```
header "pre_include_cpp" {
#pragma warning( disable : 4251 4267 4101 4267 )
}

header "post_include_hpp" {
#include "stef.h"
#include "segd.h"
#include "segdir.h"
}

header "post_include_cpp" {

#include "myast.h"
//#define loc(s,t) if (NULL != static_cast<ff::ffAST*>(t.get().ptr)) \
//                                      (s)->setLine((static_cast<ff::ffAST*>(t.get().ptr))->ge
tLine());
#define loc(s,t) (s)->setLine((static_cast<ffAST*>(t.get()))->getLine());

using namespace std;
}

options {
        language="Cpp";
        namespace="ff";
        namespaceStd="std";
        namespaceAntlr="antlr";
        genHashLines=true;
}

class FjolnirCodegen extends TreeParser;

options {
        k = 2;
        importVocab = FjolnirTransformer;
        buildAST = false;
}

{

public:
        void setOutput(std::ostream& out) { this->out = &out; }

private:
        std::ostream *out;
}

forrit
        : ( veiting )*
        ;

veiting
        : #(EIN_MINNA nafn:EIN_STRENGUR start:EIN_NAFN
                {
                        *out << '"' << nafn->getText() << "\" < " << start->getText() << endl;
                }
                eining { *out << ";"; } )
        | #(EIN_JAFNTOG (s:EIN_STRENGUR {*out << '"' << s->getText() << '"';} | n:EIN_NAFN {*ou
t << n->getText();} )
                {
                        *out << " =" << endl;
                }
                eining { *out << ";"; } )
        ;

eining
        : #(EIN_ITRUD {*out << "(!";} eining {*out << ")";} )
        | #(EIN_ITRUDHLIDS  {*out << "(";} eining {*out << " & ";} eining {*out << ")";} )
        | #(EIN_HLIDSETNING {*out << "(";} eining {*out << " + ";} eining {*out << ")";} )
        | #(EIN_SAMSETNING  {*out << "(";} eining {*out << " : ";} eining {*out << ")";} )
        | #(EIN_INNFLUTT    {*out << "(";} eining {*out << " * ";} eining {*out << ")";} )
        | EIN_STRENGUR  { *out << '"' << #EIN_STRENGUR->getText() << '"'; }
        | EIN_NAFN  { *out << #EIN_NAFN->getText(); }
        | #(EIN_OPNASLAUFU
                { *out << endl << "{" << endl; }
                (vorpun)*
                { *out << endl << "}" << endl; }
          )
        ;
```

```
vorpun
{ string nafn; }
        : #(INNSETNING
                ( NAFN {*out << (nafn = #NAFN->getText());}
                | ADGERD {*out << (nafn = #ADGERD->getText());}
                )  { *out << " -> "; }
                minnissvaedi[nafn] )
        ;

minnissvaedi [string& nafn]
{ Stef* s=NULL; }
        : NAFN                  { *out << #NAFN->getText() << endl; }
        | ADGERD                { *out << #ADGERD->getText() << endl; }
        | L_BREYTA              { *out << "breyta" << endl; }
        | s=stefskilgreining[nafn] {
                *out << '@' << s->getFjoldiVidfanga(AFRIT)
                     << ',' << s->getFjoldiVidfanga(GILDI) << "@(" << endl;
                s->generate(*out);
                *out << ')' << endl;
        }
        ;

stefskilgreining[string& nafn] returns [Stef* ret]
{ Segd* s; Stef* stef; }
        : #(L_STEF                      { ret = new Stef(nafn); }
                #(NAFNARUNA (a:NAFN {
                        if (ret->isLocallyDefined(a->getText())) {
                                cerr << "Nafnið \"" << a->getText() << "\" er þegar skilg
" << endl;
                                exit(1);
                        }
                        ret->addVidfang(AFRIT,a->getText());
                } )* )
                #(NAFNARUNA (b:NAFN {
                        if (ret->isLocallyDefined(b->getText())) {
                                cerr << "Nafnið \"" << b->getText() << "\" er þegar skilg
" << endl;
                                exit(1);
                        }
                        ret->addVidfang(GILDI,b->getText());
                } )* )
                #(SKILGREININGAR
                        #(L_INNFLUTT (c:NAFN {
                                ret->addInnflutt(c->getText());
                        } )* )
                        #(L_STADVAER
                                ( d:NAFN        {
                                        if (ret->isLocallyDefined(d->getText())) {
                                                cerr << "Nafnið \"" << d->getText() << "\
egar skilgreint." << endl;
                                                exit(1);
                                        }
                                        ret->addStadvaer(d->getText());
                                }
                                | #(GILDISVEITING e:NAFN s=segd) {
                                        if (ret->isLocallyDefined(e->getText())) {
                                                cerr << "Nafnið \"" << e->getText() << "\
egar skilgreint." << endl;
                                                exit(1);
                                        }
                                        ret->addStadvaer(e->getText(), s);
                                }
                                )*
                        )
                )
                #(SEGDARUNA (s=segd {ret->addSegd(s);} )*)
                (#(SLAUFA_OPNA
                        (#(INNSETNING f:NAFN stef=stefskilgreining[f->getText()]
                                {
                                        if (ret->isLocallyDefinedUndirstef(stef->getNafn(
                                                        stef->getFjoldiVidfanga(AFRIT), s
etFjoldiVidfanga(GILDI))) {
                                                cerr << "Undirstefið \"" << stef->getNafn
"\" er þegar skilgreint." << endl;
                                                exit(1);
                                        }
                                        ret->addUndirstef(stef);
                                }
                        ))*
                ))?
