实验报告

实验二：语法分析

1.1 实验目的和要求

1. 设计并实现一个Cminus语言的语法分析器，能够解析Cminus语言的语法结构。

2. 构建抽象语法树（AST）以表示Cminus语言程序的语法结构。

3. 实现错误检测和恢复机制，确保语法分析器的健壮性。

实验难点

1.了解bison的基础知识和理解Cminus-f语法

2.了解bison与flex之间是如何协同工作，flex对bison的影响

3.参考所给的语法规则，实验要求的上下无关文法的正确编写

1.2 实验环境

1. 硬件环境：鲲鹏开发板。

2. 软件环境：openEuler操作系统。

3. 测试环境：提供一组Cminus语言的测试代码。

1.3 实验内容

1. 设计并实现一个语法分析器，能够解析Cminus语言的语法结构。

2. 编写代码以枚举类型定义Cminus语言的词法单元。

3. 实现语法分析器的核心功能，包括解析语法、构建AST。

1.4 实验过程

1.4.1 定义词法单元

定义一个枚举类型`Token`，包含Cminus语言的所有词法单元，如下所示：

|  |
| --- |
| typedef enum cminus\_token\_type {  // 运算符  ADD = 259, // +  SUB = 260, // -  MUL = 261, // \*  DIV = 262, // /  LT = 263, // <  LTE = 264, // <=  GT = 265, // >  GTE = 266, // >=  EQ = 267, // ==  NEQ = 268, // !=  ASSIN = 269, // =  // 符号  SEMICOLON = 270, // ;  COMMA = 271, // ,  LPARENTHESE = 272, // (  RPARENTHESE = 273, // )  LBRACKET = 274, // [  RBRACKET = 275, // ]  LBRACE = 276, // {  RBRACE = 277, // }  // 关键字  ELSE = 278, // else  IF = 279, // if  INT = 280, // int  FLOAT = 281, // float  RETURN = 282, // return  VOID = 283, // void  WHILE = 284, // while  // 标识符和数值  IDENTIFIER = 285, // 标识符，如变量名、函数名等  INTEGER = 286, // 整数值，如1、2、3等  FLOATPOINT = 287, // 浮点数值，如1.1、2.1等  ARRAY = 288, // 数组声明，如int data[10]等  LETTER = 289, // 字母字符，如'a'等  // 其他  EOL = 290, // 行终止符，如'\n'等  COMMENT = 291, // 注释，如// ...等  BLANK = 292, // 空白字符，如空格、制表符等  ERROR = 258 // 错误或无法识别的标记  } Token; |

1.4.2 构建语法分析器

1.4.2.1 词法分析

利用`flex`工具，根据定义的词法单元，编写词法分析器，将源代码转换为Token流。

将lab1的词法部分复制到lab2的词法部分的文件中，并进行修改，将所有的动作都直接放到语法规则的后面

进行分析得知，该实验中要构建一颗树，在该树种所有识别的词为树的叶子节点。观察.\src\parser\lexical\_analyzer.l文件

|  |
| --- |
| void pass\_node(char \*text){  yylval.node =new\_syntax\_tree\_node(text);  } |

该函数为构建名字为text的叶子节点。

所以要修改.\src\parser\lexical\_analyzer.l文件，通过观察得知该文件中也不需要analyzer函数，去除了analyzer函数，但是analyzer函数的功能还是需要保留着。

