[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
*G06F 9/45 (2006.01)*

# [12] 发明专利申请公布说明书

[21] 申请号 200710106835.5

[43] 公开日 2007 年 11 月 14 日

[11] 公开号 CN 101071385A

[22] 申请日 2007.5.10

[21] 申请号 200710106835.5

[30] 优先权

[32] 2006. 5.11 [33] JP [31] 2006 - 132380

[71] 申请人 松下电器产业株式会社

地址 日本大阪府门真市

[72] 发明人 浅尾忍

[74] 专利代理机构 北京德琦知识产权代理有限公司

代理人 陆 弋 朱登河

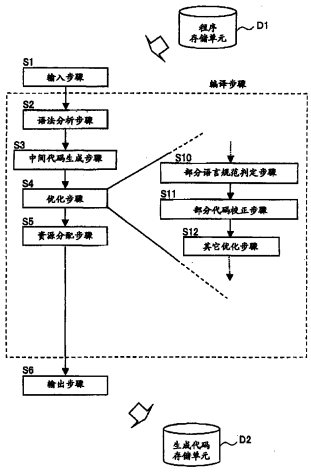
权利要求书 4 页 说明书 16 页 附图 17 页

[54] 发明名称

编译方法、调试方法、编译程序和调试程序

[57] 摘要

一种用于将输入程序转换成目标程序的编译方法，包括：部分语言规范判定步骤，用于判定在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范；判断步骤，用于判断所述特定范围中的语言规范之间是否存在差别；和部分代码校正步骤，用于在判断出所述特定范围中的语言规范之间存在差别时，对特定范围之一中的至少一部分代码进行校正。



1、一种用于将输入程序转换成目标程序的编译方法，包括:

部分语言规范判定步骤，用于判定在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范;

判断步骤，用于判断所述特定范围中的语言规范之间是否存在差别；和

部分代码校正步骤，用于在判断出所述特定范围中的语言规范之间存在差别时，对特定范围之一中的至少一部分代码进行校正。

2、根据权利要求 1 所述的编译方法，其中

在所述部分语言规范判定步骤中，基于在特定范围中使用的编程语言的工具性程序判定所述语言规范。

3、根据权利要求 1 所述的编译方法，其中

在所述部分语言规范判定步骤中，在所述输入程序中存在语言规范控制语句的情况下，基于所述语言规范控制语句判定所述语言规范。

4、根据权利要求 1 所述的编译方法，其中

在所述部分语言规范判定步骤中，在语言规范控制指令被提供给用于编译所述输入程序的编译系统的情况下，基于所述语言规范控制指令判定所述语言规范。

5、根据权利要求 1 所述的编译方法，其中

在所述部分代码校正步骤中，在所有函数中的定义语言规范与引用语言规范之间存在差别的情况下，具有子集语言规范的函数代码名被改变为具有超集语言规范的函数代码名。

6、根据权利要求 1 所述的编译方法，其中

在所述部分代码校正步骤中，所有被多重定义的外部函数都被改变为具有单独函数名并且在程序中未被多重定义的外部函数，并且当所有被多重定义的外部函数都被改变为具有该单独函数名并且在程序中未被多重定义的该外部函数时，再次执行部分语言规范判定步骤。

7、根据权利要求 1 所述的编译方法，其中

在所述部分代码校正步骤中，所有属于一名字空间的外部函数都被改变为具有单独函数名并且在程序中不属于所述名字空间的外部函数，并且当所有属于所述名字空间的外部函数都被改变为具有所述单独函数名并且在程序中不属于所述名字空间的所述外部函数时，再次执行部分语言规范判定步骤。

8、根据权利要求 1 所述的编译方法，其中

在所述部分代码校正步骤中，所有由一模板生成的外部函数都被改变为具有单独函数名并且在程序中不是由该模板生成的外部函数，并且当所有由该模板生成的外部函数都被改变为具有所述单独函数名并且在程序中不是由该模板生成的所述外部函数时，再次执行部分语言规范判定步骤。

9、一种用于调试输入程序的调试方法，包括:

部分语言规范认可步骤，用于认可在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范；和

部分语言规范显示步骤，用于一起显示每个特定范围中所认可的语言规范及其程序来源。

10、根据权利要求 9 所述的调试方法，进一步包括模板开发显示步骤，用于在分析目标是由所述模板生成的实例的情况下，显示其中模板被开发的源程序。

11、根据权利要求 9 所述的调试方法，进一步包括用于显示违背子集语言规范的部分的步骤，其中在子集语言规范被定义在所认可的语言规范中的情况下，偏离子集语言规范的一部分程序的内容被显式地显示。

12、一种用于将输入程序转换成目标程序的编译程序，该编译程序使计算机执行:

部分语言规范判定工具性程序，用于判定在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范;

用于判断所述特定范围中的语言规范之间是否存在差别的工具性程序；和

部分代码校正工具性程序，用于在判断出所述特定范围之间存在差别的情

况下，对特定范围之一中的至少一部分代码进行校正。

13、根据权利要求 12 所述的编译程序，其中

所述部分语言规范判定工具性程序基于在特定范围中使用的编程语言的工具性程序判定所述语言规范。

14、根据权利要求 12 所述的编译程序，其中

所述部分语言规范判定工具性程序在输入程序中存在语言规范控制语句的情况下，基于所述语言规范控制语句判定所述语言规范。

15、根据权利要求 12 所述的编译程序，其中

所述部分语言规范判定工具性程序在语言规范控制指令被提供给用于编译所述输入程序的编译系统的情况下，基于所述语言规范控制指令判定所述语言规范。

16、根据权利要求 12 所述的编译程序，其中

所述部分代码校正工具性程序在所有函数中的定义语言规范与引用语言规范之间存在差别的情况下，将具有子集语言规范的函数代码名改变为具有超集语言规范的函数代码名。

17、根据权利要求 12 所述的编译程序，其中

所述部分代码校正工具性程序将所有被多重定义的外部函数都改变为具有单独函数名并且在程序中未被多重定义的外部函数，并且当所有被多重定义的外部函数都被改变为具有该单独函数名并且在程序中未被多重定义的该外部函数时，再次执行部分语言规范判定工具性程序。

18、根据权利要求 12 所述的编译程序，其中

所述部分代码校正工具性程序将所有属于一名字空间的外部函数都改变为具有单独函数名并且在程序中不属于所述名字空间的外部函数，并且当所有属于所述名字空间的外部函数都被改变为具有所述单独函数名并且在程序中不属于所述名字空间的所述外部函数时，再次执行部分语言规范判定功能。

19、根据权利要求 12 所述的编译程序，其中

所述部分代码校正工具性程序将所有由一模板生成的外部函数都改变为具

有单独函数名并且在程序中不是由该模板生成的外部函数，并且当所有由该模板生成的外部函数都被改变为具有该单独函数名并且在程序中不是由该模板生成的该外部函数时，再次执行部分语言规范判定步骤。

20、一种用于调试输入程序的调试程序，所述调试程序使计算机执行:

部分语言规范认可工具性程序，用于认可在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范；和

部分语言规范显示工具性程序，用于一起显示每个特定范围中所认可的语言规范及其程序来源。

21、根据权利要求 20 所述的调试程序，其中

该程序进一步使该计算机执行一模板开发显示工具性程序，其用于在分析目标是由所述模板生成的实例的情况下，显示其中模板被开发的源程序。

22、根据权利要求 20 所述的调试程序，其中

该程序进一步使该计算机执行一子集语言规范违背显示步骤，其用于在子集语言规范被定义在所认可的语言规范中的情况下，显式地显示偏离子集语言规范的一部分程序的内容。

# 编译方法、调试方法、编译程序和调试程序

## 技术领域

本发明涉及用于将用高级语言描述的程序转换成目标程序的编译方法、调试方法、编译程序和调试程序。

## 背景技术

在近年来的软件开发中，程序规模不断增大，基于这种情况，具有高可维护性和高可重用性的面向目标语言受到了关注。面向目标语言的典型例子是C++语言。作为替代一直以来传统地广泛用于编程的C语言的语言，C++语言正在成为关注的焦点，并且在相关技术领域，编程中的语言从C语言转变到C++语言。在这种语言转变中，因为C++语言是C语言的高级兼容语言，故在常规情况下只要简单地通过将C语言编译器替换为C++语言编译器，就能生成没有任何操作问题的目标代码。然而，当用C语言中叙述的程序在C++语言中编译时，会引起代码大小和执行时间不受欢迎地增长的问题。

针对该问题的常规第一解决方案是，使用文献（见 M·A·Ellis、B·Stroustrup著，Takanori Adachi 和 Hiroshi Koyama译的“注释C++参考手册”7.4 章“连接指配”）中所叙述的关于程序的连接指配（linkage assignment）。第二解决方案是分析程序中语言规范的范围并且尽可能地对编译自动应用子集的语言规范，如同在日本专利申请公开 No. 2003-50700 中所叙述的。

然而，在第一解决方案中，迫使程序员在编程中考虑连接指配，这阻碍了C语言轻易地向C++语言转变。在第二解决方案中，没有考虑到与程序中编码符号（函数名、变量名等）相关的一致性，虽然可以将C++语言编

译成用作内建为 C++子集规范的 EC++语言。结果是，无法期望这样的优化，例如基于语言规范的兼并对代码大小和执行时间的缩减。

发明内容

因此，本发明的主要目的在于提供一种编译方法，其中程序员可以轻易地将程序转变为向上兼容的程序，而不必注意任何连接指配，并且作为程序规范的最大兼并的结果，可以实现代码大小和执行时间的缩减，而且本发明的目的还在于提供能够轻易地进行微调的调试方法。

为了解决以上问题，根据本发明的编译方法是一种用于将输入程序转换成目标程序的编译方法，包括:

部分语言规范判定步骤，用于判定在输入程序中设置的至少两个特定范围中的语言规范;

判断步骤，用于判断所述特定范围中的语言规范之间是否存在差别; 和

部分代码校正步骤，用于在判断出所述特定范围中的语言规范之间存在差别时，对特定范围之一中的至少一部分代码进行校正。

根据上述方法，由于代码被校正从而使得它们可以彼此组合在一起，包括部分不同语言规范的程序可以被组合起来并且以最优的语言规范编译，因此可以更有效地生成代码。

优选地，在所述部分语言规范判定步骤中，基于在特定范围中使用的编程语言的工具性程序判定所述语言规范。

根据上述方法，由于程序员可以将包括部分不同语言规范的程序组合在一起而不校正源程序，并且可以以最优的语言规范编译该程序，因此可以更有效地生成代码。

作为上述方法的更加优选的模式，在所述部分语言规范判定步骤中，在所述输入程序中存在语言规范控制语句的情况下，基于所述语言规范控制语句判定所述语言规范。

根据上述方法，例如，程序员可以根据源程序中的#pragma 指令来描述所

述语言规范控制语句，并且因此可以自由地选择语言规范而不受来自源程序中初始描述的工具性程序的任何影响。

更加优选地，在所述语言规范判定步骤中，在向用于编译所述输入程序的编译系统提供了语言规范控制指令的情况下，基于所述语言规范控制指令判定所述语言规范。

根据上述方法，例如，程序员基于编译系统的选项提供语言规范控制指令，并且由此可以自由地选择语言规范而不校正源程序，并且不受来自源程序中初始描述的工具性程序的影响。

更加优选地，在所述部分代码校正步骤中，在所有函数中的定义语言规范与引用语言规范之间存在差别的情况下，具有子集语言规范的函数代码名被改变为具有超集语言规范的函数代码名。

