```cpp
#include <fstream>
#include <iostream>

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

#include "../includes/GeradorDeCodigo.h"

std::string
itos( int _inteiro )
{
    std::ostringstream
    _buffer;

    _buffer << _inteiro;

    return _buffer.str();
}

int
stoi( std::string _string )
{
    if( _string == "true" )
    {
        return 1;
    }
    else if( _string == "false" )
    {
        return 0;
    }
    else
    {
        return ::atoi( _string.c_str() );
    }
}


GeradorDeCodigo::GeradorDeCodigo( std::pair<TabelaHash*,
NoArvoreSintatica*> _saidaAnalisadorSintatico )
{
```

```
    this->hash = *(_saidaAnalisadorSintatico.first);
    this->raiz = _saidaAnalisadorSintatico.second;

    this->nivelLexicoAtual = 0;
    this->contadorLabel = 1;

    this->contadorLabelCR = 1;

    this->contadorLabelCC = 1;

    this->atribuiLabel( "CorpoProgramaPrincipal" );

    this->iniciaGeracaoDeCodigo( );

    this->salvaCodigoMepa( );
}

GeradorDeCodigo::~GeradorDeCodigo( )
{

}

void
GeradorDeCodigo::salvaCodigoMepa( )
{
    std::ofstream
    _arquivoMepa;

    std::string
    _caminhoCodigo = "../execMepa/" + this->nomePrograma + ".mep";

    std::string
    _stringBufferComandos;

    _arquivoMepa.open( _caminhoCodigo.c_str(), std::ifstream::out );

    if ( _arquivoMepa.bad() ) throw ( new ErrosExecucao("O arquivo
de log nao pode ser aberto!! Sucesso;;") );

    _stringBufferComandos = this->bufferComandos.str( );
```

```
    _arquivoMepa.write( _stringBufferComandos.c_str(),
_stringBufferComandos.size() );

    _arquivoMepa.close( );
}


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

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


    for( _iteradorFilhos = _filhos.begin(); _iteradorFilhos !=
_filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() == "program" )
        {
            this->INPP( );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<IDENTIFICADOR>" )
        {
            this->nomePrograma = (*_iteradorFilhos)->getFilhos( )[0]-
>getDescricao( );
        }
        else if( (*_iteradorFilhos)->getDescricao() == "<BLOCO>" )
        {

            this->bloco( *_iteradorFilhos );

            while( !this->pilhaComandos.empty() )
            {
                this->desempilhaComando( );
            }
```

```
        }
        else if( (*_iteradorFilhos)->getDescricao() == "." )
        {
            this->PARA( );
        }
    }
}

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

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

    bool
    _parteDeclaracoesVariaveis = false;

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

        if( (*_iteradorFilhos)->getDescricao() ==
"<PARTE_DECLARACOES_ROTULOS>" )
        {
            this->declaracaoDeRotulos( *_iteradorFilhos );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<PARTE_DECLARACOES_VARIAVEIS>" )
        {
            this->declaracaoDeVariaveis( *_iteradorFilhos );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<PARTE_DECLARACOES_SUB_ROTINAS>" )
        {
            this->DSVS( this->ultimosLabelsInseridos.back() );
            _parteDeclaracoesVariaveis = true;
```

```
            this->declaracaoDeSubrotinas( *_iteradorFilhos );
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<COMANDO_COMPOSTO>" )
        {
            if( !_parteDeclaracoesVariaveis )
            {
                this->DSVS( this->ultimosLabelsInseridos.back() );
                _parteDeclaracoesVariaveis = true;
            }

            this->insereLabelNada( );
            this->comandoComposto( *_iteradorFilhos );
        }
    }
}

void
GeradorDeCodigo::declaracaoDeRotulos( NoArvoreSintatica*
