```cpp
#include <iostream>
#include <vector>

#include "../includes/AnalisadorSemantico.h"
#include "../../analisador-lexico/includes/LogErros.h"

AnalisadorSemantico::AnalisadorSemantico( std::pair<TabelaHash*,
NoArvoreSintatica*> _saidaAnalisadorSintatico )
{
    this->hash = *(_saidaAnalisadorSintatico.first);
    this->raiz = _saidaAnalisadorSintatico.second;

    this->nivelLexicoAtual = 0;

    this->analise( );
}

AnalisadorSemantico::~AnalisadorSemantico( )
{
}

void
AnalisadorSemantico::analise(  )
{
    std::vector<NoArvoreSintatica*>
    _filhos = this->raiz->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;


    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() == "<BLOCO>" )
        {
            this->bloco( (*_iteradorFilhos) );
        }
    }
}
```

```
void
AnalisadorSemantico::verificaDeclaracao( NoArvoreSintatica* _nada )
{

}

void
AnalisadorSemantico::verificaRedeclaracao( NoArvoreSintatica* _nada )
{

}

void
AnalisadorSemantico::bloco( NoArvoreSintatica* _bloco )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _bloco->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;

    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() ==
"<PARTE_DECLARACOES_SUB_ROTINAS>" )
        {
            this->declaracaoDeSubrotinas( (*_iteradorFilhos) );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<COMANDO_COMPOSTO>" )
        {
            this->comandoComposto( (*_iteradorFilhos) );
        }
    }
}

void
AnalisadorSemantico::declaracaoDeSubrotinas( NoArvoreSintatica*
```

```
_parteDeclaracoesSubrotinas )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _parteDeclaracoesSubrotinas->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;

    std::string
    _descricao;

    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos)
    {
        if( (*_iteradorFilhos)->getDescricao() ==
"<DECLARACAO_FUNCAO>" )
        {
            this->bloco( (*_iteradorFilhos)->getFilhos()[6] );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<DECLARACAO_PROCEDIMENTO>" )
        {
            if( (*_iteradorFilhos)->getFilhos()[2]->getDescricao()
== "<PARAMETROS_FORMAIS>" )
            {
                this->bloco( (*_iteradorFilhos)->getFilhos()[4] );
            }
            else
            {
                this->bloco( (*_iteradorFilhos)->getFilhos()[3] );
            }
        }
    }
}

void
AnalisadorSemantico::comandoComposto( NoArvoreSintatica*
_comandoComposto )
{
    std::vector<NoArvoreSintatica*>
```

```
    _filhos = _comandoComposto->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;

    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {

        if( (*_iteradorFilhos)->getDescricao() == "<COMANDO>" )
        {
            this->comando( *_iteradorFilhos );
        }
    }
}

void
AnalisadorSemantico::comando( NoArvoreSintatica* _comando )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _comando->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;

    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() ==
"<COMANDO_SEM_ROTULO>" )
        {
            this->comandoSemRotulo( *_iteradorFilhos );
        }
    }
}

void
AnalisadorSemantico::comandoSemRotulo( NoArvoreSintatica*
_comandoSemRotulo )
{
```

```
    std::vector<NoArvoreSintatica*>
    _filhos = _comandoSemRotulo->getFilhos( );

    if( _filhos[0]->getDescricao() == "<COMANDO_REPETITIVO>" )
    {
        this->comandoRepetitivo( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<COMANDO_CONDICIONAL>" )
    {
        this->comandoCondicional( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<COMANDO_COMPOSTO>" )
    {
        this->comandoComposto( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<COMANDO_LEITURA>" )
    {
        this->comandoLeitura( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<ATRIBUICAO>" )
    {
        this->atribuicao( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<CHAMADA_PROCEDIMENTO>" )
    {
