一、实验目的

根据 C 语言设计编译器，包含词法分析、语法分析、语义分析、中间代码生成及优化部分。学习汇编语言，将所生产中间代码转换为汇编语言，所得汇编语言用汇编器转换为二进制程序后运行。

二、实验内容及要求

1. 词法分析器

输入源程序，输出对应的 token 表，符号表和词法错误信息。过滤空白符，跳过注释、换行符等无用符号；对源代码进行行列计数，用于指出含有错误的行列号，进行提醒。

2. 语法分析器

输入 token 串，通过语法分析，寻找其中语法错误。自行实现文法，进行自底向上的移入归约，构建语法分析树。文法实现基本的语法单位分析：6 种算术运算（+，-，*，/，%，^）、6 种关系运算（==，>，<，>=，<=，!=）、3 种逻辑运算（&&（与），||（或），!（非））、if 语句、else 语句、else if 语句、for 语句、while 语句、声明语句、赋值语句以及输入输出语句。

3. 语义分析和中间代码生成

在词法分析返回词法单元时，我们附带着返回一些额外的信息，诸如类型、数值和运算符等。在语法分析构建语法树时，我们利用从词法分析返回的附加信息，在规约时添加相应的语义动作，对类型等语义规则进行分析，在出现错误时给出相应的错误提示。

此外，除了在语法分析的同时进行语义分析，在完成抽语法树的构建后，我们还将对语法树进行遍历，同时为变量、函数、数组和指针等建立相应的符号表，进行全局的语义分析，以及提供更为完备的错误提示。通过建立相应的符号表，我们在语义层面上对变量的定义与否、类型问题、函数参数的检查、函数定义与否等，进行信息维护，从而对变量未定义、变量重复定义、函数参数不匹配、函数未定义、函数已定义进行错误信息的收集以及报告。

4. 中间代码优化

输入三地址代码（选用四元式）序列，根据变量的活跃性，我们恰当的删除在基本块出口处不活跃的变量的定值语句，并且回溯进行连锁删除非活跃变量的定值语句。为了保证跳转指令的正常，我们又遍历跳转指令的目标标号对应的代码，如果已经删除便将其恢复。

三、实验方案设计

1. 词法分析程序设计

（1）工具介绍：采用 lex 进行实现。Lex 是一种生成扫描器的工具，常用 yacc 结合使用。其实现主要为两步，第一，以 Lex 可以理解的格式指定模式相关的动作，第二，在这一文件上运行 Lex，生成扫描器的 C 代码。

（2）设计原理：根据实验要求设计 lex 词法规则段，对于注释、空白符、数据类型(int、char、short、long 等)、布尔值(true、false)、控制流关键词(case、if、else、while、do、for、goto、return 等)、

运算符(++、--、>>、<<、<=、>=、==、!=、=、[]、&&、||、&、!、+、-、*、/、%、^、<、>、|等)、输入输出函数名、标识符、数字、字符串进行处理，返回 token 序列，同时对于非法格式调用 yyerror 进行错误处理。

（3）输入输出效果图：

```
int main() {
    int a = 1, b = 2;
    int c = 2;
    int i = 0;
    for (i = 0; i < 10; i = i + 1) {
        if (a < 5) a = a + 1;
        else if (a >= 5) {
            a = a + 2;
        }
    }
    // 或者是其他自定义的输出函数
    print_int(i);
    while (a < 40) {
        a = a + 1;
    }
    print_int(a);
    return 0;
}
```

单词	词素	属性
INT	int	
MAIN	main	
LP	(	
RP	)	
LBRACE	{	
INT	int	
ID	a	-529004800
ASSIGN	=	
NUMBER	1	1
COMMA	,	
ID	b	-529004736
ASSIGN	=	
NUMBER	2	2
SEMICOLON	;	
INT	int	
ID	c	-529004672
ASSIGN	=	
NUMBER	2	2
SEMICOLON	;	
INT	int	
ID	i	-529004608
ASSIGN	=	
NUMBER	0	0
SEMICOLON	;	
FOR	for	
LP	(	
ID	i	-529004608
ASSIGN	=	
NUMBER	0	0
SEMICOLON	;	
ID	i	-529004608
LESS	<	

