Лабораторная работа № 1.5 «Порождение лексического анализатора с помощью flex»

20 марта 2024 г.

Кирилл Киселёв, ИУ9-61Б

Цель работы

Целью данной работы является изучение генератора лексических анализаторов flex.

Индивидуальный вариант

- Комментарии: начинаются с «//» и продолжаются до окончания строки текста.
- Идентификаторы: любой текст, не содержащий «/» и ограниченный символами «/».
- Ключевые слова: «/while/», «/do/», «/end/».

Реализация

```
%option noyywrap bison-bridge bison-locations


%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define IDENT_ARRAY_SIZE 1024
#define MAX_IDENT_SIZE 128
#define ERR_ARRAY_SIZE 1024


#define COMMENTS_ARRAY_SIZE 1024
```

```
char* ident_array[IDENT_ARRAY_SIZE];
size_t ident_array_len = 0;

enum DOMAIN_TAG {
  _EOF,
  IDENT,
  WHILE,
  DO,
  END,
  NUMBER,
  COMMENT,
};

int add_ident(char* ident) {

  char* tmp = (char*)calloc(MAX_IDENT_SIZE, sizeof(char));
  strcpy(tmp, ident);

  if (ident_array_len > IDENT_ARRAY_SIZE) {
    return -1;
  }

  for (int i = 0; i < ident_array_len; i++) {
    if (strcmp(tmp, ident_array[i]) == 0) {
      return i;
    }
  }

  ident_array[ident_array_len] = tmp;
  return ident_array_len++;
}

int find_ident(char* ident) {
  for (int i = 0; i < ident_array_len; i++) {
    if (strcmp(ident, ident_array[i]) == 0) {
      return i;
    }
  }

  return -1;
}

char* tag_to_str(enum DOMAIN_TAG d) {
```

```
  switch (d)
  {
    case _EOF:
      return "EOF";
    case IDENT:
      return "IDENT";
    case WHILE:
      return "WHILE";
    case DO:
      return "DO";
    case END:
      return "END";
    case COMMENT:
      return "COMMENT";
    case NUMBER:
      return "NUMBER";
    default:
      return "UNDEFINED TAG";
  }
}


struct Position {
  int line, pos, index;
};

void print_pos(struct Position *p) { printf("(%d, %d)", p->line, p->pos); }

struct Fragment {
  struct Position starting, following;
};

typedef struct Fragment YYLTYPE;

int continued;
struct Position cur;

struct Fragment comments[COMMENTS_ARRAY_SIZE];
size_t comments_array_len = 0;

void print_frag(struct Fragment f) {
  print_pos(&(f.starting));
  printf(" - ");
  print_pos(&(f.following));
}
```

```c
union Token {
  int ident;
};

struct ErrorMsg {
  char* msg;
  struct Position pos;
};

struct ErrorMsg errors[ERR_ARRAY_SIZE];
size_t errors_array_len = 0;


#define YY_USER_ACTION                                                  \
  {                                                                     \
    int i;                                                              \
    if (!continued)                                                     \
      yylloc->starting = cur;                                           \
    continued = 0;                                                      \
    for (i = 0; i < yyleng; i++) {                                      \
      if (yytext[i] == '\n') {                                          \
        ++cur.line;                                                     \
        cur.pos = 1;                                                    \
      } else {                                                          \
        ++cur.pos;                                                      \
      }                                                                 \
      ++cur.index;                                                      \
    }                                                                   \
    yylloc->following = cur;                                            \
  }

typedef union Token YYSTYPE;

void init_scanner(char *program) {
  continued = 0;
  cur.line = 1;
  cur.pos = 1;
  cur.index = 0;
  yy_scan_string(program);
}

void err(char *msg) {
  printf("Error");
  print_pos(&cur);
  printf(": %s\n ", msg);
}
```

```
%}

letter [a-zA-Z]
digit [0-9]
ident \/[^\/]+\/
number {digit}*
comment \/\/.*\n
while "\/while\/"
do "\/do\/"
end "\/end\/"

%%

[\n\t ]+

{while}  {
             return WHILE;
            }
{do}     {
             return DO;
            }
{end}    {
             return END;
            }
{number}     {
             return NUMBER;
            }
{comment}    {
             if (comments_array_len > COMMENTS_ARRAY_SIZE) {
               return COMMENT;
             }
             comments[comments_array_len++] = (struct Fragment){.starting = yylloc->startin
             return COMMENT;
            }
{ident}      {
             yylval->ident = add_ident(yytext);
             return IDENT;
            }
.            {
              errors[errors_array_len++] = (struct ErrorMsg){.msg = "undefined character",
            }
<<EOF>>      {
             return 0;
            }
%%
```

```c
void yyerror (YYLTYPE *locp, char const *msg) {
  fprintf(stderr, "SYNTAX ERROR\n");
}

int main() {

  YYSTYPE value;
  YYLTYPE coords;
  int tag;

  // init current pos

  cur.line = 1;
  cur.pos = 1;
  cur.index = 0;

  do
  {
    tag = yylex(&value, &coords);
    print_frag(coords);
    printf(" %s", tag_to_str(tag));
    if (tag == IDENT) {
      printf(" %d\n", value.ident);
    } else {
      printf("\n");
    }
  }

  while (tag);


  printf("\nIDENTIFIERS:\n");

  for (int i = 0; i < ident_array_len; ++i) {
    printf("%d: %s\n", i, ident_array[i]);
  }

  printf("\nERRORS:\n");

  for (int i = 0; i < errors_array_len; ++i) {
    printf("%s ", errors[i].msg);
    print_pos(&errors[i].pos);
    printf("\n");
```

```
  }

  printf("\nCOMMENTS:\n");

  for (int i = 0; i < comments_array_len; ++i) {
    print_frag(comments[i]);
  }

  return 0;
}
```

Тестирование

Входные данные

```
/do/ /abcd/ /while/ /end/ /abcd1/
abcd /abcd312/
// test abcd
```

Вывод на `stdout`

```
(1, 1) - (1, 5) DO
(1, 6) - (1, 12) IDENT 0
(1, 13) - (1, 20) WHILE
(1, 21) - (1, 26) END
(1, 27) - (1, 34) IDENT 1
(2, 6) - (2, 15) IDENT 2
(3, 1) - (4, 1) COMMENT
(3, 1) - (4, 1) EOF

IDENTIFIERS:
0: /abcd/
1: /abcd1/
2: /abcd312/

ERRORS:
undefined character (2, 1)
undefined character (2, 2)
undefined character (2, 3)
undefined character (2, 4)

COMMENTS:
(3, 1) - (4, 1)
```

Вывод

В ходе выполнения лабораторной работы были получены навыки по разработке лексических анализаторов с помощью flex. Для выполнения ЛР была разработана лексическая структура для заданного языка, которая была также описана с помощью flex.