文法解读

```c
#include <stdio.h>
// 函数
const int a;
const int b = 1, c[3] = {1+2, 2*(-3)};
int d = 10-2;

void fv(){}
void f1(int num1){
    return ;
}
int f2(int num1, int num2[]){
    printf("FUNC 传参 TEST: num1 = %d = 100?, num2[0] = %d = 3?\n", num1, num2[0]);
    return num1-num2[0];
}
int main(){
    // 声明
    int g = -1;
    int e = d+b, f[3] = {6/2, 7%3};
    // 语句
    {
        d = 100;
        f1(1);
        f[0] = 100;
        printf("FUNC2 TEST: f2(d, f) = %d = 0?\n", f2(d, f));
    }
    ;

    for(int i = 0; i < 3; i++){ // 测试FOR111
        printf("FOR111 & BREAK & ! TEST: i = %d = 0?\n", i);
        if(!(0 && g++)){ // 测试 ! && &&短路求值 break            break;
        }
    }
    printf("DUANLUQIUZHI&& TEST: g = %d = -1?\n", g);

    for(int i = 0; i < 3;){ // 测试FOR110
        i++;
        if(i < 2 || ++g){ // 测试 || ||短路求值 continue            continue;
        }else{
            printf("FOR110 & CONTINUE TEST: i = %d = 2?\n", i);
            printf("DUANLUQIUZHI|| TEST: g = %d = 0?\n", g);
        }
        printf("CONTINUE TEST: print only 1 time\n");
    }

    for(int i = 0; ;i++){ // 测试FOR101 ==
        if(i == 0){
            printf("FOR101 & == TEST: i = %d = 0?\n", i);
```

```
        }else{
            printf("== TEST: i = %d = 1?\n", i);
            break;
        }
    }

    for(int i = 0; ; ){ // 测试FOR100 !=
        if(i != 1){
            printf("FOR100 & != TEST: i = %d = 0?\n", i);
        }else{
            printf("!= TEST: i = %d = 1?\n", i);
            break;
        }
        i++;
    }

    int i = 0;
    for( ; i < 3; i++){ // 测试FOR011 <
        if(i < 1){
            printf("FOR011 & < TEST: i = %d = 0?\n", i);
        }else{
            break;
        }
    }
    for( ; i < 4;){ // 测试FOR010 <=
        if(i <= 1){
            printf("FOR010 & <= TEST: i = %d = 1?\n", i);
        }else{
            break;
        }
        i++;
    }
    for( ; ; i--){ // 测试FOR001 >
        if(i > 1){
            printf("FOR001 & > TEST: i = %d = 2?\n", i);
        }else{
            break;
        }
    }
    for( ; ; ){ // 测试FOR000 >=
        if(i >= 1){
            printf("FOR000 & >= TEST: i = %d = 1?\n", i);
        }else{
            break;
        }
        i--;
    }
    printf(">= TEST: i = %d = 0?\n", i);

    return 0;
}
```

编译单元 `compUnit → {Decl} {FuncDef} MainFuncDef`

- [ ] 存在Decl
- [ ] 不存在Decl
- [ ] 存在FuncDef
- [ ] 不存在FuncDef

1 常量 & 变量

```
const int a;
const int b = 1, c[3] = {1+2, 2*(-3)};
int d = 10-2;
int e = d+b, f[3] = {6/2, 7%3};
```

声明 `Decl → ConstDecl | VarDecl`

- [ ] 存在常量
- [ ] 存在变量

1.1 常量

常量声明 `constDecl → 'const' BType constDef { ',' constDef } ';'`

- [ ] 花括号内重复0次 const int a;
- [ ] 花括号内重复多次 const int b, c[3];

基本类型 `BType → 'int'`

常量定义 `constDef → Ident [ '[' constExp ']' ] '=' constInitval`

- [ ] 存在普通常量 b
- [ ] 存在一维数组 c[3]

