|  |
| --- |
| **《编译原理》**  **实践报告**  项目名称 C-- 编译器  专业班级 计算机 T2001  学 号 8208200412  姓 名 黄道煜  **实验成绩：**  **批阅教师：**  2022 年 6 月 28 日 |

# 《C--编译器》

**一、实践目的**

加深对编译器工作过程的理解，包括：对词法分析器、语法分析器、语义分析器以及目标代码生成器的工作过程的理解；同时，加强对词法分析方法、语法分析方法、语义分析方法以及目标代码生成方法的掌握；能够采用一种编程语言实现简单的编译器；能够使用自己编写的分析程序对简单的程序进行编译分析。

**二、实践内容**

自定义一种程序设计语言，或者选择已有的一种高级语言，编制它的词法分析/语法分析/语义分析/目标代码生成程序。词法分析/语法分析/语义分析/目标代码生成程序的实现可以采用任何一种编程语言和编程工具。



词法分析：输入一种高级语言的源程序（如：C，C++，Java，Python等），识别出各个具有独立意义的单词，即：关键字、标识符、常数、运算符、界符。并依次输出各个单词的内部编码及单词符号自身值，遇到错误时应该显示“Error”及错误的位置。

语法分析：对于输入的源程序语言编制语法分析程序，语法分析程序的实现可以采用任何一种编程语言和工具，遇到错误时应该显示“Error”及错误的位置。

语义分析：对于输入的源程序语言编制语义分析程序，语义分析程序的实现可以采用任何一种编程语言和工具，遇到错误时应该显示“Error”及错误的位置。

代码生成：对于输入的源程序语言生成目标代码。

**三、实践方法**

**1. 词法分析**

词法分析程序根据构词规则对输入的 C-- 源代码进行模式匹配，对源程序进行单词切分和识别，直到源程序结束。每个单词用二元式 (单词类别, 单词属性) 表示。

**2. 语法分析**

编译器首先将语法转化为 DFA 并处理出 First 集和 Follow 集，然后使用 SLR 分析方法对代码进行语法制导翻译。

**3. 语义分析**

引入标记非终结符将原文法转化为 S-属性文法，为每个产生式分配一条语义规则。然后使用自下而上的一遍扫描方法，在语法分析的同时进行语义规则的计算，并输出四元式作为中间代码。

**4.目标代码生成**

根据语义分析生成四元式的中间代码，并进行基本块的划分。根据基本块进行活跃变量分析，并为变量分配寄存器。然后根据符号表进行内存分配。最终输出 mips 汇编代码。

**四、实践步骤**

**1. 词法分析**

词法分析部分首先定义了C-- 语言对构词规则。 然后对 C-- 语言源代码文件 \*.cmm 进行词法分析，将源程序进行单词切分和识别，每个单词用二元式 (单词类别, 单词属性) 表示，并输出至 lexical.txt 文件中。

对于 C-- 语言，编译器将单词符号分为五大类：标识符，关键字，常数，运算符，界符。具体的构词规则如下：

标识符：字母(字母|数字)\* - {关键字}

关键字：int | void | if | else | while | return

常数：数字(数字)\*

运算符：= | + | - | \* | / | = | == | > | >= | < | <= | != | ( | )

界符： ; | , | /\* | \*/ | // | { | } | #

目前 C-- 语言只支持 int 型变量和基本的流程控制语句，关键字种类较少。词法分析时只需要对源代码进行简单的模式匹配即可。当无法进行匹配时，则说明存在非法字符串，进行报错处理。具体代码见第六部分 LexicalAnalyser.cpp 文件。

**2. 语法和语义分析**

语法和语义分析部分首先定义了 C-- 语言的基本语法，然后引入标记非终结符 “A，N，M”并为每个产生式规定语义动作，构造出 S-属性文法，然后根据修改后的文法构造 DFA，并求出 First 集和 Follow 集。最后使用自下而上的一遍扫描方法，对词法分析生成的二元式序列进行分析，分析栈的具体变化情况输出至文件 analyse.txt 中。

C-- 语言参考了 C 语言的部分语法，支持 if - else 语句，while 语句，赋值语句和函数。具体的语法规则如下：

Program ::= <声明串>

<声明串> ::=<声明>{<声明>}

<声明> ::=int <ID> <声明类型> | void <ID> <函数声明>

<声明类型>::=<变量声明> | <函数声明>  
<变量声明> ::= ;

<函数声明> ::=’ (‘<形参>’) ‘<语句块>

<形参> ::= <参数列表> | void

<参数列表> ::= <参数> {, <参数>}

<参数> ::= int <ID>

<语句块> ::= ‘{‘<内部声明>; <语句串>’}’

