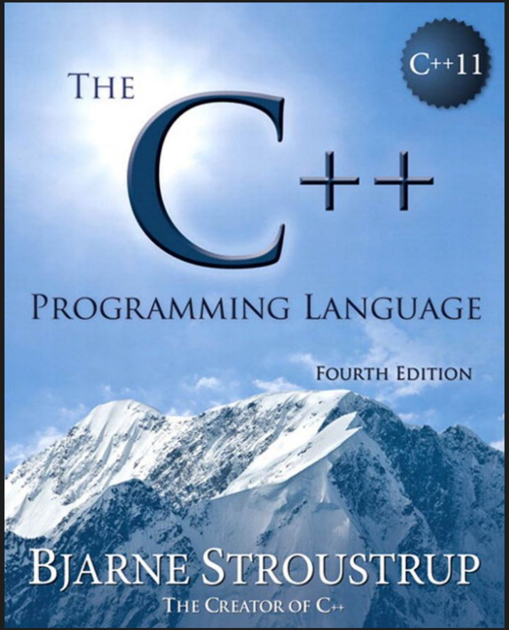
Desktop Calculator

diseño basado en capítulo 10.2, Desk Calculator

The C++ Programming Language Fourth Edition, Bjarne Stroustrup.



Partes principales

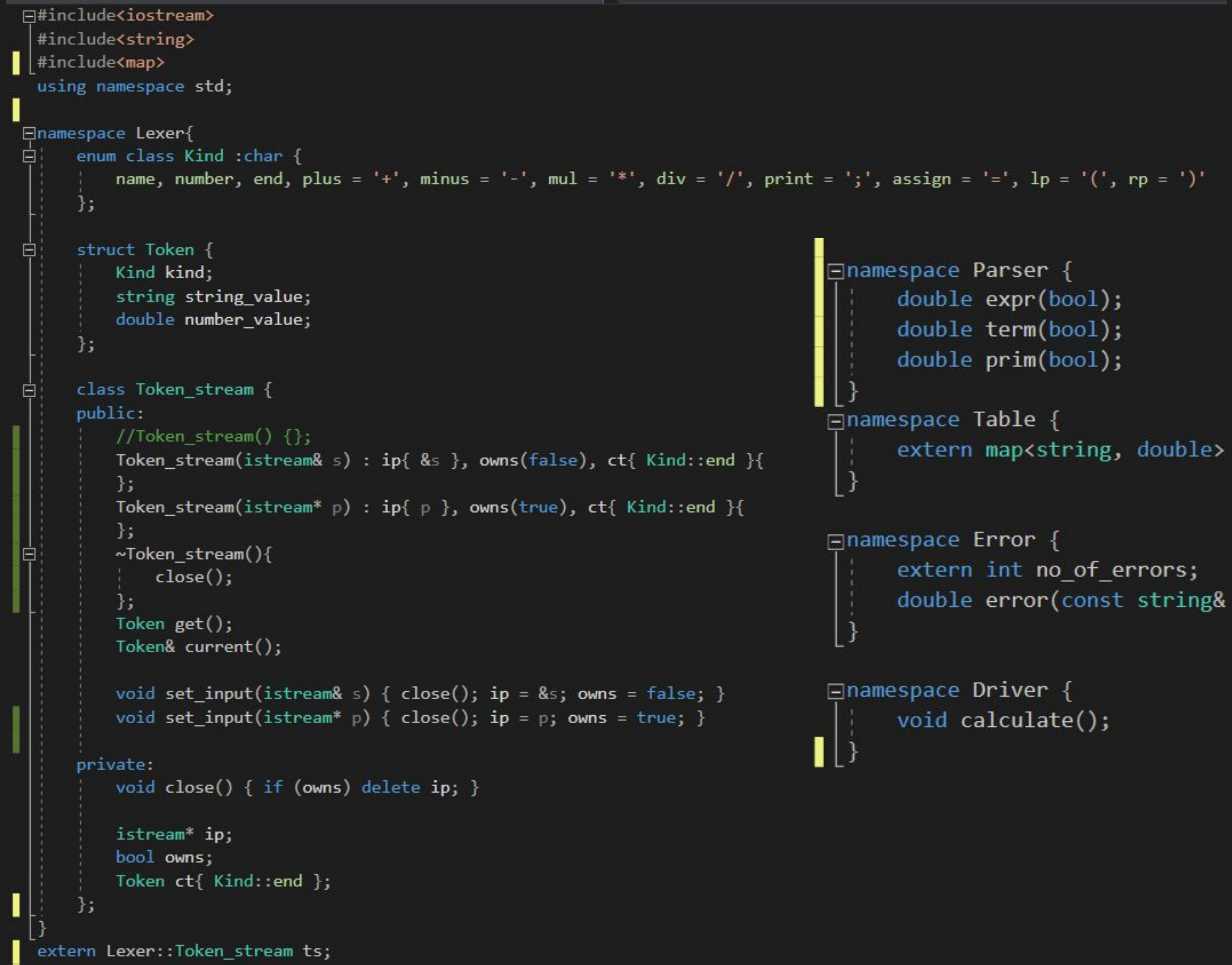
Parser: Detecta y realiza las operaciones.

