

合肥工业大学

系统软件综合设计报告

编译原理分册

设计题目:	局部优化程序的实现
学生姓名:	文 华
学 号:	2017218007
专业班级:	物联网工程 17-2 班
指导教师:	李宏芒、唐益明
完成日期:	2020 年 8 月 30 日

目 录

1	设计概览	1
1.1	课程设计目的和要求	1
1.2	课程设计任务	1
2	开发环境	2
3	相关原理及算法	3
3.1	基本原理	3
3.1.1	基本块的相关概念	3
3.1.2	基本块的 DAG 表示	4
3.1.3	中间代码类型	5
3.2	有关算法	6
4	设计的输入和输出形式	9
4.1	程序输入内容与格式	9
4.2	程序输出内容与格式	9
5	程序运行的主要界面和结果截图	12
5.1	程序运行界面	12
5.2	程序运行结果	14
6	总结和感想体会	17
	参考文献	18
	附录	18

1 设计概览

1.1 课程设计目的和要求

①目的

《编译原理》是计算机专业的一门重要课程，其中包含大量软件设计思想。大家通过课程设计，实现一些重要的算法或一个完整编译程序模型能够进一步加深理解和掌握所学知识，对提高自己的软件设计水平具有十分重要的意义^[1]。

②要求

按照《编译原理课程设计指导书（含参考选题）》（2016版）的有关要求完成算法设计、代码编写与调试以及课设报告的撰写。

1.2 课程设计任务

题目：局部优化程序的实现

设计内容及要求：根据基本块转换成 DAG 的算法，实现：对于任意输入的一个基本块（四元式程序），将其转换为 DAG；然后按照 DAG 节点构造顺序，重构基本块四元式代码。以 P.283 例 10.4 为输入，完成并输出局部优化。

2 开发环境

硬件：Dell G3579 笔记本电脑；

软件：Visual Studio Enterprise 2019、gcc、Notepad++。

3 相关原理及算法

3.1 基本原理

代码优化是指编译程序为了生成高质量的目标程序而做的各种加工和处理。而高质量的目标程序是指对同一源程序在运行时所占的内存空间较小，且在同一台机器上运行时间也较短的目标程序。代码优化并不能保证得到的目标代码是最优的，而只能是相对较优的。优化的原则是在编译阶段能计算的量绝不留到运行时刻去做，能在外层循环中计算的量绝不放到内层去做，能够共用存储单元(寄存器)的尽量共用。

优化可在编译的各个阶段进行，最主要的时机是在语法、语义分析生成中间代码之后，在中间代码上进行。这一类优化不依赖于具体的计算机，而取决于语言的结构。代码优化有以下三种类型：

局部优化：是指在只有一个入口和一个出口，且语句为顺序执行的程序段上所进行的优化。这种程序段，称为基本块。将基本块作为优化要考虑的主要范围为优化决策提供了基础，在一般情况下，将会产生更高质量的代码。

循环优化：是对循环语句所生成的中间代码序列所进行的优化处理，这类优化包括外提不变表达式、强度削弱和删除归纳变量。

全局优化：是在非线性程序段上的优化。因为程序段是非线性的，因此需要分析程序的控制流和数据流，处理比较复杂。

其中，局部优化是指局部范围内的优化。这个“局部范围”是指基本块，即只有一个入口和一个出口且语句为顺序执行的程序段。局部优化就是把程序划分为若干个“基本块”，优化的工作分别在每个基本块内进行。下面介绍有关概念。

3.1.1 基本块的相关概念

所谓基本块，是指程序中一组顺序执行的语句序列，其中只有一个入口和一个出口。执行时只能从其入口进入，从其出口退出。基本块内的语句要么都被执行，要么都不

执行。入口语句的定义如下：

- ①程序的第一个语句；或者，
- ②条件转移语句或无条件转移语句的转移目标语句；
- ③紧跟在条件转移语句后面的语句。

有了入口语句的概念之后，就可以给出划分中间代码（四元式程序）为基本块的算法，其步骤如下：

①求出四元式程序中各个基本块的入口语句。

②对每一入口语句，构造其所属的基本块。它是由该入口语句到下一入口语句（不包括下一入口语句），或到一转移语句（包括该转移语句），或到一停语句（包括该停语句）之间的语句序列组成的。

③凡未被纳入某一基本块的语句、都是程序中控制流程无法到达的语句，因而也是不会被执行到的语句，可以把它们删除。

在一个基本块内通常可以实行下面的优化：

①删除公共子表达式：如果一个表达式 E 已经计算过了，并且从先前的计算到现在 E 中所有变量的值都没有发生变化，那么 E 的这次出现就成为了公共子表达式。对于这种表达式，没有必要花时间再对它进行计算，只需直接用前面计算过的表达式结果代替 E 即可。

②删除无用赋值：删除无用赋值或删除无用代码是指在程序中有些变量的赋值对程序运行结果没有任何作用，对这些变量赋值的代码可以删除。

③合并已知量：也称为常数合并，是指将能在编译时计算出值的表达式用其相应的值替代。如果在编译时，编译程序能知道一个表达式的所有操作数的值，则此表达式就可由其计算出的值替代。

④临时变量改名：总可以将一个基本块变换成等价的另一个基本块，使其中定义临时变量的语句改成定义新的临时变量。

⑤交换语句的位置：在交换语句位置不影响基本块值的情况下，有时通过改变其次序可产生更高效的代码。

⑥代数变换：对基本块中求值的表达式，用代数上等价的形式变换，以期使复杂运算变成简单运算。

3.1.2 基本块的 DAG 表示

一个基本块的 DAG 是一种其结点带有下列标记或附加信息的 DAG。

- 1.图的叶结点(没有后继的结点)以一标识符(变量名)或常数作为标记,表示该结点代表该变量或常数的值。如果叶结点用来代表某变量 A 的地址,则用 `addr(A)`作为该结点的标记。通常把叶结点上作为标记的标识符加上下标 0,以表示它是该变量的初值。
- 2.图的内部结点(有后继的结点)以一运算符作为标记,表示该结点代表应用该运算符对其后继结点所代表的值进行运算的结果。
- 3.图中各个结点上可能附加一个或多个标识符,表示这些变量具有该结点所代表的值。

