语法分析程序的算法思想设计方案

1. 程序主结构。

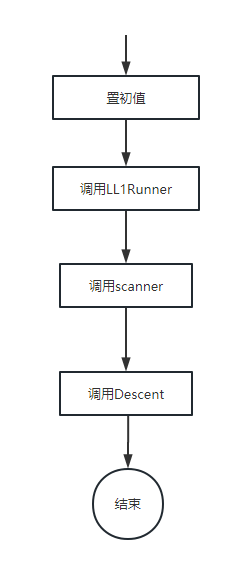


图 1语法分析主程序示意图

1. 基础文法及其语法制导翻译方案

以下是一个C语言子集的上下文无关文法的定义，为了防止分析器陷入无限循环，所以消除了该文法中的左递归。其中以@字母开头的符号是特定的语义动作。此文发不能描述C语言的所有特性，如宏定义、include语句、结构体等。

|  |
| --- |
| //不含左递归的LL(1) 文法  program -> func funcs  funcs -> func funcs | ε  func -> type IDN ( args ) @MAKE\_FUNC\_TABLE func\_body  type -> int | short | long |char |float |double| void  args -> type IDN @ARG\_INFO @ARGS arg | ε  arg -> , type IDN @ARG\_INFO arg | ε  func\_body -> ; | block  block -> { @FUNC\_INIT @GEN\_FUNC\_PROCESS\_LABEL define\_stmts stmts }  define\_stmts -> define\_stmt define\_stmts | ε  define\_stmt -> type IDN @CHECK\_VAR\_DEFINE @VAR\_INIT init vars ;  init -> = expression @CHECK\_EXPR\_TYPE @GEN\_DEFINE\_EXPR\_CODE | ε  vars -> , IDN @CHECK\_VAR\_DEFINE init vars | ε  stmts -> stmt stmts | ε  stmt -> assign\_stmt @GEN\_EXPR\_ASSIGN\_CODE | jump\_stmt | iteration\_stmt | branch\_stmt  assign\_stmt -> expression @VAR\_INIT @CHECK\_EXPR\_TYPE ;  jump\_stmt -> continue ; | break ; | return innull\_expr ;  iteration\_stmt -> while ( @LABEl\_OUT\_IN logical\_expression ) @GEN\_IF\_CODE @GEN\_TRUE\_LABEL block\_stmt @GEN\_ENTER\_CODE | for ( isnull\_expr ; isnull\_expr ; isnull\_expr ) block\_stmt | do block\_stmt while ( logical\_expression ) ;  branch\_stmt -> if (@LABEL\_OUT\_IN logical\_expression ) @GEN\_IF\_CODE @GEN\_TRUE\_LABEL block\_stmt @GEN\_FALSE\_LABEL result | switch ( IDN ) { case\_stmt case\_stmts default\_stmt }  result -> else block\_stmt | ε  logical\_expression -> ! expression @REVERS\_EQ\_OPER @GEN\_MID\_EXPR @CHECK\_BOOL\_EXPR bool\_expression | expression @GEN\_MID\_EXPR @CHECK\_BOOL\_EXPR bool\_expression  bool\_expression -> lop @GEN\_LABEL\_OR\_CODE expression @GEN\_MID\_EXPR bool\_expression | ε  //下面的or 是 ||  lop -> && | or  case\_stmts -> case\_stmt case\_stmts | ε  case\_stmt -> case const : stmts  default\_stmt -> default : stmts  block\_stmt -> { @MAKE\_CHILD\_TABLE stmts }  expression -> value operation  operation -> compare\_op value | equal\_op value | ε  compare\_op -> > | >= | < | <= | == | != @EN\_EXPRESSION\_STACK  equal\_op -> = @CAL\_LEFTID\_INFO | += | -= | \*= | /= | %=  value -> item value'  value' -> + item @GEN\_EXPR\_CODE value' | - item @GEN\_EXPR\_CODE value' | ε  item -> factor item'  item' -> \* factor @GEN\_EXPR\_CODE item' | / factor @GEN\_EXPR\_CODE item' | % factor @GEN\_EXPR\_CODE item' | ε  factor -> ( value ) | IDN @CHECK\_UNDEFINE call\_func | const  call\_func -> ( es ) @GEN\_CALL\_FUNC | ε  es -> isnull\_expr isnull\_es  isnull\_expr -> expression | ε  isnull\_es -> , isnull\_expr isnull\_es | ε  const -> num\_const | FLOAT | CHAR | STR  num\_const -> INT10 @NUM\_CONST |

