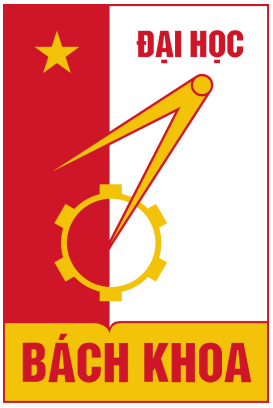
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Báo cáo bài tập lớn môn học:

Lý thuyết ngôn ngữ và phương pháp dịch

PHÂN TÍCH CÚ PHÁP CHO NGÔN NGỮ MINI-C

Sinh viên thực hiện:
Tạ Quang Tùng
MSSV: 20154280

Giáo viên hướng dẫn:
TS. Phạm Đăng Hải

Hà Nội, Ngày 27 tháng 12 năm 2018

Mục lục

1 Giới thiệu chung về ngôn ngữ Mini-C

Cú pháp của ngôn ngữ Mini-C được trình bày trong báo cáo này gần giống cú pháp của ngôn ngữ C99, ngoại trừ:

- Không có tiền xử lý (#include, #define, ...).
- Không có typedef.
- Không có con trỏ hàm.
- Kiểu dữ liệu số nguyên, số thực đơn giản hơn.
  - Kiểu số nguyên gồm: char, short, int, long, unsigned char, unsigned short, unsigned int, unsigned long.
  - Kiểu số thực gồm: float, double.
- Hằng số nguyên, số thực, kí tự đơn giản hơn.
- Không có định nghĩa cho hàm không giới hạn số tham số (VD: printf).
- Không có biểu thức chuyển đổi kiểu. Ví dụ: **(double)(6 + 5)**.
- Không có biểu thức sizeof một kiểu.

Được dựa trên chuẩn ISO về ngôn ngữ C. [2]

2 Phân tích từ vựng cho ngôn ngữ Mini-C

2.1 Bộ kí tự đầu vào

Bao gồm các kí tự sau:

- Chữ cái: A..Z, a..z
- Chữ số: 0..9
- Kí tự đặc biệt: ' " , : ? ; > < = . \ ^ + - * / % _ & | ~ ! ( ) [ ] { }

2.2 Các từ vựng của Mini-C

Bao gồm:

- Hằng số: Số nguyên, số thực hoặc kí tự (Mã ASCII).
- Xâu
- Định danh
- Từ khóa: break, case, char, const, continue, default, do, double, else, enum, float, for, goto, if, int, long, return, short, sizeof, struct, switch, union, void, unsigned, while

- Dấu phép toán:
  - Hậu tố / tiền tố: . -> ++ –
  - Số học: + - * / %
  - Bit: ~& ^|
  - Logic: ! && ||
  - Dịch bit: « »
  - Đặc biệt: ? ,
- Dấu ngoặc: ( ) [ ] { }
- Kết thúc lệnh / nhãn: : ;
- Phép gán: = += -= *= /= %= ^= |= «= »=

2.3 Phân tích từ vựng bằng Flex & Bison

Các từ tố được khái báo trong file Bison

```
%token KEYWORD
%token IDENT
%token CONSTANT
%token STRING_LITERAL

%token BREAK
%token CASE
%token CHAR
%token CONST
%token CONTINUE
%token DEFAULT
%token DO
%token DOUBLE
%token ELSE
%token ENUM
%token FLOAT
%token FOR
%token GOTO
%token IF
%token INT
%token LONG
%token RETURN
%token SHORT
%token SIZEOF
%token STRUCT
%token SWITCH
%token UNION
%token UNSIGNED
%token VOID
%token WHILE

%token ARROW
%token INCREASE
```

```
%token DECREASE
%token SHIFT_LEFT
%token SHIFT_RIGHT

%token GT
%token LT
%token GE
%token LE
%token EQ
%token NE

%token AND
%token OR

%token ADD_ASSIGN
%token SUB_ASSIGN
%token MUL_ASSIGN
%token DIV_ASSIGN
%token REM_ASSIGN
%token SL_ASSIGN
%token SR_ASSIGN
%token AND_ASSIGN
%token OR_ASSIGN
%token XOR_ASSIGN
```

