Цель работы

Целью данной работы является приобретение навыков использования генератора синтаксических анализаторов bison.

Индивидуальный вариант

Объявления типов и констант в Паскале: В `record`'е точка с запятой *разделяет* поля и после `case` дополнительный `end` не ставится. См. https://bernd-oppolzer.de/PascalReport.pdf, третья с конца страница.

```
Type
  Coords = Record x, y: INTEGER end;
Const
  MaxPoints = 100;
type
  CoordsVector = array 1..MaxPoints of Coords;

(* графический и текстовый дисплеи *)
const
  Heigh = 480;
  Width = 640;
  Lines = 24;
  Columns = 80;
type
  BaseColor = (red, green, blue, highlited);
  Color = set of BaseColor;
  GraphicScreen = array 1..Heigh of array 1..Width of Color;
  TextScreen = array 1..Lines of array 1..Columns of
    record
      Symbol : CHAR;
      SymColor : Color;
      BackColor : Color
    end;

{ определения токенов }
TYPE
  Domain = (Ident, IntNumber, RealNumber);
  Token = record
    fragment : record
      start, following : record
        row, col : INTEGER
      end
    end;
    case tokType : Domain of
      Ident : (
        name : array 1..32 of CHAR
```

```
      );
      IntNumber : (
        intval : INTEGER
      );
      RealNumber : (
        realval : REAL
      )
  end;

  Year = 1900..2050;

  List = record
    value : Token;
    next : ^List
  end;
```

Реализация

Makefile

```
TESTS = $(shell ls test/*)
SRC_DIR         = src${w}
BUILD_DIR       = build${w}
INCLUDE_DIR     = include
PARSER_SRC      = ${SRC_DIR}/parser${w}.y
LEXER_SRC       = ${SRC_DIR}/lexer${w}.l
LEXER_OUT       = ${BUILD_DIR}/lex${w}.yy.c
PARSER_OUT_H    = ${BUILD_DIR}/y${w}.tab.h
PARSER_OUT      = ${BUILD_DIR}/y${w}.tab.c
PROG            = ${BUILD_DIR}/prog${w}
TEST_N          = ${TEST_DIR}/test_${n}

TEST_DIR        = test

build: flex bison gcc

flex: ${LEXER_SRC}
    flex  -o${LEXER_OUT} ${LEXER_SRC}

bison: ${PARSER_SRC}
    bison ${PARSER_SRC} -H${PARSER_OUT_H} -o${PARSER_OUT} -Wconflicts-
sr -Wconflicts-rr -Wcounterexamples;

gcc: ${LEXER_OUT} ${PARSER_OUT} ${PARSER_OUT_H}
    gcc -o ${PROG} ${BUILD_DIR}/*.c -I${BUILD_DIR} -I${INCLUDE_DIR} -w

run: input
    ./${PROG} input ${ARGS}
```

```
crun: build run

tests:
    ./${PROG} ${TEST_N} ${ARGS} > out${w}

ctests: build tests

# make ctests n=1 — запуск теста 1
# make crun — запуск на файле input
# -k — не выравнивать регистр ключевых слов
# -w — не расставлять необязательные точки с запятой
```

config.h

```
#ifndef CONFIG_H
#define CONFIG_H

#define DEBUG false  //  трассировка

#define MARGIN              env[0]
#define KEYWORDS_FORMAT     env[1]
#define WEAK                env[2]
#define DEV_MODE            env[3]

#define trace(str) DEBUG && printf("%s\n", str);

#define MAX_STRING_LENGTH 1000

#endif /* CONFIG_H */
```

format.h

```
#ifndef FORMAT_H
#define FORMAT_H

#include "config.h"

#define margin()                        \
    for (int i = 0; i < MARGIN; ++i) \
        printf("\t");

#define inc MARGIN++;

#define dec MARGIN--;

#define n printf("\n");

#define s printf(" ");

#define m margin();
```

```
#define nd inc n m; // \n down
#define nu dec n m; // \n up
#define ns     n m; // \n straight

#define p(str) \
    printf("%s", str);

#define alt(str, temp) printf("%s", KEYWORDS_FORMAT ? temp : str);

#define f(arg) free(arg);

#define TYPE_TEMP       "Type"
#define NIL_TEMP        "Nil"
#define PACKED_TEMP     "Packed"
#define OF_TEMP         "Of"
#define ARRAY_TEMP      "Array"
#define FILE_TEMP       "File"
#define SET_TEMP        "Set"
#define RECORD_TEMP     "Record"
#define END_TEMP        "End"
#define VAR_TEMP        "Var"
#define CASE_TEMP       "Case"
#define CONST_TEMP      "Const"
#define TYPE_TEMP       "Type"

#endif /* FORMAT_H */
```