最终的.l文件修改如下

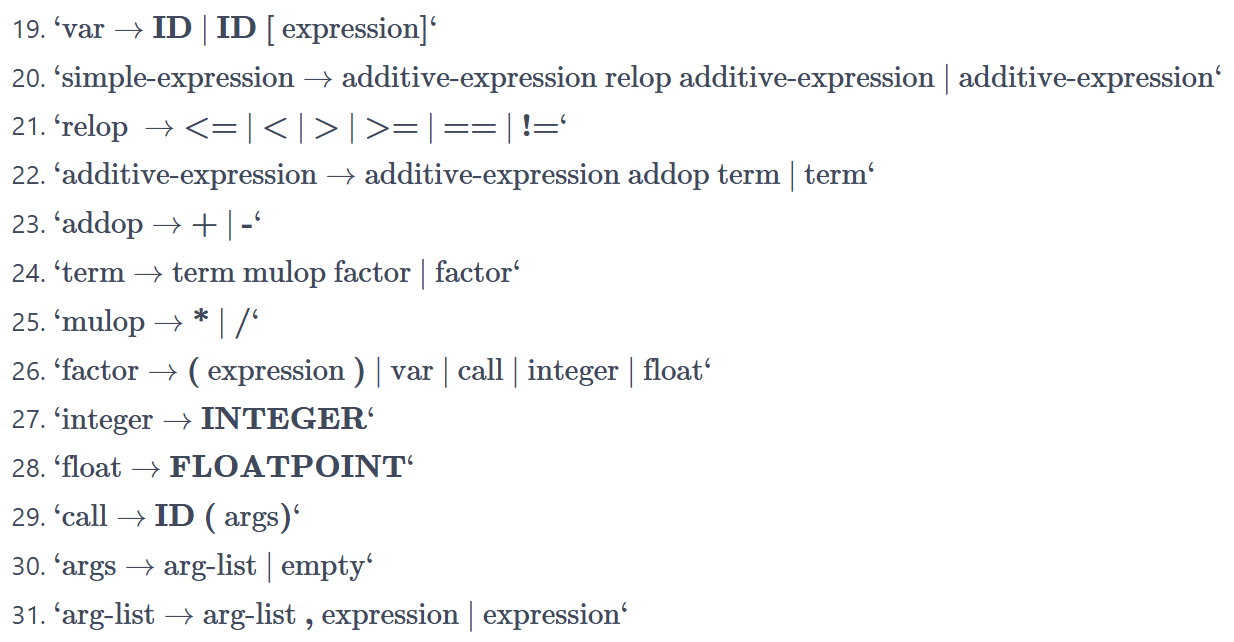
|  |
| --- |
| %option noyywrap  %{  #include <stdio.h>  #include <stdlib.h>  #include "syntax\_tree.h"  #include "syntax\_analyzer.h"  int files\_count;  int lines;  int pos\_start;  int pos\_end;  void pass\_node(char \*text){  yylval.node = new\_syntax\_tree\_node(text);  }  %}  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  TO STUDENTS: Copy your Lab1 here. Make adjustments if necessary.  Note: don't modify the prologue unless you know what you are doing.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  IDENTIFIER [a-zA-Z]+  INTEGER 0|[1-9][0-9]\*  FLOATPOINT [0-9]\*\.[0-9]\*  ARRAY \[\]  LETTER \'.\'  COMMENT ("//"[^\n]\*)|("/\*"([^\*]|\\*+[^\*/])\*\\*+"/")  BLANK ([ ]|[\t])+  %%  "+" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return ADD;}  "-" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return SUB;}  "\*" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return MUL;}  "/" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return DIV;}  "<" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LT;}  "<=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LTE;}  ">" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return GT;}  ">=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return GTE;}  "==" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return EQ;}  "!=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return NEQ;}  "=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return ASSIN;}  ";" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return SEMICOLON;}  "," {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return COMMA;}  "(" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LPARENTHESE;}  ")" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return RPARENTHESE;}  "[" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LBRACKET;}  "]" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return RBRACKET;}  "{" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LBRACE;}  "}" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return RBRACE;}  "else" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return ELSE;}  "if" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return IF;}  "int" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return INT;}  "float" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return FLOAT;}  "return" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return RETURN;}  "void" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return VOID;}  "while" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return WHILE;}  {IDENTIFIER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return IDENTIFIER;}  {INTEGER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return INTEGER;}  {FLOATPOINT} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return FLOATPOINT;}  {ARRAY} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return ARRAY;}  {LETTER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;pass\_node(yytext);return LETTER;}  [\n]+ {pos\_start=1;pos\_end=1;lines+=yyleng;}  {COMMENT} {for(int i=0;i<yyleng;i++)  if(yytext[i]=='\n'){  pos\_end=1;  pos\_start=1;  lines++;  }else pos\_end++;  }  {BLANK} {pos\_start = pos\_end;pos\_end+=yyleng;}  . {printf("error!\n");}  %%  // /\* Example for you :-) \*/  // \+ { pos\_start = pos\_end; pos\_end += 1; pass\_node(yytext); return ADD; } |

