语法分析实验报告

何宜晖
计算机46
2140504137
电信学院
heyihui@stu.xjtu.edu.cn

2016年11月

1 实验目的

1. 强化对系统软件综合工程实现能力的训练；
2. 加强对语法分析原理、方法和基本实现技术的理解；

2 实验内容

1. 用C语言或者其他的高级语言作为宿主语言完成C0语言的词法分析器的设计和实现.
2. 针对if语句的文法编写一个递归下降分析程序，输出结果为抽象语法树。注意，if语句文法中的表达式E采用四则运算表达式的文法；抽象语法树的格式自行设计，如果需要降低难度的话，也可用具体语法树而不用抽象语法树作为输出.

3 实验要求

1. 编写C0语言的语法分析器的源程序并调试通过。其中语法分析程序既可以自己手动去完成，也可以利用YACC自动生成。
2. 通过测试程序的验收；
3. 实验报告按照提供的模板填写：

4 功能描述

该程序要实现的是一个读程序的过程，从输入的源程序中生成一颗抽象语法树[6][1][5][2].，并将结果保存为xml。

我编写了一些简单的c0语言程序，输出结果见附录B.

大部分语法，终结符非终结符的定义，我都是参照实验指导书给出的c0语法如下。部分语法参照 ANSI C [3]

```
program→ { var-declaration | fun-declaration }
    var-declaration→ int ID { , ID }
    fun-declaration→ ( int | void ) ID ( params ) compound-stmt
    params → int ID { , int ID } | void | empty
    compound-stmt→ { { var-declaration } { statement } }
    statement→ expression-stmtcompound-stmt if-stmt while-stmt
    |return-stmt
    expression-stmt→ [ expression ] ;
    if-stmt→ if( expression ) statement [ else statement ]
    while-stmt→ while( expression ) statement
    return-stmt→ return [ expression ] ;
    expression→ ID = expression | simple-expression
    simple-expression→ additive-expression [ relop additive-expression ]
```

```
 relop → < | <= | > | >= | == | !=
 additive-expression→ term [( + | - ) term ]
 term→ factor [ ( * | / ) factor ]
factor→ ( expression )| ID | call | NUM
call→ ID( args )
args→ expression { , expression } | empty
ID →... ;参见C语言标识符定义
NUM →... ;参见C语言数的定义
```

5 程序结构描述

5.1 语法设计

5.1.1 ifelse

本程序有一个**移进归约冲突**：

```
IF '(' expression ')' statement _ ELSE statement
```

由于yacc默认的优先级是从上到下，只需要把if写在if else前面，这个冲突可以默认被消解。

5.1.2 双目运算

双目运算$+, -$fi $*$ fi/ 需要具有左结合性，否则会产生歧义，并且乘除应当比加减先结合。具体到yacc如下：

```
%left PLUS MINUS
%left STAR SLASH
```

5.1.3 单目运算

单目运算+-，结合紧密型应当比乘除高，然而在前面已经定义了结合紧密性低于乘除，所以需要在单目运算中单独处理，具体到yacc时，使用prec语法，临时更换其结合紧密性。

```
additive_expression
    ...
    ...
    | PLUS additive_expression %prec STAR {...}
    | MINUS additive_expression %prec STAR {...}
```

5.1.4 $++, --$运算

这是在c，c0,c++等语言中的特殊语法。如果在词法分析时，只单个提取+，-。后面是无法区分–i与-(-i)的，会有**归约-归约冲突**。故需要定义两个单独的token，与+，-区别开来。

```
%token INC_OP DEC_OP
```

在词法分析时，就要单独识别：

```
"++" {return INC_OP; }
"--" {return DEC_OP; }
```

在语法分析时，就能轻松区别开了：

```
unary_expression
    : INC_OP ID {...}
    | DEC_OP ID {...}
    | postfix_expression {...}
    ;

postfix_expression
    : ID INC_OP {...}
    | ID DEC_OP {...}
    ;
```

5.2 算法设计

5.2.1 语法树

我设计了简单的语法树节点，便于输出打印, child便于递归打印：

```
struct treeNode{
    struct treeNode *child[MAXCHILD];
    char* nodeType;
    char* string;
    char* value;
    char* dataType;
    int lineNo;
    int Nchildren;
};
```

5.2.2 终结符与非终结符

终结符直接用string表示，非终结符用语法树的节点表示。

```
%union {
    char* str;
    struct treeNode * ast;
}
```

在yacc中，需要分别定义类型：

```
%type<ast> atree program ...
%type<str> relop declaration_specifiers
```