        this->chamadaProcedimento( _filhos[0] );
    }
}

void
AnalisadorSemantico::comandoRepetitivo( NoArvoreSintatica*
_comandoRepetitivo )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _comandoRepetitivo->getFilhos( );

    std::string
    _resultadoExpressao;
```

```
    _resultadoExpressao = this->expressao( _filhos[1] );

    if( _resultadoExpressao != "boolean" )
    {
        LogErros::getInstancia().insereErro( _comandoRepetitivo-
>getLinha(), "Expressao deveria retornar integer." );
    }

    this->comandoSemRotulo( _filhos[3] );
}


void
AnalisadorSemantico::comandoCondicional( NoArvoreSintatica*
_comandoCondicional )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _comandoCondicional->getFilhos( );

    this->expressao( _filhos[1] );

    this->comandoSemRotulo( _filhos[3] );

    if( _filhos.size() > 4 )
    {
        this->comandoSemRotulo( _filhos[5] );
    }
}

void
AnalisadorSemantico::comandoLeitura( NoArvoreSintatica*
_comandoLeitura )
{
    std::vector<NoArvoreSintatica*>
    _filhosListaIdentificadores = _comandoLeitura->getFilhos( )[2]-
>getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;
```

```
    TabelaHash::iterator
    _resultadoBusca;

    unsigned int
    _contador;

    std::string
    _classificacao;

    std::string
    _descricao;


    for( _iteradorFilhos = _filhosListaIdentificadores.begin();
_iteradorFilhos != _filhosListaIdentificadores.end(); +
+_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao( ) ==
"<IDENTIFICADOR>" )
        {
            _descricao =  (*_iteradorFilhos)->getFilhos( )[0]-
>getDescricao( );

            /*      'this->nivelLexicoAtual' nunca podera ser menor
que 0 por definicao
             *      para evitar o estouro para cima de unsigned int
foi utilizada a comparação '!= 0'
             */
            for( _contador = 0; (this->nivelLexicoAtual-_contador)
+1 != 0; ++_contador )
            {
                if( this->hash[std::pair<const std::string, const
unsigned int>(_descricao, this->nivelLexicoAtual-_contador)] != this-
>hash.end() )
                {
                    _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual-
_contador)];

                    _classificacao = _resultadoBusca->second-
```

```
>getConteudo();

                    break;
                }
            }

            if( _resultadoBusca != this->hash.end() )
            {
                if( _classificacao == "variavel" )
                {
                    if( (*_resultadoBusca).second->getTipo() !=
"integer" )
                    {
                        LogErros::getInstancia().insereErro
( _comandoLeitura->getLinha(), "Parametro '" + _descricao + "' nao e
integer." );
                    }
                }
                else if( _classificacao == "parametrosFormais" )
                {
                    if( (*_resultadoBusca).second->getTipo() !=
"integer" )
                    {
                        LogErros::getInstancia().insereErro
( _comandoLeitura->getLinha(), "Parametro '" + _descricao + "' nao e
integer." );
                    }
                }
                else if( _classificacao == "procedimento|funcao" )
                {
                    if( (*_resultadoBusca).second->getTipo() !=
"integer" )
                    {
                        LogErros::getInstancia().insereErro
( _comandoLeitura->getLinha(), "Parametro '" + _descricao + "' nao e
integer." );
                    }
                }
            }
        }
```

```
    }
}

void
AnalisadorSemantico::atribuicao( NoArvoreSintatica* _atribuicao )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _atribuicao->getFilhos( );

    TabelaHash::iterator
    _resultadoBusca;

    unsigned int
    _contador;

    std::string
    _classificacao;

    std::string
    _descricao = _filhos[0]->getFilhos()[0]->getFilhos()[0]-
>getDescricao();

    std::string
    _resultadoExpressao;

    /*      'this->nivelLexicoAtual' nunca podera ser menor que 0