2. 语法分析程序设计

（1）工具介绍：采用 yacc 实现。Yacc 的 GNU 版叫做 Bison。它是一种工具，将任何一种编程语言的所有语法翻译成针对此种语言

的 Yacc 语法解析器。它用巴科斯范式(BNF)来书写。用 Yacc 来创建一个编译器包括四个步骤：

第一，通过在语法文件上运行 Yacc 生成一个解析器。

第二，编写一个 .y 的语法文件，将 lex 处理输入后的标记传递给解析器，编写一个函数，通过调用 yyparse() 来开始解析，编写错误处理例程（如 yyerror()）。

第三，编译 Yacc 生成的代码以及其他相关的源文件。第四，将目标文件链接到适当的可执行解析器库。

（2）设计原理:根据课上所学，自行设计文法，总体思路为，C 语言源码由多个函数构成，函数按照函数定义格式组成，其中函数体部分由多条语句组成。语句包含 if 语句、if-else 语句、while 语句、for 循环语句、变量声明语句、数组声明语句、return 返回语句和表达式。而表达式包括布尔表达式，函数调用表达式、算术表达式、赋值表达式、标识符、数字、字符串、数组取值、指针、取地址等。

（3）设计特点：

语法分析中，通过规定结合性和优先级，解决了运算符运算顺序的问题，从而正常构建表达式相关的语法树，对于复杂表达式也可正常求值。同时规定词素的优先级以及结合性如 else，我们解决了“else”悬挂问题。通过建立相关的错误产生式（包括利用内置的 error 非终结符），我们得以在问题出现时， 知道在源代码中的何处有此问题，从而在词法分析和语法分析中给出一定错误提示。为了使得错误提示更加醒目，我们将错误提示以错误输出流的方式输出

到屏幕，同时以特殊控制字符将错误信息标志为红色。

（4）已实现语法：

if-elseif-else 语句

do-while\ while 语句

for 循环语句

变量、数组、指针的定义以及使用

支持函数定义以及函数的调用

支持众多的表达式以及相关运算符

3. **语义分析的设计**

（1）设计原理：在语法分析的基础上，我们为每一个词法单元附上一个类型，从而在词法分析返回词法单元时，我们可以从词法分析之时所处的上下文环境中，将更多的信息随着词法单元的返回一同传递给语法树。

（2）具体实现方法：

在词法单元具有诸如类型、常量、标识符、运算符等相关信息后，我们在产生式自底向上规约时，在产生式的最后附加语义动作，进行类型等语义检查。

为了进行后续的语法制导翻译，我们仍然建立了抽象语法树，通过对抽象语法树的多趟遍历，建立完备的符号表，从而根据符号表，检查变量的作用域，类型信息等，对变量的定义和使用，检查重复定义、未定义的错误和函数参数错误。通过为函数建立相关符号表，我们在函数的使用上进行了参数的检查以及相关错误的提示。

4. **中间代码的生成级优化程序设计**

（1）设计原理：在中间代码生成时，我们利用语法分析和语义分析建立的抽象语法树，结合回填的方式，对指令标号以及跳转指令的标号进行回填。

（2）具体实现方法：

由于受限于 yacc 自底向上的语法分析过程，在语法树上进行语法制导翻译过程中，会造成中间代码乱序问题，因此我们必须在每生成一条中间代码时，为其标志序号，遍历结束后根据序号排序。同时在“归约”诸如 if-else 等具有复杂跳转指令的语句时，必须在恰当位置及时的回填目标标号。

在处理诸如 a and b，c or d 的具有跳转指令的条件判断表达式时，使用的是递归回填标号的方式，从而弥补了 yacc 不能推出空产生式也即无标记非终结符的问题。

在跳转指令的优化方面，我们充分地利用程序会顺序向下执行的特点，对于条件判断的跳转指令做到了去冗余以及条件跳转指令的精简化。

在代码优化方面， 我们根据活跃变量与非活跃变量，恰当的删除非活跃变量的定值语句，并且递归的回溯查找非活跃变量，从而一定程度上减少了临时变量的产生，提高生成汇编代码时的效率。