        )
        ;

        /*
nafnaruna
        : #(NAFNARUNA (NAFN)*)
        ;

skilgreiningar
        : #(SKILGREININGAR #(L_INNFLUTT (NAFN)*) #(L_STADVAER frumstillingaruna))
        ;

frumstillingaruna
        : ( NAFN | #(GILDISVEITING NAFN segd) )*
        ;

innriTextaeining
        : #(SLAUFA_OPNA (innraStef)* )
        ;

innraStef
        : #(INNSETNING NAFN stefskilgreining)
        ;

segdaruna
        : #(SEGDARUNA (segd)*)
        ;
*/

segd returns [Segd* rets=NULL]
{ Segd* a=NULL; Segd* b=NULL; }
        : #(L_OG a=segd b=segd) { rets = new OgSegd(a,b); loc(rets,#L_OG); }
        | #(L_EDA a=segd b=segd) { rets = new EdaSegd(a,b); loc(rets,#L_EDA); }
        | #(L_EKKI a=segd) { rets = new EkkiSegd(a); loc(rets, #L_EKKI); }
        | #(GILDISVEITING NAFN a=segd) { rets = new GildisveitingarSegd(#NAFN->getText(), a); l
oc(rets,#GILDISVEITING); }
        | #(SVIGI_OPNA NAFN { KallSegd* kall = new KallSegd(#NAFN->getText()); }
                #(NAFNARUNA (NAFN {kall->addAfritsVidfang(#NAFN->getText());} )*)
                #(SEGDARUNA (a=segd {kall->addGildisVidfang(a);} )*)
        )       { rets = kall; loc(rets, #SVIGI_OPNA); }
        | #(L_STOFN             { StofnSegd* stofn = new StofnSegd(); }
                #(SEGDARUNA (a=segd {stofn->addSegd(a);} )* )
        )       { rets = (Segd*) stofn; loc(rets,#L_STOFN); }
        | #(L_EF a=segd { EfSegd* ef = new EfSegd(); ef->addSkilyrdi(a); }
                #(SEGDARUNA (a=segd {ef->addSegd(a);} )* )
                (#(L_ANNARSEF a=segd {ef->addSkilyrdi(a);}
                        #(SEGDARUNA (a=segd {ef->addSegd(a);} )* )
                ))*
                #(SEGDARUNA (a=segd {ef->addAnnarsSegd(a);} )* )
        )       { rets = (Segd*) ef; loc(rets,#L_EF); }
        | rets=lykkjusegd
        | #(L_VAL { cerr << "Valsegð er ekki útfærð." << endl; exit(1); }
                /* segd (#(L_KOSTUR valfasti_range segdaruna))* segdaruna */ )
        | #(L_SKILA a=segd) { rets = new SkilaSegd(a); loc(rets,#L_SKILA) }
        | L_UT { rets = new UtSegd(); loc(rets,#L_UT); }
        | NAFN { rets = new NafnSegd(#NAFN->getText()); loc(rets,#NAFN); }
        | STRENGFASTI { rets = new StrengSegd(#STRENGFASTI->getText()); loc(rets,#STRENGFASTI);
 }
        | STAFFASTI   { rets = new HeiltoluSegd(#STAFFASTI->getText()[0]); loc(rets,#STAFFASTI)
 }
        | FJOLDATALA  { rets = new HeiltoluSegd(#FJOLDATALA->getText()); loc(rets,#FJOLDATALA);
 }
        | HEILTALA    { rets = new HeiltoluSegd(#HEILTALA->getText()); loc(rets,#HEILTALA); }
        | FLEYTITALA  { rets = new FleytitoluSegd(#FLEYTITALA->getText()); loc(rets,#FLEYTITALA
); }
        | TOMAGILDI   { rets = new TomaSegd(); loc(rets,#TOMAGILDI); }
        | #(L_STEF NAFN FJOLDATALA FJOLDATALA)
                { cerr << "Stefgildi eru ekki útfærð." << endl; exit(1); }
        ;

lykkjusegd returns [Segd* rets]
{ Segd* a; }
        : #(L_LYKKJA { LykkjuSegd* lykkja = new LykkjuSegd(); }
                #(SEGDARUNA (a=segd {lykkja->addSegd(a);} )* )
        )       { rets = (Segd*) lykkja; loc(rets,#L_LYKKJA); }
```

```
        | #(L_MEDAN a=segd { MedanSegd* medan = new MedanSegd(a); }
                #(SEGDARUNA (a=segd {medan->addSegd(a);} )* )
        )       { rets = (Segd*) medan; loc(rets,#L_MEDAN); }
        ;

//valfasti_range
//      : STAFFASTI
//      | HEILTALA
//      | #(PUNKTURPUNKTUR valfasti_range valfasti_range) /* þáttari skilar aldrei (a..b)
//      ;
```

```
#ifndef __smali_h__
#define __smali_h__

#include <string>

namespace ff {

#define emit(x,y)   out << '\t' << x << '\t' << y << '\n';

#define emit_forskeyti(x) out << '\t' << x << ":\n";

#define emit_push(x)  out << "\tPUSH\t" << x << '\n'; \
        _umlykjandiStef->stackDelta(2);

#define emit_pop(x)  out << "\tPOP\t" << x << '\n'; \
        _umlykjandiStef->stackDelta(-2);

#define emit_popn(n)  out << "\tADD\tSP," << (n) << '\n'; \
        _umlykjandiStef->stackDelta(-(n));

#define emit_label(x) out << x << ":\n";

extern unsigned int __nextLabel;
#define newlabel() (__nextLabel++)
#define l(x) "_" << (x)

std::string quote(const std::string& s);

};

#endif /* __smali_h__ */
```

```
#include "smali.h"

unsigned int ff::__nextLabel = 1;

std::string ff::quote(const std::string& s) {
        std::string retval;
        std::string::const_iterator i;
        for (i = s.begin(); i != s.end(); i++) {
                retval.push_back('\\');
                retval.push_back(*i);
        }
        return retval;
}
```

```
#ifndef __stef_h__
#define __stef_h__

#include <vector>
#include <map>
#include <list>
#include <string>
#include <iostream>

#include "segd.h"
#include "smali.h"

#include <stack>

namespace ff {

using namespace std;

typedef enum {
        AFRIT = 1,
        GILDI = 2
} vidfangsTegund;

typedef list<pair<string, int> > symtab;
typedef list<string> stringlist;

struct symloc {
        unsigned int foldun;
        int offset;
        symloc() : foldun(0), offset(0) {}
};

class Stef {
        string _nafn;
        symtab _afritsVidfong;
        symtab _gildisVidfong;
        symtab _localBreytur;
        list<string> _innfluttarBreytur;

        list<Segd*> _frumstillingar;
        list<Segd*> _segdaruna;

        int _stackSize;
        int _fjoldiAfritsVidfanga;
        int _fjoldiGildisVidfanga;
        int _fjoldiLocalBreyta;
        int _steflokLabel;

        Stef* _parent;
        int _nestingLevel;
        map<string, Stef*> _undirStef;

        stack<int> _stackMarks; /* fyrir lykkjur */
        stack<int> _utLabels;   /* ditto */

public:
        Stef(string& nafn)
                : _nafn(nafn), _parent(NULL), _nestingLevel(0),
                  _stackSize(0), _fjoldiAfritsVidfanga(0), _fjoldiGildisVidfanga(0),
                  _fjoldiLocalBreyta(0), _steflokLabel(newlabel())
        {}
        virtual ~Stef();

        /** Nafn stefsins.
         \return nafn stefsins.
         */
        const string& getNafn() const { return _nafn; }

        /** Lokamerki stefsins.
         \return merki sem er skrifað í þulu strax á undan eftirmála
         */
        int getEndLabel() const { return _steflokLabel; }

        /** Athugar hvort nafn er skilgreint inni í stefinu.
         \return true þ.þ.a.a. nafn er skilgreint viðfangs eða breytunafn
         */
        bool isDefined(const string& nafn);
```

```
        /** Athugar hvort undirstef er skilgreint innan í stefinu
         \return 0 ef ekki er til undirstef sem er hægt að kalla í frá núverandi
           staðsetningu í þulu, annars földunarhæð viðkomandi undirstefs.
         */
        int isDefinedUndirstef(const string& nafn, int n, int m);

        /** Skilar nafni á merki undirfalls
         \pre isDefinedUndirstef(...) == true
         */
        string getUndirstefLabel(const string& nafn, int n, int m);

        /** Bætir við viðfangi af tegund t með nafni nafn.
         \pre isLocallyDefined(nafn) == false
         \post Stefið þekkir staðsetningu viðfangsins á stafla
         */
        void addVidfang(vidfangsTegund t, string& nafn);

        /** Sækir fjölda þegar skilgreindra viðfanga.
