语义分析与中间代码生成

本章实验为**实验二**，任务是在词法分析和语法分析程序的基础上编写一个程序，**对CMINUS源代码进行语义分析和类型检查，并打印分析结果**；在此基础上，**完成中间代码生成**。

与实验一不同的是，实验二不再借助已有的工具，所有的任务都必须手写代码来完成。虽然语义分析在整个编译器的实现中并不是难度最大的任务，但却是最细致、琐碎的任务。因此需要用心地设计诸如符号表、变量类型等数据结构的实现细节，从而正确、高效地实现语义分析的各种功能。另外，请继续保持良好的代码风格。

实验内容

1. 语义分析和类型检查

CMINUS语言有如下特性：

1) **假设1**：整型（int）变量不能与浮点型（float）变量相互赋值或者相互运算。
2) **假设2**：仅有int型变量才能进行逻辑运算或者作为if和while语句的条件；仅有int型和float型变量才能参与算术运算。
3) **假设3**：任何函数只进行一次定义，无法进行函数声明。
4) **假设4**：所有变量（包括函数的形参）的作用域都是全局的，即程序中所有变量均不能重名。
5) **假设5**：结构体间的类型等价机制采用**名等价（Name Equivalence）**的方式。
6) **假设6**：函数无法进行嵌套定义。
7) **假设7**：结构体中的域不与变量重名，并且不同结构体中的域互不重名。

PS: 你的实验应基于上述假设1至7完成，即违反假设会导致各种语义错误，不过本实验将只对后面讨论的17种错误类型进行考察。此外，你可以安全地假设输入文件中不包含注释、八进制数、十六进制数、以及指数形式的浮点数，也不包含任何词法或语法错误（除了特别说明的针对选做要求的测试）。

你的程序需要对输入文件进行语义分析（输入文件中可能包含函数、结构体、一维和高维数组）并检查如下类型的错误：

1) **错误类型1**：变量在使用时未经定义。
2) **错误类型2**：函数在调用时未经定义。
3) **错误类型3**：变量出现重复定义，或变量与前面定义过的结构体名字重复。
4) **错误类型4**：函数出现重复定义（即同样的函数名出现了不止一次定义）。
5) **错误类型5**：赋值号两边的表达式类型不匹配。
6) **错误类型6**：赋值号左边出现一个只有右值的表达式。
7) **错误类型7**：操作数类型不匹配或操作数类型与操作符不匹配（例如整型变量与数组变量相加减，或数组（或结构体）变量与数组（或结构体）结构体变量相加减）。
8) **错误类型8**：return语句的返回类型与函数定义的返回类型不匹配。

9) **错误类型9**：函数调用时实参与形参的数目或类型不匹配。
10) **错误类型10**：对非数组型变量使用“[…]”（数组访问）操作符。
11) **错误类型11**：对普通变量使用“(…)”或“()”（函数调用）操作符。
12) **错误类型12**：数组访问操作符“[…]”中出现非整数（例如a[1.5]）。
13) **错误类型13**：对非结构体型变量使用“.”操作符。
14) **错误类型14**：访问结构体中未定义过的域。
15) **错误类型15**：结构体中域名重复定义（指同一结构体中），或在定义时对域进行初始化（例如struct A { int a = 0; }）。
16) **错误类型16**：结构体的名字与前面定义过的结构体或变量的名字重复。
17) **错误类型17**：直接使用未定义过的结构体来定义变量。

其中，要注意两点：一是关于数组类型的等价机制，同C语言一样，只要数组的基类型和维数相同我们即认为类型是匹配的，例如int a[10][2]和int b[5][3]即属于同一类型；二是我们允许类型等价的结构体变量之间的直接赋值（见后面的测试样例），这时的语义是，对应的域相应赋值（数组域也如此，按相对地址赋值直至所有数组元素赋值完毕或目标数组域已经填满）。

输入

你的程序的输入是一个包含CMINUS源代码的文本文件，该源代码中可能会有语义错误。你的程序需要能够接收一个输入文件名作为参数。例如，假设你的程序名为cc、输入文件名为test1、程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1即可获得以test1作为输入文件的输出结果。

输出

助教将检查附录中17个必做样例，请保证你的程序能够正确处理这些样例，并按照下述格式打印输出。本实验要求通过标准输出打印程序的运行结果。对于那些没有语义错误的输入文件，你的程序不需要输出任何内容。对于那些存在语义错误的输入文件，你的程序应当输出相应的错误信息，这些信息包括错误类型、出错的行号以及说明文字，其格式为：

Error type [错误类型] at Line [行号]: [说明文字].

