

**课 程 实验 报 告**

**题目： 编译技术实验**

**课程名称： 编译技术实验**

**专业班级： 软工2003班**

**学 号： U202010783**

**姓 名： 刘铭宸**

**指导教师： 胡雯蔷**

**报告日期： 2023年1月17日**

**软件学院**

目录

[1概述 1](#_Toc125018375)

[2系统描述 2](#_Toc125018376)

[2.1 自定义语言概述 2](#_Toc125018377)

[2.1.1 语法 2](#_Toc125018378)

[2.2单词文法与语言文法 4](#_Toc125018379)

[2.3 符号表结构定义 4](#_Toc125018380)

[2.4 可检查的静态语义错误 4](#_Toc125018381)

[2.5中间代码结构定义 5](#_Toc125018382)

[2.6目标代码指令集选择 6](#_Toc125018383)

[3系统设计与实现 8](#_Toc125018384)

[词法分析器 8](#_Toc125018385)

[语法分析器 8](#_Toc125018386)

[符号表管理 10](#_Toc125018387)

[语义检查 11](#_Toc125018388)

[报错功能 11](#_Toc125018389)

[中间代码生成 12](#_Toc125018390)

[汇编代码生成 13](#_Toc125018391)

[4系统测试与评价 15](#_Toc125018392)

[测试用例 15](#_Toc125018393)

[正确性测试 15](#_Toc125018394)

[报错功能测试 27](#_Toc125018395)

[系统的优点 29](#_Toc125018396)

[系统的缺点 29](#_Toc125018397)

[5实验小结或体会 30](#_Toc125018398)

[**参考文献** 31](#_Toc125018399)

# 1概述

本次实验是构造一个高级语言的子集的编译器，目标代码是汇编语言。按照任务书，实现的方案可以有很多种选择。

可以根据自己对编程语言的喜好选择实现。建议大家选用decaf语言或C语言的简单集合SC语言。

实验的任务主要是通过对简单编译器的完整实现，加深课程中关键算法的理解，提高学生系统软件研发技术。

# 2系统描述

## 2.1 自定义语言概述

本次课程设计自定义的语言为“tzlC”,采用了C语言的部分关键语法规则。数据类型包括char类型、int类型和float类型；基本运算包括算术运算、比较运算、自增自减运算和复合赋值运算；控制语句包括if语句、while语句、break、continue语句、for语句和switch语句；支持多维数组、行注释与块注释。

## 2.2单词文法与语言文法

1. program: ExtDefList
3. ExtDefList:
4. | ExtDef ExtDefList
6. ExtDef:     Specifier ExtDecList SEMI
7. | Specifier ID LP ParamList RP CompSt
8. | Specifier ID LP ParamList RP SEMI
10. Specifier:  TYPE
12. ExtDecList: VarDec
13. | VarDec COMMA ExtDecList
15. VarDec:     ID
16. | VarDec LB **INT** RB
18. ParamVarDec:ID
20. ParamList:
21. | ParamDec
22. | ParamList COMMA ParamDec
24. ParamDec:   Specifier ParamVarDec
26. CompSt:     LC DefList StmList RC
28. StmList:
29. | Stmt StmList
31. DefList:
32. | Def DefList
34. Def:        Specifier DecList SEMI
36. DecList:    Dec
37. | Dec COMMA DecList
39. Dec:        VarDec
40. | VarDec ASSIGN Exp
42. Case:       CASE Exp COLON StmList
44. CaseList:   Case
45. | Case CaseList
47. Stmt:       Exp SEMI
48. | CompSt
49. | RETURN Exp SEMI
50. | RETURN SEMI
51. | IF LP Exp RP Stmt %prec LOWER\_THEN\_ELSE
52. | IF LP Exp RP Stmt ELSE Stmt
53. | WHILE LP Exp RP Stmt
54. | FOR LP Exp SEMI Exp SEMI Exp RP Stmt
55. | SWITCH LP Exp RP LC CaseList RC
56. | SWITCH LP Exp RP LC CaseList DEFAULT COLON StmList RC
57. | BREAK SEMI
58. | CONTINUE SEMI
59. | error SEMI
61. Exp:        Exp ASSIGN Exp
62. | Exp PLUS Exp
63. | Exp MINUS Exp
64. | Exp STAR Exp
65. | Exp DIV Exp
66. | Exp MOD Exp
67. | LP Exp RP
68. | MINUS Exp %prec UMINUS
69. | PLUS Exp %prec UPLUS
71. | Exp AND Exp
72. | Exp OR Exp
73. | NOT Exp
75. | Exp GT Exp
76. | Exp GE Exp
77. | Exp LT Exp
78. | Exp LE Exp
79. | Exp NE Exp
80. | Exp EQ Exp
82. | DPLUS Exp
83. | DMINUS Exp
84. | PLUSD Exp
85. | MINUSD Exp
87. | ID LP Args RP
88. | ID
89. | ID SubList
90. | **INT**
91. | **FLOAT**
93. Args:
94. |  Exp
95. |  Args COMMA  Exp
97. Sub:         LB Exp RB
99. SubList:     Sub
100. |  SubList Sub

## 2.3 符号表结构定义

本实验采用顺序表这种方式管理符号表。此时的符号表symbolTable是一个顺序栈，栈顶指针index初始值为0，每次填写符号时，将新的符号填写到栈顶位置，再栈顶指针加1。