根据上述方法，即使在定义的语言规范和引用的语言规范之间存在任何差别，也可以将程序组合起来。

作为上述方法的更加优选的模式，在所述部分代码校正步骤中，所有被多重定义的外部函数都被改变为具有单独函数名并且在程序中未被多重定义的外部函数，并且当所有被多重定义的外部函数都被改变为具有所述单独函数名并且在程序中未被多重定义的所述外部函数时，再次执行部分语言规范判定步骤。

根据上述方法，使用多重定义函数的C++程序可以作为C程序进行编译，这提高了代码生成的效率。

作为上述方法的更加优选的模式，在所述部分代码校正步骤中，所有属于一名字空间的外部函数都被改变为具有单独函数名并且在程序中不属于所述名字空间的外部函数，并且当所有属于所述名字空间的外部函数都被改变为具有所述单独函数名并且不属于在程序中所述名字空间的所述外部函数时，再次执行部分语言规范判定步骤。

根据上述方法，使用所述名字空间的C++程序可以作为C程序编译，这提高了代码生成的效率。

作为上述方法的更加优选的模式，在所述部分代码校正步骤中，所有由一

模板生成的外部函数都被改变为具有单独函数名并且在程序中不是由该模板生成的外部函数，并且当所有由该模板生成的外部函数都被改变为具有该单独函数名并且在程序中不是由该模板生成的外部函数时，再次执行部分语言规范判定步骤。

根据上述方法，使用所述名字空间的C++程序可以作为C程序编译，这提高了代码生成的效率。

根据本发明的调试方法是一种用于调试输入程序的调试方法，包括:

部分语言规范认可步骤，用于认可在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范；和

部分语言规范显示步骤，用于一起显示每个特定范围中所认可的语言规范及其程序来源。

根据上述方法，程序员可以轻易地确认程序的哪个部分以何种语言规范被编译。结果是，可以有效地进行调试和微调操作。

作为上述方法的优选模式，该方法进一步包括模板开发显示步骤，其用于在待分析的目标是由所述模板生成的实例的情况下，显示其中模板被开发的源程序。

根据上述方法，在开发模板之后，程序员可以轻易地掌握源程序，并且可以有效地进行调试和微调操作。

作为上述方法的更加优选的模式，该方法进一步包括子集语言规范违背部分显示步骤，其用于在子集的语言规范被定义在所认可的语言规范中的情况下，显式地显示偏离子集的语言规范的一部分程序的内容。

根据上述方法，程序员可以轻易地掌握偏离子集的部分程序，并且可以更有效地进行调试和微调操作。

本发明不仅可以实现包括这些特征步骤的编译方法和调试方法，而且可以实现使计算机执行包含在所述编译方法和所述调试方法中的这些特征步骤的编译程序和调试程序，以及执行包含在所述编译方法和所述调试方法中的这些特征步骤的编译装置和调试器装置。进一步，显而易见地，可以通过例如CD-

ROM（光盘只读存储器）的记录介质和例如互联网的传输介质来分布所述编译器和调试器。

根据本发明的编译程序是一种用于将输入程序转换成目标程序的编译程序，该编译程序使计算机执行:

部分语言规范判定工具性程序，用于判定在输入程序中设置的至少两个特定范围的每个特定范围中的语言规范;

用于判断所述特定范围之间的语言规范中是否存在差别的工具性程序；和

部分代码校正工具性程序，用于在判断出所述特定范围之间的语言规范中存在差别的情况下，对特定范围之一中的至少一部分代码进行校正。

作为上述程序的优选模式，所述部分语言规范判定工具性程序，基于所述特定范围中所使用的编程语言的工具性程序来判定所述语言规范。

作为上述程序的更加优选的模式，所述部分语言规范判定工具性程序在输入程序中存在语言规范控制语句的情况下，基于所述语言规范控制语句判定所述语言规范。

作为上述程序的更加优选的模式，所述部分语言规范判定工具性程序在向用于编译所述输入程序的编译系统提供了语言规范控制指令的情况下，基于所述语言规范控制指令判定所述语言规范。

作为上述程序的更加优选的模式，所述部分代码校正工具性程序在所有函数中的定义语言规范和引用语言规范之间存在差别的情况下，将具有子集语言规范的函数代码名改变为具有超集语言规范的函数代码名。

作为上述程序的更加优选的模式，所述部分代码校正工具性程序将所有被多重定义的外部函数都改变为具有单独函数名并且在程序中未被多重定义的外部函数，并且当所有被多重定义的外部函数都被改变为具有该单独函数名并且在程序中未被多重定义的该外部函数时，再次执行部分语言规范判定工具性程序。

作为上述程序的更加优选的模式，所述部分代码校正工具性程序将所有属于一名字空间的外部函数都改变为具有单独函数名并且不属于在程序中所述名

字空间的外部函数，并且当所有属于所述名字空间的外部函数都被改变为具有该函数名并且不属于所述名字空间的该外部函数时，再次执行部分语言规范判定功能。

作为上述程序的更加优选的模式，所述部分代码校正工具性程序将所有由一模板生成的外部函数都改变为程序中具有单独函数名并且在不是由该模板生成的外部函数，并且当所有由该模板生成的外部函数都被改变为在程序中具有该单独函数名并且不是由该模板生成的外部函数时，再次执行部分语言规范判定步骤。

根据本发明的调试程序是一种用于调试输入程序的调试程序，所述调试程序用于使计算机执行以下工具性程序：

部分语言规范认可工具性程序，用于认可其中输入程序被设置的至少两个特定范围的每个特定范围中的语言规范；和

部分语言规范显示工具性程序，用于一起显示每个特定范围中所认可的语言规范及其程序来源。

作为上述程序的优选模式，该调试程序用于进一步使计算机执行一模板开发显示工具性程序，其用于在分析目标是由所述模板生成的实例的情况下，显示其中所述模板被开发的源程序。

作为上述程序的更加优选的模式，该调试程序进一步使计算机执行一子集语言规范违背显示步骤，其用于在子集的语言规范被定义在所认可的语言规范中的情况下，显式地显示偏离子集语言规范的一部分程序的内容。

根据本发明的编译方法，包括部分不同语言规范的程序可以被轻易地组合在一起，并且以最优的语言规范进行编译。结果是，可以更加有效地生成代码。进一步，根据本发明的调试方法，程序员可以轻易地掌握程序的该部分以何种语言规范进行编译，以及源程序的哪一部分在其被改变为子集语言规范时应该被校正。结果是，可以有效地执行调试和微调操作。

根据本发明的编译方法和调试方法可以有效地应用于针对诸如移动电话和 PDA（个人数字助理）的嵌入式装置的编译方法和调试方法等，所述

嵌入式装置要求具有小代码大小的目标代码。

附图说明

本发明的这些和其它目的以及优点将通过以下对本发明优选实施例的描述将变得清晰。在实施本发明之后，本领域技术人员将会注意到在本说明书中未叙述的许多优点。

图 1 是根据本发明优选实施例由编译器执行的处理步骤的流程图。

图 2 是示出部分语言规范判定步骤的细节的流程图。

图 3 是示出部分代码校正步骤的细节的流程图。

图 4 是根据优选实施例由调试器执行的处理步骤的流程图。

图 5 是示出部分语言规范显示步骤的细节的流程图。

图 6 示出具体示例 1 中使用的程序存储单元中所存储的源程序的示例。

图 7 示出根据现有技术的中间代码信息的示例。

图 8 示出由传统方法校正的源程序的示例。

图 9 示出在具体示例 1 中使用的源程序被应用到部分语言规范判定步骤之后的中间代码信息的示例。

图 10 示出在具体示例 1 中使用的源程序被应用到部分语言代码校正步骤之后的中间代码信息的示例。

图 11 示出在具体示例 2 中用的程序存储单元中所存储的源程序的示例。

图 12 示出在其中具体示例 2 中所用源程序以 C + + 语言被编译的汇编代码，和在其中具体示例 2 中所用源程序以 C 语言被编译的汇编代码。

图 13 示出在具体示例 2 中中使用的源程序被应用到第一部分语言规范判定步骤之后中间代码信息的示例。

图 14 示出在具体示例 2 中中使用的源程序被应用到第一部分代码校正步骤之后中间代码信息的示例。

图 15 示出在具体示例 2 中中使用的源程序被应用到第二部分语言规范判定步骤之后中间代码信息的示例。

图 16 示出其中根据本发明的方法未被使用到具体示例 2 中所用源程序的汇编代码，和其中相关方法被应用到具体示例 2 中所用源程序的汇编代码。

图 17 示出显示具体示例 3 中所使用的源程序的调试器监视器。

图 18 示出显示具体示例 3 中所用源程序的语言规范信息的调试器监视器。

图 19 示出显示其中具体示例 3 中所用源程序模板被开发的源程序的调试器监视器。

图 20 示出显示从具体示例 3 中所用源程序中的 C 语言规范导出的程序的一部分内容的调试器监视器。

图 21 是传统程序转换方法（第二解决方案）的流程图。

具体实施方式

以下，参照附图描述根据本发明优选实施例的编译方法。图 1 是由编译器执行的处理步骤的流程图。

编译器读取存储在程序存储单元 D1 中的头文件和源程序（步骤 S1）。编译器分析所读取源文件的语法，并且生成符号表和语法树（步骤 S2）。接下来，编译器基于所生成的语法树生成中间代码（步骤 S3）。然后，编译器对所生成的中间代码执行各种优化处理（步骤 S4）。进一步，编译器分配硬件资源，例如寄存器和存储器，给经过优化的中间代码中包括的所有变量（步骤 S5），然后将分配了资源的中间代码转换成目标代码并且输出其目标程序到生成代码存储单元 D2（步骤 S6）。

优化步骤 S4 包括部分语言规范判定步骤 S10、部分代码校正步骤 S11 和其它优化步骤 S12。在部分语言规范判定步骤 S10 中，分析中间代码，并且判定每个程序的部分范围中的语言规范。在部分代码校正步骤 S11 中，对在部分语言规范判定步骤 S10 中所判定的每个程序的部分范围中的语言规范进行分析，并且对相应范围中的代码进行校正，从而使其能够被基于相应

范围中的语言规范的差别而组合。稍后描述部分语言规范判定步骤 S10 和部分代码校正步骤 S11 的细节。不描述其它优化步骤 S12，因为这是传统的优化步骤，而不是本发明的主题。

对源程序输入步骤 S1、语法分析步骤 S2、中间代码生成步骤 S3、其它优化步骤 S12、资源分配步骤 S5 和目标输出步骤 S6 不进行详细描述，因为它们类似于传统的步骤而不是本发明的主要主题。

以下给出对作为本发明主题的部分语言规范判定步骤 S10 和部分代码校正步骤 S11 的描述。图 2 是示出部分语言规范判定步骤 S10 细节的流程图。作为循环处理 L1，对于每个中间代码执行步骤 S21－25，所述中间代码对应于所输入程序中设置的至少两个特定范围。在存在分析目标的情况下，处理前进到步骤 S21；而在没有分析目标的情况下，处理前进到步骤 S11。在步骤 S21 中，基于分析目标的范围中所用的编程语言的工具性程序判定语言规范，将语言规范信息记录在中间语言中，并且处理前进到步骤 S22。这里，语言规范信息是至少能够判断其所编译的是何种语言规范的信息（例如，当其为 C＋＋语言时，是诸如“C＋＋”的字符串）。

在步骤 S22 中，判断分析目标的范围中是否存在语言规范控制语句。此处所叙述的语言规范控制语句是这样的语句，其中通过例如#pragma 指令在源程序中进行描述，而将特定操作指定给编译器。当判断结果为真时，处理前进到步骤 S23，而当结果为非真时，处理前进到步骤 S24。