说明文字的内容没有具体要求，但是错误类型和出错的行号一定要正确，因为这是判断输出的错误提示信息是否正确的唯一标准。请严格遵守实验要求中给定的错误分类，否则将影响你的实验评分。

2. 中间代码生成

在本实验中，我们对输入的CMINUS语言源代码文件做如下假设：

1) **假设1**：不会出现注释、八进制或十六进制整型常数、浮点型常数或者变量。
2) **假设2**：不会出现类型为结构体或高维数组（高于1维的数组）的变量。
3) **假设3**：任何函数参数都只能为简单变量，也就是说，结构体和数组都不会作为参数传入函数中。
4) **假设4**：没有全局变量的使用，并且所有变量均不重名。

5) **假设5**：函数不会返回结构体或数组类型的值。
6) **假设6**：函数只会进行一次定义（没有函数声明）。
7) **假设7**：输入文件中不包含任何词法、语法或语义错误。

你的程序需要将符合以上假设的CMINUS源代码翻译为中间代码，中间代码的形式及操作规范如表1所示，表中的操作大致可以分为如下几类：

表1. 中间代码的形式及操作规范。

语法	描述
LABEL x :	定义标号x。
FUNCTION f :	定义函数f。
x := y	赋值操作。
x := y + z	加法操作。
x := y - z	减法操作。
x := y * z	乘法操作。
x := y / z	除法操作。
x := &y	取y的地址赋给x。
x := *y	取以y值为地址的内存单元的内容赋给x。
*x := y	取y值赋给以x值为地址的内存单元。
GOTO x	无条件跳转至标号x。
IF x [relop] y GOTO z	如果x与y满足[relop]关系则跳转至标号z。
RETURN x	退出当前函数并返回x值。
DEC x [size]	内存空间申请，大小为4的倍数。
ARG x	传实参x。
x := CALL f	调用函数，并将其返回值赋给x。
PARAM x	函数参数声明。
READ x	从控制台读取x的值。
WRITE x	向控制台打印x的值。

1) 标号语句LABEL用于指定跳转目标，注意LABEL与x之间、x与冒号之间都被空格或制表符隔开。
2) 函数语句FUNCTION用于指定函数定义，注意FUNCTION与f之间、f与冒号之间都被空格或制表符隔开。
3) 赋值语句可以对变量进行赋值操作（注意赋值号前后都应由空格或制表符隔开）。赋值号左边的x一定是一个变量或者临时变量，而赋值号右边的y既可以是变量或临时变量，也可以是立即数。如果是立即数，则需要在其前面添加“#”符号。例如，如果要将常数5赋给临时变量t1，可以写成t1 := #5。
4) 算术运算操作包括加、减、乘、除四种操作（注意运算符前后都应由空格或制表符隔开）。赋值号左边的x一定是一个变量或者临时变量，而赋值号右边的y和z既可以是变量或临时变量，也可以是立即数。如果是立即数，则需要在其前面添加“#”符号。例

