CACT语言规范（2025版）

实验目标：通过一系列任务，完成CACT语言编译器的开发。

关于CACT语言：为C语言的迷你子集，设计受SysY2020启发。

注意：CACT不支持许多常见C特性，合法代码均以本规范为准。

词法规范

关键字

CACT对大小写敏感。关键字均小写，例如**if**是关键字，IF是标识符。

关键字表：

const　int　double　char

float　void　if　else

while　break　continue　return

基本类型

三种基本类型：int，float，char。

- 整数：int为32位有符号整数，支持以下整数常量格式：
  - 十进制：除了“0”之外，十进制整数的首位数字不为0。如：0、234
  - 八进制：以整数0开头，后由0-7的数字组成，如：000、0127
  - 十六进制：以0x或0X开头，后由0-9、a-f、A-F中的数字组成。如：0XFa32

    数字与数字间没有如下划线' _'等分隔符，数字后也没有后缀。
- 浮点数：遵循IEEE754标准，有符号，支持两种形式。
  - 普通形式：必须包含小数点，小数点的前或后必须有数字出现。例如9.00、12.43、.5、8.f

    注意：为简单起见，整数加f后缀也不是浮点数，如3f算错误。
  - 指数形式：必须依次出现基数、指数符号和指数三个部分。基数可带小数点，也可不带；指数符号为“E”或“e”；指数部分可带符号“+”或“-”（也可不带）后接一串数字（0-9）组成。例如-43.0e-4F（表示32位浮点数-43.0×10^{-4}）、3E2f（为32位浮点数3×10^{2}）。
- 字符：char为有符号8位整数，字符常量由单引号包围
  - 可以是单个ASCII字符，如'a'、'Z'、'0'、'+'
  - 支持转义序列：'\n'（换行）、'\t'（制表符）、''（反斜杠）、'''（单引号）、'"'（双引号）、'\0'（空字符）
  - 每个字符常量占用1个字节存储空间，内部表示为对应的ASCII码值

- 不可以与其他类型进行算术运算

标识符、空白与注释

CACT标识符可以由大小写字母、数字以及下划线组成，但必须以字母或者下划线开头。

一个及以上的空白符、空行、Tab字符可以出现在任意的词法符号之间。

CACT支持单行注释和多行注释，如下：

- 单行注释：以‘//’开始，直到换行符结束，不包括换行符。
- 多行注释：以‘/*’开始，直到第一次出现‘*/’时结束。注意，CACT多行注释不支持形如/* /* */* /的嵌套，注释嵌套代码会识别为文法错误。

运算符优先级

CACT支持以下运算符，优先级从高到低排列。

优先级	运算符	含义	结合方向
1	[...]	数组下标	左到右
	(...)	圆括号	
2	-	单目负号	**右到左**
	+	单目正号	
	!	逻辑非	
3	*	乘法	左到右
	/	除法	
	%	取余	
4	+	双目加法	
	-	双目减法	
5	<=	小于等于	
	>=	大于等于	
	<	小于	
	>	大于	
6	==	等于	
	!=	不等于	
7	&&	逻辑与	
8	||	逻辑或	

注：赋值、逗号不作为运算符。

1. 赋值语句用等号'='，不支持连续赋值。
2. 逗号使用范围受限，只出现在变常量声明、初始值、函数声明&调用。

文法规范

在文法规范中使用以下符号：

符号	释义
foo	表示foo是非终结符
Ident	加粗，表示**Ident**是终结符
'foo'	表示'foo'是字面值终结符，加粗为关键字
[...]	方括号内包含的项可重复0次或1次
{...}	花括号内包含的项可重复任意次（0次、1次或多次）
(...)	匹配括号内的字符
~(...)	匹配除括号内的字符之外的字符
.	匹配任何字符
|	表示可选项“或”
-	ASCII字符范围