_parteDeclaracoesRotulos )
{

}


void
GeradorDeCodigo::declaracaoDeVariaveis( NoArvoreSintatica*
_parteDeclaracoesVariaveis )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _parteDeclaracoesVariaveis->getFilhos( );

    unsigned int
    _quantidadeVariaveis = 0;

    unsigned int
    _contador;

    for( _contador = 1; _contador < _filhos.size(); _contador += 2 )
    {
```

```
        _quantidadeVariaveis += ( _filhos[_contador]->getFilhos()[0]-
>getFilhos().size() + 1 ) / 2;
    }

    this->AMEM( _quantidadeVariaveis );

    this->empilhaComando( "DMEM " + itos(_quantidadeVariaveis) );
}

void
GeradorDeCodigo::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>" )
        {
            _descricao = (*_iteradorFilhos)->getFilhos( )[1]-
>getFilhos( )[0]->getDescricao( );
            this->atribuiLabel( _descricao );
            this->insereLabelNada( );

            this->ENPR( ++this->nivelLexicoAtual );

            _descricao.append( "Corpo" );
            this->atribuiLabel( _descricao );

            this->bloco( (*_iteradorFilhos)->getFilhos()[6] );
```

```
            if( (*_iteradorFilhos)->getFilhos()[6]->getFilhos()[0]-
>getDescricao() == "<PARTE_DECLARACOES_VARIAVEIS>")
            {
                this->desempilhaComando( );
            }

            this->RTPR( this->nivelLexicoAtual, this->hash
[std::pair<const std::string, const unsigned int>((*_iteradorFilhos)-
>getFilhos()[1]->getFilhos()[0]->getDescricao(), this-
>nivelLexicoAtual)]->second->procedureFunction-
>quantidadeParametros );
            --this->nivelLexicoAtual;
        }
        else if( (*_iteradorFilhos)->getDescricao() ==
"<DECLARACAO_PROCEDIMENTO>" )
        {
            _descricao = (*_iteradorFilhos)->getFilhos( )[1]-
>getFilhos( )[0]->getDescricao( );
            this->atribuiLabel( _descricao );
            this->insereLabelNada( );

            this->ENPR( ++this->nivelLexicoAtual );

            _descricao.append( "Corpo" );
            this->atribuiLabel( _descricao );

            if( (*_iteradorFilhos)->getFilhos()[2]->getDescricao()
== "<PARAMETROS_FORMAIS>" )
            {
                this->bloco( (*_iteradorFilhos)->getFilhos()[4] );

                if( (*_iteradorFilhos)->getFilhos()[2]->getFilhos()
[0]->getDescricao() == "<PARTE_DECLARACOES_VARIAVEIS>")
                {
                    this->desempilhaComando( );
                }
            }
            else
            {
                this->bloco( (*_iteradorFilhos)->getFilhos()[3] );
```

```
                if( (*_iteradorFilhos)->getFilhos()[3]->getFilhos()
[0]->getDescricao() == "<PARTE_DECLARACOES_VARIAVEIS>")
                {
                    this->desempilhaComando( );
                }
            }

            this->RTPR( this->nivelLexicoAtual, this->hash
[std::pair<const std::string, const unsigned int>((*_iteradorFilhos)-
>getFilhos()[1]->getFilhos()[0]->getDescricao(), this-
>nivelLexicoAtual)]->second->procedureFunction-
>quantidadeParametros );
            --this->nivelLexicoAtual;
        }
    }
}

void
GeradorDeCodigo::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
```

```
GeradorDeCodigo::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() == "<NUMERO>" )
        {

        }
        if( (*_iteradorFilhos)->getDescricao() ==
"<COMANDO_SEM_ROTULO>" )
        {
            this->comandoSemRotulo( *_iteradorFilhos );
        }
    }
}

void
GeradorDeCodigo::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() == "<DESVIOS>" )
    {
        this->desvios( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<COMANDO_LEITURA>" )
    {
