目录

[一、组内分工与贡献介绍 2](#_Toc167994770)

[二、系统功能概述 2](#_Toc167994771)

[（一）、词法分析器 2](#_Toc167994772)

[（二）、语法分析器 2](#_Toc167994773)

[（三）、语法制导的三地址码生成器 2](#_Toc167994774)

[三、词法分析子系统（实验一）： 3](#_Toc167994775)

[（一）、实验目的 3](#_Toc167994776)

[（二）、实验内容 3](#_Toc167994777)

[（三）、词法的正规式描述 3](#_Toc167994778)

[（四）、变换后的正规文法 3](#_Toc167994779)

[（五）、实现 4](#_Toc167994780)

[（六）、实验结果 8](#_Toc167994781)

[四、语法分析（实验二）： 10](#_Toc167994782)

[（一）语法分析系统结构 10](#_Toc167994783)

[（二）语法分析子系统的主要数据结构与算法 10](#_Toc167994784)

[（三）采用的自动生成技术为Bison自动生成工具 11](#_Toc167994785)

[（四）实验结果 12](#_Toc167994786)

[五、三地址码生成（实验三）： 14](#_Toc167994787)

[（一）、实验目的 14](#_Toc167994788)

[（二）、实验内容 14](#_Toc167994789)

[（三）、具体实现 14](#_Toc167994790)

[六、实验体会 17](#_Toc167994791)

[七、思考题 18](#_Toc167994792)

# 一、组内分工与贡献介绍

**图片包含 文本

描述已自动生成**

# 二、系统功能概述

编译系统整体上是由词法分析、语法分析、三地址码生成和错误识别与处理四部分组成。首先是通过词法分析器识别词法符号，之后进行语法分析，同时完成相应的语义处理，最终输出语法树以及生成代码段对应的三地址码，并完成对与错误代码的识别、定位及纠正。

### （一）、词法分析器

词法分析器主要的功能是识别十进制、八进制、十六进制数、标识符、主要运算符和主要关键字等。词法分析器可以从文件中读入代码，并将解析的结果输出到对应的结果文件中。

### （二）、语法分析器

语法分析器主要的功能是对词法分析器的结果进行语法识别，主要运用了Bison自带的LALR（1）分析法根据定义好的产生式进行归约并建立抽象语法树。最后输出最左派生的产生式序列。同时语法分析器还有一定的错误处理功能，可以实现错误的定位，判断错误类型，以及续编译。

### （三）、语法制导的三地址码生成器

语义分析与三地址码生成是根据前两部分的功能将读入的代码翻译成等价的三地址代码，同时将结果输出到文件中。

# 三、词法分析子系统（实验一）：

### （一）、实验目的

基本掌握计算机高级程序设计语言的词法分析程序的开发方法。

### （二）、实验内容

选择适当的方法（自行设计、使用Lex等自动生成工具），设计实现一个能够分析三种整数、标识符、主要运算符和主要关键字的词法分析程序。

### （三）、词法的正规式描述

1、基本要求

标识符 <字母>(<字母>|<数字字符>)\*

十进制整数 0 | (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)\*

八进制整数 0(1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)\*

十六进制整数 0x(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)\*

运算符和分隔符 + | - | \* | / | > | < | = | ( | ) | ；

关键字 if | then | else | while | do

2、附加要求

标识符 <字母>(<字母>|<数字字符>)\*(ε|((\_|.)(<字母>|<数字字符>)+))

十进制数 ((1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)\*|0)(ε|(. (0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)\* ))

八进制数 0( (0|1|2|3|4|5|6|7)+ (ε|(.(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)+))

十六进制数

0x(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)\*

(ε|(. (0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)+))

### （四）、变换后的正规文法

1、正则文法

（1）标识符（l表示字母，d表示数字）：

S→lA|lC

A→lA|dA|ε

C→lC|dC|.B|\_B

B→lA|dA

（2）十进制整数：

S→(1|2|3|4|5|6|7|8|9)A|(0|1|2|3|4|5|6|7|8|9)

