# Flex & Bison
# (Lex & YACC)

Kristo Heero

# Lühidalt

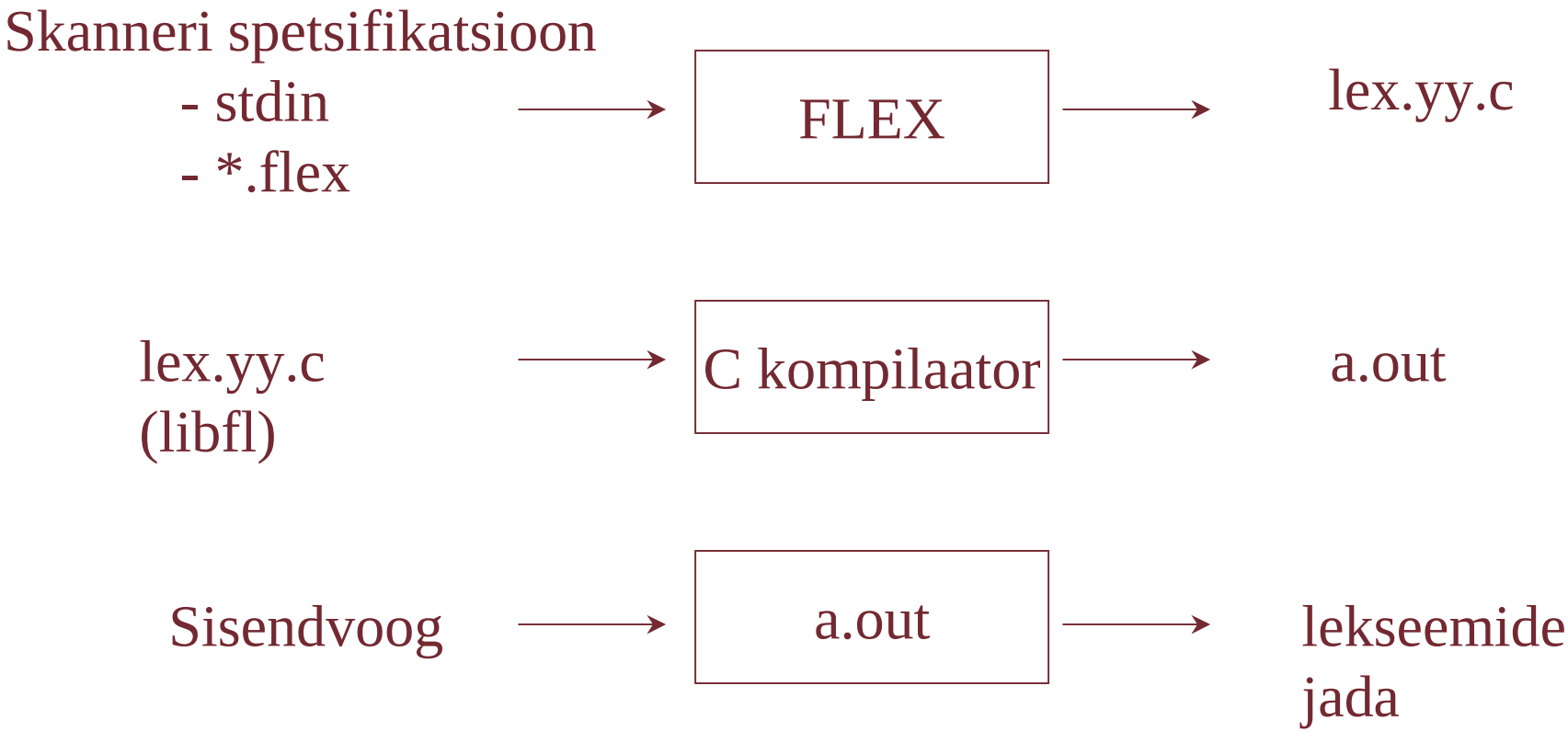
- Flex – Skannerigeneraator. Konverteerib vastavalt koostatud lekseeme kirjeldavad regulaaravaldised C-programmiks, mis suudab tekstist leida kirjeldatud lekseeme (leksiline analüüs).

