

lab5/L5_lab5plantilla.cpp

```
1  #include <sstream>
2  #include <iostream>
3  #include <stdlib.h>
4  #include <cstring>
5
6  using namespace std;
7
8
9  class Token {
10 public:
11     enum Type {PLUS, MINUS, MUL, DIV, NUM, ERR, PD, PI, END};
12     Type type;
13     string text;
14     Token(Type);
15     Token(Type, char c);
16     Token(Type, const string& source, int first, int last);
17 };
18
19 class Scanner {
20 private:
21     string input;
22     int first, current;
23 public:
24     Scanner(const char* in_s);
25     Token* nextToken();
26     ~Scanner();
27 };
28
29
30 enum BinaryOp { PLUS, MINUS, MUL, DIV };
31
32 class Exp {
33 public:
34     virtual void print() = 0;
35     virtual int eval() = 0;
36     virtual ~Exp() = 0;
37     static char binopToChar(BinaryOp op);
38 };
39
40
41 class BinaryExp : public Exp {
42 public:
43     Exp *left, *right;
44     BinaryOp op;
45     BinaryExp(Exp* l, Exp* r, BinaryOp op);
46     void print();
47     int eval();
48     ~BinaryExp();
49 };
50
51 class NumberExp : public Exp {
```

```

52 public:
53     int value;
54     NumberExp(int v);
55     void print();
56     int eval();
57     ~NumberExp();
58 };
59
60 class Parser {
61 private:
62     Scanner* scanner;
63     Token *current, *previous;
64     bool match(Token::Type ttype);
65     bool check(Token::Type ttype);
66     bool advance();
67     bool isAtEnd();
68     Exp* parseExpression();
69     Exp* parseTerm();
70     Exp* parseFactor();
71     bool tokenToOp(Token* tk, BinaryOp& op);
72 public:
73     Parser(Scanner* scanner);
74     Exp* parse();
75 };
76
77
78 Token::Token(Type type):type(type) { text = ""; }
79
80 Token::Token(Type type, char c):type(type) { text = c; }
81
82 Token::Token(Type type, const string& source, int first, int last):type(type) {
83     text = source.substr(first,last);
84 }
85
86 std::ostream& operator << ( std::ostream& outs, const Token & tok )
87 {
88     if (tok.text.empty())
89         return outs << tok.type;
90     else
91         return outs << "TOK" << "(" << tok.text << ")";
92 }
93
94 std::ostream& operator << ( std::ostream& outs, const Token* tok ) {
95     return outs << *tok;
96 }
97
98 // SCANNER //
99
100 Scanner::Scanner(const char* s):input(s),first(0), current(0) { }
101
102 Token* Scanner::nextToken() {
103     Token* token;
104     while (input[current]==' ') current++;

```

```

105     if (input[current] == '\\0') return new Token(Token::END);
106     char c = input[current];
107     first = current;
108     if (isdigit(c)) {
109         current++;
110         while (isdigit(input[current]))
111             current++;
112         token = new Token(Token::NUM, input, first, current - first);
113     } else if (strchr("+-* /()", c)) {
114         switch(c) {
115             case '+': token = new Token(Token::PLUS, c); break;
116             case '-': token = new Token(Token::MINUS, c); break;
117             case '*': token = new Token(Token::MUL, c); break;
118             case '/': token = new Token(Token::DIV, c); break;
119             case '(': token = new Token(Token::PI, c); break;
120             case ')': token = new Token(Token::PD, c); break;
121             default: cout << "No deberia llegar aca" << endl;
122         }
123         current++;
124     } else {
125         token = new Token(Token::ERR, c);
126         current++;
127     }
128     return token;
129 }
130
131 Scanner::~Scanner() { }
132
133 // PARSER //
134
135 bool Parser::match(Token::Type ttype) {
136     if (check(ttype)) {
137         advance();
138         return true;
139     }
140     return false;
141 }
142
143 bool Parser::check(Token::Type ttype) {
144     if (isAtEnd()) return false;
145     return current->type == ttype;
146 }
147
148 bool Parser::advance() {
149     if (!isAtEnd()) {
150         Token* temp = current;
151         if (previous) delete previous;
152         current = scanner->nextToken();
153         previous = temp;
154         if (check(Token::ERR)) {
155             cout << "Parse error, unrecognised character: " << current->text <<
endl;
156             exit(0);