por definicao
     *      para evitar o estouro para cima de unsigned int foi
utilizada a comparação '!= 0'
     */
    for( _contador = 0; (this->nivelLexicoAtual-_contador) +1 != 0; +
+_contador )
    {
        if( this->hash[std::pair<const std::string, const unsigned
int>(_descricao, this->nivelLexicoAtual-_contador)] != this->hash.end
() )
        {
            _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual-
_contador)];
```

```
            _classificacao = _resultadoBusca->second->getConteudo();

            break;
        }
    }

    if( _resultadoBusca != this->hash.end() )
    {
        _resultadoExpressao = this->expressao( _filhos[2] );

        if( _classificacao == "variavel" )
        {
            if( _resultadoExpressao != (*_resultadoBusca).second-
>getTipo() )
            {
                LogErros::getInstancia().insereErro( _atribuicao-
>getLinha(), "Impossivel atribuir '" + _resultadoExpressao + "' a
uma variavel '" + (*_resultadoBusca).second->getTipo() + "'." );
            }
        }
        else if( _classificacao == "parametrosFormais" )
        {
            if( _resultadoExpressao != (*_resultadoBusca).second-
>getTipo() )
            {
                LogErros::getInstancia().insereErro( _atribuicao-
>getLinha(), "Impossivel atribuir '" + _resultadoExpressao + "' a
uma variavel '" + (*_resultadoBusca).second->getTipo() + "'." );
            }
        }
        else if( _classificacao == "procedimento|funcao" )
        {
            if( _resultadoExpressao != (*_resultadoBusca).second-
>getTipo() )
            {
                LogErros::getInstancia().insereErro( _atribuicao-
>getLinha(), "Impossivel atribuir '" + _resultadoExpressao + "' a
uma variavel '" + (*_resultadoBusca).second->getTipo() + "'." );
            }
```

```
        }
    }
}

std::string
AnalisadorSemantico::chamadaFuncao( NoArvoreSintatica*
_chamadaFuncao )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _chamadaFuncao->getFilhos( );

    std::vector<NoArvoreSintatica*>
    _listaExpressoes;

    std::vector<NoArvoreSintatica*>
    _verificacaoParametro;

    NoArvoreSintatica*
    _paiVerificacaoParametro;

    TabelaHash::iterator
    _resultadoBusca;

    TabelaHash::iterator
    _resultadoBuscaParametro;

    unsigned int
    _contador;

    std::string
    _identificadorVariavel;

    std::string
    _descricao = _filhos[0]->getFilhos( )[0]->getDescricao( );

    std::string
    _tipoParametro;

    std::string
    _quantidadeParametrosStr;
```

```
    unsigned int
    _quantidadeParametros;

    bool
    _encontrado = false;

    bool
    _erro = false;

    std::string
    _resultadoExpressao;

    std::string
    _bufferContador;


    if( this->hash[std::pair<const std::string, const unsigned int>
(_descricao, this->nivelLexicoAtual+1)] != this->hash.end() )
    {
        _resultadoBusca = this->hash[std::pair<const std::string,
const unsigned int>(_descricao, this->nivelLexicoAtual+1)];

        _encontrado = true;
    }
    /*      'this->nivelLexicoAtual' nunca podera ser menor que 0
por definicao
     *      para evitar o estouro para cima de unsigned int foi
utilizada a comparação '!= 0'
     */

    if( !_encontrado )
    {
        for( _contador = 0; (this->nivelLexicoAtual-_contador) +1 !=
0; ++_contador )
        {
            if( this->hash[std::pair<const std::string, const
unsigned int>(_descricao, this->nivelLexicoAtual-_contador)] != this-
>hash.end() )
            {
```

```
                _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual-
_contador)];

                _encontrado = true;
                break;
            }
        }
    }

    _quantidadeParametros = _resultadoBusca->second-
>procedureFunction->quantidadeParametros;

    if( _quantidadeParametros != 0 )
    {
        _listaExpressoes = _filhos[2]->getFilhos( );
    }

    if( _quantidadeParametros != ((_listaExpressoes.size()+1)/2) )