<内部声明> ::= 空 | <内部变量声明>{; <内部变量声明>}

<内部变量声明>::=int <ID>

<语句串> ::= <语句>{ <语句> }

<语句> ::= <if语句> |< while语句> | <return语句> | <赋值语句>

<赋值语句> ::= <ID> = <表达式>;

<return语句> ::= return [ <表达式> ]

<while语句> ::= while ‘( ‘<表达式> ‘)’ <语句块>

<if语句> ::= if ‘(‘<表达式>’)’ <语句块> [ else <语句块> ]

<表达式>::=<加法表达式>{ relop  <加法表达式> }

<加法表达式> ::= <项> {+ <项> | -<项>}

<项> ::= <因子> {\* <因子> | /<因子>}

<因子> ::=num | ‘(‘<表达式>’)’ |<ID> (参数列表)  
<参数列表> ::= [ <表达式>,  <参数列表> ]

<ID>::=字母(字母 | 数字)\*

根据处理出的 DFA 和 First 集和 Follow 集进行自下而上分析，若某个状态的转移不在 DFA 中，或者代码处理完后为到达接收状态则进行报错处理。具体代码见第六部分 ParserAndSemanticAnalyser.cpp文件。

**3. 中间代码生成**

中间代码生成部分是对上一步生成的四元式代码进行基本块的划分，构造出数据流图作为中间代码保存至文件 mid.txt中。具体代码见第六部分 IntermediateCode.cpp 文件。

**4. 最终代码生成**

最终代码生成部分根据上一步生成的数据流图进行活跃变量分析，然后为变量分配寄存器。因为 C-- 语言只支持 int 型变量，不支持数组等其他类型变量，因此这部分中符号表构造和内存分配较为简单。最终得到 mips 汇编代码，并将其保存至 program.asm 文件中。具体代码见第六部分 ObjectCodeGenerator.cpp 文件。

**五、实践结果**

输入程序 test.cmm

int mul(int x, int y)

{

return x\*y;

}

void main(void)

{

int a;

int b;

a=10;

b=8;

while(a>=0)

{

b=mul(b, b);

a=a-1;

}

return;

}

#

词法分析结果 lexical.txt

INT int

ID mul

LPAREN (

INT int

ID x

COMMA ,

INT int

ID y

RPAREN )

LBRACE {

RETURN return

ID x

MULT \*

ID y

SEMI ;

RBRACE }

VOID void

ID main

LPAREN (

VOID void

RPAREN )

LBRACE {

INT int

ID a

SEMI ;

INT int

ID b

SEMI ;

ID a

ASSIGN =

NUM 10

SEMI ;

ID b

ASSIGN =

NUM 8

SEMI ;

WHILE while

LPAREN (

ID a

GTE >=

NUM 0

RPAREN )

LBRACE {

ID b

ASSIGN =

ID mul

LPAREN (

ID b

COMMA ,

ID b

RPAREN )

SEMI ;

ID a

ASSIGN =

ID a

MINUS -

NUM 1

SEMI ;

RBRACE }

RETURN return

SEMI ;

RBRACE }

ENDFILE #

语法分析结果 analyse.txt

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/analyse.txt>

中间代码 mid.txt

[main]

main:

(=,10,\_,a)

(=,8,\_,b)

next1 = 1

next2 = -1

Label1:

(j<,a,0,Label4)

next1 = 2

next2 = 4

Label2:

(par,b,\_,\_)

(par,b,\_,\_)

(call,mul,\_,\_)

next1 = 3

next2 = -1

Label3:

(=,@RETURN\_PLACE,\_,T1)

(=,T1,\_,b)

(-,a,1,T2)

(=,T2,\_,a)

(j,\_,\_,Label1)

next1 = 1

next2 = -1

Label4:

(return,\_,\_,\_)

next1 = -1

next2 = -1

[mul]

mul:

(get,\_,\_,x)

(get,\_,\_,y)

(\*,x,y,T0)

(return,T0,\_,\_)

next1 = -1

next2 = -1

汇编代码 test.asm

lui $sp,0x1001

j main

main:

addi $s7 $zero 10

addi $s6 $zero 8

sw $s7 8($sp)

sw $s6 12($sp)

Label1:

lw $s7 8($sp)

addi $s6 $zero 0

blt $s7 $s6 Label4

Label2:

lw $s7 12($sp)

sw $s7 24($sp)

sw $s7 28($sp)

sw $sp 16($sp)

addi $sp $sp 16

jal mul

lw $sp 0($sp)

