Compiler Construction
BSCS-6C
Lab 3
Group lab: done in group of 2
submitted by:

Abdul Ghaffar Kalhoro 194699
Ahmad Amjad Mughal 121672

1. Regular Expression:

Identifier: [A-Za-z_][_A-Za-z0-9]*

INT = [+-]?[0-9]$^+$

Float = `[+-]?([0-9]*[.])?[0-9]`$^+$
`Punctuation: [{}()[]=,.;:]`

`Relational Op: [==|!=|>|<|>=|<=]`

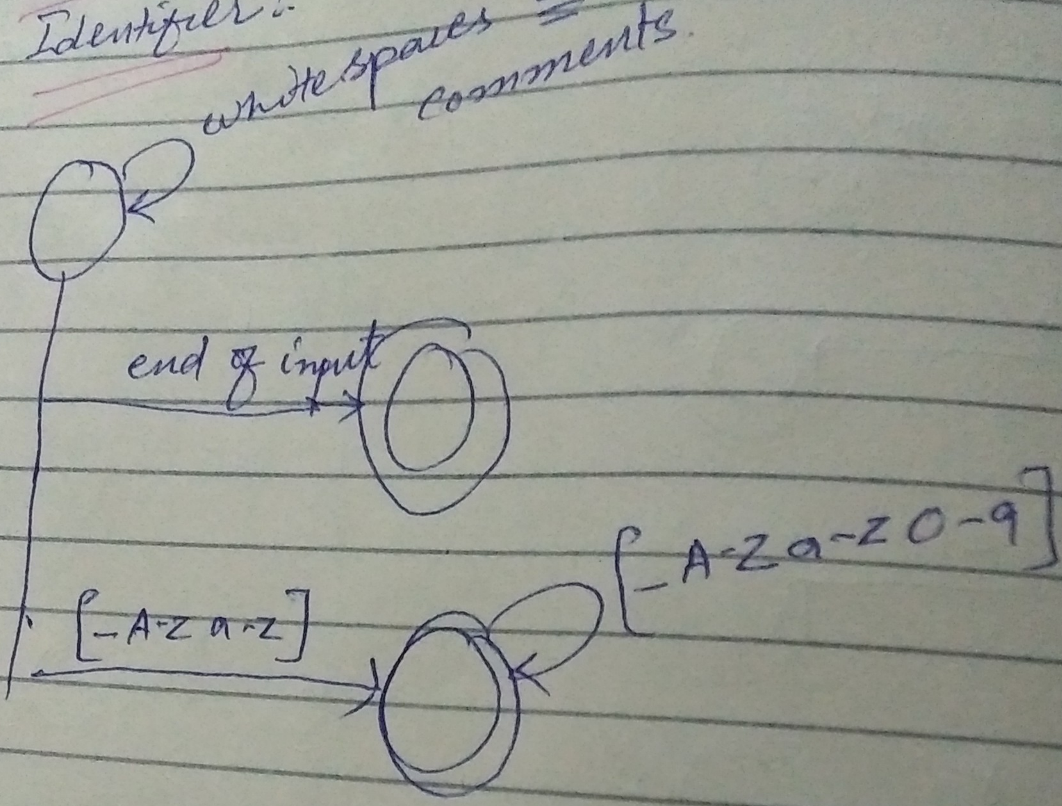
`Arithematic Operator: [+|-|*|/|%|++|--]`

```
Keywords = (break|case|char|const|continue|
default|double|else|enum|extern|float|for|
goto|if|int|long|return|short|static|struct,|
switch|void|while)
```