        this->comandoLeitura( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<COMANDO_ESCRITA>" )
    {
        this->comandoEscrita( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<ATRIBUICAO>" )
    {
        this->atribuicao( _filhos[0] );
    }
    else if( _filhos[0]->getDescricao() == "<CHAMADA_PROCEDIMENTO>" )
    {
        this->chamadaProcedimento( _filhos[0] );
    }
}

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

    this->ultimosLabelsInseridosCR.push_back( "CR" + itos(this-
>contadorLabelCR) );
    ++this->contadorLabelCR;
    this->bufferComandos << this->ultimosLabelsInseridosCR.at( this-
>ultimosLabelsInseridosCR.size() - 1 ) << " NADA" << std::endl;

    this->expressao( _filhos[1] );

    this->ultimosLabelsInseridosCR.push_back( "CR" + itos(this-
```

```
>contadorLabelCR) );
    ++this->contadorLabelCR;
    this->DSVF( this->ultimosLabelsInseridosCR.at(this-
>ultimosLabelsInseridosCR.size() - 1) );

    this->comandoSemRotulo( _filhos[3] );

    this->DSVS( this->ultimosLabelsInseridosCR.at(this-
>ultimosLabelsInseridosCR.size() - 2) );
    this->bufferComandos << this->ultimosLabelsInseridosCR.at( this-
>ultimosLabelsInseridosCR.size() - 1 ) << " NADA" << std::endl;

    /*
     * Consertar Aqui: Eliminar duplicidade de POPs
     */
    this->ultimosLabelsInseridosCR.pop_back( );
    this->ultimosLabelsInseridosCR.pop_back( );
}

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

    this->ultimosLabelsInseridosCC.push_back( "CC" + itos(this-
>contadorLabelCC) );
    ++this->contadorLabelCC;

    this->expressao( _filhos[1] );
    this->DSVF( this->ultimosLabelsInseridosCC.at(this-
>ultimosLabelsInseridosCC.size() - 1) );

    this->comandoSemRotulo( _filhos[3] );

    this->ultimosLabelsInseridosCC.push_back( "CC" + itos(this-
>contadorLabelCC) );
    ++this->contadorLabelCC;
    this->DSVS( this->ultimosLabelsInseridosCC.at(this-
```

```
>ultimosLabelsInseridosCC.size() - 1) );

    this->bufferComandos << this->ultimosLabelsInseridosCC.at( this-
>ultimosLabelsInseridosCC.size() - 2 ) << " NADA" << std::endl;

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

    this->bufferComandos << this->ultimosLabelsInseridosCC.at( this-
>ultimosLabelsInseridosCC.size() - 1 ) << " NADA" << std::endl;

    /*
     * Consertar Aqui: Eliminar duplicidade de POPs
     */
    this->ultimosLabelsInseridosCC.pop_back( );
    this->ultimosLabelsInseridosCC.pop_back( );
}

void
GeradorDeCodigo::desvios( NoArvoreSintatica* _desvios )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _desvios->getFilhos( );
}

void
GeradorDeCodigo::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( _classificacao == "variavel" )
            {
                this->LEIT( );

                this->ARMZ( _resultadoBusca->second->variavel-
>nivelLexico, _resultadoBusca->second->variavel->deslocamento );
            }
            else if( _classificacao == "parametrosFormais" )
            {
                if( _resultadoBusca->second->parametrosFormais-
>passagem == false )
                {
                    this->LEIT( );

                    this->ARMZ( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
>parametrosFormais->deslocamento );
                }
                else if( _resultadoBusca->second->parametrosFormais-
>passagem == true )
                {
                    this->LEIT( );

                    this->ARMI( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
>parametrosFormais->deslocamento );
                }
            }

        }
    }
}

void
GeradorDeCodigo::comandoEscrita( NoArvoreSintatica* _comandoEscrita )
{
    std::vector<NoArvoreSintatica*>
```