A→(0|1|2|3|4|5|6|7|8|9)A|(0|1|2|3|4|5|6|7|8|9)

（3）十进制浮点数：

S→(1|2|3|4|5|6|7|8|9)A|(0|1|2|3|4|5|6|7|8|9)C

A→(0|1|2|3|4|5|6|7|8|9)A|(0|1|2|3|4|5|6|7|8|9)C

C→.D

D→(0|1|2|3|4|5|6|7|8|9)D|(0|1|2|3|4|5|6|7|8|9)

（4）八进制整数：

S→0A

A→(0|1|2|3|4|5|6|7)A|(0|1|2|3|4|5|6|7)

（5）八进制浮点数：

S→0A

A→(0|1|2|3|4|5|6|7)A|(0|1|2|3|4|5|6|7)B

B→.C

C→(0|1|2|3|4|5|6|7)C|(0|1|2|3|4|5|6|7)

（6）十六进制整数：

S→0A

A→xB

B→(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)B|(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)

（7）十六进制浮点数：

S→0A

A→xB

B→(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)B|(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)C

C→.D

D→(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)D|(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)

### （五）、实现

1、定义词法规则

根据之前所述，定义如下正则表达式，用于匹配特定的字符：

文本

描述已自动生成

图1.1 正则表达式图

（1）单字符模式

letter [a-zA-Z]: 匹配任何一个字母（大小写均可）。

dig [0-9]: 匹配任何一个数字（0-9）。

number [1-9]: 匹配任何一个非零数字（1-9）。

oct [0-7]: 匹配任何一个八进制数字（0-7）。

hex [a-f0-9]: 匹配任何一个十六进制数字（a-f 或 0-9）。

（2）标识符模式

IDN1 {letter}({letter}|{dig})\*:

{letter}: 首先匹配一个字母。

({letter}|{dig})\*: 接下来可以有零个或多个字母或数字。

例如，abc123、a1b2c3。

IDN2 {letter}({letter}|{dig})\*(\_|\.)({letter}|{dig})+:

{letter}: 首先匹配一个字母。

({letter}|{dig})\*: 接下来可以有零个或多个字母或数字。

(\_|\.): 接下来必须是一个下划线\_或一个点.。

({letter}|{dig})+: 最后是一个或多个字母或数字。

例如，abc\_123、a1b2.c3。

（3）整数和实数模式（十进制）

INT10 0|{number}{dig}\*:

0: 匹配单个零。

{number}{dig}\*: 或者匹配一个非零数字开头，后面跟零个或多个数字。

例如，0、123。

REAL10 (0|{number}{dig}\*)\.{dig}+:

(0|{number}{dig}\*): 首先匹配一个零，或一个非零数字开头，后面跟零个或多个数字。

\.{dig}+: 接下来是一个点.，后面跟一个或多个数字。

例如，0.1、123.456。

（4）整数和实数模式（八进制）

INT8 0{oct}+:

0: 首先匹配一个零。

{oct}+: 接下来匹配一个或多个八进制数字（0-7）。

例如，0755、0532。

REAL8 0{oct}+\.{oct}+:

0: 首先匹配一个零。

{oct}+: 接下来匹配一个或多个八进制数字（0-7）。

\.{oct}+: 接下来是一个点.，后面跟一个或多个八进制数字（0-7）。

例如，07.123、02.429。

（5）整数和实数模式（十六进制）

INT16 0x{hex}+:

0x: 首先匹配字符0x。

{hex}+: 接下来匹配一个或多个十六进制数字（0-9, a-f）。

例如，0x1f4、0x627。

REAL16 0x{hex}+\.{hex}+:

0x: 首先匹配字符0x。

{hex}+: 接下来匹配一个或多个十六进制数字（0-9, a-f）。

\.{hex}+: 接下来是一个点.，后面跟一个或多个十六进制数字（0-9, a-f）。

例如，0x1f.4a、0x623.78。

2、整体实现思路

（1）采用Flex自动生成工具。

（2）flex根据定义的正则表达式规则识别token，每识别到一个token后, 执行事先写好的对应的一系列动作。

3、具体实现思路

(1) 首先，需要从文件中读取需要进行词法分析的内容，并成功获取文件中的数据。

(2) 然后，根据预先定义的正则表达式规则对输入的字符串进行词法分析，执行以下操作：如果识别为运算符、关键字、标识符等，则直接将识别信息输出到目标文件中；如果识别为数字，则将其转换为十进制数后输出到目标文件中。

(3) 在每次识别后，判断是否还有未识别的数据。如果全部识别完毕，则结束分析；否则，返回步骤(2)继续进行词法分析。

3、实现功能

首先根据实验指导书中的要求，我们最终实现了如下功能：

（1）识别八进制、十进制、十六进制整数、浮点数。

（2）识别关键字、运算符和分隔符。

（3）识别标识符（含“\_”“.”）。

（4）将if、then、else、while、do作为非保留字，同时保证其作为关键字本身用法时能被识别为关键字。

（5）将输出的八进制和十六进制数转换为十进制数

（6）简单的错误处理功能

3、状态图

图示

描述已自动生成

图1.2 状态转移图

4、词法规则实现

电脑屏幕的照片上有文字

描述已自动生成

图1.3 部分代码截图1

在这一部分，我定义了具体的词法规则，每条规则的左侧是一个正则表达式，右侧是对应的C代码。当扫描器匹配到相应的模式时，执行对应的C代码。例如，匹配到"if"时，输出"IF\n"到输出文件。其中，strtol 函数strtol (string to long) 函数用于将字符串转换为长整型数值，strtod (string to double) 函数用于将字符串转换为双精度浮点数。



图1.4 纠错代码截图

同时，我还加入了一个简单的错误检测，到遇到错误时，在exp1out.txt中，输出“ERROR”。

5、主函数部分

文本

描述已自动生成

图1.4 部分代码截图2

main函数打开输入文件exp1in.txt和输出文件exp1out.txt，然后调用yylex()函数启动词法分析，将结果输入到exp1out.txt文件中。yywrap()函数返回1，表示输入处理完毕。

### （六）、实验结果

1、先将.l文件通过如下指令，生成.c和exe文件

文本

描述已自动生成

图1.5 运行指令截图

文本

描述已自动生成

图1.6 运行指令结果图

2、在exp1in.txt中输入测试数据

图形用户界面, 应用程序

描述已自动生成

图1.7 测试数据截图

3、运行exe文件，查看exp1out.txt中的结果

图形用户界面, 应用程序

描述已自动生成

图1.8 测试结果截图

# 四、语法分析（实验二）：

我们构造了两种语法分析程序：通过文件输入和输出分别实现输出语法分析树和最左规约的顺序输出派生的产生式序列。

对于第一种，需要在每个低层语句的动作,也就是翻译模式中新建终结符节点并把树结构回传给非终结符。在最顶层语句对应的动作中深度遍历构建的语法分析树，按格式打印在终端和输出文件。

对于第二种，由于bison使用自下而上的处理顺序，直接打印对应规约语句即可

### （一）语法分析系统结构

图示

描述已自动生成

图3-1

### （二）语法分析子系统的主要数据结构与算法

①print2file 函数：接收一个字符串参数 s，将字符串打印到控制台并写入输出文件 o 中。

②yyerror 函数：接收一个字符串参数 s，将字符串错误消息打印到控制台中。

③newnode 函数：接收三个参数，分别是 len，name 和 brothers。其中，len 表示 brothers 数组的长度，name 表示新节点的名称，brothers 是一个指向结构体 node 的指针数组，表示新节点的兄弟节点们。函数通过 malloc 分配新节点的内存，然后将 name 设置为新节点的名称，将 brothers[0] 设置为新节点的第一个孩子，将其中的兄弟节点连接起来，最后返回指向新节点的指针。