```

```

157     }
158     return true;
159 }
160 return false;
161 }
162
163 bool Parser::isAtEnd() {
164     return (current->type == Token::END);
165 }
166
167 Parser::Parser(Scanner* sc):scanner(sc) {
168     previous = current = NULL;
169     return;
170 };
171
172 Exp* Parser::parse() {
173     current = scanner->nextToken();
174     if (check(Token::ERR)) {
175         cout << "Error en scanner - caracter invalido" << endl;
176         exit(0);
177     }
178     Exp* exp = parseExpression();
179     if (current) delete current;
180     return exp;
181 }
182
183 Exp* Parser::parseExpression() {
184
185     Exp* left = parseTerm();
186
187     while (match(Token::PLUS) || match(Token::MINUS)) {
188         BinaryOp op;
189         if (previous->type == Token::PLUS){
190             op = PLUS;
191         }
192         else if (previous->type == Token::MINUS){
193             op = MINUS;
194         }
195         Exp* right = parseTerm();
196         left = new BinaryExp(left, right, op);
197     }
198
199     return left;
200 }
201
202 Exp* Parser::parseTerm() {
203
204     Exp* left = parseFactor();
205
206     while (match(Token::MUL) || match(Token::DIV)) {
207         BinaryOp op;
208         if (previous->type == Token::MUL){
209             op = MUL;

```

```

210     }
211     else if (previous->type == Token::DIV){
212         op = DIV;
213     }
214     Exp* right = parseFactor();
215     left = new BinaryExp(left, right, op);
216 }
217 return left;
218 }
219
220 Exp* Parser::parseFactor() {
221     Exp* e;
222     if (match(Token::NUM)) {
223         return new NumberExp(stoi(previous->text));
224     }
225     else if (match(Token::PI)){
226         e = parseExpression();
227         if (!match(Token::PD)){
228             cout << "Falta parentesis derecho" << endl;
229         }
230         return e;
231     }
232     cout << "Error: se esperaba un número." << endl;
233     exit(0);
234 }
235
236 char Exp::binopToChar(BinaryOp op) {
237     char c=' ';
238     switch(op) {
239         case PLUS: c = '+'; break;
240         case MINUS: c = '-'; break;
241         case MUL: c = '*'; break;
242         case DIV: c = '/'; break;
243         default: c = '$';
244     }
245     return c;
246 }
247
248 // AST //
249
250
251 BinaryExp::BinaryExp(Exp* l, Exp* r, BinaryOp op):left(l),right(r),op(op) {}
252 NumberExp::NumberExp(int v):value(v) {}
253
254 Exp::~~Exp() {}
255 BinaryExp::~~BinaryExp() { delete left; delete right; }
256 NumberExp::~~NumberExp() { }
257
258
259 void BinaryExp::print() {
260     left->print();
261     char c = binopToChar(this->op);
262     cout << ' ' << c << ' ';

```

```

263     right->print();
264 }
265
266
267
268 void NumberExp::print() {
269     cout << value;
270 }
271
272 int BinaryExp::eval() {
273     int result;
274     int v1=left->eval();
275     int v2=right->eval();
276     switch(this->op) {
277         case PLUS: result = v1+v2; break;
278         case MINUS: result = v1-v2; break;
279         case MUL: result = v1*v2; break;
280         case DIV: result = v1/v2; break;
281         default:
282             cout << "Operador desconocido" << endl;
283             result = 0;
284     }
285     return result;
286 }
287
288 int NumberExp::eval() {
289     return value;
290 }
291
292
293 void test_scanner(Scanner * scanner) {
294     Token* current;
295     current = scanner->nextToken();
296     while (current->type != Token::END) {
297         if (current->type == Token::ERR) {
298             cout << "Error en scanner - caracter invalido: " << current->text <<
endl;
299             break;
300         } else
301             cout << current << endl;
302         current = scanner->nextToken();
303     }
304     exit(1);
305 }
306
307
308 int main(int argc, const char* argv[]) {
309
310     if (argc != 2) {
311         cout << "Incorrect number of arguments" << endl;
312         exit(1);
313     }
314

```

```
315 Scanner scanner(argv[1]);
316
317 //test_scanner(&scanner);
318
319 Parser parser(&scanner);
320
321 Exp *exp = parser.parse();
322
323 cout << "expr: ";
324 exp->print();
325 cout << endl;
326
327 cout << "eval: ";
328 cout << exp->eval() << endl;
329 // linux
330 // cd ".\Compiladores\"
331 // g++ L5_lab5plantilla.cpp -o lab5eP
332 // ./lab5eP "3+4+5"
333
334
335
336 // windows
337 // g++ L4_ejer1.cpp -o L4_ejer1.exe windows
338 // .\L4_ejer1.exe input.txt
339 delete exp;
340 }
```