Comment

Khai báo các điều kiện bắt đầu:

```
%x COMMENT_SINGLE
%x COMMENT_MULTIPLE
```

Xử lý comment của Mini-C:

```
<INITIAL>"//"           BEGIN(COMMENT_SINGLE);
<INITIAL>"/*"           BEGIN(COMMENT_MULTIPLE);
<COMMENT_SINGLE,COMMENT_MULTIPLE>.  ;
<COMMENT_SINGLE>"\n"    BEGIN(INITIAL);
<COMMENT_MULTIPLE>"\n"  ;
<COMMENT_MULTIPLE>"*/"  BEGIN(INITIAL);
```

Hằng số

Các định nghĩa được sử dụng:

```
DIGIT       [0-9]
HEXDIGIT    [0-9a-fA-F]
OCTDIGIT    [0-7]
EXP_PART    [eE][+-]?{DIGIT}+
EXP_PART_OPT    ({EXP_PART})?
C_CHAR      ([^'\\\n]|{ESCAPE_CHAR})
ESCAPE_CHAR \\['"?\\abfnrtv]
```

Xử lý hằng số:

```
[1-9]{DIGIT}* return CONSTANT;
0{OCTDIGIT}* return CONSTANT;
0x{HEXDIGIT}+ return CONSTANT;
```

```
0X{HEXDIGIT}+ return CONSTANT;

{DIGIT}*"."{DIGIT}+{EXP_PART_OPT} return CONSTANT;
{DIGIT}+"."{EXP_PART_OPT} return CONSTANT;
{DIGIT}+{EXP_PART} return CONSTANT;

"'"{C_CHAR}"'" return CONSTANT;
```

Các hằng số bao gồm số nguyên thập phân, số nguyên bát phân, số nguyên hexa, số phẩy động có phần mũ và kí tự (gồm các kí tự thông thường và các kí tự đặc biệt).

Xâu

Định nghĩa được sử dụng:

```
S_CHAR      ([^"\\\n]|{ESCAPE_CHAR})
```

Xử lý xâu:

```
\"{S_CHAR}+\" return STRING_LITERAL;
```

Các từ khóa

```
"break" return BREAK;
"case" return CASE;
"char" return CHAR;
"const" return CONST;
"continue" return CONTINUE;
"default" return DEFAULT;
"do" return DO;
"double" return DOUBLE;
"else" return ELSE;
"enum" return ENUM;
"float" return FLOAT;
"for" return FOR;
"goto" return GOTO;
"if" return IF;
"int" return INT;
"long" return LONG;
"return" return RETURN;
"short" return SHORT;
"sizeof" return SIZEOF;
"struct" return STRUCT;
"switch" return SWITCH;
"union" return UNION;
"void" return VOID;
"unsigned" return UNSIGNED;
"while" return WHILE;
```

Định danh

Các định nghĩa được sử dụng:

```
ALPHA       [_A-Za-z]
ALPHADIGIT  [_A-Za-z0-9]
```

Xử lý định danh:

```
{ALPHA}{ALPHADIGIT}* return IDENT;
```

Dấu phép toán

```
"." return '.';
"->" return ARROW;
"++" return INCREASE;
"--" return DECREASE;

"+" return '+';
"-" return '-';
"*" return '*';
"/" return '/';
"%" return '%';
"~" return '~';
"!" return '!';

"<<" return SHIFT_LEFT;
">>" return SHIFT_RIGHT;

">" return GT;
"<" return LT;
">=" return GE;
"<=" return LE;
"==" return EQ;
"!=" return NE;

"^" return '^';
"|" return '|';
"&" return '&';

"&&" return AND;
"||" return OR;

"?" return '?';
"," return ',';
```

Dấu ngoặc

```
"[" return '[';
"]" return ']';
"{" return '{';
"}" return '}';
"(" return '(';
")" return ')';
```

Dấu kết thúc lệnh / nhãn

```
":" return ':';
";" return ';';
```

Dấu phép gán

```
"=" return '=';
"+=" return ADD_ASSIGN;
```

```
"-=" return SUB_ASSIGN;
"*=" return MUL_ASSIGN;
"/=" return DIV_ASSIGN;
"%=" return REM_ASSIGN;
"&=" return AND_ASSIGN;
"|=" return OR_ASSIGN;
"^=" return XOR_ASSIGN;
"<<=" return SL_ASSIGN;
">>=" return SR_ASSIGN;
```