声明&定义

	非终结符	→	规则
编译单元	CompUnit	→	[CompUnit] (Decl | FuncDef)
声明	Decl	→	ConstDecl | VarDecl
常量声明	ConstDecl	→	'**const**' BType ConstDef { ',' ConstDef } ';'
基本类型	BType	→	'**int**' | '**float**' | '**char**'
常量定义	ConstDef	→	**Ident** { '[' IntConst ']' } '=' ConstInitVal
初始值	ConstInitVal	→	ConstExp | '{' [ConstInitVal { ',' ConstInitVal }] '}'
变量声明	VarDecl	→	BType VarDef { ',' VarDef } ';'
变量定义	VarDef	→	**Ident** { '[' IntConst ']' } ['=' ConstInitVal]
函数定义	FuncDef	→	FuncType **Ident** '(' [FuncFParams] ')' Block
函数类型	FuncType	→	'**void**' | '**int**' | '**float**' | '**char**'
形参列表	FuncFParams	→	FuncFParam { ',' FuncFParam }
函数形参	FuncFParam	→	BType Ident ['[' IntConst? ']' { '[' IntConst ']' }]

语句&表达式

	非终结符	→	**规则**
语句块	Block	→	'{' { BlockItem } '}'
语句块项	BlockItem	→	Decl | Stmt
语句	Stmt	→	LVal '=' Exp ';' | [Exp] ';' | Block | '**return**' Exp? | '**if**' '(' Cond ')' Stmt ['**else**' Stmt] | '**while**' '(' Cond ')' Stmt | '**break**' ';' | '**continue**' ';'
表达式	Exp	→	AddExp
常量算式	ConstExp	→	AddExp 注：使用的Ident必须是常量
条件算式	Cond	→	LOrExp
左值算式	LVal	→	**Ident** { '[' Exp ']' }
数值	Number	→	**IntConst** | **CharConst** | **FloatConst**
函数实参表	FuncRParams	→	Exp { ',' Exp }
优先级1	PrimaryExp	→	'(' Exp ')' | LVal | Number
优先级2	UnaryExp	→	PrimaryExp | ('+' | '-' | '!') UnaryExp | **Ident** '(' [FuncRParams] ')' 注：'!'仅出现在条件表达式中
优先级3	MulExp	→	UnaryExp | MulExp (' *' | '/' | '%') UnaryExp
优先级4	AddExp	→	MulExp | AddExp ('+'|'-') MulExp
优先级5	RelExp	→	AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp
优先级6	EqExp	→	RelExp | EqExp ('==' | '!=') RelExp
优先级7	LAndExp	→	EqExp | LAndExp '&&' EqExp
优先级8	LOrExp	→	LAndExp | LOrExp '||' LAndExp

注意：

1. 常量表达式不在语句块中出现。
2. 条件表达式只在If和While的条件语句中出现，不出现在赋值语句中。
3. 连加（a+b+c）会出现，但诸如连等(a=b=c)、连续比较(a==b==c，a < b < c)则不会出现，不必考虑。

终结符

	终结符	→	规则
标识符	**Ident**	→	(a-zA-Z_) { (a-zA-Z0-9_) }
换行符	NewLine	→	'\r' ['\n'] | '\n'
空白符	WhiteSpace	→	{ ' ' | '\t' }
单行注释	LineComment	→	'//' { ~('\r'|'\n') }
多行注释	BlockComment	→	'/*' { . } '*/'
整型常量	**IntConst**	→	**DecimalConst** | **OctalConst** | **HexadecConst**
十进制常量	**DecimalConst**	→	nonzero-digit | digit
八进制常量	**OctalConst**	→	'0' | OctalConst octal-digit
十六进制常量	**HexadecConst**	→	hexadecimal-prefix hexadecimal-digit | HexadecConst hexadecimal-digit
8位字符常量	**CharConst**	→	'''character'''
十进制非零数字	nonzero-digit	→	'1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
十进制数字	digit	→	'0' | nonzero-digit
八进制数字	octal-digit	→	'0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'
十六进制前缀	hexadecimal-prefix	→	'0x' | '0X'
十六进制数字	hexadecimal-digit	→	'0'...'9' | 'a'...'f' | 'A'...'F'
字符	character	→	除换行符和单引号外的任意ASCII字符