    {
        _quantidadeParametrosStr = _quantidadeParametros;
        LogErros::getInstancia().insereErro( _chamadaFuncao->getLinha
(), "Quantidade de parametros incorreta. Sao esperados [" +
_quantidadeParametrosStr + "] parametros" );
    }

    for( _contador = 0; _contador < _listaExpressoes.size();
_contador += 2 )
    {
        if( _resultadoBusca->second->procedureFunction->parametros
[_contador/2].first == true )
        {
            _bufferContador = (_contador+1);
            _paiVerificacaoParametro = _listaExpressoes[_contador];
            _verificacaoParametro = _listaExpressoes[_contador]-
>getFilhos( );

            while( _verificacaoParametro.size() < 2 )
            {
                _bufferContador = (_contador+1);
```

```
                if( _verificacaoParametro.size() == 0 )
                {
                    if( _paiVerificacaoParametro->getDescricao( ) ==
"<VARIAVEL>" )
                    {
                        break;
                    }
                    else
                    {
                        LogErros::getInstancia().insereErro
( _chamadaFuncao->getLinha(), "O " + _bufferContador + "deveria ser
uma variavel." );
                        _erro = true;
                    }
                }
                else if( _verificacaoParametro.size() == 1 )
                {
                    _paiVerificacaoParametro = _verificacaoParametro
[0];
                    _verificacaoParametro = _verificacaoParametro[0]-
>getFilhos( );
                }
                else
                {
                    LogErros::getInstancia().insereErro
( _chamadaFuncao->getLinha(), "O " + _bufferContador + "deveria ser
uma variavel." );
                    _erro = true;
                }
            }
            if( _verificacaoParametro.size() < 2 )
            {
                LogErros::getInstancia().insereErro( _chamadaFuncao-
>getLinha(), "O " + _bufferContador + "deveria ser uma variavel." );
                _erro = true;
            }

            if( _erro )
            {
```

```
                _identificadorVariavel = _listaExpressoes[_contador]-
>getFilhos( )[0]->getFilhos( )[0]->getFilhos( )[0]->getFilhos( )[0]-
>getFilhos( )[0]->getFilhos( )[0]->getDescricao( );

                _resultadoBuscaParametro = this->hash
[std::pair<const std::string, const unsigned int>
(_identificadorVariavel, this->nivelLexicoAtual)];

                if( _resultadoBuscaParametro != this->hash.end() )
                {
                    if( _resultadoBusca->second->procedureFunction-
>parametros[_contador].second != _resultadoBuscaParametro->second-
>getTipo() );
                    {
                        LogErros::getInstancia().insereErro
( _chamadaFuncao->getLinha(), "Tipos imcompatíveis de parametros." );
                    }
                }
            }
        }
        else
        {
            _resultadoExpressao = this->expressao( _listaExpressoes
[_contador] );

            if( _resultadoBuscaParametro != this->hash.end() )
            {
                if( _resultadoBusca->second->procedureFunction-
>parametros[_contador].second != _resultadoBuscaParametro->second-
>getTipo() );
                {
                    LogErros::getInstancia().insereErro
( _chamadaFuncao->getLinha(), "Tipos imcompatíveis de parametros." );
                }
            }
        }
    }

    return _resultadoExpressao;
}
```

```
std::string
AnalisadorSemantico::chamadaProcedimento( NoArvoreSintatica*
_chamadaProcedimento )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _chamadaProcedimento->getFilhos( );

    std::vector<NoArvoreSintatica*>
    _listaExpressoes;

    std::vector<NoArvoreSintatica*>
    _verificacaoParametro;

    NoArvoreSintatica*
    _paiVerificacaoParametro;

    TabelaHash::iterator
    _resultadoBusca;

    TabelaHash::iterator
    _resultadoBuscaParametro;

    unsigned int
    _contador;

    std::string
