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

18 марта 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является изучение генератора лексических анализаторов flex.

Индивидуальный вариант

- Идентификаторы: последовательности буквенных символов ASCII и десятичных цифр, начинающиеся и заканчивающиеся на букву.
- Числовые литералы: последовательности шестнадцатеричных цифр (чтобы литерал не был похож на идентификатор, его можно предварять нулём, цифры в любом регистре).
- Ключевые слова: «`qeq`», «`xx`», «`xxx`».

Реализация

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

%{

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

enum TAGS {
    TAG_IDENT = 1,
    TAG_HEXNUMBER,
    TAG_QEQ,
```

```
    TAG_XX,
    TAG_XXX,
};

char *tag_names[] = {
    "END_OF_PROGRAM", "IDENT", "HEXNUMBER",
    "KEYWORD qeq", "KEYWORD xx", "KEYWORD xxx",
};

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;

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

union Token {
    size_t code;
    unsigned long long hexnum;
};

typedef union Token YYSTYPE;

int continued;
struct Position cur;
struct ErrorList errorList;
struct NameDict nameDict;

#define YY_USER_ACTION {                \
    size_t 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;            \
}

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

struct ErrorList {
    struct Error *array;
    size_t length;
    size_t capacity;
};

void err(char *msg) {
    if (errorList.length == errorList.capacity) {
        errorList.capacity *= 2;
        errorList.array = (struct Error*)realloc(
            errorList.array, errorList.capacity * sizeof(struct Error));

    }

    ++errorList.length;
    errorList.array[errorList.length - 1].pos = cur;
    errorList.array[errorList.length - 1].msg = msg;
}

struct Name {
    char *str;
};

struct NameDict {
    struct Name *array;
    size_t length;
```

```
    size_t capacity;
};

size_t getNameCode(char *name) {
    size_t i;
    for (i = 0; i != nameDict.length; ++i) {
        if (strcmp(nameDict.array[i].str, name) == 0) {
            return i;
        }
    }
    return SIZE_MAX;
}

size_t addName(char *name) {
    if (nameDict.length == nameDict.capacity) {
        nameDict.capacity *= 2;
        nameDict.array = (struct Name*)realloc(
            nameDict.array, nameDict.capacity * sizeof(struct Name));
    }

    ++nameDict.length;
    char **str = &nameDict.array[nameDict.length - 1].str;
    *str = (char*)malloc((strlen(name) + 1) * sizeof(char));
    strcpy(*str, name);

    return nameDict.length - 1;
}

void init_scanner() {
    continued = 0;

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

    errorList.array = (struct Error*)malloc(sizeof(struct Error));
    errorList.length = 0;
    errorList.capacity = 1;

    nameDict.array = (struct Name*)malloc(sizeof(struct Name));
    nameDict.length = 0;
    nameDict.capacity = 1;
}

%}
```

```
LETTER      [a-zA-Z]
DIGIT       [0-9]
HEXDIGIT    [a-hA-F0-9]
IDENT       {LETTER}({DIGIT}*{LETTER})*
HEXNUMBER   0{HEXDIGIT}*

%%

[\n\t ]+
qeq                             return TAG_QEQ;
xx                              return TAG_XX;
xxx                             return TAG_XXX;

{IDENT}                         {
                                    size_t code;
                                    if ( (code = getNameCode(yytext)) == SIZE_MAX ) {
                                        code = addName(yytext);
                                    }
                                    yylval->code = code;
                                    return TAG_IDENT;
                                }
{HEXNUMBER}                     {
                                    while ( strlen(yytext) >= 17 && *yytext == '0' ) {
                                        ++yytext;
                                    }
                                    if ( strlen(yytext) < 17 )
                                    {
                                        yylval->hexnum = strtoull(yytext, NULL, 16);
                                        return TAG_HEXNUMBER;
                                    } else {
                                        err("number length overflow");
                                        BEGIN(0);
                                    }
                                }

.                               err("unexpected character"); BEGIN(0);

%%

int main( int argc, char *argv[] ) {
    size_t i;
    int tag;
    YYSTYPE value;
    YYLTYPE coords;

    if (argc > 1) {
```

```
        yyin = fopen(argv[1], "r");
    }
    init_scanner();

    printf("IDENTS:\n");
    do {
        tag = yylex(&value, &coords);
        printf("\t");
        print_frag(&coords);
        printf(" %s", tag_names[tag]);
        switch (tag) {
            case TAG_IDENT:
                printf(" %s", nameDict.array[value.code].str);
                break;
            case TAG_HEXNUMBER:
                printf(" 0x%llX", value.hexnum);
                break;
        }
        printf("\n");
    } while (tag != 0);

    printf("ERRORS:\n");
    for (i = 0; i != errorList.length; ++i) {
        printf("\tError ");
        print_pos(&errorList.array[i].pos);
        printf(": %s\n", errorList.array[i].msg);
    }

    free(errorList.array);
    for (i = 0; i != nameDict.length; ++i) {
        free(nameDict.array[i].str);
    }
    free(nameDict.array);
    if (argc > 1) {
        fclose(yyin);
    }

    return 0;
}
```

Тестирование

Входные данные

```
a aa xx xxx
```

```
0fff
0000000000000000000001000000f
0000100000000000000000000fff
qeq
sd2312ewsd23
```

Вывод на `stdout`

```
IDENTS:
        (1,1)-(1,2) IDENT a
        (1,3)-(1,5) IDENT aa
        (1,6)-(1,8) KEYWORD xx
        (1,9)-(1,12) KEYWORD xxx
        (2,1)-(2,5) HEXNUMBER 0xFFF
        (3,1)-(3,30) HEXNUMBER 0x1000000F
        (5,1)-(5,4) KEYWORD qeq
        (6,1)-(6,11) IDENT sd2312ewsd
        (6,12)-(6,13) END_OF_PROGRAM
ERRORS:
        Error (4,30): number length overflow
        Error (6,12): unexpected character
        Error (6,13): unexpected character
```

Вывод

Результатом выполнения данной работы стало изучение генератора лексических анализаторов flex.