5. **汇编生成程序的设计**

（1）设计原理

选用精简指令集 MIPS 指令集，MIPS 包含 32 个通用寄存器，这些寄存器的用法都遵循一系列约定，按照规定，通过对中间代码（四元式序列）进行格式识别，寄存器分配，翻译为对应的汇编指令。

（2）具体实现：以下为所写转换程序，能够识别的四元式格式。(因为 MIPS 立即数寻址和寄存区寻址不同，因此要特殊考虑。)

['15:', '=', '#1', '_', 'a']

['15:', '=', 't1', '_', 'a']

['2:', '+', 'a', '#1', 't2']

['2:', '+', '#1', 'a', 't2']

['2:', '+', 'b', 'a', 't2']

['2:', '-', 'a', '#1', 't2']

['2:', '-', '#1', 'a', 't2'] ['2:', '-', 'b', 'a', 't2']

['3:', '*', 't2', 'a', 't3']

['3:', '/', 't2', 'a', 't3']

['3:', '%', 't2', 'a', 't3']

['3:', '<', 'var0', 't4', 't5']

['3:', '>', 'var0', 't4', 't5']

['1:','CALL','INPUT/PRINT','_',t1]

['1:','CALL','fun','_',t1]

['1:','CALL','INPUT/PRINT','_','_']

['1:','CALL','fun','_','_']

['1:','GOTO','_','_','Label'+2]

['1:','RETURN','t1','_','_']

['1:','JEQ','t1','t2','Label'+2]

['1:','JNE','t1','t2','Label'+2]

['1:','J>','t1','t2','Label'+2]

['1:','J<','t1','t2','Label'+2]

['1:','JGE','t1','t2','Label'+2]

['1:','JLE','t1','t2','Label'+2]

['1:','FUNCTION','_','_','main']

['1:','ARG','var2','_','_']

['1:','param','_','_','var2']

['1:','^','a','b','t1']

四、结果及测试分析

1. 测试数据说明

（1）sample.c 其中包含基本运算以及 while、for、if、print 等语句。其输出结果为：10 40。

```
int main() {
    int a = 1, b = 2;
    int c = 2;
    int d;
    int i = 0;
    for (i = 0; i < 10; i = i + 1) {
        if (a < 5) {
            a = a + 1;
        }
        else if (a >= 5) {
            a = a + 2;
        }
    }
    // 或者是其他自定义的输出函数
    print(i);
    while (a < 40) {
        a = a + 1;
        //print(a);
    }
    print(a);
    return 0;
}
```

（2）array.c 用于测试数组以及指针。输出结果应为 6、5。

```c
int main()
{
    int a[2];
    a[1]=2;
    print(a[1] * 3);
    a[1]=333;

    int *p=a;
    p[0]=1;
    p[1]=p[0]+3;
    print(p[1] + 1);
    return 0;
}
```

（3）function.c 为计算阶乘的程序，根据用户输入的数字计算阶乘。 例如：输入 4 输出 24

```c
int fact(int n){
        int temp;
    if(n==1){
        return n;
    }
    else{
        temp=(n*fact(n-1));
        return temp;
    }
}

int main()
{
    int result;
    int i;
    int m = input();
    if( m > 1) {
        result=fact(m);
    }
    else {
         result = 1;
    }
    print(result);
    return 0;
}
```

（4）wrong.c 用于进行错误测试。测试时应只保留一个错误，观察结果，多个错误，程序遇到的首个错误便会退出。输出：参数个数不匹配。

```
int fun(int c)
{
    print(c);
}


int main()
{
    //错误：类型错误
    //int a = 1.2;

    //错误：语句后使用了冒号而非分号
    //int a,b = 1:

    //错误：变量未定义
    //a = c + 10;

    //错误：变量已定义
    //int a = b;

    //错误：传参个数不匹配
    fun();

    //错误：函数未定义
    //fuc();

    //错误：传入参数为空
    //print();

    //错误：return 语句漏掉了分号
    return 0;
}
```