    _identificadorVariavel;

    std::string
    _descricao = _filhos[0]->getFilhos( )[0]->getDescricao( );

    std::string
    _tipoParametro;

    std::string
    _bufferContador;

    std::string
    _resultadoExpressao;
```

```
    std::string
    _quantidadeParametrosStr;

    unsigned int
    _quantidadeParametros;

    bool
    _encontrado = false;

    bool
    _erro = false;

    /*      'this->nivelLexicoAtual' nunca podera ser menor que 0
por definicao
     *      para evitar o estouro para cima de unsigned int foi
utilizada a comparação '!= 0'
     */

    if( this->hash[std::pair<const std::string, const unsigned int>
(_descricao, this->nivelLexicoAtual+1)] != this->hash.end() )
    {
        _resultadoBusca = this->hash[std::pair<const std::string,
const unsigned int>(_descricao, this->nivelLexicoAtual+1)];

        _encontrado = true;
    }

    if( !_encontrado )
    {
        for( _contador = 0; (this->nivelLexicoAtual-_contador) +1 !=
0; ++_contador )
        {
            if( this->hash[std::pair<const std::string, const
unsigned int>(_descricao, this->nivelLexicoAtual-_contador)] != this-
>hash.end() )
            {
                _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual-
_contador)];
```

```
                _encontrado = true;
                break;
            }
        }
    }

    _resultadoExpressao = _resultadoBusca->second->getTipo( );
    _quantidadeParametros = _resultadoBusca->second-
>procedureFunction->quantidadeParametros;

    if( _quantidadeParametros != 0 )
    {
        _listaExpressoes = _filhos[2]->getFilhos( );
    }

    if( _quantidadeParametros != ((_listaExpressoes.size()+1)/2) )
    {
        _quantidadeParametrosStr = _quantidadeParametros;
        LogErros::getInstancia().insereErro( _chamadaProcedimento-
>getLinha(), "Quantidade de parametros incorreta. Sao esperados [" +
_quantidadeParametrosStr + "] parametros" );
    }

    for( _contador = 0; _contador < _listaExpressoes.size();
_contador += 2 )
    {
        if( _resultadoBusca->second->procedureFunction->parametros
[_contador/2].first == true )
        {
            _bufferContador = (_contador+1);
            _paiVerificacaoParametro = _listaExpressoes[_contador];
            _verificacaoParametro = _listaExpressoes[_contador]-
>getFilhos( );

            while( _verificacaoParametro.size() < 2 )
            {
                _bufferContador = (_contador+1);

                if( _verificacaoParametro.size() == 0 )
```

```
                {
                    if( _paiVerificacaoParametro->getDescricao( ) ==
"<VARIAVEL>" )
                    {
                        break;
                    }
                    else
                    {
                        LogErros::getInstancia().insereErro
( _chamadaProcedimento->getLinha(), "O " + _bufferContador +
"deveria ser uma variavel." );
                        _erro = true;
                    }
                }
                else if( _verificacaoParametro.size() == 1 )
                {
                    _paiVerificacaoParametro = _verificacaoParametro
[0];
                    _verificacaoParametro = _verificacaoParametro[0]-
>getFilhos( );
                }
                else
                {
                    LogErros::getInstancia().insereErro
( _chamadaProcedimento->getLinha(), "O " + _bufferContador +
"deveria ser uma variavel." );
                    _erro = true;
                }
            }
            if( _verificacaoParametro.size() < 2 )
            {
                LogErros::getInstancia().insereErro
( _chamadaProcedimento->getLinha(), "O " + _bufferContador +
"deveria ser uma variavel." );
                _erro = true;
            }

            if( _erro )
            {
                _identificadorVariavel = _listaExpressoes[_contador]-
```

```
>getFilhos( )[0]->getFilhos( )[0]->getFilhos( )[0]->getFilhos( )[0]-
>getFilhos( )[0]->getFilhos( )[0]->getDescricao( );