## 2.4 可检查的静态语义错误

（1）使用未定义的变量；

（2）调用未定义或未声明的函数；

（3）在同一作用域，名称的重复定义（变量名、函数名）。为更清楚说明语义错误，这里也可以拆分成几种类型的错误，如变量重复定义、函数重复定义、形参名重复定义；

（4）对非函数名采用函数调用形式；

（5）对函数名采用非函数调用形式访问；

（6）函数调用时参数个数不匹配，如实参表达式个数太多、或实参表达式个数太少；

（7）函数调用时实参和形参类型不匹配；

（8）对非数组变量采用下标变量的形式访问；

（9）数组变量的下标不是整型表达式；

（10）赋值号左边不是左值表达式；

（11）对非左值表达式进行自增、自减运算；

（12）类型不匹配。需要指出类型不匹配错误；有些需要根据定义的语言的语义自行进行界定，比如：32+'A'，10\*12.3，如果使用强类型规则，则需要报错，如果按C语言的弱类型规则，则是允许这类运算的，但需要在后续阶段需要进行类型转换，类型统一后再进行对应运算；

（13）函数返回值类型与函数定义的返回值类型不匹配；

（14）函数没有返回语句（当函数返回值类型不是void时）；

（15）break语句不在循环语句或switch语句中；

（16）continue语句不在循环语句中；

（17）switch 语句的key值不是常数

（18）switch 语句的key值相等

（19）调用的函数已经声明但未定义

（20）函数声明和定义的返回类型不同，或参数数目不同，或声明和定义的形参类型不一致

## 2.5中间代码结构定义

表2-1 中间代码定义

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **语法** | **描述** | **Op** | **Opn1** | **Opn2** | **Result** |
| LABEL x | 定义标号x | LABEL |  |  | x |
| FUNCTION f: | 定义函数f | FUNCTION |  |  | f |
| x := y | 赋值操作 | ASSIGN | y |  | x |
| x := y + z | 加法操作 | PLUS | y | z | x |
| x := y - z | 减法操作 | MINUS | y | z | x |
| x := y \* z | 乘法操作 | STAR | y | z | x |
| x := y / z | 除法操作 | DIV | y | z | x |
| GOTO x | 无条件转移 | GOTO |  |  | x |
| IF x [relop] y GOTO z | 条件转移 | [relop] | x | y | z |
| RETURN x | 返回语句 | RETURN |  |  | x |
| ARG x | 传实参x | ARG |  |  | x |
| x:=CALL f | 调用函数(有返回值) | CALL | f |  | x |
| CALL f | 调用函数(无返回值) | CALL | f |  |  |
| PARAM x | 函数形参 | PARAM |  |  | x |

## 2.6目标代码指令集选择

表2-2 中间代码与MIPS32指令对应关系

|  |  |
| --- | --- |
| **中间代码** | **MIPS32指令** |
| LABEL x | x： |
| x :=#k | li reg(x),k |
| x := y | move reg(x), reg(y) |
| x := y + z | add reg(x), reg(y) , reg(z) |
| x := y - z | sub reg(x), reg(y) , reg(z) |
| x := y \* z | mul reg(x), reg(y) , reg(z) |
| x := y / z | div reg(y) , reg(z)  mflo reg(x) |
| GOTO x | j x |
| RETURN x | move $v0, reg(x)  jr $ra |
| IF x==y GOTO z | beq reg(x),reg(y),z |
| IF x!=y GOTO z | bne reg(x),reg(y),z |
| IF x>y GOTO z | bgt reg(x),reg(y),z |
| IF x>=y GOTO z | bge reg(x),reg(y),z |
| IF x<y GOTO z | blt reg(x),reg(y),z |
| IF x<=y GOTO z | ble reg(x),reg(y),z |
| X:=CALL f | jal f  move reg(x),$v0 |

# 3系统设计与实现

（写清楚自己的实现步骤，详细介绍完成该步骤的目标及思路，以及在程序开发时出现错误时的调整过程）

### 词法分析器

词法分析器的构造技术线路，首选一个就是设计能准确表示各类单词的正则表达式。用正则表达式表示的词法规则等价转化为相应的有穷自动机FA，确定化、最小化，最后依据这个FA编写对应的词法分析程序。

实验中，词法分析器可采用词法生成器自动化生成工具GNU Flex，该工具要求以正则表达式（正规式）的形式给出词法规则，遵循上述技术线路，Flex自动生成给定的词法规则的词法分析程序。于是，设计能准确识别各类单词的正则表达式就是关键。

编写lex.l文件，首先定义YY\_USER\_ACTION，维护每个单词的位置属性；接着定义YYLVAL，包括整型常量、浮点数常量和标识符的属性；定义标识符、整型常量、浮点数常量以及行注释与块注释的正则表达式。其中，行注释与块注释的正则表达式分别为"//"[^\n]\*和"/\*"([^\\*]|(\\*)\*[^\\*/])\*(\\*)\*"\*/"。然后编写规则部分，罗列出所有种类的单词，表示词法分析器一旦识别出该正则表达式所对应的单词，就执行动作所对应的操作，返回单词的种类码。

### 语法分析器

语法分析采用生成器自动化生成工具GNU Bison，该工具采用了LALR的自底向上的分析技术，完成语法分析。在语法分析阶段，当语法正确时，生成抽象语法树，作为后续语义分析的输入。

本次课设采用C++实现语法分析程序。在语法分析源程序parser.ypp中，首先定义抽象语法树结点的类。首先建立一个基类AST，由其派生出各结点的类，如图3-1所示为结点类的关系图。为了能准确给出有关语法、语义错误的位置，在每个结点中保存了对应语法成分的行列号，所以在基类AST中定义了属性Line和Column；以及后续显示语法树，语义分析和中间代码生成等需要遍历语法树，在基类中定义了虚函数DisplayAST、Semantics和GenIR体现了多态性，另外静态成员符号表和几个静态变量等，起到了全局量的作用。