如，如果要将变量a与常数5相加并将运算结果赋给b，则可以写成b := a + #5。

5) 赋值号右边的变量可以添加“&”符号对其进行取地址操作。例如，b := &a + #8代表将变量a的地址加上8然后赋给b。
6) 当赋值语句右边的变量y添加了“*”符号时代表读取以y的值作为地址的那个内存单元的内容，而当赋值语句左边的变量x添加了“*”符号时则代表向以x的值作为地址的那个内存单元写入内容。
7) 跳转语句分为无条件跳转和有条件跳转两种。无条件跳转语句GOTO x会直接将控制转移到标号为x的那一行，而有条件跳转语句（注意语句中变量、关系操作符前后都应该被空格或制表符分开）则会先确定两个操作数x和y之间的关系（相等、不等、小于、大于、小于等于、大于等于共6种），如果该关系成立则进行跳转，否则不跳转而直接将控制转移到下一条语句。
8) 返回语句RETURN用于从函数体内部返回值并退出当前函数，RETURN后面可以跟一个变量，也可以跟一个常数。
9) 变量声明语句DEC用于为一个函数体内的局部变量声明其所需要的空间，该空间的大小以字节为单位。这个语句是专门为数组变量和结构体变量这类需要开辟一段连续的内存空间的变量所准备的。例如，如果我们需要声明一个长度为10的int类型数组a，则可以写成DEC a 40。对于那些类型不是数组或结构体的变量，直接使用即可，不需要使用DEC语句对其进行声明。变量的命名规范与之前的实验相同。另外，在中间代码中不存在作用域的概念，因此不同的变量一定要避免重名。
10) 与函数调用有关的语句包括CALL、PARAM和ARG三种。其中PARAM语句在每个函数开头使用，对于函数中形参的数目和名称进行声明。例如，若一个函数func有三个形参a、b、c，则该函数的函数体内前三条语句为：PARAM a、PARAM b和PARAM c。CALL和ARG语句负责进行函数调用。在调用一个函数之前，我们先使用ARG语句传入所有实参，随后使用CALL语句调用该函数并存储返回值。仍以函数func为例，如果我们需要依次传入三个实参x、y、z，并将返回值保存到临时变量t1中，则可分别表述为：ARG z、ARG y、ARG x和t1 := CALL func。注意ARG传入参数的顺序和PARAM 声明参数的顺序正好相反。ARG语句的参数可以是变量、以#开头的常数或以&开头的某个变量的地址。
11) 输入输出语句READ和WRITE用于和控制台进行交互。READ语句可以从控制台读入一个整型变量，而WRITE语句可将一个整型变量的值写到控制台上。

除以上说明外，注意关键字及变量名都是大小写敏感的，也就是说“abc”和“AbC”会被作为两个不同的变量对待，上述所有关键字（例如CALL、IF、DEC等）都必须大写，否则虚拟机小程序会将其看作一个变量名。

在本实验中，你可能需要在“**1. 语义分析和类型检查**”的程序中做如下更改：在符号表中预先添加read和write 这两个预定义的函数。其中read函数没有任何参数，返回值为int型（即读入的整数值），write 函数包含一个int类型的参数（即要输出的整数值），返回值也为int型（固定返回0）。添加这两个函数的目的是让CMINUS源程序拥有可以与控制台进行交互的接口。在中间代码翻译的过程中，read函数可直接对应READ操作，write函数可直接对应WRITE操作。

输入

你的程序的输入是一个包含CMINUS源代码的文本文件，你的程序需要能够接收一个输入文件名和一个输出文件名作为参数。例如，假设你的程序名为cc、输入文件名为test1、输出文件名为out1.ir，程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1 out1.ir即可将输出结果写入当前目录下名为out1.ir的文件中。

输出

本实验要求你的程序将运行结果输出到文件。输出文件要求每行一条中间代码，每条中间代码的含义如前文所述。如果输入文件包含多个函数定义，则需要通过FUNCTION语句将这些函数隔开。FUNCTION语句和LABEL语句的格式类似，具体例子见后面的样例。

对每个特定的输入，并不存在唯一正确的输出。我们将使用虚拟机小程序对你的中间代码的正确性进行测试。任何能被虚拟机小程序顺利执行并得到正确结果的输出都将被接受。此外，虚拟机小程序还会统计你的中间代码所执行过的各种操作的次数，以此来估计你的程序生成的中间代码的效率。

测试环境和提交要求

你的程序将在如下环境中被编译并运行：

GNU Linux Release: Ubuntu 12.04, kernel version 3.2.0-29；

1) GCC version 4.6.3；
2) GNU Flex version 2.5.35；
3) GNU Bison version 2.5。

一般而言，只要避免使用过于冷门的特性，使用其它版本的Linux或者GCC等，也基本上不会出现兼容性方面的问题。注意，实验二的检查过程中不会去安装或尝试引用各类方便编程的函数库（如glib等），因此请不要在你的程序中使用它们。

实验二要求提交如下内容：

1) Flex、Bison以及C语言的可被正确编译运行的源程序。
2) 一份PDF格式的实验报告，内容包括：
    a) 你的程序实现了哪些功能？简要说明如何实现这些功能。清晰的说明有助于助教对你的程序所实现的功能进行合理的测试。
    b) 你的程序应该如何被编译？可以使用脚本、makefile或逐条输入命令进行编译，请详细说明应该如何编译你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数。
    c) 实验报告的长度不得超过三页！所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，而相对简单的、任何人都可以做的内容则可不提或简单地提一下，尤其要避免大段地向报告里贴代码。实验报告中所出现的最小字号不得小于五号字（或英文11号字）。