5.2.3 递归生成语法树

在yacc中 $$,$1,$2,$3可以用来返回指针，以及获得下一级的指针。yacc会递归地，先按照需求计算对应位置非终结符的返回值。使用方法如下：

```
params
    : params_list {$$=$1;}
    | VOID {$$ = newnode(yylineno,"params", none, none, "VOID",  0);}
    ;
```

而newnode，是用来将节点穿起来的函数,利用了yacc递归的性质：

```
struct treeNode * newnode(int lineNo, char* nodeType, char* string,
    char* value, char* dataType, int Nchildren, ...){
   struct treeNode * node = (struct treeNode*) malloc(sizeof(struct
       treeNode));
   node->nodeType = nodeType;
   node->string = string;
   node->value = value;
   node->dataType = dataType;
   node->lineNo = lineNo;
   node->Nchildren = Nchildren;
   va_list ap;
   int i;
   va_start(ap, Nchildren);
   for (i=0;i<Nchildren;i++){
      node->child[i]=va_arg(ap, struct treeNode *);
   }
   va_end(ap);
   return node;
}
```

5.3 打印语法树

为了打印语法树，需要使用**拓广文法**。单独设定一个非终结符atree在程序开始，它没有任何语法意义，仅仅是为了开始节点有区分性，用来打印语法树。

```
%start atree
%%
atree:program {printNode($1);}
```

```
void printNode(struct treeNode* node){
   printf("%s<Tree lineNo=\"%d\" nodeType=\"%s\" string=\"%s\"
       value=\"%s\" dataType=\"%s\">\n",
      indent,
      node->lineNo,
      node->nodeType,
```

```
        node->string,
        node->value,
        node->dataType);
    int i;
    if (node->Nchildren > 0){
        printf("%s<Child>\n", indent);
        incIndent();
        for (i=0;i<node->Nchildren;i++){
            printNode(node->child[i]);
        }
        decIndent();
        printf("%s</Child>\n", indent);
    }
    printf("%s</Tree>\n", indent);
}
```

5.4 程序的缺陷

本程序有一个缺陷，就是打印行号时，会打印出匹配到的非终结符的最后一行的行号，而不是样例中给出的第一行的行号。这是自下而上分析的栈读取的特性导致的，我暂时还不知道怎么解决。

5.5 程序代码

程序代码，见附录A
语法分析结果,见附录B

6 实验总结

1. 你在编程过程中花时多少？我大概花了两个晚上。
2. 多少时间在纸上设计？花了一小时在纸上设计。
3. 多少时间上机输入和调试？6个小时左右。
4. 多少时间在思考问题？3个小时左右。
5. 遇到了哪些难题，你是怎么克服的？ 实验中还是遇到了很多难题。首先是如何使用yacc，经过仔细阅读参考书目lex&yacc [4], 让我了解了yacc的基本用法。其他大部分问题都是通过上网查找相关资料解决的，具体困难和解决可以参考实验报告前面部分
6. 你对你的程序的评价？我的程序使用yacc和lex编写，token定义模仿标准的ANIS C。只有一个shift/reduce冲突，自己非常满意。
7. 你的收获有哪些？这次实验学会了使用yacc，以及如何将lex和yacc配合使用。这次实验后，我对编译原理产生了浓厚的兴趣。

参考文献

[1] A. W. Appel. *Modern compiler implementation in C*. Cambridge university press, 2004.

[2] A. W. Appel. 现代编译原理 c 语言描述, 2006.

[3] J. Degener. Ansi c grammar, http://www.quut.com/c/ansi-c-grammar-y.html.

[4] J. R. Levine, T. Mason, and D. Brown. *Lex & yacc*. " O'Reilly Media, Inc.", 1992.

[5] K. C. Louden and 编译原理及实践. 冯博琴, 冯岚, 等译. 编译原理及实践. 北京: 机械工业出版社, 2000.

[6] 陈火旺, 刘春林, 谭庆平, et al. 程序设计语言编译原理. 第三版). 北京: 国防工业出版社, 2000.