④newleaf 函数：接收一个参数 name，表示新叶子节点的名称。函数通过 malloc 分配新节点的内存，然后将 name 设置为新节点的名称，将该节点的兄弟和孩子都设置为 NULL，最后返回指向新节点的指针。

⑤traverse 函数：接收两个参数，分别是 depth 和 root。其中，depth 表示当前打印字符时需要缩进的空格数，root 是要遍历的根节点。函数首先将 depth 个空格写入到输出文件中，然后将 root 节点的名称也写入到输出文件中，并在结尾处加一个换行符。接着，如果 root 节点有孩子，就递归地调用 traverse 函数，将深度加 2。如果 root 节点有兄弟，则递归地调用 traverse 函数，将深度保持不变。

⑥treefree 函数：接收一个参数，表示要释放其内存的根节点。函数首先判断该节点的孩子和兄弟是否都为空节点，如果是，就释放这些空节点的内存，并将它们的指针设置为 NULL。然后，递归地调用 treefree 函数，继续处理该节点的孩子和兄弟。

⑦main 函数：程序的主函数，通过调用 yyparse 函数解析输入文件中的语法，并将其转换为抽象语法树。在函数开头，先打开输入文件并设置为 yyin 的值，打开输出文件并将其指针存储到全局变量 o 中。最后，返回 yyparse 函数的值。

文本

描述已自动生成

图3-2

⑧错误处理函数：函数首先将全局变量 errflag 设置为 1，表示当前解析出现了错误。然后，通过全局变量 yylloc 记录下当前词法分析器所读取的 Token 的位置信息，从中获取到当前错误所在的列数 start 和结束的列数 end。最后，函数将错误消息、行号和列号写入到输出文件 errorFile 中。具体地，通过 fprintf 函数输出 "Error: %s on Line: %d:c%d to %d:c%d\n" 这样一个格式化的字符串，使用参数 s 替换第一个 %s，使用 yylineno 替换第二个 %d，使用 start 替换第三个 %d，使用 yylineno 替换第四个 %d，使用 end 替换第五个 %d。其中，":c" 是用于表示列数的格式化信息。

### （三）采用的自动生成技术为Bison自动生成工具

Bison通过获得flex分析出的token以及用户定义的产生式进行归约，每当根据一条产生式进行归约时，执行用户预先定义好的一系列动作。

关于bison：bison不分析二义性文法。需要事先告诉优先级和分组规则。

bison使用两种分析方法，一种是LALR(1)，一种是LR我们采用默认的LALR分析：不能处理有歧义的语法，不能前向查看多个token确定是否匹配规则。合并同心闭包集。而判断语句“if else”和“if else then”需要前向查看多个符号。而bison好处是在编译环节自动报告语法存在的冲突。

其中，Bison核心代码如下：

文本, 信件

描述已自动生成

图3-3

文本, 信件

描述已自动生成

图3-3续

### （四）实验结果

徽标, 公司名称

描述已自动生成

图3-2：第一句最左规约的顺序输出派生的产生式序列

图片包含 示意图

描述已自动生成

图3-4

文本

描述已自动生成

图3-4 续

# 五、三地址码生成（实验三）：

### （一）、实验目的

掌握计算机语言的语法分析程序设计与属性文法应用的实现方法。

### （二）、实验内容

采用适当的方法（递归子程序法、LL分析法、LR分析法，使用Yacc等自动生成工具），即实现一个能够进行语法分析并生成三地址代码的微型编译程序。

### （三）、具体实现

1、语法指导定义

语法制导是一种在编译器设计中用于指导编译器如何根据源程序生成目标代码的方法。它结合了上下文无关文法和属性以及语义规则，每个文法符号都关联着一组属性，而每个产生式则配备了一组用于计算这些属性值的语义规则。这种方法的目的是在语法分析的过程中，同时进行语义分析和正确性检查，最终将源语言代码翻译成中间代码或目标代码。

