

华中科技大学

实践课程报告

题目: C—语言编译器

课程名称: 编译原理实践

专业班级: CS1409

学 号: U201414803

姓 名: 许荣仁

指导教师: 邵志远

报告日期: 2016 年 12 月 26 日

计算机科学与技术学院

目录

1 选题背景.....	1
1.1 任务.....	1
1.2 目标.....	1
1.3 源语言定义.....	1
2 实验一 词法分析和语法分析.....	2
2.1 单词文法描述.....	2
2.2 语言文法描述.....	2
2.3 词法分析器的设计.....	3
2.4 语法分析器设计.....	3
2.5 语法分析器实现结果展示.....	3
3 语义分析.....	8
3.1 语义表示方法描述.....	8
3.2 符号表结构定义.....	8
3.3 错误类型码定义.....	8
3.4 语义分析实现技术.....	9
3.5 语义分析结果展示.....	9
4 中间代码生成.....	13
4.1 中间代码格式定义.....	13
4.2 中间代码生成规则定义.....	13
4.3 中间代码生成过程.....	13
4.4 代码优化.....	14
4.5 中间代码生成结果展示.....	14
5 结束语.....	18
5.1 实践课程小结.....	18
5.2 自己的亲身体会.....	18
参考文献.....	19

1 选题背景

1.1 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，提高学生对系统软件编写的能力。

1.2 目标

本次课程实践目标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。

1.3 源语言定义

这次实验要编译的语言是 C 语言的简单集合 C--语言。

2 实验一 词法分析和语法分析

2.1 单词文法描述

在 C—语言中，单词包括关键字（if, else, while 等）、标识符（常量名、变量名、函数名等）、运算符（-, +, * 等）、常数（123, -2.3 等）和界符（逗号、分号和括号等）。

2.2 语言文法描述

语言文法可以分为几个部分：

1 高层语法：

语法单元 Program 是初始语法单元，表示整个程序；

Program 可以产生一个或多个 ExtDefList，表示零个或多个 ExtDef；

ExtDef 表示一个全局变量、结构体或函数的定义；

2 与变量类型有关的文法：

Specifier 是类型描述符，可以产生两种取值，TYPE 表示基本类型 int 或 float，另一种是 StructSpecifier，表示结构体类型；

3 与变量和函数的定义有关的文法：

VarDec 表示对一个变量的定义，该变量可以是一个标识符，也可以是一个标识符后跟着若干对方括号括起来的数字（表示数组）；

FunDec 表示对一个函数头的定义，它包括一个表示函数名的标识符以及由一对圆括号括起来的形参列表，该列表由 VarList 表示（也可以为空）。VarList 包括一个或多个 ParamDec，其中每个 ParamDec 都是对一个形参的定义，该定义由类型描述符 Specifier 和变量定义 VarDec 组成；

4 与语句有关的文法

CompSt 表示一个由一对花括号括起来的语句块，该语句块内部先是一系列变量定义，然后是一系列语句定义 StmtList；

StmtList 是零个或多个语句 Stmt 的组合，也可以是另一个语句块；

5 与表达式有关的文法

表达式的形式有多种，包括：

包含二元运算符的表达式（赋值，逻辑与，逻辑或，s 四则运算等）；

包含一元运算符的表达式（括号，取负，逻辑非等）；

不带运算符但又比较特殊的表达式（函数调用，数组访问等）；

最基本的表达式（常量，普通变量等）；

语法单元 `Args` 表示实参列表，每个实参都可以变成一个表达式 `Exp`；
由于表达式中可以包含各种各样的运算符，为了消除二义性问题，需要给出这些运算符的优先级和结合性；

6 与注释有关的

可以使用两种风格的注释：第一种是双斜线“//”；第二种是“/*”和“*/”。

2.3 词法分析器的设计

词法分析由 `flex` 书写正则表达式完成，先把数字表示出来，然后利用数字再表示出整数，浮点数，把字母表示出来，再进一步表示出标识符，其他需要表示的还有空白，注释等；对不同的单词返回特定的值，就完成了对一段代码的词法分析。