lexer.h

```
#ifndef LEXER_H
#define LEXER_H

#include <stdbool.h>
#include <stdio.h>

#ifndef YY_TYPEDEF_YY_SCANNER_T
#define YY_TYPEDEF_YY_SCANNER_T
typedef void *yyscan_t;
#endif /* YY_TYPEDEF_YY_SCANNER_T */

#define NEWLINE 0
#define TAB 1
#define SLASH 2
#define QUOTE 3

char spec_symbols_prod[];
char spec_symbols_dev[];

struct Extra {
```

```
    bool continued;
    char* spec_symbols;
    int string_index;
    int cur_line;
    int cur_column;
    long* env;
};

void init_scanner(FILE *input, yyscan_t *scanner, struct Extra *extra,
long* flags);
void destroy_scanner(yyscan_t);

#endif /* LEXER_H */

// * [ ] ( ) " \ { } специальные символы, требующие экранирования
```

lexel.l

```
%option reentrant noyywrap bison-bridge bison-locations
%option extra-type="struct Extra *"

/* Подавление предупреждений для -Wall */
%option noinput nounput

%{

#include <stdio.h>
#include <stdlib.h>
#include "lexer.h"
#include "config.h"

#include "y.tab.h"  /* файл генерируется Bison'ом */

#define margin()                        \
    for (int i = 0; i < yyextra->MARGIN; ++i) \
        printf("\t");

#define m margin();

#define reflect(arg) { \
    yylval->string = copy(yytext); \
    return arg; \
} \

#define YY_USER_ACTION \
  { \
    int i; \
    struct Extra *extra = yyextra; \
    if (! extra->continued ) { \
      yylloc->first_line = extra->cur_line; \
```

```
      yylloc->first_column = extra->cur_column; \
    } \
    extra->continued = false; \
    for (i = 0; i < yyleng; ++i) { \
      if (yytext[i] == '\n') { \
        extra->cur_line += 1; \
        extra->cur_column = 1; \
      } else { \
        extra->cur_column += 1; \
      } \
    } \
    yylloc->last_line = extra->cur_line; \
    yylloc->last_column = extra->cur_column; \
}

void yyerror(YYLTYPE *loc, yyscan_t scanner, long env[26], const char
*message) {
    printf("Error (%d,%d): %s\n", loc->first_line, loc->first_column,
message);
}

void error(YYLTYPE *loc, yyscan_t scanner, const char *message) {
    printf("Error (%d,%d): %s\n", loc->first_line, loc->first_column,
message);
}

char spec_symbols_prod[] = {
    '\n', '\t', '\\', '\'',
};

char spec_symbols_dev[] = {
    'N', 'T', '\\', '\'',
};

char* copy(const char* s) {
    char* res = (char*)malloc(strlen(s));
    strcpy(res, s);
    return res;
}

%}

%x STRING_STATE

%%

[\r\t\n ]+

\(      return LPAREN;
```

```
\)      return RPAREN;
,       return COMMA;
\.\.    {trace(".."); return POINTS;};
\^      return CARET;
\+      return PLUS;
-       return MINUS;
\[      return LBRACKET;
\]      return RBRACKET;
=       {trace(yytext); return EQUAL;};
:       return COLON;
;       return SEMICOLON;

[Nn][Ii][Ll]                reflect(NIL);
[Pp][Aa][Cc][Kk][Ee][Dd]    reflect(PACKED);
[Oo][Ff]                    reflect(OF);
[Aa][Rr][Rr][Aa][Yy]        reflect(ARRAY);
[Ff][Ii][Ll][Ee]            reflect(FILE_);
[Ss][Ee][Tt]                reflect(SET);
[Rr][Ee][Cc][Oo][Rr][Dd]    reflect(RECORD);
[Ee][Nn][Dd]                reflect(END);
[Tt][Yy][Pp][Ee]            reflect(TYPE);
[Vv][Aa][Rr]                reflect(VAR);
[Cc][Aa][Ss][Ee]            reflect(CASE);
[Cc][Oo][Nn][Ss][Tt]        reflect(CONST);

\(\*.*\\*\) // TODO: оставлять в коде комментарии
\{.*\}

[a-zA-Z]*                                       {
                                                      yylval->string =
copy(yytext);
                                                      trace(yytext);
                                                      return IDENTIFIER;
                                                };
[0-9]+(\.[0-9]+)?([eE][-+]?[0-9]+)?             {
                                                      yylval->string =
copy(yytext);
                                                      trace(yytext);
                                                      return
UNSIGNED_NUMBER;
                                                };

' {
    BEGIN(STRING_STATE);
    yyextra->continued=1;
    yyextra->string_index=0;
    yylval->string = (char*)malloc(MAX_STRING_LENGTH);
}

<STRING_STATE>\\n {
```

```
    yylval->string[yyextra->string_index++] = yyextra-
>spec_symbols[NEWLINE];
}

<STRING_STATE>\\t {
    yylval->string[yyextra->string_index++] = yyextra-
>spec_symbols[TAB];
}

<STRING_STATE>\\' {
    yylval->string[yyextra->string_index++] = yyextra-
>spec_symbols[QUOTE];
}

<STRING_STATE>\\\\ {
    yylval->string[yyextra->string_index++] = yyextra-
>spec_symbols[SLASH];
}

<STRING_STATE>[\n] {
    error(yylloc, yyscanner, "newline in string literal");
    return 0;
}

<STRING_STATE><<EOF>> {
    error(yylloc, yyscanner, "eof in string");
    return 0;
}

<STRING_STATE>[^'\\\n]* {
    int len = strlen(yytext);
    strncpy(&yylval->string[yyextra->string_index], yytext, len);
    yyextra->string_index += len;
    yyextra->continued = 1;
}

<STRING_STATE>' {
    BEGIN(0);
    return STRING;
}

%%

void init_scanner(FILE *input, yyscan_t *scanner, struct Extra *extra,
long* env) {
    extra->continued = false;
    extra->string_index = 0;
    extra->cur_line = 1;
    extra->cur_column = 1;
```

```
    extra->env = env;

    if (DEV_MODE) {
        extra->spec_symbols = spec_symbols_dev;
    } else {
        extra->spec_symbols = spec_symbols_prod;
    }

    yylex_init(scanner);
    yylex_init_extra(extra, scanner);
    yyset_in(input, *scanner);
}

void destroy_scanner(yyscan_t scanner) {
    yylex_destroy(scanner);
}
```

