# CACT语言规范

本实验的任务是为CACT语言写一个编译器。CACT语言的设计受SysY2020启发，CACT是C语言的一个子集，并对C语言做了一些扩展。

# 词法约束

## 关键字

CACT语言的所有关键字均为小写。关键字和标识符是大小写敏感的，如：**if**是关键字，但IF是一个标识符；foo和Foo是两个不同的标识符。

CACT语言的关键字为：

**break continue if else while return true false void bool int float double**

## 注释

CACT支持单行注释和多行注释，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，包括结束处的'*/'。

注意，CACT不支持形如/* /* */ */的嵌套注释，代码中出现的嵌套注释会被识别为语法错误。

空白符可以出现在任意的词法符号之间，可以由一个及以上的空格、Tab字符组成。

## 基本类型

CACT有四种基本类型：int、float、double、bool。

int表示32位有符号整型常数，支持十进制、十六进制、八进制。十进制整数由0-9十个数字组成，数字与数字中间没有如空格之类的分隔符。除了"0"之外，十进制整数的首位数字不为0。如：0、234、10000。八进制整数以0开始，后由连续的0-7之间的数字组成，如0127。十六进制整数以0x或0X开始，后由连续的0-9、a-f、A-F中的数字组成。如：0XFa32。整型常数均无后缀。

CACT支持有符号浮点型常数。普通形式的浮点数由一串数字与一个小数点组成，小数点的前或后必须有数字出现。如：0.7、9.00、12.43、.5、8.等。浮点型常数还可以以指数形式（即科学计数法）表示。指数形式的浮点数必须包括基数、指数符号和指数三个部分，且三部分依次出现。基数部分由可带小数点（也可不带）的一串数字（0-9）组成；若有小数点，则小数点可以出现在基数数字串的任何位置；指数符号为"E"或"e"；指数部分由可带"+"或"-"（也可不带）的一串数字（0-9）组成，"+"或"-"（如果有）必须出现在数字串之前。例如01.23E12（表示$1.23\times10^{12}$）、43.0e-4F（表示$43.0\times10^{-4}$）、.5E03（表示$0.5\times10^{3}$）、3E2f（表示$3x10^{2}$）。你可以假设输入的浮点数都符合IEEE754标准。浮点型常数后可跟'F'或 'f'后缀。

CACT中，浮点型字面值分为double与float类型，若字面值后面跟'F'或'f'后缀，则为float型，否则默认为double型。double为8字节有符号浮点型常数，float为4字节有符号浮点型常数。

bool型变量的值为true或false。

## 标识符

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。

# 语法规范

下文中涉及的符号释义如下：

| 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是终结符 |
| [...] | 表示方括号内包含的为可重复0次或1次的项 |
| {...} | 表示花括号内包含的为可重复0次或多次的项 |
| ~(...) | 表示除括号内的字符之外的字符 |
| . | 表示匹配任何字符 |
| \| | 区分可选项 |

CACT语言的文法表示如下：其中CompUnit为开始符号：

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**' | '**bool**' | '**float**' | '**double**'

常量定义 ConstDef → **Ident** [ '[' IntConst ']' ] '=' ConstInitVal

初值 ConstInitVal → ConstExp

| '{' [ ConstExp { ',' ConstExp } ] '}'

变量声明 VarDecl → BType VarDef { ',' VarDef } ';'

变量定义 VarDef → **Ident** [ '[' IntConst ']' ]

| **Ident** [ '[' IntConst ']' ] '=' ConstInitVal

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

函数类型 FuncType → '**void**' | '**int**' | '**float**' | '**double**' | '**bool**'

函数形参表 FuncFParams → FuncFParam { ',' FuncFParam }

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

语句块 Block → '{' { BlockItem } '}'