                _resultadoBuscaParametro = this->hash
[std::pair<const std::string, const unsigned int>
(_identificadorVariavel, this->nivelLexicoAtual)];

                if( _resultadoBuscaParametro != this->hash.end() )
                {
                    if( _resultadoBusca->second->procedureFunction-
>parametros[_contador].second != _resultadoBuscaParametro->second-
>getTipo() );
                    {
                        LogErros::getInstancia().insereErro
( _chamadaProcedimento->getLinha(), "Tipos imcompatíveis de
parametros." );
                    }
                }
            }
        }
        else
        {
            _resultadoExpressao = this->expressao( _listaExpressoes
[_contador] );

            if( _resultadoBuscaParametro != this->hash.end() )
            {
                if( _resultadoBusca->second->procedureFunction-
>parametros[_contador].second != _resultadoBuscaParametro->second-
>getTipo() );
                {
                    LogErros::getInstancia().insereErro
( _chamadaProcedimento->getLinha(), "Tipos imcompatíveis de
parametros." );
                }
            }
        }
    }

    return _resultadoExpressao;
```

```
}

std::string
AnalisadorSemantico::expressao( NoArvoreSintatica* _expressao )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _expressao->getFilhos( );

    std::string
    _resultado;

    _resultado = this->expressaoSimples( _filhos[0] );

    if( _filhos.size() != 1 )
    {
        _resultado = this->relacao( _filhos[1] );
    }

    return _resultado;
}

std::string
AnalisadorSemantico::relacao( NoArvoreSintatica* _relacao )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _relacao->getFilhos( );

    if( (_filhos[0]->getDescricao( ) == "=")  ||
        (_filhos[0]->getDescricao( ) == "<>") ||
        (_filhos[0]->getDescricao( ) == "<")  ||
        (_filhos[0]->getDescricao( ) == "<=") ||
        (_filhos[0]->getDescricao( ) == ">")  ||
        (_filhos[0]->getDescricao( ) == ">=") )
    {
        return "boolean";
    }
}

std::string
AnalisadorSemantico::expressaoSimples( NoArvoreSintatica*
```

```
_expressaoSimples )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _expressaoSimples->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos = _filhos.begin( );

    std::string
    _resultado;


    if( _filhos[0]->getDescricao() == "+" )
    {
        ++_iteradorFilhos;
        if( this->termo((*_iteradorFilhos)) != "integer" );
        {
            LogErros::getInstancia().insereErro( _expressaoSimples-
>getLinha(), "Operador '+' nao pode ser usado com termo
'boolean'." );
            return "boolean";
        }
    }
    else if ( _filhos[0]->getDescricao() == "-" )
    {
        ++_iteradorFilhos;
        if( this->termo((*_iteradorFilhos)) != "integer" );
        {
            LogErros::getInstancia().insereErro( _expressaoSimples-
>getLinha(), "Operador '-' nao pode ser usado com termo
'boolean'." );
            return "boolean";
        }
    }
    else
    {
        _resultado = this->termo( (*_iteradorFilhos) );
    }

    ++_iteradorFilhos;
```

```
    for( ; _iteradorFilhos != _filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() == "+" )
        {
            if( this->termo(++(*_iteradorFilhos)) != "integer" );
            {
                LogErros::getInstancia().insereErro
( _expressaoSimples->getLinha(), "Operador '+' nao pode ser usado
com termo 'boolean'." );
                return "boolean";
            }
        }
        else if( (*_iteradorFilhos)->getDescricao() == "-" )
        {
            if( this->termo(++(*_iteradorFilhos)) != "integer" );
            {
                LogErros::getInstancia().insereErro
( _expressaoSimples->getLinha(), "Operador '-' nao pode ser usado
com termo 'boolean'." );
                return "boolean";
            }
        }
        else if( (*_iteradorFilhos)->getDescricao() == "or" )
        {
            return "boolean";
        }
        else