         \return fjölda þegar skilgreindra viðfanga af tegund t
         */
        int getFjoldiVidfanga(vidfangsTegund t);

        /** Bætir við nafni innfluttrar breytu
         \pre isLocallyDefined(nafn) == false
         \post Stefið þekkir nafn sem nafn innfluttrar breytu
         */
        void addInnflutt(string& nafn);

        /** Bætir við staðværri breytu, hugsanlega með frumstillingu
         \pre isLocallyDefined(nafn) == false og ef
              frumstilling er annað hvort null eða bendir á löglega Segð
         \post Stefið þekkir nafn sem breytunafn ásamt staðsetningu á stafla,
               og mun skrifa út þulu til að frumstilla breytuna. Þetta stef
                    mun sjá um að losa minni fyrir frumstillinguna
     */
        void addStadvaer(string& nafn, Segd* frumstilling = NULL);

        /** Bætir við undirstefi undir þetta stef.
         \pre isLocallyDefinedUndirstef(stef->_nafn) == false og stef er
              bendir á löglegt Stef.
         \post Þetta stef þekkir nafn sem nafn undirstefs og mun skrifa út
               þulu þess. Kallað hefur verið í stef->setParent með réttu viðf.
                    Þetta Stef mun sjá um að losa minni sem stef bendir í.
         */
        void addUndirstef(Stef* stef);

        /** Bætir við segð í stefið.
         \pre s er bendir í löglega Segð
         \post stefið mun skrifa út þulu segðarinnar strax á eftir þulum þeirra
               segða sem þegar hafa verið settar inn með þessu boði. Kallað hefur
                    verið í s->setUmlykjandiStef. Þetta stef mun sjá um að losa minni
                    sem stef bendir í
         */
        void addSegd(Segd* s);

        /** Setur bendi í stefið í næstu földunarhæð fyrir ofan.
         \pre parent er bendir í löglegt Stef
         \post Þetta stef inniheldur bendi í parent og heiltölu földunardýpt, sem
               er einum hærri en samsvarandi tala í parent, þetta stef um sjá
                    um að losa minni sem s bendir í
         */
        void setParent(Stef* parent);

        /** Sækir földunardýpt þessa falls.
         \return földunardýpt þessa falls (0 ef þetta er grunnfall)
         */
        int getNestingLevel();

        /** Sækir staðsetningu viðfangs eða breytu á stafla.
         \pre isDefined(name) == true
         \return symloc struct sem inniheldur tölur föidun og offset. Föidun
           segir til um hvað þarf að fara upp um margar vakningarfærslur til að finna
           viðkomandi breytu, og offset inniheldur staðsetningu breytunnar
           m.v. grunnstak (BP) þeirrar vakningarfærslu í bætum
           Ef name er nafn á innfluttri breytu skilar fallið sérgildinu {0,0}
         */
        symloc getSymbolLocation(const string& name);
        /** Skilar streng sem auðkennir fallið.
         \return streng sem auðkennir fallið út frá nafni þess, fjölda viðfanga að hvorri
          gerð og næsta falli fyrir ofan í földunarhæð.
         */
        string getInternalNafn();

        /** Gefur fallinu tilkynningu um að þula breyti stærð staflans.
         \post Stærð staflans hefur breyst um d bæti.
         */
        void stackDelta(int d);

        /** Sækir stærð staflans m.v. þá þulu sem hefur verið skrifuð út.
         \return Stærð staflans frá síðustu staðværu breytu í bætum eftir að
           sú þula sem hefur verið skrifuð út hefur keyrt.
         */
        int getStackSize();

        /** Setur núverandi staflastærð efst á stafla.
          \post næsta kall í lastStackMark mun skila núverandi staflastærð
          */
        void markStack();

        /** Gleymir síðasta gildi úr markStack.
          \pre Kallað hefur verið oftar í markStack en unmarkStack
          \post næsta gildi úr lastStackMark mun verða stærð staflans við þarsíðasta markStack
          */
        void unmarkStack();

        /** Sækir stærð staflans við síðasta markStack
         \return Stærð staflans þegar kallað var í markStack síðast, eða 0 ef
           kallað hefur verið jafn oft í markStack og unmarkStack
         */
        int lastStackMark();

        /* TODO: docs, gera ekki inline? */
        void pushUtLabel(int l) { _utLabels.push(l); }
        void popUtLabel() { _utLabels.pop(); }
        int getUtLabel() { return _utLabels.top(); }

        /** Smíðar þulu fyrir stefið.
         \pre Ekki verður kallað aftur í föllin addInnflutt, addSegd, addStadvaer,
           addUndirstef, addVidfang eða setParent. out er löglegur úttaksstraumur.
         \post Búið er að skrifa í out þulu stefsins.
         */
        void generate(ostream& out);

        /** Athugar hvort nafn er locally skilgreint
          \return true þ.þ.a.a. nafn sé skilgreint breytu- eða viðfangsnafn í þessu stefi
          */
        bool isLocallyDefined(const string& nafn);

        /** Athugar hvort nafn er nafn á beinu undirstefi
          \return true þ.þ.a.a. nafn sé nafn á beinu undirstefi þessa falls
          */
        bool isLocallyDefinedUndirstef(const string& nafn, int n, int m);

private:
        /** Leitar í symboltöflu.
         \pre s er lögleg symtab, nafn er löglegur strengur
         \return iterator sem bendir á parið <str,i> með str==nafn ef það er til í s,
           s.end() annars
         */
        symtab::iterator findSymbol(symtab& s, const string& nafn);

};

}

#endif /* __stef_h__ */
```

```
#include "stef.h"
#include "smali.h"

#include <stdio.h>

using namespace ff;

Stef::~Stef() {
        list<Segd*>::iterator s;
        for (s = _frumstillingar.begin(); s != _frumstillingar.end(); s++) {
                delete (*s);
        }
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                delete (*s);
        }
        map<string,Stef*>::iterator u;
        for (u = _undirStef.begin(); u != _undirStef.end(); u++) {
                delete (*u).second;
        }
}

bool Stef::isLocallyDefined(const string& nafn) {
        if (findSymbol(_localBreytur, nafn) != _localBreytur.end()) {
                return true;
        }
        list<string>::iterator i;
        for (i = _innfluttarBreytur.begin(); i != _innfluttarBreytur.end(); i++) {
                if ((*i) == nafn) return true;
        }
        if (findSymbol(_gildisVidfong, nafn) != _gildisVidfong.end()) {
                return true;
        }
        if (findSymbol(_afritsVidfong, nafn) != _afritsVidfong.end()) {
                return true;
        }
        return false;
}

bool Stef::isDefined(const string& nafn) {
        if (isLocallyDefined(nafn))
                return true;
        if (_parent)
                return _parent->isDefined(nafn);
        return false;
}

bool Stef::isLocallyDefinedUndirstef(const string& nafn, int n, int m) {
        char prefix[32];
        ::_snprintf(prefix, 32, "@%d,%d@", n, m);
        string realname = prefix + nafn;
        if (_undirStef.find(realname) != _undirStef.end())
                return true;
        return false;
}

int Stef::isDefinedUndirstef(const string& nafn, int n, int m) {
        if (isLocallyDefinedUndirstef(nafn, n, m))
                return _nestingLevel + 1;
        if (_parent)
                return _parent->isDefinedUndirstef(nafn, n, m);
        return 0;
}

string Stef::getUndirstefLabel(const string& nafn, int n, int m) {
        char prefix[32];
        if (isLocallyDefinedUndirstef(nafn, n,m)) {