- parser.y

```
%{
    #include <stdio.h>
    #include <string.h>
    #include "lexer.h"
    #include "config.h"
    #include "format.h"
%}

%define api.pure
%locations
%lex-param {yyscan_t scanner}  /* параметр для yylex() */
/* параметры для yyparse() */
%parse-param {yyscan_t scanner}
%parse-param {long env[26]}

%token LPAREN
%token RPAREN
%token COMMA;
%token POINTS
%token CARET;
%token PLUS;
%token MINUS;
%token LBRACKET;
%token RBRACKET;
%token EQUAL;
%token COLON;
%token SEMICOLON;

%token <string> NIL
%token <string> PACKED
%token <string> OF
```

```
%token <string> ARRAY
%token <string> FILE_
%token <string> SET
%token <string> RECORD
%token <string> END
%token <string> VAR
%token <string> CASE
%token <string> CONST
%token <string> TYPE

%union {
    char* string;
    int margin;
}

%token <string> IDENTIFIER;
%token <string> UNSIGNED_NUMBER;
%token <string> STRING;
%token <string> COMMENT;

%{
int yylex(YYSTYPE *yylval_param, YYLTYPE *yylloc_param, yyscan_t
scanner);
void yyerror(YYLTYPE *loc, yyscan_t scanner, long env[26], const char
*message);
%}

%%

program: blocks { ns }
    ;

blocks: blocks { ns } block | block
    ;

block: type_block | constant_block
    ;

/* Определение константы */

constant_block: const_ { nd } constant_definition_list
possible_semicolon { dec }
    ;

constant_definition_list: constant_definition_list _semicolon_ { ns }
constant_definition | constant_definition
    ;

constant_definition: _ident_ { s } _equal_ { s } constant
```

```
    ;

constant: sign unsigned_constant_number | unsigned_constant_number |
_string_
    ;

unsigned_constant_number: _unsigned_number_ | constant_ident
    ;

constant_ident: _ident_

sign: _plus_ | _minus_
    ;

/* sing_: / пусто / | sign  почему-то не работает в начале правила
    ; */

/* Определение константы */

/* Определение типа */
type_block: _type_ { nd } type_definition_list possible_semicolon
{ dec }
    ;

type_definition_list: type_definition_list _semicolon_ { ns }
type_definition | type_definition
    ;

type_definition: _ident_ { s } _equal_ { s } type
    ;

type: simple_type | pointer_type | structured_type
    ;

simple_type: scalar_type | subrange_type | type_ident
    ;

scalar_type: _lparen_ ident_list _rparen_
    ;

ident_list: ident_list _comma_ { s } _ident_ | _ident_
    ;

subrange_type: constant { s } _points_ { s } constant
    ;

structured_type: _packed_ { s } unpacked_structured_type |
unpacked_structured_type
    ;
```

```
unpacked_structured_type: array_type | record_type | file_type |
set_type
    ;

array_type: _array_ { s } _lbracket_ index_type_list _rbracket_ { s }
_of_ { s } component_type
    ;

index_type_list: index_type_list _comma_ { s } index_type | index_type
    ;

index_type: simple_type
    ;

component_type: type
    ;

/*  Тип записи */

record_type: _record_ { nd } field_list { nu } _end_
    ;

field_list: fixed_part | fixed_part _semicolon_ { ns } variant_part |
variant_part
    ;

fixed_part: fixed_part _semicolon_ { ns } record_section |
record_section
    ;

record_section: field_ident_list _colon_ { s } type