图3-1 AST节点类关系图

在parser.ypp中，在归约的过程中，语义子程序完成创建相应的结点对象，并给对象的属性确定相应的值。一旦建立了完整的语法树后，即可开始后续的编译过程。在语法分析实验阶段，采用凹入表的方式显示语法树，每个结点对象使用成员函数DisplayAST把该结点的语法成分显示出来。

在定义结点类时，当其语法成分有多个同类别的项组成时，使用vector向量进行保存，例如一个程序（类名ProgAST）有多个外部项组成，外部项（类名ExtDefAST）包括外部变量定义（类名ExtVarAST）、函数定义（类名FunDefAST）等，所以类ProgAST中有ExtDefAST的vector向量，依次存放各外部项的结点。类似的还有符合语句结点，包含说明语句结点组成的向量和执行语句结点组成的向量。节点类定义如图3-2所示。

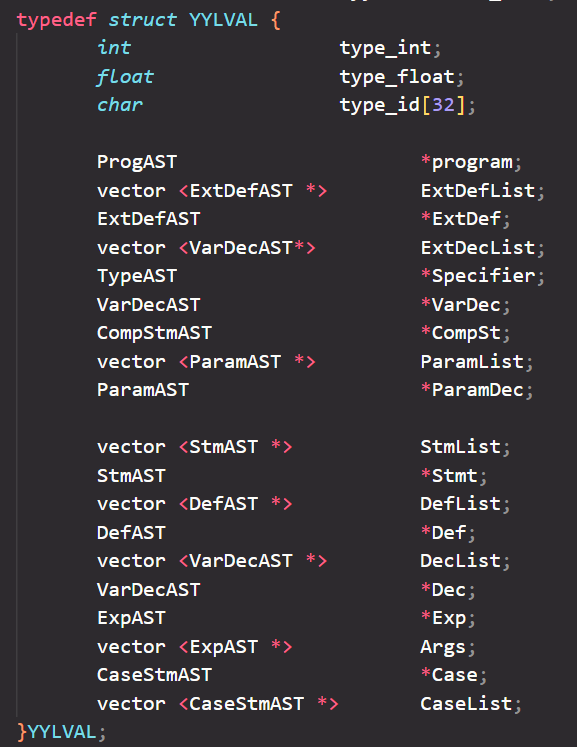


图3-2结点类定义

最后编写定义文件def.h和语法树显示程序ast.cpp，对语法分析程序建立的抽象语法树进行显示。

### 符号表管理

语义分析这部分的一个非常重要的工作就是符号表的管理，在编译过程中，编译器使用符号表来记录源程序中各种名字的特性信息。所谓“名字”包括：程序名、过程名、函数名、用户定义类型名、变量名、常量名、枚举值名、标号名等，所谓“特性信息”包括： 上述名字的种类、具体类型、维数、函数参数个数、常量数值及目标地址（存储单元偏移地址）等。

本次课设采用顺序表的方式管理符号表。此时的符号表symbolTable是一个顺序栈，栈顶指针index初始值为0，每次填写符号时，将新的符号填写到栈顶位置，再栈顶指针加1。

### 语义检查

语义分析这部分完成的是静态语义分析，主要包括：

（1）控制流检查。控制流语句必须使得程序跳转到合法的地方。例如一个跳转语句会使控制转移到一个由标号指明的后续语句。如果标号没有对应到语句，那么就出现一个语义错误。再者，break、continue语句必须出现在循环语句当中。

（2）唯一性检查。对于某些不能重复定义的对象或者元素，如同一作用域的标识符不能同名，需要在语义分析阶段检测出来。

（3）名字的上下文相关性检查。名字的出现在遵循作用域与可见性的前提下应该满足一定的上下文的相关性。如变量在使用前必须经过声明，如果是面向对象的语言，在外部不能访问私有变量等等。

（4）类型检查包括检查函数参数传递过程中形参与实参类型是否匹配、是否进行自动类型转换等等。

这部分实现时，考虑到除了外部项定义的符号外，剩下的符号都是定义在复合语句中，函数形参可以作为函数体复合语句中定义的局部变量，所以除了全局符号表外，每一个复合语句结点维护一张符号表。由于各层符号表是单独管理的，和前面用一个顺序表管理全部符号方式不同，所以可以省略层号LEVEL这个属性。在语法树的遍历过程中，每次访问到一个复合语句结点，就将其符号表进栈，离开时退栈。查表方式从栈顶到栈底方向，逐层查找符号表。当某符号（函数名或变量名）查找成功时，将其符号表的位置保存在结点中，以备后续阶段使用。

### 报错功能

在语义分析过程中，各个变量名有其对应的作用域，一个作用域内不允许名字重复，为此，通过一个全局变量LEV来管理，LEV的初始值为0。这样在处理外部变量名，以及函数名时，对应符号的层号值都是0；处理函数形式参数时，固定形参名在填写符号表时，层号为1。由于允许有复合语句，复合语句中可定义局部变量，函数体本身也是一个复合语句，这样在AST的遍历中，通过LEV的修改来管理不同的作用域。

（1）每次遇到一个复合语句的结点COM\_STM，首先对LEV加1，表示准备进入一个新的作用域，为了管理这个作用域中的变量，使用栈symbol\_scope\_TX，记录该作用域变量在符号表中的起点位置，即将符号表symbolTable的栈顶位置symbolTable.index保存在栈symbol\_scope\_TX中。

（2）每次要登记一个新的符号到符号表中时，首先在symbolTable中，从栈顶向栈底方向查层号为LEV的符号，是否有和当前待登记的符号重名，是则报重复定义错误，否则使用LEV作为层号将新的符号登记到符号表中。