2.4 语法分析器设计

语法分析程序主要分三个部分来实现，第一部分是对 C--语言文法的描述，第二部分是抽象语法树的建立，第三部分是抽象语法树的遍历。

首先要定义运算符的优先级和结合性，消除移进/规约冲突，需要在 `bison` 文件的前部指明：

主体部分是对 C--语言文法的完整描述，根据实验指导书后面的附录就可完成，额外的工作是优先给予结合性的定义；

语法树的生成是穿插在每一条语法规则之中的，树的建立顺序是自底向上的，`flex` 完成了词法分析，给每一个词法单元建立了对应的节点，是语法树的叶子节点；在 `bison` 中，由词法单元的叶节点规约出父节点，再由新的父节点与其他节点规约出新的父节点，不断规约，直到规约出初始语法单元 `Program`，表示语法树建立完成；在建立语法树时，使用的表示树的方法是孩子兄弟表示法，对每一条语法规则来说，左部是父节点，右部是孩子节点，用此方法表示，则是将右侧第一个节点设为父节点的左孩子，第二个节点设为第一个节点的右孩子，也就是逻辑意义上的兄弟，第三个节点设为第二个节点的右孩子，依次操作，直到最后一个节点；

在语法树建立完成后，需要对语法树进行遍历，遍历采用递归的方法，只需要从语法初始符号开始，把当前节点的信息打印完成（除产生空的语法单元不需要打印信息外，其他的需要打印语法单元的名字行号等，如果是常量数字等，要打印出它的值），再依次（递归）遍历左孩子节点和右孩子节点。

2.5 语法分析器实现结果展示

如图 2-1(a)与 2-1(b)，是语法分析程序对一样例代码的运行结果（样例如图 2-2 所示），通过两者之间的对比可以看出，分析程序完成了要求，比较清晰地打印出了语法树的结构，由 Program 单元也就是树根开始，一级一级推导出了整个语法树，其中中间的语法单元和运算符界符等打印出所在行号，变量常量等打印出语法单元的名称和值。由于整个语法树太长，故只截取了前一部分和后一部分。

```

./parser test2.c
打印syntax tree:
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE :int
      FunDec (1)
        identifier :main
        LB (1)
        RB (1)
      CompSt (1)
        LD (1)
        DefList (2)
          Def (2)
            Specifier (2)
              TYPE :int
            DecList (2)
              Dec (2)
                VarDec (2)
                  identifier :num1
                AS (2)
                Exp (2)
                  intnumber :3
            FH (2)
          DefList (3)
            Def (3)
              Specifier (3)
                TYPE :float
              DecList (3)
                Dec (3)
                  VarDec (3)
                    identifier :cd
                  AS (3)
                  Exp (3)
                    floatnumber :0.900000
          FH (3)
        FH (3)
  FH (1)

```

图 2-1(a) 语法分析程序的运行结果


```

        Exp (10)
          intnumber :1
        RB (10)
        Stmt (10)
          CompSt (10)
            LD (10)

          StmtList (11)
            Stmt (11)
              Exp (11)
                Exp (11)
                  identifier :num1
                AS (11)
              Exp (11)
                intnumber :2
              FH (11)

            RD (12)

          RD (13)

        StmtList (14)
          Stmt (14)
            RETURN (14)
          Exp (14)
            intnumber :0
          FH (14)

        RD (15)
syntax tree打印完毕！

```

图 2-1(b) 语法分析程序的运行结果

```

1  int main() { //ve li ui vu ui
2      int num1 = 3;
3      float cd=0.9;
4      struct andj{
5          int a;
6          float bhj;
7      } sg;
8      if(a=1){
9          num1 = 1+num1;
10         if(b=1) {
11             num1 = 2;
12         }
13     }
14     return 0;
15 }
16

```

图 2-2 语法分析的样例代码

对于有错误的源程序，分析程序可以给出错误所在的位置，如图 2-3 所示，这是一个有多个错误的源程序，用分析程序对它进行分析，运行结果如图 2-4 所示：找到了全部三个错误并精确给出了错误行号列号。

```

int main() {
    float i=0;
    int b=5, c[i];
    b= c ) 4;
    c= i + b;
    c=i * int;
    b=6;
}