- Bison – Parserigeneraator. Konverteerib vastavalt kirjeldatud kontekstivaba grammatika C-programmiks, mis seda grammatikat parsida suudab (süntaktiline analüüs).

# Flex

## (Fast Lexical analyzer generator)

# Kuidas kasutada?



# Skanneri spetsifikatsioon

definitsioonid

%%

reeglid

%%

kasutaja kood

# Definitsioonide sektsioon

- Nime definitsioonid kujul: *name definitsion*
- Näiteks:

  DIGIT [0-9]
- Nimi peab algama tähe või “_”-ga, millele võib järgneda tähed, “_”, numbrid ja ka “–”.
- Hiljem saab sellele nimele viidata kujul {nimi}.
- Näiteks:

  FLOAT {DIGIT}+ “.” {DIGIT}*

- Saab lisada C-koodi, mis lisatakse genereeritava skanneri koodile.
- Peab olema %{ ja %} vahel või rea algusesse lisatud tühikuid/tabulaatoreid (taandega).
- Näiteks

```
%{
#include <stdio.h>
%}
    int i = 0;
```

- Ka taandeta kommentaarid kujul /* */ kopeeritakse skanneri koodi.

# Reeglite sektsioon

- Koosneb järjestikustest reeglitest kujul:

  *pattern action*

- Näiteks:

  [\t ]+ putchar(‘ ’);

  [\n] ; //ignore

- Kui action koosneb mitmest reast, siis tuleb ta panna { } vahele.

- Analoogselt definitsiooni sektsiooniga saab siia lisada C-koodi, kuid ainult esimese reegli ette. Teistes kohtades olev kood küll kopeeritakse, kuid nende käsitlus on määramata.

- Kasutatakse lokaalsete muutujate defineerimiseks ning koodi lisamiseks, mida täidetakse alati skanneerimismeetodi (**yylex()**) väljakutsumisel.

# Kasutaja koodi sektsioon

- Selles sektsioonis olev kood lihtsalt kopeeritakse genereeritavasse skannerisse (mittekohustuslik sektsioon).
- Näiteks:

```
int main(void)
{
        yylex();
        return 0;
}
```

# Mustrid läbi regulaaravaldiste

- x — mätsib sümboliga ‘x’
- . — suvaline sümbol (*byte*) v.a. “\n”
- [xyz] — sümbolite klass: ‘x’ | ‘y’ | ‘z’
- [^A-Z] — sümbolite klassi eitus
- r* — 0-mitu r’i (r on regulaaravaldis)
- r + — 1-mitu r’i
- r? — 0-1 r’i
- r{2,5} — 2-5 r’i
- r{2, } — 2-mitu r’i
- r{4} — 4 r’i

- {name} nime definitsiooni viide
- “[xyz]\”foo” string *[xyz]”foo*
- \X kui X on ‘a’, ‘b’, ‘f’, ‘n’, ‘r’, ‘t’, ‘v’, siis ANSI-C interpretatsioon \x vastasel korral kasutatakse eri-tähenduse maha võtmiseks
- \0 NUL sümbol (ASCII kood 0)
- \123 sümbol 8-nd süsteemi koodis
- \x2a sümbol 16-nd koodis
- (r) prioriteedi määramine sulgudega
- r s konkatenatsioon
- r | s r või s

- r/s r, ainult siis, kui talle järgneb s
- ^r r, ainult siis, kui ta on rea alguses
- r$ r, ainult siis, kui ta on rea lõpus

  ekvivalentne “r/\n”. Ettevaatust DOS-i reavahetustega “\r\n”

- <s>r r, kui ollakse starditingimuses s
- <s1, s2, s3>r sama, kui ollakse ühes neist
- <*>r r suvalises starditingimuses

- <<EOF>> EOF
- <s1,s2><<EOF>> EOF, kui vastavas starditing

- Sümbolite klassides ([a-z]) saab kasutada ka spets vahemikke (mis vastavad C funktsioonidele isalpha(), isblank(), ...).
- [:alnum:], [:alpha:], [:blank:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:] [:punct:], [:space:], [:upper:], [:xdigit:]
- Näiteks:

  [a-zA-Z0-9] = [[:alnum:]] = [[:alpha:]0-9]

- Mustris võib esineda ülimalt üks kord järgneva konteksti sümbol ‘/’ või ‘$’.
- ‘^’ (ka <<EOF>>) peab esinema mustri alguses ja $ lõpus, muidu kaotavad nad oma eritähenduse.