3.1.3 中间代码类型

一般地,中间代码有 4 中类型,具体示例与说明如图 3.1.3.1 所示。



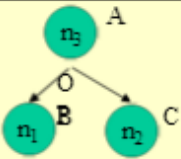
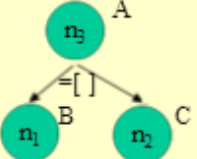
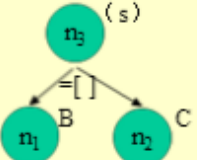
类型	四元式	DAG结点	说明
0型	$(=, B, A)$		把B赋给变量A,即A、B具有同样的值。无条件转向语句也可这样表示。
1型	(OP, B, A)		OP是单目运算符,与0型类似
2型	(OP, B, C, A)		B、C为两个叶结点,OP为运算符,运算结果赋给内部结点右边的标记A
3型	$(=[], B, C, A)$		B是数组,C是数组下标地址,=[]表示对数组B中下标变量地址为C的元素进行运算,结果赋给变量A
4型	$(JROP, B, C, (s))$		运算的结果将转向内部结点右边标出的语句(s)

图 3.1.3.1 四元式示例与说明

接下来介绍与本设计有关的算法。

3.2 有关算法

对于仅含 0、1、2 型中间代码的基本块的 DAG 构造算法描述如下 [2]。

开始，DAG 为空。

对基本块中的每一条中间代码式，依次执行以下步骤。

1. 如果 NODE (B) 无定义，则构造一标记为 B 的叶结点并定义 NODE (B) 为这个结点；

如果当前四元式是 0 型，则记 NODE (B) 的值为 n，转 4。

如果当前四元式是 1 型，则转 2. (1)。

如果当前四元式是 2 型，则：(i) 如果 NODE (C) 无定义，则构造一标记为 C 的叶结点并定义 NODE (C) 为这个结点，(ii) 转 2. (2)。

2.

(1) 如果 NODE (B) 是标记为常数的叶结点，则转 2. (3)，否则转 3. (1)。

(2) 如果 NODE (B) 和 NODE (C) 都是标记为常数的叶结点，则转 2. (4)，否则转 3. (2)。

(3) 执行 op B (即合并已知量)，令得到的新常数为 P。如果 NODE (B) 是处理当前四元式时新构造出来的结点，则删除它。如果 NODE (P) 无定义，则构造一用 P 做标记的叶结点 n。置 NODE (P) = n，转 4。

(4) 执行 B op C (即合并已知量)，令得到的新常数为 P。如果 NODE (B) 或 NODE (C) 是处理当前四元式时新构造出来的结点，则删除它。如果 NODE (P) 无定义，则构造一用 P 做标记的叶结点 n。置 NODE (P) = n，转 4。

3.

(1) 检查 DAG 中是否已有一结点，其唯一后继为 NODE (B)，且标记为 op (即找公共子表达式)。如果没有，则构造该结点 n，否则就把已有的结点作为它的结点并设该结点为 n，转 4。

(2) 检查 DAG 中是否已有一结点，其左后继为 NODE (B)，右后继为 NODE (C)，且标记为 op (即找公共子表达式)。如果没有，则构造该结点 n，否则就把已有的结点作为它的结点并设该结点为 n。转 4。

4.

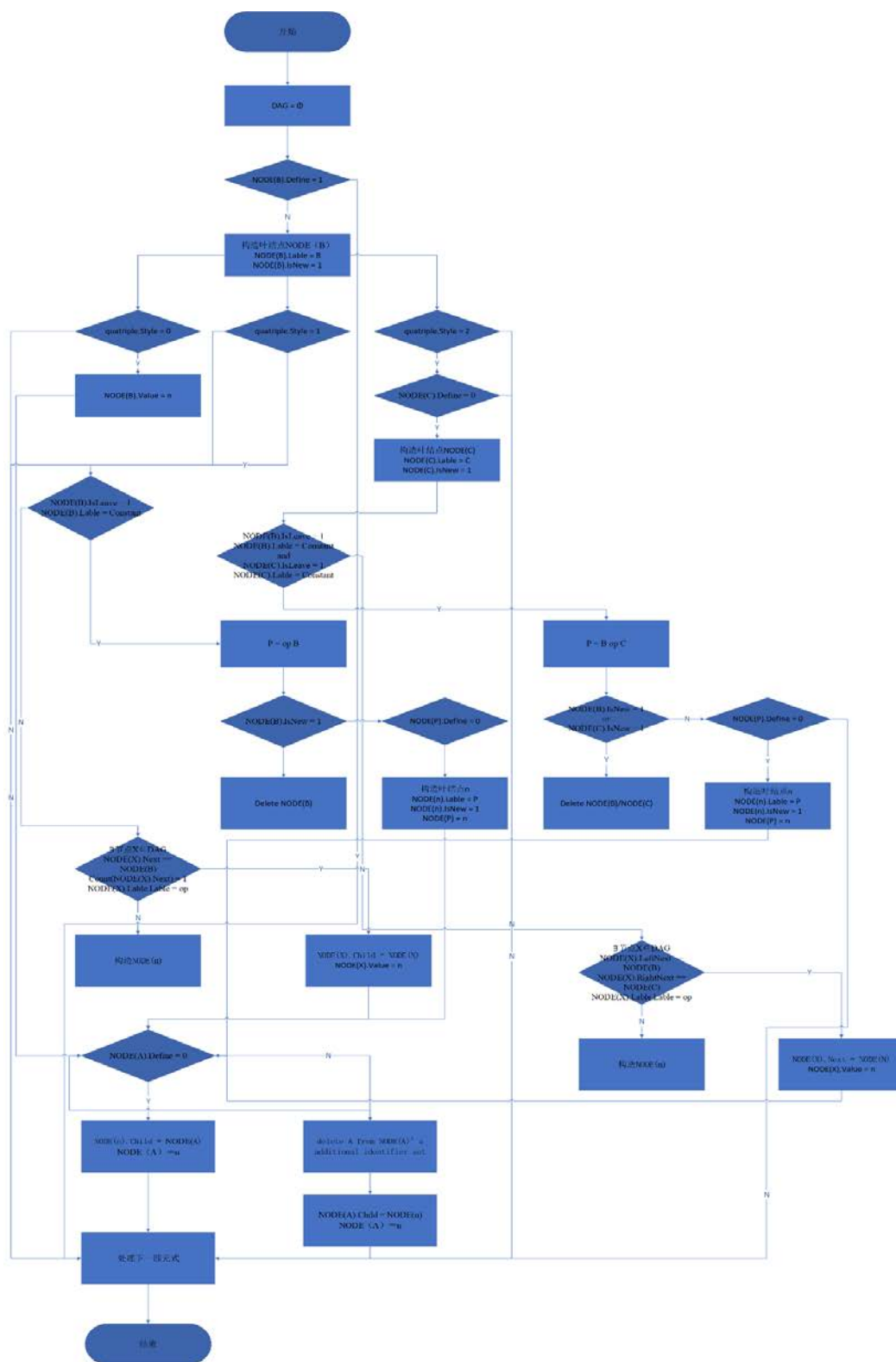
如果 NODE (A) 无定义，则把 A 附加在结点 n 上并令 NODE (A) = n；否则先把