                ::_snprintf(prefix, 32, "@%d,%d@", n, m);
                string realname = prefix + nafn;
                return (*(_undirStef.find(realname))).second->getInternalNafn();
        }
        if (_parent)
                return _parent->getUndirstefLabel(nafn, n, m);
        /* ættum ekki að komast hingað m.v. forskilyrði */
        return "(vitleysa)";
}

void Stef::addVidfang(vidfangsTegund t, string& nafn) {
        switch (t) {
```

```cpp
        case AFRIT:
                _afritsVidfong.push_back(symtab::value_type(nafn, ++_fjoldiAfritsVidfanga));
                break;
        case GILDI:
                _gildisVidfong.push_back(symtab::value_type(nafn, ++_fjoldiGildisVidfanga));
                break;
        }
}

int Stef::getFjoldiVidfanga(vidfangsTegund t) {
        switch (t) {
        case AFRIT:
                return _afritsVidfong.size();
        case GILDI:
                return _gildisVidfong.size();
        }
        return 0;
}

void Stef::addInnflutt(string& nafn) {
        _innfluttarBreytur.push_back(nafn);
}

void Stef::addStadvaer(string& nafn, Segd* frumstilling) {
        _localBreytur.push_back(symtab::value_type(nafn, ++_fjoldiLocalBreyta));
        _frumstillingar.push_back(frumstilling); /* má vera null */
}

void Stef::addUndirstef(Stef* stef) {
        char prefix[32];
        ::_snprintf(prefix, 32, "@%d,%d@", stef->_fjoldiAfritsVidfanga, stef->_fjoldiGildisVidf
anga);
        string realname = prefix + stef->_nafn;
        _undirStef.insert(map<string,Stef*>::value_type(realname, stef));
        stef->setParent(this);
}

void Stef::addSegd(Segd* s) {
        s->setUmlykjandiStef(this);
        _segdaruna.push_back(s);
}

void Stef::setParent(Stef* parent) {
        _parent = parent;
        _nestingLevel = parent->getNestingLevel()+1;
}

int Stef::getNestingLevel() {
        return _nestingLevel;
}

symloc Stef::getSymbolLocation(const string& nafn) {
        symloc loc;
        Stef* s = this;
        symtab::iterator i;
        list<string>::iterator l;
        while (s) {
                if ((i = s->findSymbol(s->_localBreytur, nafn)) != s->_localBreytur.end()) {
                        loc.offset = -(*i).second<<2;
                        break;
                }
                for (l = _innfluttarBreytur.begin(); l != _innfluttarBreytur.end(); l++) {
                        if ((*l) == nafn) {
                                loc.foldun = 0;
                                loc.offset = 0;
                                goto foundVar;  /* break virkar á for líka :o( */
                        }
                }
                if ((i = s->findSymbol(s->_gildisVidfong, nafn)) != s->_gildisVidfong.end()) {
                        loc.offset = (s->_nestingLevel + s->_fjoldiGildisVidfanga + 2 - (*i).se
cond) << 2;
                        break;
                }
                if ((i = s->findSymbol(s->_afritsVidfong, nafn)) != s->_afritsVidfong.end()) {
                        loc.offset = (s->_nestingLevel + s->_fjoldiGildisVidfanga + s->_fjoldiA
fritsVidfanga + 2
                                - (*i).second) << 2;
                        break;
```

```cpp
                        }
                        loc.foldun++;
                        s = s->_parent;
                }
foundVar:
                return loc;
}

string Stef::getInternalNafn() {
        string s;
        if (_parent) s = _parent->getInternalNafn();
        char prefix[32];
        ::_snprintf(prefix, 32, "__%d_%d_", _fjoldiAfritsVidfanga, _fjoldiGildisVidfanga)
        s += '_' + (prefix + _nafn);
        /* TODO: ef s.length > 255 þá villa */
        return s;
}

void Stef::stackDelta(int d) {
        _stackSize += d;
}

int Stef::getStackSize() {
        return _stackSize;
}

void Stef::markStack() {
        _stackMarks.push(_stackSize);
}

void Stef::unmarkStack() {
        _stackMarks.pop();
}

int Stef::lastStackMark() {
        if (_stackMarks.empty()) {
                return -1;
        } else {
                return _stackMarks.top();
        }
}

void Stef::generate(ostream& out) {
        /* formáli */
        emit("PUSH", "SI");
        emit("PUSH", "BP");
        emit("MOV", "BP,SP");
        emit("PUSH", "SI");

        list<Segd*>::iterator f;
        for (f = _frumstillingar.begin(); f != _frumstillingar.end(); f++) {
                if (NULL == (*f)) {
                        emit("PUSH", "ES");
                        emit("PUSH", "ES");
                } else {
                        (*f)->setUmlykjandiStef(this);
                        (*f)->generatePUSH(out);
                }
        }
        _stackSize = 0;

        list<Segd*>::iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                list<Segd*>::iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        (*s)->setHali();
                        (*s)->generateAXDX(out);
                }
        }
        /* ASSERT(_stackSize == 0) */

        /* eftirmáli */
        emit_label(l(_steflokLabel));
        emit("MOV", "SP,BP");
        emit("POP", "BP");
        emit("POP", "BX");
        emit("RET", ((_fjoldiGildisVidfanga + _nestingLevel) << 2));

        map<string, Stef*>::iterator u;
        for (u = _undirStef.begin(); u != _undirStef.end(); u++) {
                string s = (*u).second->getInternalNafn();
                emit_label(s);
                (*u).second->generate(out);
        }
}

symtab::iterator Stef::findSymbol(symtab& s, const string& nafn) {
        for (symtab::iterator i = s.begin(); i != s.end(); i++)
                if ((*i).first == nafn) return i;
        return s.end();
}
```

```
#ifndef __segd_h__
#define __segd_h__

#include <iostream>

namespace ff {

using namespace std;

class Stef;

class Segd {
protected:
        bool _hali;
        Stef* _umlykjandiStef;
        int _line;

public:
        Segd() : _hali(false), _umlykjandiStef(NULL), _line(0) {}
        virtual ~Segd() {}

        void setHali() { _hali = true; }
        void clearHali() { _hali = false; }
        bool isHali() const { return _hali; }
        void setLine(int line) { _line = line; }

        virtual void setUmlykjandiStef(Stef* stef) { _umlykjandiStef = stef; }

        virtual void generateAXDX(ostream& out) const = 0;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateJUMP(ostream& out, int iftrue, int iffalse) const;
        virtual void generateNOVAL(ostream& out) const;

        void reportError(const char* villa, ...) const;
};

}

#endif /* __segd_h__ */
```

```
#include "segd.h"
#include "smali.h"
#include "stef.h"

#include <iostream>
#include <stdarg.h>

using namespace ff;

void Segd::generatePUSH(ostream& out) const {
        generateAXDX(out);
        emit_push("AX");
        emit_push("DX");
}

void Segd::generateNOVAL(ostream& out) const {
        generateAXDX(out);
}

void Segd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        generateAXDX(out);
        if (iftrue) {
                emit("TEST", "DL,1");
                emit("JZ", l(iftrue));
        }
        if (iffalse) {
                emit("TEST", "DL,1");
                emit("JNZ", l(iffalse));
        }
}

extern std::ostream* __ff_errors;
void Segd::reportError(const char* villa, ...) const {
        char buffer1[32], buffer2[128];
        va_list vl;
        va_start(vl,villa);