```

图 2-3 错误检测

```
~/Work/Source/Mylibs/compilers ▶ complier-syntax ●  
./parser test.c  
3:13:error: syntax error, unexpected identifier, expecting intnumber  
4:7:error: syntax error, unexpected RB  
4:9:error: syntax error, unexpected intnumber  
6:8:error: syntax error, unexpected TYPE
```

图 2-4 错误检测

3 语义分析

3.1 语义表示方法描述

用于语义分析的理论工具是属性文法，核心思想是为上下文无关文法中的每一个终结符或非终结符赋予一个或多个属性值。终结符的属性值通过词法分析就可以得到，非终结符的属性值通过产生式对应的语义动作来计算。

3.2 符号表结构定义

符号表对编译器至关重要，编译器使用符号表来记录记录源程序中各种名字的特性信息。对符号表的操作包括填表和查表两种，当分析到程序中的说明或定义语句时，应该进行填表操作，查表操作使用更多，填表之前要先查表，生成目标指令时也要查表。此次实验使用的数据结构是线性链表，把同一类型的符号串在同一条链表中，插入新符号只要将该符号放在链表的表头。在本次实验中分了变量、函数、形参、数组和结构体等不同的链表。

3.3 错误类型码定义

错误类型有 17 种，具体如下：

- 1) 变量未定义 Error type 1 at Line XX: Undefined variable “xx”.
- 2) 函数未定义 Error type 2 at Line XX: Undefined function “xx”.
- 3) 变量重复定义 Error type 3 at Line XX: Redefined variable “xx”.
- 4) 函数重复定义 Error type 4 at Line XX: Redefined function “xx”.
- 5) 赋值类型不匹配 Error type 5 at Line XX: Type mismatched for assignment.
- 6) 不能被赋值的数被赋值 Error type 6 at Line XX: The left-hand side of an assignment must be variable.
- 7) 运算符的操作数类型不匹配 Error type 7 at Line XX: Type mismatched for operands.
- 8) 函数返回类型不匹配 Error type 8 at Line XX: Type mismatched for return.
- 9) 函数参数错误 Error type 9 at Line XX: “xx” is not applicable for arguments“(type,type)”.
- 10) 非数组以数组形式使用 Error type 10 at Line XX: “xx” is not an array.
- 11) 非函数以函数形式调用 Error type 11 at Line XX: “xx” is not a function.

- 12) 非整数作为数组下标 Error type 12 at Line XX: “xx” is not an integer.
- 13) 非结构体以结构形式调用 Error type 13 at Line XX: Illegal use of “.”.
- 14) 未定义的域 Error type 14 at Line XX: Non-existent field “xx”.
- 15) 域重复定义 Error type 15 at Line XX: Redefined field “xx”.
- 16) 结构体名重复 Error type 16 at Line XX: Duplicated name “xx”.
- 17) 结构体未定义 Error type 17 at Line XX: Undefined structure “xx”.

3.4 语义分析实现技术

主要使用语法结点的综合属性，在.y 文件的代码中插入语义分析的相关代码，如：在语法规则中，变量、函数、结构体、数组定义的规则后面，先检测这个符号是否重复定义，如果发现重复定义，则报对应的语义分析错误（但是查询到错误后不要立即退出，要进行错误恢复后继续分析），如果没有定义，则将其添加到相应的符号链表中。在每个语法结点中添加一个标志位，来区别常量、变量、函数、结构体和数组。在不同的结点中，需要进行的操作也有区别，比如在给结构体变量赋作用域时，应当等当前结构体内所有变量分析完，再回退遍历符号表，在变量后面附上当前结构体的作用域，其他的结点处理方法也类似。当遇到 ExtDef 或者 Def 语法单元时，说明此结点包含了定义信息，需要对子节点进行遍历找出符号插入符号表。需要对子节点进行使用时，先检查符号表以确定当前变量是否定义以及它们的类型是否匹配，出现问题则需要及时报错。

3.5 语义分析结果展示

对一个文件启动语义分析程序，运行结果如图 3-1 所示，被分析的源程序如图 3-2 所示。

变量符号表

类型	变量名	作用域
int	x	point
float	y	point
int	rad	circle
int	num1	global
float	flo1	global

