

```

1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  #include <fstream>
5
6  using namespace std;
7
8  class Token {
9  public:
10     enum Type { LPAREN=0, RPAREN, PLUS, MINUS, MULT, DIV, POW, NUM, ERR, END };
11     static const char* token_names[10]; // Arreglo estatico que contiene los tipos de token
12     Type type; // Tipo de Token
13     string lexema; // Guarda la representacion de la cadena
14     Token(Type); // Solo token que sea lexema
15     Token(Type, char c); // lo usamos para simbolos individuales, como +
16     Token(Type, const string source); // Usa Token + string completo
17 };
18
19 // Arreglo
20 const char* Token::token_names[10] = { "LPAREN", "RPAREN", "PLUS", "MINUS", "MULT", "DIV", "POW", "NUM", "ERR", "END" };
21
22 Token::Token(Type type):type(type) { lexema = ""; }
23
24 Token::Token(Type type, char c):type(type) { lexema = c; }
25
26 Token::Token(Type type, const string source):type(type) {
27     lexema = source;
28 }
29
30 std::ostream& operator << ( std::ostream& outs, const Token & tok )
31 {
32     if (tok.lexema.empty())
33         return outs << Token::token_names[tok.type]; // Si lexema vacío Plus
34     else
35         return outs << Token::token_names[tok.type] << "(" << tok.lexema << ")"; // Con lexema Plus(123)
36 }
37
38 std::ostream& operator << ( std::ostream& outs, const Token* tok ) {
39     return outs << *tok; // Printea Token
40 }
41
42 class Scanner { // Construye Tokens
43 public:
44     Scanner(const char* in_s); // Constructor
45     Token* nextToken(); // Retorna token
46     ~Scanner(); // Destructor
47 private:
48     string input; // Contiene toda la cadena de entrada
49     int first, current; // Índices de la cadena
50     int state; // Estado actual
51     char nextChar(); // Retorna caracter
52     void rollBack(); // Retorna caracter
53     void startLexema(); // Inicia lexema
54     string getLexema(); // Retorna lexema
55 };
56
57 Scanner::Scanner(const char* s):input(s),first(0),current(0) { // Convierte la cadena en un string y la
58     // Guarda en input, inicia en first y current
59 }
60
61 Token* Scanner::nextToken() { // Retorna pointer a token que es un tokenizado simbolos o
62     // numero, y retorna un Token
63     Token* token;
64     char c;
65     state = 0;
66     startLexema();
67     while (1) {
68         switch (state) {
69             case 0: c = nextChar(); // Lee un caracter
70                 if (c == '\0') { startLexema(); state = 0; }
71                 else if (c == '(') return new Token(Token::END);
72                 else if (c == ')') state = 1;
73                 else if (c == '+') state = 2;
74                 else if (c == '-') state = 3;
75                 else if (c == '*') state = 4;
76                 else if (c == '/') state = 5;
77                 else if (c == '^') state = 6;
78                 else if (isdigit(c)) { state = 8; } // c es un dígito
79                 else return new Token(Token::ERR, c); // Si no entra en los casos anteriores se crea Token de tipo ERR.
80                 break;
81             case 1: return new Token(Token::LPAREN);
82             case 2: return new Token(Token::RPAREN);
83             case 3: return new Token(Token::PLUS, c);
84             case 4: return new Token(Token::MINUS, c);
85             case 5: return new Token(Token::MULT, c);
86             case 6: return new Token(Token::DIV, c);
87             case 7: return new Token(Token::POW, c);
88             case 8: // Numero
89                 while (isdigit(c)) c = nextChar();
90                 return new Token(Token::NUM, input.substr(first, current - first));
91             default: return new Token(Token::ERR, c);
92         }
93     }
94 }
95
96 Scanner::~Scanner() {
97     delete input;
98 }
99
100 char* Scanner::nextChar() {
101     if (current == first) return '\0';
102     return input[current++];
103 }
104
105 void Scanner::startLexema() {
106     first = current;
107 }
108
109 void Scanner::rollBack() {
110     current--;
111 }
112
113 string Scanner::getLexema() {
114     return input.substr(first, current - first);
115 }

```

```

84     case 6: return new Token(Token::DIV,c);
85     case 7: return new Token(Token::POW,c);
86     case 8: c = nextChar();
87     if (isdigit(c)) state = 8;
88     else state = 9;
89     break;
90     case 9: rollBack();
91     return new Token(Token::NUM, getLexema());
92 }
93 }
94 }
95 }
96
97 Scanner::~Scanner() { }
98
99 char Scanner::nextChar() {
100     int c = input[current];
101     if (c != '\0') current++;
102     return c;
103 }
104
105 void Scanner::rollBack() { //Este rollback siempre se hace al final de un lexema el hace retroceder un caracter
106     if (input[current] != '\0')
107         current--;
108 }
109
110 void Scanner::startLexema() {
111     first = current;
112     return;
113 }
114
115 string Scanner::getLexema() {
116     return input.substr(first,current-first);
117 }
118
119 int main(int argc, const char* argv[]) {
120     if (argc != 2) {
121         cout << "Incorrect number of arguments" << endl;
122         exit(1);
123     }
124     Scanner scanner(argv[1]);
125     Token* tk = scanner.nextToken();
126     while (tk->type != Token::END) {
127         cout << "next token " << tk << endl;
128         delete tk;
129         tk = scanner.nextToken();
130     }
131     cout << "last token " << tk << endl;
132     delete tk;
133 }
134
135
136
137
138

```

Handwritten notes:

- Constructor:** Se construye y retorna directamente el Token.
- nextChar():** Devuelve el siguiente caracter de la cadena y aumenta current++.
- rollBack():** Devuelve un caracter, hace rollback al final del lexema.
- startLexema():** Guarda la posición de inicio del lexema.
- getLexema():** Devuelve el lexema, más precisely (current-first).
- main():** Verifica que se pasen 2 argumentos.
- Scanner scanner(argv[1]):** Se crea objeto, usando el 1º argumento.
- Token* tk = scanner.nextToken();** Para obtener el 1º token.
- while (tk->type != Token::END):** Mientras Token no sea None::END.
- cout << "next token " << tk << endl;** Se imprime.
- delete tk;** Liberamos memoria del token.
- tk = scanner.nextToken();** Obtenemos el siguiente token.
- cout << "last token " << tk << endl;** Token tipo END, se imprime.
- delete tk;** Liberamos memoria del token.

lo único que editó para el lab