        {
            _resultado = this->termo( (*_iteradorFilhos) );
        }
    }

    return _resultado;
}

std::string
AnalisadorSemantico::termo( NoArvoreSintatica* _termo )
{
    std::vector<NoArvoreSintatica*>
```

```
    _filhos = _termo->getFilhos( );

    std::vector<NoArvoreSintatica*>::iterator
    _iteradorFilhos;

    std::string
    _resultado;

    for( _iteradorFilhos = _filhos.begin( ); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() == "*" )
        {
            if( this->fator(++(*_iteradorFilhos)) != "integer" );
            {
                LogErros::getInstancia().insereErro( _termo->getLinha
(), "Operador '*' nao pode ser usado com fator 'boolean'." );
                return "boolean";
            }
        }
        else if( (*_iteradorFilhos)->getDescricao() == "div" )
        {
            if( this->fator(++(*_iteradorFilhos)) != "integer" );
            {
                LogErros::getInstancia().insereErro( _termo->getLinha
(), "Operador 'div' nao pode ser usado com fator 'boolean'." );
                return "boolean";
            }
        }
        else if( (*_iteradorFilhos)->getDescricao() == "and" )
        {
            return "boolean";
        }
        else
        {
            _resultado = this->fator( (*_iteradorFilhos) );
        }
    }

    return _resultado;
```

```
}

std::string
AnalisadorSemantico::fator( NoArvoreSintatica* _fator )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _fator->getFilhos( );

    TabelaHash::iterator
    _resultadoBusca;

    unsigned int
    _contador;

    std::string
    _classificacao;

    std::string
    _descricao;

    bool
    _encontrado = false;

    std::string
    _resultado;

    if( _filhos[0]->getDescricao() == "not")
    {
        if( this->fator(_filhos[1]) != "boolean" );
        {
            LogErros::getInstancia().insereErro( _fator->getLinha(),
"Operador 'not' nao pode ser usado com fator 'integer'." );
            return "boolean";
        }
        _resultado = "boolean";
    }
    else if( _filhos[0]->getDescricao() == "(" )
    {
        _resultado = this->expressao( _filhos[1] );
    }
```

```
    else if( _filhos[0]->getDescricao() == "<NUMERO>" )
    {
        if( (_filhos[0]->getFilhos()[0]->getDescricao() == "true") ||
            (_filhos[0]->getFilhos()[0]->getDescricao() == "true") )
        {
            _resultado = "boolean";
        }
    }
    else if( _filhos[0]->getDescricao() == "<VARIAVEL>" )
    {
        _descricao = _filhos[0]->getFilhos()[0]->getFilhos()[0]-
>getDescricao();

        /*      'this->nivelLexicoAtual' nunca podera ser menor que
0 por definicao
         *      para evitar o estouro para cima de unsigned int foi
utilizada a comparação '!= 0'
         */
        for( _contador = 0; (this->nivelLexicoAtual-_contador) +1 !=
0; ++_contador )
        {
            if( this->hash[std::pair<const std::string, const
unsigned int>(_descricao, this->nivelLexicoAtual-_contador)] != this-
>hash.end() )
            {
                _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual-
_contador)];
                _classificacao = _resultadoBusca->second->getConteudo
();

                _encontrado = true;
                break;
            }
        }

        if( !_encontrado )
        {
            if( this->hash[std::pair<const std::string, const
unsigned int>(_descricao, this->nivelLexicoAtual+1)] != this-
```

```
>hash.end() )
            {
                _resultadoBusca = this->hash[std::pair<const
std::string, const unsigned int>(_descricao, this->nivelLexicoAtual
+1)];
                _classificacao = _resultadoBusca->second->getConteudo
();

                _encontrado = true;
            }
        }

        _resultado = _resultadoBusca->second->getTipo( );
    }
    else if( _filhos[0]->getDescricao() == "<CHAMADA_FUNCAO>" )
    {
        _resultado = this->chamadaFuncao( _filhos[0] );
    }

    return _resultado;
}
```