A 程序文件

yacc 文件：q2.y

```
%{

//#include "y.tab.h"
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

//typedef char* string;
//#define YYSTYPE string
#define STR(VAR) (#VAR)
#define release 1
#define MAXCHILD 10

extern void yyerror(const char *); /* prints grammar violation message */
extern int yylex(void);
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;

char* tab=" ";
char indent[100]="";

char* integer="INT";
char* floating="float";
char* none = "none";
char* assign = "=";

void incIndent(){
    strcat(indent, tab);
}
void decIndent(){
    int len = strlen(indent);
    indent[len-2]='\0';
}

struct treeNode{
    struct treeNode *child[MAXCHILD];
    char* nodeType;
    char* string;
    char* value;
    char* dataType;
    int lineNo;
    int Nchildren;
};
```

```
void printNode(struct treeNode* node){
    printf("%s<Tree lineNo=\"%d\" nodeType=\"%s\" string=\"%s\"
        value=\"%s\" dataType=\"%s\">\n",
        indent,
        node->lineNo,
        node->nodeType,
        node->string,
        node->value,
        node->dataType);
    int i;
    if (node->Nchildren > 0){
        printf("%s<Child>\n", indent);
        incIndent();
        for (i=0;i<node->Nchildren;i++){
            printNode(node->child[i]);
        }
        decIndent();
        printf("%s</Child>\n", indent);
    }
    printf("%s</Tree>\n", indent);
}

struct treeNode * newnode(int lineNo, char* nodeType, char* string,
    char* value, char* dataType, int Nchildren, ...){
    struct treeNode * node = (struct treeNode*) malloc(sizeof(struct
        treeNode));
    node->nodeType = nodeType;
    node->string = string;
    node->value = value;
    node->dataType = dataType;
    node->lineNo = lineNo;
    node->Nchildren = Nchildren;
    va_list ap;
    int i;
    va_start(ap, Nchildren);
    for (i=0;i<Nchildren;i++){
        node->child[i]=va_arg(ap, struct treeNode *);
    }
    va_end(ap);
    return node;
}
%}
%code requires {

}

%union {
    char* str;
    struct treeNode * ast;
}
```

```
%token IF ELSE WHILE RETURN VOID INT FLOAT
%token INC_OP DEC_OP PLUS MINUS STAR SLASH LT LTEQ GT GTEQ EQ NEQ ASSIGN
%token SEMI COMMA LPAREN RPAREN LSQUAR RSQUAR LBRACE RBRACE LCOMMENT
    RCOMMENT
%token <str> ID NUM
%token LETTER DIGIT
%token NONTOKEN ERROR ENDFILE

%left PLUS MINUS
%left STAR SLASH

%type<ast> atree program external_declaration var_declaration
    init_declarator_list fun_declaration params_list compound_stmt
    declarator params block_item_list block_item factor call term
    additive_expression simple_expression unary_expression
    postfix_expression assignment_expression return_stmt while_stmt
    if_stmt expression statement args expression_stmt
%type<str> relop declaration_specifiers

%start atree
%%

atree:program {printNode($1);}

program
    : external_declaration {$$=$1;}
    | program external_declaration {$$=newnode(yylineno, STR(program),
        none, none, none, 2, $1, $2); }
    ;

external_declaration
    : var_declaration {$$=$1;}
    | fun_declaration {$$=$1;}
    ;

var_declaration
    : declaration_specifiers init_declarator_list SEMI
    {$$=newnode(yylineno, "var_declaration", none, none, $1, 1, $2); }
    ;

init_declarator_list
    : ID {$$ = newnode(yylineno, "init_declarator_list", $1, none, none,
        0);}
    | ID ASSIGN expression {$$ =
        newnode(yylineno,"init_declarator_list", $1, none, none, 1, $3);}
    | init_declarator_list COMMA ID {$$ =
        newnode(yylineno,"init_declarator_list", $3, none, none, 1, $1);}
    ;
```

```
declarator
    : LPAREN RPAREN {$$ = newnode(yylineno,"declarator", none, none,
        none, 0);}
    | LPAREN params RPAREN {$$ = newnode(yylineno,"declarator", none,
        none, none, 1, $2);}
    ;

fun_declaration
    : declaration_specifiers ID declarator compound_stmt
        {$$=newnode(yylineno,STR(fun_declaration), $2, none, $1, 1, $4);}
    ;

declaration_specifiers
    : INT {$$=integer;}
    | VOID {$$="VOID";}
    | FLOAT {$$="VOID";}
    ;

params_list
    : INT ID {$$ = newnode(yylineno,"params_list", $2, none, integer,
        0);}
    | FLOAT ID {$$ = newnode(yylineno,"params_list", $2, none, floating,
        0);}
    | params_list COMMA INT ID {$$ = newnode(yylineno,"params_list", $4,
        none, integer, 1, $1);}
    | params_list COMMA FLOAT ID {$$ = newnode(yylineno,"params_list",