1.4.2.2 语法分析

利用`bison`工具，根据Cminus语言的语法规则，编写语法分析器，解析Token流并构建AST。

* 还需要补全.\src\parser\syntax\_analyzer.y

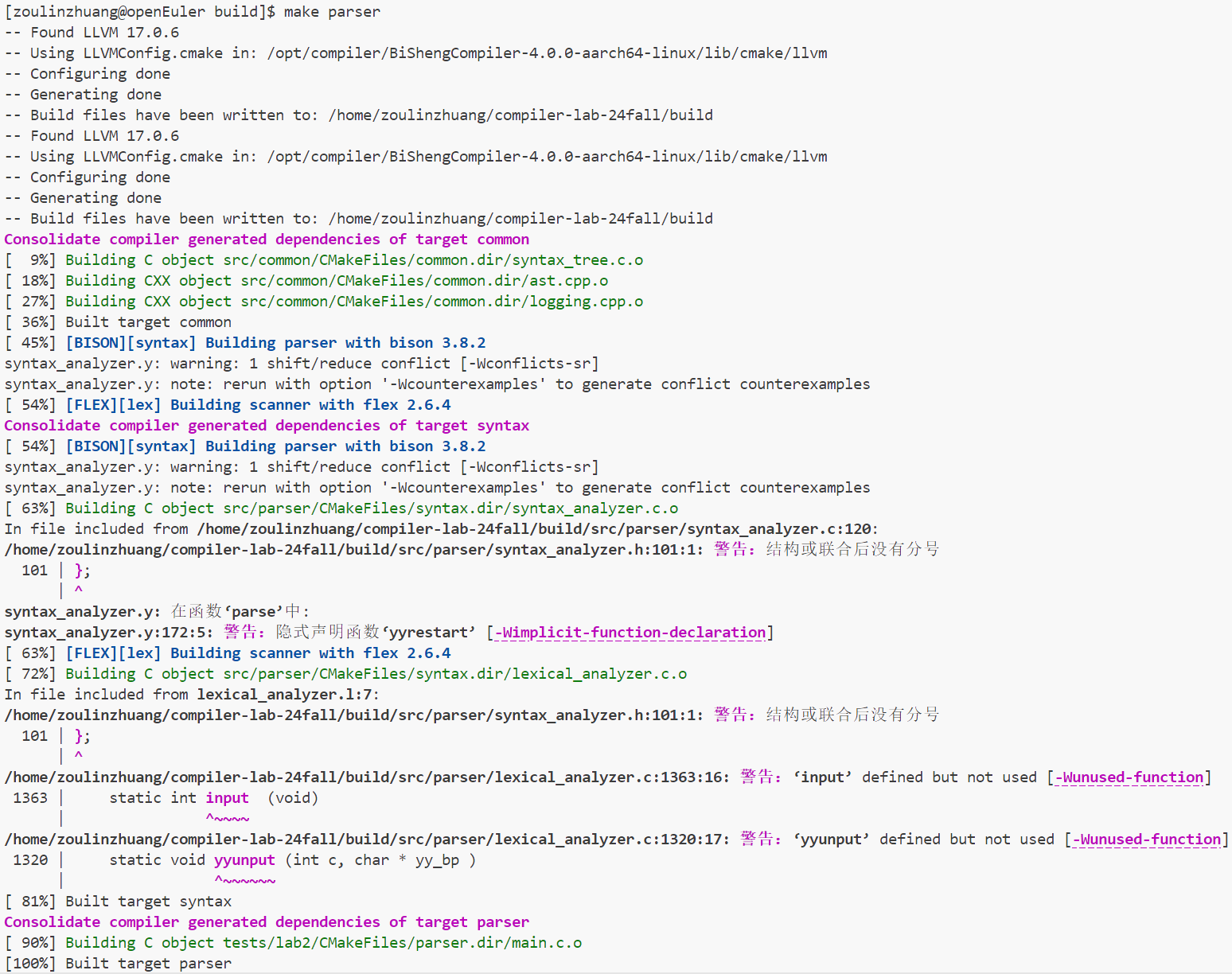


 根据以上下文无关文法，补全文件，补全部分：

|  |
| --- |
| %{  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <stdarg.h>  #include "syntax\_tree.h"  *// external functions from lex*  **extern** **int** **yylex**();  **extern** FILE\* yyin;  *// external variables from lexical\_analyzer module*  **extern** **int** lines;  **extern** **char** \*yytext;  **extern** **int** pos\_end;  **extern** **int** pos\_start;  *// Global syntax tree*  syntax\_tree \*gt;  *// Error reporting*  **void** **yyerror**(**const** **char** \*s);  *// Helper functions written for you with love*  syntax\_tree\_node \***node**(**const** **char** \*node\_name, **int** children\_num, ...);  %}  */\* TODO: Complete this definition. \*/*  %**union** {syntax\_tree\_node \*node}  %type <node> program declaration-list declaration var-declaration  %type <node> type-specifier fun-declaration params param-list param  %type <node> compound-stmt local-declarations statement-list statement  %type <node> expression expression-stmt selection-stmt iteration-stmt **return**-stmt  %type <node> var simple-expression relop additive-expression  %type <node> addop term mulop factor integer **float** call args arg-list  %token <node> ADD SUB MUL DIV LT LTE GT GTE EQ NEQ ASSIN  %token <node> SEMICOLON COMMA LPARENTHESE RPARENTHESE LBRACKET RBRACKET LBRACE RBRACE  %token <node> ELSE IF INT FLOAT RETURN VOID WHILE  %token <node> IDENTIFIER INTEGER FLOATPOINT ARRAY LETTER  */\* TODO: Your tokens here. \*/*  %start program  %%  */\* TODO: Your rules here. \*/*  program : declaration-list { $$ = node("program", 1, $1); gt->root = $$; }  ;  declaration-list : declaration-list declaration { $$ = node("declaration-list", 2, $1,$2);}  | declaration { $$ = node("declaration-list", 1, $1);}  ;  declaration : var-declaration { $$ = node("declaration", 1, $1);}  | fun-declaration { $$ = node("declaration", 1, $1);}  ;  var-declaration : type-specifier IDENTIFIER SEMICOLON {$$ = node("var-declaration",3,$1,$2,$3);}  | type-specifier IDENTIFIER LBRACKET INTEGER RBRACKET SEMICOLON {$$ = node("var-declaration",6,$1,$2,$3,$4,$5,$6);}  ;  type-specifier : INT { $$ = node("type-specifier", 1, $1);}  | FLOAT { $$ = node("type-specifier", 1, $1);}  | VOID { $$ = node("type-specifier", 1, $1);}  ;  fun-declaration : type-specifier IDENTIFIER LPARENTHESE params RPARENTHESE compound-stmt { $$ = node("fun-declaration", 6, $1,$2,$3,$4,$5,$6);}  ;  params : param-list { $$ = node("params", 1, $1);}  | VOID { $$ = node("params", 1, $1);}  ;  param-list : param-list COMMA param { $$ = node("param-list", 3, $1,$2,$3);}  | param { $$ = node("param-list", 1, $1);}  ;  param : type-specifier IDENTIFIER { $$ = node("param", 2, $1,$2);}  | type-specifier IDENTIFIER ARRAY { $$ = node("param", 3, $1,$2,$3);}  ;  compound-stmt : LBRACE local-declarations statement-list RBRACE { $$ = node("compound-stmt", 4, $1,$2,$3,$4);}  ;  local-declarations : local-declarations var-declaration { $$ = node("local-declarations", 2, $1,$2);}  | { $$ = node("local-declarations", 0);}  ;  statement-list : statement-list statement { $$ = node("statement-list", 2, $1,$2);}  | {$$ = node("statement-list",0);}  ;  statement : expression-stmt { $$ = node("statement", 1, $1);}  | compound-stmt { $$ = node("statement", 1, $1);}  | selection-stmt { $$ = node("statement", 1, $1);}  | iteration-stmt { $$ = node("statement", 1, $1);}  | **return**-stmt { $$ = node("statement", 1, $1);}  ;  expression-stmt : expression SEMICOLON { $$ = node("expression-stmt", 2, $1,$2);}  | SEMICOLON { $$ = node("expression-stmt", 1, $1);}  ;  selection-stmt : IF LPARENTHESE expression RPARENTHESE statement { $$ = node("selection-stmt", 5, $1,$2,$3,$4,$5);}  | IF LPARENTHESE expression RPARENTHESE statement ELSE statement { $$ = node("selection-stmt", 7, $1,$2,$3,$4,$5,$6,$7);}  ;  iteration-stmt : WHILE LPARENTHESE expression RPARENTHESE statement { $$ = node("iteration-stmt", 5, $1,$2,$3,$4,$5);}  ;  **return**-stmt : RETURN SEMICOLON { $$ = node("return-stmt", 2, $1,$2);}  | RETURN expression SEMICOLON { $$ = node("return-stmt", 3, $1,$2,$3);}  ;  expression : var ASSIN expression { $$ = node("expression", 3, $1,$2,$3);}  | simple-expression { $$ = node("expression", 1, $1);}  ;  var : IDENTIFIER { $$ = node("var", 1, $1);}  | IDENTIFIER LBRACKET expression RBRACKET { $$ = node("var", 4, $1,$2,$3,$4);}  ;  simple-expression : additive-expression relop additive-expression { $$ = node("simple-expression", 3, $1,$2,$3);}  | additive-expression { $$ = node("simple-expression", 1, $1);}  ;  relop : LTE { $$ = node("relop", 1, $1);}  | LT { $$ = node("relop", 1, $1);}  | GT { $$ = node("relop", 1, $1);}  | GTE { $$ = node("relop", 1, $1);}  | EQ { $$ = node("relop", 1, $1);}  | NEQ { $$ = node("relop", 1, $1);}  ;  additive-expression : additive-expression addop term { $$ = node("additive-expression", 