（3） 每次遍历完一个复合语句结点COM\_STM的所有子树，准备回到其父结点时，这时该复合语句语义分析完成，需要从符号表中删除该复合语句的变量，方法是首先symbol\_scope\_TX退栈，取出该复合语句作用域的起点，再根据这个值修改symbolTable.index，同时LEV减一，很简单地完成了符号表的符号删除操作。

（4）符号表的查找操作，在AST的遍历过程中，分析各种表达式，遇到变量的访问时，在symbolTable中，从栈顶向栈底方向查询是否有相同的符号定义，如果全部查询完后没有找到，就是该符号没有定义；如果相同符号在符号表中有多处定义，按查找的方向可知，符合就近优先的原则。如果查找到符号后，就进一步进行语义分析，如：（1）函数调用时，根据函数名在符号表找到的是一个变量，不是函数，需要报错；（2）函数调用时，根据函数名找到这个函数，需要判断参数个数、类型是否匹配；（3）根据变量名查找的是一个函数。等等，需要做出各种检查。

### 中间代码生成

这部分实现时，对每一个结点定义一个列表属性，list <IRCode> IRCodes，这里IRCode是中间代码的四元组类，成员函数GenIR，根据该结点对应的语法特征，将其各部分组合成对应的中间代码序列。

例如while循环语句结点，首先生成3个标号，LoopCond（循环条件计算入口）、LoopEntry（循环体入口）、LoopEnd（循环退出点）。将LoopEntry和LoopEnd作为参数调用循环条件的子树，为循环条件计算提供真假出口，生成循环条件的中间代码；接着生成循环体的中间代码，最后将各片段连接起来，得到while语句的中间代码为：label LoopCond ||循环条件中间代码|| label LoopEntry ||循环体中间代码|| goto LoopCond || label LoopEnd。

### 汇编代码生成

当选择朴素的寄存器分配方案后，目标代码生成时，每当运算操作时，都需要将操作数读入到寄存器中，运算结束后将结果写到对应的单元。由于选择朴素的寄存器分配，只会用到几个寄存器，这里约定操作数使用$t1和$t2，运算结果使用$t3，翻译的方法如表5-2所示。

表3-1 朴素寄存器分配的翻译

|  |  |
| --- | --- |
| **中间代码** | **MIPS32指令** |
| x :=#k | li $t1,k  sw $t1, x的偏移量($sp) |
| x := y | lw $t1, y的偏移量($sp)  sw $t1, x的偏移量($sp) |
| x := y + z | lw $t1, y的偏移量($sp)  lw $t2, z的偏移量($sp)  add $t3,$t1,$t2  sw $t3, x的偏移量($sp) |
| x := y - z | lw $t1, y的偏移量($sp)  lw $t2, z的偏移量($sp)  sub $t3,$t1,$t2  sw $t3, x的偏移量($sp) |
| x := y \* z | lw $t1, y的偏移量($sp)  lw $t2, z的偏移量($sp)  mul $t3,$t1,$t2  sw $t3, x的偏移量($sp) |
| x := y / z | lw $t1, y的偏移量($sp)  lw $t2, z的偏移量($sp)  mul $t3,$t1,$t2  div $t1,$t2  mflo $t3  sw $t3, x的偏移量($sp) |
| RETURN x | move $v0, x的偏移量($sp)  jr $ra |
| IF x==y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  beq $t1,$t2,z |
| IF x!=y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  bne $t1,$t2,z |
| IF x>y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  bgt $t1,$t2,z |
| IF x>=y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  bge $t1,$t2,z |
| IF x<y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  blt $t1,$t2,z |
| IF x<=y GOTO z | lw $t1, x的偏移量($sp)  lw $t2, y的偏移量($sp)  blt $t1,$t2,z |
| X:=CALL f |  |

对于函数调用X:=CALL f，需要完成开辟活动记录的空间、参数的传递和保存返回地址等，函数调用返回后，需要恢复返回地址，读取函数返回值以及释放活动记录空间。活动记录的空间布局没有一个统一的标准，可根据自己的理解保存好数据，并能正确使用即可。

通常，使用4个寄存器完成参数的传递，多余4个的参数使用活动记录空间，这里做了简单处理，所有参数都使用活动记录空间。具体步骤：

1. 首先根据符号表中函数定义项得到该函数活动记录的大小，开辟活动记录空间和保存返回地址。思考一下main函数的活动记录如何处理？
2. 根据函数定义中的参数个数paramnum，即在X:=CALL f之前有paramnum个ARG形式的中间代码，可获得各个实参值所存放的单元，取出后送到形式参数的单元中。

