2021编译技术实验文法定义及相关说明

零、概要

SysY 语言是本次课程要实现的编程语言，是 C 语言的一个子集。每个 SysY 程序的源码存储在一个扩展名为 sy 的文件中。该文件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。SysY 语言支持 **int 类型**和元素为 int 类型且按行优先存储的**多维数组类型**，其中 int 型整数为 32 位有符号数；**const** 修饰符用于声明常量。

SysY 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，参数的类型可以是 int 或者数组类型；函数可以返回 int 类型的值，或者不返回值(即声明为 void 类型)。当参数为int 时，按值传递；而参数为数组类型时，实际传递的是数组的起始地址，并且形参只有第一维的长度可以空缺。函数体由若干变量声明和语句组成。
- **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、输入输出函数定义

```
<InStmt> → <LVal> = 'getint''('')'
<OutStmt> → 'printf''('<FormatString>{,<Exp>}')'
```

其中，**FormatString**为格式字符串终结符，其规范如下：为了与 gcc 评测结果一致，(<NormalChar> 的转义字符有且仅有 '\n'，其余情况，'\' 单独出现是不合法的)

```
<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。

对应的，语句<Stmt>的右侧需要添加上述输入输出语句，即

```
<Stmt> → <Stmt>
    | <LVal> = 'getint''('')'';'
    |'printf''('<FormatString>{,<Exp>}')'';'
```

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（注意函数形参表中 FuncFParam 数组声明需要为 ConstExp ）。

2. 文法

SysY 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

SysY 语言的文法表示如下，其中 CompUnit 为开始符号：

重要：建议同时对照文法第三部分的语义约束，使用支持侧栏书签的PDF阅读器进行跳转。

```
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、二维数组共三种情况
常量初值  ConstInitVal → ConstExp
  | '{' [ ConstInitVal { ',' ConstInitVal } ] '}' // 1.常表达式初值 2.一维数组初值 3.二维数组初值
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、二维数组定义
  | Ident { '[' ConstExp ']' } '=' InitVal
变量初值  InitVal → Exp | '{' [ InitVal { ',' InitVal } ] '}'// 1.表达式初值 2.一维数组初值 3.二维数组初值
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 ['[' ']' { '[' ConstExp ']' }] // 1.普通变量 2.一维数组变量 3.二维数组变量
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 3.二维数组
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] ')' // 3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 3.Exp需要覆盖数组传参和部分数组传参
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.+ 需覆盖 3.-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
```

标识符Ident（需要覆盖的情况以注释形式给出）：

```
identifier  → identifier-nondigit
    | identifier identifier-nondigit
    | identifier digit
```

数值常量（需要覆盖的情况以注释形式给出）：

```
integer-const → decimal-const
decimal-const → nonzero-digit | decimal-const digit
```

3. 终结符特征

(1) 标识符 Ident

SysY 语言中标识符 **Ident** 的规范如下 (identifier)：

标识符　identifier　→ identifier-nondigit　// 存在标识符即可

| identifier identifier-nondigit

| identifier digit

其中 identifier-nondigit 为**下划线**或以下之一

a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

digit 为以下之一

0 1 2 3 4 5 6 7 8 9

注：请参考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

第 51 页关于标识符的定义**同名标识符**的约定：

- 全局变量和局部变量的作用域可以重叠，重叠部分局部变量优先；同名局部变量的作用域不能重叠；
- SysY 语言中变量名可以与函数名相同。

(2) 注释

SysY 语言中注释的规范与 C 语言一致，如下：

- 单行注释：以序列 '//' 开始，直到换行符结束，不包括换行符。
- 多行注释：以序列 '/*' 开始，直到第一次出现 '*/' 时结束，包括结束处'*/'。

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

(3) 数值常量

SysY 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

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

nonzero-digit 为以下之一

1 2 3 4 5 6 7 8 9

注：请参考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf第 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。下文将对难度等级进行详细介绍：

- 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、数组传参和部分传参等。

  部分传参样例：

```
void func(int a[]) {
    //...
}

int main() {
    int arr[2][2] = {{1,2},{3,4}};
    func(arr[1]);
    return 0;
}
```

常数定义 ConstDef → Ident { '[' ConstExp ']' } '=' ConstInitVal // 包含普通变量、一维数组、二维数组共三种情况
常量初值 ConstInitVal → ConstExp
| '{' [ConstInitVal { ',' ConstInitVal }] '}' // 1.常表达式初值 2.一维数组初值 3.二维数组初值
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、二维数组定义
| Ident { '[' ConstExp ']' } '=' InitVal
变量初值 InitVal → Exp | '{' [InitVal { ',' InitVal }] '}'// 1.表达式初值 2.一维数组初值 3.二维数组初值
函数形参 FuncFParam → BType Ident ['[' ']' { '[' ConstExp ']' }] // 1.普通变量 2.一维数组变量 3.二维数组变量
左值表达式 LVal → Ident {'[' Exp ']'} //1.普通变量 2.一维数组 3.二维数组
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 3. Exp需要覆盖数组传参和部分数组传参

- A：在 B 级的基础上，**新增**复杂条件的运算和判断，要遵守短路求值的规则。

  短路求值样例：