浮点型数 FloatConst 的定义参 考 ISO/IEC 9899 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf 第 57 页关于浮点常量的定义。

语义规范

符合上述文法规范的代码集合，并不都是合法的CACT程序，而是CACT语言的超集。

下面进一步给出CACT语言的语义。

文件

每个CACT程序的源码存储在一个扩展名为cact的文件中。

CACT文件有且仅有一个标识为 main、无参数、返回类型为int的函数定义，main函数是程序运行的起始点。文件中还可以包含若干全局变量声明、常量声明和其他函数定义。

数组类型

CACT支持三种基本类型：int、float、char。CACT支持以基本类型为元素的多维数组，索引从0开始，以行优先顺序存储。如：int a[M][N]，其中M和N都是整数常量，通过a[i][j]方式访问，其中i取值范围0到M-1，j取值范围0到N-1。

数组可以在任何作用域中声明。函数声明中可以隐藏最外层维度，如int f(int a[][N])。除此之外，数组长度需要显式给出IntConst终结符，而不允许是变量。

初始化

每个变量/常量/数组在声明时都会被初始化，分为显式、隐式两种：

- 显式初始化：CACT限制初值表达式必须是常数，例如
  - int a = 0; //合法
  - int b = a + 5; //非法
  - const int c = 4; //合法
  - const int d = 4 + 5; //非法
  - float e = 1.5; //非法
  - float f = 3.2f; //合法
  - int a[4] = 4; //非法
  - int a[4] = {1,{2,3}}; //非法
  - int a[2][2] = {4, 5, 6}; //合法，数组内容为{ {4,5}, {6,0} }
  - int a[2][2] = {{4,5},{6,7}}; //合法
- 隐式初始化：未被显式初始化的整型和浮点型变量/常量/数组被默认初始化为0，布尔型被默认初始化为false。

全局变量/常量/数组在程序开始执行时被初始化，局部数组在程序每次进入作用域时被重新初始化。

提示：由于CACT所有变量/常量、所有数组的各维都有确定的长度，所以不需要做堆管理。全局变量/常量/数组放置在静态数据区中，局部变量/常量/数组在栈上动态分配。

函数

C语言文法中定义了函数声明，这是为了函数调用可能出现的循环依赖而存在的：

```
void a();
void b(){
  a();
}
void a(){
  b();
}
```

为简单起见，CACT语言不考虑循环依赖的情况，函数声明永不孤立存在。保证在所有实例中，函数调用都会出现在函数定义之后。

函数定义由返回类型、函数名、参数列表（可零）、函数体。下面作逐一说明：

1. 返回类型：只会是三种基本类型或void。
    1. 基本类型：函数内所有路径都应当结束于return Exp的语句。如果路径不含有return语句，或return不含有表达式，则被认为是文法错误。
    2. void：函数体中可以不包含return语句，如果有，不能包含返回值。
2. 函数名：与变量名要求相同。
3. 形参列表：可以包含0或多个形式参数。合法的形式参数类型分为以下两类。
    - 基本类型：三种基本类型，按值传递。
    - 数组类型：实际传递的是数组的起始地址，除了最外层维度外，各内层维度的长度需显式声明。最外层长度分为两种情况：
        - 显式情形：如int a[A][M][N]，则调用点传入数组各维度长度都需进行静态检查。
        - 隐式情形：如int a[][M][N]，最外层维度用一对中括号[]隐去，不检查调用点实际参数的最外层维度长度，但各内层维度长度需要进行静态检查。
4. 函数体：由若干变量声明和语句组成。