在步骤 S23 中，基于语言规范控制指令判定语言规范，将语言规范信息记录在中间语言中，然后处理前进到步骤 S24。在步骤 S24 中，判断语言规范控制指令是否提供给编译系统。此处所叙述的语言规范控制指令是直接给到编译系统的指令，从而使得特定操作被作为命令行选项指定给编译器。当判断结果为真时，处理前进到步骤 S25，而当结果为非真时，处理前进到循环处理 L1。

图 3 是示出部分代码校正步骤 S11 的细节的流程图。在循环处理 L2 中，步骤 S31－S38 由所有函数的每个中间代码执行。当存在分析目标时，处理

前进到步骤 S31；而当不存在分析目标时，处理前进到步骤 S39。

在步骤 S31 中，判断相关函数的定义与引用中的语言规范之间是否有差别。语言规范之间的差别表示这样的情况，即：函数定义中的语言规范在中间代码中被记录为“C++”，而函数引用中的语言规范在中间代码中被记录为“C”。当判断结果为真，则处理前进到步骤 S32，而当判断结果为非真时前进到步骤 S33。

在步骤 S32，具有子集语言规范的函数代码名被改变为具有超集语言规范的函数代码名，处理前进到步骤 S33。

在步骤 S33 中，判断相关函数是否为被多重定义的外部函数。多重定义表示这种情况的函数定义，即函数名相同但是函数自变量不同，例如 f( void ) 和 f ( int )。当判断结果为真时处理前进到步骤 S34，而当结果为非真时前进到步骤 S35。

在步骤 S34 中，相关函数被改变为具有单独函数名并且在程序中没有被多重定义的外部函数。接下来，处理前进到步骤 S35。在步骤 S35 中，判断相关函数是否为属于名字空间的外部函数。属于名字空间的外部函数是在作为名字空间 S{int f (void)}的名字空间的作用域中声明的函数。当判断结果为真则处理前进到步骤 S36，当结果为非真则前进到步骤 S37。

在步骤 S36 中，相关函数被改变为具有单独函数名并且不属于程序中的名字空间的外部函数。接下来，处理前进到步骤 S37。在步骤 S37 中，判断相关函数是否为由模板生成的外部函数。这里，由模板生成的外部函数是由如同在模板<class T> Tf(T a) 中所定义的函数模板例示的函数。当判断结果为真则处理前进到步骤 S38，当结果为非真则前进到循环处理 L2。

在步骤 S38 中，相关函数被改变为具有单独函数名并且不是由程序中模板生成的外部函数。接下来，处理前进到 L2。在步骤 S39 中，判断步骤 S33、S35 和 S37 的判断结果中的任意一个是否为真。当判断结果是真则处理前进到步骤 S10，当结果为非真则前进到步骤 S12。

如上所述，对中间代码执行包括部分语言规范判定步骤 S10 和部分代码

校正步骤 S11 的优化步骤 S4，并且然后对经过优化的中间代码执行资源分配步骤（步骤 S5）和目标程序输出步骤（步骤 S6）。因此，程序员可以轻易地将程序转变为向上兼容的程序而不用注意任何连接指配，并且由于语言规范的兼并可以最大程度地缩减代码大小和执行时间。

本发明不限于对中间代码的分析，可以对任何类型的数据进行分析，只要该数据是表示源程序的语法分析结果的数据。

接下来，参照附图描述根据本优选实施例的调试方法。图 4 是由调试器执行的处理步骤的流程图。调试器读取存储在程序存储单元 D1 中的程序（步骤 N1）。调试器分析所读取的程序，并且转到命令输入等待步骤 N2。接下来，调试器判断是否输入了部分语言规范显示命令（步骤 N3）。当判断结果为真，则执行用于部分地显示语言规范的步骤 N10。然后，执行用于显示其它源信息的步骤 N11，并且处理前进到命令输入等待步骤 N2。当结果为非真时，执行另一调试步骤（步骤 N4），并且处理前进到命令输入等待步骤 N2。稍后将描述部分语言规范显示步骤 N10 的细节。由于其它源显示步骤 N11 是用于显示一般源信息的步骤，并且不是本发明的主题，因此不对其进行描述。

另外，不详细描述程序输入步骤 N1、命令输入等待步骤 N2 和另一调试步骤 N4，因为它们是类似于传统步骤的处理步骤，并且不是本发明的主题。

以下，描述本发明关键结构的部分语言规范显示步骤 N10。图 5 是示出部分语言规范显示步骤 N10 细节的流程图。在循环处理 L3 中，由与所输入程序的特定范围对应的每个调试信息执行步骤 N21－N26。在存在分析目标时处理前进到步骤 N21，在不存在分析目标时处理前进到步骤 N11。

在步骤 N21 中，判断是否存在语言规范信息。当判断为真则处理前进到步骤 N22，当判断为非真则前进到循环处理 L3。在步骤 N22 中，基于记录在调试信息中的语言规范信息显示每个范围的语言规范及其程序来源，然后处理前进到步骤 N23。

在步骤 N23 中，判断分析目标是否为由模板生成的实例。当判断结果为真则处理前进到步骤 N24，当结果为非真则前进到步骤 N25。在步骤 N24 中，显示其中模板被开发的源程序，并且处理前进到步骤 N25。

在步骤 N25 中，判断子集的语言规范是否被定义为所认可的语言规范。当判断结果为真则处理前进到步骤 N26，当结果为非真则前进到循环处理 L3。在步骤 N26 中，清楚地描述了与子集的语言规范偏离的程序部分的内容。

当执行上述包括部分语言规范显示步骤 N10 的调试过程时，程序员可以轻易地掌握程序的每个部分以何种语言规范被编译，以及源程序的哪个部分在其被改变为子集的语言规范时应该被校正。结果是，可以有效地执行调试和微调操作。

以下，参照具体示例更加具体地描述根据本发明的编译方法和调试方法，在具体示例中，C++语言用作超集的语言规范而 C 语言用作子集的语言规范。

具体示例 1

图 6 示出存储在程序存储单元 D1 中的源程序的示例。以下描述在输入源程序的情况下的编译方法。假设用户指定<main.cpp>作为 C++语言被编译而<sub.c>作为 C 语言被编译。可以采用这样的方式指定语言规范，即在源程序中描述#pragma，然而这类似于基于命令行指定语言规范的情况，因此在此说明书中省略对其的描述。图 6 中所示的源文件<main.cpp>引用外部函数 f。源文件<sub.c>定义外部函数 f。

参照图 7，预先提及要解决的问题。图 7 示出在中间代码生成步骤 S3 中生成的分别位于函数 f 引用侧和定义侧的中间代码中的信息的一部分。

在根据传统方法编译具体示例 1 的情况下，进行连接而不考虑分析目标的中间代码的语言规范。因此，在具体示例 1 中函数代码名不相同（_f_Fv 和_f），这导致在连接时产生错误。因此，如图 8 所示，有必要以用户校正程序的方式合并函数代码名，从而使得连接指配显式地示出（见“C”）。

这样，传统上需要校正步骤来在C语言中和在C++语言中使用源，这降低了开发过程中的效率程度。

接下来，示出本优选实施例的具体示例。在本具体示例中，在函数作用域的范围内进行分析。在部分语言规范判定步骤S10中，由每个中间代码执行图2所示的循环处理L1（步骤S21－S25），所述中间代码对应于与中间代码生成步骤中生成的中间代码相对应的各个函数作用域的范围。在本具体示例中，假设用户指定<main.cpp>以C++语言编译，而<sub.c>以C语言编译。因此，步骤S24的判断结果为真，则执行步骤S25。结果是，指示语言规范为C++语言的信息记录在引用侧的中间代码中，而指示语言规范为C语言的信息记录在定义侧的中间代码中（见图9）。

接下来，执行部分代码校正步骤S11。首先，执行图3所示的循环处理L2（步骤 S31－S38）。在本具体示例中，在分别位于定义侧和引用侧的中间代码的语言规范信息之间存在差别（C++语言和C语言）。因此，步骤S31的判断结果为真，则执行步骤S32。相互比较C++语言和C语言，C++语言是超集的语言规范，而C语言是子集的语言规范。因此，在具有子集语言规范的定义侧的中间代码的函数代码名被改变为在具有超集语言规范的引用侧的中间代码的函数代码名（图10，合并为_f_Fv）。在之后的步骤S33－S38中，判断出所有的步骤都为非真。

如上所述，通过考虑分析目标的中间代码的语言规范，使得相同函数代码名被合并，并且允许连接而不显式地指配连接。因此，传统上为了如同使用C++语言中的源那样使用C语言中的源所必须的校正源的步骤，变得不必要。结果是，开发过程可以达到提高的效率。

具体示例2

图11示出使用存储在程序存储单元D1中的多重定义函数的源程序的示例。以与多重定义函数情况下的具体示例相似的方式，对在其中使用名字空间和模板的源程序进行处理，在此省略对它的描述。以下给出对输入了相关源程序的情况下的编译方法的描述。假设<test.cpp>被编译，而用户没有

指定语言规范。

参照图 12，预先指出问题区域。在根据传统方法编译本具体示例中的源程序<test.cpp>的情况下，所有函数都被作为 C + +语言的函数进行编译，因为在源程序中使用了作为 C + +语言的工具性程序的多重定义函数。因为这些函数被作为 C + +语言的函数进行编译，即使在函数体中只使用了 C 语言的工具性程序，还是会冗余地生成代码。图 12 示出了“test.cpp”的“void f(void)”被作为 C + +语言的函数进行编译的情况下的代码生成结果，和“test.cpp”的“void f(void)”被作为 C 语言的函数进行编译的情况下的代码生成结果。如同从图 12 中清晰所见，与作为 C 语言的函数进行编译相比，作为 C + +语言的函数进行编译的代码生成结果中冗余地生成了代码。

接下来，示出本优选实施例的另一具体示例。以类似于具体示例 1 的方式，对本具体示例的函数作用域执行分析。在部分语言规范判定步骤 S10 中，由每个中间代码执行图 2 所示的循环处理 L1（步骤 S21 - S25），所述中间代码对应于与中间代码生成步骤中生成的中间代码相对应的各个函数作用域。在本具体示例中，由于用户没有指定语言规范，步骤 S22 和 S24 的判断结果均为非真，因此不执行步骤 S23 和 S25。因此，基于分析目标范围中所用编程语言的工具性程序而判定的语言规范信息被记录在中间代码中（步骤 S21）。在本具体示例中，函数“void f (void)”和“void f (int)”是具有相同函数名的多重定义函数，并且是 C + +语言的工具性程序（函数）。因此，语言规范为 C + +语言的信息被记录在使用这些工具性程序的范围中的中间代码中（如图 13 所示的所有中间代码中的 C + +语言）。

接下来，执行部分代码校正步骤 S11。首先，执行图 3 所示的循环处理 L2（步骤 S31 - S38）。在本具体示例中，由于定义侧中间代码的语言规范信息与引用侧中间代码的语言规范信息之间不存在差别（都是 C + 语言），因此不执行步骤 S32，而是执行步骤 S34，因为相关函数是多重定义函数。在本具体示例中，如图 14 所示，函数名被改变为诸如__L 1、__L2 的函数名，它们是具有单独函数名的外部函数名并且在程序中未被多重定义，并在

此后保持每个语言规范和函数代码名。这里将多重定义函数叙述为本具体示例中的一个例子，然而，函数名可以被改变为程序中具有单独函数名并且在程序中未被多重定义的外部函数，其中以类似于多重定义函数的方式使用名字空间和模板（步骤 S36 和 S38）。