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

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

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

            this->IMPR( );
        }
    }
}

void
GeradorDeCodigo::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();


    /*      '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() )
    {
        this->expressao( _filhos[2] );

        if( _classificacao == "variavel" )
        {
            this->ARMZ( _resultadoBusca->second->variavel-
>nivelLexico, _resultadoBusca->second->variavel->deslocamento );
        }
        else if( _classificacao == "parametrosFormais" )
        {
            if( _resultadoBusca->second->parametrosFormais->passagem
== false )
            {
                this->ARMZ( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
>parametrosFormais->deslocamento );
            }
            else if( _resultadoBusca->second->parametrosFormais-
>passagem == true )
            {
```

```
                this->ARMI( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
>parametrosFormais->deslocamento );
            }
        }
        else if( _classificacao == "procedimento|funcao" )
        {
            this->ARMZ( _resultadoBusca->second->procedureFunction-
>nivelLexico, _resultadoBusca->second->procedureFunction-
>deslocamento );
        }
    }
}

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

    std::vector<NoArvoreSintatica*>
    _listaExpressoes;

    TabelaHash::iterator
    _resultadoBusca;

    TabelaHash::iterator
    _resultadoBuscaParametro;

    unsigned int
    _contador;

    std::string
    _identificadorVariavel;

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

    std::string
    _tipoParametro;
```

```
    unsigned int
    _quantidadeParametros;

    bool
    _encontrado = false;


    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;
            }
        }
    }
```

```
    this->AMEM( 1 );

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

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

    for( _contador = 0; _contador < _listaExpressoes.size();
_contador += 2 )
    {
        if( _resultadoBusca->second->procedureFunction->parametros
[_contador/2].first == false )
        {
            this->expressao( _listaExpressoes[_contador] );
        }
        else
        {
            _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() )
            {
                _tipoParametro = _resultadoBuscaParametro->second-
>getConteudo( );

                if( _tipoParametro == "parametrosFormais")
                {
                    if( _resultadoBuscaParametro->second-
>parametrosFormais->passagem == true )
                    {
                        this->CRVL( _resultadoBuscaParametro->second-
```

```
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                    }
                    else
                    {
                        this->CREN( _resultadoBuscaParametro->second-
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                    }
                }
                else
                {
                    this->CREN( _resultadoBuscaParametro->second-
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                }
            }
        }
    }

    this->CHPR( this->indexLabel[_descricao] );
}

void
GeradorDeCodigo::chamadaProcedimento( NoArvoreSintatica*
_chamadaProcedimento )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _chamadaProcedimento->getFilhos( );

    std::vector<NoArvoreSintatica*>
    _listaExpressoes;

    TabelaHash::iterator
    _resultadoBusca;

    TabelaHash::iterator
    _resultadoBuscaParametro;

    unsigned int
```

```
    _contador;

    std::string
    _identificadorVariavel;

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

    std::string
    _tipoParametro;

    unsigned int
    _quantidadeParametros;

    bool
    _encontrado = 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;
            }
        }
    }


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

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

    for( _contador = 0; _contador < _listaExpressoes.size();
_contador += 2 )
    {
        if( _resultadoBusca->second->procedureFunction->parametros
[_contador/2].first == false )
        {
            this->expressao( _listaExpressoes[_contador] );
        }
        else
        {
            _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() )
            {
```

```
                _tipoParametro = _resultadoBuscaParametro->second-
>getConteudo( );

                if( _tipoParametro == "parametrosFormais")
                {
                    if( _resultadoBuscaParametro->second-
>parametrosFormais->passagem == true )
                    {
                        this->CRVL( _resultadoBuscaParametro->second-
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                    }
                    else
                    {
                        this->CREN( _resultadoBuscaParametro->second-
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                    }
                }
                else
                {
                    this->CREN( _resultadoBuscaParametro->second-
>variavel->nivelLexico, _resultadoBuscaParametro->second->variavel-
>deslocamento );
                }
            }
        }
    }

    this->CHPR( this->indexLabel[_descricao] );
}

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

    this->expressaoSimples( _filhos[0] );
```

```
    if( _filhos.size() != 1 )
    {
        this->relacao( _filhos[1] );
        this->expressaoSimples( _filhos[2] );
        this->desempilhaComando( );
    }
}

