《未来程序·改》解题报告

宁波镇海蛟川书院 卢啸尘

1 试题来源

IOI2015国家集训队候选队员互测第四场

清澄题号P7003

2 试题大意

给出一种语言，它是C++语言的一个子集。这一语言只支持int类型，使用cin, cout的输入输出，以及使用putchar函数的输出，并且在语句和运算符种类上有很大的简化。

为该语言编写一个解释器。

第一个点的源代码的全文已给出。

第二个点是只含加减乘除模，没有括号的表达式计算。

第三、四个点是包含了更多运算以及括号的表达式计算。

第五个点中没有除main以外的函数，并且整个程序中只有顺序结构。

第六、七个点只保证没有除main以外的函数。

第八个点中的所有的变量都是全局变量。

最后两个点不保证任何特别的性质。

3 预期得分

为了得到第一个点的分数，选手只需提交给出的源代码的全文。因此所有的选手都能轻易地得到这10分。

为了得到第二个点的分数，选手只需编写一个经典的简单表达式计算。估计几乎所有选手都能得到这10分。

为了得到第三、四个点的分数，选手需要掌握处理有多个优先级的表达式计算的方法，或至少对递归下降分析法有一定的了解。估计只有三分之一的选手可以得到这20分。

第五个点引入了变量以及变量作用域的概念。从这个点开始，选手至少要编写一个标程级别的词法分析器，并且对于变量作用域有正确的管理。估计只有个别选手可以得到这10分。

第六、七个点引入了循环语句。从这个点开始，选手至少要编写一个标程级别的语法分析器。估计在四个小时内没有，如果有的话，至多只有一个选手可以得到这20分。

第八个点是为写了完整算法，但是在变量作用域方面出现了问题的选手准备的；第九、十个点是完整算法。这三个点引入了函数的概念。从这个点开始，选手要支持函数的调用，对return语句的处理方法（在前三个点中执行到return时只需简单的调用”exit(0)”），并且对处理变量作用域有了更高的要求。估计在四个小时内不会有任何选手得到这30分。

综上，这道题对12名集训队候选队员的测试的预期结果是：0到1人70分，0到2人50分，2到4人40分，至少4人20分，0人0分，其他人10分。

4 实际得分

待录入。

5 测试点1的解法

由于第二个整数开始就是program1.in，把附加文件program1.cpp的第255行复制一份到空出来的第254行后直接提交即可。

如果还想做下面几个点的话，可以根据hash值或program1.cpp的一些特点（如最后两行的变量定义形式的注释）来特判出这个点。

除了特判以外，这个测试点的强度比最后两个点的强度有过之而无不及。

6 测试点2到4的解法

在输入文件中找到子串"cout"，定位出所需计算的表达式，即数据规模部分所指的(1)。

接下来就是对(1)做表达式求值。

6.1 测试点2

在测试点2中，表达式只有两种优先级并且没有括号。找到所有的加号和减号，用这两个符号将表达式分成若干项。直接从左到右计算各项的答案即可。

6.2 测试点3、4

由于测试数据是手打的，可以估计表达式的长度不会很长。

递归的做表达式求值即可。函数参数：最高含有的运算符优先级，左界，右界。每次找出最高优先级的那些运算符，用它们将表达式分割成几个子表达式并进入下一层。在最低级，检查两侧是否是括号，如果是的话，调用(最高优先级, 左界+1，右界-1)。

关于如何区分二元运算符（加、减）和一元运算符（正、负）：如果一个运算符处在串头部，或它前面的一个非空字符不是右括号或数字，那么它就是一元运算符。

6.3 进一步地以测试点5为目标时

在上面的程序中，我们再定义cin, cout, endl为常量，在计算双左尖括号和双右尖括号这两个运算的时候进行相应的语义动作（输入或输出）。

那么现在每个输出动作和每个输入动作都可以被当成一个表达式而被计算。这在测试点5中是很方便的。

类似的，赋值号也可以被整合入表达式计算，但是关于这一部分的细节，请参见7.3。

7 测试点5的解法

在这个测试点，变量，变量定义域的概念被引入。考察每个语句，它要么是个表达式（当我们按照6.3中的做法，将输入输出和赋值整合进表达式后），要么是个putchar函数（稍后函数调用也会被整合进表达式），要么是个变量定义，要么是个块。