（3） 使用jal f 转到函数f处

（4） 释放活动记录空间和恢复返回地址。

（5） 使用sw $v0, X的偏移量($sp) 获取返回值送到X的存储单元中。

# 4系统测试与评价

### 测试用例

### 正确性测试

1. 数组测试

源程序代码如图4-1所示。

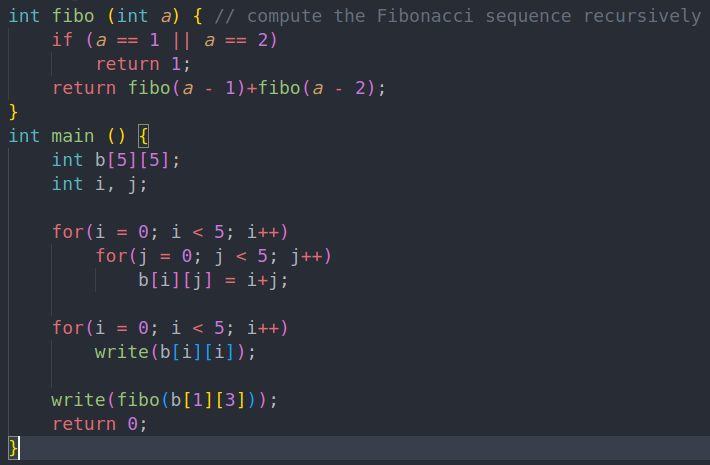


图 4-1

抽象语法树如图4-2、4-3、4-4、4-5所示。

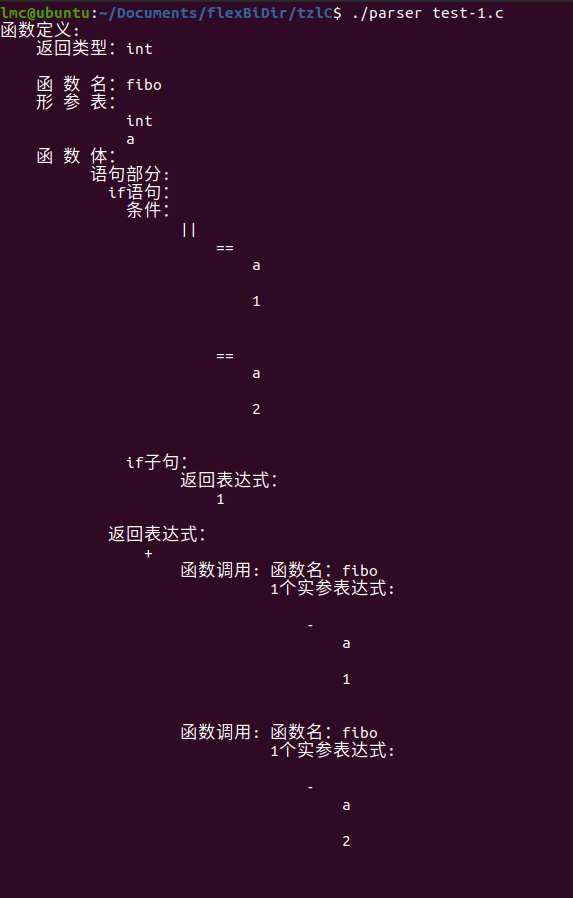


图 4-2

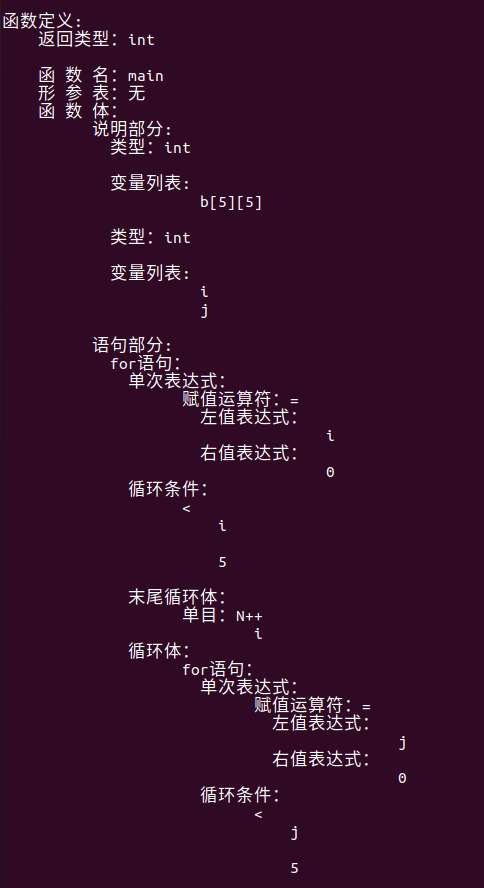


图 4-3



图 4-4

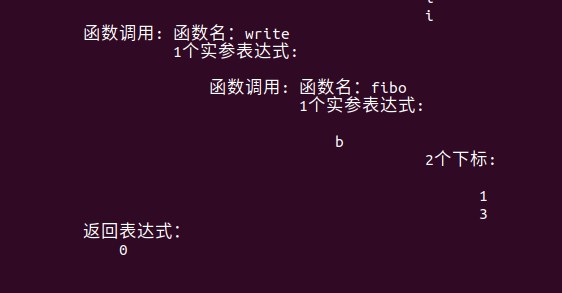


图 4-5

符号表如图4-6所示。

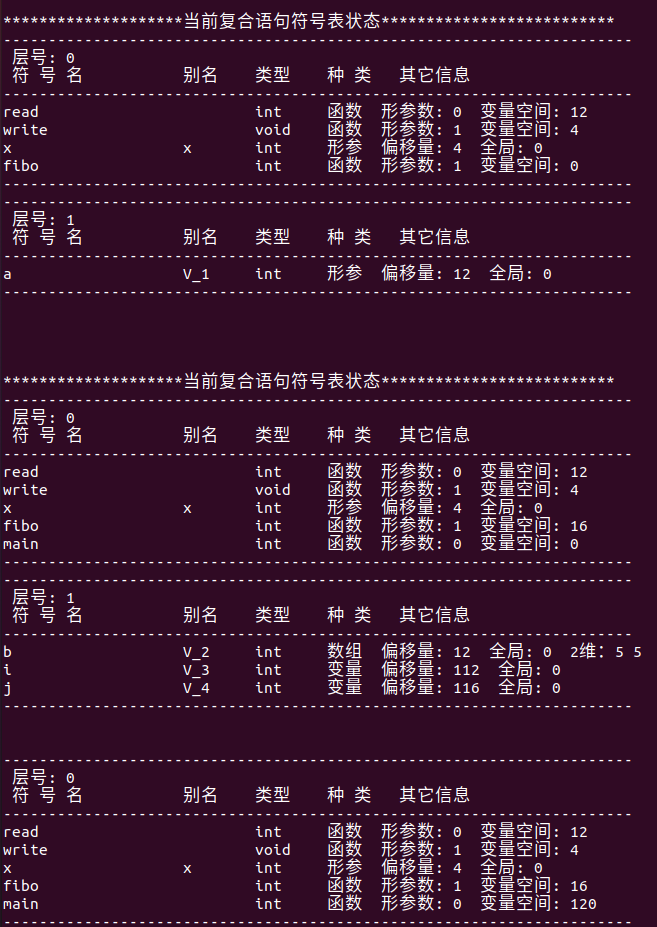


图 4-6

生成的汇编代码如图4-7、4-8所示。

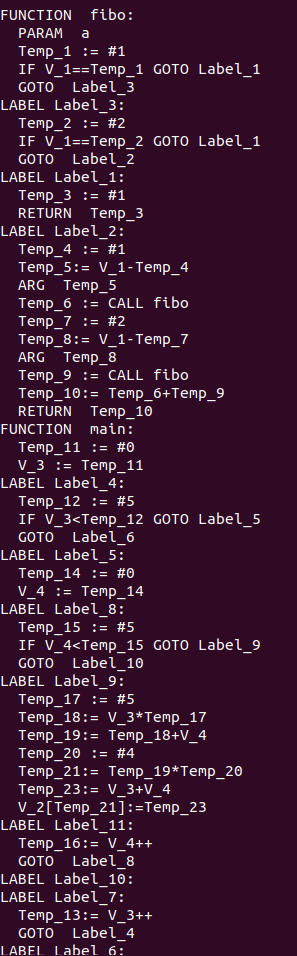


图 4-7

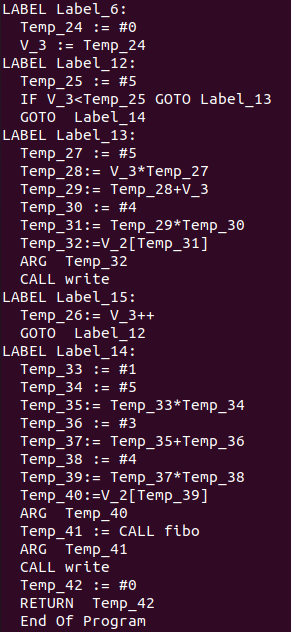


图 4-8

运行结果如图4-9所示。

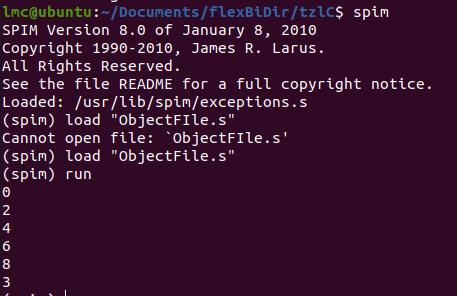


