Proyecto 3

Desktop Calculator

Angélica Sánchez

Token.h

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

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

Un token es un {kind-of-token,value} pair.

Ejemplo:

{number,123.45} en donde 123.45 es convertido en un valor flotante.

Al leer un string en el input es guardado aquí.

Al leer un numero (int ,double) en el input lo guarda aquí.

Namespaces.h // Table

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

Cuando la tabla está indexada por un string, el valor resultado es un string.

Ejemplo:

radio = 123.45

double& v = table[“radio];

v = 123.45

Namespaces.h // Error

```
namespace Error {
    int no_of_Errors;
    double error(const string& s);
}
```

Cuenta los errores y escribe un mensaje de error.

Token_stream.cpp

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

Los errores suelen ocurrir a la mitad de la evaluación de una expresión por lo que se aborta completamente la evaluación o retorna un valor.

ccerr is an unbuffered output stream usually used to report errors

Namespaces.h // Driver

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

Maneja el cálculo que se está realizando en un preciso momento.

Token_stream.cpp

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

La tarea principal es leer las expresiones

Cuando ts.get() encuentra un input error o un end-of-file

Toma la responsabilidad de manejar las expresiones vacías en lugar de expr().

El argumento false le dice a expr() que no necesita llamar a ts.get() para leer el token en el cual se va a travajar.

Namespaces.h // Parser

```
namespace Parser {
    double term(bool);
    double expr(bool);
    double prim(bool);
}
```

Cada función del Parser tiene un argumento booleano, el cual indica si es que la función necesita llamar a Token_stream::get() para tener el siguiente token

Parser.cpp

```
double Parser::term(bool get)
{
    double left = prim(get);
    for (;;) {
        switch (ts.current().kind) {
            case Kind::mul:
                left *= prim( get: true);
                break;
            case Kind::exp:
                left *=prim( get: true);
                break;
            case Kind::div:
                if (auto d = prim( get: true)) {
                    left /= d;
                    break;
                }
                return Error::error( s: "divide by 0");
            default:
                return left;
        }
    }
}
```

Función para la multiplicación y división.

Un loop infinito, equivalente a while(true).

Se ejecuta mientras se encuentren * o /.

Parser.cpp

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

Función para la suma y resta.

Un loop infinito, equivalente a while(true).

Se ejecuta mientras se encuentren + o –.

Parser.cpp

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

Lee el siguiente token.

Cuando el Token es un número (integer, float) se aloja en number_value.

Cuando el Token es un nombre (integer, se aloja en string_value.

prim() siempre lee un token más del que usa para analizar lsu expresión primaria

Token_stream.h // public

Encapsula la lectura de los caracteres y su composición en Tokens.

Lee y retorna el siguiente token.

Lee y retorna el token más reciente.

```
class Token_stream {
public:
    explicit Token_stream(istream& s) : ip{&s}, owns{false} { }
    explicit Token_stream(istream* p) : ip{p}, owns{true} { }
    ~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; }
```

Token_stream.h // private

```
private:
    void close() { if (owns) delete ip; }
    istream* ip;
    bool owns;
    Token ct { .kind: Kind::end};
};

Token_stream ts ( &: cin);
```

Puntero a un input stream que inicializa el Token_stream y es de donde este mismo obtiene los caracteres.

Indica si es que el Token_stream posee al istream

Es el token actual.
Se le da el valor inicial de Kind::end por si es que current() no es utilizado de la manera correcta y no devuelva un valor que no estaba en el input stream.

Es el input del cin.

Token_stream.cpp

```
Token Token_stream::get() {
    char ch;
    do {
        if (!ip->get( &: ch)) return ct={ .kind: Kind::end};
    } while (ch!='\n' && isspace(ch));
    switch (ch) {
        case ';':
        case '\n':
            return ct = { .kind: 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); // put the first digit (or .) back into the input stream
            *ip >> ct.number_value; // read the number into ct
            ct.kind=Kind::number;
            return ct;
        default: // name, name =, or error
            if (isalpha(ch)) {
                ip->putback(ch);
                *ip>>ct.string_value; // put the first character back into the input stream
                ct.kind = Kind::name; // read the string into ct
                return ct;
            }
            Error::error( s: "bad token");
            return ct={ .kind: Kind::print};
    }
}
```

Salta los espacios excepto '\n', y es más rápido que probar cada caracter de espacio en blanco.

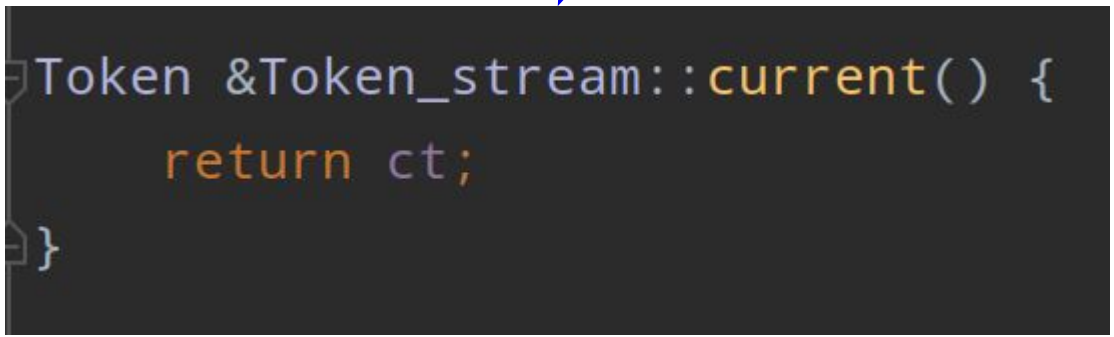
Lee un solo caracter del input stream *ip en ch.

El if statement es exitoso si es que no se puede leer ningún caracter en el input.

El loop se va a ejecutar mientras que ch no sea '\n' y sean espacios

Token_stream.cpp

Lee y retorna el token más reciente.



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

main.cpp

```
#include "token_stream.cpp"
#include "parser.cpp"
using namespace std;
using namespace Table;
using namespace Error;
using namespace Driver;
```

```
int main() {
    table["pi"] = 3.14159265358979323385;
    table["e"] = 2.71828182845904523354;
    calculate();
    return no_of_Errors;
    return 0;
}
```

Sus funciones son: reportar errores y hacer el setup

Devuelve cero si el programa termina normalmente y un valor distinto de cero si es lo contrario.

GRACIAS!

¿Alguna pregunta?