Lexer: Lee el input y crea “tokens” que pueden ser ejecutados por el parser.

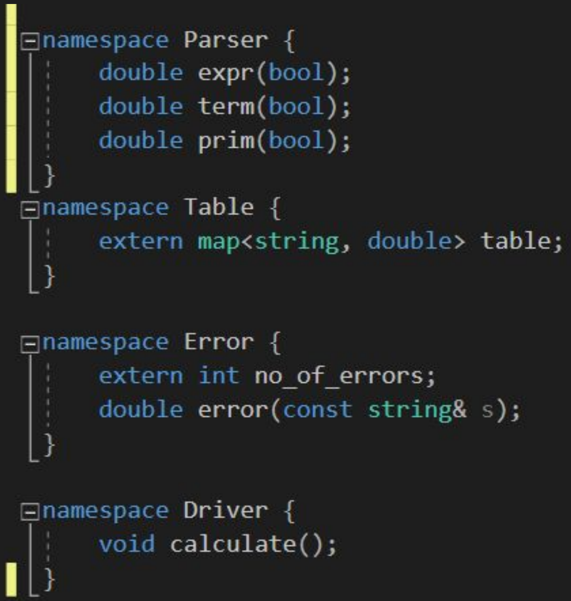
Symbol Table: Almacena las variables definidas.

Driver: Setup, detección de errores y cálculo.

