2025 编译技术实验文法定义及相关说明

概要

我们本次课程要实现的编程语言是 C 语言的一个子集。每个程序中有且仅有一个名为 main 的主函数定义，还可以包含若干全局变量声明、常量声明和其他函数定义。语言仅支持 int 类型和元素为 int 类型且按行优先存储的一维数组类型，其中 int 型整数为 32 位有符号数；const 修饰符用于声明常量。

语言通过getint与printf函数完成IO交互，函数用法已在文法中给出，需要同学们自己实现。

- **函数**：函数可以带参数也可以不带参数，参数的类型只能是 int；函数可以返回 int 类型的值，或者不返回值(即声明为 void 类型)。
- **变量/常量声明**：可以在一个变量/常量声明语句中声明多个变量或常量，声明时可以带初始化表达式。所有变量/常量要求先定义再使用。在函数外声明的为全局变量/常量，在函数内声明的为局部变量/常量。
- **语句**：语句包括赋值语句、表达式语句(表达式可以为空)、语句块、if 语句、while 语句、break 语句、continue 语句、return 语句。语句块中可以包含若干变量声明和语句。
- **表达式**：支持基本的算术运算（+、-、、/、%）、关系运算（== 、!=、<、>、<=、>=）和逻辑运算（!、&&、||），非 0 表示真、0 表示假，而关系运算或逻辑运算的结果用 1 表示真、0 表示假。**算符的优先级和结合性以及计算规则(含逻辑运算的"短路计算")与 C 语言一致。**

一、输入输出函数

```
<InStmt> → <LVal> = 'getint''('')'

<OutStmt> → 'printf''('<FormatString>{,<Exp>}')'

<FormatChar> → %d

<NormalChar> → 十进制编码为32,33,40-126的ASCII字符，'\'（编码92）出现当且仅当为'\n'

<Char> → <FormatChar> | <NormalChar>

<FormatString> → '"'{<Char>}'"'
```

格式字符串中**仅可能会出现一种转义字符**'\n'，用以标注此处换行，其他转义情况无需考虑。其余情况，" 单独出现是不合法的。

在我们的语言中，输入函数不同于C语言的scanf，如果希望在C语言中测试测试程序，只需要将getint加入函数声明即可，示例如下：

```
int getint(){
    int n;
    scanf("%d",&n);
    return n;
}
```

二、文法及覆盖要求

1. 覆盖要求

测试程序需覆盖文法中所有的语法规则（即每一条推导规则的每一个候选项都要被覆盖），并满足文法的语义约束（换而言之，测试程序应该是正确的）。在下一节中，文法正文中以注释形式给出需要覆盖的情况。

2. 文法

语言的文法采用扩展的 Backus 范式（EBNF，Extended Backus-Naur Form）表示，其中：

- 符号[...]表示方括号内包含的为可选项
- 符号{...}表示花括号内包含的为可重复 0 次或多次的项
- 终结符或者是由单引号括起的串，或者是 Ident、InstConst 这样的记号

语言的文法表示如下，其中 CompUnit 为开始符号：

```
编译单元 CompUnit → {Decl} {FuncDef} MainFuncDef // 1.是否存在Decl 2.是否存在FuncDef

声明 Decl → ConstDecl | VarDecl // 覆盖两种声明

常量声明 ConstDecl → 'const' BType ConstDef { ',' ConstDef } ';' // 1.花括号内重复0次 2.花括号内重复多次

基本类型 BType → 'int' // 存在即可

常数定义 ConstDef → Ident [ '[' ConstExp ']' ] '=' ConstInitVal // 包含普通变
```