结果是，执行步骤 S34，步骤 S39 的判断结果为真，并且再次执行部分语言规范判定步骤 S10。在步骤 S10 的第二次执行中，由于多重定义函数的函数名被改变为__L1、__L2，所以，相关函数不被认可为多重定义函数。接下来，在步骤 S21 记录表示语言规范为 C 语言的信息，并且作为 C 语言的函数代码名被赋给相关函数（图 15）。在第二步骤 S11 的执行中没有造成特定改变，并且处理前进到步骤 S12 及其后的处理步骤。

图 16 示出本发明未应用到本具体示例的情况下的代码生成结果，和本发明应用到本具体示例的情况下的代码生成结果。如图 16 清晰所示，较之未应用本发明的情况，在应用本发明的情况下，任何冗余代码都被删除了。

具体示例 3

示出了调试方法的具体示例。在本具体示例中，假设根据本发明编译方法记录在中间代码中的语言规范信息也记录在调试信息中。图 17 示出执行调试程序的监视器。当部分语言规范显示命令被输入时，执行部分语言规范显示步骤 N10。在本具体示例中，由于假设语言规范信息记录在调试信息中，因此执行步骤 N22，并且基于记录在调试信息中的语言规范信息，一起显示在所输入程序中设置的至少两个范围当中每个范围中的语言规范及其程序来源。图 18 示出语言规范显示的示例。该显示例仅仅示出示例，而且任何方式都是可接受的，只要显示关于语言规范的信息。

接下来，判断分析目标是否为由模板生成的实例（步骤 N23）。在本具体示例中，在分析目标是 A<int>obj 的情况下，因为实例生成自模板，所以判断结果为真，并且如图 19 所示，显示其中模板被开发的源程序。该显示例仅仅示出示例，而且任何方式都是可接受的，只要显示关于其中模板被开发的源程序的信息。

接下来，判断子集的语言规范是否被定义为被认可的规范（步骤 N25）。在本具体示例中，由于定义的是作为 C++语言的子集规范的 C 语言，因此判断结果为真。然后，如图 20 所示，在语言规范被判断为 C++语言的范围内，偏离 C 语言语言规范（模板工具性程序）的一部分的内容被显式地描述。该显示示例仅仅示出示例，而任何方式都是可接受的，只要显示关于偏离子集语言规范的那部分内容的信息。

如从图 18－20 清晰所见，当使用根据本发明的调试方法时，程序员可以轻易地掌握程序的哪一部分基于何种语言规范进行编译以及源程序的哪一部分应该在其被改变为子集语言规范时被校正。结果是，可以有效率地执行调试和微调操作。

虽然详细描述了本发明的优选实施例，但是应该理解，可以对本发明进行多种修改，并且本发明意在所附权利要求中覆盖所有落入本发明真正精神和范围中的这些修改。

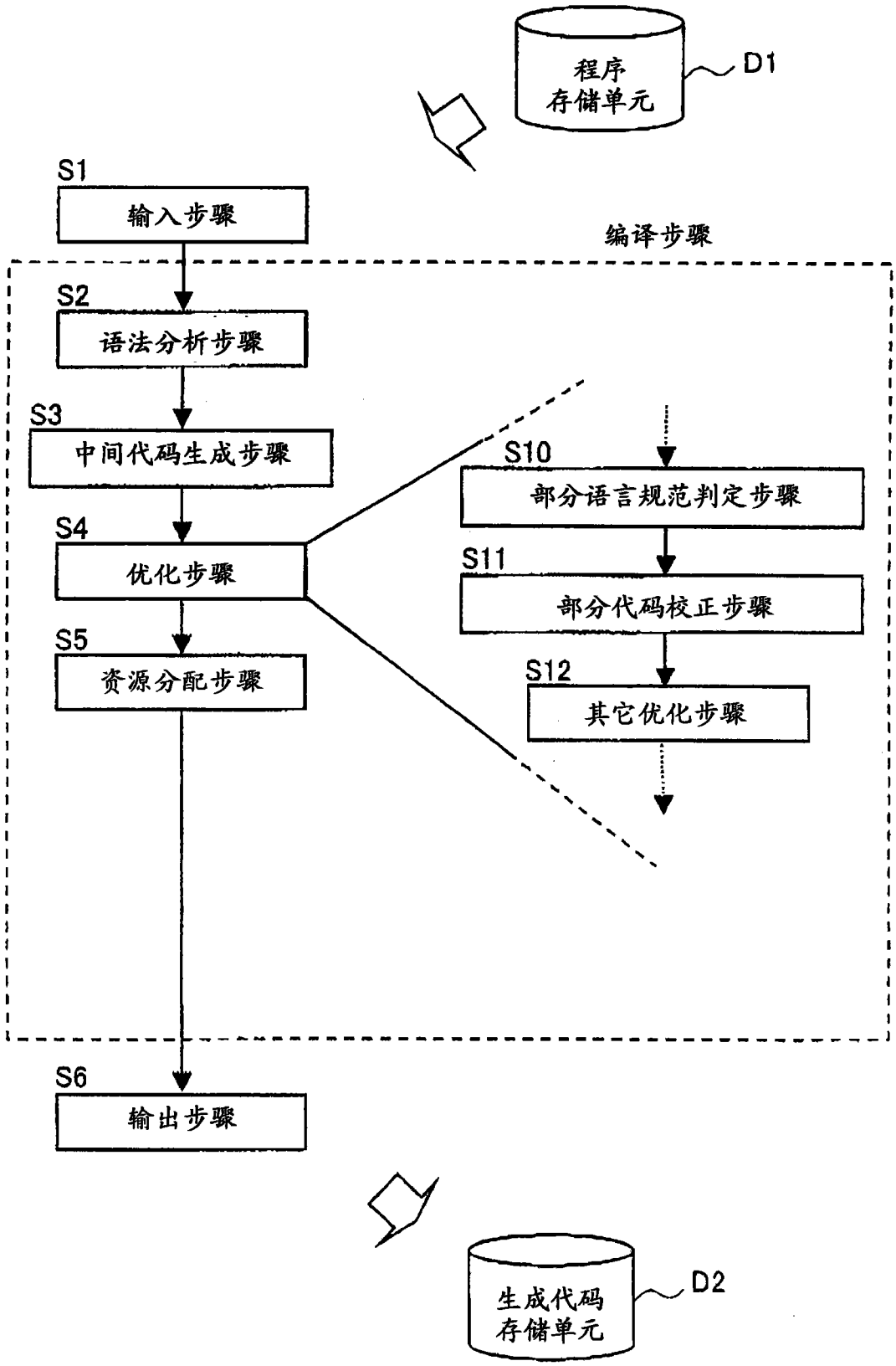

程序
存储单元
D1
S1
输入步骤
编译步骤
S2
语法分析步骤
S3
中间代码生成步骤
S4
优化步骤
S5
资源分配步骤
S10
部分语言规范判定步骤
S11
部分代码校正步骤
S12
其它优化步骤
S6
输出步骤
生成代码
存储单元
D2

图 1

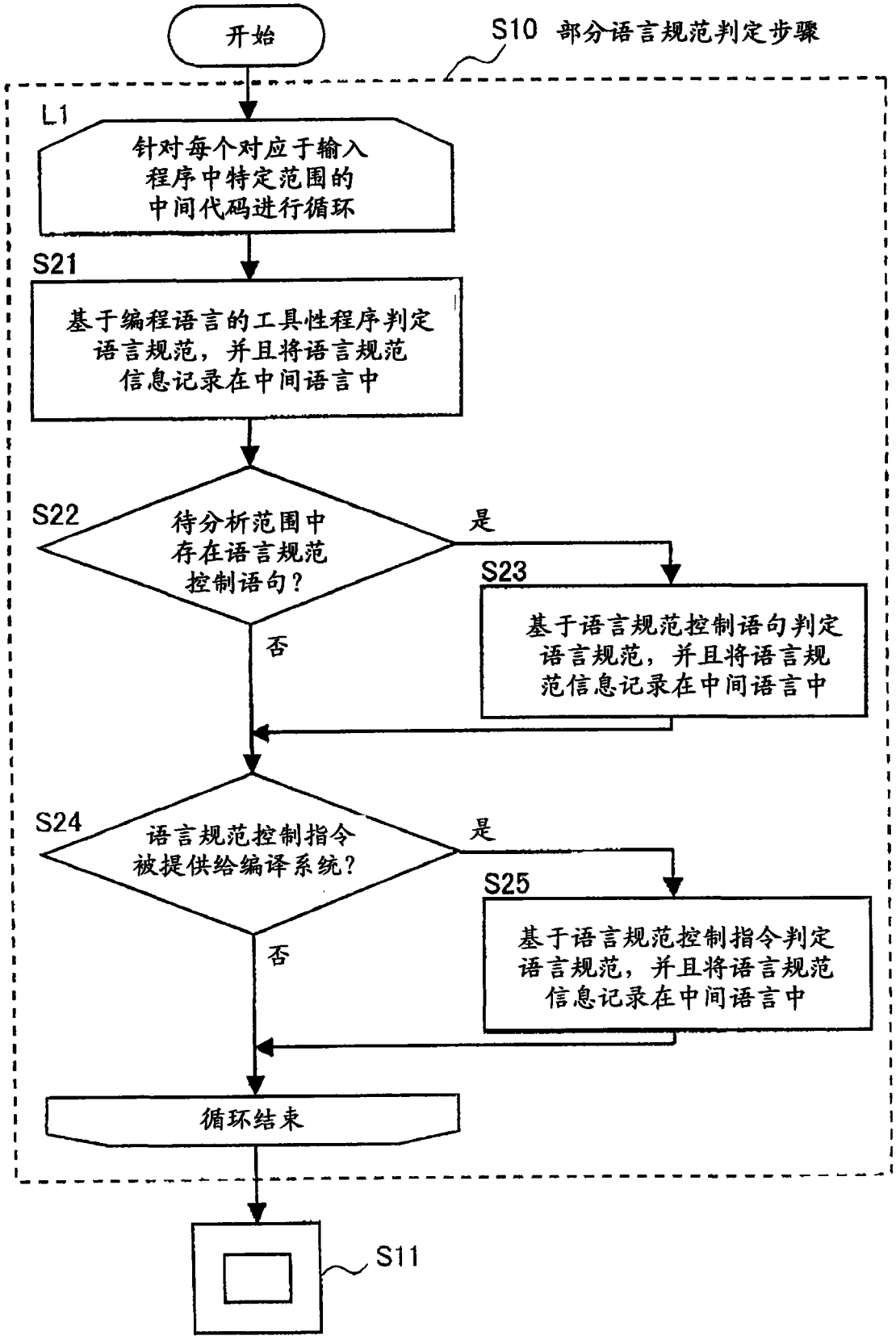

开始
S10 部分语言规范判定步骤
L1
针对每个对应于输入
程序中特定范围的
中间代码进行循环
S21
基于编程语言的工具性程序判定
语言规范，并且将语言规范
信息记录在中间语言中
S22
待分析范围中
存在语言规范
控制语句？
是
否
S23
基于语言规范控制语句判定
语言规范，并且将语言规
范信息记录在中间语言中
S24
语言规范控制指令
被提供给编译系统？
是
否
S25
基于语言规范控制指令判定
语言规范，并且将语言规范
信息记录在中间语言中
循环结束
S11

