QT

$QR)QR

$QR


***Input string is ACCEPTED by the grammar***

Would you like to continue(y/n):
```

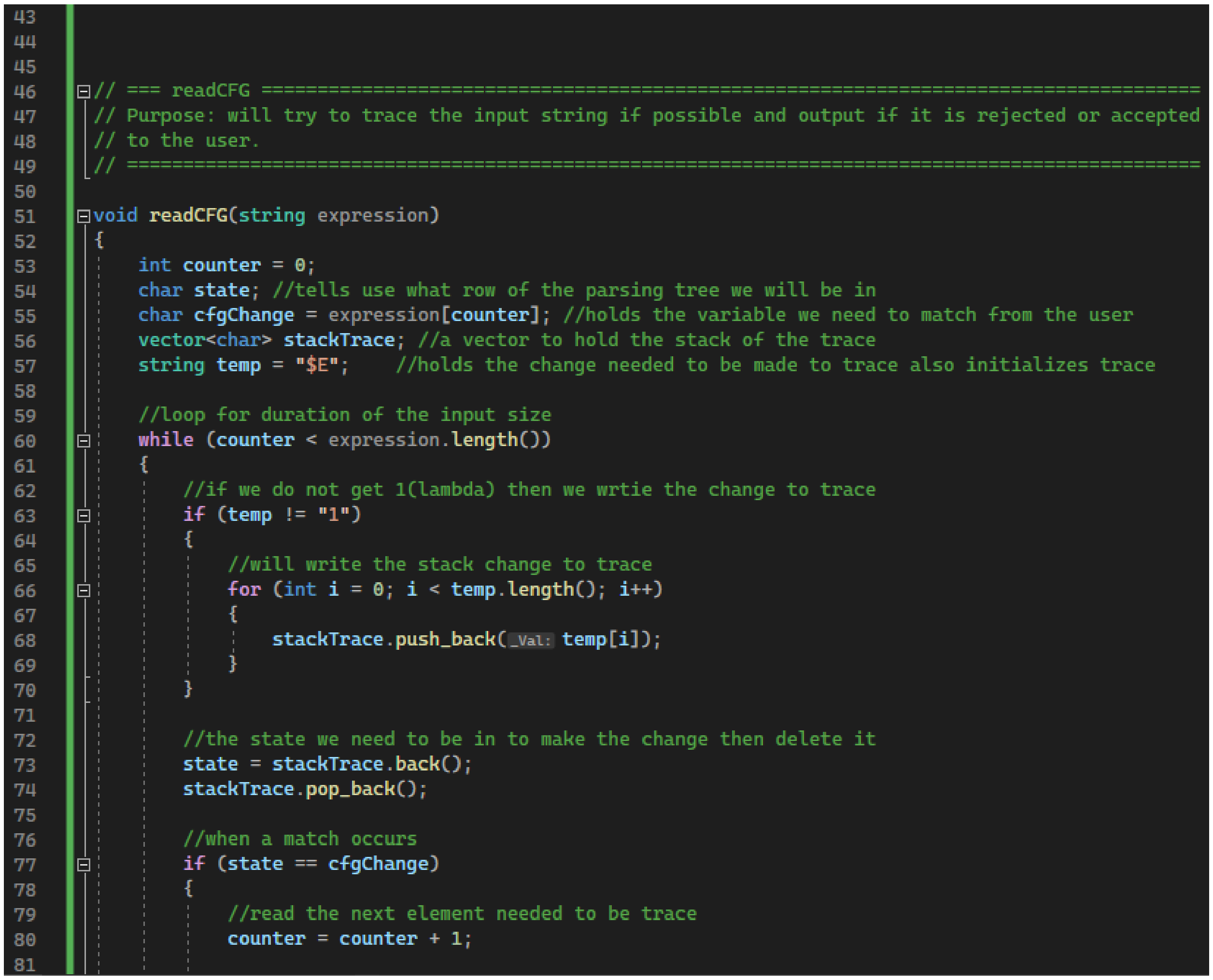


```
Please enter a string with $ at the end: (a+a)a$
$QR)E

$QR)QR

$QR)QT

$QR)QR

$QR

***Input string is REJECTED by the grammar***

Would you like to continue(y/n):
```

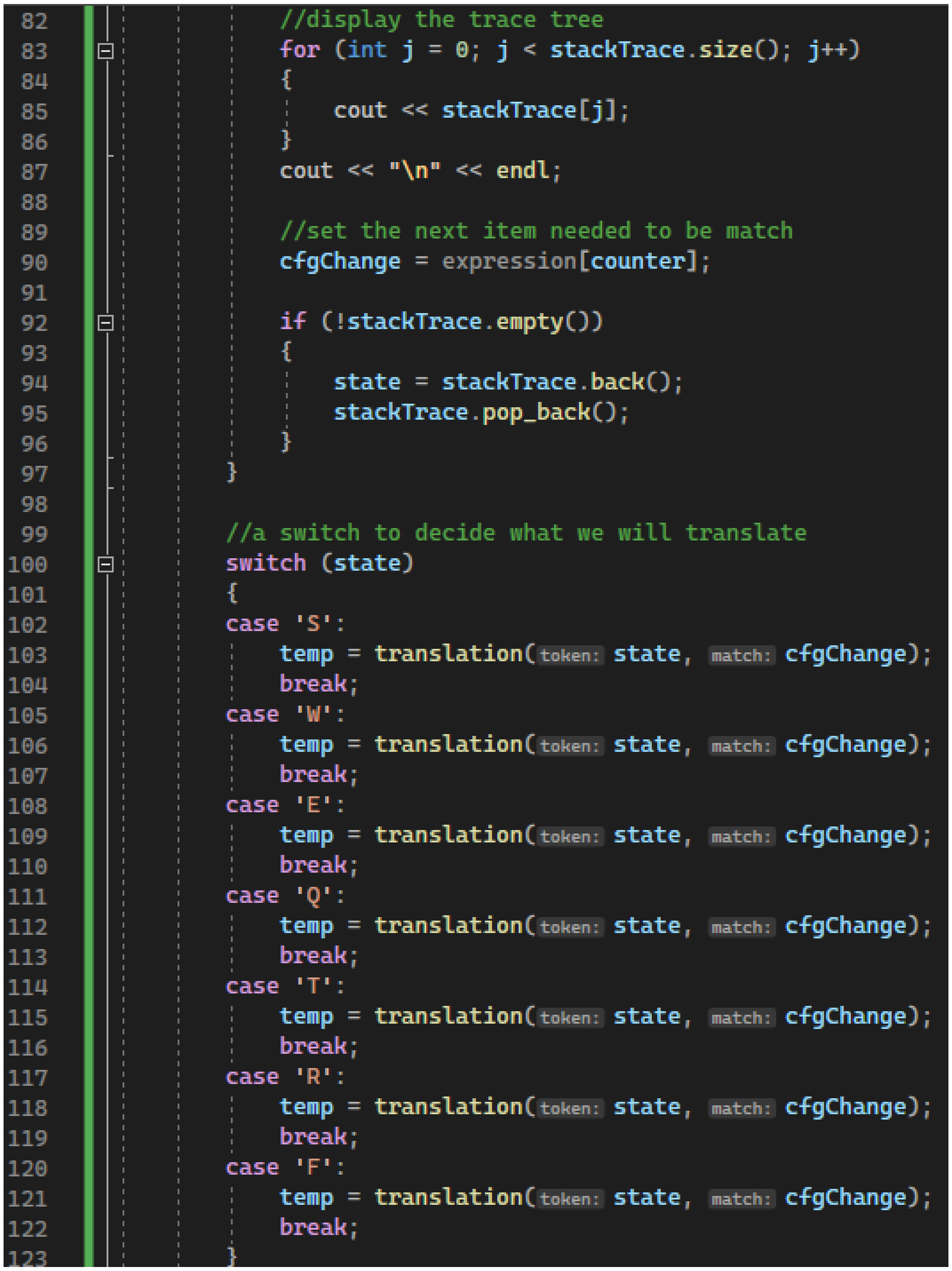