A 从 NODE (A) 结点上的附加标识符集中删除 (注意, 如果 NODE (A) 是叶结点, 则其标记 A 不删除), 把 A 附加到新结点 n 上并令 NODE (A) = n。转处理下一四元式。

上述构造 DAG 算法的流程图如图 3.2.1 所示。

据图 3.2.1 所示流程图所得的伪代码描述如下所示。

```
DAG = NULL;
for exp in blocks:
    1. If node (b) is not defined, a leaf node marked b is constructed and node (b) is defined as
    this node;
        If the current quaternion is type 0, mark the value of node (b) as N and turn to 4.
        If the current quaternion is type 1, turn to 2. (1).
        If the current quaternion is type 2, then: (I) if node (c) has no definition, then construct a
        leaf node marked as C and define node (c) as this node, (II) turn to 2. (2).
    2.
        (1) If node (b) is a leaf node marked as a constant, turn to 2. (3), otherwise turn to 3. (1).
        (2) If node (b) and node (c) are leaf nodes marked as constants, turn to 2. (4), otherwise
        turn to 3. (2).
        (3) Execute OP B (i.e. merge known quantities) and make the new constant P. If node (b) is
        a newly constructed node when processing the current quaternion, delete it. If node (P) is
        undefined, a leaf node n marked with P is constructed. Set node (P) = n and turn to 4.
        (4) Execute B OP C (i.e. merge known quantities) so that the new constant obtained is p. If
        node (b) or node (c) is a newly constructed node when processing the current quaternion, delete
        it. If node (P) is undefined, a leaf node n marked with P is constructed. Set node (P) = n and turn
        to 4.
    3. (1) Check whether there is a node in DAG whose unique successor is node (b) and
    marked as op (i.e. find common subexpression). If not, the node n is constructed. Otherwise, the
    existing node is regarded as its node and the node is set as N, then turn to 4.
        (2) Check whether there is a node in DAG, whose left successor is node (b), right
        successor is node (c), and marked as op (that is to find common subexpression). If not, the node
        n is constructed, otherwise the existing node is regarded as its node and the node is set as n. Turn
        to 4.
    4. If node (a) has no definition, a is attached to node n and node (a) = n; otherwise, a is
    deleted from the set of additional identifiers on node (a) (note that if node (a) is a leaf node, its
    mark a is not deleted), and a is attached to the new node n and node (a) = n. Turn to the next
    quaternion.
endfor
return DAG
```

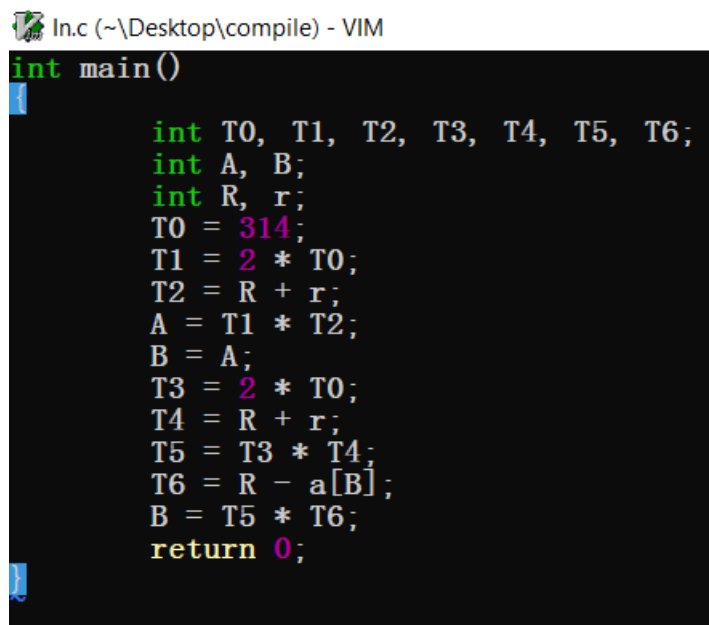


4 设计的输入和输出形式

本设计所构建局部优化程序不单是将四元式代码优化后输出，而是可以对完整的 C 语言文件进行词法分析、语法分析、语义分析、中间代码生成与中间代码优化，故程序支持对 C 语言程序的直接读入与处理。

4.1 程序输入内容与格式

本设计所构建局部优化程序的输入内容为一 TXT 格式的文本文件，文件里包含了一段待处理的 C 语言程序。输入文件的形式如图 4.1.1 所示。



```
int main()  
{  
    int T0, T1, T2, T3, T4, T5, T6;  
    int A, B;  
    int R, r;  
    T0 = 314;  
    T1 = 2 * T0;  
    T2 = R + r;  
    A = T1 * T2;  
    B = A;  
    T3 = 2 * T0;  
    T4 = R + r;  
    T5 = T3 * T4;  
    T6 = R - a[B];  
    B = T5 * T6;  
    return 0;  
}
```

图 3.1.3.1 输入文件的形式

值得一提的是，本设计所构建的程序支持 C 语言的 14 个关键词及有关语句：int、char、void、if、else、switch、case、default、scanf、printf、main 和 return。


4.2 程序输出内容与格式

本设计所构建局部优化程序的输出内容分两部分：1) 输出到控制台的信息，包括对程序每一行语句的判断、语法错误提示、程序所含变量和优化后的四元式代码；2) 写入

到 TXT 格式的文本文件包括程序所含变量与函数以及优化后的四元式代码。输出到控制台的内容如图 4.2.1 所示，写入文本的内容如图 4.2.2 所示（所用测试用例为 P.283 例 10.4 的四元式代码）。

```
C:\Users\hadoop001\Desktop\compile>main.exe
打开文本成功
This is a variable declaration statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
[Line 14] Error: Identifier not define.
This is an assignment statement.
This is an assignment statement.
This is a return statement.
[Line 16] Error: Semicolon lost.
This is a statement list.
int main()
int T0
int T1
int T2
int T3
int T4
int T5
int T6
int A
int B
int R
int r
T0 = 314
#t0 = 2 * T0
T1 = #t0
#t1 = R + r
T2 = #t1
#t2 = T1 * T2
A = #t2
B = A
#t3 = 2 * T0
T3 = #t3
#t4 = R + r
T4 = #t4
#t5 = T3 * T4
T5 = #t5
#t6 = a[B]
#t7 = R - #t6
T6 = #t7
#t8 = T5 * T6
B = #t8
return
```