图 2

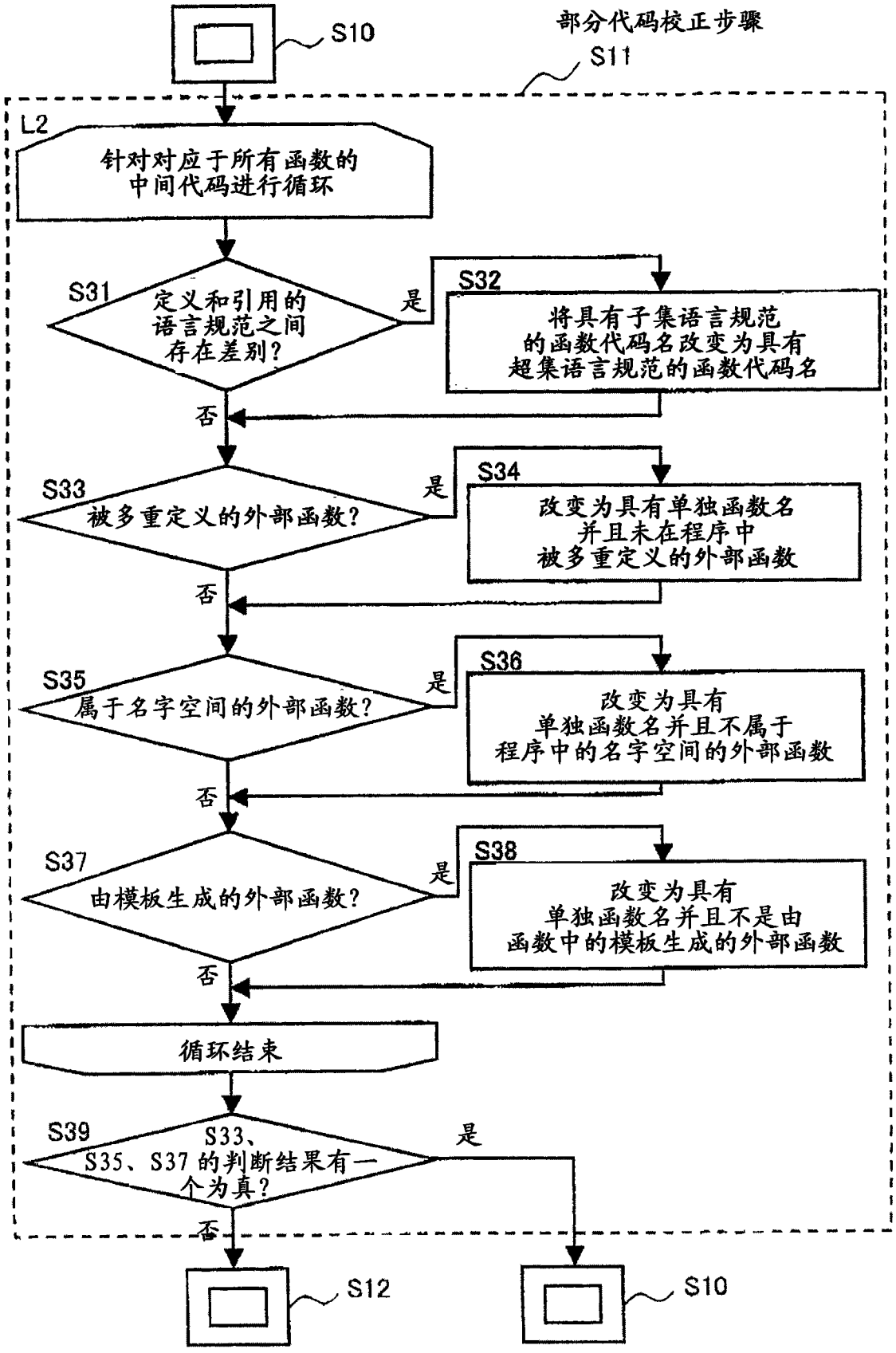

S10
部分代码校正步骤
S11
L2
针对对应于所有函数的
中间代码进行循环
S31
定义和引用的
语言规范之间
存在差别?
是
S32
将具有子集语言规范
的函数代码名改变为具有
超集语言规范的函数代码名
否
S33
被多重定义的外部函数?
是
S34
改变为具有单独函数名
并且未在程序中
被多重定义的外部函数
否
S35
属于名字空间的外部函数?
是
S36
改变为具有
单独函数名并且不属于
程序中的名字空间的外部函数
否
S37
由模板生成的外部函数?
是
S38
改变为具有
单独函数名并且不是由
函数中的模板生成的外部函数
否
循环结束
S39
S33、
S35、S37 的判断结果有一
个为真?
是
否
S12
S10

图 3

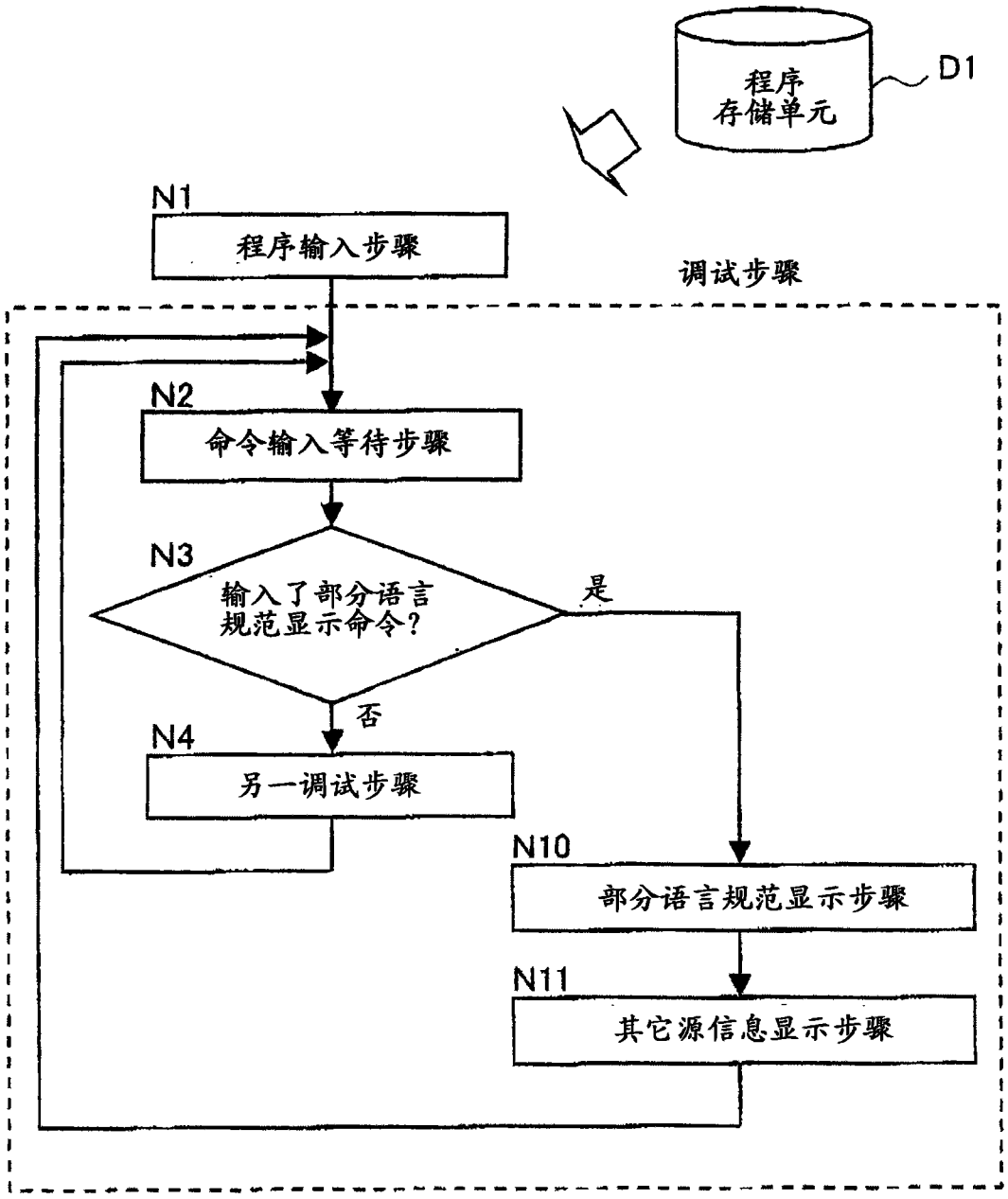

程序
存储单元
D1
N1
程序输入步骤
调试步骤
N2
命令输入等待步骤
N3
输入了部分语言
规范显示命令?
是
否
N4
另一调试步骤
N10
部分语言规范显示步骤
N11
其它源信息显示步骤

图 4

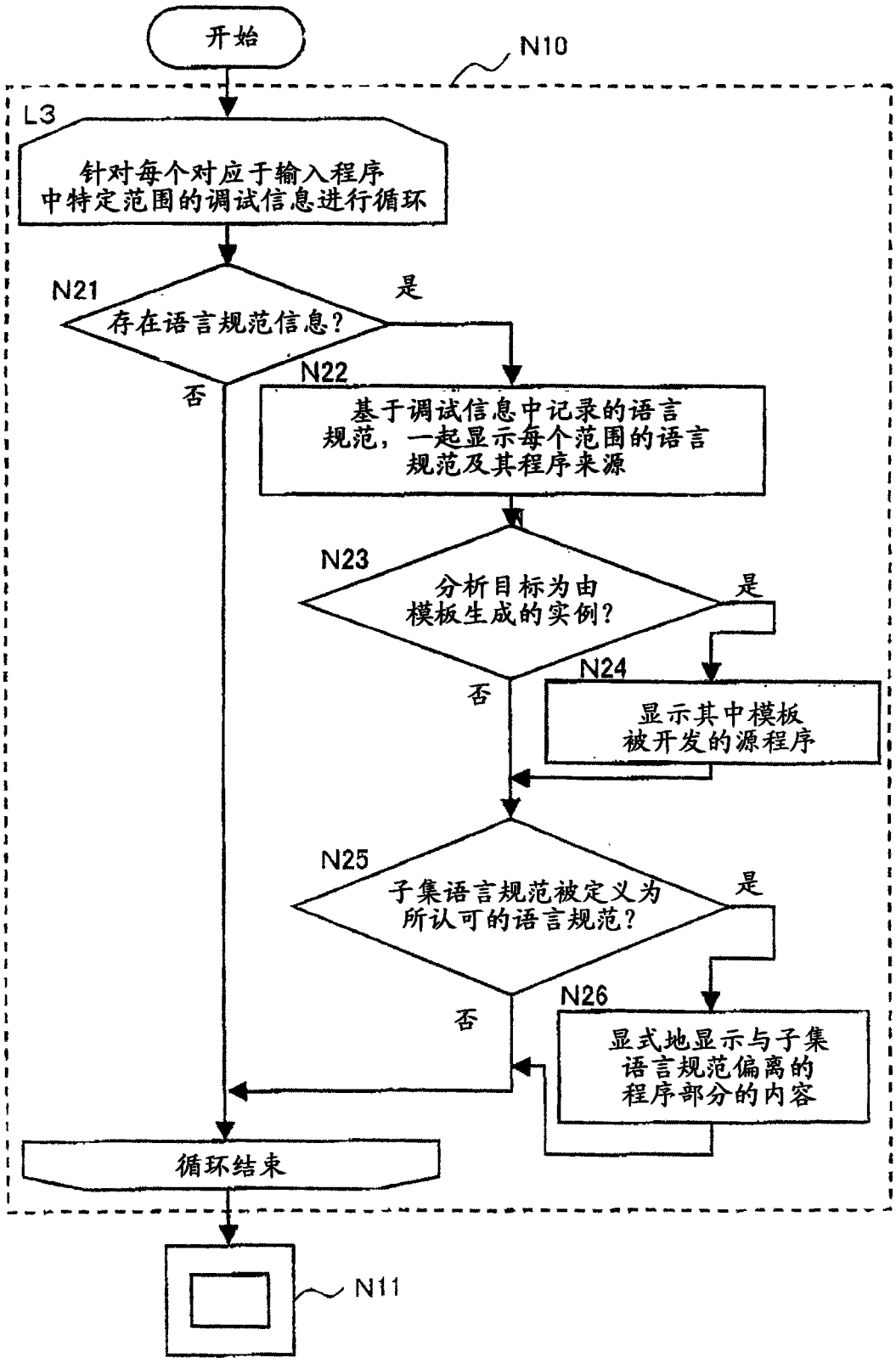

开始
N10
L3
针对每个对应于输入程序
中特定范围的调试信息进行循环
N21
存在语言规范信息?
是
否
N22
基于调试信息中记录的语言
规范，一起显示每个范围的语言
规范及其程序来源
N23
分析目标为由
模板生成的实例?
是
否
N24
显示其中模板
被开发的源程序
N25
子集语言规范被定义为
所认可的语言规范?
是
否
N26
显式地显示与子集
语言规范偏离的
程序部分的内容
循环结束
N11