# Sisendi mätsimine

- Mätsitakse mustri järgi, mis annab pikima lekseemi.
- Kui kaks (või enam) mustrit mätsivad sama pikka lekseemi, siis valitakse see muster, mis asub flex'i reeglikirjeldustes eespool.

- Mätsiv lekseem kirjutatakse globaalsesse muutujasse char* **yytext** ja tema pikkus globaalsesse muutujasse int **yyleng**. Vastava mustri aktsioon käivitatakse.

- Kui sisend ei mätsu, siis käivitatakse vaike-reegel:  vaadeldud senine sisend prinditakse väljundisse (vaikimisi stdout) ja jätkatakse uue sümboliga sisendist edasi.
- Lihtsaim flex’i sisend on

  %%

  mis lihtsalt kopeerib sisendi sümbolhaaval väljundisse.

# Aktsioonid

- Iga muster reeglite sektsioonis võib omada aktsiooni, mis võib olla suvaline C-kood.
- Kui mustril aktsioon puudub, siis vastava reegli poolt mätsitud lekseem heidetakse lihtsalt kõrvale.
- Näiteks:

```
            int count = 0;
%%
[0-9]+      {       printf("INTEGER");
                    count++;
            }
[  \t]+$    /*eemaldame realõppudest tühikud*/
```

- Kui aktsiooniks on ‘|’, siis see tähendab, et aktsioon on sama, mis järgmisel reeglil.

```
...
dog         |
cat         |
cow         count_animals++;
...
```

- Eksisteerib hulk direktiive, mida saab aktsioonides kasutada.
  - ECHO

    kopeerib yytext'i skanneri väljundisse
  - BEGIN

    määrab starditingimuse (vaatame hiljem)
  - REJECT

    käsib skanneril protsessida "second best" reeglit.

    (Kallis featuur, tõmbab kogu skanneri jõudlust alla!)

Näide:

Sisendi “abc” korral prinditakse “abcaba”

```
%%
a       |
ab      |
abc     ECHO; REJECT;
.|\n     /* sööme ülejäänud sümbolid ära */
```

- **yymore**() – Järgmine mätsiv lekseem lisatakse **yytext**’ile otsa selle asendamise asemel.

- **yyless**(n) – **yytext**’i jäetakse mätsinud lekseemi n esimest sümbolit, ülejäänud pannakse sisendvoogu tagasi, kust neid reskaneeritakse. Kui n = 0 ja aktsioonis ei muudeta reeglite alustamistingimusi, siis võib tekkida lõpmatu tsükkel.

- **unput**(c) – Paneb sümboli ‘c’ tagasi sisendvoo algusesse. Hävitab **yytext** sisu. EOF’i ei saa tagasi panna.

- **input**() – Loeb sisendist järgmise sümboli.

- **yyterminate**() – Katkestab skanneri töö (meetodi **yylex**()) ja tagastab selle väljakutsujale 0’i, st. et “all done”.

  Ka EOF-i korral kutsutakse see meetod vaikimisi välja.

- Aktsioonides võib modifitseerida ka **yytext** ja **yyleng** väärtusi, kuid siis peab sellega arvestama vastavaid direktiive kasutades.

```
%%
“/*”        {
                int c;
                for ( ; ; ) {
                        while ((c = input()) != ‘*’ && c != EOF)
                                ;
                        if (c == ‘*’) {
                                while ((c = input()) == ‘*’)
                                        ;
                                if (c == ‘/’)
                                        break;    //end of comment
                        }

                        if (c == EOF) {
                                error(“EOF in comment”);
                                break;
                        }
                }
        }
```

# Genereeritud skanner

- Faili lex.yy.c luuakse skaneerimismeetod **int yylex**() ja lisaks veel muud vajalikku sodi.
- Nii kui yylex() meetod välja kutsutakse, siis hakatakse skanneerima lekseeme globaalsest sisendfailist **FILE *yyin** (vaikimisi stdin).
- Meetod jätkab seni, kuni saadakse EOF (tagastatakse 0) või tehakse mõnes aktsioonis return.
- Uuest yylex() meetodit välja kutsudes jätkatakse skaneerimist sealt, kus varem pooleli jäädi.