Xử lý lỗi

```
{WHITESPACE}            ;
. {
    printf("Khong nhan dien duoc ki tu: %s\n", yytext);
    exit(-1);
}
```

3 Phân tích cú pháp cho ngôn ngữ Mini-C

3.1 Quy ước đặt tên

Các kí tự không kết thúc có thể có những hậu tố có dạng:

- ***_seq**: Ví dụ: **declaration_seq** là đại diện cho một dãy các **declaration** mà không có dấu ngăn cách giữa chúng. Hay:
  declaration_seq → **declaration** | **declaration_seq declaration**
- ***_list**: Ví dụ: **param_list** là đại diện cho một dãy các **param** mà giữa chúng có dấu ngăn cách, thông thường là dấu phẩy. Hay:
  param_list → **param** | **param_list , param**
- ***_opt**: Ví dụ: **pointer_opt** là đại diện cho một kí tự **pointer** hoặc không có kí tự nào cả. Hay:
  pointer_opt → **pointer** | ϵ

3.2 Các sản xuất của ngôn ngữ Mini-C

Các sản xuất được viết bằng ngôn ngữ định nghĩa của Bison, bao gồm:

translation_unit

```
translation_unit
    : external_declaration
    | translation_unit external_declaration
    ;
```

Là kí tự bắt đầu của văn phạm. Bao gồm một dãy các **external_declaration**.

external_declaration

```
external_declaration
    : init_declaration
    | function_declaration
    | function_definition
    ;
```

Là các khai báo có thể được viết bên ngoài hàm. Bao gồm khai báo thông thường **init_declaration**, khai báo hàm hoặc định nghĩa hàm.

function_declaration và function_definition

Khai báo và định nghĩa hàm có cấu trúc như sau:

```
function_declaration
    : declaration_specifier function_declarator
        '(' parameter_list_opt ')' ';' ;

function_definition
    : declaration_specifier function_declarator
        '(' parameter_list_opt ')'
        compound_statement ;
```

parameter_list

Danh sách các tham số của hàm:

```
parameter_list_opt
    : %empty
    | parameter_list
    ;

parameter_list
    : declaration_specifier declarator
    | parameter_list ',' declaration_specifier declarator
    ;
```

Trong đó **declaration_specifier** là chỉ định phần tiền tố của kiểu (không bao gồm con trỏ và mảng), **declarator** sẽ bao gồm khai báo con trỏ, mảng và một định danh. Ngôn ngữ Mini-C được định nghĩa ở đây không hỗ trợ khai báo hay định nghĩa hàm mà tham số có kiểu nhưng không có tên. số. Ví dụ: **void func(int **);** .

declaration_specifier

Phần chỉ định khai báo, có cấu trúc như sau:

```
const_opt
    : %empty
    | CONST
    ;

declaration_specifier: const_opt type_specifier ;

type_specifier
    : VOID
    | UNSIGNED type_integer
```

```
    | type_integer
    | FLOAT
    | DOUBLE
    | struct_or_union_specifier
    | enum_specifier
    ;

type_integer
    : CHAR
    | SHORT
    | INT
    | LONG
    ;
```

Trong đó **CONST** là từ khóa **const**, được sử dụng cho các biến hằng. Các kiểu cho phép bao gồm:

- void
- Số phẩy động: float, double
- Số nguyên có dấu hoặc không dấu: có 4 loại ứng với 4 kích thước khác nhau: char (1 byte), short (2 byte), int (4 byte), long (8 byte).
- struct hoặc union
- enum

init_declaration

Khai báo thông thường (có thể trong hàm hoặc ngoài hàm). Ở đây cho phép khai báo mà không có tên của biến mục đích cho các định nghĩa struct, enum hoặc union có thể không có tên biến sau định nghĩa.

```
init_declaration
    : declaration_specifier init_declarator_list ';'
    | declaration_specifier ';'
    ;

init_declarator_list
    : init_declarator
    | init_declarator_list ',' init_declarator
    ;

init_declarator
    : declarator
    | declarator '=' initializer
    ;
```