Code:



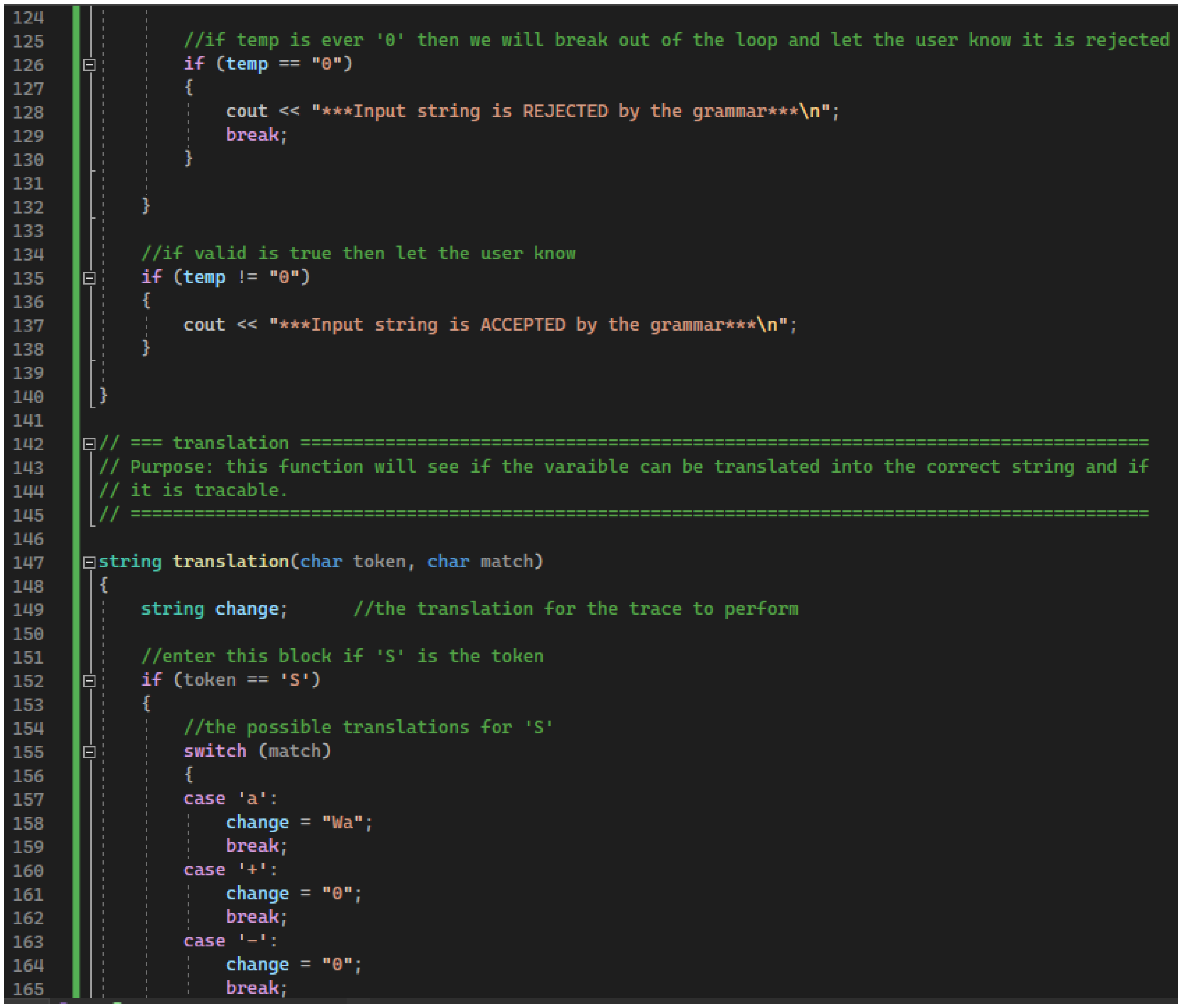
```cpp
// ===============================================================================================
// Group Names: Jay Vang, Jesse Shaihor, Cristian Salinas
// Homework 7
// Due Date: Mar 23,2023
// Purpose:
// ===============================================================================================