- Meetod **yyrestart**() heidab minema hetke sisendpuhvri ja hakkab uuesti sama sisendfaili skanneerima.
- Meetod **yyrestart**(FILE *new_file) on selle analoog, kus toimub ka sisendi vahetus. NB! Mõlemad meetodid ei initsialiseeri reeglite algustingimust.

- Vaikimisi pannakse väljund (ECHO, ...) faili **FILE *yyout** (vaikimisi stdout).

- Kui skanner jõuab EOF-ni, siis kontrollitakse funktsiooni **yywrap**() (saab ise kirjutada) tagastusväärtust edasi toimimiseks.

- Vaikimisi tagastab see 1, mispeale skanner lõpetab oma töö tagastades selle väljakutsujale 0'i.
- Kui yywrap() tagastab 0'i, siis eeldatakse, et yyin on saanud väärtuseks uue faili ning skaneerimine jätkub.
- Jällegi reeglite starditingimust ei initsialiseeita.

- Kui puuduvad main() ja yywrap() meetodid, siis tuleb skanner linkida libfl'ga, mis need loob.

# Reeglite starditingimused

- Tegemist on skannerile defineeritud olekutega, milles olles ainult vastavalt märgistatud reegleid pruugitakse.
- Skanneri olekud defineeritakse sisendi definitsioonide plokis kujul:

  **%s  state1, state2**

  või

  **%x  state3**

  Esimene on inklusiivne (st. lisaks aktiveeruvad need reeglid, mille ees ei ole stardiolekute märgistust) teine eksklusiivne starditingimus.

- Algolek on INITIAL.
- Olekuid saab muuta kasutades direktiivi BEGIN, mille argumendiks on uue oleku nimi.
- Näide mitmerealise kommentaari töötlemisest:

```
%x comment
%%
        int line_num = 1;
“/*” BEGIN(comment);
<comment>[^*\n]*          //kõik v.a ‘*’ ja ‘\n’
<comment>”*”+[^*/\n]      //sööb ‘*’-d, kui ei järgne ‘/’
<comment>\n               ++line_num;
<comment>”*”+”/”          BEGIN(INITIAL);
```