Código

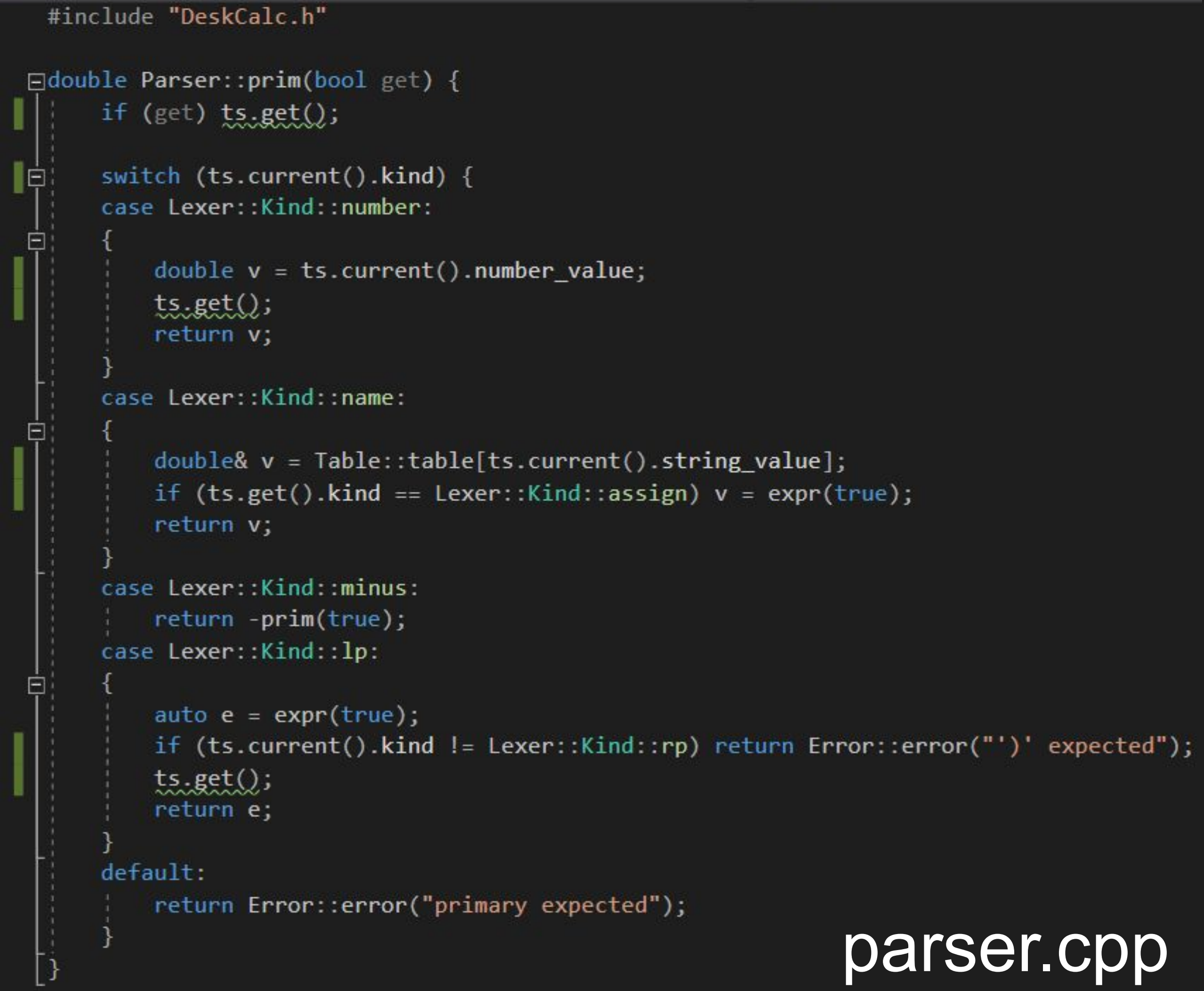


```cpp
#include<iostream>
#include<string>
#include<map>
using namespace std;

namespace Lexer{
    enum class Kind :char {
        name, number, end, plus = '+', minus = '-', mul = '*', div = '/', print = ';', assign = '=', lp = '(', rp = ')'
    };

    struct Token {
        Kind kind;
        string string_value;
        double number_value;
    };

    class Token_stream {
    public:
        //Token_stream() {};
        Token_stream(istream& s) : ip{ &s }, owns(false), ct{ Kind::end }{
        };
        Token_stream(istream* p) : ip{ p }, owns(true), ct{ Kind::end }{
        };
        ~Token_stream(){
            close();
        };
        Token get();
        Token& current();

        void set_input(istream& s) { close(); ip = &s; owns = false; }
        void set_input(istream* p) { close(); ip = p; owns = true; }

    private:
        void close() { if (owns) delete ip; }

        istream* ip;
        bool owns;
        Token ct{ Kind::end };
    };
}
extern Lexer::Token_stream ts;
```