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

    if( _filhos[0]->getDescricao( ) == "=" )
    {
        this->empilhaComando( "CMIG" );
    }
    else if( _filhos[0]->getDescricao( ) == "<>" )
    {
        this->empilhaComando( "CMDG" );
    }
    else if( _filhos[0]->getDescricao( ) == "<" )
    {
        this->empilhaComando( "CMME" );
    }
    else if( _filhos[0]->getDescricao( ) == "<=" )
    {
        this->empilhaComando( "CMEG" );
    }
    else if( _filhos[0]->getDescricao( ) == ">" )
    {
        this->empilhaComando( "CMMA" );
    }
    else if( _filhos[0]->getDescricao( ) == ">=" )
    {
        this->empilhaComando( "CMAG" );
    }
}
```

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

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

    bool
    _desempilha = false;

    if( _filhos[0]->getDescricao() == "+" )
    {
        ++_iteradorFilhos;
        this->termo( *_iteradorFilhos );

        this->CRCT( 1 );
        this->MULT( );
    }
    else if ( _filhos[0]->getDescricao() == "-" )
    {
        ++_iteradorFilhos;
        this->termo( *_iteradorFilhos );

        this->CRCT( -1 );
        this->MULT( );
    }
    else
    {
        this->termo( _filhos[0] );
    }

    ++_iteradorFilhos;

    for( ; _iteradorFilhos != _filhos.end(); ++_iteradorFilhos )
    {
        if( (*_iteradorFilhos)->getDescricao() == "+" )
        {
```

```
            this->empilhaComando( "SOMA" );
            _desempilha = true;
        }
        else if( (*_iteradorFilhos)->getDescricao() == "-" )
        {
            this->empilhaComando( "SUBT" );
            _desempilha = true;
        }
        else if( (*_iteradorFilhos)->getDescricao() == "or" )
        {
            this->empilhaComando( "DISJ" );
            _desempilha = true;
        }
        else
        {
            this->termo( (*_iteradorFilhos) );

            if( _desempilha )
            {
                this->desempilhaComando( );
                _desempilha = false;
            }
        }
    }
}

void
GeradorDeCodigo::termo( NoArvoreSintatica* _termo )
{
    std::vector<NoArvoreSintatica*>
    _filhos = _termo->getFilhos( );

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

    bool
    _desempilha = false;

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

```
    {
        if( (*_iteradorFilhos)->getDescricao() == "*" )
        {
            this->empilhaComando( "MULT" );
            _desempilha = true;
        }
        else if( (*_iteradorFilhos)->getDescricao() == "div" )
        {
            this->empilhaComando( "DIVI" );
            _desempilha = true;
        }
        else if( (*_iteradorFilhos)->getDescricao() == "and" )
        {
            this->empilhaComando( "CONJ" );
            _desempilha = true;
        }
        else
        {
            this->fator( (*_iteradorFilhos) );

            if( _desempilha )
            {
                this->desempilhaComando( );
                _desempilha = false;
            }
        }
    }
}

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

    TabelaHash::iterator
    _resultadoBusca;

    unsigned int
    _contador;
```

```
    std::string
    _classificacao;

    std::string
    _descricao;

    bool
    _encontrado = false;

    if( _filhos[0]->getDescricao() == "not")
    {
        this->fator( _filhos[1] );
        this->NEGA( );
    }
    else if( _filhos[0]->getDescricao() == "(" )
    {
        this->expressao( _filhos[1] );
    }
    else if( _filhos[0]->getDescricao() == "<NUMERO>" )
    {
        this->CRCT( stoi(_filhos[0]->getFilhos()[0]->getDescricao
()) );
    }
    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;
            }
        }

        if( _classificacao == "variavel" )
        {
            this->CRVL( _resultadoBusca->second->variavel-
>nivelLexico, _resultadoBusca->second->variavel->deslocamento );
        }
        else if( _classificacao == "parametrosFormais" )
        {
            if( _resultadoBusca->second->parametrosFormais->passagem
== false )
            {
                this->CRVL( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
```

```
>parametrosFormais->deslocamento );
            }
            else if( _resultadoBusca->second->parametrosFormais-
>passagem == true )
            {
                this->CRVI( _resultadoBusca->second-
>parametrosFormais->nivelLexico, _resultadoBusca->second-
>parametrosFormais->deslocamento );
            }
        }
    }
    else if( _filhos[0]->getDescricao() == "<CHAMADA_FUNCAO>" )
    {
        this->chamadaFuncao( _filhos[0] );
    }
}

void
GeradorDeCodigo::empilhaComando( std::string _comando )
{
    this->pilhaComandos.push_back( _comando );
}

void
GeradorDeCodigo::desempilhaComando( )
{
    if( this->pilhaComandos.size() != 0 )
    {
        this->bufferComandos << *(this->pilhaComandos.rbegin() ) <<
std::endl;
        this->pilhaComandos.pop_back( );
    }
}

void
GeradorDeCodigo::atribuiLabel( std::string _nomeFuncao )
{
    this->indexLabel[_nomeFuncao] = "LBC" + itos(this-
>contadorLabel);
```

```
    this->ultimosLabelsInseridos.push_back( "LBC" + itos(this-
>contadorLabel) );