#include <iostream>
#include <vector>
#include<string>
using namespace std;

//function prototypes
void readCFG(string expression);
string translation(char token, char match);

// === main ======================================================================================
// Purpose: to ask the user for the input string and then see if it is traceable
// ===============================================================================================
int main()
{
    string input;
    string choice = "y";

    //loop until user says no
    while (choice[0] == 'y')
    {
        //get the input string
        cout << "Please enter a string with $ at the end: ";
        getline(& _Istr: cin, & _Str: input);

        //call readCFG to trace the input string
        readCFG(expression: input);

        //ask if your would like to continue
        cout << "\nWould you like to continue(y/n): ";
        getline(& _Istr: cin, & _Str: choice);
        cout << endl;
    }
    return 0;

}
```



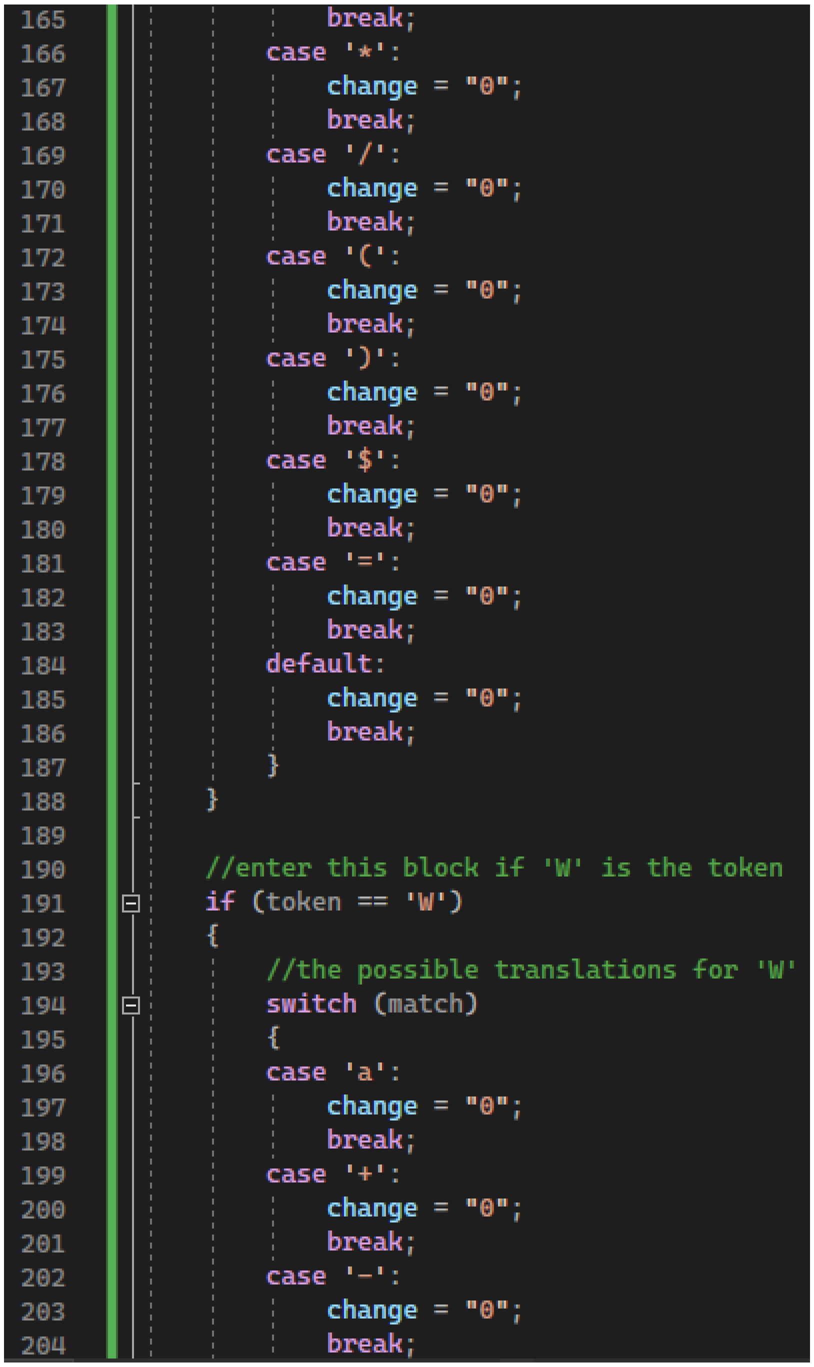
```cpp



// === readCFG ==========================================================================
// Purpose: will try to trace the input string if possible and output if it is rejected or accepted
// to the user.
// ======================================================================================