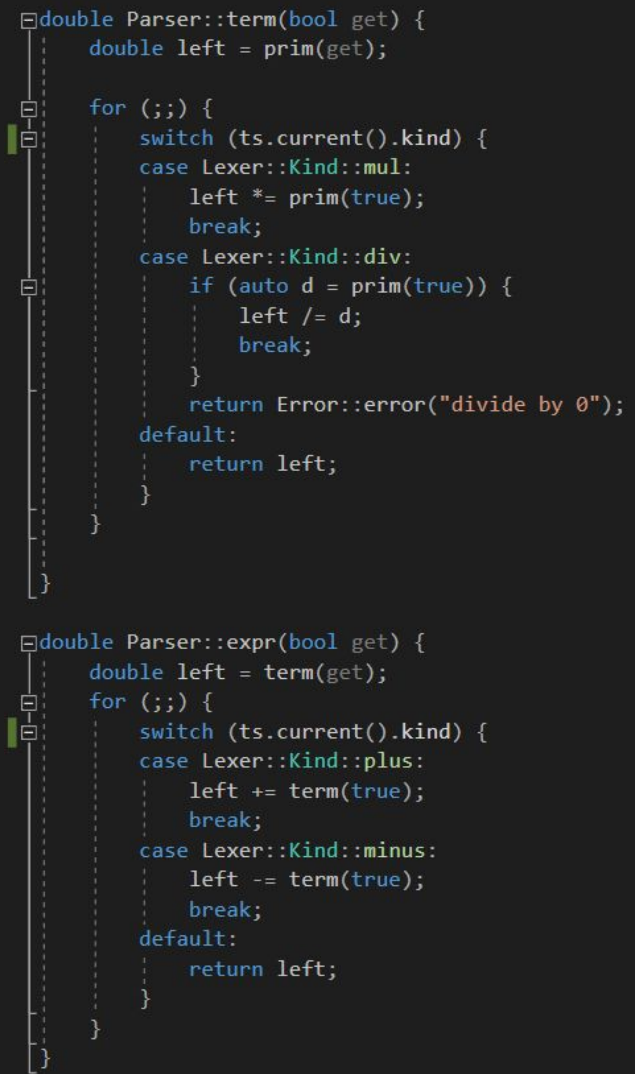


```cpp
namespace Parser {
    double expr(bool);
    double term(bool);
    double prim(bool);
}
namespace Table {
    extern map<string, double> table;
}

namespace Error {
    extern int no_of_errors;
    double error(const string& s);
}

namespace Driver {
    void calculate();
}
```

DeskCalc.h

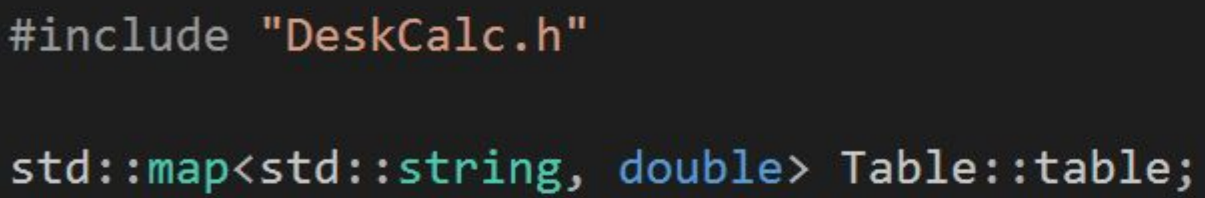


```cpp
#include "DeskCalc.h"

double Parser::prim(bool get) {
    if (get) ts.get();

    switch (ts.current().kind) {
    case Lexer::Kind::number:
    {
        double v = ts.current().number_value;
        ts.get();
        return v;
    }
    case Lexer::Kind::name:
    {
        double& v = Table::table[ts.current().string_value];
        if (ts.get().kind == Lexer::Kind::assign) v = expr(true);
        return v;
    }
    case Lexer::Kind::minus:
        return -prim(true);
    case Lexer::Kind::lp:
    {
        auto e = expr(true);
        if (ts.current().kind != Lexer::Kind::rp) return Error::error("')' expected");
        ts.get();
        return e;
    }
    default:
        return Error::error("primary expected");
    }
}
```



