目录

实验目标

本次实验题目要求是实现一个 PL0 语言的编译器，实现词法分析，语法分析和执行代码三个功能。

具体要求如下：

一. 要实现两个函数，getsym()用于得到一个 symbol，getch() 用于读取代码中一个一个的字符。

二. 语法分析程序 block()函数要实现两件事：1）将说明部分的相关变量放入到 table 表中，2）生成 PL0 语言的目标指令，存入 code 数组中。

三. 建立一个解释执行目标程序的函数。

我采用的是递归处理程序的形式实现的，没有采用建立分析表的方式实现。下面依次讲解各个方法的实现。

词法分析程序

这个板块我使用一个 lexer 类实现了相关的功能，下面对其所有的属性和方法进行介绍。

- br，文件读写流，私有属性，用于对代码文件的读取。
- line，字符串，私有属性，使用 br 一次读取一行，读取的结果放入此变量中。
- index，数，私有属性，对 line 字符串是一个字符一个字符处理的，index 即为即将要处理的字符的下标。
- linelength，数，私有属性，读取的一行文本的长度，用于判定 index 是否越界。
- lineNum，数，公有属性，表示的是当前处理的是第几行，这个属性在错误处理模块需要使用，因此将其设为公有属性。
- ch，字符，私有属性，用于当前处理的字符也就是 line[index]
- sym，SYMBOL，公有静态属性，SYMBOL 类型是一个枚举类型，也就是说 sym 是一个

当前处理的 symbol，这个 symbol 是用 character 组成的。由于在语法处理过程中需要大量的引用当前处理的 sym，所以将其设为了公有静态变量。

- id，字符串，公有静态属性，这个用于存储读取到的标识符。
- num，数，公有静态属性，用于存储读取到的数字。
- word，字符串数组，私有属性，此属性存储了 PL0 所允许的几个保留字，用于区分保留字和标识符。
- wsym，SYMBOL 数组，私有属性，与 word 数组是一一对应的，同一个下标的 word 和 wsym 分别代表的是保留字和保留字所对应的 symbol。
- wordLen，数，私有属性，用于表示 word 数组的大小，也就是关键字的多少。
- puncLen，punc，psym，数，字符数组，SYMBOL 数组，私有属性，这三个和 word 对应的三个属性是含义相似的，不过这三个指的是标点符号。
- 构造函数，此函数只需要一个形参，就是一个文件的读取流对象。
- 一些 getter，这是为了应对私有属性被访问的情况。
- getch 函数，一次读取一行，赋值给 line，并对 index，linelength，ch 等变量进行更新，返回新读取的一个字符。
- getsym 函数，无参，不需要向 getch 有一个新读取的一个 symbol 的返回值，因为它把结果写入了 sym 这个静态变量里面。它的具体实现，以读取标识符为例。

```
if (Character.isLetter(ch)) {
    id = "";
    do {
        id += ch;
        ch = getch();
    } while (Character.isLetter(ch) ||
Character.isDigit(ch));
    word[0] = id;
    int i = wordLen;
    while(!word[i].equals(id)){
        i--;
    }
    if(i>0){
        sym = wsym[i];
```

```
        }else{
            sym = SYMBOL.SYM_IDENTIFIER;
        }
    }
```

遇到的问题

Java 的文本流对象读取一行时，会自动将末尾的换行符给处理掉，如果没有换行符的特殊判断的话，getsym 这个函数面对封装好的 getch 函数完全不知道内部的实现情况，只是一味的要 character，如果没有末尾的特殊判定，那么在读取到标识符的时候十分容易上一行的尾和下一行的头的一堆字符连起来形成一个标识符。

我的处理方式是在每一行的末尾添加了一个空格，因为空格既不影响语法结构，又是可以忽略的元素，还可以作为一个间断隔开首尾。

语法分析程序

语法分析程序需要实现两个任务，一是将所有说明部分的变量放到 table 表中，这里补充一点 PL0 语言是先声明再有执行语句的，所以不用担心在执行语句的时候突然蹦出一个变量声明来，这两个区域是严格分开的。二是生成目标代码，PL0 预设的指令格式为(f,l,a)，分别代表功能码，层次差，位移量。功能码有 8 个，如下所示：