1. 预测分析

为了避免不必要的回溯，提高递归下降分析器的效率，这里采用预测分析的方法。

* 1. 预测分析表

通过该文法的First集和Follow集来构造如下图的Select集，Select集中，产生式相同左部的Select集合互不相交，所以该文法是一个LL(1)型文法。



1. 递归下降分析程序示意图如图2-2所示。

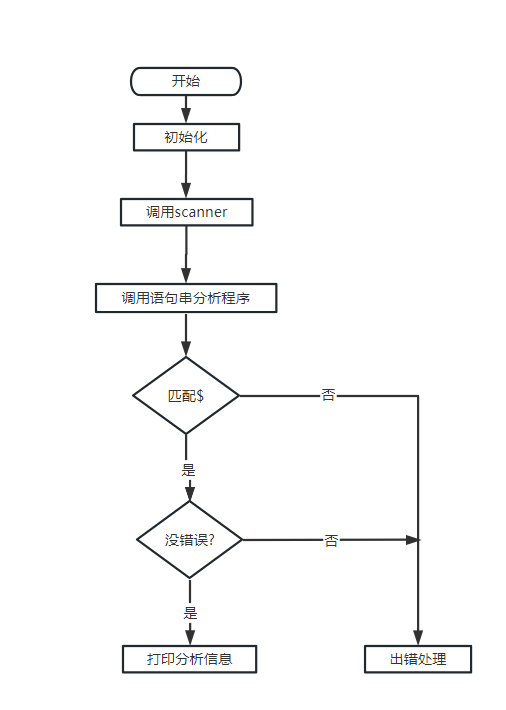


图 2 递归下降分析程序示意图

1. 语句串分析过程示意图

在上述的LL1文法中，每个非终结符都对应者一个分析函数，分析函数名均与非终结符符号名称对应，以下仅是部分分析函数流程的示意图，其它函数均用相同方法实现。

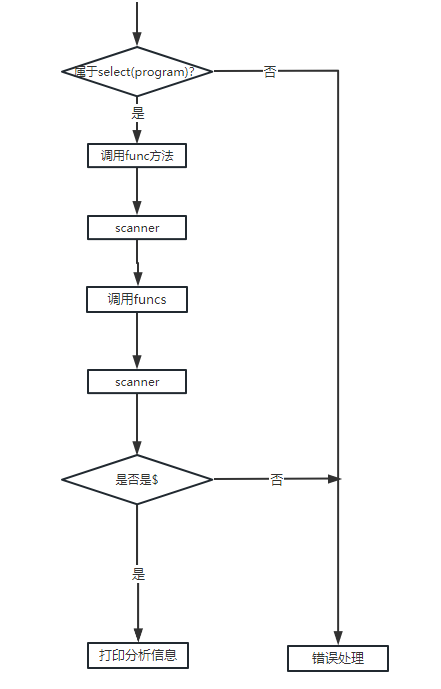


图 3program分析函数流程图

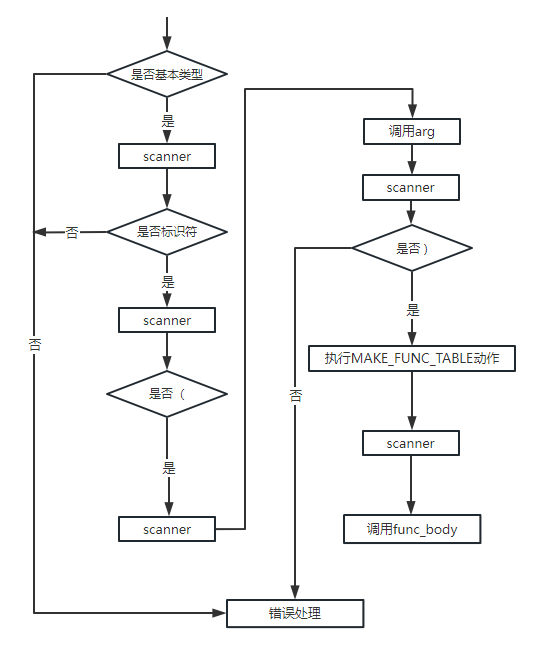


图 4 func分析函数流程图

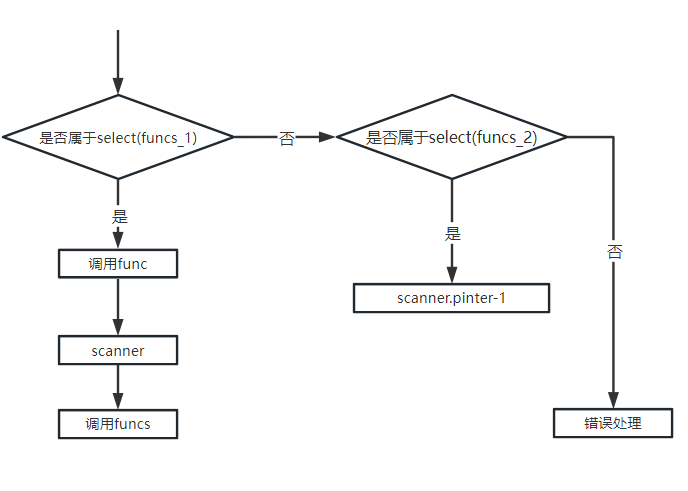