语句块项 BlockItem → Decl | Stmt

语句 Stmt → LVal '=' Exp ';' | [Exp] ';' | Block

| '**if**' '(' Cond ')' Stmt [ '**else**' Stmt ]

| | | |
|---|---|---|
| | | \| '**while**' '(' Cond ')' Stmt |
| | | \| **'break**' ';' \| '**continue**' ';' |
| | | \| '**return**' [Exp] ';' |
| 表达式 | Exp | → AddExp \| **BoolConst** |
| 条件表达式 | Cond | → LOrExp |
| 左值表达式 | LVal | → **Ident** ['[' Exp ']'] |
| 基本表达式 | PrimaryExp | → '(' Exp ')' \| LVal \| Number |
| 数值 | Number | → **IntConst** \| **DoubleConst** \| **FloatConst** |
| 一元表达式 | UnaryExp | → PrimaryExp \| **Ident** '(' [FuncRParams] ')' |
| | | \| UnaryOp UnaryExp |
| 单目运算符 | UnaryOp | → '+' \| '−' \| '!' |
| 函数实参表 | FuncRParams | → Exp { ',' Exp } |
| 乘除模表达式 | MulExp | → UnaryExp \| MulExp ('*' \| '/' \| '%') UnaryExp |
| 加减表达式 | AddExp | → MulExp \| AddExp ('+' \| '−') MulExp |
| 关系表达式 | RelExp | → AddExp \| RelExp ('<' \| '>' \| '<=' \| '>=') AddExp \| **BoolConst** |
| 相等性表达式 | EqExp | → RelExp \| EqExp ('==' \| '!=') RelExp |
| 逻辑与表达式 | LAndExp | → EqExp \| LAndExp '&&' EqExp |
| 逻辑或表达式 | LOrExp | → LAndExp \| LOrExp '\|\|' LAndExp |
| 常量表达式 | ConstExp | → Number \| **BoolConst** |
| **终结符:** | | |
| 标识符 | Ident | → /* 根据上文描述自行设计 */ |
| 整型常量 | IntConst | → /* 根据上文描述自行设计 */ |
| 布尔型常量 | BoolConst | → /* 根据上文描述自行设计 */ |
| 单精度浮点常量 | FloatConst | → /* 根据上文描述自行设计 */ |
| 双精度浮点常量 | DoubleConst | → /* 根据上文描述自行设计 */ |
| 换行符 | NewLine | → '\r' '\n'? \| '\n' |
| 空白符 | WhiteSpace | → { ' ' \| '\t'} |
| 单行注释 | LineComment | → '//' {~('\r'\|'\n')} |
| 多行注释 | BlockComment | → '/*' {.} '*/' |

# 语义约束

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

# 文件

每个CACT程序的源码存储在一个扩展名为cact的文件中。该文件中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。

# 类型

CACT支持四种基本类型：int、float、double、bool，以及元素为四种基本类型且按行优先存储的一维数组，如：int [N]。const修饰符用于声明常量。

数组可以在任何作用域中声明，所有的数组均是一维的，数组长度需要显式给出，而不允许是未知的。数组的长度必须由整数常量，即IntConst指定。数组的的长度为N时，其索引为0到N-1，数组的元素通过*a[i]*的方式引用。

每个变量/常量/数组在声明时都必须被（显式或默认）初始化。显式初始化时变量/常量/数组声明中指定的初值表达式必须是常数，即IntConst、DoubleConst、FloatConst或BoolConst。因为CACT**不支持任何形式的类型转换**，所以需要特别注意变量的类型。如变量声明：

- 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

变量/常量/数组声明中指定的初值要与其类型一致。如下形式的定义不满足CACT语义约束：

- int a[4] = 4;
- int a[3] = {1, {2, 3}};
- int a = {1, 2, 3};

未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，布尔型变量/常量/数组被默认初始化为false。程序开始执行时，全局数组的每个元素都要被初始化。局部数组在程序执行进入该数组的作用域时被初始化。每次程序执行进入一个变量/常量/数组的作用域时，其值都要被重置为默认值。

因为在CACT中，**数组大小都是静态确定**的，所以全局数组可以被分配在程序的静态数据区而不必在堆上分配。局部数组可以在栈上动态分配。

# 作用域及标识符

在程序的任意位置，都至少有两个有效的作用域：全局作用域和函数作用域。全局作用域包括顶层变量/常量、定义的函数。函数作用域包括全局变量/常量和局部变量/常量，以及函数定义中的形参。其余的局部作用域由代码中跟在**if**、**while**之后的Block（语句块）创建。函数作用域中定义的标识符可以隐藏掉全局作用域中的同名标识符。同理，局部作用域中定义的标识符可以隐藏掉该局部作用域外的同名标识符。局部作用域中的变量的生存期在该函数或语句块内；