declarator

Ví dụ trong khai báo:

```
const int * const p, **p[100];
```

Thì **const int** là **declaration_specifier**.
Còn *** const p**, ****p[100]** là các **declarator**.
Cũng cho phép khai báo mảng mà không có số lượng phần tử.

```
declarator
    : direct_declarator
    | pointer_seq direct_declarator
    ;

pointer: '*' const_opt

pointer_seq
    : pointer
    | pointer_seq pointer
    ;

assignment_expression_opt
    : %empty
    | assignment_expression
    ;

direct_declarator
    : IDENT
    | direct_declarator '[' assignment_expression_opt ']'
    ;

function_declarator
    : IDENT
    | pointer_seq IDENT
    ;
```

declarator của hàm không có dãy các mảng.

initializer

```
initializer
    : assignment_expression
    | '{' initializer_list '}'
    | '{' initializer_list ',' '}'
    ;

initializer_list
    : initializer
    | initializer_list ',' initializer
    ;
```

struct_or_union_specifier

Định nghĩa hoặc khai báo struct và union:

```
ident_opt
    : %empty
    | IDENT
    ;

struct_or_union_specifier
    : struct_or_union ident_opt '{' struct_declaration_seq '}'
```

```
    | struct_or_union ident_opt '{' '}'
    | struct_or_union IDENT

struct_or_union
    : STRUCT
    | UNION
    ;

struct_declaration_seq
    : struct_declaration
    | struct_declaration_seq struct_declaration
    ;

struct_declaration
    : declaration_specifier struct_declarator_list ';'
    ;

struct_declarator_list
    : declarator
    | struct_declarator_list ',' declarator
    ;
```

enum_specifier

Định nghĩa hoặc khai báo enum:

```
enum_specifier
    : ENUM ident_opt '{' enumerator_list '}'
    | ENUM ident_opt '{' enumerator_list ',' '}'
    | ENUM IDENT
    ;

enumerator_list
    : enumerator
    | enumerator_list ',' enumerator
    ;

enumerator
    : IDENT
    | IDENT '=' constant_expression
    ;
```

Trong đó có hỗ trợ dấu phẩy sau danh sách các **enumerator** (hay trailing comma).

expression

Các biểu thức:

```
primary_expression
    : IDENT
    | CONSTANT
    | STRING_LITERAL
    | '(' expression ')'
    ;
```

```
postfix_expression
    : primary_expression
    | postfix_expression '[' expression ']'
    | postfix_expression '(' argument_expression_list ')'
    | postfix_expression '(' ')'
    | postfix_expression '.' IDENT
    | postfix_expression ARROW IDENT
    | postfix_expression INCREASE
    | postfix_expression DECREASE
    ;

argument_expression_list
    : assignment_expression
    | argument_expression_list ',' assignment_expression
    ;

unary_expression
    : postfix_expression
    | INCREASE unary_expression
    | DECREASE unary_expression
    | unary_operator cast_expression
    | SIZEOF unary_expression
    ;

unary_operator
    : '&'
    | '*'
    | '+'
    | '-'
    | '~'
    | '!'
    ;

cast_expression
    : unary_expression
    ;

multiplicative_expression
    : cast_expression
    | multiplicative_expression '*' cast_expression
    | multiplicative_expression '/' cast_expression
    | multiplicative_expression '%' cast_expression
    ;

additive_expression
    : multiplicative_expression
    | additive_expression '+' multiplicative_expression
    | additive_expression '-' multiplicative_expression
    ;

shift_expression
    : additive_expression
    | shift_expression SHIFT_LEFT additive_expression
    | shift_expression SHIFT_RIGHT additive_expression
    ;
```

```
relational_expression
    : shift_expression
    | relational_expression LT shift_expression
    | relational_expression GT shift_expression
    | relational_expression LE shift_expression
    | relational_expression GE shift_expression
    ;

equality_expression
    : relational_expression
    | equality_expression EQ relational_expression
    | equality_expression NE relational_expression
    ;

and_expression
    : equality_expression
    | and_expression '&' equality_expression
    ;

xor_expression
    : and_expression
    | xor_expression '^' and_expression
    ;

or_expression
    : xor_expression
    | or_expression '|' xor_expression
    ;

logical_and_expression
    : or_expression
    | logical_and_expression AND or_expression
    ;

logical_or_expression
    : logical_and_expression
    | logical_or_expression OR logical_and_expression
    ;

conditional_expression
    : logical_or_expression
    | logical_or_expression '?' expression ':' conditional_expression
    ;

assignment_expression
    : conditional_expression
    | unary_expression assignment_operator assignment_expression
    ;

assignment_operator
    : '='
    | ADD_ASSIGN
    | SUB_ASSIGN
    | MUL_ASSIGN
    | DIV_ASSIGN
    | REM_ASSIGN
```

```
    | AND_ASSIGN
    | OR_ASSIGN
    | XOR_ASSIGN
    | SL_ASSIGN
    | SR_ASSIGN
    ;

expression
    : assignment_expression
    | expression ',' assignment_expression
    ;

constant_expression: conditional_expression ;
```