图 4-9

1. Switch语句和自增语句测试

源程序代码如图4-10所示。

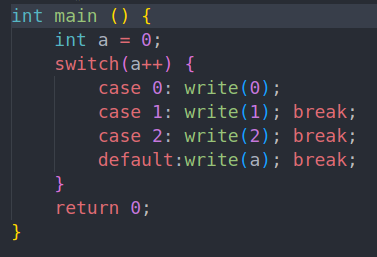


图 4-10

语法树如图4-11所示。

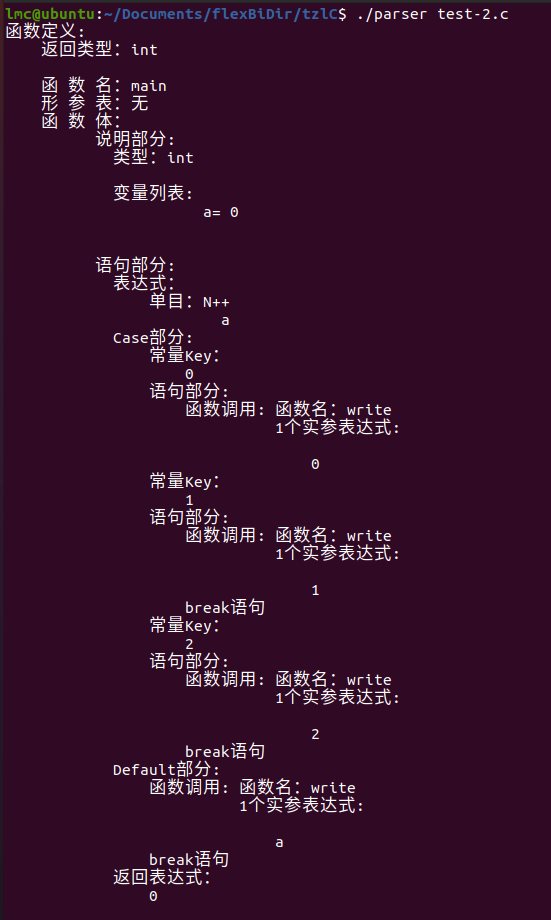


图 4-11

符号表如图4-12所示。

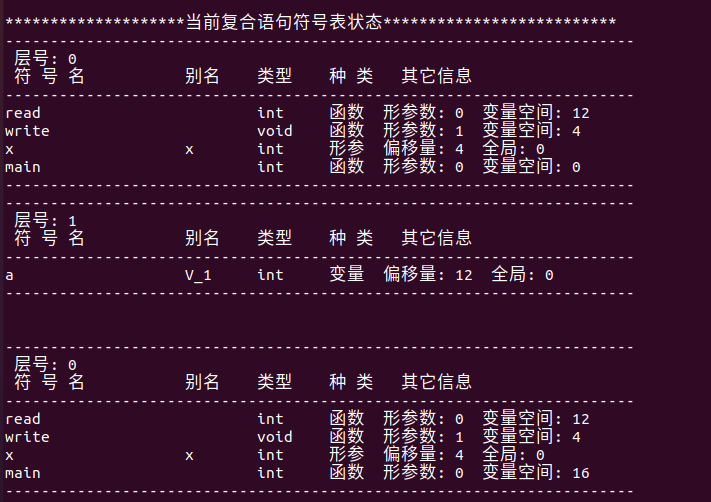


图 4-12

运行结果如图4-13所示。

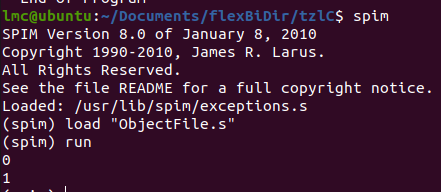


图 4-13

1. for循环测试

源程序代码如图4-14所示。

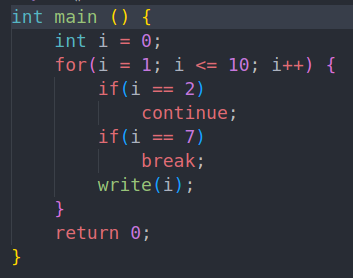


图 4-14

语法树如图4-15所示。

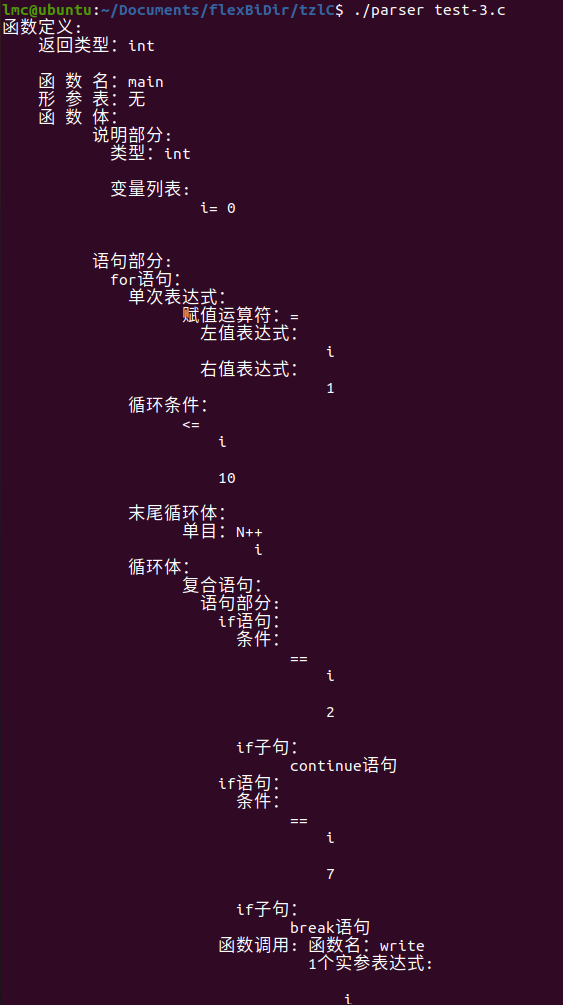


图 4-15

符号表如图4-16所示。

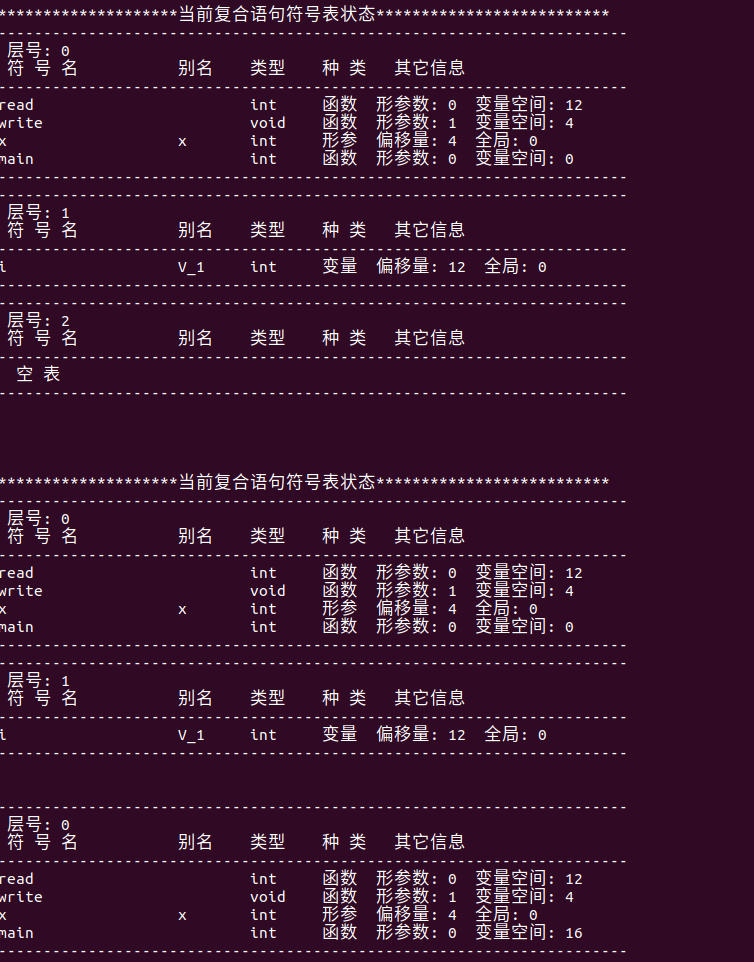


图 4-16

运行结果如图4-17所示。

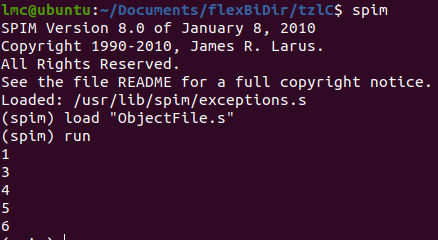


图 4-17

### 报错功能测试

1. 静态语义错误（switch语句）