图 3.1.3.1 输出到控制台的内容

 betterInfixes.txt (~\Desktop\compile) - VIM

```
int main()
int T0
int T1
int T2
int T3
int T4
int T5
int T6
int A
int B
int R
int r
#t1 = R + r
T0 = 314
#t0 = 2 * T0
#t2 = #t0 * #t1
#t6 = a[#t2]
#t7 = R - #t6
#t8 = #t2 * #t7
return
```

图 3.1.3.2 写入文本的内容

5 程序运行的主要界面和结果截图

5.1 程序运行界面

本设计所构建局部优化程序的程序运行界面如图 5.1.1 至图 5.1.3 所示（所用测试样例在附录中给出）。

```
C:\Users\hadooop001\Desktop>compile>main.exe  
打开文件成功  
This is a variable declaration statement.  
This is an assignment statement.  
This is a return statement.  
This is a statement list.  
This is a definition of function with return.  
This is a variable declaration statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
This is an assignment statement.  
[Line 25] Error: Identifier not define.  
This is an assignment statement.  
This is a return statement.  
[Line 26] Error: Semicolon lost.  
This is a statement list.  
  
int compute()  
{  
    param int x1  
    param int x2  
    int y1  
    #t0 = x1 + x2  
    y1 = #t0  
    return y1  
}  
int main()  
{  
    int T0  
    int T1  
    int T2  
    int T3  
    int T4  
    int T5  
    int T6  
    int A  
    int B  
    int R  
    int r  
    R = 2  
    r = 3  
    T0 = 5  
    #t1 = 2 * T0  
    T1 = #t1  
    #t2 = R + r  
    T2 = #t2  
    #t3 = T1 * T2  
    A = #t3  
    B = A  
    #t4 = 2 * T0  
    T3 = #t4  
    #t5 = R + r  
    T4 = #t5  
    #t6 = T3 * T4  
    T5 = #t6  
    #t7 = R - r  
    T6 = #t7  
    #t8 = T5 * T6  
    B = #t8  
    push T1  
    push T2  
    call compute  
    #t9 = #RET  
    y = #t9  
    return
```

图 3.1.3.1 程序运行界面-1

```

C:\Users\hadoop001\Desktop\compile>main.exe
打开文本成功
This is an assignment statement.
This is a variable declaration statement.
This is an assignment statement.
This is a statement list.
[Line 10] Error: Left brace lost.
This is a condition statement.
This is a statement list.
This is an if statement.
This is a return statement.
This is a statement list.
This is a definition of function with return.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is a variable declaration statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
This is an assignment statement.
[Line 44] Error: Right parenthesis lost.
[Line 45] Error: Semicolon lost.
This is a statement list.
int select()
param int score
int level
#t0 = -1
level = #t0
#t1 = score < 60
goto LABEL0, if #t1 == 0
level = 0
jmp LABEL1
LABEL0
#t2 = score / 10
LABEL3
LABEL1
return level
int main()
int a1
a1 = 59
int a2
a2 = 60
int a3
a3 = 73
int a4
a4 = 85
int a5
a5 = 94
int a6
a6 = 100
#t3 = 1 + a1
a1 = #t3
#t4 = a1 + a2
a2 = #t4
#t5 = a3 * a2
a3 = #t5
#t6 = a4 - 20
a4 = #t6
#t7 = a6 - a4
a5 = #t7
a6 = a5

```

图 3.1.3.2 程序运行界面-2


```
C:\Windows\system32\cmd.exe

C:\Users\hadoop001\Desktop\compile>main.exe
打开文本成功
This is an assignment statement.
This is an assignment statement.
This is a variable declaration statement.
This is an assignment statement.
This is a statement list.
[Line 6] Error: Right brace lost.
int main()
char id_1
id_1 = 87
char id_2
id_2 = 87
#t0 = 68 + id_2
#t1 = #t0 + id_1
id_1 = #t1
```

图 3.1.3.3 程序运行界面-3


5.2 程序运行结果

本设计所构建局部优化程序的程序运行结果如图 5.2.1 至图 5.2.3 所示。

 betterInfixes.txt (~\Desktop\compile) - VIM


```
int compute()
param int x1
param int x2
int y1
y1 = x1 + x2
return y1
int main()
int T0
int T1
int T2
int T3
int T4
int T5
int T6
int A
int B
int R
int r
r = 3
R = 2
#t7 = R - r
T2 = R + r
T0 = 5
T1 = 2 * T0
#t3 = T1 * T2
#t8 = #t3 * #t7
push T1
push T2
call compute
#t9 = #RET
return
~
```

图 3.1.3.1 程序运行结果-1

 betterInfixes.txt (~\Desktop\compile) - VIM

```
int select()
param int score
int level
level = -1
#t1 = score < 60
goto LABEL0, if #t1 == 0
level = 0
jmp LABEL1
LABEL0
#t2 = score / 10
LABEL3
LABEL1
return level
int main()
int a1
a1 = 59
int a2
a2 = 60
int a3
a3 = 73
int a4
a4 = 85
int a5
a5 = 94
int a6
a10 = a1
a20 = a2
a30 = a3
a40 = a4
a4 = a40 - 20
#t8 = 100
a5 = #t8 - a4
a1 = 1 + a10
a2 = a1 + a20
a3 = a30 * a2
~
```

图 3.1.3.2 程序运行结果-2

 betterInfixes.txt (~\Desktop\compile) - VIM

```
int main()
char id_1
id_1 = 87
char id_2
id_10 = id_1
id_2 = 87
#t0 = 68 + id_2
id_1 = #t0 + id_10
~
```

图 3.1.3.3 程序运行结果-3

6 总结和感想体会

本次设计让我加深了对基本块优化和编译器的理解：对于一个给定的程序，我们可以把它划分为一系列的基本块。在各个基本块范围内，分别进行优化。最终的优化结果不仅减少了冗余代码，还降低了对内存的消耗与运行时间，时空都得到了优化。

所谓代码优化是指对程序代码进行等价（指不改变程序的运行结果）变换。程序代码可以是中间代码（如四元式代码），也可以是目标代码。等价的含义是使得变换后的代码运行结果与变换前代码运行结果相同。优化的含义是最终生成的目标代码短（运行时间更短、占用空间更小），时空效率优化。原则上，优化可以在编译的各个阶段进行，但最主要的一类是对中间代码进行优化，这类优化不依赖于具体的计算机^[3]。简而言之，在不改变程序运行效果的前提下，对被编译的程序进行等价变换，使之能生成更加高效的目标代码^[4]。

参考文献

- [1]李宏芒. 编译原理课程设计指导书（含参考选题）（2016版）[M]. 合肥：合肥工业大学，2016.
- [2]陈火旺，刘春林，谭庆平，赵克佳，刘越. 程序设计语言编译原理[M]. 北京：国防科技大学出版社，1999.
- [3]<https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81%E4%BC%98%E5%8C%96/571727?fr=aladdin>
- [4]陈英. 编译原理[M]. 北京：清华大学出版社，2009.

