4. 中间代码生成

本章实验为**实验三**，任务是在词法分析、语法分析和语义分析程序的基础上，将C--源代码翻译为中间代码。理论上中间代码在编译器的内部表示可以选用**树形结构（抽象语法树）**或者**线形结构（三地址代码）**等形式，为了方便检查你的程序，我们要求将中间代码输出成线性结构，从而可以使用我们提供的虚拟机小程序（附录B）来测试中间代码的运行结果。

需要注意的是，由于在后面的实验中还会用到本次实验已经写好的代码，因此保持一个良好的代码风格、系统地设计代码结构和各模块之间的接口对于整个实验来讲相当重要。

4.1 实验内容

4.1.1 实验要求

在本次实验中，我们对输入的C--语言源代码文件做如下假设（注意，假设2和3可能因后面的不同选做要求而有所改变）：

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（函数也必有return语句）。

你的程序需要将符合以上假设的C--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：

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

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。

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。注意：当函数参数是结构体或数组时，ARG语句的参数为结构体或数组的地址（即以传引用的方式实现函数参数传递）。

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。

在实验三中，你可能需要在实验二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C--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。

除此之外，你的程序可以选择完成以下部分或全部的要求：

1) **要求3.1**：修改前面对C--源代码的假设2和3，使源代码中：
   a) 可以出现结构体类型的变量（但不会有结构体变量之间直接赋值）。
   b) 结构体类型的变量可以作为函数的参数（但函数不会返回结构体类型的值）。
2) **要求3.2**：修改前面对C--源代码的假设2和3，使源代码中：
   c) 一维数组类型的变量可以作为函数参数（但函数不会返回一维数组类型的值）。
   d) 可以出现高维数组类型的变量（但高维数组类型的变量不会作为函数的参数或返回类值）。

此外，实验三还会考察你的程序输出的中间代码的执行效率，因此你需要考虑如何优化中间代码的生成。在你的程序可以生成正确的中间代码（“正确”是指该中间代码在虚拟机小程序上运行结果正确）的前提下，如果该中间代码在我们的测试用例上能比50%甚至80%的同学的中间代码效率都高，你将获得额外奖励。

4.1.2 输入格式

你的程序的输入是一个包含C--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。

4.1.3 输出格式

实验三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。

4.1.4 测试环境

你的程序将在如下环境中被编译并运行（同实验一）：

1) GNU Linux Release: Ubuntu 20.04, kernel version 5.13.0-44-generic;

2) GCC version 7.5.0;

3) GNU Flex version 2.6.4;

4) GNU Bison version 3.5.1。

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，实验三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。

4.1.5 提交要求

实验三要求提交如下内容（同实验一）：

1) Flex、Bison以及C语言的可被正确编译运行的源代码程序。

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）。

4.1.6 样例（必做内容）

实验三的样例包括**必做内容样例**与**选做要求样例**两部分，分别对应于实验要求中的必做内容和选做要求。请仔细阅读样例，以加深对实验要求以及输出格式要求的理解。这节列举必做内容样例。

样例1：

输入：

```
1  int main()
2  {
3    int n;
4    n = read();
5    if (n > 0) write(1);
```

```
6    else if (n < 0) write (-1);
7    else write(0);
8    return 0;
9  }
```

输出:

这段程序读入一个整数n，然后计算并输出符号函数sgn(n)。它所对应的中间代码可以是这样的:

```
 1  FUNCTION main :
 2  READ t1
 3  v1 := t1
 4  t2 := #0
 5  IF v1 > t2 GOTO label1
 6  GOTO label2
 7  LABEL label1 :
 8  t3 := #1
 9  WRITE t3
10  GOTO label3
11  LABEL label2 :
12  t4 := #0
13  IF v1 < t4 GOTO label4
14  GOTO label5
15  LABEL label4 :
16  t5 := #1
17  t6 := #0 - t5
18  WRITE t6
19  GOTO label6
20  LABEL label5 :
21  t7 := #0
22  WRITE t7
23  LABEL label6 :
24  LABEL label3 :
25  t8 := #0
26  RETURN t8
```

需要注意的是，虽然样例输出中使用的变量遵循着字母后跟一个数字（如t1、v1等）的方式，标号也遵循着label后跟一个数字的方式，但这并不是强制要求的。也就是说，你的程序输出完全可以使用其它符合变量名定义的方式而不会影响虚拟机小程序的运行。

可以发现，这段中间代码中存在很多可以优化的地方。首先，0这个常数我们将其赋给了t2、t4、t7、t8这四个临时变量，实际上赋值一次就可以了。其次，对于t6的赋值我们可以直接写成t6 := #–1而不必多进行一次减法运算。另外，程序中的标号也有些冗余。如果你的程序足够“聪明”，可能会将上述中间代码优化成这样:

```
 1  FUNCTION main :
 2  READ t1
 3  v1 := t1
 4  t2 := #0
 5  IF v1 > t2 GOTO label1
 6  IF v1 < t2 GOTO label2
 7  WRITE t2
 8  GOTO label3
 9  LABEL label1 :
10  t3 := #1
11  WRITE t3
12  GOTO label3
```

```
13  LABEL label2 :
14  t6 := #-1
15  WRITE t6
16  LABEL label3 :
17  RETURN t2
```