        if (0 == _line)
                ::strncpy(buffer1, "Villa: ", 32);
        else
                ::_snprintf(buffer1, 32, "Villa í línu %d: ", _line);
        ::_vsnprintf(buffer2, 128, villa, vl);

        *__ff_errors << endl << buffer1 << buffer2 << endl;

        exit(1);
}
```

```
#ifndef __segdir_h__
#define __segdir_h__

#include "segd.h"
#include "smali.h"

#include <vector>
#include <list>

namespace ff {

/**** segd_operators.cpp ****/
class BinOpSegd : public Segd {
protected:
        Segd* _right;
        Segd* _left;
public:
        BinOpSegd(Segd* l, Segd* r) : _left(l), _right(r) {}
        void setUmlykjandiStef(Stef* stef);
        virtual ~BinOpSegd();
};

class OgSegd : public BinOpSegd {
public:
        OgSegd(Segd* l, Segd* r) : BinOpSegd(l,r) {}
        virtual ~OgSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
};

class EdaSegd : public BinOpSegd {
public:
        EdaSegd(Segd* l, Segd* r) : BinOpSegd(l,r) {}
        virtual ~EdaSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
};

class EkkiSegd : public Segd {
        Segd* _segd;
public:
        EkkiSegd(Segd* s) : _segd(s) {}
        virtual ~EkkiSegd();
        void setUmlykjandiStef(Stef* stef);

        virtual void generateAXDX(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
};

/**** segd_assign.cpp ****/
class GildisveitingarSegd : public Segd {
        string _nafn;
        Segd* _s;
public:
        GildisveitingarSegd(string nafn, Segd* s) : _nafn(nafn), _s(s) {}
        virtual ~GildisveitingarSegd() { delete _s; }
        void setUmlykjandiStef(Stef* stef);

        virtual void generateAXDX(ostream& out) const;
};

class SkilaSegd : public Segd {
        Segd* _s;
public:
        SkilaSegd(Segd* s) : _s(s) {}
        virtual ~SkilaSegd() { delete _s; }
        void setUmlykjandiStef(Stef* stef);

        virtual void generateAXDX(ostream& out) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
};

/**** segd_kall.cpp ****/
class KallSegd : public Segd {
        string _nafn;
        list<string> _afritsVidfong;
        list<Segd*> _gildisVidfong;
public:
        KallSegd(const string& nafn) : _nafn(nafn) {}
        virtual ~KallSegd();
        void setUmlykjandiStef(Stef* stef);

        void addAfritsVidfang(string& nafn);
        void addGildisVidfang(Segd* s);

        virtual void generateAXDX(ostream& out) const;
};

/**** segd_cond.cpp ****/
class EfSegd : public Segd {
        vector<Segd*> _skilyrdi;
        vector<list<Segd*> > _segdarunur;
        list<Segd*> _annarsruna;
public:
        EfSegd() {};
        virtual ~EfSegd();
        void setUmlykjandiStef(Stef* stef);

        void addSkilyrdi(Segd* s);
        void addSegd(Segd* s);
        void addAnnarsSegd(Segd* s);

        virtual void generateAXDX(ostream& out) const;
};
/* class ValSegd : public Segd {}; */

/**** segd_loop.cpp ****/
class LykkjuSegd : public Segd {
        list<Segd*> _segdaruna;
        int _exitLabel;
public:
        LykkjuSegd() { _exitLabel = newlabel(); }
        virtual ~LykkjuSegd();
        void setUmlykjandiStef(Stef* stef);

        void addSegd(Segd* s);

        virtual void generateAXDX(ostream& out) const;
};

class MedanSegd : public Segd {
        list<Segd*> _segdaruna;
        Segd* _cond;
        int _exitLabel;
public:
        MedanSegd(Segd* cond) : _cond(cond)
                { _exitLabel = newlabel(); }
        virtual ~MedanSegd();
        void setUmlykjandiStef(Stef* stef);

        void addSegd(Segd* s);

        virtual void generateAXDX(ostream& out) const;
};

class UtSegd : public Segd {
public:
        UtSegd() {}
        virtual ~UtSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
        virtual void generatePUSH(ostream& out) const;
};

/**** segd_stofn.cpp ****/
class StofnSegd : public Segd {
        list<Segd*> _segdaruna;
public:
        StofnSegd() {}
        virtual ~StofnSegd() {}
        void setUmlykjandiStef(Stef* stef);
```

```
        void addSegd(Segd* s);

        virtual void generateAXDX(ostream& out) const;
        virtual void generateJUMP(ostream& out, int, int) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const;
};

/**** segd_value.cpp ****/
class NafnSegd : public Segd {
        string _nafn;
public:
        NafnSegd(const string& nafn) : _nafn(nafn) {}
        virtual ~NafnSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const {};
};

class StrengSegd : public Segd {
        string _s;
public:
        StrengSegd(const string& s) : _s(s) {}
        virtual ~StrengSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const {};
        virtual void generateJUMP(ostream& out, int, int) const;
};

class HeiltoluSegd : public Segd {
        int _tala;
public:
        HeiltoluSegd(int tala);
        HeiltoluSegd(string& les);
        virtual ~HeiltoluSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const {};
        virtual void generateJUMP(ostream& out, int, int) const;
};

class FleytitoluSegd : public Segd {
        unsigned short _ax, _dx;
public:
        FleytitoluSegd(string& les);
        virtual ~FleytitoluSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const {};
        virtual void generateJUMP(ostream& out, int, int) const;
};

class TomaSegd : public Segd {
public:
        TomaSegd() {}
        virtual ~TomaSegd() {}

        virtual void generateAXDX(ostream& out) const;
        virtual void generatePUSH(ostream& out) const;
        virtual void generateNOVAL(ostream& out) const {}
        virtual void generateJUMP(ostream& out, int, int) const;
};

/* class StefgildisSegd : public Segd {}; */

/**** segd_oo.cpp ****/
/*
class ThessiSegd : public Segd {};
class ArfurSegd : public Segd {};
*/

}

#endif /* __segdir_h__ */
```

```cpp
#include "segdir.h"
#include "smali.h"

using namespace ff;

BinOpSegd::~BinOpSegd() {
        delete _left;
        delete _right;
}

void BinOpSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        _right->setUmlykjandiStef(stef);
        _left->setUmlykjandiStef(stef);
}

void OgSegd::generateAXDX(ostream& out) const {
        int ut = newlabel();
        _left->generateAXDX(out);
        emit("TEST", "DL,1");
        emit("JNZ", l(ut));
        _right->generateAXDX(out);
        emit_label(l(ut));
}

void OgSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        if (iffalse) {
                _left->generateJUMP(out,0,iffalse);
                _right->generateJUMP(out,iftrue,iffalse);
        } else {
                int ut = newlabel();
                _left->generateJUMP(out,0,ut);
                _right->generateJUMP(out,iftrue,0);
                emit_label(l(ut));
        }
}

void EdaSegd::generateAXDX(ostream& out) const {
        int ut = newlabel();
        _left->generateAXDX(out);
        emit("TEST", "DL,1");
        emit("JZ", l(ut));
        _right->generateAXDX(out);
        emit_label(l(ut));
}

void EdaSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        if (iftrue) {
                _left->generateJUMP(out,iftrue,0);
                _right->generateJUMP(out,iftrue,iffalse);
        } else {
                int ut = newlabel();
                _left->generateJUMP(out,ut,0);
                _right->generateJUMP(out,0,iffalse);
                emit_label(l(ut));
        }
}

EkkiSegd::~EkkiSegd() {
        delete _segd;
}

void EkkiSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        _segd->setUmlykjandiStef(stef);
}

void EkkiSegd::generateAXDX(ostream& out) const {
        _segd->generateAXDX(out);
        emit("MOV", "AX,ES");
        emit("AND", "DX,1");
        emit("INC", "DX");
}

void EkkiSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        _segd->generateJUMP(out, iffalse, iftrue);
}
```

```cpp
#include "segdir.h"
#include "smali.h"
#include "stef.h"

using namespace ff;

void GildisveitingarSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        _s->setUmlykjandiStef(stef);
}

void GildisveitingarSegd::generateAXDX(ostream& out) const {
        if (!_umlykjandiStef->isDefined(_nafn)) {
                reportError("Nafnið \"%s\" er ekki skilgreint.", _nafn);
        } else {
                _s->generateAXDX(out);
                symloc loc = _umlykjandiStef->getSymbolLocation(_nafn);
                if (0 == loc.foldun && 0 == loc.offset) {
                        /* innflutt breyta */
                        emit("MOV","BX,%" << quote(_nafn));
                        emit_forskeyti("DS");
                        emit("MOV","[BX],DX");
                        emit_forskeyti("DS");
                        emit("MOV","[BX+2],AX");
                } else {
                        if (loc.foldun > 0) {
                                /* assert(nest <= _umlykjandiStef->getNestingLevel() */
                                unsigned int nest = loc.foldun;
                                nest++;    /* fram hjá vendivistf. */
                                nest = nest << 2; /* margf. m. 4 */
                                emit("MOV","BX,[BP+"<<nest<<"]");
                                /* Við höfum fremri addressuna á undan því þá er líklegra
                                   sú seinni verði dregin inn í cache á cpu. */
                                emit_forskeyti("SS");
                                emit("MOV","[BX+" << loc.offset-2 << "],DX");
                                emit_forskeyti("SS");
                                emit("MOV","[BX+" << loc.offset << "],AX");
                        } else {
                                emit("MOV","[BP+" << loc.offset-2 << "],DX");
                                emit("MOV","[BP+" << loc.offset << "],AX");
                        }
                }
        }
}

void SkilaSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        _s->setUmlykjandiStef(stef);
}

void SkilaSegd::generateAXDX(ostream& out) const {
        _s->setHali();
        _s->generateAXDX(out);
        emit("JMP",l(_umlykjandiStef->getEndLabel()));
}

void SkilaSegd::generatePUSH(ostream& out) const {
        generateAXDX(out);
}

void SkilaSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        generateAXDX(out);
}
```

```
#include "segdir.h"
#include "smali.h"
#include "stef.h"

#pragma warning( disable : 4267) /* size_t -> int conversion */

using namespace ff;

KallSegd::~KallSegd() {
        list<Segd*>::iterator i;
        for (i = _gildisVidfong.begin(); i != _gildisVidfong.end(); i++)
                delete (*i);
}

void KallSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        list<Segd*>::iterator i;
        for (i = _gildisVidfong.begin(); i != _gildisVidfong.end(); i++)
                (*i)->setUmlykjandiStef(stef);
}

void KallSegd::addAfritsVidfang(string& nafn) {
        _afritsVidfong.push_back(nafn);
}

void KallSegd::addGildisVidfang(Segd* s) {
        _gildisVidfong.push_back(s);
}

void KallSegd::generateAXDX(ostream& out) const {
        int level = _umlykjandiStef->isDefinedUndirstef(_nafn,
                _afritsVidfong.size(), _gildisVidfong.size());
        int thislevel = _umlykjandiStef->getNestingLevel();

        string steflabel;
        if (level > 0) {
                steflabel = _umlykjandiStef->getUndirstefLabel(_nafn,
                        _afritsVidfong.size(), _gildisVidfong.size());
        } else {
                char forskeyti[32];
                ::_snprintf(forskeyti, 32, "@%d,%d@", _afritsVidfong.size(), _gildisVidfong.siz
e());
                steflabel = forskeyti + quote(_nafn);
        }

        int offset = 0;
        list<string>::const_iterator s;
        for (s = _afritsVidfong.begin(); s != _afritsVidfong.end(); s++) {
                NafnSegd* n = new NafnSegd(*s);
                n->setUmlykjandiStef(_umlykjandiStef);
                n->generatePUSH(out);
                offset += 4;
                delete n;
        }
        list<Segd*>::const_iterator i;
        for (i = _gildisVidfong.begin(); i != _gildisVidfong.end(); i++) {
                (*i)->generatePUSH(out);
                offset += 4;
        }

        for (int l = 0; l < thislevel && l < level; l++) {
                int from = (thislevel + 1 - l) << 2;
                emit_push("[BP+" << from << "]");
                emit_push("SI");
                offset += 4;
        }
        if (level > thislevel) {
                /* assert thislevel+1 == level */
                emit_push("BP");
                emit_push("SI");
                offset += 4;
        }

        if (_hali && _umlykjandiStef->getNestingLevel() >= level
                        && 0 == _umlykjandiStef->getFjoldiVidfanga(AFRIT)
                        && 0 == _afritsVidfong.size()) {
                /* hér er okkur óhætt að henda núverandi vakningarfærslu */
                emit_push("[BP+4]");  /* vendivistfang þess sem kallaði í okkur */
```

```
                offset += 2;
                emit("MOV", "BX,BP");
                emit("MOV", "DX,[BP]");   /* geymum stýrihl í DX */
                emit("MOV", "BP,SP");
                emit("ADD", "BP," << offset-2);  /* BP -> dx hluta fremsta staks í nýja s
*/
                int henda = ( 1 + thislevel + _umlykjandiStef->getFjoldiVidfanga(GILDI))
                emit("ADD", "BX," << henda-1);  /* BX -> ný staðsetning staflans */
                /* færum offset fjölda bæta frá [BP] í [BX], tvö og tvö í einu */
                int loop = newlabel();
                emit("XOR", "SI,SI");
                emit_label(l(loop));
                emit("MOV", "AX,[BP+SI]");
                emit_forskeyti("SS");
                emit("MOV", "[BX+SI],AX");
                emit("DEC", "SI");
                emit("DEC", "SI");
                emit("CMP", "SI," << -offset);
                emit("JA ", l(loop));
                emit("ADD", "BX," << 2-offset);
                emit("MOV", "SP,BX");
                emit("MOV", "BP,DX");
                emit("MOV", "SI,2");
                emit("JMP", steflabel);
        } else {
                emit("CALL", steflabel);
        }
        _umlykjandiStef->stackDelta(-offset + (_afritsVidfong.size()<<2));

        list<string>::const_reverse_iterator rs;
        for (rs = _afritsVidfong.rbegin(); rs != _afritsVidfong.rend(); rs++) {
                symloc loc = _umlykjandiStef->getSymbolLocation(*rs);
                if (0 == loc.foldun && 0 == loc.offset) {
                        /* innflutt breyta */
                        emit("MOV","BX,%" << quote(_nafn));
                        emit_forskeyti("DS");
                        emit_pop("[BX+2]");
                        emit_forskeyti("DS");
                        emit_pop("[BX]");
                } else {