常量初值 `constInitval → constExp | '{' [ constExp {',' constExp } ] '}'`

- [ ] 常表达式初值 1
- [ ] 一维数组初值 {1, 2}

1.2 变量

变量声明 `varDec1 → [ 'static' ] BType varDef { ',' varDef } ';'`

- [ ] 一个普通变量
- [ ] 一堆普通变量
- [ ] 一个静态变量
- [ ] 一堆静态变量

变量定义 `varDef → Ident [ '[' constExp ']' ] | Ident [ '[' constExp ']' ] '=' Initval`

- [ ] 普通变量 e 有初始值

- 普通变量 e 无初始值
- 一维数组 f[3] 有初始值
- 一维数组 f[3] 无有初始值

变量初值 `Initval → Exp | '{' [ Exp { ',' Exp } ] '}'`

- 表达式初值 int e=d + b;
- 一维数组初值 int f[3] = {1, 2}

2 函数

```
// 函数
void fv(){}
void f1(int num1){
    return ;
}
int f2(int num1, int num2[]){
    return num1-num2[0];
}
int main(){
    // 声明
    int g = -1;

    // 语句
    {
        d = 100;
        f1(1);
        f[0] = 100;
        printf("FUNC TEST: f2(d, f) = %d = 0?\n", f2(d, f));
    }
    ;

    for(int i = 0; i < 3; i++){ // 测试FOR111
        printf("FOR111 & BREAK & ! TEST: i = %d = 0?\n", i);
        if(!(0 && g++)){ // 测试 ! && &&短路求值 break
            break;
        }
    }
    printf("DUANLUQIUZHI&& TEST: g = %d = -1?");

    for(int i = 0; i < 3;){ // 测试FOR110
        i++;
        if(i < 2 || ++g){ // 测试 || ||短路求值 continue
            continue;
        }else{
            printf("FOR110 & CONTINUE TEST: i = %d = 2?\n", i);
            printf("DUANLUQIUZHI|| TEST: g = %d = 0?\n", g)
        }
        printf("CONTINUE TEST: print only 1 time\n")
    }
```

```
    for(int i = 0; ;i++){ // 测试FOR101 ==
        if(i == 0){
            printf("FOR101 & == TEST: i = %d = 0?\n", i);
        }else{
            printf("== TEST: i = %d = 1?\n", i);
            break;
        }
    }

    for(int i = 0; ; ){ // 测试FOR100 !=
        if(i != 1){
            printf("FOR100 & != TEST: i = %d = 0?\n", i);
        }else{
            printf("!= TEST: i = %d = 1?\n", i);
            break;
        }
        i++;
    }

    int i = 0;
    for( ; i < 3; i++){ // 测试FOR011 <
        if(i < 1){
            printf("FOR011 & < TEST: i = %d = 0?\n", i);
        }else{
            break;
        }
    }
    for( ; i < 4;){ // 测试FOR010 <=
        if(i <= 1){
            printf("FOR010 & <= TEST: i = %d = 1?\n", i);
        }else{
            break;
        }
        i++;
    }
    for( ; ; i--){ // 测试FOR001 >
        if(i > 1){
            printf("FOR001 & > TEST: i = %d = 2?\n", i);
        }else{
            break;
        }
    }
    for( ; ; ){ // 测试FOR000 >=
        if(i >= 1){
            printf("FOR000 & >= TEST: i = %d = 1?\n", i);
        }else{
            break;
        }
        i--;
    }
    printf(">= TEST: i = %d = 0?\n", i);
```

```
    return 0;
}
```

2.1 函数

函数定义 `FuncDef → FuncType Ident '(' [FuncFParams] ')' Block`

- 无形参 void fv(){}
- 有1形参 void f1(int num1){ return ; }
- 有多形参 int f2(int num1, int num2[]){ return num1-num2[0]; }