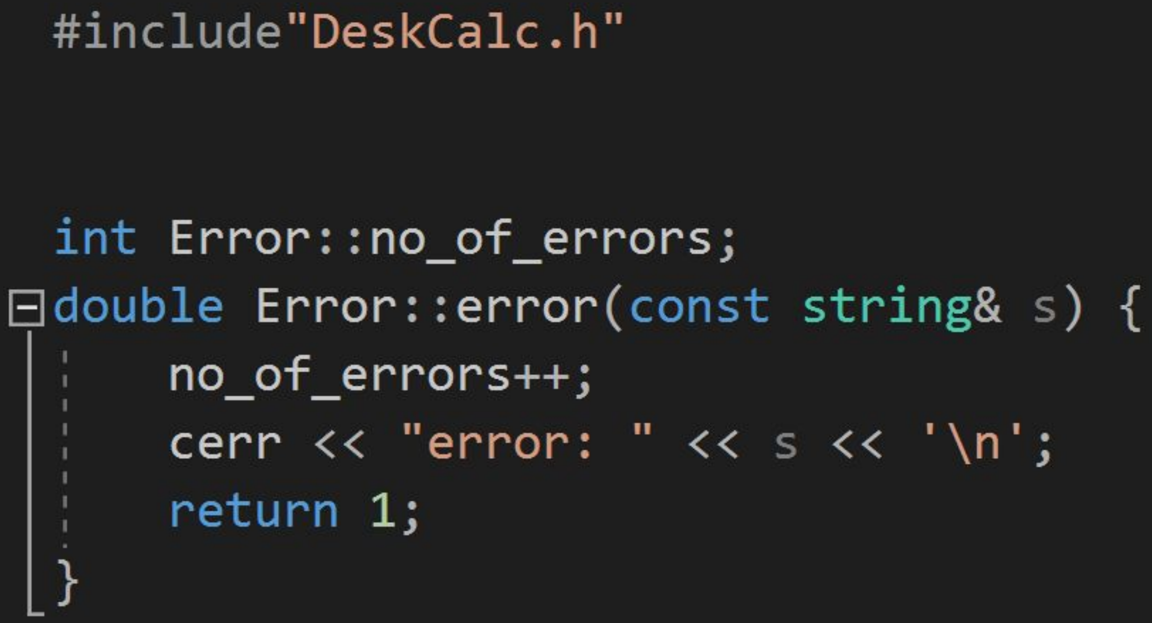
```cpp
double Parser::term(bool get) {
    double left = prim(get);

    for (;;) {
        switch (ts.current().kind) {
        case Lexer::Kind::mul:
            left *= prim(true);
            break;
        case Lexer::Kind::div:
            if (auto d = prim(true)) {
                left /= d;
                break;
            }
            return Error::error("divide by 0");
        default:
            return left;
        }
    }