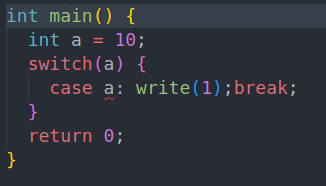


图 4-18

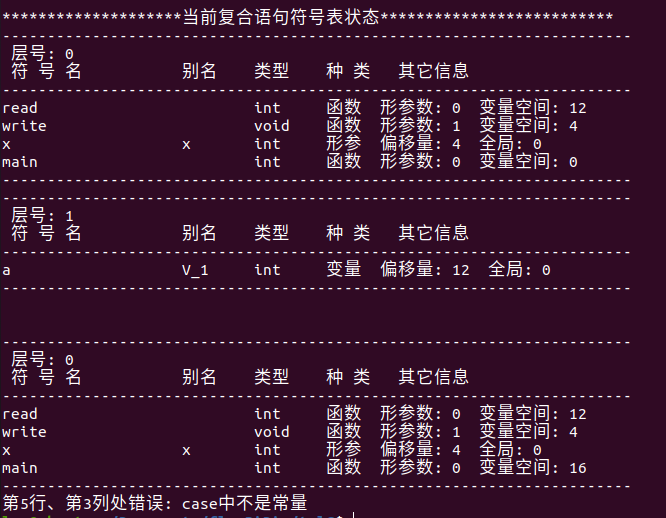


图 4-19

1. 词法语法错误

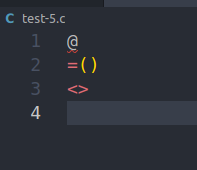


图 4-20



图 4-21

1. 静态语义错误

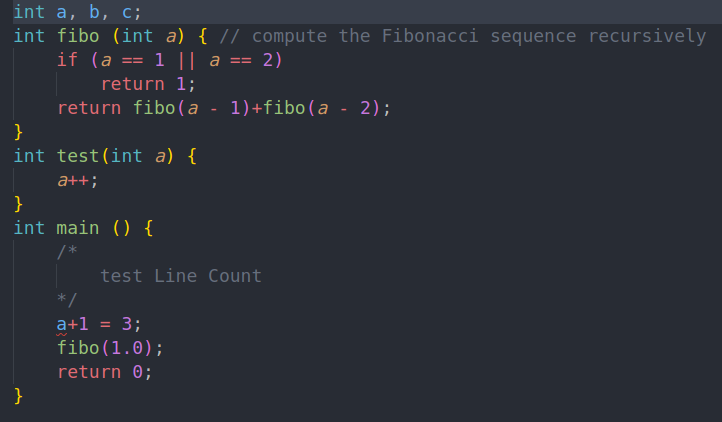


图 4-22

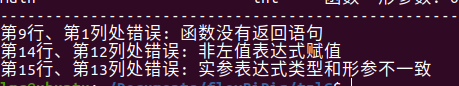


图 4-23

### 系统的优点

1. 数据类型包括char类型、int类型和float类型；
2. 基本运算包括算术运算、比较运算、自增自减运算和复合赋值运算；
3. 控制语句包括if语句、while语句、break、continue语句和for语句，并实现了switch语句；
4. 实现多维数组。
5. 支持行注释与块注释。
6. 实现了对20种静态语义错误的处理。

### 系统的缺点

1. 数据类型未实现字符串；
2. 没有实现goto语句；
3. 没有实现结构体；
4. 没有进行额外的中间代码优化。

