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

17 апреля 2024 г.

Ольга Александрова, ИУ9-61Б

Цель работы

Целью данной работы является изучение генератора лексических анализаторов flex.

Индивидуальный вариант

Строковые литералы: ограничены двойными кавычками, могут содержать Escape-последовательности ”\”“, ”\n”, ”\t” и ”\”, не пересекают границы строк текста. Числовые литералы: последовательности десятичных цифр, разбитые точками на группы по три цифры («100», «1.000», «1.000.000»). Идентификаторы: последовательности латинских букв, знаков подчёркивания и цифр, начинающиеся с буквы или подчёркивания.

Реализация

```
%option noyywrap bison-bridge bison-locations
%{
    #include <stdio.h>
    #include <stdlib.h>

    enum TAGS {
        TAG_IDENT = 1,
        TAG_DOTNUMBER = 2,
        TAG_STRINGLIT = 3,
        TAG_ERROR = 4,
    };
    char *tag_names[] = {
        "END_OF_PROGRAM", "IDENT", "DOTNUMBER", "STRINGLIT",
    };
```

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

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

struct Fragment {
    Position starting, following;
};

typedef struct Fragment YYLTYPE;
typedef struct Fragment Fragment;

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

union Token {
    int charCode;
    int ident_num;
    char* ident_name;
    char* strLit;
    long long dotnum;
};

typedef union Token YYSTYPE;

int continued;
struct Position cur;
#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;            \
}


struct ErrorList {
    struct Error *array;
    size_t length;
    size_t capacity;
};
struct Error {
    struct Position pos;
    char *msg;
};
struct ErrorList errorList;


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;
}


typedef struct{
    int size;
    char** names;
} identTable;

void create_ident_table(identTable * t){
    t->size = 0;
    t->names = NULL;
}

char *add_ident(identTable *table, char *name) {
    for (int i = 0; i < table->size; i++) {
```

```
            if (strcmp(name, table->names[i]) == 0) {
                return table->names[i];
            }
        }

        table->size++;
        if (table->size == 1) {
            table->names = (char **)malloc(sizeof(char *) * (table->size));
        } else {
            table->names = (char **)realloc(table->names, sizeof(char *) * (table->size));
        }
        table->names[table->size - 1] = (char *)malloc(sizeof(char) * (strlen(name) + 1));
        strcpy(table->names[table->size - 1], name);

        return table->names[table->size - 1];
    }


    identTable table;

    void init_scanner (char *program){
        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;
        yy_scan_string(program);
    }

%}


LETTER      [a-zA-Z_]
DIGIT       [0-9]
IDENT       {LETTER}({LETTER}|{DIGIT})*
DOTNUMBER   {DIGIT}{1,3}(?:\.{DIGIT}{3})*
STRINGLIT   \"([^"\\\n]|(\\[tn\\\"]))*\"



%%

[\n\t ]+
```

```
{IDENT}               {
                        yylval->ident_name = add_ident(&table, yytext);
                        return TAG_IDENT;
                      }

{DOTNUMBER}          {

                        long long num = 0;
                        long long multiplier = 1;

                        int i = strlen(yytext) - 1;
                        long long group = 0;
                        long long mult2 = 1;
                        while (i >= 0) {
                            if (yytext[i] == '.') {
                                num += group * multiplier;
                                group = 0;
                                multiplier *= 1000;
                                mult2 = 1;
                            } else {
                                group += (yytext[i] - '0') * mult2;
                                mult2 = mult2 * 10;
                            }
                            i--;
                        }
                        num += group * multiplier;

                        yylval->dotnum = num;
                        return TAG_DOTNUMBER;
                    }
{STRINGLIT}          {
                        yytext++;
                        char* str = (char*) malloc(strlen(yytext) + 1);
                        int unescapedIndex = 0;

                        while (*yytext != '"' && *yytext != '\0') {
                            if (*yytext == '\\') {
                                yytext++;
                                switch (*yytext) {
                                    case '\"':
                                        str[unescapedIndex] = '\"';
                                        break;
                                    case '\\':
                                        str[unescapedIndex] = '\\';
                                        break;
                                    case 'n':
```

```
                                    str[unescapedIndex] = '\n';
                                    break;
                                case 't':
                                    str[unescapedIndex] = '\t';
                                    break;
                                default:
                                    str[unescapedIndex] = '\\';
                                    unescapedIndex++;
                                    str[unescapedIndex] = *yytext;
                                    break;
                            }
                        } else {
                            str[unescapedIndex] = *yytext;
                        }
                        unescapedIndex++;
                        yytext++;
                    }
                    str[unescapedIndex] = '\0';

                    yytext++;
                    yylval->strLit = str;
                    return TAG_STRINGLIT;
                }
.                     err ("ERROR unknown symbol");


<<EOF>>      {
               return 0;
             }


%%

int main(){

    int tag;
    YYSTYPE value;
    YYLTYPE coords;
    FILE *input;
    long size;
    char *buf;
    union Token token;

    input = fopen("in.txt", "r");
    if (input == NULL) {
        fputs("File not found", stderr);
```

```
    exit(1);
}


fseek(input, 0, SEEK_END);
size = ftell(input);
rewind(input);

buf = (char*)malloc(size + 1);
if (buf == NULL) {
    fputs("Memory error", stderr);
    exit(2);
}

if (fread(buf, 1, size, input) != size) {
    fputs ("Reading error", stderr);
    exit(3);
}

buf[size] = '\0';
fclose (input);

init_scanner(buf);
create_ident_table(&table);
do{
    tag = yylex(&value,&coords);
    if (tag == 0)
        break;

    printf("%s ",tag_names[tag]);
    print_frag(&coords);

    if (tag == TAG_DOTNUMBER){
        printf(": %lld", value.dotnum);
    }

    if (tag == TAG_IDENT){
        printf(": %s", value.ident_name);
    }
    if (tag == TAG_STRINGLIT){
        printf(": %s", value.strLit);
    }

    printf("\n");
} while (tag != 0);
```

```
    if (errorList.array == NULL) {
        printf("Error: errorList.array is NULL\n");
        return 1;
    }
    size_t i;
    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);
    }

    if (table.names != NULL) {
        for (int i = 0; i < table.size; i++) {
            free(table.names[i]);
        }
        free(table.names);
    }


    free(errorList.array);
    /* free(table.names); */
    free(buf);
    return 0;
}
```

Тестирование

Входные данные

```
100 _ku_ku _rg2e3   10.100.444  99 9 "hwgfs\n pidgd"
"hcsfb
df"
```

Вывод на `stdout`

```
DOTNUMBER (1,1)-(1,4): 100
IDENT (1,5)-(1,11): _ku_ku
IDENT (1,12)-(1,18): _rg2e3
DOTNUMBER (1,21)-(1,31): 10100444
DOTNUMBER (1,33)-(1,35): 99
DOTNUMBER (1,36)-(1,37): 9
STRINGLIT (1,38)-(1,53): hwgfs
 pidgd
IDENT (2,2)-(2,7): hcsfb
```

```
IDENT (3,1)-(3,3): df
ERRORS:
        Error (2,2): ERROR unknown symbol
        Error (3,4): ERROR unknown symbol
```

Вывод

В ходе лабораторной работы был изучен генератор лексических анализаторов flex.