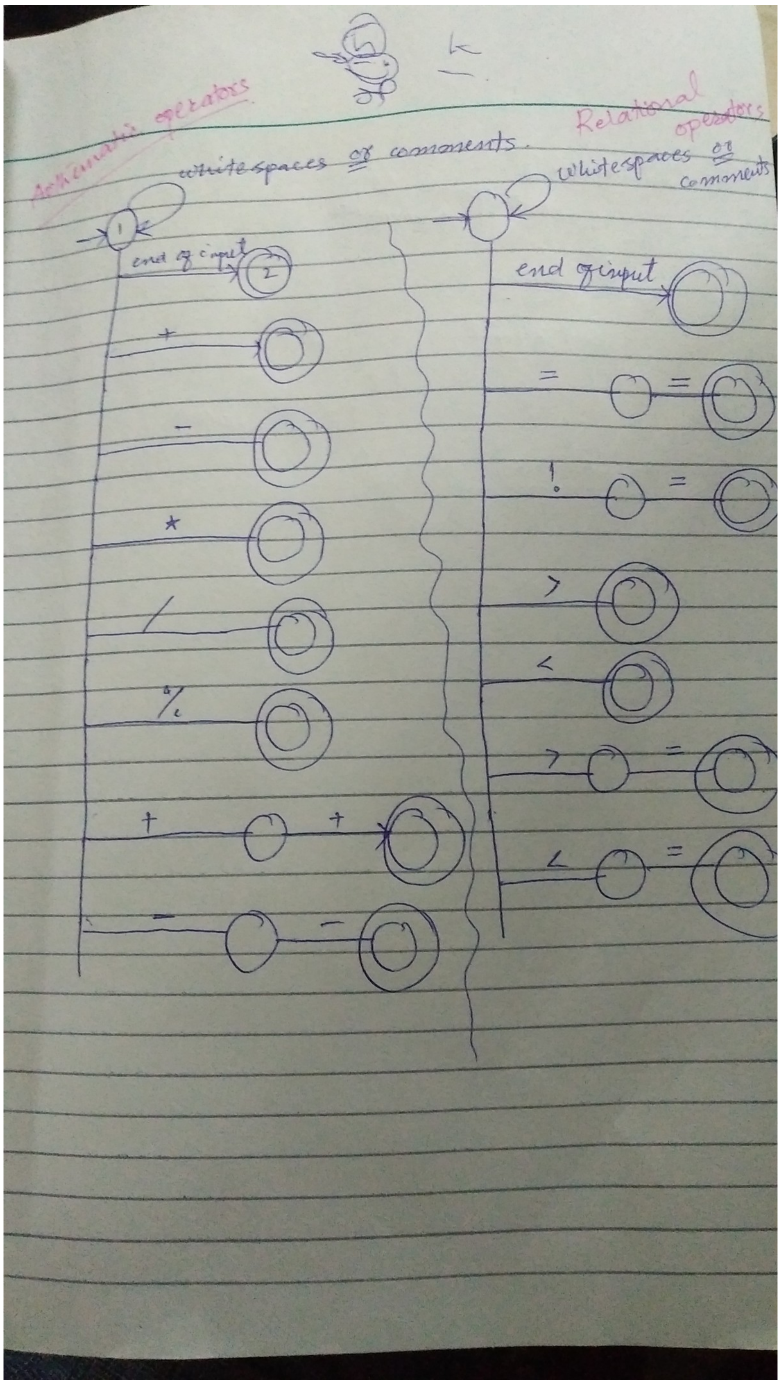
2. DFA:

Identifier:

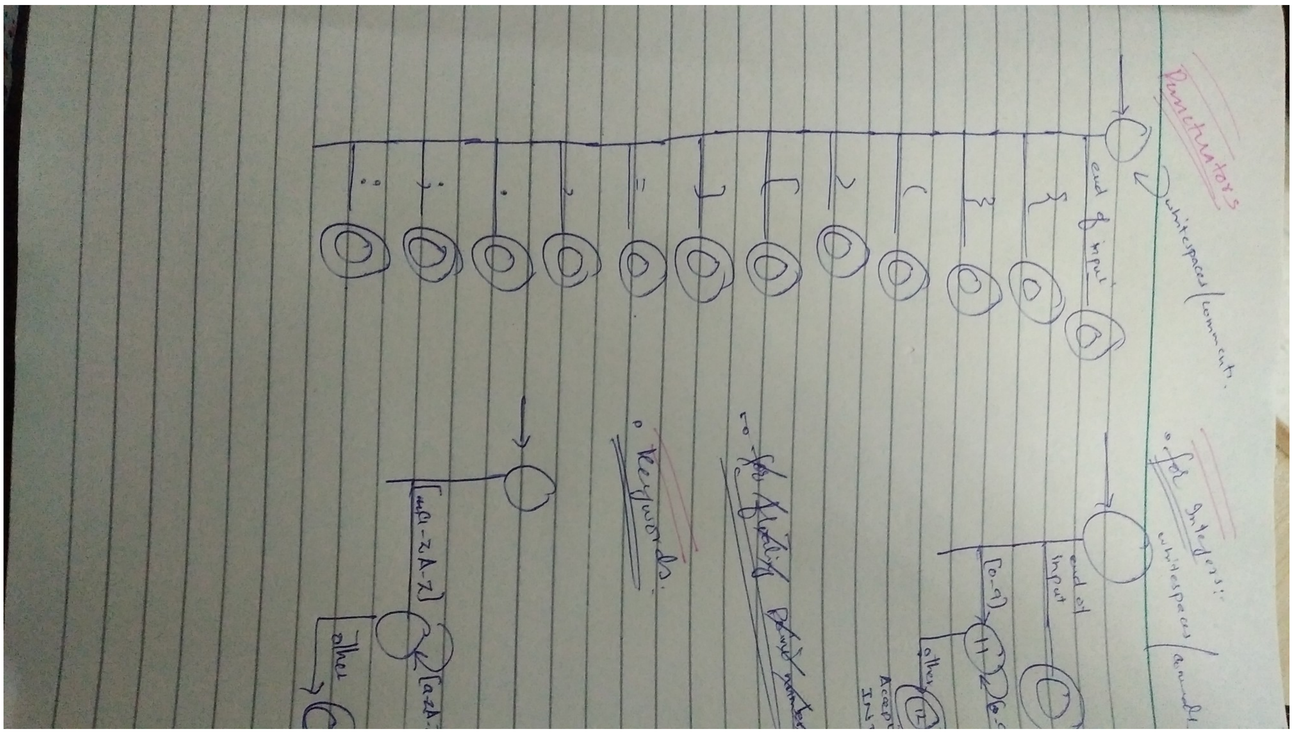
Identifier:



Arithematic operators and logical operators



Punctuations:



adfa

Source.cpp

```
//libraries packages for this program
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

//global variables
int arthm_total = 0;
int relational_total= 0;
static int verify_check = 0;
int punctuator_total = 0;


//function used for counting the number of int declarations used.
bool int_find(string line) {

        bool verify_check = false;
        int i;
        //loop for checking condition of 'int'
        for (i = 0; i < line.size(); i++) {

                if (line[i] == 'i' && line[i + 1] == 'n' && line[i + 2] == 't' && line[i+3] == ' ') {
                        verify_check = true;
                        i++;
                }

        }
        return verify_check;
}


//function used for counting the number of arthematic operations used.
bool arthm_checking(string line)
{
        bool verify_check = false;
        int i;

        for (i = 0; i < line.size(); i++) {

                if (line[i] == '+' && line[i+1]!='+') {
                        arthm_total++;
                        verify_check = true;
                        i++;
                }
                else if (line[i] == '-' && line[i+1]!='-') {
                        arthm_total++;
                        verify_check = true;
```

```
				i++;
			}
			else if (line[i] == '*' && line[i-1]!='/' && line[i+1] != '/') {
				arthm_total++;
				verify_check = true;
				i++;
			}
			else if (line[i] == '/' && line[i+1]!='*' && line[i+1]!='/' && line[i-1]!='*') {
				arthm_total++;
				verify_check = true;
				i++;
			}
			else if (line[i] == '%') {
				arthm_total++;
				verify_check = true;
				i++;
			}
			else if (line[i] == '+' && line[i+1] == '+') {
				arthm_total++;
				verify_check = true;
				i+=2;
			}
			else if (line[i] == '-' && line[i+1] == '-') {
				arthm_total++;
				verify_check = true;
				i+=2;
			}

		}
		return verify_check;

}


//function used for counting the number of 'float' used.
bool float_finding(string line)
{

		bool verify_check = false;
		int i;

		for (i = 0; i < line.size(); i++) {

			if (line[i] == 'f' && line[i + 1] == 'l' && line[i + 2] == 'o' && line[i+3] == 'a' &&
line[i+4]=='t' && line[i+5] == ' ') {
				verify_check = true;
				i++;
			}

		}
		return verify_check;
```