    ++this->contadorLabel;
}

void
GeradorDeCodigo::insereLabelNada( )
{
    this->bufferComandos << this->ultimosLabelsInseridos.back( ) <<
" NADA" << std::endl;

    this->ultimosLabelsInseridos.pop_back( );
}

void
GeradorDeCodigo::CRCT( int K )
{
    this->bufferComandos << "CRCT " << K << std::endl;
}

void
GeradorDeCodigo::CRVL( int k, int n )
{
    this->bufferComandos << "CRVL " << k << ", " << n << std::endl;
}

void
GeradorDeCodigo::SOMA( )
{
    this->bufferComandos << "SOMA" << std::endl;
}

void
GeradorDeCodigo::SUBT( )
{
    this->bufferComandos << "SUBT" << std::endl;
}

void
```

```
GeradorDeCodigo::MULT( )
{
    this->bufferComandos << "MULT" << std::endl;
}

void
GeradorDeCodigo::DIVI( )
{
    this->bufferComandos << "DIVI" << std::endl;
}

void
GeradorDeCodigo::CMIG( )
{
    this->bufferComandos << "CMIG" << std::endl;
}

void
GeradorDeCodigo::CMMA( )
{
    this->bufferComandos << "CMMA" << std::endl;
}

void
GeradorDeCodigo::CMME( )
{
    this->bufferComandos << "CMME" << std::endl;
}


void
GeradorDeCodigo::CMAG( )
{
    this->bufferComandos << "CMAG" << std::endl;
}

void
GeradorDeCodigo::CMEG( )
{
    this->bufferComandos << "CMEG" << std::endl;
```

```
}

void
GeradorDeCodigo::CMDG( )
{
    this->bufferComandos << "CMDG" << std::endl;
}

void
GeradorDeCodigo::CONJ( )
{
    this->bufferComandos << "CONJ" << std::endl;
}

void
GeradorDeCodigo::DISJ( )
{
    this->bufferComandos << "DISJ" << std::endl;
}

void
GeradorDeCodigo::NEGA( )
{
    this->bufferComandos << "NEGA" << std::endl;
}

void
GeradorDeCodigo::INVR( )
{
    this->bufferComandos << "INVR" << std::endl;
}


void
GeradorDeCodigo::ARMZ( int k, int n )
{
    this->bufferComandos << "ARMZ " << k << ", " << n << std::endl;
}

void
```

```
GeradorDeCodigo::LEIT( )
{
    this->bufferComandos << "LEIT" << std::endl;
}

void
GeradorDeCodigo::IMPR( )
{
    this->bufferComandos << "IMPR" << std::endl;
}

void
GeradorDeCodigo::NADA( )
{
    this->bufferComandos << "NADA" << std::endl;
}

void
GeradorDeCodigo::DSVS( std::string label )
{
    this->bufferComandos << "DSVS " << label << std::endl;
}

void
GeradorDeCodigo::DSVF( std::string label )
{
    this->bufferComandos << "DSVF " << label << std::endl;
}

void
GeradorDeCodigo::INPP( )
{
    this->bufferComandos << "INPP" << std::endl;
}

void
GeradorDeCodigo::AMEM( int m )
{
    this->bufferComandos << "AMEM " << m << std::endl;
}
```

```
void
GeradorDeCodigo::DMEM( int n )
{
    this->bufferComandos << "DMEM " << n << std::endl;
}

void
GeradorDeCodigo::PARA( )
{
    this->bufferComandos << "PARA" << std::endl;
}

void
GeradorDeCodigo::CHPR( std::string label )
{
    this->bufferComandos << "CHPR " << label << std::endl;
}

void
GeradorDeCodigo::ENPR( int k )
{
    this->bufferComandos << "ENPR " << k << std::endl;
}

void
GeradorDeCodigo::RTPR( int k, int n )
{
    this->bufferComandos << "RTPR " << k << ", " << n << std::endl;
}


void
GeradorDeCodigo::CRVI( int k, int n )
{
    this->bufferComandos << "CRVI " << k << ", " << n << std::endl;
}

void
GeradorDeCodigo::ARMI( int k, int n )
```

```
{
    this->bufferComandos << "ARMI " << k << ", " << n << std::endl;
}

void
GeradorDeCodigo::CREN( int k, int n )
{
    this->bufferComandos << "CREN " << k << ", " << n << std::endl;
}
```