void readCFG(string expression)
{
    int counter = 0;
    char state; //tells use what row of the parsing tree we will be in
    char cfgChange = expression[counter]; //holds the variable we need to match from the user
    vector<char> stackTrace; //a vector to hold the stack of the trace
    string temp = "$E";    //holds the change needed to be made to trace also initializes trace

    //loop for duration of the input size
    while (counter < expression.length())
    {
        //if we do not get 1(lambda) then we wrtie the change to trace
        if (temp != "1")
        {
            //will write the stack change to trace
            for (int i = 0; i < temp.length(); i++)
            {
                stackTrace.push_back(_Val: temp[i]);
            }
        }

        //the state we need to be in to make the change then delete it
        state = stackTrace.back();
        stackTrace.pop_back();

        //when a match occurs
        if (state == cfgChange)
        {
            //read the next element needed to be trace
            counter = counter + 1;

```



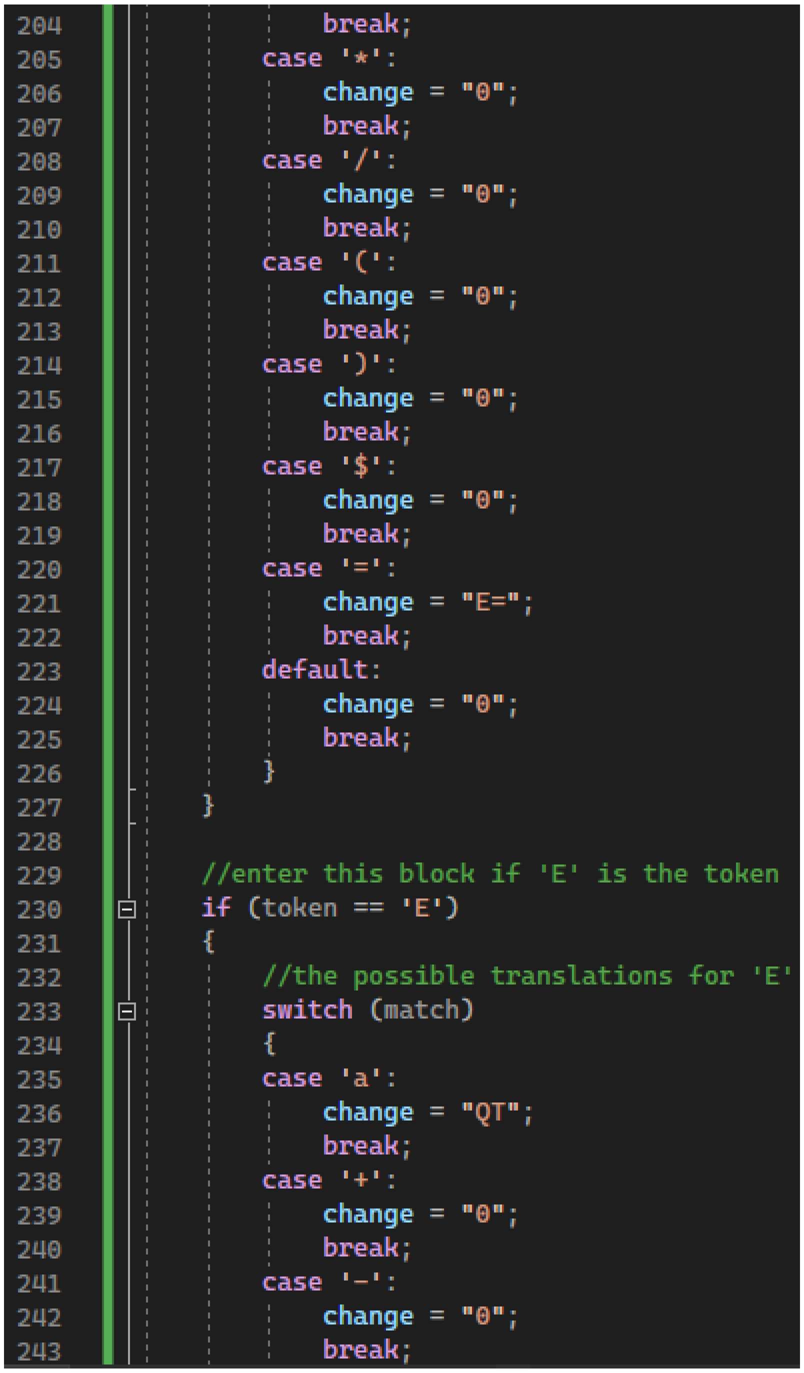
```
            //display the trace tree
            for (int j = 0; j < stackTrace.size(); j++)
            {
                cout << stackTrace[j];
            }
            cout << "\n" << endl;