                        if (loc.foldun > 0) {
                                /* assert(nest <= _umlykjandiStef->getNestingLevel() */
                                unsigned int nest = loc.foldun;
                                nest++;    /* fram hjá vendivistf. */
                                nest = nest << 2;
                                emit("MOV","BX,[BP+"<<nest<<"]");
                                emit_forskeyti("SS");
                                emit_pop("[BX+" << loc.offset-2 << "]");
                                emit_forskeyti("SS");
                                emit_pop("[BX+" << loc.offset << "]");
                        } else {
                                emit_pop("[BP+" << loc.offset-2 << "]");
                                emit_pop("[BP+" << loc.offset << "]");
                        }
                }
        }
}
```

```
#include "segdir.h"
#include "smali.h"
#include "stef.h"

using namespace ff;

EfSegd::~EfSegd() {
        vector<Segd*>::iterator s;
        for (s = _skilyrdi.begin(); s != _skilyrdi.end(); s++)
                delete (*s);
        vector<list<Segd*> >::iterator i;
        list<Segd*>::iterator j;
        for (i = _segdarunur.begin(); i != _segdarunur.end(); i++)
                for (j = (*i).begin(); j != (*i).end(); j++)
                        delete (*j);
        for (j = _annarsruna.begin(); j != _annarsruna.end(); j++)
                delete (*j);
}

void EfSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        vector<Segd*>::iterator s;
        for (s = _skilyrdi.begin(); s != _skilyrdi.end(); s++)
                (*s)->setUmlykjandiStef(stef);
        vector<list<Segd*> >::iterator i;
        list<Segd*>::iterator j;
        for (i = _segdarunur.begin(); i != _segdarunur.end(); i++)
                for (j = (*i).begin(); j != (*i).end(); j++)
                        (*j)->setUmlykjandiStef(stef);
        for (j = _annarsruna.begin(); j != _annarsruna.end(); j++)
                (*j)->setUmlykjandiStef(stef);
}

void EfSegd::addSkilyrdi(Segd* s) {
        _skilyrdi.push_back(s);
        list<Segd*> a;
        _segdarunur.push_back(a);
}

void EfSegd::addSegd(Segd* s) {
        _segdarunur.back().push_back(s);
}

void EfSegd::addAnnarsSegd(Segd* s) {
        _annarsruna.push_back(s);
}

void EfSegd::generateAXDX(ostream& out) const {
        int ut = newlabel();
        /* assert _skilyrdi.size() == _segdarunur.size() */

        size_t fj_blokka = _skilyrdi.size();
        for (size_t i = 0; i < fj_blokka; i++) {
                int next = newlabel();
                _skilyrdi[i]->generateJUMP(out, 0, next);
                list<Segd*>::const_iterator s;
                for (s = _segdarunur[i].begin(); s != _segdarunur[i].end(); s++) {
                        list<Segd*>::const_iterator t = s;
                        if (++t != _segdarunur[i].end()) {
                                (*s)->generateNOVAL(out);
                        } else {
                                if (_hali) (*s)->setHali();
                                (*s)->generateAXDX(out);
                        }
                }
                emit("JMP",l(ut));
                emit_label(l(next));
        }

        list<Segd*>::const_iterator s;
        for (s = _annarsruna.begin(); s != _annarsruna.end(); s++) {
                list<Segd*>::const_iterator t = s;
                if (++t != _annarsruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        if (_hali) (*s)->setHali();
                        (*s)->generateAXDX(out);
                }
```

```
        }
        emit_label(l(ut));
}
```

```cpp
#include "segdir.h"
#include "smali.h"
#include "stef.h"

using namespace ff;

LykkjuSegd::~LykkjuSegd() {
        list<Segd*>::iterator i;
        for (i = _segdaruna.begin(); i != _segdaruna.end(); i++)
                delete (*i);
}

void LykkjuSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        list<Segd*>::iterator i;
        for (i = _segdaruna.begin(); i != _segdaruna.end(); i++)
                (*i)->setUmlykjandiStef(stef);
}

void LykkjuSegd::addSegd(Segd* s) {
        _segdaruna.push_back(s);
}

void LykkjuSegd::generateAXDX(ostream& out) const {
        _umlykjandiStef->markStack();
        _umlykjandiStef->pushUtLabel(_exitLabel);

        int begin = newlabel();
        emit_label(l(begin));

        list<Segd*>::const_iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                (*s)->generateNOVAL(out);
                /* list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        (*s)->generateAXDX(out);
                }*/
        }
        emit("JMP",l(begin));

        emit_label(l(_exitLabel));
        _umlykjandiStef->popUtLabel();
        _umlykjandiStef->unmarkStack();
        emit("MOV", "AX,ES");
        emit("MOV", "DX,ES");
}

MedanSegd::~MedanSegd() {
        list<Segd*>::iterator i;
        for (i = _segdaruna.begin(); i != _segdaruna.end(); i++)
                delete (*i);
        delete _cond;
}

void MedanSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        list<Segd*>::iterator i;
        for (i = _segdaruna.begin(); i != _segdaruna.end(); i++)
                (*i)->setUmlykjandiStef(stef);
        _cond->setUmlykjandiStef(stef);
}

void MedanSegd::addSegd(Segd* s) {
        _segdaruna.push_back(s);
}

void MedanSegd::generateAXDX(ostream& out) const {
        _umlykjandiStef->markStack();
        _umlykjandiStef->pushUtLabel(_exitLabel);

        int begin = newlabel();
        emit_label(l(begin));

        _cond->generateJUMP(out, 0, _exitLabel);

        list<Segd*>::const_iterator s;
```

```cpp
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                (*s)->generateNOVAL(out);
                /*list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        (*s)->generateAXDX(out);
                }*/
        }
        emit("JMP",l(begin));

        emit_label(l(_exitLabel));
        _umlykjandiStef->popUtLabel();
        _umlykjandiStef->unmarkStack();
        emit("MOV", "AX,ES");
        emit("MOV", "DX,ES");
}

void UtSegd::generateAXDX(ostream& out) const {
        int mark = _umlykjandiStef->lastStackMark();
        if (-1 == mark) {
                reportError("Út-segð getur aðeins komið fyrir innan í lykkjusegð.");
        }
        int pop = _umlykjandiStef->getStackSize() - mark;
        if (pop > 0) emit("ADD","SP," << pop);
        emit("JMP",l(_umlykjandiStef->getUtLabel()));
}

void UtSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        generateAXDX(out);
}

void UtSegd::generatePUSH(ostream& out) const {
        generateAXDX(out);
}
```

```cpp
#include "segdir.h"
#include "smali.h"
#include "stef.h"

using namespace ff;

void StofnSegd::setUmlykjandiStef(Stef* stef) {
        Segd::setUmlykjandiStef(stef);
        list<Segd*>::iterator i;
        for (i = _segdaruna.begin(); i != _segdaruna.end(); i++)
                (*i)->setUmlykjandiStef(stef);
}

void StofnSegd::addSegd(Segd* s) {
        _segdaruna.push_back(s);
}

void StofnSegd::generateAXDX(ostream& out) const {
        list<Segd*>::const_iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        if (_hali) (*s)->setHali();
                        (*s)->generateAXDX(out);
                }
        }
}

void StofnSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        list<Segd*>::const_iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        if (_hali) (*s)->setHali();
                        (*s)->generateJUMP(out, iftrue, iffalse);
                }
        }
}

void StofnSegd::generatePUSH(ostream& out) const {
        list<Segd*>::const_iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        if (_hali) (*s)->setHali();
                        (*s)->generatePUSH(out);
                }
        }
}

void StofnSegd::generateNOVAL(ostream& out) const {
        list<Segd*>::const_iterator s;
        for (s = _segdaruna.begin(); s != _segdaruna.end(); s++) {
                list<Segd*>::const_iterator t = s;
                if (++t != _segdaruna.end()) {
                        (*s)->generateNOVAL(out);
                } else {
                        if (_hali) (*s)->setHali();
                        (*s)->generateNOVAL(out);
                }
        }
}
```

```cpp
#include "segdir.h"
#include "smali.h"
#include "stef.h"

using namespace ff;

void NafnSegd::generateAXDX(ostream& out) const {
        if (!_umlykjandiStef->isDefined(_nafn)) {
                reportError("Nafnið \"%s\" er ekki skilgreint.", _nafn.c_str());
        } else {
                symloc loc = _umlykjandiStef->getSymbolLocation(_nafn);
                if (0 == loc.foldun && 0 == loc.offset) {
                        /* innflutt breyta */
                        emit("MOV","BX,%" << quote(_nafn));
                        emit_forskeyti("DS");
                        emit("MOV","DX,[BX]");
                        emit_forskeyti("DS");
                        emit("MOV","AX,[BX+2]");
                } else {
                        if (loc.foldun > 0) {
                                /* assert(nest <= _umlykjandiStef->getNestingLevel() */
                                unsigned int nest = loc.foldun;
                                nest++;    /* fram hjá vendivistf. */
                                nest = nest << 2; /* margf. m. 4 */
                                emit("MOV","BX,[BP+"<<nest<<"]");
                                /* Við höfum fremri addressuna á undan því þá er líklegra
                                   sú seinni verði dregin inn í cache á cpu. */
                                emit_forskeyti("SS");
                                emit("MOV","DX,[BX+" << loc.offset-2 << "]");
                                emit_forskeyti("SS");
                                emit("MOV","AX,[BX+" << loc.offset << "]");
                        } else {
                                emit("MOV","DX,[BP+" << loc.offset-2 << "]");
                                emit("MOV","AX,[BP+" << loc.offset << "]");
                        }
                }
        }
}

void NafnSegd::generatePUSH(ostream& out) const {
        if (!_umlykjandiStef->isDefined(_nafn)) {
                reportError("Nafnið \"%s\" er ekki skilgreint.", _nafn.c_str());
        } else {
                symloc loc = _umlykjandiStef->getSymbolLocation(_nafn);
                if (0 == loc.foldun && 0 == loc.offset) {
                        emit("MOV","BX,%" << quote(_nafn));
                        emit_forskeyti("DS");
                        emit_push("[BX+2]");
                        emit_forskeyti("DS");
                        emit_push("[BX]");
                } else {
                        if (loc.foldun > 0) {
                                /* assert(nest <= _umlykjandiStef->getNestingLevel() */
                                unsigned int nest = loc.foldun;
                                nest++;
                                nest = nest << 2;
                                emit("MOV","BX,[BP+"<<nest<<"]");
                                emit_forskeyti("SS");
                                emit_push("[BX+" << loc.offset << "]");
                                emit_forskeyti("SS");
                                emit_push("[BX+" << loc.offset-2 << "]");
                        } else {
                                emit_push("[BP+" << loc.offset << "]");
                                emit_push("[BP+" << loc.offset-2 << "]");
                        }
                }
        }
}


HeiltoluSegd::HeiltoluSegd(int tala) {
        _tala = tala;
}

HeiltoluSegd::HeiltoluSegd(string& les) {
        bool formerki = false;
        int radix = 10;
        _tala = 0;
        string::iterator i = les.begin();
        while (i != les.end()) {
                char c = *(i++);
                if (c == '-') formerki = !formerki;
                else if (c == '$') radix = 16;
                else switch (c) {
                case '0': case '1': case '2':
                case '3': case '4': case '5':
                case '6': case '7': case '8':
                case '9':
                        _tala *= radix;
                        _tala += (c - '0');
                        break;
                case 'a': case 'b': case 'c':
                case 'd': case 'e': case 'f':
                        _tala <<= 4;
                        _tala |= (c - 'a')+10;
                        break;
                case 'A': case 'B': case 'C':
                case 'D': case 'E': case 'F':
                        _tala <<= 4;
                        _tala |= (c - 'A')+10;
                        break;
                }
        }
}

void HeiltoluSegd::generateAXDX(ostream& out) const {
        emit("MOV", "AX,"<<_tala);
        emit("MOV", "DX,SI");
}

void HeiltoluSegd::generatePUSH(ostream& out) const {
        emit_push(_tala);
        emit_push("SI");
}

void HeiltoluSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        emit("JMP",l(iftrue));
}

void StrengSegd::generateAXDX(ostream& out) const {
        emit("CALL","@@@\\[náístreng\\]");
        out << "\t\"" << _s << "\"\n";
}

void StrengSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        emit("JMP",l(iftrue));
}

FleytitoluSegd::FleytitoluSegd(string& les) {
        bool formerki = false;
        int i = 0;
        while ('-' == les[i++]) formerki = !formerki;
        --i;
        double d = 0.0;
        sscanf(les.substr(i).c_str(), "%lf", &d);
        if (0.0 == d) {
                _ax = 0;
                _dx = 0x0004;
        } else {
                _ax = _dx = 0;
                unsigned char* pd = (unsigned char*) &d;
                unsigned int ieee_exponent = ((pd[7] & 0x7f)<<4) | ((pd[6] & 0xf0) >> 4);
                short exponent = ieee_exponent - 1023;
                _ax = (pd[6] & 0x0f) << 12;
                _ax |= pd[5] << 4;
                _ax |= pd[4] >> 4;
                _dx = (exponent << 5) | (formerki ? 0x10 : 0) | 0x04;
                _dx ^= 0x8000;
        }
}

void FleytitoluSegd::generateAXDX(ostream& out) const {
        emit("MOV","AX," << _ax);
        emit("MOV","DX," << _dx);
}
```

```
void FleytitoluSegd::generatePUSH(ostream& out) const {
        emit_push(_ax);
        emit_push(_dx);
}

void FleytitoluSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        emit("JMP",l(iftrue));
}

void TomaSegd::generateAXDX(ostream& out) const {
        emit("MOV","AX,ES");
        emit("MOV","DX,ES");
}

void TomaSegd::generatePUSH(ostream& out) const {
        emit_push("ES");
        emit_push("ES");
}

void TomaSegd::generateJUMP(ostream& out, int iftrue, int iffalse) const {
        emit("JMP",l(iffalse));
}
```