编译原理实验汇报

徐煜森 PB16110173

第一部分：小组合作

一． 简述实验内容

1. 实验 1

实现 PL0 词法分析器，可以识别出 PL0 的四种 token 类型（忽略掉注释），并在类型 PL0Lex 中保留 token 的相关信息。四种类型分别为标识符 identifiers、数字 numbers、符号 symbols、保留字 reserved word。

2. 实验 2

根据语法图写出 PL0 的 LL(1)文法，使用递归下降的方法实现 PL0 基础的语法分析，并建立分析树和分析栈，进行初步的错误处理。

3. 实验 3

增强 PL0 语法分析器，在实验 2 的基础上增加了实验 3 要求中的所有扩展项：增加数组、逻辑运算符及推广条件表达式、增加返回值、do-while 循环、elif/else 分支语句、增加参数传递、引入负数。其中一部分内容需要修改之前的文法。

4. 实验 4

增加 print 语句，最终实现可将 PL0 源代码编译成可在助教提供的虚拟机上执行的汇编代码的编译器，得到满分 156 分。

二． PL/0 编译器总体设计与实现功能详细介绍

1. 实验 1

充分利用 ctype.h 库中函数辅助进行判断字符的类型。当读到一个字符时对其类型进行判断：

若是空格类型或换行符则跳过；

若是注释开始符号则跳过注释；

若是下划线或字母则先对保留字进行匹配，若匹配失败则匹配标识符，将标识符存在 lex->last_id 中；

若是符号的开头字符则按照最长匹配原则，先匹配长为两个字符的符号，若失败再匹配长为一个字符的符号；

若是数字开头则匹配数字并分析出数字的值，存储在 lex->last_num 中。

按照以上各类型，设置 lex->last_token_type。

2. 实验 2 和实验 3

首先设计文法，需要注意的是设计文法一定不要有左递归，在实验 3 中一位同学设计的文法有一个隐藏了很多层的左递归，等到我写完相关代码之后才发现，最后重新修改文法又大改代码。

之后将文法编码进程序，设计分析栈和分析树。其中分析栈中存储的是文法符号，对外部提供接口有 stack_pop（参数为栈指针）和 produce 函数（参数为栈指针和产生式编号）。Stack_pop 函数用于弹出已分析结束或在分析中出错的文法符号；produce 函数用于将栈顶非终结符按某一产生式进行推导，将产生式右边的符号反向进栈，按照实验要求每进行一次推导打印一次栈的内容。

分析树的实现需要用到 mermaid 来画图，建树的方法是在每次调用 produce 时为产生式右边的文法符号建立新节点，作为产生式左边的文法符号的儿子添加到树上。

然后按照文法编写递归下降的分析程序，在这里需要提出一点：虽然课本上是为每个非终结符单独编写一个分析函数，但是这样写的话在翻译一些循环文法的时候会出现一个很麻烦的问题，就是需要在函数间传递大量的参数，增加翻译方案的设计难度。因此，将原先一部分循环文法（如变量常量定义、表达式）的递归分析展开写成 while 循环的形式，有利于之后的翻译方案的设计。

最后，在递归下降分析中遇到错误时，打印错误信息（错误发生位置和可能的错误原因），并根据具体情况 pop 出多余的文法符号从错误中恢复，继续分析。

在实验 3 中实现了所有扩展功能。首先修改实验 2 中文法，为所有扩展项写新文法。文法写成之后发现逻辑表达式的文法有隐藏了很多层的左递归，因为层数太多所以展开困难较大。写出的其他文法又无法支持对逻辑表达式使用括号改变运算次序，最终解决方案是仅对逻辑表达式使用 SLR 分析法，即递归下降在遇到逻辑表达式时，调用 SLR 分析，分析结束后再跳回至递归下降分析。因为一个非终结符的推导和它的上层推导没有关系，所以这样做是可行的。

3. 实验 4

实验 4 中实现了所有功能的翻译，最终得到 156 满分。

首先为每一个产生式设计大致的翻译方案，之后定义运行时内存栈的管理方法，若下图所示：

```
MEM[0]: SP 栈顶
MEM[1]: BP 控制链
MEM[2]: ACS 当前访问链
MEM[3]: TMP 临时寄存
MEM[4]: TMP2
MEM[10]: trash bin

    /\
   /||\
  //||\\
 // || \\
    ||
    ||
    ||
    ||
   STACK

----------------
局部变量
----------------
控制链
================(之上为被调用者责任，之下为调用者责任)
访问链
----------------
返回地址
----------------
参数
----------------

sp指针指向栈顶元素
```