函数符号表

类型	函数名	形参个数
int	addfun	2
int	main	0

形参符号表

类型	形参名	作用域
int	a	global
int	b	global

数组符号表

类型	变量名	元素个数	作用域
int	nums	10	global
float	flos	5	global

结构体符号表

类型	结构名
struct	point
struct	circle

图 3-1 语义分析结果

```

1  struct circle {
2      struct point {
3          int x;
4          float y;
5      };
6      int rad;
7  };
8  int addfun(int a,int b){
9      a = a + b;
10     return a;
11 }
12 int main(){
13     int num1;
14     float flo1;
15     int nums[10];
16     float flos[5];
17     num1 = 3;
18     nums[1] = 4;
19     if(num1 == 1) {
20         flos[2] = flo1 + flos[1];
21         if(num1 == 1) {
22             num1 = 4;
23         }
24     }
25 }
26

```

图 3-2 被分析的源程序

对于存在语义错误的代码，程序也可以给出错误信息，错误的代码如图 3-3 所示，程序给出的错误信息如图 3-4 所示。

<pre> 1 int main(){ 2 int i = 0; 3 j = i + 1; 4 } </pre>	<pre> 1 int main(){ 2 int i = 0; 3 inc(i); 4 } </pre>
<pre> 1 int func(int i){ 2 return i; 3 } 4 int func() { 5 return 0; 6 } 7 int main(){ 8 9 } </pre>	<pre> 1 struct Position { 2 float x; 3 float y; 4 }; 5 int main(){ 6 struct Position p; 7 if(p.n == 3.7) 8 return 0; 9 } </pre>

图 3-3 存在错误的代码

```

test1.c
Error type 1 at line 3: 'j' Undefined variable

test2.c
Error type 2 at line 3: Undefined function 'inc'
~/Work/Source/MyLab/compiler2

Error type 4 at line 4: Redefined function 'func'
~/Work/Source/MyLab/compiler2

test14.c
4 Error type 1 at line 7: 'n' Undefined variable
5 Error type 14 at line 7: Non-existent field 'n'

```

图 3-4 对错误代码的分析结果

4 中间代码生成

4.1 中间代码格式定义

中间代码的格式，主要有以下几种三地址码：

语法	描述
LABEL x:	定义标号 x
FUNCTION f:	定义函数 f
x := y	赋值
x := y op z	运算
GOTO x	无条件跳转
IF x [relop] y GOTO z	x 与 y 满足[relop]关系则跳转
x := &y	取地址赋值
x := *y	取内存单元的值赋值
*x := y	赋值到内存单元中
RETURN x	函数返回

4.2 中间代码生成规则定义

中间代码的功能是把输入的 C--语言代码解析为三地址码，算术运算可能需要拆分，把嵌套的运算拆成两个一组，期结果保存在中间变量中再与其余的符号进行运算；对于 if 判断语句，如果存在多个条件，要把判断条件分开，每个三地址码只能判断一个条件进行跳转或不跳转；while 循环语句，在循环体的前面加一个标号，在判断还不能跳出循环的时候跳转到标号。

4.3 中间代码生成过程

生成的方法仍然是遍历语法树的每一个结点，当发现语法树中有特定的结构出现时，就借用语法制导翻译工具产生相应的中间代码。基本的运算表达式语句、复合语句、返回语句、跳转语句循环语句、函数调用等都有各自独特的翻译模式，按照它们的模式就可以翻译出对应的中间代码，在表达式的翻译中，用到了继承属性处理内容，所以没有使用代码回填。

4.4 代码优化

代码优化的技术有很多，包括删除冗余的取存，常量合并，代数化简，常量传播、控制流优化、强度削弱、死代码删除、代码外提等，但是由于时间不足，本次实验并没有把代码优化做出来。

4.5 中间代码生成结果展示

对代码进行中间代码生成的操作，源代码如图 4-1（对 if 语句的测试）和图 4-3（while 语句）所示，生成的中间代码如图 4-2 和图 4-4 所示。

```
1  int b;  
2  int main()  
3  {  
4      int n;  
5      int a=0;  
6      b=3;  
7      if (n > 0) a = 1;  
8      else if (n < 0) a = 2 * b;  
9          else a = b + 3;  
10     if(a==b || b!=n)  
11         b = 5;  
12     else  
13         b = b + 2;  
14     return 0;  
15 }
```