附录

测试文件：

```
//case 1
int compute(int x1, int x2)
{
    int y1;
    y1 = x1 + x2;
    return (y1);
}

int main()
{
    int T0, T1, T2, T3, T4, T5, T6;
    int A, B;
    int R, r;
    R = 2;
    r = 3;
    T0 = 5;
    T1 = 2 * T0;
    T2 = R + r;
    A = T1 * T2;
    B = A;
    T3 = 2 * T0;
    T4 = R + r;
    T5 = T3 * T4;
```

```

        T6 = R - r;
        B = T5 * T6;
        y = compute(T1, T2);
        return 0;
    }

//case 2
int cmp(int a, int b)
{
    int tmp = 0;
    if(a > b)
    {
        tmp = a;
    }
    else
    {
        tmp = b;
    }
    return (tmp);
}

int main()
{
    int a, b, bigger;
    scanf("%d%d", &a, &b);
    bigger = cmp(a, b);
    int x = bigger + 100;
    printf("%d\n", x);
    return (0);
}

//case 3
int select(int score)
{
    int level = -1;
    if(score < 60)
    {
        level = 0;
    }
    else
    {
        switch(score/10):
        {
            case(6):
                level = 1;
                break;

```

```

        case(7):
            level = 2;
            break;
        case(8):
            level = 3;
            break;
        case(9):
            level = 4;
            break;
        case(10):
            level = 5;
            break;
        default:
            level = -1;
            break;
    }
}
return (level);
}

//case 4
int main()
{
    int a1 = 59, a2 = 60, a3 = 73, a4 = 85, a5 = 94, a6 = 100;
    a1 = 1 + a1;
    a2 = a1 + a2;
    a3 = a3 * a2;
    a4 = a4 - 20;
    a5 = a6 - a4;
    a6 = a5;
    printf("a1 = %d, level = %d\n a2 = %d, level = %d\n a3 = %d, level = %d\n a4 = %d, level = %d\n a5 = %d, level = %d\n a6 = %d, level = %d\n", a1, select(a1), a2, select(a2), a3, select(a3), a4, select(a4), a5, select(a5), a6, select(a6));
    return (0);
}

int main()
{
    char id_1 = 'W';
    char id_2 = 'W';
    id_1 = 'D' + id_2 + id_1;
    int id_3 = 134;
    printf("%c--%c\n", id_1, id_2);
    return 0;
}

```

源代码:

```
#include "main.h"

struct DAGNode {
    int number;//节点的唯一标识, 与 lastNum 有关
    string content;//操作
    std::set<DAGNode*> parents;//父节点塞入
    DAGNode* leftChild;
    DAGNode* rightChild;
};

int lastNum = 0;//记录节点数
std::vector<DAGNode*> allNodes;
map<string, int> varNodeTable; // Node table
map<string, int> varsWithInitial; // Vars serving as initial nodes
std::set<string> crossingVars;
string currentFunc2;
std::vector<infixNotation> newInfixTable;
fstream betterInfixFile;
string newOutputBuff;

// Only codes with operand of ADD, SUB, MUL, DIV, NEG, ASSIGN, GETARR are manageable
inline bool isManageable(string ioperater) {
    if (ioperater == "ADD" || ioperater == "SUB" || ioperater == "MUL" ||
        ioperater == "DIV" || ioperater == "NEG" || ioperater == "ASSIGN" ||
        ioperater == "GETARR") {
        return true;
    }
    return false;
}

inline bool isMidNode(DAGNode *dagNode) {
    if (dagNode->leftChild != NULL || dagNode->rightChild != NULL) {
        return true;
    }
    return false;
}

// Variable name can't begin with space or number
inline bool isVarName(string operand) {
    if (operand[0] != ' ' && (operand[0] < '0' || operand[0] > '9')) {
        return true;
    }
    return false;
}
```

```

}

inline bool isAlpha(string content) {
    if (content[0] == '_' || (content[0] > 'a' && content[0] < 'z') || (content[0] > 'A' && content[0] < 'Z')) {
        return true;
    }
    return false;
}

DAGNode* findNodeWithNumber(int number) {
    for (int i = 0; i < allNodes.size(); ++i) {
        if (allNodes[i]->number == number) {
            return allNodes[i];
        }
    }
}

void insertNewInfix(string ioperator, string operand1, string operand2, string operand3) {
    infixNotation notation;
    notation.ioperator = ioperator;
    notation.operand1 = operand1;
    notation.operand2 = operand2;
    notation.operand3 = operand3;
    newInfixTable.push_back(notation);

    if (ioperator == "CONST") {
        newOutputBuff = newOutputBuff + "const " + operand2 + " " + operand3 + " " + operand1 + "\n";
    }
    else if (ioperator == "VAR") {
        if (operand1 != " ") { // Array
            newOutputBuff = newOutputBuff + operand2 + " " + operand3 + "[" + operand1 + "]" + "\n";
        }
        else { // General variable
            newOutputBuff = newOutputBuff + operand2 + " " + operand3 + "\n";
        }
    }
    else if (ioperator == "FUNC") {
        newOutputBuff = newOutputBuff + operand2 + " " + operand3 + "(" + "\n";
    }
    else if (ioperator == "PARAM") {

```



```

        newOutputBuff = newOutputBuff + "param " + operand2 + " " + operand3 + "\n";
    }
    else if (ioperator == "ADD") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " + " + operand2 + "\n";
    }
    else if (ioperator == "SUB") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " - " + operand2 + "\n";
    }
    else if (ioperator == "MUL") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " * " + operand2 + "\n";
    }
    else if (ioperator == "DIV") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " / " + operand2 + "\n";
    }
    else if (ioperator == "ASSIGN") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand2 + "\n";
    }
    else if (ioperator == "NEG") {
        newOutputBuff = newOutputBuff + operand3 + " = -" + operand2 + "\n";
    }
    else if (ioperator == "EQL") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " == " + operand2 + "\n";
    }
    else if (ioperator == "NEQ") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " != " + operand2 + "\n";
    }
    else if (ioperator == "LSS") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " < " + operand2 + "\n";
    }
    else if (ioperator == "LEQ") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " <= " + operand2 + "\n";
    }
    else if (ioperator == "GTR") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " > " + operand2 + "\n";
    }
}