由于是顺序结构，在我们定义一个“处理一个语句”的程序后，只需要反复调用这个程序直到遇到右括号即可。

在介绍各主分析程序之前，首先将介绍词法分析器。这个部分的引入将简化后面的工作。

之后将介绍主分析程序的大致实现，这个部分分析了“块”这样的程序结构。

之后将介绍变量部分的大致实现。

7.1 词法分析器

在自然语言中有字母（单字）和词语两个概念。将字母组织成词语就是词法分析器的工作。

任何一个掌握了读入优化实现方法的选手都能从这一方法类推得到词法分析器的实现。

具体的来说，就是根据输入流头部的第一个非空白字符类型判断出下一个词的类型，并且读取字符直到遇到并非该类型的字符。

7.2 主分析程序

主分析程序是一个函数。它处理一个块。（注意函数main里也有这么一个块）

主分析程序每次检查输入流头部的一个词语：

7.2.1 如果这个词语是右花括号

右花括号指示到达了这个块的终点。退出这一层程序。

向变量管理器宣称一个变量作用域结束。关于变量管理器的实现将在7.3中进行介绍。

7.2.2 如果这个词语是关键字int

关键字int指示下一个语句是变量定义。读入词语直至分号，中间的词语序列就是需要定义的变量列表。

找出序列中所有的逗号，它将序列分割成若干节，每节的第一个词语是变量名，后面的是这个数组的规模。

在变量管理器中注册这些变量。

7.2.3 如果这个词语是左花括号

左花括号指示一个新的块的开始。进入新的一层主分析程序。

向变量管理器宣称一个变量作用域开始。

7.2.4 如果这个词语是预置函数标识符putchar

提取括号内的那个表达式，用前面的表达式求值程序对其求值，输出得到的值对应的字符。

7.2.5 其他情况

扫描直至遇到分号。使用表达式求值程序对其求值。

7.3 变量部分

7.3.1 如何将赋值号整合入表达式计算

题面中已经给出了，赋值号是一个优先级介于逻辑运算和输入输出符号之间的运算。因此在6.3的基础上只要增加一级运算就可以了。

主要的问题是如何设计其语义动作。

我们定义一个新int。它是一个布尔型和一个int型组成的结构体。当这个布尔型为真时，int保存的是这个值在内存中的地址。内存可以用一个数组来模拟。当布尔型为假，说明这个新int是由两个值运算得来的，它在内存中没有位置。

那么只要保证等号的左边是一个有地址值就可以设计出其语义动作了。由于程序保证合法所以这一点是始终成立的。

7.3.2 变量管理

使用一个map<string,vector<newint>>下简写为”MSVI”来保存各变量的历史地址，用一个vector<vector<string>>下简写为”VVS”来记录各作用域中定义了哪些变量。

在声明变量var时：向VVS的back压入var，向内存管理器申请内存，将得到的地址压入MSVI[var]的底部。

在寻找变量var的地址时，只需返回MSVI[var]的底部即可。

在声明新变量作用域时，将一个空白vector压入VVS即可。

在离开变量作用域时，将VVS底部vector内的所有元素在MSVI中的对应vector的底部弹掉即可。

请注意，这一实现只是针对本测试点中只有主函数的情况设计的，当出现其他函数时情况将变得更复杂。举例：有全局变量x，main中定义了局部变量x，main了调用函数a。那么，在函数a中寻找x时应当返回全局变量的x，但在这种实现下就会返回main中的局部变量x。

7.3.3 内存管理

由于没有动态开内存，因此只需要栈就可以描述内存结构了。

使用一个大数组来模拟内存即可。

关于如何模拟数组，以下将给出一个示例：

申请a[2][3]前的第一个空地址在80。给[80]赋值81，给[81]赋值83，给[82]赋值86，初始化[83..88]。现在要定位a[1][1]：a的地址是80，从中得到a[x]的初始地址为81，得到a[1]的地址为81+1=82。类似地得到a[1][1]的地址在86+1=87。

由于各维的规模至少为2，这种数组模拟方法只需要两倍的空间，即16M即可。

7.4 还有什么遗漏的?

putchar函数是有返回值的。它也应当被整合到表达式计算中。(虽然第5个测试点内没有这种情况。)

请参见后面的章节。

8 测试点6、7的解法

在这个测试点，循环语句被引入。

循环语句本身是容易解释的，但是直接解释的效率是不高的。因为这样一来，一段源代码可能被扫描很多次，然后对源代码进行的分析也会进行很多次。

出题人并没有测试过完全采取解释方法的效率，但是考虑到解释方法一般比编译方法慢数十倍，而标程在后几个数据上的运行时间在0.015s规模，解释方法的时间可能会比较紧张。

在这两个点中，选手应当尝试对源代码只做一次解码，在一次解码后转换为一种容易解码的中间表示形式，如语法树，然后后续的解释在语法树上执行。

在题面中，已经给出了描述此语言的上下文无关文法。读者可以利用这一文法为源代码建立具体语法树，并为其设计适当的抽象语法树。这里不再详细说明其细节。

9 测试点8到10的解法

在这三个点中，函数被引入。

9.1 分析解释部分

一般而言，在抽象语法树上，函数是根节点的一个子节点。

那么对函数的调用就是对一个子节点的执行。

在解释的版本下，就是执行某一个块。

函数参数可表示成函数体初的一系列变量定义和变量赋值语句。

9.2 变量管理部分

我们考察函数存在下的作用域情况。

如果在当前这层函数的各层作用域中有至少一次定义了这一变量，那么就取最近的一层中的定义，否则取全局变量中的定义。

那么在变量管理器中，在作用域级之上增加一个函数级。使用vector<multiset<string>>来保存每层函数中某变量是否有定义即可。

9.3 关于return

在同一时刻最多只有一个return语句在计算完毕执行前的状态。使用全局变量记录即可。