            //set the next item needed to be match
            cfgChange = expression[counter];

            if (!stackTrace.empty())
            {
                state = stackTrace.back();
                stackTrace.pop_back();
            }
        }

        //a switch to decide what we will translate
        switch (state)
        {
        case 'S':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'W':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'E':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'Q':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'T':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'R':
            temp = translation(token: state, match: cfgChange);
            break;
        case 'F':
            temp = translation(token: state, match: cfgChange);
            break;
        }
```



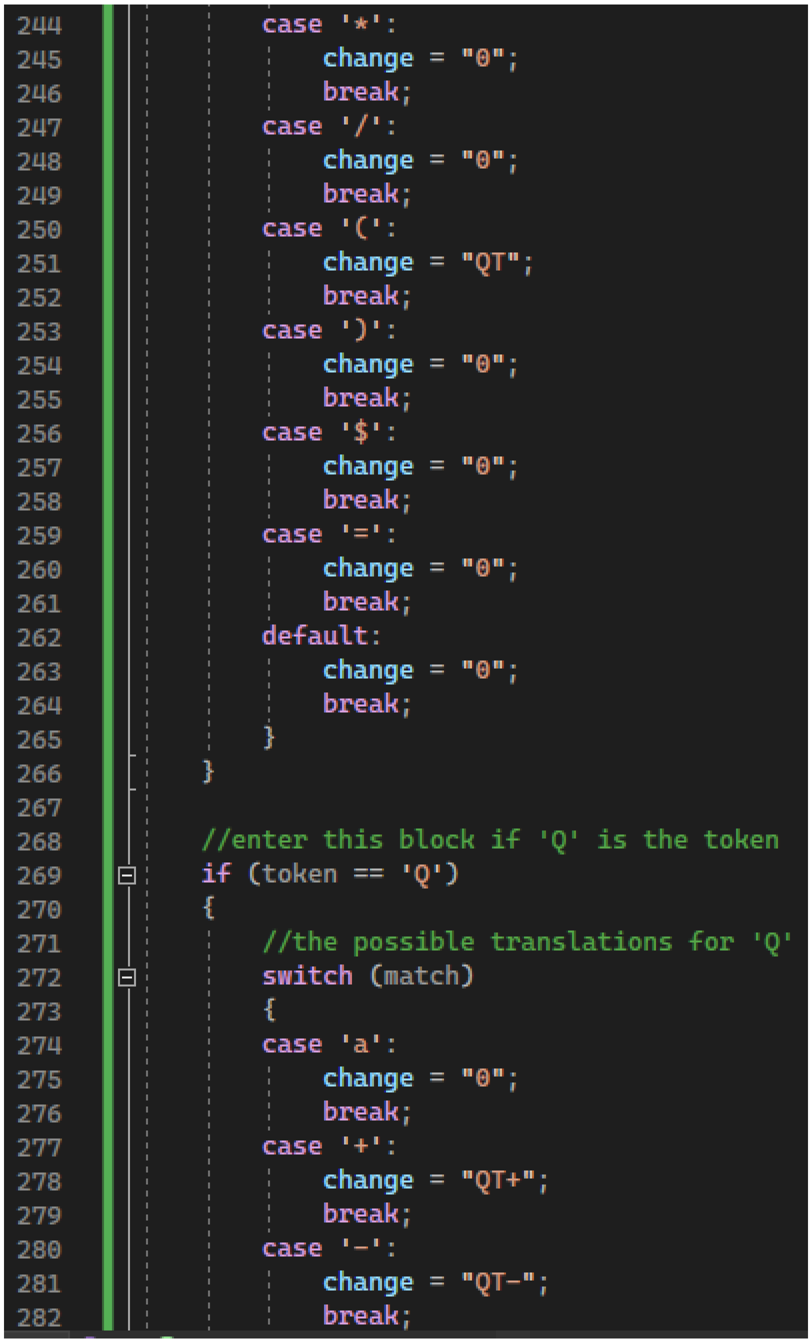
```cpp

        //if temp is ever '0' then we will break out of the loop and let the user know it is rejected
        if (temp == "0")
        {
            cout << "***Input string is REJECTED by the grammar***\n";
            break;
        }

    }

    //if valid is true then let the user know
    if (temp != "0")
    {
        cout << "***Input string is ACCEPTED by the grammar***\n";
    }

}

// === translation ==============================================================================
// Purpose: this function will see if the varaible can be translated into the correct string and if
// it is tracable.
// ==============================================================================================

string translation(char token, char match)
{
    string change;      //the translation for the trace to perform