图 5 funcs分析函数流程图

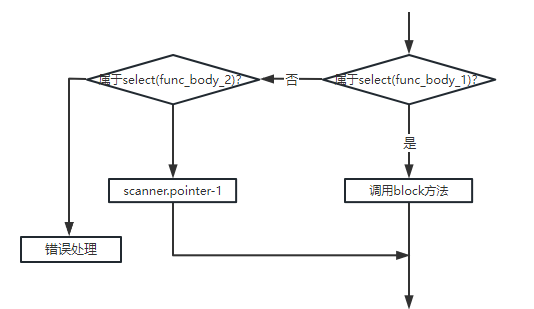


图 6 func\_body分析函数流程图

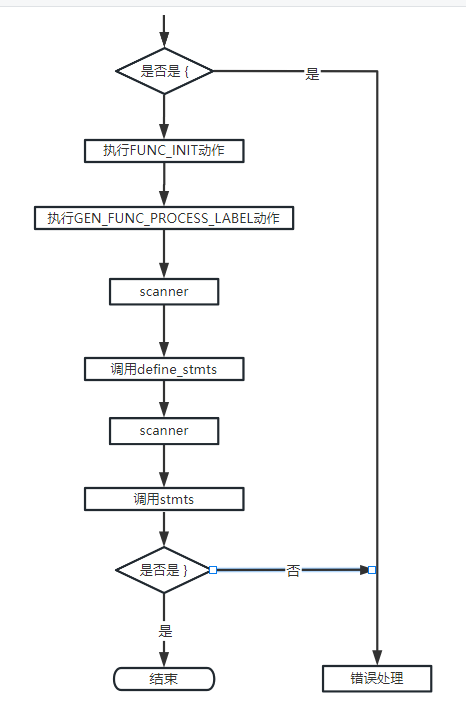


图 7 block分析函数流程图

1. 语义处理

其在递归下降分析时，在特点的位置执行特定的语义动作来达到基本的语义处理的目的。在上面的文法中，以@符号开头的大写字母标号就是其特定的语义动作，其动作将会在递归下降分析中分析完成@符号前一个文法符号时执行。其作用包括生成符号表、从符号表中添加记录、类型判断、生成中间代码等。

如在上面的产生式中，

func -> type IDN ( args ) @MAKE\_FUNC\_TABLE func\_body

是一个函数的产生式，其中插入了一条MAKE\_FUNC\_TABLE的语义动作，此动做的作用是用来构建函数符号表、以及记录函数返回类型、名称的。此动作执行时机如图8所示。