栈从高地址向低地址生长，其中有特殊用途的是：

MEM[0]为栈顶指针 SP；

MEM[1]为当前控制链 BP；

MEM[2]为访问链 ACS；

MEM[3]、MEM[4]为临时变量，主要用于交换运算栈顶和运算栈顶下一元素在运算栈中的位置。

MEM[10]为回收站，因为虚拟机没有提供运算栈的 POP 指令，print 语句打印后的值不再使用但依然留在栈顶，使用 ST 10 指令将其出栈。

每个过程的活动记录中的参数、返回地址、访问链由调用者处理好进栈，控

制链和局部变量的声明由被调用者处理。主函数没有参数和返回地址。过程的返回值与表达式的值处理方法一样保留在运算栈顶。过程的参数由调用者计算后反向压入栈中。

访问链和控制链的作用与课本一致，唯一的区别是在变量寻址时使用的是相对于访问链的偏移地址，而不是相对于控制链的偏移。

然后设计代码链表，包含代码和相关信息（如代码链表的长度），链表中节点按顺序记录翻译的指令（和相应的参数）。对外接口有 code_append（code_append_arg）函数和 code_extend 函数。其中 code_append 函数是在一个代码链表末尾插入一条指令，code_append_arg 是插入一条带参数的指令；code_extend 函数是将两个代码链表合并，其中一个接在另一个的末尾。

设计过程标识符表和变量常量标识符表。过程表中每一项记录过程的 id、定义该过程的过程在表中的位置、该过程定义的局部变量占用的空间、层次和入口地址。变量常量表中每一项记录 id、维数（如果是数组则为数组的维数，否则为 0）、每一维的大小、定义该变量或常量的过程在过程表中位置、是否是常量（BOOL 类型）、层次、常量值（若为常量则有用，否则无用）、相对访问链的偏移地址和储存动态数组长度的 id 在表中的位置。

两个表分别对外提供接口有 table_append 和 look_up。其中 table_append 函数用于记录每个常量变量或过程标识符的信息。Look_up 函数用于查找某一标识符并返回其在表中的索引。

设计在函数间传递的参数 Attr，其中包含最终翻译的代码链表、当前翻译的过程在表中位置、当前翻译的过程所声明的局部变量占用的空间大小（也用于记录调用过程时传递的参数个数)、过程开始地址和一些用于小范围使用的临时变量（包括一些数值的传递和某一标识符在表中位置)。

最早对分支和循环语句中 JMP 和 JPC 指令翻译的是基于当前指令的偏移地址，翻译结束后再将偏移地址换算为绝对地址输出。但是这种翻译的鲁棒性不够，因为可能会在翻译中引入空指令，而测试中发现助教提供的虚拟机不支持指令中有空行的情况。然后就实现了类似标号加回填的技术来计算地址，先给一个指令打上标记（tag)，等到知道跳转地址后将标记与相应的地址对应起来。

三． 小组成员之间的合作形式

实验 1 是每人都做了一个，之后的每个实验都采用相同的合作形式：实验布置下来后，先三个人一起进行讨论：商量出大致的策略；想清楚一些思考难度较大的细节。之后分工完成自己的部分，完成后合并代码进行调试。

四． 小组合作效果

总体来讲小组合作的效果还是不错的，每个人都能按时完成自己的任务，小组开小会时都在认真听取他人的意见并贡献出自己的见解。不过也有美中不足的地方：同学对自己任务时间的安排有些不妥，之后的几次实验基本都开始的很晚，

不过最后也能按时交上。我个人更习惯将所有工作向前推，给后面留充足的时间调试或改进。

第二部分：个人任务

一． 主要任务及完成过程

1. 实验 1

实现实验 1 中的全部要求，包括对各种词法单元的识别读取、忽略注释和捕获词法错误并报出错误信息。

实验 1 中主要繁琐的地方在对字符的判断和处理上，经过队友提醒发现 ctype.h 库中有十分好用的函数来帮助判断字符的种类，使用库函数使得代码更加清晰。之后需要注意的是一些细节：如匹配保留字和标识符时应先匹配保留字，匹配失败再匹配为标识符；匹配符号时应按照最长匹配的原则，先匹配两个字符的符号，再匹配一个字符的符号。

2. 实验 2

2.1 通过明确合理的分工来减少可能出现的 merge 冲突；规范函数调用操作。

如在递归下降调用函数前，PL0lex 中保留的一定是该函数所分析文法符号的 First 集合中的终结符。原因是有些产生式进行推导时需要向前看一个符号，而另一些无需向前看即可完成推导，这样规定可以让所有推导统一向前看一个字符，避免调用他人编写的函数时出现问题。

2.2 语法分析中使用 LL(1)文法，根据语法图写出文法产生式。

其中我完成了程序、程序体相关的产生式，并帮助队友修改语法、语句序列和条件的产生式，最后大家一起核对文法。

2.3 实现程序体中变量定义 var_declaration 函数、子程序定义 procedure_declaration 函数、条件 condition 函数、表达式 expression 函数，项 term 函数、因子 factor 函数。

其中 var_delaration、expression、term 函数中尽量避免使用递归下降，而是将递归展开为循环，这样做的好处是在翻译时可以避免在递归下降分析时函数间传递大量的参数。

