Code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <vector>
#include <unordered_set>

using namespace std;

bool isIdentifier(const string& word) {
    if (word.empty() || isdigit(word[0])) {
        return false;
    }
    for (char ch : word) {
        if (!isalpha(ch) && !isdigit(ch) && ch != '_') {
            return false;
        }
    }
    return true;
}
bool isKeyword(const string& word) {
    static unordered_set<string> keywords = {
        "int", "float", "double", "char",
"bool","cin","cout","while","for","if","else","do","buffer","string"
        // Add more keywords as needed
    };
    return keywords.count(word) > 0;
}

int main()
{
    string line;
    vector<string> keywords = { "int", "float", "double", "char",
"bool","cin","cout","while","for","if","else","do","buffer","string"};
    vector<string> constants;
    vector<string> identifiers;
    bool isComment = false;

    ifstream file("Input.txt");
    ofstream output("output.txt", ios::app); // Open file in append mode

    if (file.is_open())
    {
        while (getline(file, line))
        {

            // Remove white spaces
            if (!isComment)
            {
                output << line << endl;
                line.erase(remove(line.begin(), line.end(), ' '),
line.end());
```

```
                // Recognize keywords
                for (auto keyword : keywords)
                {
                    if (line.find(keyword) != string::npos)
                    {
                        output << "Keyword: " << keyword << endl;
                    }
                }


            }
            // Recognize constants
            string::size_type pos = 0;
            while (pos < line.length())
            {
                if (isdigit(line[pos]))
                {
                    string constant;
                    while (pos < line.length() && (isdigit(line[pos]) ||
line[pos] == '.'))
                    {
                        constant += line[pos];
                        pos++;
                    }
                    constants.push_back(constant);
                }
                else
                {
                    pos++;
                }
            }

            // Recognize identifiers
            pos = 0;
            while (pos < line.length())
            {
                if (isalpha(line[pos]) || line[pos] == '_')
                {
                    string identifier;
                    while (pos < line.length() && (isalnum(line[pos]) ||
line[pos] == '_'))
                    {
                        identifier += line[pos];
                        pos++;
                    }
                    identifiers.push_back(identifier);
                }
                else
                {
                    pos++;
                }
            }
            // Remove comments
```

```
            line.erase(remove(line.begin(), line.end(), ' '), line.end());
            if (line.find("//") != string::npos)
            {
                line.erase(line.find("//"), line.length());
            }
            if (line.find("/*") != string::npos)
            {
                isComment = true;
                line.erase(line.find("/*"), line.length());
            }
            if (line.find("*/") != string::npos)
            {
                isComment = false;
                line.erase(0, line.find("*/") + 2);
            }

        }
          // Output the recognized constants to console and file
    for (const auto& constant : constants)
    {
        cout << "Constant:"<< constant << endl;   // Print to console
        output <<"Constant:" << constant << endl; // Write to file
    }

    // Output the recognized identifiers to console and file

    for (const auto& identifier : identifiers)
    {
        if (!isKeyword(identifier)) {
            cout << "Identifier:" << identifier << endl;   // Print to
console
            output << "Identifier:" << identifier << endl; // Write to
file
        }
    }

