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

#include "iofilter.h"

#pragma warning(push)
#pragma warning( disable : 4251 4267 4101 4267 )

#include "FjolnirForritLexer.hpp"
#include "FjolnirEiningLexer.hpp"
#include "FjolnirParser.hpp"
#include "FjolnirTransformer.hpp"
#include "FjolnirCodegen.hpp"
#include <antlr/AST.hpp>
#include <antlr/CommonAST.hpp>
#include <antlr/TokenStreamSelector.hpp>
#include "myast.h"

#pragma warning(pop)

int main_lex(std::istream& input, std::ostream& output);
int main_parse(std::istream& input, std::ostream& output, int dotformat);

typedef enum {
        MODE_LEX,
        MODE_PARSE,
        MODE_COMPILE
} runmode;

#define UTGAFA  "1.0"

void useage(std::ostream& out) {
        using namespace std;
        out <<
                "Falskur Fjölnir, þýðandi - útgáfa " UTGAFA << endl <<
                " 2003 (c) Arnar Birgisson, Háskóli Íslands" << endl <<
                " Byggt á þýðandanum og forritunarmálinu Fjölni, höfundar:" << endl <<
                "   Páll Björnsson, Jón Harðarson, Snorri Agnarsson" << endl <<
                " Notkun:" << endl <<
                "   falskur_fjolnir [-l | -p [-dN]] [-iso] [-n] [-o skrá] [skrá]" << endl <<
                "      -l      framkvæmir aðeins lesgreiningu" << endl <<
                "      -p      framkvæmir aðeins þáttun" << endl <<
                "      -dN     skrifar út máltré á formi sem nota má sem inntak í" << endl <<
                "                forritið \"dot\". N er annað hvort 1 eða 2, ef 1 er" << endl <
<
                "                skrifað út máltré fyrir umbreytingu, annars eftir" << endl <<
                "      -iso    úttak þýðandans er í ISO-8859-1, annars CP861" << endl <<
                "      -n      inntak er lesið sem CP-861, annars ISO-8859-1" << endl <<
                "      -o skrá úttak er skrifað í skrá, annars stdout" << endl <<
                "      skrá    inntak er lesið úr skrá, annars stdin" << endl << endl;
        exit(2);
}

std::ostream* __ff_errors = NULL;

int main(int argc, char** args)
{
        using namespace std;
        using namespace antlr;
        using namespace ff;

        runmode mode = MODE_COMPILE;
        bool convert_input = false;
        bool convert_output = true;
        int dot_output = 0;
        char* output_filename = NULL;
        char* input_filename = NULL;
        ostream* output;
        istream* input;

        /* Skrifum allt úttak í 861 */
        ostream cerr(new ofilterbuf(_trans_iso_861, cerr.rdbuf()));
        __ff_errors = &cerr;

        for (int i = 1; i < argc; i++) {
                char* arg = args[i];
                if ('-' == arg[0]) {
                        switch (arg[1]) {
```

```cpp
                        case 'l': case 'L':
                                mode = MODE_LEX;
                                break;
                        case 'd': case 'D':
                                if ('1' == arg[2]) {
                                        dot_output = 1;
                                } else if ('2' == arg[2]) {
                                        dot_output = 2;
                                } else {
                                        useage(cerr);
                                }
                                /* fall trough */
                        case 'p': case 'P':
                                mode = MODE_PARSE;
                                break;
                        case 'i': case 'I':
                                if (0 == stricmp("-iso", arg))
                                        convert_output = false;
                                break;
                        case 'n': case 'N':
                                convert_input = true;
                                break;
                        case 'o': case 'O':
                                if (i+1 >= argc || '-' == args[i+1][0])
                                        useage(cerr);
                                output_filename = args[++i];
                        case '?': case 'h': case 'H':
                                useage(cerr);
                                break;
                        }
                } else {
                        if (input_filename)
                                useage(cerr);
                        input_filename = arg;
                }
        }

        if (input_filename) {
                input = new ifstream(input_filename, ios::in);
        } else {
                input = &cin;
        }

        if (output_filename) {
                output = new ofstream(output_filename, ios::out);
        } else {
                output = &cout;
        }

        if (convert_input) {
                input = new istream(new ifilterbuf(_trans_861_iso, input->rdbuf()));
        }

        if (convert_output) {
                output = new ostream(new ofilterbuf(_trans_iso_861, output->rdbuf()));
        }

        if (MODE_LEX == mode) {
                return main_lex(*input, *output);
        } else if (MODE_PARSE == mode) {
                return main_parse(*input, *output, dot_output);
        }

        try {
                cerr << " Fasi 0: Uppsetning... ";

                TokenStreamSelector selector;

                FjolnirForritLexer forritLexer(*input);
                forritLexer.initialize(&selector);

                FjolnirEiningLexer einingLexer(forritLexer.getInputState());
                einingLexer.initialize(&selector);

                selector.addInputStream(&forritLexer, "forritlexer");
                selector.addInputStream(&einingLexer, "einingLexer");
                selector.select("forritlexer");
                ASTFactory my_factory("ffAST", ffAST::factory);
                FjolnirParser parser(selector);

                parser.initializeASTFactory(my_factory);
                parser.setASTFactory(&my_factory);

                cerr << "lokið." << endl;

                cerr << " Fasi 1: Lesgreining og þáttun... ";
                parser.forrit();
                RefAST ast = RefAST(parser.getAST());
                cerr << "lokið." << endl;

                cerr << " Fasi 2: Umbreyting máltrés... ";
                FjolnirTransformer tparser;
                tparser.initializeASTFactory(my_factory);
                tparser.setASTFactory(&my_factory);
                tparser.forrit(ast);
                RefAST transformed = RefAST(tparser.getAST());
                cerr << "lokið." << endl;

                cerr << " Fasi 3: Þulusmíði... ";
                FjolnirCodegen cgparser;
                cgparser.setOutput(*output);
                cgparser.forrit(transformed);
                cerr << "lokið." << endl;