- LIT，将常数放到栈顶，a 域为常数
- LOD，将变量放到栈顶，a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值。
- STO，将栈顶的内容送到某变量单元中。a,l 域的含义与 LOD 的相同。
- CAL，调用过程的指令。a 为被调用过程的目标程序的入中地址，l 为层差。
- INT，为被调用的过程（或主程序）在运行栈中开辟数据区。a 域为开辟的个数。
- JMP，无条件转移指令，a 为转向地址。
- JPC，条件转移指令，当栈顶的布尔值为非真时，转向 a 域的地址，否则顺序执行。
- OPR，关系和算术运算。具体操作由 a 域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a 域为 0 时是退出数据区。

这个部分实现了一个 parser 类，用于实现语法分析相关的功能，下面对其所有的属性和方法进行一个说明。

- TMAX，数，私有静态属性。用于初始化 table 数组。
- CMAX，数，私有静态属性，用于初始化目标代码数组时指定数组的大小。
- table，mask 数组，私有属性，mask 是用于填表任务的一个自定义类，包含五个属性，name，kind，level，address，value。对于声明的变量常量，procedure，都可以声明一个 mask 对象来进行保存信息。table 这个数组就是用于存储全局范围内的声明部分。
- tx，数，私有属性，用于代表 table 数组当前处理的项的下标。
- code，instruction 数组，私有属性，用于存储目标代码的一个数组。instruction 类是一个目标代码类，包含三个属性，f，l，a，这三个属性和上面描述的是一致的。
- cx，数，私有属性，code 数组的下标。

- level，数，私有属性，表明当前处理的层次，它是用于标记嵌套调用时不同函数的层次，比如 main 函数调用 a，a 调用 b，main 函数的层次是 0，a 函数的层次就是 1，b 的层次是 2.
- dx，数，私有属性，指向空地址的一个下标，是运行时栈的指针。
- Instuction，字符串数组，私有属性，是八种指令的一个数组。
- listcode，用于打印 code 数组，接受两个参数，开始下标和结束下标。
- 构造函数。无参，函数内将上面的一些属性进行初始化。
- 一些 getter 函数。
- enter 函数，将变量，常量或者是 procedure 的声明放入 table 数组中，只需要接受一个参数，用于说明是哪一类说明。
- constdeclaration，参数是一个词法分析器，用于处理一个常量声明语句。
- vardeclaration，参数是一个词法分析器，用于处理一个 变量声明的语句。
- gen 函数，用于产生一个新的指令，并将新的指令赋值到 code 数组，接受的参数是 f，l，a，此处三个参数的意义和上面说的一样。
- position 函数，输入为 id，如果该 id 是 table 数组某一项的名字，则返回它的下标，反之的话，返回 0.
- factor 函数，用于解析文法中的因子，因子的结构是：<因子>-> <标识符><无符号整数>|(<表达式>)
- term 函数，用于解析文法中的项，项的结构是：<项> -> <因子>{<乘除运算符><因子>}
- expression 函数，用于解析文法中的表达式，表达式的结构是：<表达式> -> [+|-]<项>{<加减运算符><项>}
- condition 函数，用于解析文法中的条件，条件的结构是：<条件>-><表达式><关系运算符><表达式>|ood<表达式>
- statement 函数，用于处理文法规定的几个语句。下面以条件语句为例进行说明

```
case SYM_IF:
    l.getsym();
    condition(l);
    if(lexer.sym == SYMBOL.SYM_THEN)
    {
        l.getsym();
    }
    else{
        ErrHandle.HandleErr(13);
```

```
}
cx1 = cx;
gen(opcode.JPC, 0, 0);
statement(l);
code[cx1].setA(cx);
break;
```