图 5

<main.cpp>

```
extern void f(void);

int main() {
  //外部函数f()，引用
  f();
}
```

<sub.c>

```
static int count = 0;

//外部函数f()，定义
void f(void) {
  count++;
}
```

图 6

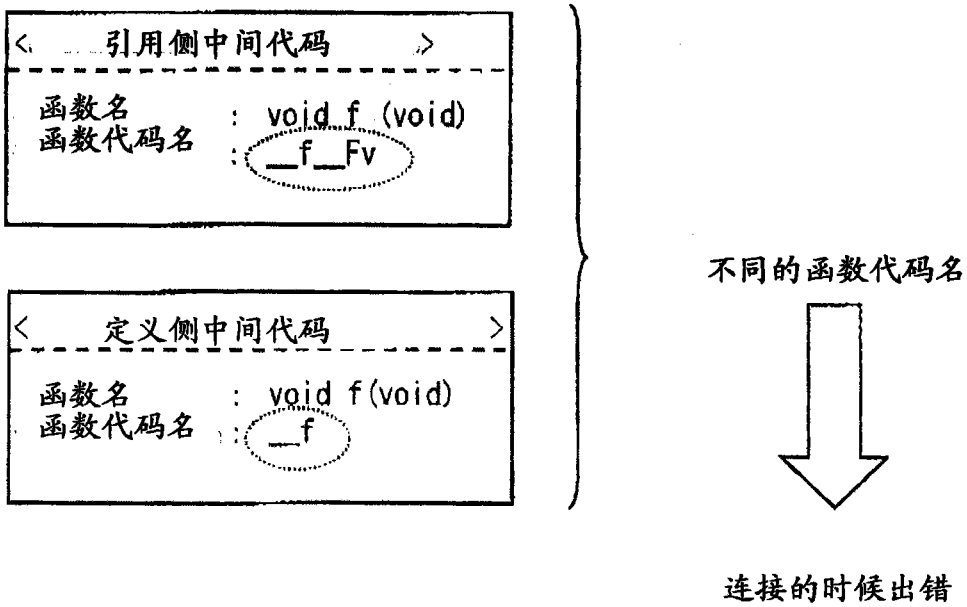



图 7

<main.cpp>

```
// C  //显式指定C的函数
extern( "C" )void f(void);

int main() {
  //  外部函数    ，引用
  f();
}
```

<sub.c>

```
static int count = 0;

//外部函数f()，定义
void f(void) {
  count++;
}
```

图 8

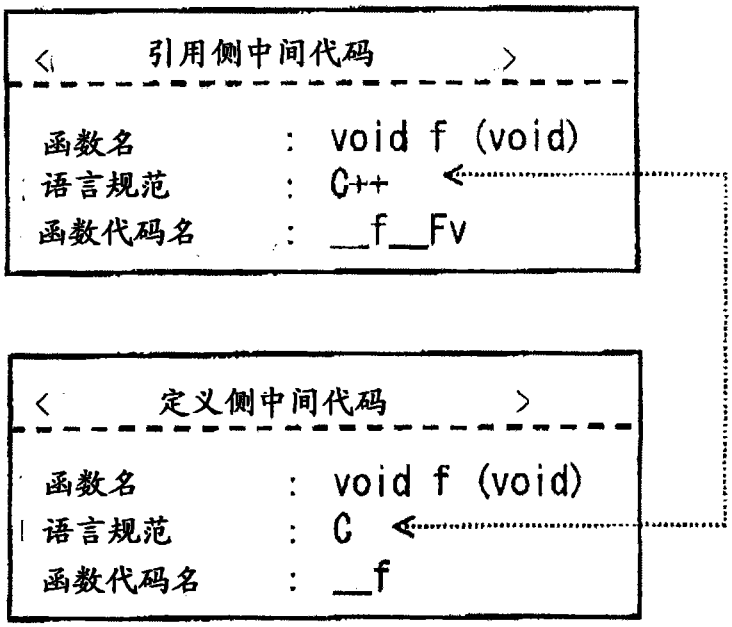

引用侧中间代码
函数名 : void f (void)
语言规范 : C++
函数代码名 : __f__Fv
定义侧中间代码
函数名 : void f (void)
语言规范 : C
函数代码名 : __f

图 9

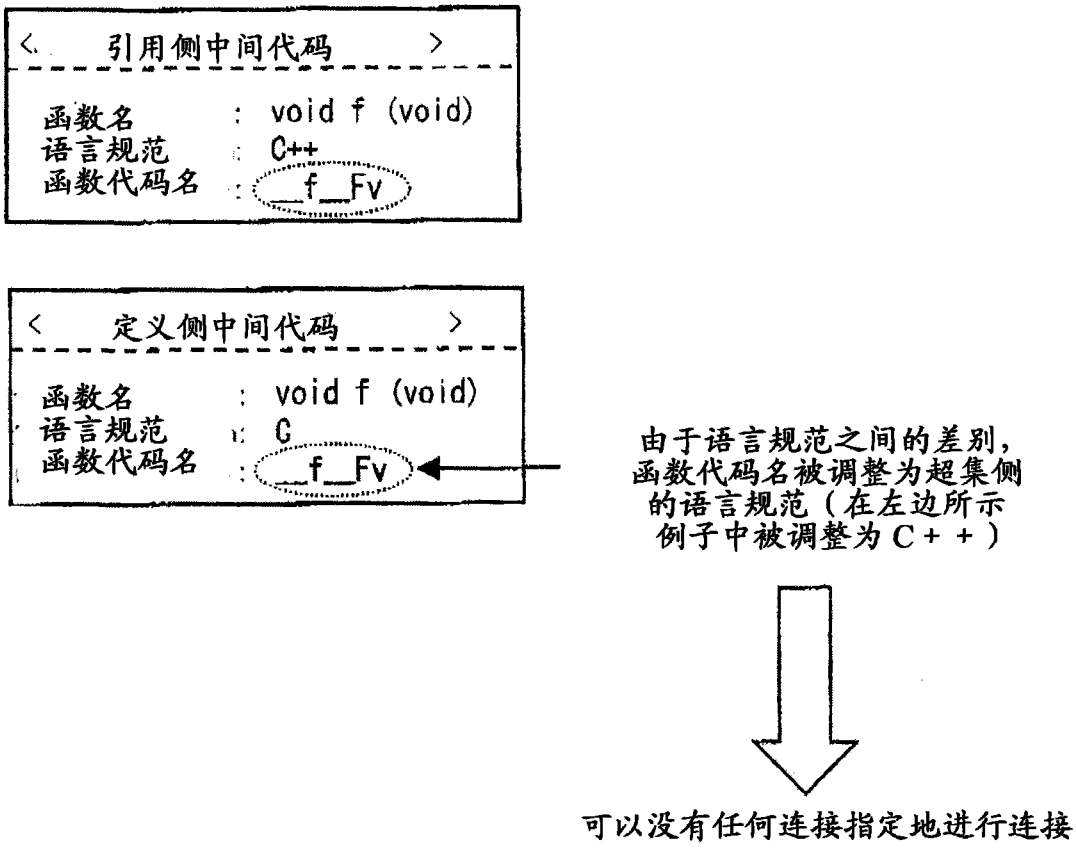

引用侧中间代码
函数名 : void f (void)
语言规范 : C++
函数代码名 : __f__Fv
定义侧中间代码
函数名 : void f (void)
语言规范 : C
函数代码名 : __f__Fv
由于语言规范之间的差别，
函数代码名被调整为超集侧
的语言规范（在左边所示
例子中被调整为 C + + ）
可以没有任何连接指定地进行连接

图 10

<test.cpp>

```
struct A {
  int x;
  int y;
}a={1,2};

//多重定义函数 f 的定义
void f(void) {
  struct A b = {2,4};
  a=b;
}

void f(int x) {
  struct A b = {2,x};
  a=b;
}

void g(void) {
  f();
}
```

图 11

<test.s>按照C++编译

```
...
f__Fv:
.LFB1:
        pushl %edp
.LCFI0:
        movl %esp, %ebp
.LCFI1:
        subl $24, %esp
.LCFI2:
        movl $0, -8(%ebp)
        movl $0, -4(%ebp)
        movl $2, -8(%ebp)
        movl $3, -4(%ebp)
        movl -8(%ebp),%eax
        movl -4(%ebp),%edx
        movl %eax, a
        movl %edx, a+4
        jmp .L3
        jmp .L2
        .p2align 4,,7
.L3:
.L2:
        leave
        ret
.LFE1:
.Lfe1:
...
```

<test.s> 按照C 编译

```
...
f:
        pushl %edp
        movl %esp, %ebp
        subl $24, %esp
        movl $2, -8(%ebp)
        movl $3, -4(%ebp)
        movl -8(%ebp),%eax
        movl -4(%ebp),%edx
        movl %eax, a
        movl %edx, a+4
.L2:
        leave
        ret
.Lfe1:
...
```

图 12

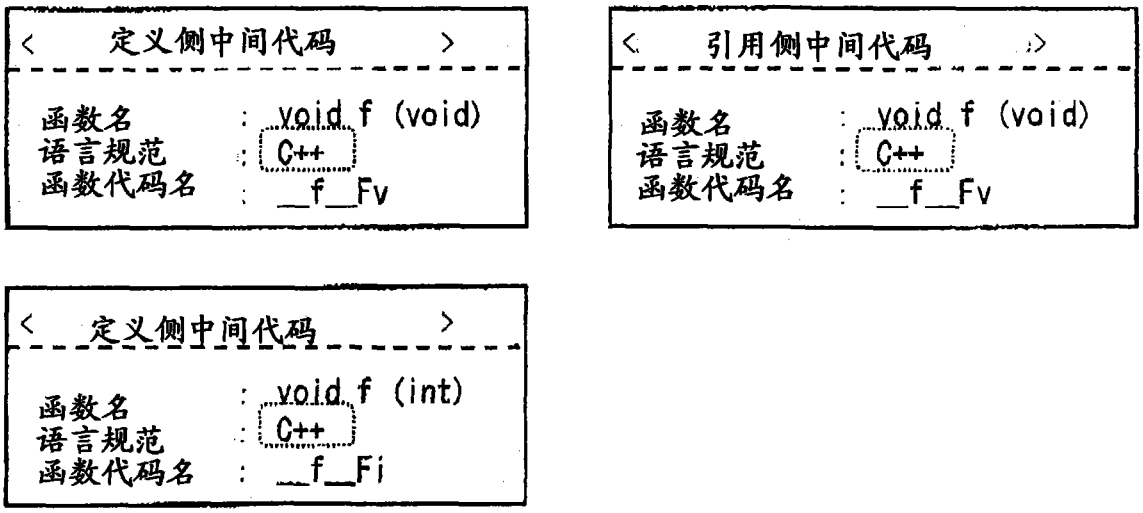

定义侧中间代码
函数名 : void f (void)
语言规范 : C++
函数代码名 : __f__Fv
引用侧中间代码
函数名 : void f (void)
语言规范 : C++
函数代码名 : __f__Fv
定义侧中间代码
函数名 : void f (int)
语言规范 : C++
函数代码名 : __f__Fi