        $4, none, floating, 1, $1);}
    ;

params
    : params_list {$$=$1;}
    | VOID {$$ = newnode(yylineno,"params", none, none, "VOID", 0);}
    ;

compound_stmt
    : LBRACE RBRACE {$$ = newnode(yylineno,"compound_stmt", none, none,
        none, 0);}
    | LBRACE block_item_list RBRACE {$$ = $2;}
    ;

block_item_list
    : block_item {$$ = $1;}
    | block_item_list block_item {$$ =
        newnode(yylineno,"block_item_list", none, none, none, 2, $1,
        $2);}
    ;

block_item
    : var_declaration {$$=$1;}
    | statement {$$=$1;}
```

```
    ;

statement
    : expression_stmt {$$=$1;}
    | compound_stmt {$$=$1;}
    | if_stmt {$$=$1;}
    | while_stmt {$$=$1;}
    | return_stmt {$$=$1;}
    ;

expression_stmt
    : SEMI {$$ = newnode(yylineno,"expression_stmt", none, none, none,
        0);}
    | expression SEMI {$$=$1;}
    ;

if_stmt
    : IF LPAREN expression RPAREN statement ELSE statement {$$ =
        newnode(yylineno,"if_stmt", none, none, none, 3, $3, $5, $7);}
    | IF LPAREN expression RPAREN statement {$$ =
        newnode(yylineno,"if_stmt", none, none, none, 2, $3, $5);}
    ;

while_stmt
    : WHILE LPAREN expression RPAREN statement {$$ =
        newnode(yylineno,"while_stmt", none, none, none, 2, $3, $5);}
    ;

return_stmt
    : RETURN SEMI {$$ = newnode(yylineno,"return_stmt", none, none,
        none, 0);}
    | RETURN expression SEMI {$$ = newnode(yylineno,"return_stmt", none,
        none, none, 1, $2);}
    ;

expression
    : assignment_expression {$$=$1;}
    | simple_expression {$$=$1;}
    ;

assignment_expression
    : ID ASSIGN expression {$$ =
        newnode(yylineno,"assignment_expression", $1, none, none, 1,
        $3);}
    | unary_expression {$$=$1;}  ;

unary_expression
    : INC_OP ID {$$ = newnode(yylineno,"unary_expression", $2, none,
        "++", 0);}
    | DEC_OP ID {$$ = newnode(yylineno,"unary_expression", $2, none,
```

```
        "--", 0);}
    | postfix_expression {$$=$1;}
    ;

postfix_expression
    : ID INC_OP {$$ = newnode(yylineno,"postfix_expression", $1, none,
        "++", 0);}
    | ID DEC_OP {$$ = newnode(yylineno,"postfix_expression", $1, none,
        "--", 0);}
    ;

simple_expression
    : additive_expression {$$=$1;}
    | additive_expression relop additive_expression {$$ =
        newnode(yylineno,"simple_expression", none, none, $2, 2, $1,
        $3);}
    ;

relop
    : LT {$$ = "<";}
    | LTEQ {$$ = "<=";}
    | GT   {$$ = ">";}
    | GTEQ {$$ = ">=";}
    | EQ   {$$ = "==";}
    | NEQ  {$$ = "!=";}
    ;

additive_expression
    : term {$$=$1;}
    | additive_expression PLUS term {$$ =
        newnode(yylineno,"additive_expression", none, none, "+", 2, $1,
        $3);}
    | additive_expression MINUS term {$$ =
        newnode(yylineno,"additive_expression", none, none, "-", 2, $1,
        $3);}
    | PLUS additive_expression %prec STAR {$$ =
        newnode(yylineno,"additive_expression", none, none, "+", 1, $2);}
    | MINUS additive_expression %prec STAR {$$ =
        newnode(yylineno,"additive_expression", none, none, "-", 1, $2);}
    ;

term
    : factor {$$=$1;}
    | term STAR factor {$$ = newnode(yylineno,"term", none, none, "*",
        2, $1, $3);}
    | term SLASH factor {$$ = newnode(yylineno,"term", none, none, "/",
        2, $1, $3);}
    ;

factor
```

```
    : LPAREN expression RPAREN {$$=$2;}
    | ID {$$ = newnode(yylineno,"factor", $1, none, none, 0);}
    | call {$$=$1;}
    | NUM {$$ = newnode(yylineno,"factor", none, $1, none, 0);}
    ;

call
    : ID LPAREN RPAREN {$$ = newnode(yylineno,"call", $1, none, none,
        0);}
    | ID LPAREN args RPAREN {$$ = newnode(yylineno,"call", $1, none,
        none, 1, $3);}
    ;

args
    : expression {$$=$1;}
    | expression COMMA args {$$ = newnode(yylineno,"args", none, none,
        none, 2, $1, $3);}
    ;

%%
#include <stdio.h>
main(argc, argv)
int argc;
char** argv;
{
if (argc > 1)
{
    FILE *file;
    file = fopen(argv[1], "r");
    if (!file)
    {
        fprintf(stderr, "failed open");
        exit(1);
    }
    yyin=file;
    //printf("success open %s\n", argv[1]);
}
else
{
    printf("no input file\n");
    exit(1);
}
printf("<?xml version=\"1.0\"?>\n<root>\n");
yyparse();
printf("</root>\n");
return 0;
}

void yyerror(const char *s)
{
```

```
   fflush(stdout);
   fprintf(stderr, "*** %s\n", s);
}
```