1. 算法基本思想

基于实验二的语法分析程序，通过进一步改进.y文件中的语义规则，实现三地址码的生成。

根据调研和讨论我发现，拉链回填法需要先在右推导时对转移目标进行标记，而后再从左至右回填，代码在回填时一并生成。这就带来了非常多的代码量。通过观察右表所示的语法指导，我想到了一种更简便的方法——本实验实现的算法。该算法利用了语法分析算法自底向上的特点，实现边推导边生成代码，从而实现一次扫描即可完成三地址码生成。

2.1 变量&数据结构

实验用到的变量和数据结构，及其解释，如下文所示，实现如图3-1所示。

block结构体：在语法分析器中，负责随着推导过程，传递分析结果。bCode用于存储推导过程中生成的代码；bPlace用于存储变量地址，而在本算法中，实际存储的是变量（或临时变量）的名称，模拟变量地址；bTrue与bFalse存储的是if语句或while语句跳转地址；begin和next存储的是程序段的起始和结束地址，在本实验中虽然没有用到，但是却是一种允许自制的编译器编译多个代码段的思路。

char\*类型codes数组：用于存储推导过程中生成的代码。

totalTemp与totalLabel：用于生成临时变量和跳转标签的序号。

flag指针（指向pushFlag变量）：压栈指针，模拟生成的代码（字符串格式）向codes数组堆栈。

图形用户界面, 文本, 应用程序

描述已自动生成

图3-1 变量&数据结构

2.2 关键函数解释

void gen\_XXX(char\*place1, ...)，该函数用于生成某语法对应的代码，及跳转标签等。

struct block \* newBlock(char\*bPlace, int \* flag, char \*codes[50], char\* bTrue, char\* bFalse, char\* begin, char\* next)，该函数根据接收的数据，生成block节点，并返回，以节点的形式保存推导过程中生成的代码。

char \* newtemp()，该函数用于生成新的临时变量标签。

char \* newlabel()，该函数用于生成新的跳转标签。

char \* num2dec(char \* s, int base)，该函数用于将数字统一转化为十进制。

2.3语法分析器改进思路

如图3-2所示，在语法推导时，对实验指导书中的语法制导表的语义规则进行复现，复现核心为代码生成操作“codes[(\*flag)++] = xxx;”当语义规则执行完毕，调用newBlock函数进行节点生成，代码也会一并保存在节点中，随着下一步推导继续传递。

图片包含 应用程序

描述已自动生成

图3-2 语法分析器改进思路

1. 简单错误处理&续编译

本实验还添加了简单的错误处理，在语法出错时进行报错提示，并指出哪一类语法出错，比如WHILE后面没有写DO，就会提示是WHILE后方出现了错误。本实验还实现了续编译，使得语法错误发生时，编译器会尽可能继续向下寻找语法正确的代码，继续进行编译。

1. 实验结果

输入如图3-3所示，输出如图3-4，3-5，和3-6所示。

文本, 信件

描述已自动生成文本

描述已自动生成文本

描述已自动生成

图3-3 input 图3-4 output-part1 图3-5 output-part2

文本

描述已自动生成

图3-6 output-part3

图3-3所示的输入，由回车分割为三部分，这三部分的输出对应着图3-4至3-6。第一部分是指导书中的原测试代码，对应的输出如图3-4所示，可以看到生成的代码正确，但是有标签冗余的情况。第二部分是错误的输入，用于测试简单错误处理能力和续编译能力。可见while和if语句都出现了错误，而图3-5的结果表明，编译器能正确分辨这些错误并输出对应的提示，且a=b+c，c=b+d和g=l+y这些正确的语句能够续编译。第三部分是自己设计的测试代码，结果如图3-6所示，可以看到生成的代码还是正确的，且该部分代码在错误代码的后方，能够继续被编译出来，同样体现了续编译能力。