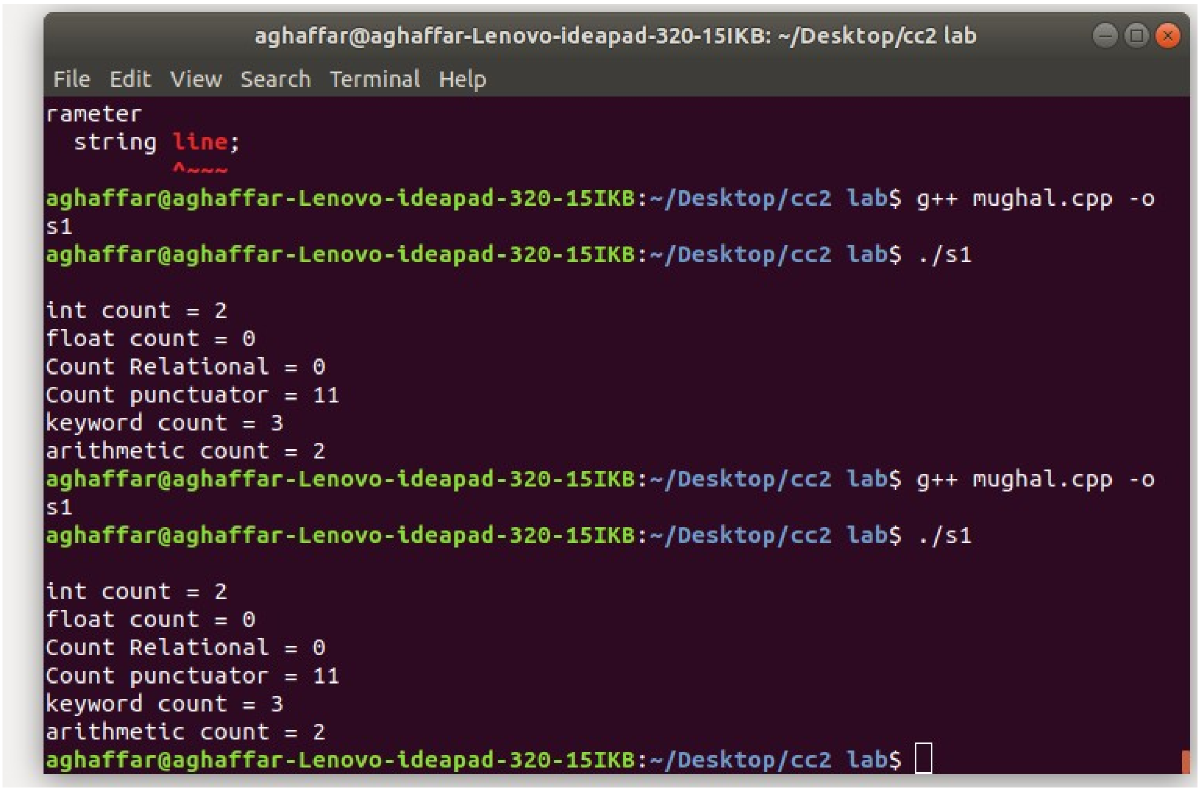
```
}
//function used for counting the number of relational operations used.
bool relational_find(string line)
{
	bool verify_check = false;
	int i;

	for (i = 0; i < line.size(); i++) {
		if (line[i] == '=' && line[i+1]=='=') {
			relational_total++;
			verify_check = true;
			i+=2;
		}
		else if (line[i] == '!' && line[i+1]=='=') {
		relational_total++;
			verify_check = true;
			i+=2;
		}
		else if (line[i] == '>') {
			relational_total++;
			verify_check = true;
			i++;
		}
		else if (line[i] == '<' && line[i+1]!='<') {
			relational_total++;
			verify_check = true;
			i++;
		}
		else if (line[i] == '>' && line[i+1]=='=') {
			relational_total++;
			verify_check = true;
			i+=2;
		}
		else if (line[i] == '<' && line[i+1]=='=') {
			relational_total++;
			verify_check = true;
			i+=2;
		}

	}
	return verify_check;

}

//function used for counting the number of keywords used.
bool keywords_find(string line)
{
	bool verify_check = false;
	int i;

	for (i = 0; i < line.size(); i++) {
			if (line[i] == 'b' && line[i + 1] == 'r' && line[i + 2] == 'e' && line[i+3] == 'a' &&
line[i+4]=='k'){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'c' && line[i + 1] == 'a' && line[i + 2] == 's' && line[i+3] == 'e'
&& line[i+4] == ' '){
				verify_check = true;
				i++;
			}
			else if(line[i] == 's' && line[i + 1] == 'w' && line[i + 2] == 'i' && line[i+3] == 't'
&& line[i+4] == 'c' && line[i+5] == 'h' && line[i+6] == ' '){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'c' && line[i + 1] == 'h' && line[i + 2] == 'a' && line[i+3] == 'r'
&& line[i+4] == ' '){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'c' && line[i + 1] == 'o' && line[i + 2] == 'n' && line[i+3] == 's'
&& line[i+4] == 't' && line[i+5] == ' ' ){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'c' && line[i + 1] == 'o' && line[i + 2] == 'n' && line[i+3] == 't'
&& line[i+4] == 'i' && line[i+5] == 'n' && line[i+6] == 'u' && line[i+7] == 'e' && line[i+8] == ';' )
{
				verify_check = true;
				i++;
			}
			else if(line[i] == 'd' && line[i + 1] == 'e' && line[i + 2] == 'f' && line[i+3] == 'a'
&& line[i+4] == 'u' && line[i+5] == 'l' && line[i+6] == 't' && line[i+7] == ':'){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'd' && line[i + 1] == 'o' && line[i + 2] == 'u' && line[i+3] == 'b'
&& line[i+4] == 'l' && line[i+5] == 'e' && line[i+6] == ' '){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'e' && line[i + 1] == 'l' && line[i + 2] == 's' && line[i+3] == 'e' &&
line[i+4] == ' '){
				verify_check = true;
				i++;
			}
			else if(line[i] == 'e' && line[i + 1] == 'n' && line[i + 2] == 'u' && line[i+3] == 'm'
&&  line[i+4] == ' '){
				verify_check = true;
				i++;
			}
```

```
		else if(line[i] == 'e' && line[i + 1] == 'x' && line[i + 2] == 't' && line[i+3] == 'e' 
&&  line[i+4] == 'r' &&  line[i+5] == 'n' && line[i+6] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'f' && line[i + 1] == 'l' && line[i + 2] == 'o' && line[i+3] == 'a' && 