yacc文件对应的lex文件：q2.l

```
%e  1019
%p  2807
%n  371
%k  284
%a  1213
%o  1117



D   [0-9]
NZ  [1-9]
L   [a-zA-Z_]
A   [a-zA-Z_0-9]
WS  [ \t\v\n\f]

%option yylineno

%{
#define ifprint 0

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
//extern char* yylval;

extern void yyerror(const char *); /* prints grammar violation message */

extern int sym_type(const char *); /* returns type from symbol table */

#define sym_type(identifier) IDENTIFIER /* with no symbol table, fake it
    */

static void comment(void);

//if (ifprint) {ECHO;fprintf(yyout, " ");}
%}

%%
"/*"                                { comment(); }
"//".*                                { /* consume //-comment */ }

"if"            {if (ifprint) {ECHO;fprintf(yyout, " ");}; return(IF); }
"else"             {if (ifprint) {ECHO;fprintf(yyout, " ");};
```

```
    return(ELSE); }
"while"            {if (ifprint) {ECHO;fprintf(yyout, " ");};
    return(WHILE); }
"return"           {if (ifprint) {ECHO;fprintf(yyout, " ");};
    return(RETURN); }
"void"             {yylval.str = strdup(yytext);
                       return(VOID); }
"int"            {yylval.str = strdup(yytext);
                      return(INT); }
"float"            {yylval.str = strdup(yytext);
                      return(FLOAT); }

"++"          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return INC_OP; }
"--"          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return DEC_OP; }
"+"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return PLUS; }
"-"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return MINUS; }
"*"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return STAR; }
"/"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return SLASH; }
"<"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return LT; }
"<="          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return LTEQ; }
">"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return GT; }
">="          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return GTEQ; }
"=="          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return EQ; }
"!="          {if (ifprint) {ECHO;fprintf(yyout, " ");}; return NEQ; }
"="           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return ASSIGN; }

("[")         {if (ifprint) {ECHO;fprintf(yyout, " ");}; return LSQUAR; }
("]")         {if (ifprint) {ECHO;fprintf(yyout, " ");}; return RSQUAR; }
("{")         {if (ifprint) {ECHO;fprintf(yyout, " ");}; return LBRACE; }
("}")         {if (ifprint) {ECHO;fprintf(yyout, " ");}; return RBRACE; }
";"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return SEMI; }
","           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return COMMA; }
"("           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return LPAREN; }
")"           {if (ifprint) {ECHO;fprintf(yyout, " ");}; return RPAREN; }

{L}{A}*             { yylval.str = strdup(yytext); return ID; }
{D}+       {if (ifprint) {ECHO;fprintf(yyout, " ");};yylval.str =
    strdup(yytext); return NUM; }


{WS}+           {if (ifprint) {ECHO;fprintf(yyout, " ");}; /*
    whitespace separates tokens */ }
.             {if (ifprint) {ECHO;fprintf(yyout, " ");}; /* discard bad
    characters */ }

%%

int yywrap(void)      /* called at end of input */
{
   return 1;          /* terminate now */
```

```
}

static void comment(void)
{
    int c;

    while ((c = input()) != 0)
        if (c == '*')
        {
            while ((c = input()) == '*')
                ;

            if (c == '/')
                return;

            if (c == 0)
                break;
        }
    yyerror("unterminated comment");
}
```