        } catch(exception& e) {
                cerr << "Villa í þýðingu: " << e.what() << endl;
        }

        /* Aðvörun: Hér lekum við hugsanlega minni í formi
           [io]filterbuf og [io]stream hluta, látum það gott heita
           þar eð keyrslu lýkur hér eftir.
             Þar sem destructor í ofilterbuf er hins vegar aldrei
           framkvæmdur reynist okkur nauðsynlegt að framkvæma eftir−
           farandi kall til að skrifa út úttak úr honum ef eitthvert er.
        */

        output−>flush();

}

int main_lex(std::istream& input, std::ostream& output) {
        using namespace std;
        using namespace antlr;
        using namespace ff;

        TokenStreamSelector selector;

        FjolnirForritLexer forritLexer(input);
        forritLexer.initialize(&selector);

        FjolnirEiningLexer einingLexer(forritLexer.getInputState());
        einingLexer.initialize(&selector);

        selector.addInputStream(&forritLexer, "forritlexer");
        selector.addInputStream(&einingLexer, "einingLexer");
        selector.select("forritlexer");

        /* fyrir tók−nöfn */
        FjolnirParser parser(selector);

        RefToken t;
        char buffer[128];
        while ( (t=selector.nextToken())−>getType()!=Token::EOF_TYPE ) {
                ::_snprintf(buffer, 128, "%−30s <%2d> %s\n", parser.getTokenName(t−>type), t−>t
ype, t−>getText().c_str());
                output << buffer;
        }
        return 0;
}

void printTree(antlr::RefAST tree, std::ostream& out, antlr::Parser& p, int indent = 0);
int printDotTree(antlr::RefAST tree, std::ostream& out, antlr::Parser& p);

int main_parse(std::istream& input, std::ostream& output, int dotformat)
{
```

```
        using namespace std;
        using namespace antlr;
        using namespace ff;

        try {
                TokenStreamSelector selector;

                FjolnirForritLexer forritLexer(input);
                forritLexer.initialize(&selector);

                FjolnirEiningLexer einingLexer(forritLexer.getInputState());
                einingLexer.initialize(&selector);

                selector.addInputStream(&forritLexer, "forritlexer");
                selector.addInputStream(&einingLexer, "einingLexer");
                selector.select("forritlexer");

                ASTFactory my_factory;
                FjolnirParser parser(selector);

                parser.initializeASTFactory(my_factory);
                parser.setASTFactory(&my_factory);

                parser.forrit();
                RefAST ast = RefAST(parser.getAST());

                if (1 == dotformat) {
                        if (ast) {
                                output << "digraph G {" << endl;
                                output << "edge [fontname=\"Helvetica\",fontsize=10,label
me=\"Helvetica\",labelfontsize=10];" << endl;
                                output << "node [fontname=\"Helvetica\",fontsize=10,shape
" << endl;
                                printDotTree(ast, output, parser);
                                output << "}" << endl;
                        }
                } else if (2 != dotformat) {
                        output << "Fyrir umbreytingu:" << endl;
                        if (ast) {
                                printTree(ast, output, parser);
                        } else {
                                output << "null AST" << endl;
                        }
                }

                FjolnirTransformer tparser;
                tparser.initializeASTFactory(my_factory);
                tparser.setASTFactory(&my_factory);
                tparser.forrit(ast);
                RefAST transformed = RefAST(tparser.getAST());

                if (2 == dotformat) {
                        if (transformed) {
                                output << "digraph G {" << endl;
                                output << "edge [fontname=\"Helvetica\",fontsize=10,label
me=\"Helvetica\",labelfontsize=10];" << endl;
                                output << "node [fontname=\"Helvetica\",fontsize=10,shape
" << endl;
                                printDotTree(transformed, output, parser);
                                output << "}" << endl;
                        }
                } else if (1 != dotformat) {
                        output << "Eftir umbreytingu:" << endl;
                        if (transformed) {
                                printTree(ast, output, parser);
                        } else {
                                output << "null AST" << endl;
                        }
                }
        } catch(exception& e) {
                output << "Villa í þáttun: " << e.what() << endl;
        }

        return 0;
}

void printTree(antlr::RefAST tree, std::ostream& out, antlr::Parser& p, int indent) {
        int j = indent;
        std::string i = ""; while (j-- > 0) i += "  ";
        if (tree->getFirstChild()) {
                out << i << "( " << tree->toString() << " <" <<
                        p.getTokenName(tree->getType()) << ">" << std::endl;
                printTree(tree->getFirstChild(), out, p, indent+1);
                out << i << ")" << std::endl;
        } else {
                out << i << tree->toString() << " <" <<
                        p.getTokenName(tree->getType()) << ">" << std::endl;
        }
        if (tree->getNextSibling()) {
                printTree(tree->getNextSibling(), out, p, indent);
        }
}

static int _dot_node = 0;
int printDotTree(antlr::RefAST tree, std::ostream& out, antlr::Parser& p) {
        int me = ++_dot_node;
        out << me << " [label=\"" << p.getTokenName(tree->getType())
                << "\\n" << tree->getText() << "\"];" << std::endl;
        antlr::RefAST c = tree->getFirstChild();
        while (antlr::nullAST != c) {
                int child = printDotTree(c, out, p);
                out << me << " -> " << child << ";" << std::endl;
                c = c->getNextSibling();
        }
        return me;
}
```