    ;

field_ident_list: field_ident_list _comma_ { s } field_ident |
field_ident
    ;

field_ident: _ident_
    ;

variant_part: _case_ { s } tag_field { s } _colon_ { s } type_ident
{ s } _of_ { nd } variant_list { dec }
    ;

variant_list: variant_list _semicolon_ { ns } variant | variant
    ;

variant: case_label_list _colon_ { s } _lparen_ { nd } field_list { nu
```

```
} _rparen_ | case_label_list
    ;

case_label_list: case_label_list _comma_ { s } case_label | case_label
    ;

case_label: unsigned_constant
    ;

unsigned_constant: _unsigned_number_ | _string_ | _nil_ |
constant_ident
    ;

/*  Тип записи */

set_type: _set_ { s } _of_ { s } base_type
    ;

base_type: simple_type
    ;

file_type: _file_ { s } _of_ { s } base_type  type
    ;

pointer_type: _caret_ type_ident
    ;

tag_field: _ident_
    ;

type_ident: _ident_
    ;

/* Определение типа */

_ident_: IDENTIFIER { p($1) } { f($1) }
    ;

_string_: STRING { p($1) } { f($1) }
    ;

_unsigned_number_: UNSIGNED_NUMBER { p($1)} { f($1) }
    ;

/* Ключевые слова */

_type_: TYPE { alt($1, TYPE_TEMP) } { f($1) }
    ;
```

```
const_: CONST { alt($1, CONST_TEMP) } { f($1) }
    ;

_packed_: PACKED { alt($1, PACKED_TEMP) } { f($1) }
    ;

_array_: ARRAY { alt($1, ARRAY_TEMP) } { f($1) }
    ;

_of_: OF { alt($1, OF_TEMP) } { f($1) }
    ;

_record_: RECORD { alt($1, RECORD_TEMP) } { f($1) }
    ;

_end_: END { alt($1, END_TEMP) } { f($1) }
    ;

_case_: CASE { alt($1, CASE_TEMP) } { f($1) }
    ;

_nil_: NIL { alt($1, NIL_TEMP) } { f($1) }
    ;

_set_: SET { alt($1, SET_TEMP) } { f($1) }
    ;

_file_: FILE_ { alt($1, FILE_TEMP) } { f($1) }
    ;

/* Ключевые слова */

_semicolon_: SEMICOLON { p(";") }
    ;

possible_semicolon: /* пусто */ { !WEAK && p(";") }
{trace("semicolon_false")}
    | _semicolon_
{trace("semicolon_true")}
    ;

_equal_: EQUAL { p("=") }
    ;

_plus_: PLUS { p("+") }
    ;

_minus_: MINUS { p("-") }
```

```
    ;

_lparen_: LPAREN { p("(") }
    ;

_rparen_: RPAREN { p(")") }
    ;

_comma_: COMMA { p(",") }
    ;

_points_: POINTS { p("..") }
    ;

_lbracket_: LBRACKET { p("[") }
    ;

_rbracket_: RBRACKET { p("]") }
    ;

_colon_: COLON { p(":") }
    ;

_caret_: CARET { p("^") }
    ;

%%

int main(int argc, char *argv[]) {
    FILE *input = 0;
    long env[26] = { 0 };
    yyscan_t scanner;
    struct Extra extra;

    if (argc > 1) {
        input = fopen(argv[1], "r");
    } else {
        printf("No file in command line, use stdin\n");
        input = stdin;
    }

    KEYWORDS_FORMAT = 1;

    char* arg;
    for (int i = 0; i < argc; ++i) {
        arg = argv[i];
        if (!strcmp(arg, "-d")) {
            DEV_MODE = 1;
        }
```

```
        if (!strcmp(arg, "-k")) {
            KEYWORDS_FORMAT = 0;
        }
        if (!strcmp(arg, "-w")) {
            WEAK = 1; // отвечает только за расстановку необязательных
точек с запятой
        }
    }

    init_scanner(input, &scanner, &extra, env);
    yyparse(scanner, env);
    destroy_scanner(scanner);

    if (input != stdin) {
        fclose(input);
    }

    return 0;
}
```