}

double Parser::expr(bool get) {
    double left = term(get);
    for (;;) {
        switch (ts.current().kind) {
        case Lexer::Kind::plus:
            left += term(true);
            break;
        case Lexer::Kind::minus:
            left -= term(true);
            break;
        default:
            return left;
        }
    }
}
```

parser.cpp



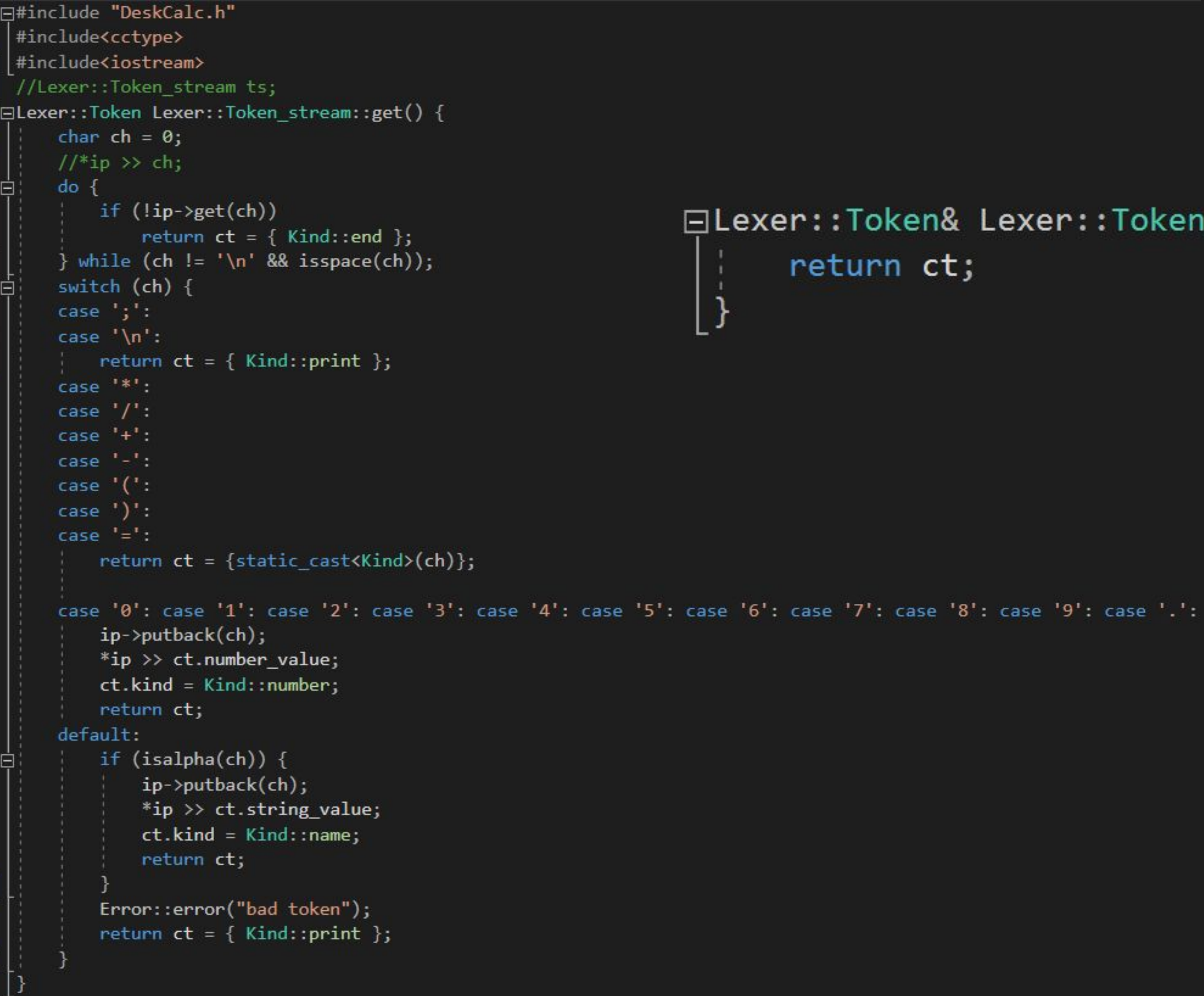
```cpp
#include "DeskCalc.h"

std::map<std::string, double> Table::table;
```