B 样例代码与输出

c_0 语言文件 orig.c

```
void main()
{
   int a=0;
   int b=2;
   while(a==b)
     ++a;
}
```

c_0 语言文件 orig.c的输出

```
<?xml version="1.0"?>
<root>
  <Tree dataType="VOID" lineNo="7" nodeType="fun_declaration"
      string="main" value="none">
    <Child>
      <Tree dataType="none" lineNo="6" nodeType="block_item_list"
          string="none" value="none">
        <Child>
          <Tree dataType="none" lineNo="4" nodeType="block_item_list"
              string="none" value="none">
            <Child>
              <Tree dataType="INT" lineNo="3" nodeType="var_declaration"
                  string="none" value="none">
                <Child>
                  <Tree dataType="none" lineNo="3"
                      nodeType="init_declarator_list" string="a"
                      value="none">
                    <Child>
                      <Tree dataType="none" lineNo="3" nodeType="factor"
                          string="none" value="0"></Tree>
                    </Child>
                  </Tree>
                </Child>
              </Tree>
              <Tree dataType="INT" lineNo="4" nodeType="var_declaration"
                  string="none" value="none">
                <Child>
                  <Tree dataType="none" lineNo="4"
                      nodeType="init_declarator_list" string="b"
                      value="none">
                    <Child>
                      <Tree dataType="none" lineNo="4" nodeType="factor"
                          string="none" value="2"></Tree>
                    </Child>
                  </Tree>
                </Child>
```

```
            </Tree>
          </Child>
        </Tree>
        <Tree dataType="none" lineNo="6" nodeType="while_stmt"
            string="none" value="none">
          <Child>
            <Tree dataType="==" lineNo="5" nodeType="simple_expression"
                string="none" value="none">
              <Child>
                <Tree dataType="none" lineNo="5" nodeType="factor"
                    string="a" value="none"></Tree>
                <Tree dataType="none" lineNo="5" nodeType="factor"
                    string="b" value="none"></Tree>
              </Child>
            </Tree>
            <Tree dataType="++" lineNo="6" nodeType="unary_expression"
                string="a" value="none"></Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
  </Child>
 </Tree>
</root>
```

c_0 语言文件 simple.c

```
int a=0;
```

c_0 语言文件 simple.c的输出

```
<?xml version="1.0"?>
<root>
 <Tree dataType="INT" lineNo="1" nodeType="var_declaration"
     string="none" value="none">
   <Child>
     <Tree dataType="none" lineNo="1" nodeType="init_declarator_list"
         string="a" value="none">
       <Child>
         <Tree dataType="none" lineNo="1" nodeType="factor"
             string="none" value="0"></Tree>
       </Child>
     </Tree>
   </Child>
 </Tree>
</root>
```

c_0 语言文件 test.c

```
void main()
{
   int sum1=100;
   int a;
   int b;
   if (1 == 1+0){
       a=3;
   }
    else {
       b=5;
    }
}
```

c_0 语言文件 test.c的输出

```
<?xml version="1.0"?>
<root>
  <Tree dataType="VOID" lineNo="12" nodeType="fun_declaration"
      string="main" value="none">
    <Child>
      <Tree dataType="none" lineNo="11" nodeType="block_item_list"
          string="none" value="none">
        <Child>
          <Tree dataType="none" lineNo="5" nodeType="block_item_list"
              string="none" value="none">
            <Child>
              <Tree dataType="none" lineNo="4" nodeType="block_item_list"
                  string="none" value="none">
                <Child>
                  <Tree dataType="INT" lineNo="3"
                      nodeType="var_declaration" string="none"
                      value="none">
                    <Child>
                      <Tree dataType="none" lineNo="3"
                          nodeType="init_declarator_list" string="sum1"
                          value="none">
                        <Child>
                          <Tree dataType="none" lineNo="3"
                              nodeType="factor" string="none"
                              value="100"></Tree>
                        </Child>
                      </Tree>
                    </Child>
                  </Tree>
                  <Tree dataType="INT" lineNo="4"
                      nodeType="var_declaration" string="none"
                      value="none">
                    <Child>
                      <Tree dataType="none" lineNo="4"
                          nodeType="init_declarator_list" string="a"
```

```
                value="none"></Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
    <Tree dataType="INT" lineNo="5" nodeType="var_declaration"
        string="none" value="none">
      <Child>
        <Tree dataType="none" lineNo="5"
            nodeType="init_declarator_list" string="b"
            value="none"></Tree>
      </Child>
    </Tree>
  </Child>
</Tree>
<Tree dataType="none" lineNo="11" nodeType="if_stmt"
    string="none" value="none">
  <Child>
    <Tree dataType="==" lineNo="6" nodeType="simple_expression"
        string="none" value="none">
      <Child>
        <Tree dataType="none" lineNo="6" nodeType="factor"
            string="none" value="1"></Tree>
        <Tree dataType="+" lineNo="6"
            nodeType="additive_expression" string="none"
            value="none">
          <Child>
            <Tree dataType="none" lineNo="6" nodeType="factor"
                string="none" value="1"></Tree>
            <Tree dataType="none" lineNo="6" nodeType="factor"
                string="none" value="0"></Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
    <Tree dataType="none" lineNo="7"
        nodeType="assignment_expression" string="a"
        value="none">
      <Child>
        <Tree dataType="none" lineNo="7" nodeType="factor"
            string="none" value="3"></Tree>
      </Child>
    </Tree>
    <Tree dataType="none" lineNo="10"
        nodeType="assignment_expression" string="b"
        value="none">
      <Child>
        <Tree dataType="none" lineNo="10" nodeType="factor"
            string="none" value="5"></Tree>
      </Child>
```

```
            </Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
  </Child>
 </Tree>
</root>
```