```
int global_var = 0;

int func() {
    global_var = global_var + 1;
    return 1;
}

int main() {
    if (0 && func()){
        ;
    }
    printf("%d",global_var); // 输出 0
    if (1 || func()) {
```

```
        ;
    }
    printf("%d",global_val); // 输出 0
    return 0;
}
```

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

注：低级难度的样例不包含高级难度样例新增的规则。

三、语义约束

符合上述文法的程序集合是合法的 SysY 语言程序集合的超集。下面进一步给出 SysY 语言的语义约束。

简化文法约束

- 文法中多维数组的维度至多为 2 维。
- 多维数组的初始化中，子维度的初始化一定会加入‘{’，‘}’，并且花括号内元素数量与维度一定完全一致，例如二维数组初始化为：a[2][2]={{1,2},{3,4}}。

IO语句

- 输入语句以函数调用的形式出现，对应函数声明如下。虽然输入语句以函数调用形式出现，**getint 仍被识别为关键字而不是标识符**，在词法分析阶段和语法分析阶段需要注意。

```
int getint()
```

- 与 C 语言中的 printf 类似，输出语句中，格式字符将被替换为对应标识符，普通字符原样输出。
- printf 默认不换行。

编译单元 CompUnit

编译单元 CompUnit → {Decl} {FuncDef} MainFuncDef

声明　Decl → ConstDecl | VarDecl

注意：

1. CompUnit 的**顶层**变量/常量声明语句（对应 Decl）、函数定义（对应 FuncDef）都不可以重复定义同名标识符（Ident），即便标识符的类型不同也不允许。
2. CompUnit 的变量/常量/函数声明的作用域从该声明处开始到文件结尾。

常数定义 ConstDef

常数定义　ConstDef → **Ident** { '[' ConstExp ']' } '=' ConstInitVal

1. ConstDef 用于定义符号常量。ConstDef 中的 Ident 为常量的标识符，在 Ident后、‘=’ 之前是可选的数组维度和各维长度的定义部分，在 ‘=’ 之后是初始值。

2. ConstDef 的数组维度和各维长度的定义部分不存在时，表示定义单个变量。此时 '=' 右边必须是单个初始数值。
3. ConstDef 的数组维度和各维长度的定义部分存在时，表示定义数组。其语义和 C 语言一致，比如[2][8/2]表示二维数组，第一到第二维长度分别为2、4，每维的下界从 0 编号。 ConstDef 中表示各维长度的 ConstExp 都必须能在编译时求值到**非负整数**。

   注意：SysY 在声明数组时各维长度都需要显式给出，而不允许是未知的。

   ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf 第 125 页6.7.8 节的第 6 点规定如下：

6 If a designator has the form

[*constant-expression*]

then the current object (defined below) shall have array type and the expression shall be an integer constant expression. If the array is of unknown size, any nonnegative value is valid.

4. 当 ConstDef 定义的是数组时，'=' 右边的 ConstInitVal 表示常量初始化器。ConstInitVal 中的 ConstExp 是能在编译时求值的 int 型表达式，其中可以引用已定义的符号常量。
5. ConstInitVal 初始化器必须是以下情况（注：int 型初始值可以是 Number，或者是 int 型常量表达式）：
   - 与多维数组中数组维数和各维长度完全对应的初始值，如{{1,2},{3,4},{5,6}}为 a[3][2]的初始值。

变量定义 VarDef

变量定义　VarDef → **Ident** { '[' ConstExp ']' }

| **Ident** { '[' ConstExp ']' } '**=**' InitVal

1. VarDef 用于定义变量。当不含有'='和初始值时，其运行时实际初值未定义。
2. VarDef 的数组维度和各维长度的定义部分不存在时，表示定义单个变量。存在时，和 ConstDef 类似，表示定义多维数组。（参见 ConstDef 的第 2 点）
3. 当 VarDef 含有 '=' 和初始值时， '=' 右边的 InitVal 和 CostInitVal 的结构要求相同，唯一的不同是 ConstInitVal 中的表达式是 ConstExp 常量表达式，而InitVal 中的表达式可以是当前上下文合法的任何 Exp。
4. VarDef 中表示各维长度的 ConstExp 必须是能求值到**非负整数**，但 InitVal 中的初始值为 Exp，其中可以引用变量，例如下面b的初始化：

