Лабораторная работа 3.2 «Форматтер исходных текстов»

23 мая 2023 г.

Самохвалова Полина, ИУ9-62Б

Цель работы

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

Индивидуальный вариант

Сильный форматтер, РЕФАЛ-5.

Реализация

lexer.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 "parser.tab.h"  /* файл генерируется Bison’ом */

#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);
}

%}

%%

[\r\t\n ]+

\=   return '=';
\(   return '(';
\)   return ')';
\{   return '{';
\}   return '}';
\,   return ',';
\;   return ';';
\:   return ':';
\<   return '<';
\>   return '>';

[0-9]+  {
    yylval->number = atoi(yytext);
    return NUMBER;
}

[ste]\.[0-9]+|[ste]\.[A-Za-z][A-Za-z_0-9]* {
    yylval->indentifier = yytext;
```

```
    return VAR;
}

[A-Za-z][A-Za-z_0-9]* {
    yylval->indentifier = yytext;
    return IDENT;
}

\$EXTERN|\$EXTRN|\$EXTERNAL   {
    yylval->indentifier = yytext;
    return EXTERN;
}

\$ENTRY   {
    yylval->indentifier = yytext;
    return ENTRY;
}

\'.\'   {
    yylval->indentifier = yytext;
    return SYMBOL;
}

\'[^\']*\'   {
    yylval->indentifier = yytext;
    return STRING;
}

.

%%

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

    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 <stdlib.h>
#include <string.h>
#include "lexer.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]}

%union {
    char* indentifier;
    char variable;
    long number;
}

%token <indentifier> IDENT SYMBOL STRING VAR EXTERN ENTRY
%token <number> NUMBER

%{

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

void print_newline_with_offset(long* env) {
    printf("\n");
    env[2] = 0;
    long k = env[1];
    for (int i = 0; i < k; i++) {
        printf("    ");
        env[2] += 4;
    }
}

void check_len(long len, long* env) {
    if ((len + env[2] > env[0]) &&
        (env[1] * 4 + len <= env[0])) {
        print_newline_with_offset(env);
```

```
    }
}

int number_len(long num) {
    char str[20];
    int len = snprintf(str, sizeof(str), "%ld", num);
    return len;
}

void print_str(char* str, long* env) {
    int len = strlen(str);
    check_len(len, env);
    printf("%s", str);
    env[2] += len;
}

void print_num(long num, long* env) {
    int len  = number_len(num);
    check_len(env[2] + len, env);
    printf("%ld", num);
    env[2] += len;
}

%}

%%

program:
      general program
    |
    ;

general:
      extern
    | function
    | ';' { print_str(";", env); print_newline_with_offset(env);
    print_newline_with_offset(env); }
    ;

extern:
      EXTERN { print_str($EXTERN, env); print_str(" ", env); }
      function_name function_names_list
    ;

function_names_list:
      ',' { print_str(",", env); print_str(" ", env); }
```

```
      function_name function_names_list
    |
    ;

function:
      ENTRY { print_str($ENTRY, env); print_str(" ", env); }
      function_name body
    | function_name body
    ;

function_name:
      IDENT { print_str($IDENT, env); print_str(" ", env); }
    ;

body:
      '{' { print_str("{", env); env[1] += 1; } sentences '}'
       {
          env[1] -= 1;
          print_newline_with_offset(env);
          print_str("}", env);
          print_newline_with_offset(env);
          print_newline_with_offset(env);
        }
    ;

sentences:
      sentence sentences_list
    ;

sentence:
      { print_newline_with_offset(env); } pattern
      { print_str(" ", env); } pattern_right
      | { print_newline_with_offset(env); } pattern_right
    ;

pattern_right:
      '=' { print_str("=", env); print_str(" ", env); } result
    | ',' { print_str(",", env); print_str(" ", env); } result
    ':' { print_str(":", env); print_str(" ", env); }
    sentence_or_body
    ;

sentence_or_body:
      sentence
    | body
    ;
```

```
sentences_list:
      ';' { print_str(";", env); } sentences_le_one sentences_list
    |
    ;

sentences_le_one:
      sentence
    |
    ;

pattern:
      pattern_term patterns
    ;

patterns:
      { print_str(" ", env); } pattern_term patterns
    |
    ;

pattern_term:
      common
    | '(' { print_str("(", env); }  pattern ')' { print_str(")",
      env); }
    ;

common:
      IDENT { print_str($IDENT, env); }
    | NUMBER { print_num($NUMBER, env); }
    | SYMBOL { print_str($SYMBOL, env); }
    | STRING { print_str($STRING, env); }
    | VAR { print_str($VAR, env); }
    ;

result:
      result_term results
    |
    ;

results:
      { print_str(" ", env); } result_term results
    |
    ;

result_term:
      common
```

```
    | '(' { print_str("(", env); } result ')'
    { print_str(")", env); }
    | '<' { print_str("<", env); } function_name result '>'
    { print_str(">", env); }
    ;

%%

int main(int argc, char *argv[]) {
    FILE *input = 0;
    long env[26] = { 0 };
    env[0] = 0;
    env[1] = 0;
    env[2] = 0;
    yyscan_t scanner;
    struct Extra extra;
    if (argc > 1) {
        input = fopen(argv[1], "r");
        env[0] = atoi(argv[2]);
    } else {
        printf("No file in command line, use stdin\n");
        input = stdin;
    }
    init_scanner(input, &scanner, &extra);
    yyparse(scanner, env);
    destroy_scanner(scanner);
    if (input != stdin) {
        fclose(input);
    }
    return 0;
}
```

Тестирование

Входные данные

```
$ENTRY Go{= <Prout <Calc 100 '-' 70 '-' 50>>}

Calc
{e.1 '+' e.2=<Add <Calc e.1> <Calc e.2>>;
e.1 '-' e.2 '-' e.3 = <Calc <Calc e.1 '-' e.2> '-' e.3>;
e.1 '-' e.2= <Sub <Calc e.1> <Calc e.2>>;
e.1 '*' e.2 = <Mul <Calc e.1> <Calc e.2>>;
e.1 '/' e.2 '/' e.3 = <Calc <Calc e.1 '/' e.2> '/' e.3>;
e.1 '/' e.2 = <Div <Calc e.1> <Calc e.2>>;
```

```
(e.1) = <Calc e.1>; s.1=s.1;}
```

Вывод на `stdout`

```
$ENTRY Go {
    = <Prout <Calc 100 '-' 70 '-' 50>>
}

Calc {
    e.1 '+' e.2 = <Add <Calc e.1> <Calc e.2>>;
    e.1 '-' e.2 '-' e.3 = <Calc <Calc e.1 '-' e.2> '-' e.3>;
    e.1 '-' e.2 = <Sub <Calc e.1> <Calc e.2>>;
    e.1 '*' e.2 = <Mul <Calc e.1> <Calc e.2>>;
    e.1 '/' e.2 '/' e.3 = <Calc <Calc e.1 '/' e.2> '/' e.3>;
    e.1 '/' e.2 = <Div <Calc e.1> <Calc e.2>>;
    (e.1) = <Calc e.1>;
    s.1 = s.1;
}
```

Вывод

В результате выполнения лабораторной работы был приобретен навык использования генератора синтаксических анализаторов bison, был разработан сильный форматтер языка программирования РЕФАЛ-5.