3, $1,$2,$3);}  | term { $$ = node("additive-expression", 1, $1);}  ;  addop : ADD { $$ = node("addop", 1, $1);}  | SUB { $$ = node("addop", 1, $1);}  ;  term : term mulop factor { $$ = node("term", 3, $1,$2,$3);}  | factor { $$ = node("term", 1, $1);}  ;  mulop : MUL { $$ = node("mulop", 1, $1);}  | DIV { $$ = node("mulop", 1, $1);}  ;  factor : LPARENTHESE expression RPARENTHESE { $$ = node("factor", 3, $1,$2,$3);}  | var { $$ = node("factor", 1, $1);}  | call { $$ = node("factor", 1, $1);}  | integer { $$ = node("factor", 1, $1);}  | **float** { $$ = node("factor", 1, $1);}  ;  integer : INTEGER { $$ = node("integer", 1, $1);}  ;  **float** : FLOATPOINT { $$ = node("float", 1, $1);}  ;  call : IDENTIFIER LPARENTHESE args RPARENTHESE { $$ = node("call", 4, $1,$2,$3,$4);}  ;  args : arg-list { $$ = node("args", 1, $1);}  | { $$ = node("args", 0);}  ;  arg-list : arg-list COMMA expression { $$ = node("arg-list", 3, $1,$2,$3);}  | expression { $$ = node("arg-list", 1, $1);}  ;  %%  */// The error reporting function.*  **void** **yyerror**(**const** **char** \*s)  {  *// TO STUDENTS: This is just an example.*  *// You can customize it as you like.*  fprintf(stderr, "error at line %d column %d: %s\n", lines, pos\_start, s);  }  */// Parse input from file `input\_path`, and prints the parsing results*  */// to stdout. If input\_path is NULL, read from stdin.*  *///*  */// This function initializes essential states before running yyparse().*  syntax\_tree \***parse**(**const** **char** \*input\_path)  {  **if** (input\_path != NULL) {  **if** (!(yyin = fopen(input\_path, "r"))) {  fprintf(stderr, "[ERR] Open input file %s failed.\n", input\_path);  exit(1);  }  } **else** {  yyin = stdin;  }  lines = pos\_start = pos\_end = 1;  gt = new\_syntax\_tree();  yyrestart(yyin);  yyparse();  **return** gt;  }  */// A helper function to quickly construct a tree node.*  *///*  */// e.g.*  */// $$ = node("program", 1, $1);*  */// $$ = node("local-declarations", 0);*  syntax\_tree\_node \***node**(**const** **char** \*name, **int** children\_num, ...)  {  syntax\_tree\_node \*p = new\_syntax\_tree\_node(name);  syntax\_tree\_node \*child;  **if** (children\_num == 0) {  child = new\_syntax\_tree\_node("epsilon");  syntax\_tree\_add\_child(p, child);  } **else** {  va\_list ap;  va\_start(ap, children\_num);  **for** (**int** i = 0; i < children\_num; ++i) {  child = va\_arg(ap, syntax\_tree\_node \*);  syntax\_tree\_add\_child(p, child);  }  va\_end(ap);  }  **return** p;  } |