    //enter this block if 'S' is the token
    if (token == 'S')
    {
        //the possible translations for 'S'
        switch (match)
        {
        case 'a':
            change = "Wa";
            break;
        case '+':
            change = "0";
            break;
        case '-':
            change = "0";
            break;
```



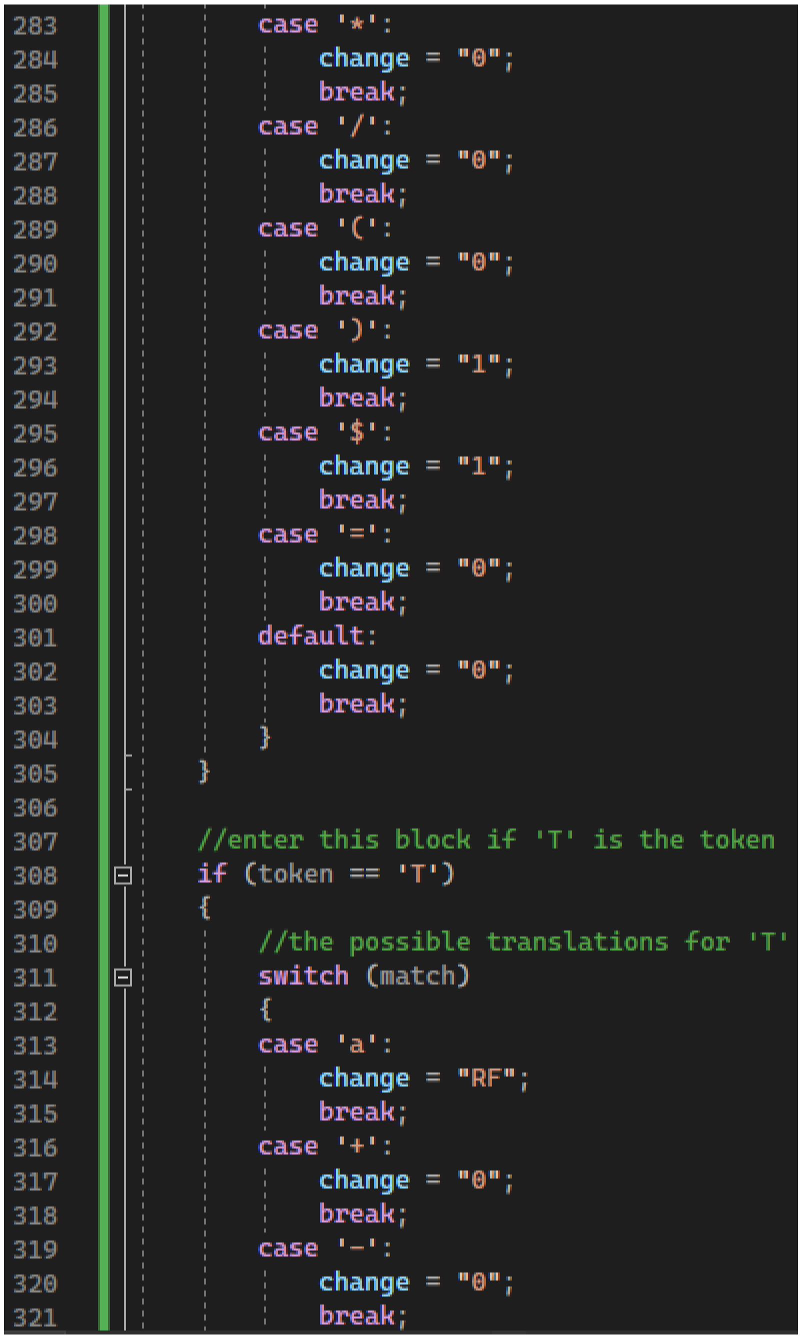
```
                break;
            case '*':
                change = "0";
                break;
            case '/':
                change = "0";
                break;
            case '(':
                change = "0";
                break;
            case ')':
                change = "0";
                break;
            case '$':
                change = "0";
                break;
            case '=':
                change = "0";
                break;
            default:
                change = "0";
                break;
            }
        }