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

- 存在main函数 int main(){定义; 赋值;}

函数类型 `FuncType →'void' | 'int'`

- void
- int

函数形参表 `FunCFParams → FunCFParam { ',' FunCFParam }`

- 花括号内重复0次
- 花括号内重复多次

函数形参 `FuncFParam → BType Ident [ '[' ']' ]`

- 普通变量
- 一维数组变量

2.2 语句

语句块 `Block → '{' { BlockItem } '}'`

- 花括号内重复0次 fv函数
- 花括号内重复多次 main函数

语句块项 `BlockItem → Decl | Stmt`

- Decl 声明
- Stmt 语句

语句 `Stmt → LVal '=' Exp ';'`

- 赋值语句 `d = 100`
  `| [Exp] ';'`
- 有Exp 函数调用 `f2();`
- 无Exp `;`
  `| Block`

- [ ] 语句块 (Block)
  `| 'if' '(' Cond ')' Stmt [ 'else' Stmt ]`
- [ ] if-else
- [ ] if
  `| 'for' '(' [ForStmt] ';' [Cond] ';' [ForStmt] ')' Stmt`
  8种缺省组合（其中一种需要ForStmt有多个定义）
- [ ] 000
- [ ] 001
- [ ] 010
- [ ] 011
- [ ] 100
- [ ] 101
- [ ] 110
- [ ] 111
  `| 'break' ';'`
- [ ] break语句
  `| 'continue' ';'`
- [ ] continue语句
  `| 'return' [Exp] ';'`
- [ ] return语句 (有返回值)
- [ ] return语句 (无返回值)
  `| 'printf' '(' StringConst { ',' Exp } ')' ';'`
- [ ] printf函数调用语句 (有Exp) `printf("%d", b})`
- [ ] printf函数调用语句 (无Exp) `printf("23373125")`

语句 `ForStmt → LVal '=' Exp { ',' LVal '=' Exp }`

- [ ] 花括号内重复0次
- [ ] 花括号内重复多次

2.3 表达式

表达式 `Exp → AddExp`

- [ ] 存在

条件表达式 `Cond → LOrExp`

- [ ] 存在（最终可规约为一个逻辑或表达式(LOrExp)。它用于`if`和`for`等需要条件判断的语句中）

左值表达式 `LVal → Ident ['[' Exp ']']`
左值代表一个可被赋值或读取的存储位置，可以是简单变量，也可以是数组元素（通过下标访问）。

- [ ] 普通变量、常量 `e`
- [ ] 一维数组元素 `f[3]`

基本表达式 `PrimaryExp → '(' Exp ')' | LVal | Number`

- 括号表达式
- 左值
- 数值

数值 `Number → IntConst`

- 存在

一元表达式 `UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' | UnaryOp UnaryExp`

- 基本表达式 (PrimaryExp)
- 函数调用 (需覆盖FuncRParams的不同情况)
- 带单目运算符的表达式

单目运算符 `UnaryOp → '+' | '−' | '!'`

- '+'
- '-'
- '!' (注：仅出现在条件表达式中)

函数实参表 `FuncRParams → Exp { ',' Exp }`

- 花括号内重复0次 (1个实参)
- 花括号内重复多次 (多个实参)
- Exp需要覆盖普通变量传参和数组传参

乘除模表达式 `MulExp → UnaryExp | MulExp ('*' | '/' | '%') UnaryExp`

- UnaryExp
- '*' (乘法)
- '/' (除法)
- '%' (取模)

加减表达式 `AddExp → MulExp | AddExp ('+' | '−') MulExp`

- MulExp (基础)
- '+' (加法)
- '-' (减法)

关系表达式 `RelExp → AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp`

- AddExp (基础)
- '<' (小于)
- '>' (大于)
- '<=' (小于等于)

- [ ] '>=' (大于等于)

相等性表达式 `EqExp → RelExp | EqExp ('==' | '!=') RelExp`

- [ ] RelExp (基础)
- [ ] '==' (等于)
- [ ] '!=' (不等于)

逻辑与表达式 `LAndExp → EqExp | LAndExp '&&' EqExp`

- [ ] EqExp (基础)
- [ ] '&&' (逻辑与，需短路求值)

逻辑或表达式 `LOrExp → LAndExp | LOrExp '||' LAndExp`

- [ ] LAndExp (基础)
- [ ] '||' (逻辑或，需短路求值)

常量表达式 `ConstExp → AddExp`
解释：常量表达式是一个能在编译时计算出确定值的加法表达式(AddExp)，其中所有标识符都必须是常量。

- [ ] 存在

3 我犯过的错

1. for()里面不能定义变量：如 `for(int i=0; ; )`
2. 不可以 i++
3. 不可以 a && (b)：如 `if(0 && (g+=1))`