        file.close();
        output.close();
    }



    return 0;}
```

Explanation:

- ✓ The code starts by including necessary header files (<iostream>, <fstream>, <string>, <algorithm>, <vector>, <unordered_set>) and using the std namespace.

- ✓ The isIdentifier function is defined to check if a given string word is a valid identifier. It checks if the word is empty or starts with a digit, and then

iterates through each character, checking if it is alphabetic, numeric, or an underscore.

- ✓ The isKeyword function is defined to check if a given string word is a keyword. It uses an unordered_set called keywords to store the predefined list of keywords. It checks if the word exists in the keywords set using the count function.

- ✓ The main function begins.

- ✓ It declares several variables, including line to store each line of the input file, and three vectors: constants, identifiers, and keywords.

- ✓ It opens the input file named "Input.txt" using an ifstream object called file, and opens an output file named "output.txt" in append mode using an ofstream object called output.

- ✓ It checks if the input file is successfully opened using the is_open function.

- ✓ It enters a while loop that reads each line from the input file using the getline function, and assigns it to the line variable.

- ✓ If isComment is false (i.e., not inside a comment block), it writes the current line to the output file using the output object.

- ✓ It removes all white spaces from the line by using the erase and remove functions from the <algorithm> header.

- ✓ It then iterates through the keywords vector and checks if each keyword exists in the line using the find function. If found, it writes "Keyword: <keyword>" to the output file.

- ✓ After recognizing the keywords, it proceeds to recognize constants. It iterates through each character of the line and checks if the character is a digit. If it is, it starts building a constant by appending subsequent digits or a dot (for floating-point constants) until a non-digit character is encountered. The constant is then added to the constants vector.

- ✓ Next, it recognizes identifiers. It iterates through each character of the line and checks if the character is alphabetic, an underscore, or alphanumeric. If it is, it starts building an identifier by appending subsequent valid characters until a non-valid character is encountered. The identifier is then added to the identifiers vector.

- ✓ After identifying constants and identifiers, it removes comments from the line. It first removes white spaces and then checks if the line contains a "//" or "/*" comment marker using the find function. If found, it erases the corresponding comment section.

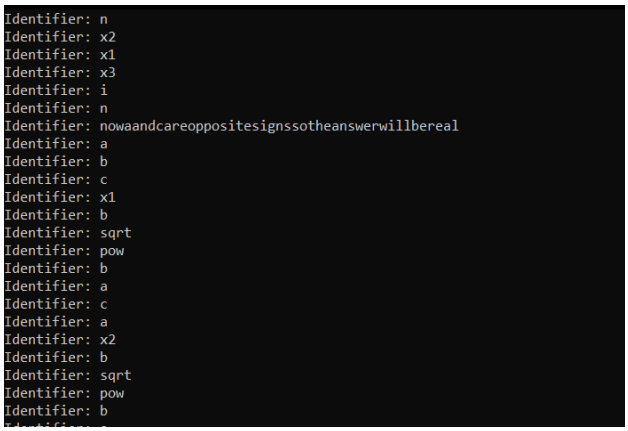
- ✓ After processing each line, it exits the while loop.

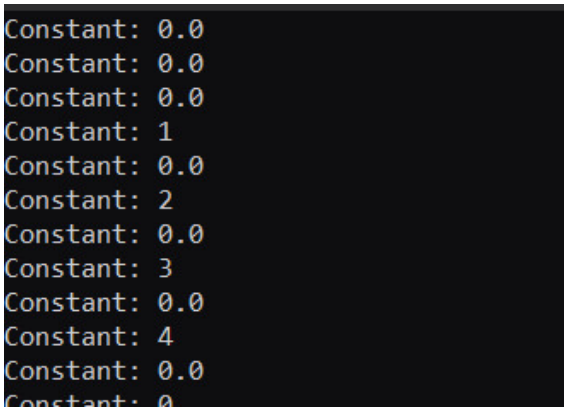
- ✓ It outputs the recognized constants to the console and the output file using a for loop. It prints "Constant: <constant>" to the console and writes the same to the output file.

- ✓ It outputs the recognized identifiers to the console and the output file using another for loop. It checks if each identifier is not a keyword using the isKeyword function and, if true, it prints "Identifier: <identifier>" to the console and writes the same to the output file.

- ✓ Finally, it closes the input and output files.

Output:





Output File Data:

```
#include <iostream>
#include <cmath>
using namespace std;

void one(){
Keyword: do

    float a = 0.0;
Keyword: float
    float b = 0.0
Keyword: float
    float c = 0.0;
Keyword: float
    float x1 = 0.0;
Keyword: float
    float x2 = 0.0;
Keyword: float
    float x3 = 0.0;
Keyword: float
    float x4 = 0.0;
Keyword: float
    cout << "Enter the coefficients a , b , c for equation in the form ax^ + bx + c = 0:\n";
```

```
Keyword: int
Keyword: cout
Keyword: for
    cout << "Enter value for a:\n";
Keyword: cout
Keyword: for
    cin >>  a;
Keyword: cin
    cout << "Enter value for b:\n";
Keyword: cout
Keyword: for
    cin >>  b;
Keyword: cin
    cout << "Enter value for c:\n";
Keyword: cout
Keyword: for
    cin >>  c;
Keyword: cin
Keyword: if
        if(a == 0 && b == 0 && c == 0){
Keyword: if
                x1 = 0;
                x2 = 0;
        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 = " << x1 << " , " << "x2 = " << x2 << "\n";
    }

Keyword: if
    if(a == 0 && b == 0 && c != 0){
Keyword: if
        c = c;
        cout << "There are no roots" "\n" << "c = " << c << "\n";
Keyword: cout
    }
Keyword: if
    if(a == 0 && b != 0 && c !=0){
Keyword: if
                cout << "The values entered do not make a quadratic expression" "\n"<< "x = "
<< -c/b << "\n";
Keyword: cout
Keyword: do
    }
Keyword: if
    if(a == 0 && b != 0 && c == 0){
Keyword: if
        x1 = 0;
        x2 = 0;
        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 = " << x1 << " , " << "x2 = " << x2 << "\n";
    }
Keyword: if
    if(a != 0 && b == 0 && c == 0){
Keyword: if
        x1 = 0;
        x2 = 0;
        cout << "The values entered result in ax^= 0; so both roots are 0" "\n"
Keyword: cout
        << "x1 = " << x1 << " , " << "x2 = " << x2 << "\n";
    }
    if(a != 0 && b != 0 && c == 0){
Keyword: if
        x1 = 0;
        x2 = -b/a;
        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 = " << x1 << " , " << "x2 = " << x2 << "\n";
    }

    if(a < 0 && b == 0 && c < 0){
Keyword: if

        x1 = -b/(2*a);
        x4 = (b*b)-(4*a*c);
        x4 = -x4;
        x2 =  sqrt(x4)/(2*a);
        x3 =  -sqrt(x4)/(2*a);


        cout << "The roots are not real numbers:" "\n"
Keyword: cout

                << "x1 =" << x1 << " + " << x2 << " * i " << "\n"
                << "x2 =" << x1 << " + " << x3 << " * i " << "\n";