table.cpp

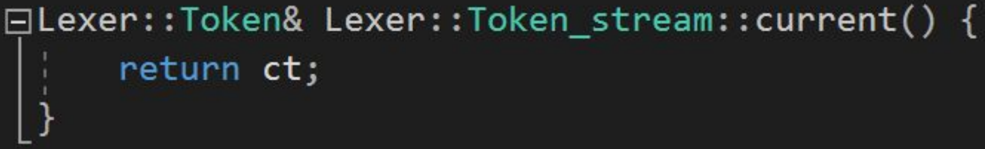


```cpp
#include"DeskCalc.h"


int Error::no_of_errors;
double Error::error(const string& s) {
    no_of_errors++;
    cerr << "error: " << s << '\n';
    return 1;
}
```

error.cpp

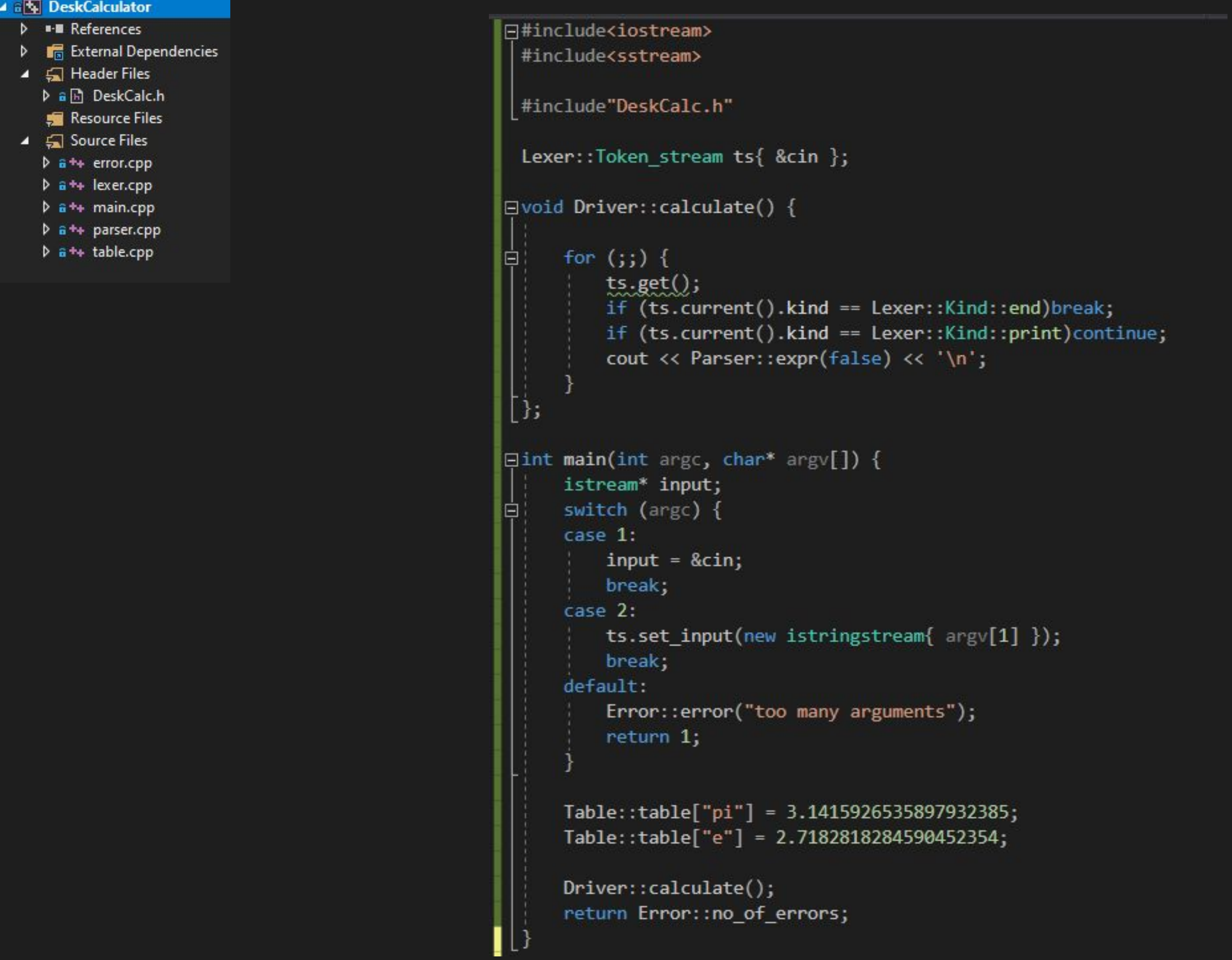


```
#include "DeskCalc.h"
#include<cctype>
#include<iostream>
//Lexer::Token_stream ts;
Lexer::Token Lexer::Token_stream::get() {
    char ch = 0;
    //*ip >> ch;
    do {
        if (!ip->get(ch))
            return ct = { Kind::end };
    } while (ch != '\n' && isspace(ch));
    switch (ch) {
    case ';':
    case '\n':
        return ct = { Kind::print };
    case '*':
    case '/':
    case '+':
    case '-':
    case '(':
    case ')':
    case '=':
        return ct = {static_cast<Kind>(ch)};

    case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9': case '.':
        ip->putback(ch);
        *ip >> ct.number_value;
        ct.kind = Kind::number;
        return ct;
    default:
        if (isalpha(ch)) {
            ip->putback(ch);
            *ip >> ct.string_value;
            ct.kind = Kind::name;
            return ct;
        }
        Error::error("bad token");
        return ct = { Kind::print };
    }
}
```



```
Lexer::Token& Lexer::Token_stream::current() {
    return ct;
}
```

lexer.cpp



```cpp
#include<iostream>
#include<sstream>

#include"DeskCalc.h"

Lexer::Token_stream ts{ &cin };

void Driver::calculate() {

	for (;;) {
		ts.get();
		if (ts.current().kind == Lexer::Kind::end)break;
		if (ts.current().kind == Lexer::Kind::print)continue;
		cout << Parser::expr(false) << '\n';
	}
};

int main(int argc, char* argv[]) {
	istream* input;
	switch (argc) {
	case 1:
		input = &cin;
		break;
	case 2:
		ts.set_input(new istringstream{ argv[1] });
		break;
	default:
		Error::error("too many arguments");
		return 1;
	}

	Table::table["pi"] = 3.1415926535897932385;
	Table::table["e"] = 2.7182818284590452354;

	Driver::calculate();
	return Error::no_of_errors;
}
```

main.cpp