```

```

    else if (ioperator == "GEQ") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + " >= " + operand2 + "\n";
    }
    else if (ioperator == "PUSH") {
        newOutputBuff = newOutputBuff + "push " + operand3 + "\n";
    }
    else if (ioperator == "CALL") {
        newOutputBuff = newOutputBuff + "call " + operand3 + "\n";
    }
    else if (ioperator == "RETURN") {
        newOutputBuff = newOutputBuff + "return " + operand3 + "\n";
    }
    else if (ioperator == "SETARR") {
        newOutputBuff = newOutputBuff + operand3 + "[" + operand2 + "] = " + operand1 + "\n";
    }
    else if (ioperator == "GETARR") {
        newOutputBuff = newOutputBuff + operand3 + " = " + operand1 + "[" + operand2 + "]" + "\n";
    }
    else if (ioperator == "JMP") {
        newOutputBuff = newOutputBuff + "jmp " + operand3 + "\n";
    }
    else if (ioperator == "BEQ") {
        newOutputBuff = newOutputBuff + "goto " + operand3 + ", if " + operand1 + " == " + operand2 + "\n";
    }
    else if (ioperator == "BNE") {
        newOutputBuff = newOutputBuff + "goto " + operand3 + ", if " + operand1 + " != " + operand2 + "\n";
    }
    else if (ioperator == "SCANF") {
        newOutputBuff = newOutputBuff + "scan " + operand3 + "\n";
    }
    else if (ioperator == "PRINTF") {
        if (operand2 != " " && operand3 != " ") {
            newOutputBuff = newOutputBuff + "print " + operand2 + ", " + operand3 + "\n";
        }
        else if (operand2 != " " && operand3 == " ") {
            newOutputBuff = newOutputBuff + "print " + operand2 + "\n";
        }
    }

```

```

        else if (operand2 == " " && operand3 != " ") {
            newOutputBuff = newOutputBuff + "print " + operand3 + "\n";
        }
        else {
            cout << "Can't print nothing!" << endl;
        } // Do nothing
    }
    else if (ioperator == "LABEL") {
        newOutputBuff = newOutputBuff + operand3 + "\n";
    }
}

// Insert operands to varsInBlock
void insertOperands(std::set<string>& varsInBlock, infixNotation notation) {
    if (isVarName(notation.operand1) &&
        varsInBlock.count(notation.operand1) == 0) { // Var not stored, insert it
        varsInBlock.insert(notation.operand1);
    }
    if (isVarName(notation.operand2) &&
        varsInBlock.count(notation.operand2) == 0) { // Var not stored, insert it
        varsInBlock.insert(notation.operand2);
    }
    if (isVarName(notation.operand3) &&
        varsInBlock.count(notation.operand3) == 0) { // Var not stored, insert it
        varsInBlock.insert(notation.operand3);
    }
}

// Store elements from vars in block to varExistenceCount
//varExistenceCount <var,cout of existence in varsInBlock>
void checkVarExistenceCount(map<string, int>& varExistenceCount, std::set<string>
& varsInBlock) {
    for (std::set<string>::iterator it = varsInBlock.begin();
        it != varsInBlock.end(); ++it) {
        map<string, int>::iterator varIt;
        varIt = varExistenceCount.find(*it); //传入的参数是要查找的 key
        if (varIt != varExistenceCount.end()) { // Variable already exists
            varIt->second++;
        }
        else { // Variable not exists, insert one
            varExistenceCount.insert(std::pair<string, int>(*it, 1));
        }
    }
}

```

```

// Split basic blocks & get crossing variables
//分割基本块
void splitBlocks() {
    map<string, int> varExistenceCount;
    std::set<string> varsInBlock;
    bool isInBlock = false;
    // Check all infix notations, get existence count
    for (int i = 0; i < infixTable.size(); ++i) {
        if (isManageable(infixTable[i].ioperator)) { // Infix code is manageable
            if (!isInBlock) { // This line of code is following an other kind of
basic block
                isInBlock = true;
                checkVarExistenceCount(varExistenceCount, varsInBlock);
                // Empty vars in block
                varsInBlock.clear();
            }
        }
        else { // Infix code is not manageable
            if (isInBlock) { // This line of code is following an other kind of b
asic block
                isInBlock = false;
                checkVarExistenceCount(varExistenceCount, varsInBlock);
                // Empty vars in block
                varsInBlock.clear();
            }
        }
        //insert infixtable node i into varsInBlock
        if (infixTable[i].ioperator != "CONST" && infixTable[i].ioperator != "VAR
" &&
            infixTable[i].ioperator != "FUNC" && infixTable[i].ioperator != "PARA
M") {
            // Declarations don't take in count
            insertOperands(varsInBlock, infixTable[i]);
        }
    }

    // Manage the rest codes  最后一个块没有判断
    checkVarExistenceCount(varExistenceCount, varsInBlock);

    // Store element with existence count over 1 to crossingVars
    for (map<string, int>::iterator it = varExistenceCount.begin(); it != varExis
tenceCount.end();
        ++it) {
        if (it->second > 1) { // Var cross basic blocks
            crossingVars.insert(it->first);
        }
    }
}

```

```

    }
}

// Find & set node table element, return operand number(lastNum+1)
//根据 operand 查找 varNodeTable, 将找到后返回 operand Num
//未找到 varNodeTable 中添加 operand,*lastNum,并生成 DAGNode 添加到 allNode 中
int setUpOperand(string operand, int *lastNum) {
    int operandNum;
    map<string, int>::iterator tempIt;
    tempIt = varNodeTable.find(operand);
    if (tempIt != varNodeTable.end()) { // Operand already exists
        operandNum = tempIt->second;
    }
    else { // Not exists, insert new node
        operandNum = *lastNum;
        if (operand[0] >= '0' && operand[0] <= '9') { // Number
            // Insert left operand as number
            varNodeTable.insert(std::pair<string, int>(operand, *lastNum));
            DAGNode* node;
            node = new DAGNode;
            node->content = operand;
            node->number = *lastNum;
            node->leftChild = NULL;
            node->rightChild = NULL;
            allNodes.push_back(node); // Insert DAG node
            *lastNum = *lastNum + 1;
        }
        else { // Identifier
            // Insert operand as variable
            varsWithInitial.insert(std::pair<string, int>(operand, *lastNum)); // Record initial node index
            varNodeTable.insert(std::pair<string, int>(operand, *lastNum));
            DAGNode* node;
            node = new DAGNode;
            node->content = operand;
            node->number = *lastNum;
            node->leftChild = NULL;
            node->rightChild = NULL;
            allNodes.push_back(node); // Insert DAG node
            *lastNum = *lastNum + 1;
        }
    }
    return operandNum;
}