CACT中，所有的标识符要求先定义再使用。例如：

- 常量/变量/数组在使用前必须先声明。
- 函数必须在函数头定义之后，才能被调用。（允许递归）

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。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。全局变量/常量和函数声明的作用域从该声明处开始到文件结尾。

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；
- 变量名可以与函数名相同。

## 左值

CACT由三种类型的左值：变量、数组元素、数组。每个左值都有一个类型，为四种基本类型或其数组类型之一。当左值为变量或数组时，不能出现后面的方括号。

## 函数

CACT中函数声明可以带参数也可以不带参数，参数的类型可以是四种基本类型或者其对应的数组类型；函数可以返回基本类型的值，或者不返回值（即声明为void类型）。当参数为基本类型时，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，并且其长度省去。当返回值类型为基本类型时，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。返回值类型为void时，函数内只能出现不带返回值的return语句。

函数体由若干变量声明和语句组成。

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。

函数调用涉及到：（1）把参数值从调用函数传递到被调用函数；（2）执行被调函数的函数体；（3）返回到调用函数，可以返回一个结果。

参数传递时，被调函数的形参被视为该函数的局部变量，并通过赋值，使用实参的值进行初始化。然后通过顺序执行被调函数的语句来执行其函数体。

没有声明返回类型（返回类型为void）的函数，只能作为一个语句（Stmt）被调用，即其不能被当作一个表达式（Exp）来使用。当return语句被调用，或者程序执行到被调函数的最后一条语句时，程序控制流返回到调用函数。

有返回值的函数可以作为表达式（Exp）的一部分被调用，也可作为语句（Stmt）被调用。作为语句被调用时，函数的返回值被忽略。

## 控制语句

CACT中的控制语句包括if和while语句，其语义分别和C语言中的if、while类似。对于if语句：首先判断cond，如果为true，执行true语句块；否则，有else语句块的话，执行else语句块。CACT中if语句遵循就近匹配原则。

## 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（!、&&、||），true表示真，false表示假。其中，!运算符只能对布尔类型的变量及表达式使用。CACT中不接受三目运算符。浮点数不支持%运算。

不同于C语言，CACT中的运算可以在数组和数组之间、标量和标量之间进行。数组与数组之间的运算为逐元素进行。

关系运算符用来比较整形和浮点型表达式，相等性运算符用于整形、浮点型和布尔型表达式，逻辑运算只能用于布尔型表达式，且所有运算符的两个表达式的类型必须相同。关系运算符和相等性运算符的结果是bool类型。

对于int、float、double型数组，CACT支持数组与数组间的算术运算（+、-、*、/、%）及赋值运算（=），如：

```
int a[100], b[100], c[100];
a = b + c;          // 合法
float fa[100], fb[100], fc[100];
fa = fb / fc;       // 合法
```

对于bool型数组，CACT仅支持bool数组间的赋值运算（=），如：

```
bool a[100], b[100], c[100];
a = b;              // 合法
a = b + c;          // 不合法
```

表达式遵循正常的求值规则。**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**，在上面的CACT文法中已体现出优先级与结合性的定义。没有其他约束时，相同优先级的运算符从左到右分析。

## 赋值

不同于C语言，CACT中的赋值运算可以在数组和数组之间、标量和标量之间进行。对于数组和标量，CACT都采用值拷贝语义，LVal = Exp语句把Exp的值拷贝到LVal中。在一个赋值语句中，LVal和Exp的类型必须相同。对于数组与数组之间的赋值，数组的大小、类型必须匹配。再次注意，CACT**不支持任何形式的类型转换。**

在CACT中，在函数作用域中对形参变量赋值是合法的，对形参变量的赋值只在该函数的作用域内有效。

## I/O

CACT语言本身没有提供输入/输出(I/O)的语言构造，I/O是以运行时库方式提供，这些库函数不用在 CACT 程序中声明即可在 CACT 的函数中使用。CACT库函数提供四种用于输出的库函数以及三种用于输入的库函数：

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_double(double)

   输出一个双精度浮点型变量的值

4. void print_bool(bool)

   输出一个布尔型变量的值

5. int get_int()

   输入一个整数，返回对应的整数值。

6. float get_float()

   输入一个浮点数，返回对应的float型浮点值。

7. double get_double()

   输入一个浮点数，返回对应的double型浮点值。