图 4-1 测试源代码（if 语句）

```

FUNCTION      main      :
a      :=      #0
b      :=      #3
IF      n      >      #0      GOTO      label1
GOTO      label2
LABEL      label1      :
a      :=      #1
GOTO      label3
LABEL      label2      :
IF      n      <      #0      GOTO      label4
GOTO      label5
LABEL      label4      :
a      :=      #2      *      b
GOTO      label6
LABEL      label5      :
a      :=      b      +      #3
LABEL      label6      :
LABEL      label3      :
IF      a      ==      b      GOTO      label7
GOTO      label9
LABEL      label9      :
IF      b      !=      n      GOTO      label7
GOTO      label8
LABEL      label7      :
b      :=      #5
GOTO      label10
LABEL      label8      :
b      :=      b      +      #2
LABEL      label10      :
RETURN      #0

```

图 4-2 if 语句测试结果

```
1  int add(int temp[2])
2  {
3      return (temp[0] + temp[1]);
4  }
5  int main()
6  {
7      int op [2];
8      int i = 0, j = 0;
9      while (i<2)
10     {
11         while (j<2)
12         {
13             op[j] = i + j;
14             j = j + 1;
15         }
16         i = i + 1;
17         j = 0;
18     }
19     return 0;
20 }
```

图 4-3 while 语句源代码

```

FUNCTION      add      :
PARAM      temp
t5      :=      #0      *      #4
t6      :=      temp      +      t5
t10     :=      #1      *      #4
t11     :=      temp      +      t10
t1      :=      *t6      +      *t11
RETURN      t1
FUNCTION      main      :
DEC      t12      8
op      :=      &t12
i      :=      #0
j      :=      #0
LABEL      label1      :
IF      i      <      #2      GOTO      label2
GOTO      label3
LABEL      label2      :
LABEL      label4      :
IF      j      <      #2      GOTO      label5
GOTO      label6
LABEL      label5      :
t20     :=      j      *      #4
t21     :=      op      +      t20
t22     :=      i      +      j
*t21     :=      t22
j      :=      j      +      #1
GOTO      label4
LABEL      label6      :
i      :=      i      +      #1
j      :=      #0
GOTO      label1
LABEL      label3      :
RETURN      #0

```

图 4-4 while 语句测试结果

5 结束语

5.1 实践课程小结

这次的实验本来有四个部分，词法语法分析、语义分析、中间代码生成和目标代码生成，很遗憾的是自己并没有完整地把实验做完，只完成了前面的三部分，目标代码生成还没有做，中间代码生成也还没有优化。就从前面的部分来说，这次实验主要完成的工作有：

词法分析，把源程序的 C 减减语言代码分析成 token 序列；语法分析，根据词法分析的结果构建语法树并遍历打印；语义分析，根据语法分析的结果进一步分析程序的语义是否正确并建立打印符号表；中间代码生成，把没有语法语义错误的代码生成四元式。

中间代码优化的工作还没有完成，直接生成的中间代码会比较繁琐，由很多无用的代码，优化后可以提高代码的效率，这一部分与最后的目标代码生成还有待以后的工作。

5.2 自己的亲身体会

通过这次实验，自己学到了很多，首先就是对工具的使用，flex 和 bison 两个工具在之前从未接触过的，但是通过这次实验，至少是已经学会了它们最基本的用法，也许还有更多的功能以后可以用到。另外的收获就是加深了对课本知识的理解，也比较深刻地理解了编译器的工作原理和过程，从高级语言到机器能够执行的目标代码，每一步都是非常严格巧妙的工作，而这一切都离不开编译器的工作。

总的来说，虽然感觉这个实验的难度是有些大的，但是带来的收获也很大，提高了自己的编程水平和分析问题的能力，再就是时间有点不足，做实验的过程比较繁琐，课下花费的时间比较多，如果时间充足，希望可以做得更好一些。。

参考文献

- [1] 吕映芝等. 编译原理(第二版). 北京: 清华大学出版社, 2005
- [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