1.5 实验结果验证

1.5.1编译

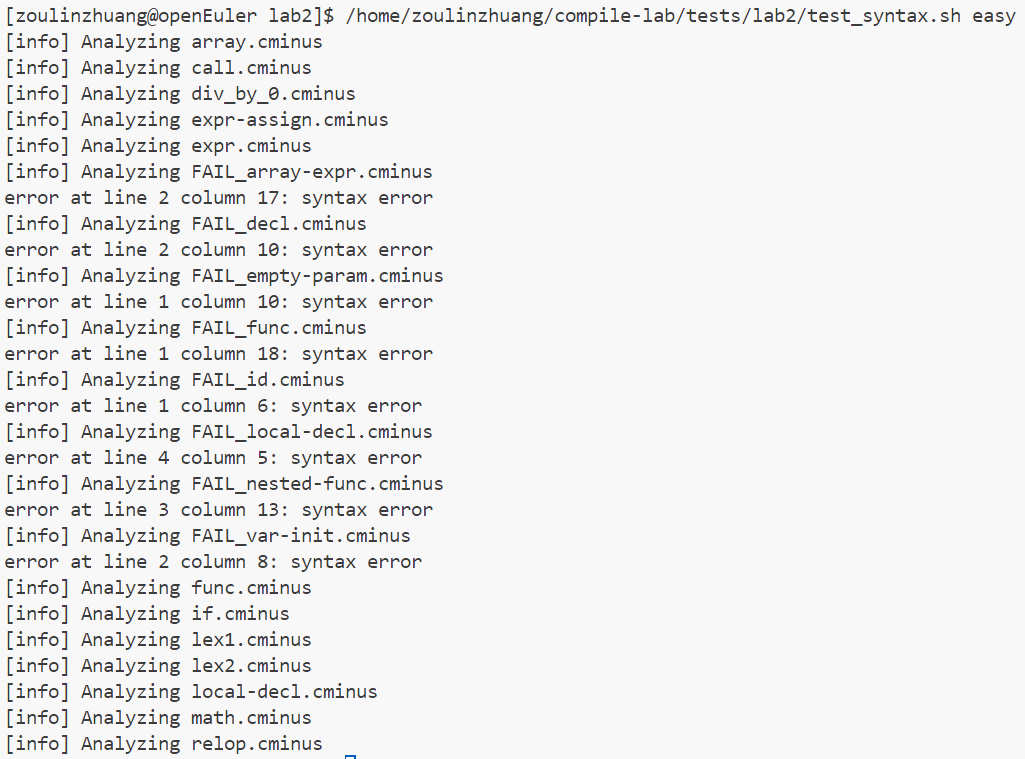
首先进行编译  编译成功

1.5.2 验证语法分析器的正确性

通过编写测试用例，验证语法分析器能够正确解析Cminus语言的语法结构，并构建正确的AST。

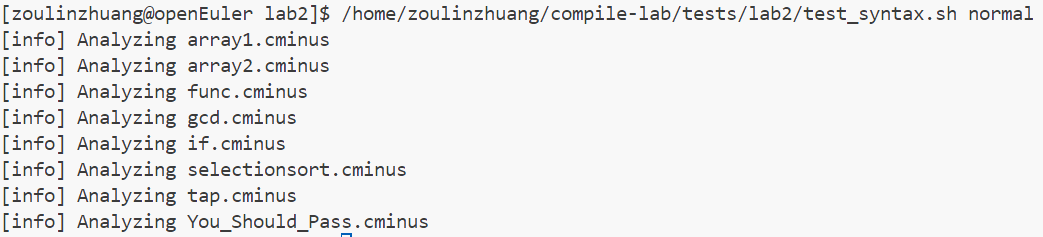
接着进行样例的测试

分别测试easy测试案例，normal测试案例

easy测试案例： 

alt text

可以看到easy测试案例的输出结果与答案完全一致

normal测试案例 

alt text

可以看到normal测试案例的输出结果与答案完全一致

1.5.3 验证语法分析器的健壮性

通过提供包含错误输入和边界情况的测试用例，验证语法分析器的健壮性。

然后是自己测试的样例

|  |
| --- |
| **int** a;  **int** **cmp**(**int** a, **int** b)  {  **int** c;  c = a;  **if**(a > b)  {  c = a - b;  }  **else**  {  c = b - a;  }    **return** c;  }  **void** **function**(**int** a){  **int** i;  **int** sum;  i = 0;  sum = 0;  **while**(i<100)  {  sum = sum + i;  i = i+1;  }  } |