        //enter this block if 'W' is the token
        if (token == 'W')
        {
            //the possible translations for 'W'
            switch (match)
            {
            case 'a':
                change = "0";
                break;
            case '+':
                change = "0";
                break;
            case '-':
                change = "0";
                break;
```



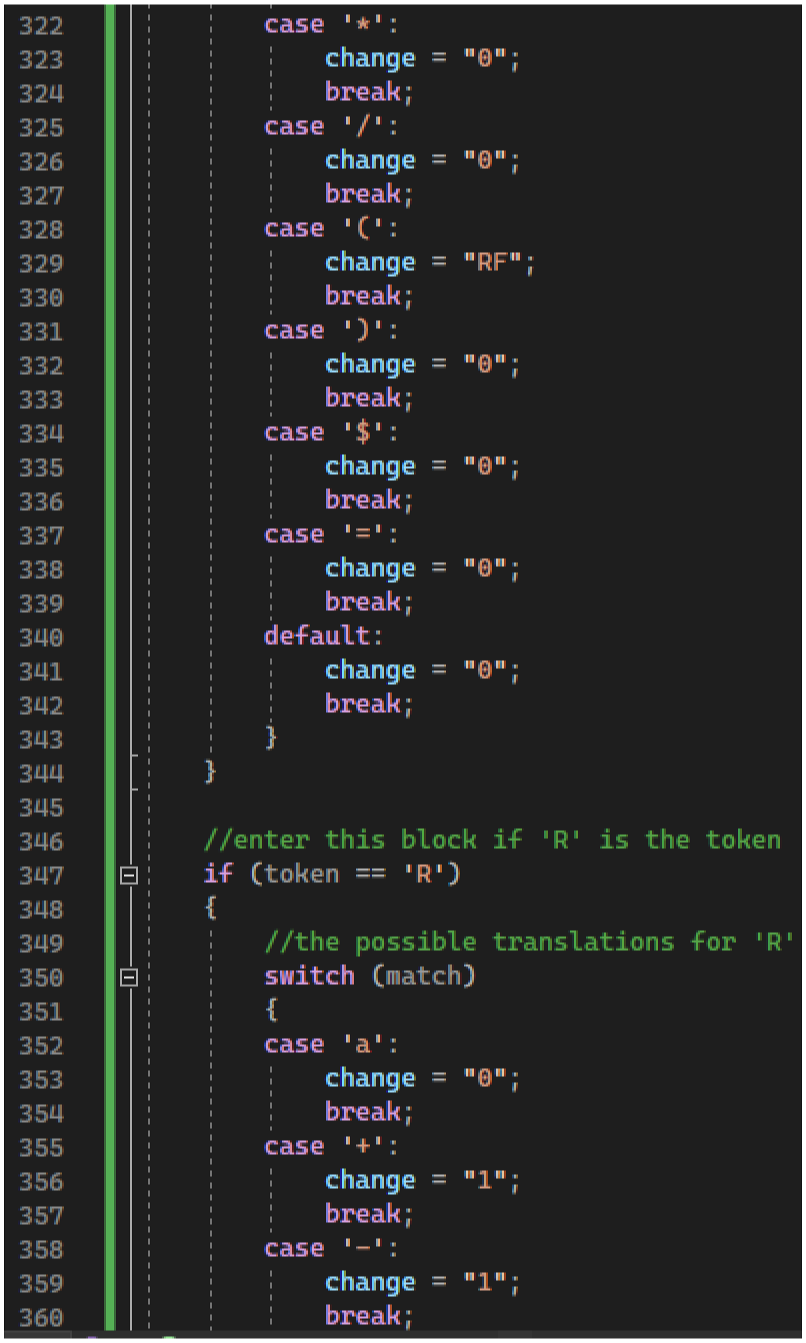
```
                break;
            case '*':
                change = "0";
                break;
            case '/':
                change = "0";
                break;
            case '(':
                change = "0";
                break;
            case ')':
                change = "0";
                break;
            case '$':
                change = "0";
                break;
            case '=':
                change = "E=";
                break;
            default:
                change = "0";
                break;
            }
        }