- block 函数，主干方法，实现两个功能，对于声明部分的处理和产生目标代码（通过调用 statement 函数来进行处理）。

遇到的问题

1. 每一个 procedure 调用 block 函数时（也包括 main），需要产生一个无条件跳转指令，跳转到语句产生的目标代码的位置，因为中间可能穿插一些在中间声明的 procedure 的代码，因此需要跳转，但是在最开始的时候是不知道要跳转到哪里的，也就是需要回填，也就是说需要保存一下这条 jmp 语句的下标，等到知道跳转到那里的时候，对该跳转语句的 a 域进行一个回填。
   我们采用了将这个跳转语句的 index 放入 table 数组的当前指向的项的 a 域作为下标的存储地方，因为一般这里是一个 procedure 的声明语句，procedure 的 a 域是不被使用的，但是初始情况怎么办呢？因此我将 table[0]设为不可用的区段，并将它初始化，这样的话，从最开始 main 函数到中间的各个 procedure，采用的下标存储地方都是安全的。
2. 关于 table 表的使用，当处理到一个 procedure 时候，procedure 中的声明部分使用的还是全局的 table 表，但是当 procedure 处理完之后，table 表中被 procedure 中占的那一部分还有没有必要维护呢？
   很显然是没有必要的，因为每个函数的声明部分，只是服务于这个函数的语句分析部分，大致的工作就是判断某个标识符是否在前面声明，因此当 procedure 的语句处理完之后，它的说明部分也没有必要保存了。因此可以通过 table 数组下标的变化，来进行对处理过的 procedure 说明部分的清除，这样的话，对内存的使用也是比较经济的。

建立一个解释执行目标程序的函数

这个板块的任务是执行目标代码，执行之前形成的 code 数组。在这里实现了一个 interpret 类来完成这个任务，下面介绍这个类的属性和方法。

- STACKSIZE，数，静态私有属性，用于设置运行时数据栈的大小。
- s，int 数组，私有属性，运行时的数据栈。
- pc，数，私有属性，用于保存下一条要执行的指令的下标。
- top，数，私有属性，用于保存当前栈顶元素的下标。
- b，数，私有属性，用于保存当前执行的 procedure 的基地址。

- i，instruction 类，私有属性，用于存储当前执行的指令。instruction 类是用来说明指令的一般格式的，有三个属性，f，l，a。这三个属性的意义和上面相同。
- 构造方法，用于初始化上面的一些属性。
- Read 方法，返回值是 int 类型，用于读取用户输入的数字。
- Write 方法，参数是 int 类型，用于将参数输出到控制台。
- Base 方法，参数有两个，一个 int 类型的当前层的基地址，另一个 int 类型，用于表现层次差。返回值就是和当前层次差为 l 的层次的基地址。
- interp 函数，参数是 code，指令数组，这个函数用于执行这个指令数组的每一条指令，大致的实现思路是根据指令的 f，来更新相关的值。以 CAL 指令为例进行说明。

```
case CAL:
    s[top+1] = base(b,i.getL());
    s[top+2] = b;
    s[top+3] = pc;
    pc = i.getA();
    b = top + 1;
    break;
```

遇到的问题

在执行指令的时候，需要找到某个变量的地址，这个地址是在它声明的时候分配的。也就是说需要找到它声明的层次的基地址，为了解决这个问题，在栈顶会存储一些控制信息。

栈顶分配三个单元，一个单元是调用链最开始的过程的数据段的基地址，一个单元是该过程运行之前正在运行的过程的数据区基地址，一个单元是当时程序寄存器 P 的值，用于过程执行完的时候继续执行之前的代码。

错误处理

这个板块主要是进行错误处理，主要的作用是报告错误信息，以及报告错误的行数，这里就使用了 lexer 类中的 lineNum 属性。下面介绍错误处理类的属性和方法。

- errMsg，字符串数组，静态私有属性，用于存储报错信息，比如“undeclared identifier”等等类似的错误。
- HandleErr 函数，错误处理，参数是错误信息数组的下标，内部就是两条输出语句，一条输出错误信息，一条输出错误的行数。