```

```

}

// Set up one DAG node & element in varNodeTable according to infix notation
void infixToDAGNode(infixNotation notation) {
    int leftOpNum = 0;
    int rightOpNum = 0;
    int resultNum = 0;
    string ioperator = notation.ioperator;
    string leftOperand = notation.operand1;
    string rightOperand = notation.operand2;
    string result = notation.operand3;

    if (ioperator != "ASSIGN") { // Infix notation isn't an assignment
        // Set up two operand
        leftOpNum = setUpOperand(leftOperand, &lastNum);
        rightOpNum = setUpOperand(rightOperand, &lastNum);

        // Set up result
        int i;
        // Find parent node
        for (i = 0; i < allNodes.size(); ++i) {
            if (allNodes[i]->content == ioperator &&
                allNodes[i]->leftChild->number == leftOpNum &&
                allNodes[i]->rightChild->number == rightOpNum) { // Find node with
h certain left & right child
                resultNum = allNodes[i]->number;
                break;
            }
        }
        if (i == allNodes.size()) { // Node doesn't exist, create a new one
            resultNum = lastNum;
            ++lastNum;
            DAGNode *leftNode = allNodes[leftOpNum];
            DAGNode *rightNode = allNodes[rightOpNum];

            // Parent node settings
            DAGNode* node;
            node = new DAGNode;
            node->number = resultNum;
            node->content = ioperator;
            node->parents.clear();
            node->leftChild = leftNode;
            node->rightChild = rightNode;
            allNodes.push_back(node); // Insert node
        }
    }
}

```

```

                                // Children node settings
        leftNode->parents.insert(node);
        rightNode->parents.insert(node);
    }
    else { // Node exists, result number is in resultNum
    }
}
else { // Infix notation is a assignment
    // Set up assigner
    rightOpNum = setUpOperand(rightOperand, &lastNum);

    // Set up assignee
    if (isMidNode(allNodes[rightOpNum])) { // Just get result number if right
Operand is middle node
        resultNum = rightOpNum;
    }
    else { // Create a parent node if rightOperand is not middle node
        resultNum = lastNum;
        ++lastNum;
        DAGNode *rightNode = allNodes[rightOpNum];

        // Parent node settings
        DAGNode* node;
        node = new DAGNode;
        node->number = resultNum;
        node->content = ioperator;
        node->parents.clear();
        node->leftChild = NULL;
        node->rightChild = rightNode;
        allNodes.push_back(node); // Insert node

                                // Children node settings
        rightNode->parents.insert(node);
    }
}

// Result table settings
map<string, int>::iterator tempIt;
tempIt = varNodeTable.find(result);
if (tempIt != varNodeTable.end()) { // Variable exists
    tempIt->second = resultNum;
}
else { // Not exist, insert new one
    varNodeTable.insert(std::pair<string, int>(result, resultNum));
}

```

```

    }
}

// Insert a mid DAGnode to calculation queue & cut off its connection with its children
//节点 node 记录到 calculationQueue, 把 Node 从子节点中的 parentNode 中删去, 并对左右子节点检测是否是 root, 进行相同操作
void setMidNode(DAGNode* node, std::vector<DAGNode*>& calculationQueue) {
    DAGNode* leftChild = node->leftChild;
    DAGNode* rightChild = node->rightChild;

    // Push node to queue
    calculationQueue.push_back(node);

    // Erase parent element in children's parents set
    std::set<DAGNode*>::iterator tempIt;
    if (leftChild != NULL) {
        tempIt = leftChild->parents.find(node);
        if (tempIt != leftChild->parents.end()) {
            leftChild->parents.erase(tempIt);
        }
    }
    if (rightChild != NULL) {
        tempIt = rightChild->parents.find(node);
        if (tempIt != rightChild->parents.end()) {
            rightChild->parents.erase(tempIt);
        }
    }

    // Set children middle nodes now without parents recursively
    if (leftChild != NULL) {
        if (isMidNode(leftChild) && leftChild->parents.size() == 0) {
            setMidNode(leftChild, calculationQueue);
        }
    }
    if (rightChild != NULL && leftChild != rightChild) {
        if (isMidNode(rightChild) && rightChild->parents.size() == 0) {
            setMidNode(rightChild, calculationQueue);
        }
    }
}

// Export optimized codes from DAG & varNodeTable
void exportCodesFromDAG() {
    std::vector<DAGNode*> rootNodes;

```



```

std::vector<DAGNode*> calculationQueue;
rootNodes.reserve(10000);
calculationQueue.reserve(10000);

// Find root nodes
for (int i = 0; i < allNodes.size(); ++i) {
    if (allNodes[i]->parents.size() == 0) { // No parents, must be root node
        rootNodes.push_back(allNodes[i]);
    }
}

// Get calculation queue
while (rootNodes.size() != 0) {
    setMidNode(rootNodes[0], calculationQueue);
    // Delete current root node from root node vector
    rootNodes.erase(rootNodes.begin());
}

// Output var with initials e.g. a0 = a
map<string, int>::iterator initIt;
for (initIt = varsWithInitial.begin(); initIt != varsWithInitial.end();
    ++initIt) {
    map<string, int>::iterator currentIt = varNodeTable.find(initIt->first);
    if (currentIt->second != initIt->second) { // Variable has been changed
        string varName = initIt->first;
        string newVarName = initIt->first + "0";

        // Insert a0 to static table
        // Skip all parameters
        int originalIdIndex;
        int insertIndex;
        for (insertIndex = lookUpStatic(currentFunc2.c_str());
            staticTable[insertIndex].cls == params; ++insertIndex);
        // Element settings
        originalIdIndex = lookUpStatic(currentFunc2.c_str(), varName.c_str());
;

        if (originalIdIndex == -1) { // Global identifier
            originalIdIndex = lookUp(varName.c_str());
        }
        insertStatic(currentFunc2, vars, staticTable[originalIdIndex].typ, newVarName.c_str(), 0, 1); // Insert table

        // Output a0 = a
        insertNewInfix("ASSIGN", " ", varName, newVarName);
    }
}