图 13

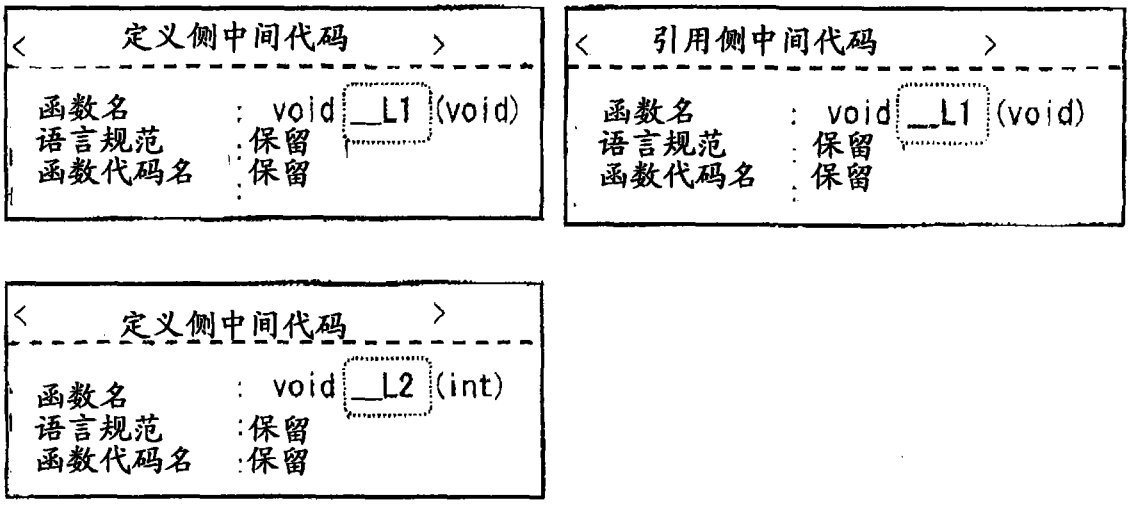

定义侧中间代码
函数名 : void __L1 (void)
语言规范 : 保留
函数代码名 : 保留
引用侧中间代码
函数名 : void __L1 (void)
语言规范 : 保留
函数代码名 : 保留
定义侧中间代码
函数名 : void __L2 (int)
语言规范 : 保留
函数代码名 : 保留

图 14

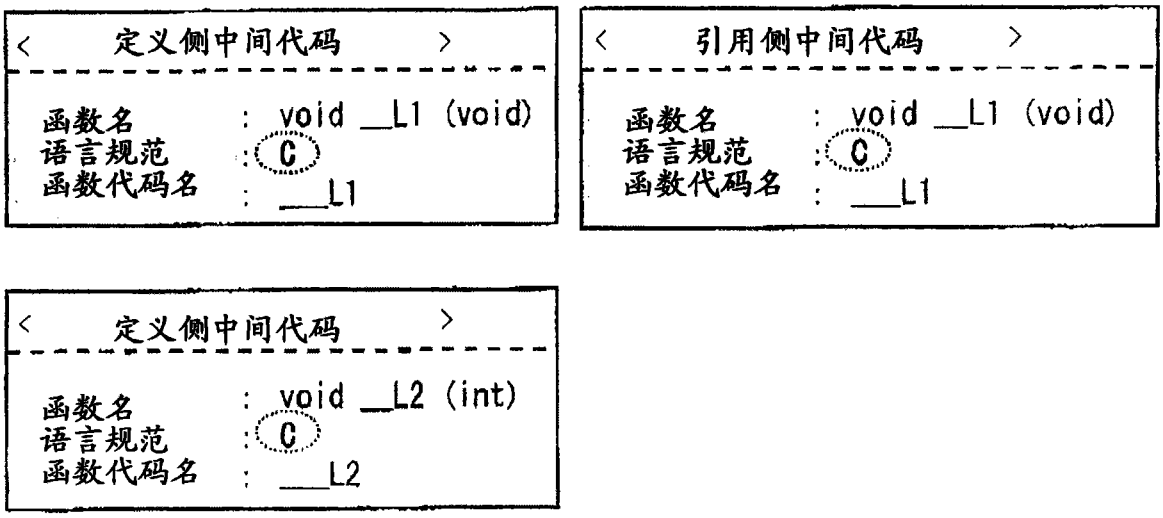

< 定义侧中间代码 >
函数名 : void __L1 (void)
语言规范 : C
函数代码名 : ___L1
< 引用侧中间代码 >
函数名 : void __L1 (void)
语言规范 : C
函数代码名 : ___L1
< 定义侧中间代码 >
函数名 : void __L2 (int)
语言规范 : C
函数代码名 : ___L2

图 15

<test.s> 优化前

```
...
f__Fv:
.LFB1:
            pushl %edp
.LCFI0:
            movl %esp, %ebp
.LCFI1:
            subl $24, %esp
.LCFI2:
            movl $0, -8(%ebp)
            movl $0, -4(%ebp)
            movl $2, -8(%ebp)
            movl $3, -4(%ebp)
            movl -8(%ebp),%eax
            movl -4(%ebp),%edx
            movl %eax, a
            movl %edx, a+4
            jmp .L3
            jmp .L2
            .p2align 4,,7
.L3:
.L2:
            leave
            ret
.LFE1:
.Lfe1:
...
```

<test.s> 优化后

```
...
___L1:
            pushl %edp
            movl %esp, %ebp
            subl $24, %esp
            movl $2, -8(%ebp)
            movl $3, -4(%ebp)
            movl -8(%ebp),%eax
            movl -4(%ebp),%edx
            movl %eax, a
            movl %edx, a+4
.L2:
            leave
            ret
.Lfe1:
...
```

图 16

File: debug.cpp

```
extern void g(void);
extern void h(void);

template <class T> class A {
  T x;
public:
  void mfunc();
};

void f(int x)
{
  if (x < 0) {g();}
  else {h();}
}

void j(void)
{
  A<int> obj;
  ...
}
```

图 17

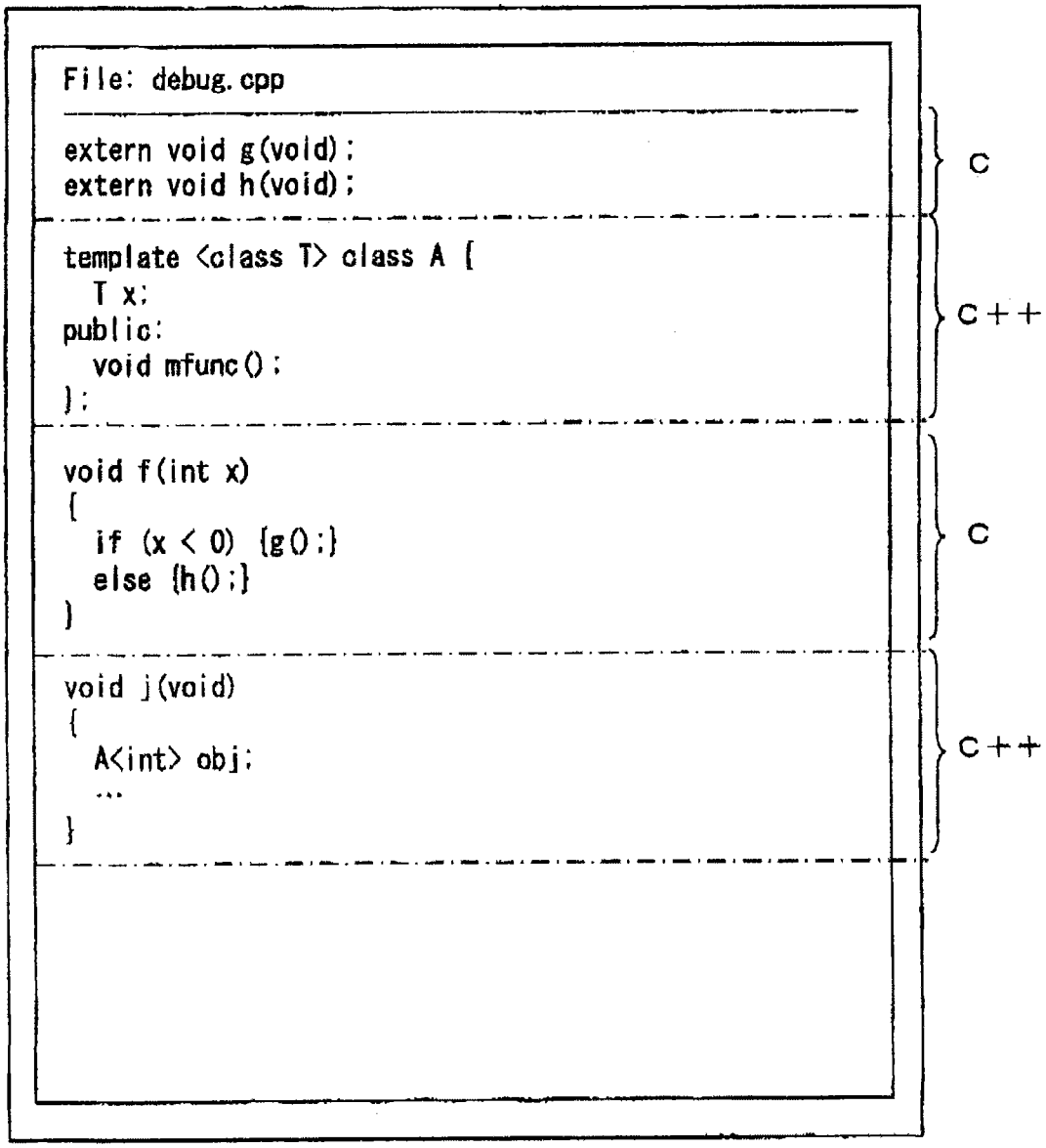

File: debug.cpp
extern void g(void);
extern void h(void);
C
template <class T> class A {
T x;
public:
void mfunc();
};
C++
void f(int x)
{
if (x < 0) {g();}
else {h();}
}
C
void j(void)
{
A<int> obj;
...
}
C++