量、一维数组两种情况

常量初值 ConstInitVal → ConstExp

| '{' [ConstInitVal { ',' ConstInitVal }] '}' // 1.常表达式初值 2.一维数组初值

变量声明 VarDecl → BType VarDef { ',' VarDef } ';' // 1.花括号内重复0次 2.花括号内重复多次

变量定义 VarDef → Ident ['[' ConstExp ']'] // 包含普通变量、一维数组定义

| Ident ['[' ConstExp ']'] '=' InitVal

变量初值 InitVal → Exp | '{' [InitVal { ',' InitVal }] '}'// 1.表达式初值 2.一维数组初值

函数定义 FuncDef → FuncType Ident '(' [FuncFParams] ')' Block // 1.无形参 2.有形参

主函数定义 MainFuncDef → 'int' 'main' '(' ')' Block // 存在main函数

函数类型 FuncType → 'void' | 'int' // 覆盖两种类型的函数

函数形参表 FuncFParams → FuncFParam { ',' FuncFParam } // 1.花括号内重复0次 2.花括号内重复多次

函数形参 FuncFParam → BType Ident // 只包含普通变量

语句块 Block → '{' { BlockItem } '}' // 1.花括号内重复0次 2.花括号内重复多次

语句块项 BlockItem → Decl | Stmt // 覆盖两种语句块项

语句 Stmt → LVal '=' Exp ';' // 每种类型的语句都要覆盖

| [Exp] ';' //有无Exp两种情况

| Block

| 'if' '(Cond ')' Stmt ['else' Stmt] // 1.有else 2.无else

| 'while' '(' Cond ')' Stmt

| 'break' ';' | 'continue' ';'

| 'return' [Exp] ';' // 1.有Exp 2.无Exp

| LVal = 'getint''('')'';'

| 'printf' '('FormatString {',' Exp} ')'';' // 1.有Exp 2.无Exp

表达式 Exp → AddExp 注：SysY 表达式是int 型表达式 // 存在即可

条件表达式 Cond → LOrExp // 存在即可

左值表达式 LVal → Ident ['[' Exp ']'] //1.普通变量 2.一维数组

基本表达式 PrimaryExp → '(' Exp ')' | LVal | Number // 三种情况均需覆盖

数值 Number → IntConst // 存在即可

一元表达式 UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' // 三种情况均需覆盖,函数调用也需要覆盖FuncRParams的不同情况

| UnaryOp UnaryExp // 存在即可

单目运算符 UnaryOp → '+' | '−' | '!' 注：'!'仅出现在条件表达式中 // 三种均需覆盖

函数实参表 FuncRParams → Exp { ',' Exp } // 1.花括号内重复0次 2.花括号内重复多次

乘除模表达式 MulExp → UnaryExp | MulExp ('*' | '/' | '%') UnaryExp // 1.UnaryExp 2.* 3./ 4.% 均需覆盖

加减表达式 AddExp → MulExp | AddExp ('+' | '−') MulExp // 1.MulExp 2.+ 3.− 均需覆盖

关系表达式 RelExp → AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp // 1.AddExp 2.< 3.> 4.<= 5.>= 均需覆盖

相等性表达式 EqExp → RelExp | EqExp ('==' | '!=') RelExp // 1.RelExp 2.== 3.!= 均需覆盖

```
逻辑与表达式 LAndExp → EqExp | LAndExp '&&' EqExp // 1.EqExp 2.&& 均需覆盖

逻辑或表达式 LOrExp → LAndExp | LOrExp '||' LAndExp // 1.LAndExp 2.|| 均需覆盖

常量表达式 ConstExp → AddExp 注：使用的Ident 必须是常量 // 存在即可

格式化字符 FormatChar → %d

普通字符 NormalChar → 十进制编码为32,33,40-126的ASCII字符

字符 Char → FormatChar | NormalChar

格式化字符串 FormatString → '"'{ Char }'"'
```

3. 终结符特征

(1) 标识符 Ident

关于标准定义，请参考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf 中第 51 页关于标识符的定义

同名标识符的约定：

- 全局变量（常量）和局部变量（常量）的作用域可以重叠，重叠部分局部变量（常量）优先
- 同名局部变量的作用域不能重叠，即同一个 Block 内不能有同名的标识符（常量/变量）
- 变量（常量）名可以与函数名相同

(2) 注释

注释的规范与 C 语言一致，包括单行和多行注释，需要同学们在编译器中进行处理。

注：请参考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf 66 页关于注释的定义

(3) 数值常量

语言中数值常量可以是整型数 IntConst，其规范如下（对应 integer- const）：

整型常量 integer-const → decimal-const | 0

decimal-const → nonzero-digit | decimal-const digit

nonzero-digit 为 1 2 3 4 5 6 7 8 9之一

注：请参考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf 54 页关于整型常量的定义，在此基础上忽略所有后缀。

4. 难度分级

为了更好的帮助同学们完成编译器的设计，课程组今年将题目难度做了区分。难度等级分为三级：A、B、C，难度依次递减：

- C：最简单的等级，重点考察编译器的基础设计。
- B：在 C 级的基础上，新增数组，包括数组定义，数组元素的使用等。
- A：在 B 级的基础上，新增复杂条件的运算和判断。