# Veel Flex’ist

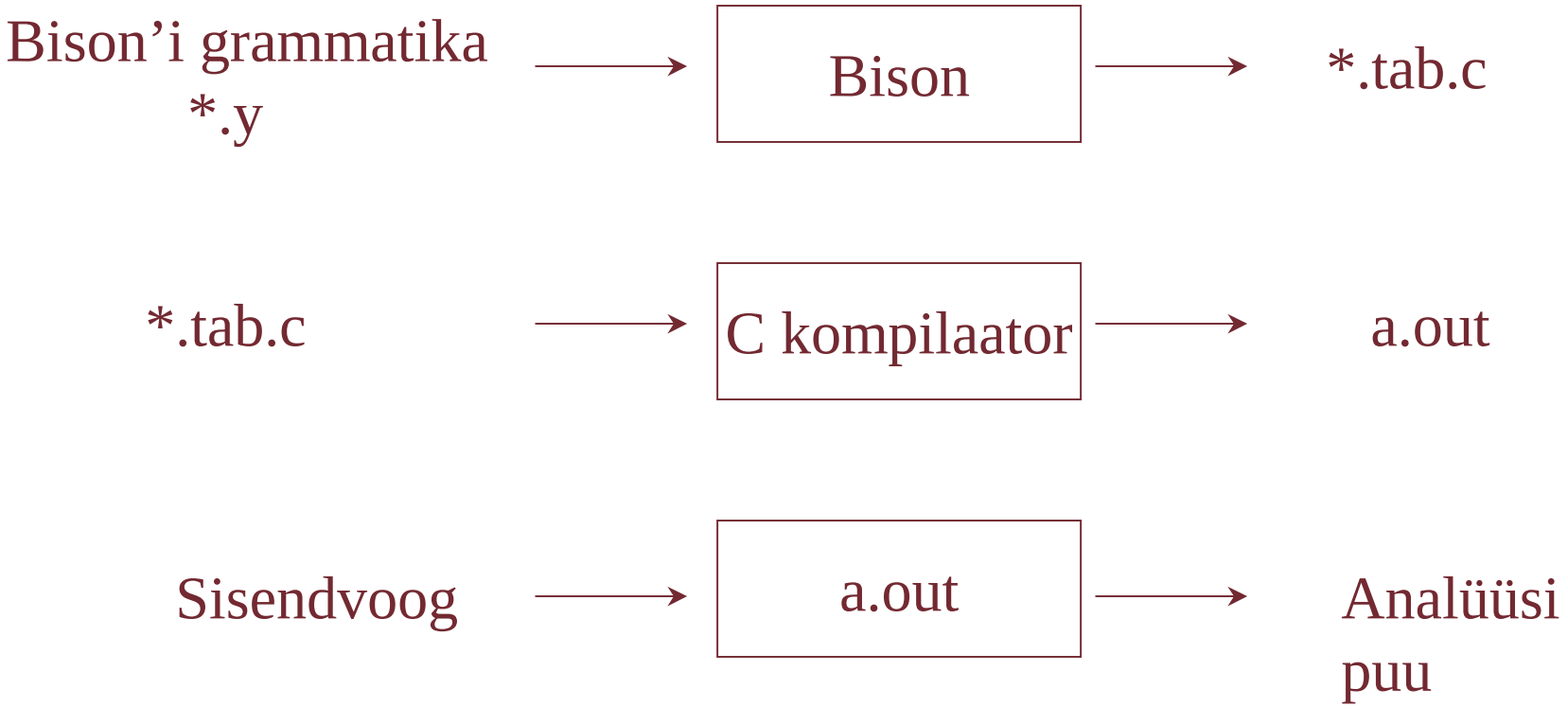
- <<EOF>> kasutades peab aktsioonis olema, kas return, yyterminate() või yyin’ile omistatud uus sisendfail.
- Võimalus kasutada paralleelselt mitut puhvrit.
- Palju optsioone
  - i tõstutundetuks
  - d debugimise moodi (info stderr’i)
  - ...
- Hulga makrosid ja muid kasulikke vidinaid.

# Flex’i integreerimine Bison’iga

- Bison kasutab lekseemide saamiseks meetodit **yytex**(), mis peab lekseemi panema muutujasse **yylval** ning tagastama lekseemi tüübi (int).
- Selleks, et Flex saaks kasutada Bison’i poolt defineeritud lekseemi tüüpe, tuleb include’da Bison’i poolt genereeritud header “**y.tab.h**”.

# Bison

# Kuidas kasutada?



# Bison’i grammatika fail

% {
C-deklaratsioonid
%}

Bison’i deklaratsioonid

%%
Grammatika reeglid
%%

Lisa C-kood

- Kommentaare /* */ võib lisada igale poole.
- Teine %% pole kohustuslik, kui puudub lisa C-kood.
- Vähemasti üks grammatika reegel peab alati olema.

# Grammatikareeglite süntaks

- Kirjutatakse stiilis:

```
expseq:      /* emty */
           |  expseq1
           ;
expseq1:   exp
           |  expseq1 ‘, ‘ exp
           ;
```

- Eelistatud on vasak-rekursioon.