c_0 语言文件 test2.c

```
int foo(int a,int b){
    return a+b;
}
void bar(int a){
    if (a==0){
        exit(-1);
    }
    else{
        exit(0);
    }
}
void main(){
    int yylineno;
    int t;
    int yytext;
    t = foo(3,4);
    bar(t);
}
```

c_0 语言文件 test2.c的输出

```
<?xml version="1.0"?>
<root>
  <Tree dataType="none" lineNo="18" nodeType="program" string="none"
      value="none">
    <Child>
      <Tree dataType="none" lineNo="11" nodeType="program" string="none"
          value="none">
        <Child>
          <Tree dataType="INT" lineNo="3" nodeType="fun_declaration"
              string="foo" value="none">
            <Child>
              <Tree dataType="none" lineNo="2" nodeType="return_stmt"
                  string="none" value="none">
                <Child>
                  <Tree dataType="+" lineNo="2"
                      nodeType="additive_expression" string="none"
                      value="none">
                    <Child>
```

```
          <Tree dataType="none" lineNo="2" nodeType="factor"
              string="a" value="none"></Tree>
          <Tree dataType="none" lineNo="2" nodeType="factor"
              string="b" value="none"></Tree>
        </Child>
      </Tree>
    </Child>
  </Tree>
 </Child>
</Tree>
<Tree dataType="VOID" lineNo="11" nodeType="fun_declaration"
    string="bar" value="none">
 <Child>
   <Tree dataType="none" lineNo="10" nodeType="if_stmt"
       string="none" value="none">
     <Child>
       <Tree dataType="==" lineNo="5"
           nodeType="simple_expression" string="none"
           value="none">
         <Child>
           <Tree dataType="none" lineNo="5" nodeType="factor"
               string="a" value="none"></Tree>
           <Tree dataType="none" lineNo="5" nodeType="factor"
               string="none" value="0"></Tree>
         </Child>
       </Tree>
       <Tree dataType="none" lineNo="6" nodeType="call"
           string="exit" value="none">
         <Child>
           <Tree dataType="-" lineNo="6"
               nodeType="additive_expression" string="none"
               value="none">
             <Child>
               <Tree dataType="none" lineNo="6"
                   nodeType="factor" string="none"
                   value="1"></Tree>
             </Child>
           </Tree>
         </Child>
       </Tree>
       <Tree dataType="none" lineNo="9" nodeType="call"
           string="exit" value="none">
         <Child>
           <Tree dataType="none" lineNo="9" nodeType="factor"
               string="none" value="0"></Tree>
         </Child>
       </Tree>
     </Child>
   </Tree>
 </Child>
```

```
    </Tree>
  </Child>
</Tree>
<Tree dataType="VOID" lineNo="18" nodeType="fun_declaration"
     string="main" value="none">
  <Child>
    <Tree dataType="none" lineNo="17" nodeType="block_item_list"
         string="none" value="none">
      <Child>
        <Tree dataType="none" lineNo="16"
             nodeType="block_item_list" string="none" value="none">
          <Child>
            <Tree dataType="none" lineNo="15"
                 nodeType="block_item_list" string="none"
                 value="none">
              <Child>
                <Tree dataType="none" lineNo="14"
                     nodeType="block_item_list" string="none"
                     value="none">
                  <Child>
                    <Tree dataType="INT" lineNo="13"
                         nodeType="var_declaration" string="none"
                         value="none">
                      <Child>
                        <Tree dataType="none" lineNo="13"
                             nodeType="init_declarator_list"
                             string="yylineno" value="none"></Tree>
                      </Child>
                    </Tree>
                    <Tree dataType="INT" lineNo="14"
                         nodeType="var_declaration" string="none"
                         value="none">
                      <Child>
                        <Tree dataType="none" lineNo="14"
                             nodeType="init_declarator_list"
                             string="t" value="none"></Tree>
                      </Child>
                    </Tree>
                  </Child>
                </Tree>
                <Tree dataType="INT" lineNo="15"
                     nodeType="var_declaration" string="none"
                     value="none">
                  <Child>
                    <Tree dataType="none" lineNo="15"
                         nodeType="init_declarator_list"
                         string="yytext" value="none"></Tree>
                  </Child>
                </Tree>
              </Child>
```

```
                </Tree>
                <Tree dataType="none" lineNo="16"
                    nodeType="assignment_expression" string="t"
                    value="none">
                  <Child>
                    <Tree dataType="none" lineNo="16" nodeType="call"
                        string="foo" value="none">
                      <Child>
                        <Tree dataType="none" lineNo="16"
                            nodeType="args" string="none" value="none">
                          <Child>
                            <Tree dataType="none" lineNo="16"
                                nodeType="factor" string="none"
                                value="3"></Tree>
                            <Tree dataType="none" lineNo="16"
                                nodeType="factor" string="none"
                                value="4"></Tree>
                          </Child>
                        </Tree>
                      </Child>
                    </Tree>
                  </Child>
                </Tree>
              </Child>
            </Tree>
            <Tree dataType="none" lineNo="17" nodeType="call"
                string="bar" value="none">
              <Child>
                <Tree dataType="none" lineNo="17" nodeType="factor"
                    string="t" value="none"></Tree>
              </Child>
            </Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
  </Child>
</Tree>
</root>
```