line[i+4] == 't' &&  line[i+5] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'f' && line[i + 1] == 'o' && line[i + 2] == 'r' && (line[i+3] == ' ' || 
line[i+3] == '(')){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'g' && line[i + 1] == 'o' && line[i + 2] == 't' && line[i+3] == 'o' 
&&  line[i+4] == ';'){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'i' && line[i + 1] == 'f' && (line[i+2] == ' ' ||  line[i+2] == '(')){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'i' && line[i + 1] == 'n' && line[i+2] == 't' &&  line[i+3] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'l' && line[i + 1] == 'o' && line[i+2] == 'n' &&  line[i+3] == 'g' && 
line[i+3] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'r' && line[i + 1] == 'e' && line[i+2] == 't' &&  line[i+3] == 'u' && 
line[i+4] == 'r' && line[i+5] == 'n' && (line[i+6] == ' ' || line[i+6] == ';')){
			verify_check = true;
			i++;
		}
		else if(line[i] == 's' && line[i + 1] == 'h' && line[i+2] == 'o' &&  line[i+3] == 'r' && 
line[i+3] == 't' && line[i+4] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 's' && line[i + 1] == 't' && line[i+2] == 'a' &&  line[i+3] == 't' && 
line[i+4] == 'i' && line[i+5] == 'c' && line[i+6] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 's' && line[i + 1] == 't' && line[i+2] == 'r' &&  line[i+3] == 'u' && 
line[i+4] == 'c' && line[i+5] == 't' && line[i+6] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'v' && line[i + 1] == 'o' && line[i+2] == 'i' &&  line[i+3] == 'd' && 
line[i+4] == ' '){
			verify_check = true;
			i++;
		}
		else if(line[i] == 'w' && line[i + 1] == 'h' && line[i+2] == 'i' &&  line[i+3] == 'l' && 
line[i+4] == 'e' && (line[i+5] == '(' || line[i+5] == ' ')){
			verify_check = true;
			i++;
		}

	}
	return verify_check;

}
//function used for counting the number of punctuator used.
void punctuator_find(string line)
{
	int i;

	for (i = 0; i < line.size(); i++) {
		if (line[i] == '=') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == ',') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == '.') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == ';') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == ':') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == '(') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == '{') {
			punctuator_total++;
			i++;
		}
		else if (line[i] == '[') {
			punctuator_total++;
```

```
				i++;
			}

		}
		return;

}




//function used for cleaning single line comments used.
string comment_singleLine(string a)
{

		int str_length = a.length();
		int itr = 0;
		string line = "";

S1:
		if(itr == str_length)
			goto terminate;

		if(a[itr] == '/')
		{
			line += a[itr];
			goto S2;
		}

		else
		{
			line += a[itr];
			itr += 1;
			goto S1;

		}

S2:
		itr += 1;

		if(itr == str_length)
			goto terminate;

		if (!(a[itr] == '/'))
		{
			line += a[itr];