2.4 代码合并后进行测试，通过打印出的每一步的栈的状态定位 bug 位置。

2.5 使用助教提供的 test_err.pl 源代码测试错误处理能力，并改进程序使之能正确报出错误并打印出规范的错误信息。

3. 实验 3

3.1 修改实验 2 中文法，满足实验 3 附加的要求

增加数组、if-elif-else 的文法，增加逻辑运算文法，并将原条件文法改为满足推广条件含义（表达式非 0 即为条件真）的文法。

3.2 修改词法分析器

增加保留字 else、elif、endif，符号&&、||、!。同时找出并修复了队友词法分析中潜藏的小 bug：在头文件中宏定义了 NSYM 为符号的种类数，队友在自己的代码中使用了数字而非 NSYM。

3.3 实现增加的新文法

按照新文法实现 if-elif-else、增加逻辑运算符并推广后的条件。完成后发现之前写的条件的文法有一个隐藏了多层的左递归，修改文法后重新实现文法分析。

3.4 提供在递归下降分析中穿插进行 SLR 分析的思路

在与队友讨论过程中发现，我们的条件文法无法处理使用括号改变||、&&运算顺序的情况，最终给出的解决方案是在递归下降中穿插进行 SLR 分析，此项工作的实现由队友完成。

4. 实验 4

4.1 设计翻译方案和运行时栈用法

此项工作耗时最长，耗时 4 天，参考课件与课本，最终设计出一套完整的翻译方案和运行时栈的使用方法。设计期间在访问链控制链建立和过程调用返回上耗时最长。最终明确参数传递、存储返回地址、建立访问链为调用者的责任（这点与课本课件不同），建立控制链为被调用者的责任，过程返回时返回值存在运算栈的栈顶。另外也参与设计代码链表，提供一些实现上的思路。

4.2 设计并实现变量常量表和过程表

设计实现变量常量表和过程表及其相关的 append 和 lookup 函数。与队友多次讨论后，因为课件上方法不易实现又不易使用，决定更换实现思路，对每个标识符（变量常量过程 id）记录其在哪一过程中定义，这样做的好处是过程表和常量变量表都只有一个，方便实现也方便 lookup。

4.3 实现变量常量声明、循环结构和传参调用等一系列的翻译

按照翻译方案及约定好的代码风格，完成翻译。之前的工作做完了，这一步就相对简单了，按照之前想好的步骤一步一步实现即可。

4.4 合并后参与调试代码。写完所有内容后，经过一天的奋战，终于按时将代码调试完毕，通过测试。

二． 遇到的困难及解决方案

1. 安装 vscode 遇到问题（实验 1）

按照博客中的方法先安装 Ubuntu-make 用于管理此类环境，在使用 umake 命令安装 vscode。在最后一步报出环境不兼容的错误。

解决方案：在网上搜索这条错误信息，在一个角落发现一篇博客中写道：微软官方建议 Ubuntu/Debian 操作系统直接从 vscode 官网上下载.deb 安装包并解压完成安装。尝试这种方法后成功。

尽信博客不如无博客，安装前可以先看一下官方建议。

2. 递归下降难以处理推广后的条件（实验 3）

在与队友讨论过程中发现，我们的条件文法无法处理使用括号改变||、&&运算顺序的情况，如果要用递归下降处理则需引入一种新的括号用于改变逻辑运算符的运算次序。

解决方案：最终讨论得出的解决方案是在递归下降中穿插进行 SLR 分析。

3. 从下向上写代码难以预料上层问题（实验 4）

在实验 4 刚开始写代码时，我们是先写下层翻译，再写到上层翻译，这样做出现的问题是不清楚上层具体需要下层代码完成什么样的工作，下层代码也就很写做得圆满。

解决方案：我和队友一起从上层到下层理清代码逻辑，确认每一部分的代码具体需要完成什么工作。

4. JMP 和 JPC 指令的疑惑（实验 4）

我们认为 JMP 和 JPC 指令可以跳转至空行，所以在翻译时使用相对地址，在一些地方引入了空指令，最后调试时发现虚拟机不允许跳到空行。

解决方案：使用给代码链表中节点打标记，最后再遍历代码链表将标记替换为绝对地址的方法来完成翻译。

三． 实验中的亮点

1. 在实验 1 和实验 2 中，我所写的代码一遍即测试通过，完全没有 bug。

原因是我会花和写代码相等甚至更长的时间对我的代码进行 review。另外为了方便队友阅读或修改我的代码，为每一部分代码都写了清晰的注释，一定程度上也让我对代码更有把握。

2. 在实验 2 和实验 3 中，我将递归下降分析的程序展开写成循环，因此在实验 4 中翻译时不必在函数间传递大量的参数。

3. 在实验 4 中设计出了自认为完美的翻译方案和运行时栈的使用方法，即充分利用所学内容，又根据具体实现做出变通。

四． 本次实验的总结与心得体会。

本课程的实验相较于其他课程的实验好的太多了，本课程实验贴近课本又为课本内容做出了补充，而反观其他课程的实验总是难以做到两全。

在实验过程中，让我对编译原理有了更深的理解，极大地帮助我进行编译课的学习。尤其是第四个实验，在熟识课本内容后设计出自己的方案，即完成了实验又辅助了即将到来的期末考试的复习。

本次实验可以说是收获颇丰，一个小组三个人总共写了几千行代码，为实验 debug 也是我第一次为千行级代码 debug，这样的经历实属难得。