        //enter this block if 'E' is the token
        if (token == 'E')
        {
            //the possible translations for 'E'
            switch (match)
            {
            case 'a':
                change = "QT";
                break;
            case '+':
                change = "0";
                break;
            case '-':
                change = "0";
                break;
```



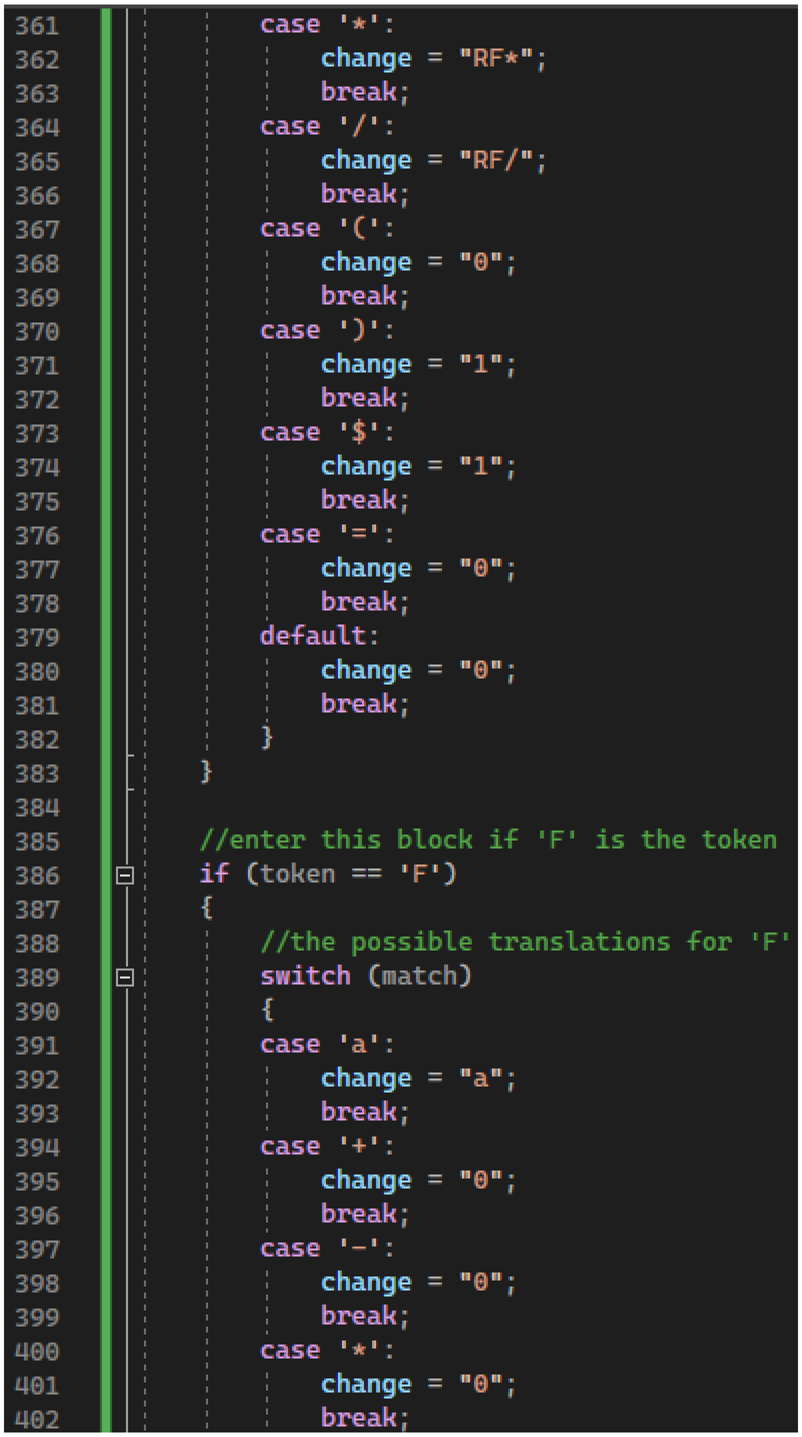
```
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = "QT";
            break;
        case ')':
            change = "0";
            break;
        case '$':
            change = "0";
            break;
        case '=':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'Q' is the token
    if (token == 'Q')
    {
        //the possible translations for 'Q'
        switch (match)
        {
        case 'a':
            change = "0";
            break;
        case '+':
            change = "QT+";
            break;
        case '-':
            change = "QT-";
            break;
```



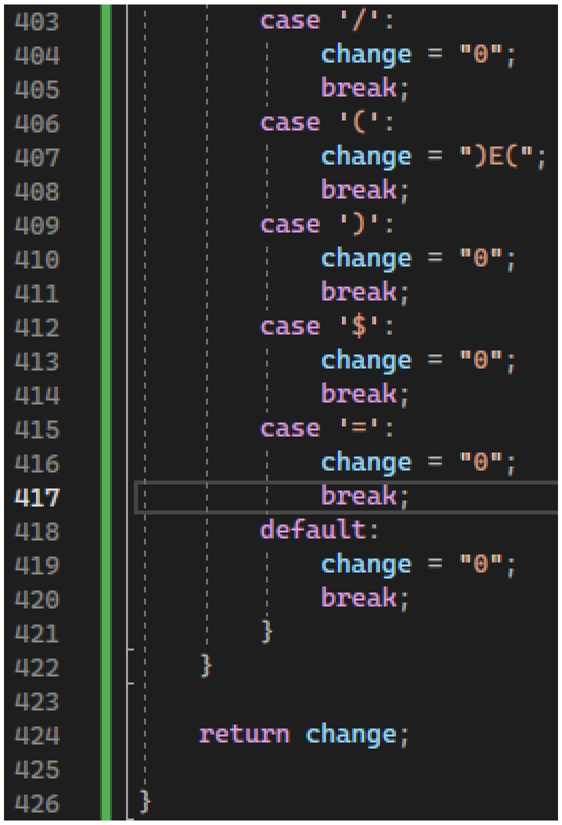
```
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = "0";
            break;
        case ')':
            change = "1";
            break;
        case '$':
            change = "1";
            break;
        case '=':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'T' is the token
    if (token == 'T')
    {
        //the possible translations for 'T'
        switch (match)
        {
        case 'a':
            change = "RF";
            break;
        case '+':
            change = "0";
            break;
        case '-':
            change = "0";
            break;
```



```
        case '*':
            change = "0";
            break;
        case '/':
            change = "0";
            break;
        case '(':
            change = "RF";
            break;
        case ')':
            change = "0";
            break;
        case '$':
            change = "0";
            break;
        case '=':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'R' is the token
    if (token == 'R')
    {
        //the possible translations for 'R'
        switch (match)
        {
        case 'a':
            change = "0";
            break;
        case '+':
            change = "1";
            break;
        case '-':
            change = "1";
            break;
```



```
        case '*':
            change = "RF*";
            break;
        case '/':
            change = "RF/";
            break;
        case '(':
            change = "0";
            break;
        case ')':
            change = "1";
            break;
        case '$':
            change = "1";
            break;
        case '=':
            change = "0";
            break;
        default:
            change = "0";
            break;
        }
    }

    //enter this block if 'F' is the token
    if (token == 'F')
    {
        //the possible translations for 'F'
        switch (match)
        {
        case 'a':
            change = "a";
            break;
        case '+':
            change = "0";
            break;
        case '-':
            change = "0";
            break;
        case '*':
            change = "0";
            break;
```



```
            case '/':
                change = "0";
                break;
            case '(':
                change = ")E(";
                break;
            case ')':
                change = "0";
                break;
            case '$':
                change = "0";
                break;
            case '=':
                change = "0";
                break;
            default:
                change = "0";
                break;
            }
        }

        return change;

}
```