```

```

        // Change "a" node content
        for (int i = 0; i < allNodes.size(); ++i) {
            if (allNodes[i]->content == varName) {
                allNodes[i]->content = newVarName;
                break;
            }
        }
    }
}

// Output queue in reverse
for (int j = calculationQueue.size() - 1; j >= 0; --j) {
    int nodeNumber = calculationQueue[j]->number;

    // Collect all suited pairs of <name, number> from node table
    map<string, int>::iterator tempIt;
    std::set<string> varNodes; //记录 varTable 中与 calculation 中值一致的变量名
    for (tempIt = varNodeTable.begin(); tempIt != varNodeTable.end(); ++tempIt) {
        if (nodeNumber == tempIt->second) {
            varNodes.insert(tempIt->first);
        }
    }

    // Pick up pairs crossing basic blocks(if none, pick the first one) & set
    one as new content of result node
    std::set<string> varsToStay;
    std::set<string> varsToLeave;
    std::set<string>::iterator tempIt2;
    DAGNode* node = findNodeWithNumber(nodeNumber);
    DAGNode* leftChild = node->leftChild;
    DAGNode* rightChild = node->rightChild;

    for (tempIt2 = varNodes.begin(); tempIt2 != varNodes.end(); ++tempIt2) {
        // Set crossing variable to stay & others to leave
        std::set<string>::iterator crossingVarIt;
        crossingVarIt = crossingVars.find(*tempIt2);
        if (crossingVarIt != crossingVars.end()) { // Variable is a crossing
            one
                varsToStay.insert(*tempIt2);
            }
        else { // Not a crossing one
            varsToLeave.insert(*tempIt2);
        }
    }
}

```

```

    }
    if (varsToStay.size() == 0) { // No crossing blocks variables
        string varName;
        if (varsToLeave.size() != 0) { // There are still other choices, choose the first one to stay
            varName = *varNodes.begin();
            varsToStay.insert(varName);
            varsToLeave.erase(varsToLeave.find(varName));
        }
        else { // No variables corresponding with this node, create a new one & insert it to staticTable
            varName = createTempVar();
            varsToStay.insert(varName);
            insertStatic(currentFunc2, vars, ints, varName.c_str(), 0, 1); // Type is not important, int can be ok
        }

        // Insert new infix notation
        string operand1;
        string operand2;
        if (leftChild == NULL) {
            operand1 = " ";
        }
        else {
            operand1 = leftChild->content;
        }
        if (rightChild == NULL) {
            operand2 = " ";
        }
        else {
            operand2 = rightChild->content;
        }
        insertNewInfix(node->content, operand1, operand2, varName);
    }
    else { // Choose the first one as result of calculation & assign it to others
        infixNotation notation;
        string varName = *varsToStay.begin();

        // Insert notation of first variable as calculation
        string operand1;
        string operand2;
        if (leftChild == NULL) {
            operand1 = " ";
        }
    }

```

```

        else {
            operand1 = leftChild->content;
        }
        if (rightChild == NULL) {
            operand2 = " ";
        }
        else {
            operand2 = rightChild->content;
        }
        insertNewInfix(node->content, operand1, operand2, varName);

        // Insert notation of other variables as assignments
        std::set<string>::iterator tempIt;
        tempIt = varsToStay.begin();
        ++tempIt;
        for (; tempIt != varsToStay.end(); ++tempIt) {
            insertNewInfix("ASSIGN", " ", varName, *tempIt);
        }
    }

    // Set node content(first to-stay variable)
    node->content = *varsToStay.begin();

    // Delete other variables in static table
    std::set<string>::iterator tempIt3;
    for (tempIt3 = varsToLeave.begin(); tempIt3 != varsToLeave.end(); ++tempIt3) {
        string varName = *tempIt3;
        int staticIndex = lookUpStatic(currentFunc2.c_str(), varName.c_str());
        ;
        if (staticIndex != -1) { // Prevent repeat deletion
            staticTable.erase(staticTable.begin() + staticIndex);
        }
    }
}

// Resettle address of elements in static table
void resettlTableAddr() {
    int staticIndex;
    for (staticIndex = 0; staticTable[staticIndex].cls != funcs; ++staticIndex);
    // Skip all globals

    // Following address should be resettled

```

```

int currentAddr = 0;
for (; staticIndex < staticTable.size(); ++staticIndex) {
    tabElement* element = &staticTable[staticIndex];

    if (element->cls == funcs) { // Skip functions
        continue;
    }
    if (staticTable[staticIndex - 1].cls == funcs) { // Last one is a function
n
        if (element->cls == consts) { // Const, reserve its addr
            currentAddr = 0;
        }
        else { // Parameter or variable
            element->addr = 0;
            if (element->cls == vars && element->length != 0) { // Array
                currentAddr = element->length;
            }
            else { // Parameter or general variable
                currentAddr = 1;
            }
        }
    }
    else { // Last one isn't function
        if (element->cls == consts) { // Const, do nothing
        }
        else if (element->cls == vars && element->length != 0) { // Array
            element->addr = currentAddr;
            currentAddr += element->length;
        }
        else { // Parameter or general variable
            element->addr = currentAddr;
            ++currentAddr;
        }
    }
}
}

void optimizeInfixes() {
    // Get variables crossing basic blocks
    splitBlocks();

    // Draw DAG & operate optimization
    newInfixTable.reserve(infixTable.size());
    for (int i = 0; i < infixTable.size(); ++i) {
        if (isManageable(infixTable[i].ioperator)) { // Infix code is manageable

```

```

        infixToDAGNode(infixTable[i]);
    }
    else { // Infix code is not manageable
        if (infixTable[i].ioperator == "FUNC") { // Enter a new function
            currentFunc2 = infixTable[i].operand3;
        }
        if (allNodes.size() != 0) { // One basic block ends, operate optimization
            exportCodesFromDAG();
            // Empty DAG & table
            varsWithInitial.clear();
            allNodes.clear();
            varNodeTable.clear();
            lastNum = 0;
        }
        //执行 NOT manageable 指令
        insertNewInfix(infixTable[i].ioperator, infixTable[i].operand1,
            infixTable[i].operand2, infixTable[i].operand3);
    }
}

if (allNodes.size() != 0) { // Still one block not managed
    exportCodesFromDAG();
    // Empty DAG & table
    varsWithInitial.clear();
    allNodes.clear();
    varNodeTable.clear();
}

// Resettle staticTable elements' address
resettleTableAddr();

// Replace elements in infix table with that in new one
infixTable.clear();
infixTable.resize(newInfixTable.size());
for (int j = 0; j < newInfixTable.size(); ++j) {
    infixTable[j] = newInfixTable[j];
}

// Output to file
betterInfixFile.open("betterInfixes.txt", ios::out);
betterInfixFile << newOutputBuff;
betterInfixFile.close();
}

```