    }
```

```
    if(a > 0 && b == 0 && c > 0){
Keyword: if

        x1 = -b/(2*a);
        x4 = (b*b)-(4*a*c);
        x4 = -x4;
        x2 =  sqrt(x4)/(2*a);
        x3 =  -sqrt(x4)/(2*a);


        cout << "The roots are not real numbers:" "\n"
Keyword: cout

               << "x1 =" << x1 << " + " << x2 << " * i " << "\n"
               << "x2 =" << x1 << " + " << x3 << " * i " << "\n";



    }

    if(a > 0 && b == 0 && c < 0){
Keyword: if

        x1 = (-b + (sqrt(pow(b,2)-(4*a*c))))/(2*a);
        x2 = (-b - (sqrt(pow(b,2)-(4*a*c))))/(2*a);

        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 =  "<< x1 << " , " << "x2 = "<< x2 << "\n";
    }
    if(a < 0 && b == 0 && c > 0){
Keyword: if

        x1 = (-b + (sqrt(pow(b,2)-(4*a*c))))/(2*a);
        x2 = (-b - (sqrt(pow(b,2)-(4*a*c))))/(2*a);

        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 =  "<< x1 << " , " << "x2 = "<< x2 << "\n";
    }

Keyword: if
Keyword: do
    if(a != 0 && b != 0 && c != 0 && (4*a*c) <= pow(b,2)){
Keyword: if


        x1 = (-b + (sqrt(pow(b,2)-(4*a*c))))/(2*a);
        x2 = (-b - (sqrt(pow(b,2)-(4*a*c))))/(2*a);

        cout << "The roots are:" "\n"
Keyword: cout
        << "x1 = "<< x1 << " , " << "x2 = " << x2 << "\n";
    }
Keyword: int
Keyword: for
    if(a != 0 && b != 0 && c != 0 && (4*a*c)> pow(b,2)){
Keyword: if


        x1 = -b/(2*a);
        x4 = (b*b)-(4*a*c);
        x4 = -x4;
        x2 =  sqrt(x4)/(2*a);
        x3 =  -sqrt(x4)/(2*a);


        cout << "The roots are not real numbers" "\n"
Keyword: cout

               << "x1 =" << x1 << " + " << x2 << " * i " << "\n"
               << "x2 =" << x1 << " + " << x3 << " * i " << "\n";




    }
    return;

}
Keyword: for
void two(){
    char c ;
Keyword: char
    cout << "Press c and then Enter to continue...." "\n";
Keyword: cout
```

```
    cin >> c;
Keyword: cin
    for(;;){
Keyword: for
        if ( c ){
Keyword: if
            break;
        }
       }

    cout << "Done" "\n";
Keyword: cout

}

int main(){
Keyword: int
    one();
    two();
    return 0;
}
Constant:0
Constant:0
Constant:0
Constant:4
Constant:2
Constant:1
Constant:2
Constant:4
Constant:4
Constant:4
Constant:4
Constant:2
Constant:4
Constant:2
Constant:3
Constant:4
Constant:2
Constant:1
Constant:1
Constant:2
Constant:2
Constant:1
Constant:3
Constant:0
Identifier:include
Identifier:iostream
Identifier:include
Identifier:cmath
Identifier:usingnamespacestd
Identifier:voidone
Identifier:floata
Identifier:herewedeclarethevariablesandusefloatbecausewe
Identifier:floatb
Identifier:aredealingwithsquareroots
Identifier:floatc
Identifier:floatx1
Identifier:floatx2
Identifier:floatx3
Identifier:floatx4
Identifier:thissectiongetsuserinputanddisplaysmessage
Identifier:Enterthecoefficientsa
Identifier:b
Identifier:cforequationintheformax
Identifier:bx
Identifier:c
Identifier:n
Identifier:Entervaluefora
Identifier:n
Identifier:a
Identifier:Entervalueforb
Identifier:n
Identifier:b
Identifier:Entervalueforc
Identifier:n
Identifier:c
Identifier:areallthecoefficients0
Identifier:ifsobothrootsare0
Identifier:a
Identifier:b
Identifier:c
Identifier:x1
Constant: 3
Constant: 4
Constant: 2
Constant: 1
Constant: 1
```

```
Constant: 2
Constant: 2
Constant: 1
Constant: 3
Constant: 0
Identifier: include
Identifier: iostream
Identifier: include
Identifier: cmath
Identifier: usingnamespacestd
Identifier: voidone
Identifier: floata
Identifier: herewedeclarethevariablesandusefloatbecausewe
Identifier: floatb
Identifier: aredealingwithsquareroots
Identifier: floatc
Identifier: floatx1
```