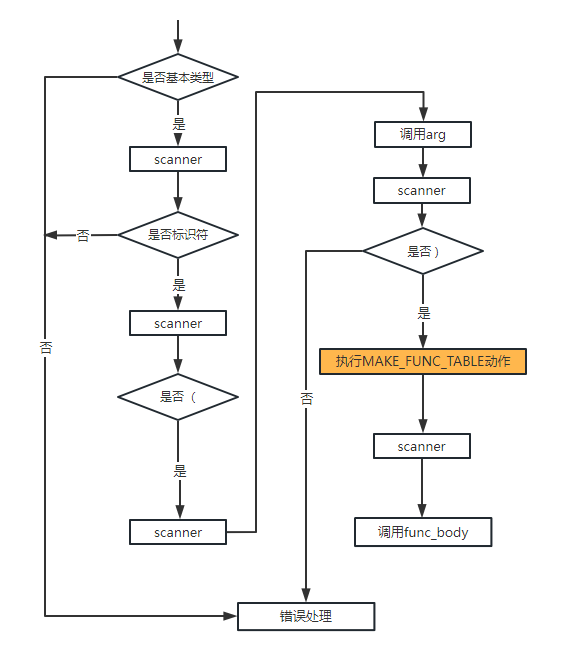


图 8 func函数语义动作处理时机

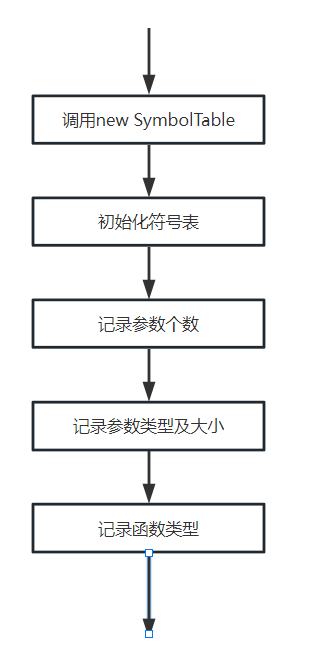


图 9 MAKE\_FUNC\_TABLE动作流程图

* 1. 语义动作

下表是部分动作标签的语义动作

表格 1 标签代表的语义动作

|  |  |
| --- | --- |
| 动作标签 | 语义动作 |
| @ARG\_INFO | 获取参数类型、大小、名称 |
| @ARGS | 将参数加入到符号表 |
| @CHECK\_VAR\_DEFINE | 从符号表检查变量是否已经定义 |
| @CHECK\_EXPR\_TYPE | 检查表达式类型是否合法，类型是子节点的综合属性 |
| @GEN\_DEFINE\_EXPR\_CODE | 生成声明语句的中间代码 |
| @GEN\_EXPR\_ASSIGN\_CODE | 生成赋值语句的中间代码 |
| @GEN\_TRUE\_LABEL | 生成特定语句为真时的出口标号 |
| @GEN\_FALSE\_LABEL | 生成特定语句为假时的出口标号 |
| @GEN\_IF\_CODE | 生成特定语句的中间代码码 |
| @MAKE\_FUNC\_TABLE | 生成符号表、记录参数、方法类型、方法名等 |
| @FUNC\_INIT | 方法已经被初始化了，记录到符号表 |
| @GEN\_FUNC\_PROCESS\_LABEL | 生成方法入口标号 |
| @VAR\_INIT | 变量初始化了，记录到符号表 |
| @LABEl\_OUT\_IN | 生成表达式为True或False的出口地址 |
| @GEN\_ENTER\_CODE | 生成跳转到循环语句入口标号地址 |
| @REVERS\_EQ\_OPER | 反转比较运算符，比如>反转为<= |
| @GEN\_MID\_EXPR | 对于由逻辑运算符连接的布尔表达式而言，生成中间表达式 |
| @CHECK\_BOOL\_EXPR | 检查是否是有效布尔表达式 |
| @EN\_EXPRESSION\_STACK | 表达式入表达式栈 |
| @NUM\_CONST | 从符号表中初始化变量值 |
| @GEN\_CALL\_FUNC | 生成调用语句中间代码，包括类型检查、参数个数是否匹配等 |