```
int a[2][2]={{1,2},{3,4}};
int b[2][2]={{a[0][0],a[0][1]},{a[1][0],a[1][1]}};
```

初值

常量初值 ConstInitVal → ConstExp

| '**{**' [ConstInitVal { '**,**' ConstInitVal }] '**}**'

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

1. 全局变量声明中指定的初值表达式必须是常量表达式。
2. 常量或变量声明中指定的初值要与该常量或变量的类型一致如下形式的 VarDef / ConstDef 不满足 SysY 语义约束：

```
a[4] = 4
a[2] = {{1,2}, 3}
a = {1,2,3}
```

3. 未显式初始化的局部变量，其值是不确定的；而未显式初始化的全局变量， 其（元素）值均被初始化为 0。

函数形参 FuncFParam 与实参

函数形参 FuncFParam → BType **Ident** ['[' ']' { '[' ConstExp ']' }]

1. FuncFParam 定义一个函数的一个形式参数。当 Ident 后面的可选部分存在时， 表示数组定义。
2. 当 FuncFParam 为数组定义时，其第一维的长度省去（用方括号[ ]表示），而后面的各维则需要用表达式指明长度，长度是常量。
3. 函数实参的语法是 Exp。对于 int 类型的参数，遵循按值传递；对于数组类型的参数，则形参接收的是实参数组的地址，并通过地址间接访问实参数组中的元素。
4. 对于多维数组，可以传递其中的一部分到形参数组中。例如，若 int a[4][3], 则 a[1] 是包含三个元素的一维数组，a[1] 可以作为参数传递给类型为 int[] 的形参。
5. 常量数组如 `const int arr[3] = {1,2,3}` ，常量数组 arr **不能**作为参数传入到函数中

函数定义 FuncDef

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

1. FuncDef 表示函数定义。其中的 FuncType 指明返回类型。
    - 当返回类型为 int 时，函数内所有分支都应当含有带有 Exp 的 return 语句。不含有 return 语句的分支的返回值未定义。
    - 当返回值类型为 void 时，函数内只能出现不带返回值的 return 语句。
2. FuncDef 中形参列表（FuncFParams）的每个形参声明（FuncFParam）用于声明 int 类型的参数，或者是元素类型为 int 的多维数组。FuncFParam 的语义参见前文。

语句块 Block

1. Block 表示语句块。语句块会创建作用域，语句块内声明的变量的生存期在该语句块内。
2. 语句块内可以再次定义与语句块外同名的变量或常量（通过 Decl 语句），其作用域从定义处开始到该语句块尾结束，它隐藏语句块外的同名变量或常量。

语句 Stmt

1. Stmt 中的 if 类型语句遵循就近匹配。
2. 单个 Exp 可以作为 Stmt。Exp 会被求值，所求的值会被丢弃。

左值 LVal

1. LVal 表示具有左值的表达式，可以为变量或者某个数组元素。

2. 当 LVal 表示数组时，方括号个数必须和数组变量的维数相同（即定位到元素）。
3. 当 LVal 表示单个变量时，不能出现后面的方括号。

Exp 与 Cond

1. Exp 在 SysY 中代表 int 型表达式，故它定义为 AddExp；Cond 代表条件表达式，故它定义为 LOrExp。前者的单目运算符中不出现'!'，后者可以出现。
2. LVal 必须是当前作用域内、该 Exp 语句之前有定义的变量或常量；对于赋值号左边的 LVal 必须是变量。
3. 函数调用形式是 Ident '(' FuncRParams ')'，其中的 FuncRParams 表示实际参数。实际参数的类型和个数必须与 Ident 对应的函数定义的形参完全匹配。
4. SysY 中算符的优先级与结合性与 C 语言一致，在上面的 SysY 文法中已体现出优先级与结合性的定义。

一元表达式 UnaryExp

1. 相邻两个 `UnaryOp` 不能相同，如 `int a = ++-i;` ，但是 `int a = +-+i;` 是可行的。
2. `UnaryOp` 为 '!' 只能出现在条件表达式中。

常量表达式 ConstExp

1. `ConstExp -> AddExp` 中使用的 ident 必须是常量

四、错误定义

错误处理的输出的报错行号定义为："**出错符号或出错符号的前一个终结符所在的行号**"。为避免歧义，可能存在歧义的报错会在解释中注明报错行号。

错误类别码定义如下：

错误类型	错误类别码	解释	对应文法及出错符号(...省略该条规则后续部分)
非法符号	a	格式字符串中出现非法字符 报错行号为**<FormatString>**所在行数。	<FormatString> → '"{<Char>}"'
名字重定义	b	函数名或者变量名在**当前作用域**下重复定义。 注意，变量一定是同一级作用域下才会判定出错，不同级作用域下，内层会覆盖外层定义。 报错行号为**<Ident>**所在行数。	<ConstDef>→<Ident> ... <VarDef>→<Ident> ... |<Ident> ... <FuncDef>→<FuncType><Ident> ...
未定义的名字	c	使用了未定义的标识符 报错行号为**<Ident>**所在行数。	<LVal>→<Ident> ... <UnaryExp>→<Ident> ...
函数参数个数不匹配	d	函数调用语句中，参数个数与函数定义中的参数个数不匹配。 报错行号为函数名调用语句的**函数名**所在行数。	<UnaryExp>→<Ident>'('[FuncRParams]')'
函数参数类型不匹配	e	函数调用语句中，参数类型与函数定义中对应位置的参数类型不匹配。 报错行号为函数名调用语句的**函数名**所在行数。	<UnaryExp>→<Ident>'('[FuncRParams]')'
无返回值的函数存在不匹配的return语句	f	报错行号为**'return'**所在行号。	<Stmt>→'return' {['Exp']}';'
有返回值的函数缺少return语句	g	只需要考虑函数末尾是否存在return语句即可，**无需考虑数据流**。 报错行号为函数**结尾的'}'**所在行号。	FuncDef → FuncType Ident '(' [FuncFParams] ')' Block
不能改变常量的值	h	<LVal>为常量时，不能对其修改。 报错行号为**<LVal>**所在行号。	<Stmt>→<LVal>'=' <Exp>';'|<LVal>'=' 'getint' '(' ')' ';'
缺少分号	i	报错行号为分号**前一个非终结符**所在行号。	<Stmt>,<ConstDecl>及<VarDecl>中的';'
缺少右小括号')'	j	报错行号为右小括号**前一个非终结符**所在行号。	函数调用(<UnaryExp>)、函数定义(<FuncDef>)及<Stmt>中的')'
缺少右中括号']'	k	报错行号为右中括号**前一个非终结符**所在行号。	数组定义(<ConstDef>,<VarDef>,<FuncFParam>)和使用(<LVal>)中的']'
printf中格式字符与表达式个数不匹配	l	报错行号为**'printf'**所在行号。	Stmt →'printf''('FormatString{,Exp}')'';'
在非循环块中使用break和continue语句	m	报错行号为**'break'与'continue'**所在行号。	<Stmt>→'break'';' |'continue'';'

五、词法分析输出要求

词法分析类别码定义如下：

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
Ident	IDENFR	!	NOT	*	MULT	=	ASSIGN
IntConst	INTCON	&&	AND	/	DIV	;	SEMICN
FormatString	STRCON	||	OR	%	MOD	,	COMMA
main	MAINTK	while	WHILETK	<	LSS	(	LPARENT
const	CONSTTK	getint	GETINTTK	<=	LEQ	)	RPARENT
int	INTTK	printf	PRINTFTK	>	GRE	[	LBRACK
break	BREAKTK	return	RETURNTK	>=	GEQ	]	RBRACK
continue	CONTINUETK	+	PLUS	==	EQL	{	LBRACE
if	IFTK	-	MINU	!=	NEQ	}	RBRACE
else	ELSETK						

输出形式为：

> 单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔)

具体样例如下：