statement

Các loại statement:

```
statement
    : labeled_statement
    | compound_statement
    | expression_statement
    | selection_statement
    | iteration_statement
    | jmp_statement
    ;

labeled_statement
    : IDENT ':' statement
    | CASE constant_expression ':' statement
    | DEFAULT ':' statement
    ;

compound_statement: '{' block_item_seq_opt '}' ;

block_item_seq_opt
    : %empty
    | block_item_seq
    ;

block_item_seq
    : block_item
    | block_item_seq block_item
    ;

block_item
    : init_declaration
    | statement
    ;

expression_statement
    : ';'
    | expression ';'
    ;

selection_statement
```

```
    : IF '(' expression ')' statement
    | IF '(' expression ')' statement ELSE statement
    | SWITCH '(' expression ')' statement
    ;

expression_opt
    : %empty
    | expression
    ;

iteration_statement
    : WHILE '(' expression ')' statement
    | DO statement WHILE '(' expression ')' ';'
    | FOR '(' expression_opt ';' expression_opt ';' expression_opt ')'
      statement
    | FOR '(' init_declaration expression_opt ';' expression_opt ')'
      statement
    ;

jmp_statement
    : GOTO IDENT ';'
    | CONTINUE ';'
    | BREAK ';'
    | RETURN expression_opt ';'
    ;
```

Trong đó có đầy đủ các lệnh trong C99.

4 Kết quả chạy thử nghiệm

Biên dịch chương trình sử dụng Bison xảy ra shift/reduce conflict tại câu lệnh **if else**.
Toàn bộ mã nguồn tại link [1].

Với đầu vào:

```
int x;
int *func();

struct name {
    const float x, y;
    int *a, b, mang[10];
};

union tung {
    int x, y[100 * 2];
};

enum token {
    ident,
    number = 20,
} x, y;

void func(int **x, float u);

void func(int a);
```

```
int *main(int x) {
    int *p = &x, mang[100][2] = {{1, 2, 3}, {4, 5},};
    int **pp = &p;

nhan:
    if (x == 10) {
        int k = 4 * 10 - 100 * 3 << 5;
        goto nhan;
    }
    else
        return 100;

    for (x = 100; x < 10; x++) {
        ++x; --y; y--;
        break;
        if (u < 20)
            continue;
        x = 199;
        x = 0732;
        x = 3.32e-33;
        x = '"';
        s = "ta quang tung\"\n";
        x += 5;
        x <<= 10;
    }

    switch (x) {
    case 1:
        x = 5;
        break;

    case 5:
        y = 3;
    default:
        x = 1;
        break;
    }

    x = 1;
    do {
        x++;
    } while (x == 3);
    return 0;
}
```

Chương trình nhận diện thành công.

Tuy nhiên, ngay cả với đầu vào:

```
int;

struct Point {
    unsigned int a, b;
};

int main() {
```

```
    return 0;
}
```

Chương trình cũng nhận diện thành công, dù cho **int;** không có tên biến, lý do vì văn phạm cho phép điều này nhằm cho phép khai báo struct như trên. Tuy nhiên có thể chuyển việc nhận diện lỗi này cho bộ phân tích ngữ nghĩa.

Tài liệu tham khảo

[1] GitHub: quangtung97-study/20181-compiler. https://github.com/quangtung97-study/20181-compiler/tree/master/mini-C.

[2] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf, May 2005.