# 5实验小结或体会

本次编译原理的课程设计可以说是我做过的最为复杂的课程设计之一，实验内容与课程所教授的理论知识差异大、对这种编程语言和编程方式的不熟悉、任务工作量大、新冠疫情等等都给我造成了较大的困难。幸好在老师、同学的指导交流和帮助下，我一步一步实现了所要求的诸多功能。在此过程中，我遇到了很多问题，比如在一开始，符号表内含次序存放不同level的SymbolsInAScope，SymbolsInAScope内存放参数，如果这样处理函数声明的话，函数声明的局部参数表就会保留在参数表内，而不是像只有函数定义的时候一样，处理完了这个复合语句后就被弹出了，导致符号表出了问题，后续出现了很多问题。最后在同学的帮助下，将符号表修改为树形，再遇到函数定义时，参数再入符号表，结束函数定义时退出符号表；处理定义时，将前面参数复制过来，接着做函数定义的处理，结束后退栈也是一样，由此解决了问题。总之，这次课设虽然难度大，但对我的编程能力是一次很大的锻炼，同时也巩固了在课堂上所学习的理论知识。最后，感谢胡雯蔷老师和祝建华老师在实验和课程设计上的耐心指导与帮助!

**参考文献**

[1] 王生元 等. 编译原理(第三版). 北京：清华大学出版社，20016

[2] 胡伦俊等. 编译原理(第二版). 北京：电子工业出版社，2005

[3] 王元珍等. 80X86汇编语言程序设计. 武汉：华中科技大学出版社,2005

[4] 王雷等. 编译原理课程设计. 北京：机械工业出版社，2005

[5] 曹计昌等. C语言程序设计. 北京：科学出版社，2008