样例2:

输入:

```
 1  int fact(int n)
 2  {
 3    if (n == 1)
 4      return n;
 5    else
 6      return (n * fact(n - 1));
 7  }
 8  int main()
 9  {
10    int m, result;
11    m = read();
12    if (m > 1)
13      result = fact(m);
14    else
15      result = 1;
16    write(result);
17    return 0;
18  }
```

输出:

这是一个读入m并输出m的阶乘的小程序，其对应的中间代码可以是:

```
 1  FUNCTION fact :
 2  PARAM v1
 3  IF v1 == #1 GOTO label1
 4  GOTO label2
 5  LABEL label1 :
 6  RETURN v1
 7  LABEL label2 :
 8  t1 := v1 - #1
 9  ARG t1
10  t2 := CALL fact
11  t3 := v1 * t2
12  RETURN t3
13
14  FUNCTION main :
15  READ t4
16  v2 := t4
17  IF v2 > #1 GOTO label3
18  GOTO label4
19  LABEL label3 :
20  ARG v2
21  t5 := CALL fact
22  v3 := t5
23  GOTO label5
24  LABEL label4 :
25  v3 := #1
26  LABEL label5 :
27  WRITE v3
28  RETURN #0
```

这个样例主要展示如何处理包含多个函数以及函数调用的输入文件。

4.1.7 样例（选做要求）

这节列举选做要求样例。

样例1:

输入:

```
1  struct Operands
2  {
3    int o1;
4    int o2;
5  };
6
7  int add(struct Operands temp)
8  {
9    return (temp.o1 + temp.o2);
10 }
11
12 int main()
13 {
14   int n;
15   struct Operands op;
16   op.o1 = 1;
17   op.o2 = 2;
18   n = add(op);
19   write(n);
20   return 0;
21 }
```

输出:

样例输入中出现了结构体类型的变量，以及这样的变量作为函数参数的用法。如果你的程序需要完成要求3.1，样例输入对应的中间代码可以是:

```
1  FUNCTION add :
2  PARAM v1
3  t2 := *v1
4  t7 := v1 + #4
5  t3 := *t7
6  t1 := t2 + t3
7  RETURN t1
8  FUNCTION main :
9  DEC v3 8
10 t9 := &v3
11 *t9 := #1
12 t12 := &v3 + #4
13 *t12 := #2
14 ARG &v3
15 t14 := CALL add
16 v2 := t14
17 WRITE v2
18 RETURN #0
```

如果你的程序不需要完成要求3.1，将不能翻译该样例输入，你的程序可以给出如下的提示信息:

```
Cannot translate: Code contains variables or parameters of structure type.
```

样例2:

输入:

```
1  int add(int temp[2])
```

```
 2  {
 3    return (temp[0] + temp[1]);
 4  }
 5
 6  int main()
 7  {
 8    int op[2];
 9    int r[1][2];
10    int i = 0, j = 0;
11    while (i < 2)
12    {
13      while (j < 2)
14      {
15        op[j] = i + j;
16        j = j + 1;
17      }
18      r[0][i] = add(op);
19      write(r[0][i]);
20      i = i + 1;
21      j = 0;
22    }
23    return 0;
24  }
```

输出:

样例输入中出现了高维数组类型的变量，以及一维数组类型的变量作为函数参数的用法。

如果你的程序需要完成要求3.2，样例输入对应的中间代码可以是:

```
 1  FUNCTION add :
 2  PARAM v1
 3  t2 := *v1
 4  t11 := v1 + #4
 5  t3 := *t11
 6  t1 := t2 + t3
 7  RETURN t1
 8  FUNCTION main :
 9  DEC v2 8
10  DEC v3 8
11  v4 := #0
12  v5 := #0
13  LABEL label1 :
14  IF v4 < #2 GOTO label2
15  GOTO label3
16  LABEL label2 :
17  LABEL label4 :
18  IF v5 < #2 GOTO label5
19  GOTO label6
20  LABEL label5 :
21  t18 := v5 * #4
22  t19 := &v2 + t18
23  t20 := v4 + v5
24  *t19 := t20
25  v5 := v5 + #1
26  GOTO label4
27  LABEL label6 :
28  t31 := v4 * #4
29  t32 := &v3 + t31
30  ARG &v2
31  t33 := CALL add
32  *t32 := t33
33  t41 := v4 * #4
34  t42 := &v3 + t41
35  t35 := *t42
36  WRITE t35
37  v4 := v4 + #1
```

```
38  v5 := #0
39  GOTO label1
40  LABEL label3 :
41  RETURN #0
```

如果你的程序不需要完成要求3.2，将不能翻译该样例输入，你的程序可以给出如下的提示信息：

```
Cannot translate: Code contains variables of multi-dimensional array type or
   parameters of array type.
```