```cpp
#include <iostream>
#include <cstring>
#include <string>
#include <fstream>

using namespace std;

class Token {
public:
    enum Type { LPAREN=0, RPAREN, PLUS, MINUS, MULT, DIV, POW, NUM, ERR, END };
    static const char* token_names[10];
    Type type;
    string lexema;
    Token(Type);
    Token(Type, char c);
    Token(Type, const string source);
};

const char* Token::token_names[10] = { "LPAREN" , "RPAREN", "PLUS", "MINUS", "MULT", "DIV", "POW", "NUM", "ERR", "END" };

Token::Token(Type type):type(type) { lexema = ""; }

Token::Token(Type type, char c):type(type) { lexema = c; }

Token::Token(Type type, const string source):type(type) {
    lexema = source;
}

std::ostream& operator << ( std::ostream& outs, const Token & tok )
{
    if (tok.lexema.empty())
        return outs << Token::token_names[tok.type];
    else
        return outs << Token::token_names[tok.type] << "(" << tok.lexema << ")";
}

std::ostream& operator << ( std::ostream& outs, const Token* tok ) {
    return outs << *tok;
}


class Scanner {
public:
    Scanner(const char* in_s);
    Token* nextToken();
    ~Scanner();
private:
    string input;
    int first, current;
    int state;
    char nextChar();
    void rollBack();
    void startLexema();
    string getLexema();
};

Scanner::Scanner(const char* s):input(s),first(0),current(0) { }

Token* Scanner::nextToken() {
    Token* token;
    char c;
    state = 0;
    startLexema();
    while (1) {
        switch (state) {
            case 0: c = nextChar();
                if (c == ' ') { startLexema(); state = 0; }
                else if (c == '\0') return new Token(Token::END);
                else if (c == '(') state = 1;
                else if (c == ')') state = 2;
                else if (c == '+') state = 3;
                else if (c == '-') state = 4;
                else if (c == '*') state = 5;
                else if (c == '/') state = 6;
                else if (c == '^') state = 7;
                else if (isdigit(c)) { state = 8; }
                else return new Token(Token::ERR, c);
                break;
            case 1: return new Token(Token::LPAREN);
            case 2: return new Token(Token::RPAREN);
            case 3: return new Token(Token::PLUS,c);
            case 4: return new Token(Token::MINUS,c);
            case 5: return new Token(Token::MULT,c);
```

Haga clic para agregar un punto de interrupción

Hemos Editado

Los valores de los arreglos tienen que ir en paralelo

Arreglo

Tipo de Token

Arreglo estático que contiene los tipos de Token

Guarda la secuencia de la cadena

Crea Token pero Sin lexema

Lo usamos para símbolos individuales como +

Crea Token + string completo

Constructores

Si lexema vacío Plus

Con lexema Plus(123)

Punteros Token

Construye Tokens

métodos

Destructor

atributos

Métodos

Devuelve el substring que va desde first hasta current

convierte la cadena en un std::string y la guarda en input, inicia en first y current

Pequeña máquina de estados que va reconociendo símbolos o números y retorna un Token*

lee un carácter

si c es vacío, se reinicia el state (next lexeme) y se queda en el estado cero hasta encontrar el primer caracter siguiente positivo

'\0' caracter nulo, token:: END (fin del input)

estados numerados del autómata

Si es símbolo, desde esos estados se retorna inmediatamente el Token correspondiente

c es un dígito

Si no entra en los casos anteriores, se crea Token de tipo ERR

```
            case 6: return new Token(Token::DIV,c);
            case 7: return new Token(Token::POW,c);
            case 8: c = nextChar();
                if (isdigit(c)) state = 8;
                else state = 9;
                break;
            case 9: rollBack();
                return new Token(Token::NUM, getLexema());
        }
    }

}

Scanner::~Scanner() { }

char Scanner::nextChar() {
    int c = input[current];
    if (c != '\0') current++;
    return c;
}

void Scanner::rollBack() { //Este rollback siempre se hace al final de un lexema el hace retroceder un caracter
    if (input[current] != '\0')
        current--;
}

void Scanner::startLexema() {
    first = current;
    return;
}

string Scanner::getLexema() {
    return input.substr(first,current-first);
}

int main(int argc, const char* argv[]) {

    if (argc != 2) {
        cout << "Incorrect number of arguments" << endl;
        exit(1);
    }

    Scanner scanner(argv[1]);
    Token* tk = scanner.nextToken();
    while (tk->type != Token::END) {
        cout << "next token " << tk << endl;
        delete tk;
        tk =  scanner.nextToken();
    }
    cout << "last token " << tk << endl;
    delete tk;
}
```

lo único que edité para el lab