图 18

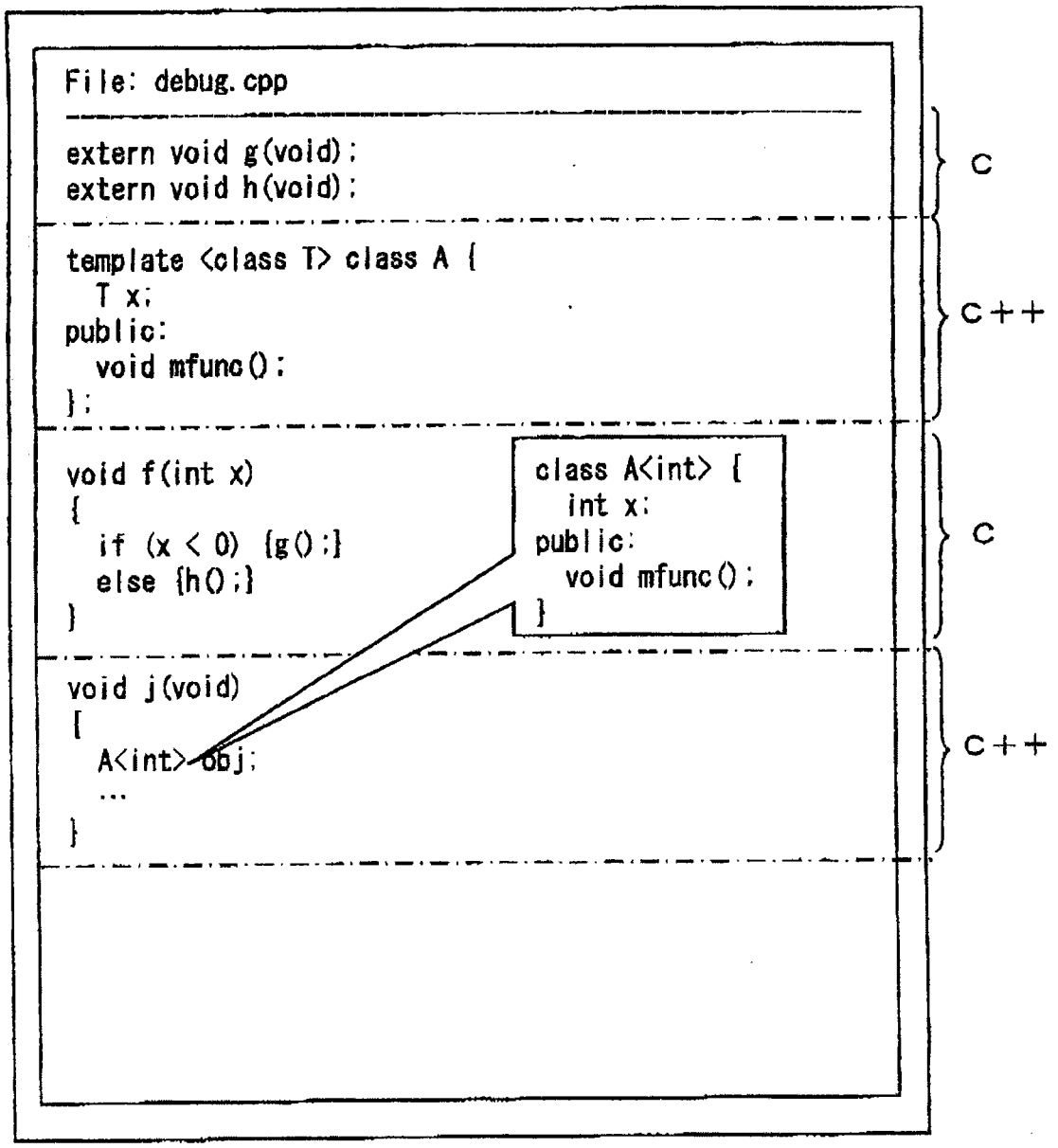

File: debug.cpp
extern void g(void);
extern void h(void);
C
template <class T> class A {
T x;
public:
void mfunc();
};
C++
void f(int x)
{
if (x < 0) {g();}
else {h();}
}
class A<int> {
int x;
public:
void mfunc();
}
C
void j(void)
{
A<int> obj;
...
}
C++

图 19

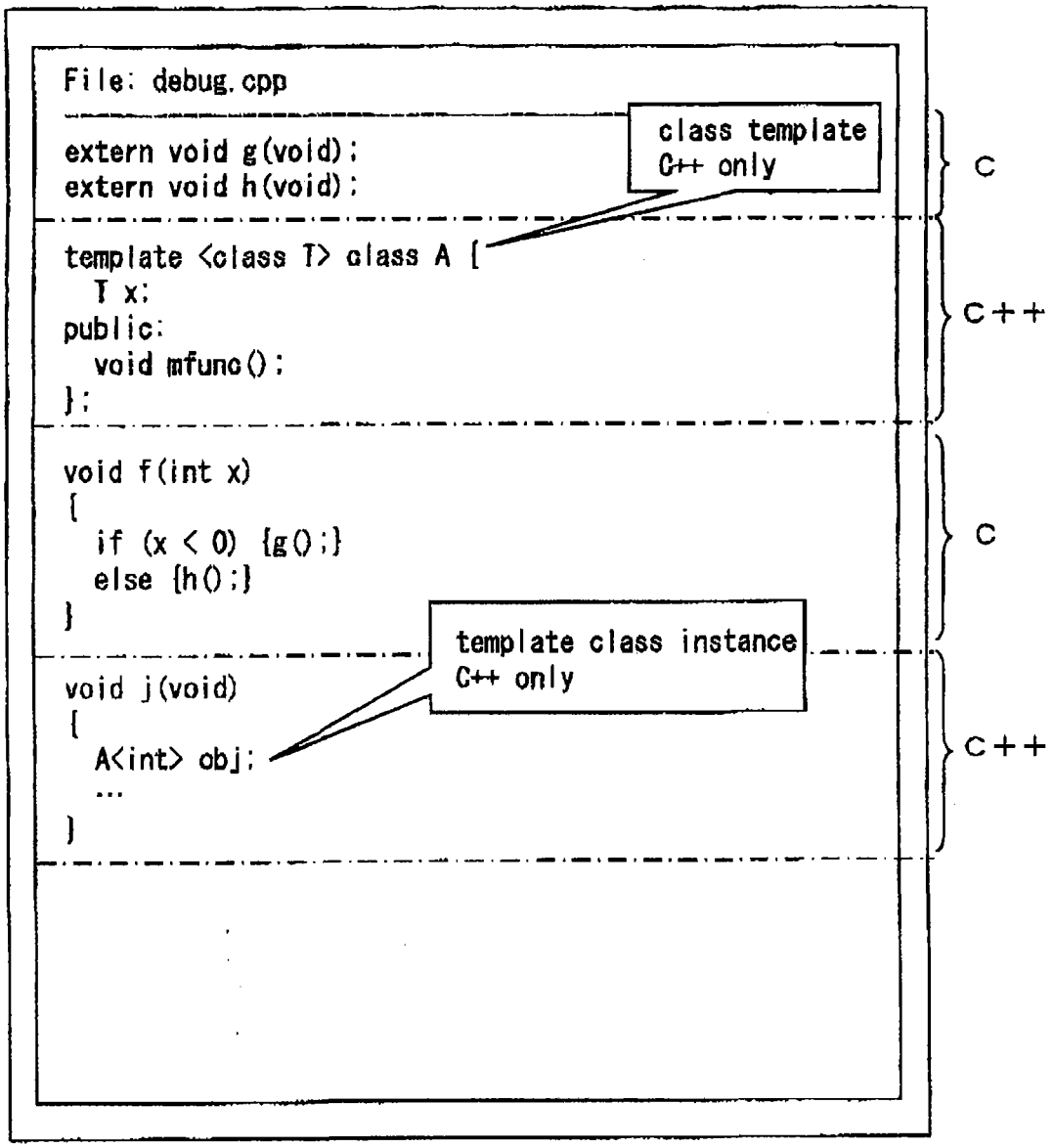

File: debug.cpp
extern void g(void);
extern void h(void);
class template
C++ only
template <class T> class A {
T x;
public:
void mfunc();
};
void f(int x)
{
if (x < 0) {g();}
else {h();}
}
template class instance
C++ only
void j(void)
{
A<int> obj;
...
}
C
C++
C
C++

图 20

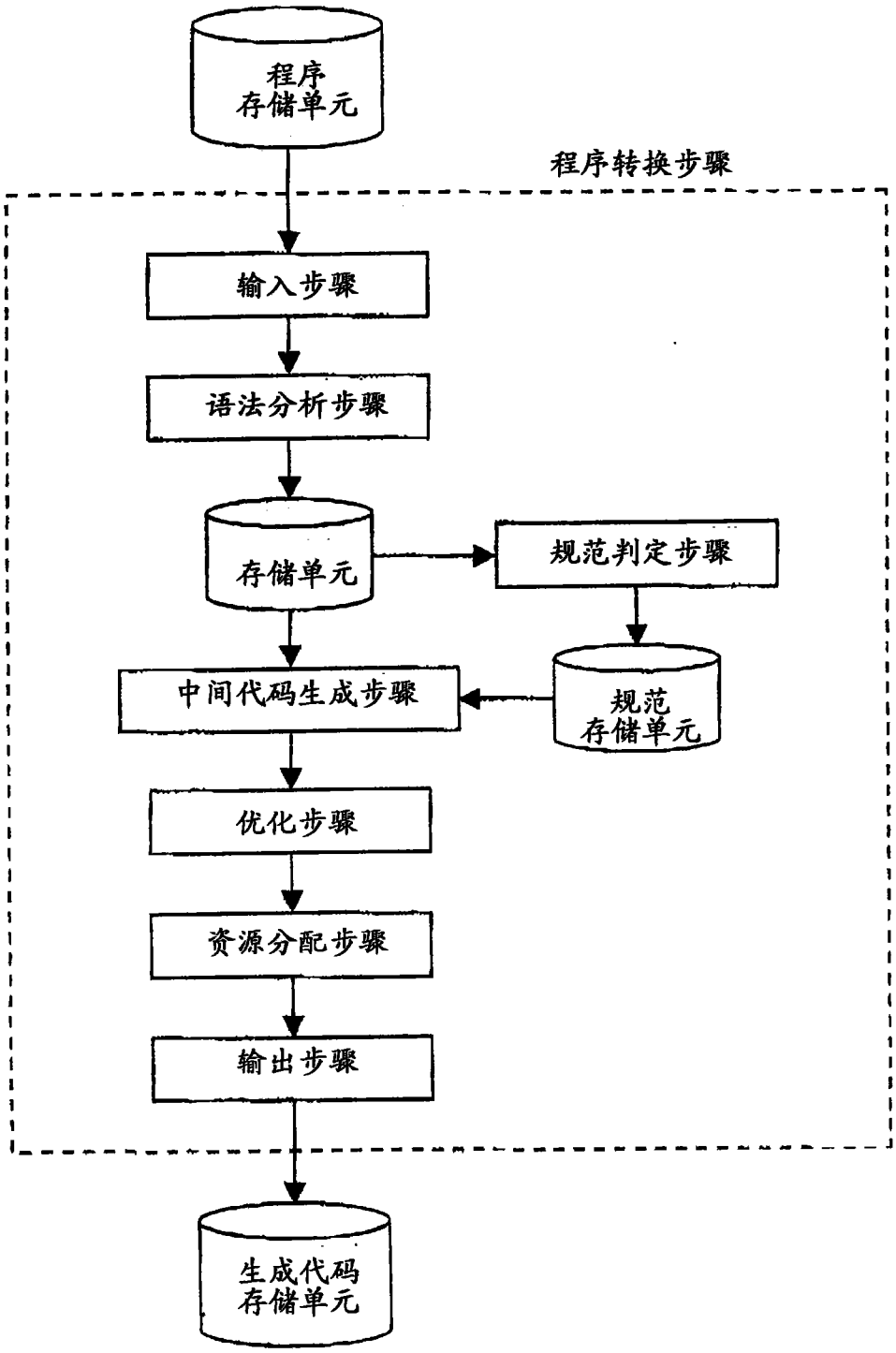

程序
存储单元
程序转换步骤
输入步骤
语法分析步骤
存储单元
规范判定步骤
规范
存储单元
中间代码生成步骤
优化步骤
资源分配步骤
输出步骤
生成代码
存储单元

图 21