c_0 语言文件 testlex.c

```
void f1(int a,int b) {
  a = 1;
  b = a+b;
}
void main()
{
  int a[100];
  int b;
```

```
 float c;
  a[b]=a;
  if(c<b){
    f1(a,b);
  }
}
```

c_0 语言文件 testlex.c的输出

```
<?xml version="1.0"?>
<root></root>
```

c_0 语言文件 testparser.c

```
int a;
int b;
int d,e,f;

void main()
{
   int a=0;
   int b=2;
   while(a==b)
     ++a;
}
```

c_0 语言文件 testparser.c的输出

```
<?xml version="1.0"?>
<root>
  <Tree dataType="none" lineNo="11" nodeType="program" string="none"
      value="none">
    <Child>
      <Tree dataType="none" lineNo="3" nodeType="program" string="none"
          value="none">
        <Child>
          <Tree dataType="none" lineNo="2" nodeType="program"
              string="none" value="none">
            <Child>
              <Tree dataType="INT" lineNo="1" nodeType="var_declaration"
                  string="none" value="none">
                <Child>
                  <Tree dataType="none" lineNo="1"
                      nodeType="init_declarator_list" string="a"
                      value="none"></Tree>
                </Child>
              </Tree>
              <Tree dataType="INT" lineNo="2" nodeType="var_declaration"
                  string="none" value="none">
```

```
          <Child>
            <Tree dataType="none" lineNo="2"
                nodeType="init_declarator_list" string="b"
                value="none"></Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
    <Tree dataType="INT" lineNo="3" nodeType="var_declaration"
        string="none" value="none">
      <Child>
        <Tree dataType="none" lineNo="3"
            nodeType="init_declarator_list" string="f" value="none">
          <Child>
            <Tree dataType="none" lineNo="3"
                nodeType="init_declarator_list" string="e"
                value="none">
              <Child>
                <Tree dataType="none" lineNo="3"
                    nodeType="init_declarator_list" string="d"
                    value="none"></Tree>
              </Child>
            </Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
  </Child>
</Tree>
<Tree dataType="VOID" lineNo="11" nodeType="fun_declaration"
    string="main" value="none">
  <Child>
    <Tree dataType="none" lineNo="10" nodeType="block_item_list"
        string="none" value="none">
      <Child>
        <Tree dataType="none" lineNo="8" nodeType="block_item_list"
            string="none" value="none">
          <Child>
            <Tree dataType="INT" lineNo="7"
                nodeType="var_declaration" string="none"
                value="none">
              <Child>
                <Tree dataType="none" lineNo="7"
                    nodeType="init_declarator_list" string="a"
                    value="none">
                  <Child>
                    <Tree dataType="none" lineNo="7"
                        nodeType="factor" string="none"
                        value="0"></Tree>
                  </Child>
```

```
                    </Tree>
                  </Child>
                </Tree>
                <Tree dataType="INT" lineNo="8"
                    nodeType="var_declaration" string="none"
                    value="none">
                  <Child>
                    <Tree dataType="none" lineNo="8"
                        nodeType="init_declarator_list" string="b"
                        value="none">
                      <Child>
                        <Tree dataType="none" lineNo="8"
                            nodeType="factor" string="none"
                            value="2"></Tree>
                      </Child>
                    </Tree>
                  </Child>
                </Tree>
              </Child>
            </Tree>
            <Tree dataType="none" lineNo="10" nodeType="while_stmt"
                string="none" value="none">
              <Child>
                <Tree dataType="==" lineNo="9"
                    nodeType="simple_expression" string="none"
                    value="none">
                  <Child>
                    <Tree dataType="none" lineNo="9" nodeType="factor"
                        string="a" value="none"></Tree>
                    <Tree dataType="none" lineNo="9" nodeType="factor"
                        string="b" value="none"></Tree>
                  </Child>
                </Tree>
                <Tree dataType="++" lineNo="10"
                    nodeType="unary_expression" string="a"
                    value="none"></Tree>
              </Child>
            </Tree>
          </Child>
        </Tree>
      </Child>
    </Tree>
  </Child>
</Tree>
</root>
```