编译原理实验报告

熊永琦 16340258

2018 年 12 月 26 日

本项目所有代码均已在 我的 Github 仓库 开源

1 实验内容

1. 在计算机上实现 PL0 语言的编译程序
2. 扩展 PL0 语言的功能, 并在计算机上实现

1.1 实验一要求

1. 在 PASCAL 系统上运行 PL0 编译程序
2. 在 PASCAL 系统中, 为 PL0 的编译程序建立输入文件和输出文件
3. PL0 的编译程序运行时, 通过输入文件输入 PL0 源程序, 在输出文件中产生源程序的中间代码, 然后运行该中间代码, 在输出文件中产生运行数据

1.2 实验二要求

1. 在 PL0 语言中增加 Read 和 Write 语句
2. 修改 PL0 编译程序，使得 PL0 源程序可以使用 Read 和 Write 语句，从文件 (或键盘) 输入数据，并可以向文件 (或屏幕) 写数据

2 实验一报告

2.1 令编译程序正常运行

首先，我们修改了 PL0 编译程序，使得其能够通过 Free Pascal Compiler 的编译并正常运行。为了达到这个目标，我们订正了 PL0 中多处语法错误，并且改换了部分 PL0 操作

符，使得其源程序和编译器的代码能够使用标准 ASCII 编码空间覆盖。在编译程序中主要存在以下几种错误：

1. 变量命名混乱：许多变量直接使用了 Pascal 语言的保留字作为变量名，导致编译时出现语法错误。

2. 使用不合法的 ASCII 字符作为 PL0 逻辑操作符：编译程序选用 $\geq$、$\leq$、$\neq$ 作为 PL0 中的逻辑判断符号，由于 FPC 不能够识别这些非标准 ASCII 的字符，从而产生报错。

3. 不合法的跨子程序跳转：当读取数据产生错误时，编译程序会尝试跳转到 Label 99 的代码段来进行错误处理，然而这种跨子程序的跳转是不合法的，无法通过 FPC 编译。

对于上面三种问题，我们首先使用 &、% 两个符号替代了小于等于号和不等号，从而在保证使用单个字符作为操作符的前提下完成了 PL0 的逻辑操作符实现。然后我们将所有保留字的变量名替换为带有 m_ 前缀的变量名，从而规避了命名冲突带来的错误。最后，我们将 Label 99 处的代码段替换到所有跳转到它的代码中，就规避了最后一个问题。至此，编译程序已经可以正常运行。

2.2 编译程序的文件读写

原本的编译程序通过键盘写入源程序，并且将编译结果直接打印到屏幕上，这使得我们在进行代码文件的编译和结果验证的时候遇到了许多不便，因而添加一套文件读写系统对于编译程序是有必要的。我们首先在全局变量生命了四个文件（分别是源程序输入、源程序输出、中间代码输出、运行时结果输出）路径和四个文件类型，然后我们在运行时要求用户通过键盘输入四个文件的路径来指定这四个路径。

在程序开始运行时，我们通过 assign 命令打开这几个文件，并通过 reset 和 rewrite 命令初始化这四个文件。然后我们将所有的 read 和 write 由键盘/屏幕改定向到文件，就实现了文件读写部分的编码。关于此部分的代码和程序的输出结果，您可以在 我的 Github 仓库 里找到，他们分别是 os、oi、or 三个文件。

3 实验二报告

3.1 为编译程序添加键盘读、屏幕写命令的支持

为了在 PL0 中使用 I/O 的功能（而非单纯执行写死的程序），我们需要首先修改编译程序，使得其能够正常编译和解释执行带有 I/O 代码的 PL0 程序。我们首先定义两个函数 Read() 和 Write()，他们都接收不定数量的变量作为参数。其中，Read 将会接收键盘的数个输入，并将他们的值赋予给参数表内的变量；Write 则会将参数表内变量的值打印在屏幕上。

为了实现对这两个函数的支持，我们首先需要在保留字枚举和函数枚举中添加相应的值。然后，我们在字典、词典和助记符表中分别加入这两个函数的对应字符串。这样就可以在 getsym 时正常解析到这两个函数了。接下来，我们改写句法分析和翻译部分的代码，在 switch case 中加入这两个语义的判断并执行相应的操作即可。

3.2 为 PL0 程序添加键盘读、屏幕写功能

只需要简单地将原本的赋值改为 I/O 语句即可。我们在这里使用 Read(x) Read(y) 来从键盘获取两个值填入到变量中，使用 Write(x, y, z) 来将输入的值和结果值输出。这一部分的代码和输出结果均可以在 我的 Github 仓库 里找到。