Тестирование

Входные данные

```
Type Coords = Record x, y: INTEGER end; cOnst MaxPoints = 100 type
CoordsVector = array [1..MaxPoints] of Coords; (* графический и
текстовый дисплеи *) const Heigh = 480; Width = 640; Lines = 24;
Columns = 80; type BaseColor = (red, green, blue, highlited); Color =
set of BaseColor; GraphicScreen = array [1..Heigh] of array [1..Width]
of Color; TextScreen = array [1..Lines] of array [1..Columns] of
record Symbol : CHAR; SymColor : Color; BackColor : Color end;
{ определения токенов } TYPE Domain = (Ident, IntNumber, RealNumber);
Token = record fragment : record start, following : record row, col :
INTEGER end end; case tokType : Domain of Ident : ( name : array
[1..32] of CHAR ); IntNumber : ( intval : INTEGER ); RealNumber :
( realval : REAL ) end; Year = 1900..2050; List = record value :
Token; next : ^List end;
```

Вывод на stdout

```
flex  -obuild/lex.yy.c src/lexer.l
bison src/parser.y -Hbuild/y.tab.h -obuild/y.tab.c -Wconflicts-sr -
Wconflicts-rr -Wcounterexamples;
gcc -o build/prog build/*.c -Ibuild -Iinclude -w
./build/prog input
Type
    Coords = Record
        x, y: INTEGER
    End;
```

```
Const
    MaxPoints = 100;
Type
    CoordsVector = Array [1 .. MaxPoints] Of Coords;
Const
    Heigh = 480;
    Width = 640;
    Lines = 24;
    Columns = 80;
Type
    BaseColor = (red, green, blue, highlited);
    Color = Set Of BaseColor;
    GraphicScreen = Array [1 .. Heigh] Of Array [1 .. Width] Of Color;
    TextScreen = Array [1 .. Lines] Of Array [1 .. Columns] Of Record
        Symbol: CHAR;
        SymColor: Color;
        BackColor: Color
    End;
Type
    Domain = (Ident, IntNumber, RealNumber);
    Token = Record
        fragment: Record
            start, following: Record
                row, col: INTEGER
            End
        End;
        Case tokType : Domain Of
            Ident: (
                name: Array [1 .. 32] Of CHAR
            );
            IntNumber: (
                intval: INTEGER
            );
            RealNumber: (
                realval: REAL
            )
    End;
    Year = 1900 .. 2050;
    List = Record
        value: Token;
        next: ^List
    End;
```

Вывод

В ходе лабораторной работы реализовал **реентрантные** лексический и синтаксический анализаторы с использованием генераторов flex и bison. Научился описывать грамматику в синтаксисе bison и отлаживать её с помощью предупреждений bison о конфликтах. Освежил навык написания

лексического анализатора на flex. Также осознал огромную выразительную силу макросов языка C, которые позволили удобно производить отладочный вывод в процессе разработки и лаконично описать в правилах bison семантические действия по форматированию. В целом flex/bison сильно облегчают написание анализаторов, так как обладают понятным синтаксисом описания правил и позволяют использовать знакомый и удобный C для описания семантики.

Замечания: - Помимо указанных в работе требований к сильному форматтеру, реализация может как унифицировать стиль ключевых слов, так и оставлять их без изменения (ключ -k) - Пропускать некоторые подстроки входа, выдавая их в выход без изменений можно с помощью макроса ECHO в описании действий lex - bison перед выполнением семантического действия после терминала/нетерминала сначала получает следующую лексему (эта особенность не даёт легко преобразовать сильный форматтер в слабый, так как мешает правильной табуляции)