# 六、实验体会

1、殷墨涵（组长）

这次编译原理实验对我们来说极具价值。在实验过程中，我们不仅掌握了编译原理的相关知识，还亲自动手完成了词法分析、语法分析以及三地址码的生成。这一系列实践操作让我对编译过程和编译器的运作机制有了更深层次的认识。通过亲身体验，我不仅加深了对课堂上编译原理理论的理解，还体会到了理论与实际应用之间的差异。在实验中，我不仅提升了自己在编译原理领域的知识水平，还增强了解决问题的能力、编程技巧以及团队协作的精神。从团队合作的角度来看，我们小组成员在实验中表现出色，这得益于每个人的辛勤工作和贡献。这次实验让我更加明白合作的重要性，每个人发挥自己的优势，可以极大提升团队的工作效率。同时，实验过程中我们团队的默契配合，使我们能够以接近完美的标准完成各项任务，这进一步凸显了团队合作的力量。

2、高立杨

本次实验并非一帆风顺，在写代码的过程中，我遇见了内存分配问题错误和代码生成的问题，通过bison的debug功能和逐步调试，最终完成了任务。通过本次实验，我回顾了词法分析、语法分析和三地址码的生成，并且动手实践来加深对理论知识的理解，做到了实践和理论相结合，使自己对编译原理的学习理解更进一步。

3、马雪

语法分析器的构建是一个将理论知识转化为实践应用的过程。在开始前，我们需要深入理解编程语言的语法规则，明确各个语法元素之间的关系和层次结构。当使用Bison编写一个解析器时，在Bison语法文件中定义语法规则，并为每个规则指定一个动作。这些动作在Bison生成的C/C++代码中自动转换为规约语句。我能够清晰地看到如何将源代码中的字符序列转化为记号（tokens），并进一步将记号组合成语法结构。这个过程不仅让我更加深入地理解了语法分析器的内部实现，也让我对编程语言的语法结构有了更加清晰的认识。

4、博文·卡崔拉

# 七、思考题

1. 词法分析能否采用空格来区分单词？

不能，如果遇到如算式格式的文字“1+1=2”这种情况的话会识别不到1之后的加号或者直接报错，如果必须加上空格来分隔的话对测试数据需要有一定的规范约束，若如此则程序不能够具备良好的鲁棒性。并且如果只能用空格来分隔，那么测试数据就不能有换行，因此对测试数据的长度也有限制。综上所述，用空格 区分单词并非一个很好的方法。

1. 程序设计中哪些环节影响词法分析的效率？如何提高效率？

在词法分析器中我们在一个大循环里嵌套分析各种情况的小循环，时间复杂度是O（n2），由此可见时间复杂度很大，效率很低，我们经过查阅资料和讨论，最终发现可以用栈的形式来解决词法分析问题，时间复杂度为O（n）。

1. 生成的三地址代码可否直接输出（不采用数据结构来实现属性code）？

不可以。因为三地址代码的生成是靠文法符号子过程递归调用实现的，如果三地址代码直接输出，不采用数据结构来实现属性code，就会使输出没有逻辑结构性；且上一层的代码输出依赖于调用的子过程代码输出，code是一个综合属性，如果没有数据结构保存子过程的代码，上一层的代码就无法实现。

1. 如何保证四则运算的优先关系和左结合性?

由于三地址代码的实现是依赖于属性文法的，而属性文法是局限于文法产生式的。所以只要在文法产生式规则中注意四则运算的优先关系和作结合性，产生的代码一定是满足四则运算的优先关系和左结合性的。

1. 如何采用代码段相对地址代替三地址代码序列中的标号?

可以在文法符号 S 的结构中增设一个新的综合属性code\_num，使其随着递归调用不断累加，统计代码所占内存单元大小，最后返回至S函数后进行属性赋值等操作。

6、完成一个实验，通常要进行实验设计、实验实现、实验结果分析，你在 本次实验中，是否都涉及的这三个类工作，在大学学习的这几年的实验中，是不是都或多或少设计了这三类工作？

是的。