```
int main(){
    int c;
    c = getint();
    printf("%d",c);
    return c;
}
```

对应的输出为：

```
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
INTTK int
IDENFR c
SEMICN ;
IDENFR c
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMICN ;
PRINTFTK printf
LPARENT (
STRCON "%d"
COMMA ,
IDENFR c
```

```
RPARENT )
SEMICN ;
RETURNRK return
IDENFR c
SEMICN ;
RBRACE }
```

六、语法分析输出要求

语法分析输出规则中，输出的符号为出现在SysY语言正文中**除去以下非终结符外**的所有非终结符。

- <BlockItem>
- <Decl>
- <BType>

输出输出样例如下。

测试程序：

```
int main(){
    int c;
    c= getint();
    printf("%d",c);
    return c;
}
```

语法输出结果：

```
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
INTTK int
IDENFR c
<VarDef>
SEMICN ;
<VarDecl>
IDENFR c
<LVal>
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMICN ;
<Stmt>
PRINTFTK printf
```

```
LPARENT (
STRCON "%d"
COMMA ,
IDENFR c
<LVal>
<PrimaryExp>
<UnaryExp>
<MulExp>
<AddExp>
<Exp>
RPARENT )
SEMICN ;
<Stmt>
RETURNTK return
IDENFR c
<LVal>
<PrimaryExp>
<UnaryExp>
<MulExp>
<AddExp>
<Exp>
SEMICN ;
<Stmt>
RBRACE }
<Block>
<MainFuncDef>
<CompUnit>
```