Label3:

lw $s7 8($sp)

addi $s6 $zero 1

sub $s7 $s7 $s6

sw $s7 8($sp)

sw $v0 12($sp)

j Label1

Label4:

j end

mul:

sw $ra 4($sp)

lw $s7 8($sp)

lw $s6 12($sp)

mul $s7 $s7 $s6

add $v0 $zero $s7

lw $ra 4($sp)

jr $ra

end:

错误程序1 test.cmm

int mul(int x, int y)

{

return x\*y;

}

void main(void)

{

int a;

int b;

a=10;

b=8;

while(a>=0)

{

b=mul(b, b);

a=a-1;

return;

}

#

报错信息1：

> Executing task: ./main <

语法错误：不期待的符号#

终端进程“/bin/zsh '-c', './main'”已终止，退出代码: 255。

错误程序2 test.cmm

void main(void)

{

int a;

int b;

a=10;

b=8;

while(a>=0)

{

b=mul(b, b);

a=a-1;

}

return;

}

#

报错信息2:

> Executing task: ./main <

语法错误：未声明的函数mul

终端进程“/bin/zsh '-c', './main'”已终止，退出代码: 255。

错误程序3 test.cmm

int mul(int x, int y)

{

return x\*y;

}

void main(void)

{

a=10;

b=8;

while(a>=0)

{

b=mul(b, b);

a=a-1;

}

return;

}

#

报错信息3:

> Executing task: ./main <

语法错误：变量a未声明

终端进程“/bin/zsh '-c', './main'”已终止，退出代码: 255。

**六、实践结论**

根据上面的实验结果可以看出，C-- 编译器能够将正确的源代码编译为 mips 汇编代码。而面对出现错误的源代码时，编译器会报错并停止编译。较好地达到了实验目的。

但从报错信息中可以看出，C-- 编译器的报错信息不完全正确，一些错误不能够正确的区分。这个问题已经超出了课堂所学的知识范围和我目前的能力水平，以后我也会更广泛地涉猎编译原理的相关知识，争取解决这个问题。

程序代码如下：

主程序 main.cpp

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/main.cpp>

词法分析部分 LexicalAnalyser.cpp

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/LexicalAnalyser.cpp>

中间代码生成部分

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/IntermediateCode.cpp>

语法分析和语义分析部分

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/ParserAndSemanticAnalyser.cpp>

最终代码生成部分

<https://github.com/CrazyDaveHDY/my-compiler/blob/master/ObjectCodeGenerator.cpp>

**七、实践小结**

首先是发现了设计一个编译系统理论和实践上还是有很大的差距，编译原理很多东西在理论上非常简单易懂，但实现过程中却有非常多的细节需要注意，在编写代码的过程中也是遇到了很多的 BUG，这里我举几个典型的例子:

首先是词法分析过程中单字符运算符双字符运算符的区分，最初版本的程序会把一个双字符运算符当作两个单字符运算符进行处理，后面的处理方法是读入到运算符时查看后一个字符能否与当前运算符构成双字符运算符。

还有语法和语义分析过程中函数的处理，函数的处理过程中有很多的细节。比如：函数调用时的参数和函数的返回值都可以是表达式，在最终代码生成的过程中，函数调用时要保护和恢复现场，函数调用时的参数列表要与声明时的参数列表匹配。这些问题的处理花费了我很多时间。

通过这次课程设计，我对一个编译器的组成有了更深层次的认识，并且对编译原理的很多知识有了新的认识。我对字符串处理，正则表达式，自动机理论，数据流分析等方面有了更深入的了解。有了其他课程设计的经验，这一次的编译原理课程设计，我更注重数据结构的设计，数据结构的好坏直接决定了代码的复杂度和代码量。最后通过这次实验也大大提升了我的代码能力，总共编写了 2000-3000 行代码，在今后面对编写代码过程中的很多细节问题我也能够更好地处理。

目前的 C-- 编译器还有很多可以改进的地方，目前的编译器只支持整型，后续可以增加对浮点数的支持和对数组的支持。后续还可以增加对 for 语句和 switch 语句的支持。不过增加新的功能后，文法会变得更为复杂，DFA的状态数会大大增加，代码的编写难度会指数级的增加，为了解决这个问题以后可能要引入 Lex, Bison 等自动分析程序，这也是我以后的学习方向。

正如之前提到的，目前 C-- 编译器的报错信息不完全正确，一些错误不能够正确的区分，而解决这个问题也还需要很多课外的知识。通过这次课程设计我才明白课本上的知识只是冰山一角，在编译器的设计过程中还有很多其他各种各样的问题，还需要我不断的学习。