			itr += 1;
			goto S1;
		}

		if(a[itr] == '/')
		{
			line[line.length() - 1] = ' ';
			goto S3;
		}

S3:
		itr += 1;

		if(itr == str_length)
			goto terminate;

		if(!(a[itr] == '\n'))
		{

			line += ' ';
			goto S3;
		}
		else
		{

			line += a[itr];
			goto terminate;
		}

terminate:

		return line;

}
//function used for removing multiple commenting lines.
string comment_multiLine(string a)
{
		string line;

		for(int i=0; i < a.length(); i++)
		{
			if(a[i] == '/' && a[i+1] == '*')
			{
				verify_check = 1;
				line += ' ';
				continue;
//				return line;
			}

			if((a[i] == '*' && a[i+1] == '/') || (a[i] == '/' && a[i-1] == '*'))
			{
				line += ' ';
//				line[i+1] = ' ';
				verify_check = 0;
				continue;
//				return line;
			}
		if(verify_check == 1)
		{
			line += ' ';
			continue;
//			return line;
		}

		else
		{
			line += a[i];
		}
	}

	return line;

}

int main()
{
	string line;
	string new_line;

	fstream myfile ("test.cpp");
	ofstream new_file;
	new_file.open("cleanedCode.txt");
	int int_count = 0;


	int float_count = 0;
	int keywords_count = 0;
	if (myfile.is_open())
	{
	while ( getline (myfile,line) )
	{


		new_line = comment_singleLine(line);
		//cout <<endl<<" ...... "<< line<<endl;
		new_line = comment_multiLine(new_line);
		new_file << new_line;

			if(int_find(new_line))
			int_count++;

			if(float_finding(new_line))
			float_count++;

			if(keywords_find(new_line))
			keywords_count++;

			arthm_checking(new_line);
			relational_find(new_line);
```

```
                        punctuator_find(new_line);



                new_line = "";
            }
    myfile.close();
    new_file.close();
            }
            else
            {
            cout << "Unable to open file";
            return 0;
            }

            cout << "\nint count = " << int_count << endl;
            cout << "float count = " << float_count << endl;
            cout << "Count Relational = " << relational_total << endl;
            cout << "Count punctuator = " << punctuator_total << endl;
            cout << "keyword count = " << keywords_count << endl;
            cout << "arithmetic count = " << arthm_total << endl;

            return 0;
}
```

OUTPUT



aghaffar@aghaffar-Lenovo-ideapad-320-15IKB: ~/Desktop/cc2 lab
File Edit View Search Terminal Help

```
rameter
  string line;
         ^~~~
aghaffar@aghaffar-Lenovo-ideapad-320-15IKB:~/Desktop/cc2 lab$ g++ mughal.cpp -o
s1
aghaffar@aghaffar-Lenovo-ideapad-320-15IKB:~/Desktop/cc2 lab$ ./s1

int count = 2
float count = 0
Count Relational = 0
Count punctuator = 11
keyword count = 3
arithmetic count = 2
aghaffar@aghaffar-Lenovo-ideapad-320-15IKB:~/Desktop/cc2 lab$ g++ mughal.cpp -o
s1
aghaffar@aghaffar-Lenovo-ideapad-320-15IKB:~/Desktop/cc2 lab$ ./s1

int count = 2
float count = 0
Count Relational = 0
Count punctuator = 11
keyword count = 3
arithmetic count = 2
aghaffar@aghaffar-Lenovo-ideapad-320-15IKB:~/Desktop/cc2 lab$
```