函数调用的执行过程包括以下几步：

1. 把函数调用点的实际参数值，作为初始值赋值给被调用函数的形式参数。实际参数的类型和个数，必须与形式参数相匹配。
2. 顺序执行被调用函数的函数体语句，遇到控制语句则依条件选择子语句执行。
3. 顺序执行直到 (1)return语句 或 (2)函数体结束 时，记录返回值并跳转到调用点的下一条语句。

标识符与作用域

CACT中，所有的标识符要求先定义再使用。

1. 常量/变量/数组：在使用前必须先声明。
2. 函数：必须在函数定义之后才能被调用，但允许递归调用。

标识符的定义-使用关系（Def-Use）由定义域规定。CACT语言支持三种作用域：

- 全局作用域：包括顶层变量/常量/函数定义。
- 函数作用域：包括函数定义中的形参，局部变量/常量。
- 局部作用域：由函数体内的语句块（Block）创建。

对程序的任意语句，都至少有两个有效的作用域：全局作用域和函数作用域。作用域存在嵌套关系：内层作用域可以隐藏掉外层作用域的同名标识符。嵌套关系用下面的例子进一步解释：

```
const int a = 1;

void foo(){
  const int a = 2;

  {
    const int a = 3;
    // local scope, a = 3
  }

  if( a > 0 ){
    const int a = 4;
    // local scope, a = 4
  }

  // function scope, a = 2
}

int main(){
  foo();
  return a; // global scope, a = 1
}
```

注意：

1. **同一作用域中，同名变量的两次定义视为文法错误。**
2. 在不违反上述条件下，**变量名可与函数名相同。**

```
int main(){
  const int a = 1;
  int a = 2; // wrong!
  return a;
}
```

控制语句

CACT中控制语句只有两种：if(else)语句和while语句，语义与C语言相同。

- 对于if语句：CACT中if语句遵循就近匹配原则。else匹配最近的if。用以下伪代码定义。

  ```
  if( 判断cond为真 ){
     执行if后面的语句(块);
  } else if( 如果有else字句 ){
     执行else后面的语句(块);
  } else {
     跳过if后面的语句(块);
  }
  ```

- 对于while语句：判断cond为真，则执行while后面的语句(块)，并无条件跳转回到cond；否则，跳过while后面的语句(块)。

左值与赋值

左值是可以出现在赋值语句左边的值，因此得名。CACT支持两种左值：变量与数组元素。每个左值都属于三种基本类型之一。

对于赋值语句，CACT完全采用值拷贝语义，LVal=Exp语句把Exp的值拷贝到LVal中，要求LVal与Exp类型必须完全相同。再次注意，**CACT不支持任何形式的类型转换。**

此外，在函数体中允许对形式参数进行再次赋值，只在该函数的作用域内有效，不会改变调用者传入参数的值。

运算

CACT支持基本的算术运算（+、-、 *、/、%）、相等性运算（==、!=）、关系运算（<、>、<=、>=）和逻辑运算（!、&&、||），语义与C语言相同。**保证样例中不出现连等（a==b==c）。**

在CACT中，逻辑非运算符(!)只对布尔类型的变量及表达式使用，取余运算(%)不支持浮点数。

运行时库

CACT语言本身没有提供输入/输出(IO)构造，但IO是程序的必要部分，只能由运行时库提供。

为了测试生成汇编的正确性，CACT内置以下7个函数：

1. **void** print_int(**int**)

   输出一个整数的值

2. **void** print_float(**float**)

   输出一个单精度浮点型变量的值

3. **void** print_char(**char**)

   输出一个字符

4. **int** get_int()

   输入一个整数，返回对应的整数值。

5. **float** get_float()

   输入一个浮点数，返回对应的float型浮点值。

6. **char** get_char()

   输入一个字符，返回对应的char型值。

注：以上API的实现不作为作业内容，最后通过课程提供的链接器生成可执行程序。只要在全局作用域中建立上述符号即可。