# Keele semantika

- Igal keelekonstruktsioonil on olemas semantiline väärtus (Lekseemide eest hoolitses juba Flex väärtustades yylval muutuja).
- Vaikimisi on semantilise väärtuse hoidmiseks int tüüpi muutuja (#define YYSTYPE int).
- Selleks, et saaks kasutada mitut tüüpi korraga

  tuleb Bison'i deklaratsioonides defineeida %union, kus loetletakse üles kõik vajaminevad väärtused terminalidele ja mitteterminalidele:

```
%union {
        int itype;
        double dtype;
}
```

# Atsioonid

- Aktsioonideks on { ja } vahel olev C-kood, mis võib esineda reegli suvalises kohas ning mis ka täidetakse sinna jõudes.
- C-koodis saab viidata semantilistele väärtustele konstruktsiooniga $n. Grupi enda semantiline väärtus on konstruktsioonis $$.
- exp: ...

  | exp ‘+’ exp

  { $$ = $1 + $3; }

- Kui mõnel grupil puudub aktsioon, siis Bison eeldab vaikimisi, et $$ = $1.
- Kui on deklareeritud %union, siis saab eri väljade poole pöörduda stiilis: $<itype>1.
- Flex’is analoogselt yylval.itype = ...

# Bison’i deklaratsioonid

- Kõik lekseemi-tüübi nimed (v.a. ühe-sümbolilised lekseemid: ‘+’, ‘\n’, ...) peavad olema deklareeritud %token abil:

  %token  ID  NUMBER  STRING ...

  %token <itype> NUMBER

- Bison võtmega –d genereeribki nendest header faili ***.tab.h**, mida Flex kasutab.
- Vaikimisi on grammatika algussümboliks grammatika esimese reegli mitteterminal. Selle omavoliliseks määramiseks saab kasutada deklaratsiooni: **%start symbol**.

# Parseri interface

- Bison genereerib meetodi **int yyparse**(). Selle meetodi väljakutsumine alustabki sisendi parsimist.
- Meetod tagastab 0, kui parsimine oli edukas ning vastasel juhul 1.
- Saab kasutada makrosid YYACCEPT ja YYABORT, mis annavad analoogselt sama tulemuse.
- Parsimisvea korral kutsutakse välja meetod **yyerror**(char *), mis tuleb endal kirjutada.

# Parseri abstraktne algoritm

- Vasakult paremale hakatakse sisendist lekseeme nihutama stack'ile.
- Kui stackil n viimast elementi on mingi produktsiooni parem pool, siis teostatakse redutseerimine (stacki n elementi asendatakse produktsiooni vasaku poolega).
- Ja nii kuni sisend on otsas ning stackile on jäänud lõpuks algusproduktsiooni vasak pool.

- Bison’i genereeritud parser, aga ei redutseeri alati kohe, kui selleks võimalus avaneb. Sest selline käitumine on paljude keelte korral ebaadekvaatne.

- Seetõttu vaatab parser otsustamiseks ette järgmise lekseemi (look ahead 1).

# Nihutamine/Redutseerimine konflikt

- Näide

```
if_stmt:
            IF expr THEN stmt
        |   IF expr THEN stmt ELSE stmt
        ;
```

- Parseri vaikekäitumine on see, et ta eelistab nihutamist redutseerimisele.
- Deklaratsiooniga **%expext n**, saab öelda antud konfliktide piiri, alates millest konflikte teavitatakse.

- Näide:

```
expr:       expr '–' expr
        |   expr '*' expr
        |   '–' exp %prec UMINUS
        ;
```

“x + y * z”, “x – y – z”

- Deklaratsioonidega %left, %right, %nonassoc abil saab määrata assotsiatiivsust ning nende paiknemise järjekorra järgi ka prioriteeti:

```
%left '+' '–'
%left '*' '\'
%left UMINUS
```

# Redutseerimine/Redutseerimine konflikt

- Näide:

  exp1: exp2
  | ID
  ;

  exp2: ID; “name”
- Grammatika ei ole pööratav, konflikt tuleks elimineerida.
- Parseri vaikekäitumine on see, et valitakse nimekirjas esimene produktsioon.

# Veel

- Optsionid:
  - t debugimis rešiimi (info stderr’i)
  - d header fail flex’ile
  - ...
- Igasugu kasulikke pudinaid on veel.