Proyecto 3 - Desktop Calculator MRG200

Maor Roizman Gheiler

Partes del proyecto:

- **Parser:** Análisis sintáctico del texto ingresado.
- **Symbol table:** Almacenamiento de información.
- **Driver:** Ejecución de inicio, valores de salida y manejo de errores.
- **Input:** Ingreso de valores y análisis.

Parser

Observación: La existencia de dependencias exige la siguiente declaración.

```
double expr(bool get);
double term(bool get);
double prim(bool get);
```

Detalles:

- El Token_stream permite procesar la información y dividirla en ‘Tokens’.
- Se indica si se debe o no llamar a la función Token_stream::get() mediante el bool de la función ‘get()’.
- .get() y .current() permiten leer el siguiente y llamar a más reciente respectivamente.

Parser - expr(bool get)

```
double expr(bool get) { //add y sub
  double left = term(get);
  while(true) {
    switch (ts.current().kind) {
    case Kind::PLUS:
      left += term(true);
      break;

    case Kind::SUB:
      left -= term(true);
      break;
      default:
      return left;
    }
  }
}
```

Parser - term(bool get)

```
double term(bool get) { //mult y div y exp
  double left = prim(get);
  while(true) {
    switch (ts.current().kind) {
    case Kind::MULT:
      left *= prim(true);
      break;
    case Kind::DIV:
      if (auto d = prim(true)) {
        left /= d;
        break;
      }
      return error("divide by 0");
    case Kind::EXP:
      left = pow(left, prim(true));
      break;
    default:
      return left;
    }
  }
}
```

Parser - prim(bool get)

```
double prim(bool get) {
  if (get) ts.get(); // read next token
    switch (ts.current().kind) {
    case Kind::NUMBER: { // floating-point constant
      double v = ts.current().number_value;
      ts.get();
      return v;

    }
    case Kind::NAME: {
      double& v = Table::table[ts.current().string_value]; //
find the corresponding
      if (ts.get().kind == Kind::ASSIGN)
        v = expr(true); // '=' seen: assignment
        return v;
    }
    case Kind::SUB: // suma y minus
      return -prim(true);
    case Kind::LP: {
      auto e = expr(true);
      if (ts.current().kind != Kind::RP)
        return error("')' expected");
        ts.get(); // eat ')'
        return e;
    }
  default:
    return error("primary expected");
  }
}
```

Lexer

Observación: Token_stream es utilizada para encapsular los caracteres leídos y convertirlos en Tokens.

```
enum class Kind : char {
    NAME, NUMBER, END,
    PLUS = '+',
    SUB = '-',
    MULT = '*',
    DIV = '/',
    PRINT = ';',
    ASSIGN = '=',
    LP = '(',
    RP = ')',
    EXP = '^'
};
```

Detalles:

- Dentro de Lexer, get() permite leer un caracter para determinar el Token a componer y si es necesario utilizar otro.

Manejo de errores

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

Detalles:

- ‘cerr’ se utiliza para reportar los errores.

Symbol table

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

Detalles:

- Almacena como llave un string y como valor un double.

- Guarda valores ya determinados.

```
table["pi"] = 3.14159265358979323 85;
table["e"] = 2.71828182845904523 54;
```

Driver

```
void calculate(){
   while(true){
       ts.get();
       if (ts.current().kind == Lexer::Kind::END)
break;
       if (ts.current().kind ==
Lexer::Kind::PRINT) continue;
       std::cout << expr(false) << '\n';
   }
}
```

Detalles:

- Si se encuentra un error finaliza el bucle.
- El ‘false’ que esta dentro de expr() le indica que NO requiere llamar a la función .get() para habilitar la lectura de un Token.

Main

```
int main(int argc, char* argv[]) {
  istream* input;
  switch (argc) {
    case 1: // read from standard input
      input=&cin;
      break;
    case 2: // read from argument string
      ts.set_input(new istringstream{argv[1]});
      break;
    default:
      error("Muchos argumentos");
      return 1;
  }
...
  calculate();


  return no_of_errors;
}
```

Detalles:

- argc y argv[] comunican los argumentos al programa. argc → cantidad de elementos. argv[] → arreglo con los elementos.

Conclusiones - Para finalizar

- La implementación de la calculadora permite aprovechar diversas funcionalidades del lenguaje c++.
- Las 4 partes cruciales que componen el programa son: El input, el Driver, el Symbol table y el Parser.
- El Token_stream permite procesar la información y dividirla en ‘Tokens’
- **‘cerr’** se utiliza para reportar los errores..