最后得到的输出为

|  |
| --- |
| >--+ program  | >--+ declaration-list  | | >--+ declaration-list  | | | >--+ declaration-list  | | | | >--+ declaration  | | | | | >--+ var-declaration  | | | | | | >--+ type-specifier  | | | | | | | >--\* **int**  | | | | | | >--\* a  | | | | | | >--\* ;  | | | >--+ declaration  | | | | >--+ fun-declaration  | | | | | >--+ type-specifier  | | | | | | >--\* **int**  | | | | | >--\* cmp  | | | | | >--\* (  | | | | | >--+ params  | | | | | | >--+ param-list  | | | | | | | >--+ param-list  | | | | | | | | >--+ param  | | | | | | | | | >--+ type-specifier  | | | | | | | | | | >--\* **int**  | | | | | | | | | >--\* a  | | | | | | | >--\* ,  | | | | | | | >--+ param  | | | | | | | | >--+ type-specifier  | | | | | | | | | >--\* **int**  | | | | | | | | >--\* b  | | | | | >--\* )  | | | | | >--+ compound-stmt  | | | | | | >--\* {  | | | | | | >--+ local-declarations  | | | | | | | >--+ local-declarations  | | | | | | | | >--\* epsilon  | | | | | | | >--+ var-declaration  | | | | | | | | >--+ type-specifier  | | | | | | | | | >--\* **int**  | | | | | | | | >--\* c  | | | | | | | | >--\* ;  | | | | | | >--+ statement-list  | | | | | | | >--+ statement-list  | | | | | | | | >--+ statement-list  | | | | | | | | | >--+ statement-list  | | | | | | | | | | >--\* epsilon  | | | | | | | | | >--+ statement  | | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | >--\* c  | | | | | | | | | | | | >--\* =  | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | >--\* a  | | | | | | | | | | | >--\* ;  | | | | | | | | >--+ statement  | | | | | | | | | >--+ selection-stmt  | | | | | | | | | | >--\* **if**  | | | | | | | | | | >--\* (  | | | | | | | | | | >--+ expression  | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | >--\* a  | | | | | | | | | | | | >--+ relop  | | | | | | | | | | | | | >--\* >  | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | >--\* b  | | | | | | | | | | >--\* )  | | | | | | | | | | >--+ statement  | | | | | | | | | | | >--+ compound-stmt  | | | | | | | | | | | | >--\* {  | | | | | | | | | | | | >--+ local-declarations  | | | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | | | | >--+ statement  | | | | | | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | >--\* c  | | | | | | | | | | | | | | | | >--\* =  | | | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | | | >--\* a  | | | | | | | | | | | | | | | | | | | >--+ addop  | | | | | | | | | | | | | | | | | | | | >--\* -  | | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | | >--\* b  | | | | | | | | | | | | | | | >--\* ;  | | | | | | | | | | | | >--\* }  | | | | | | | | | | >--\* **else**  | | | | | | | | | | >--+ statement  | | | | | | | | | | | >--+ compound-stmt  | | | | | | | | | | | | >--\* {  | | | | | | | | | | | | >--+ local-declarations  | | | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | | | | >--+ statement  | | | | | | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | >--\* c  | | | | | | | | | | | | | | | | >--\* =  | | | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | | | >--\* b  | | | | | | | | | | | | | | | | | | | >--+ addop  | | | | | | | | | | | | | | | | | | | | >--\* -  | | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | | >--\* a  | | | | | | | | | | | | | | | >--\* ;  | | | | | | | | | | | | >--\* }  | | | | | | | >--+ statement  | | | | | | | | >--+ **return**-stmt  | | | | | | | | | >--\* **return**  | | | | | | | | | >--+ expression  | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | >--\* c  | | | | | | | | | >--\* ;  | | | | | | >--\* }  | | >--+ declaration  | | | >--+ fun-declaration  | | | | >--+ type-specifier  | | | | | >--\* **void**  | | | | >--\* function  | | | | >--\* (  | | | | >--+ params  | | | | | >--+ param-list  | | | | | | >--+ param  | | | | | | | >--+ type-specifier  | | | | | | | | >--\* **int**  | | | | | | | >--\* a  | | | | >--\* )  | | | | >--+ compound-stmt  | | | | | >--\* {  | | | | | >--+ local-declarations  | | | | | | >--+ local-declarations  | | | | | | | >--+ local-declarations  | | | | | | | | >--\* epsilon  | | | | | | | >--+ var-declaration  | | | | | | | | >--+ type-specifier  | | | | | | | | | >--\* **int**  | | | | | | | | >--\* i  | | | | | | | | >--\* ;  | | | | | | >--+ var-declaration  | | | | | | | >--+ type-specifier  | | | | | | | | >--\* **int**  | | | | | | | >--\* sum  | | | | | | | >--\* ;  | | | | | >--+ statement-list  | | | | | | >--+ statement-list  | | | | | | | >--+ statement-list  | | | | | | | | >--+ statement-list  | | | | | | | | | >--\* epsilon  | | | | | | | | >--+ statement  | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | >--+ expression  | | | | | | | | | | | >--+ var  | | | | | | | | | | | | >--\* i  | | | | | | | | | | | >--\* =  | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | >--+ integer  | | | | | | | | | | | | | | | | | >--\* 0  | | | | | | | | | | >--\* ;  | | | | | | | >--+ statement  | | | | | | | | >--+ expression-stmt  | | | | | | | | | >--+ expression  | | | | | | | | | | >--+ var  | | | | | | | | | | | >--\* sum  | | | | | | | | | | >--\* =  | | | | | | | | | | >--+ expression  | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | >--+ integer  | | | | | | | | | | | | | | | | >--\* 0  | | | | | | | | | >--\* ;  | | | | | | >--+ statement  | | | | | | | >--+ iteration-stmt  | | | | | | | | >--\* **while**  | | | | | | | | >--\* (  | | | | | | | | >--+ expression  | | | | | | | | | >--+ simple-expression  | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | >--+ term  | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | >--\* i  | | | | | | | | | | >--+ relop  | | | | | | | | | | | >--\* <  | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | >--+ term  | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | >--+ integer  | | | | | | | | | | | | | | >--\* 100  | | | | | | | | >--\* )  | | | | | | | | >--+ statement  | | | | | | | | | >--+ compound-stmt  | | | | | | | | | | >--\* {  | | | | | | | | | | >--+ local-declarations  | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | >--+ statement-list  | | | | | | | | | | | | | >--\* epsilon  | | | | | | | | | | | | >--+ statement  | | | | | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | >--\* sum  | | | | | | | | | | | | | | | >--\* =  | | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | | >--\* sum  | | | | | | | | | | | | | | | | | | >--+ addop  | | | | | | | | | | | | | | | | | | | >--\* +  | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | >--\* i  | | | | | | | | | | | | | | >--\* ;  | | | | | | | | | | | >--+ statement  | | | | | | | | | | | | >--+ expression-stmt  | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | >--\* i  | | | | | | | | | | | | | | >--\* =  | | | | | | | | | | | | | | >--+ expression  | | | | | | | | | | | | | | | >--+ simple-expression  | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | >--+ additive-expression  | | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | | >--+ var  | | | | | | | | | | | | | | | | | | | | | >--\* i  | | | | | | | | | | | | | | | | | >--+ addop  | | | | | | | | | | | | | | | | | | >--\* +  | | | | | | | | | | | | | | | | | >--+ term  | | | | | | | | | | | | | | | | | | >--+ factor  | | | | | | | | | | | | | | | | | | | >--+ integer  | | | | | | | | | | | | | | | | | | | | >--\* 1  | | | | | | | | | | | | | >--\* ;  | | | | | | | | | | >--\* }  | | | | | >--\* } |

1.6 实验反馈

在实现语法分析器的过程中，需要注意处理各种边界情况，如注释、空白字符等。实验过程中，通过编写测试用例来验证语法分析器的正确性，确保其能够正确处理各种输入。实验结果表明，语法分析器能够正确解析Cminus语言中的所有语法单元，并生成正确的AST。

1.7 附录

- 枚举类型定义：定义了Cminus语言的所有词法单元，用于语法分析器的实现。

- 测试用例：提供了一组测试用例，用于验证语法分析器的正确性和健壮性。

姓名：邹林壮 学号：202208040412