2. **运行结果及功能说明**

（1）sample.c

中间代码（四元式序列）：

```
0:      FUNCTION        _       _       main
1:      =       #1      _       temphh1
2:      =       temphh1         _       a
3:      =       #2      _       temphh2
4:      =       temphh2         _       b
5:      =       #2      _       temphh3
6:      =       temphh3         _       c
7:      =       #0      _       temphh4
8:      =       temphh4         _       i
9:      =       #0      _       temphh5
10:     =       temphh5         _       i
11:     =       #10     _       temphh6
12:     J<      i       temphh6         14
13:     GOTO    _       _       30
14:     =       #5      _       temphh7
15:     J<      a       temphh7         17
16:     GOTO    _       _       20
17:     =       #1      _       temphh8
18:     +       a       temphh8         temphh9
19:     =       temphh9         _       a
20:     =       #5      _       temphh10
21:     JGE     a       temphh10        23
22:     GOTO    _       _       26
23:     =       #2      _       temphh11
24:     +       a       temphh11        temphh12
25:     =       temphh12        _       a
26:     =       #1      _       temphh13
27:     +       i       temphh13        temphh14
28:     =       temphh14                i
```

汇编程序：

```
main:
Label0:
Label1:
        li $t1,1
Label2:
        move $t2,$t1
Label3:
        li $t1,2
Label4:
        move $t3,$t1
Label5:
        li $t1,2
Label6:
        move $t4,$t1
Label7:
        li $t1,0
Label8:
        move $t5,$t1
Label9:
        li $t1,0
Label10:
        move $t5,$t1
Label11:
        li $t1,10
Label12:
        blt $t5,$t1,Label14
Label13:
        j Label30
```

运行结果：

```
sample.c asm result:
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

10
40
```

（2）array.c

中间代码（四元式序列）：

```
0:      FUNCTION        _       _       main
1:      =       #2      _       temphh2
3:      =       #3      _       temphh3
4:      *       temphh2         temphh3         temphh4
5:      ARG     temphh4         _       _
6:      CALL    PRINT   _       _
7:      =       #333    _       temphh5
9:      =       #1      _       temphh7
11:     =       #3      _       temphh8
12:     +       temphh7         temphh8         temphh9
14:     =       #1      _       temphh10
15:     +       temphh9         temphh10        temphh11
16:     ARG     temphh11        _       _
17:     CALL    PRINT   _       _
18:     =       #0      _       temphh12
19:     RETURN  temphh12        _       _
```

汇编程序：

```
main:
Label0:
Label1:
        li $t1,2
Label3:
        li $t2,3
Label4:
        mul $t3,$t1,$t2
Label5:
        move $t0,$a0
        move $a0,$t3
Label6:
        addi $sp,$sp,-4
        sw $ra,0($sp)
        jal PRINT
        lw $ra,0($sp)
        addi $sp,$sp,4
Label7:
        li $t1,333
Label9:
        li $t1,1
Label11:
        li $t2,3
Label12:
        add $t3,$t1,$t2
```

运行结果：

```
array.c asm result:
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

6
5
```

（3）function.c

中间代码（四元式序列）：

```
0:      FUNCTION        _       _       fact
1:      param   _       _       n
2:      =       #1      _       temphh1
3:      JEQ     n       temphh1         5
4:      GOTO    _       _       7
5:      RETURN  n       _       _
6:      GOTO    _       _       14
7:      =       #1      _       temphh2
8:      -       n       temphh2         temphh3
9:      ARG     temphh3         _       _
10:     CALL    fact    _       temphh4
11:     *       n       temphh4         temphh5
12:     =       temphh5         _       temp
13:     RETURN  temp    _       _
14:     RETURN  0       _       _
15:     FUNCTION        _       _       main
16:     CALL    INPUT   _       temphh6
17:     =       temphh6         _       m
18:     =       #1      _       temphh7
19:     J>      m       temphh7         21
```

汇编程序：

```
fact:
Label0:
Label2:
        li $t1,1
Label3:
        beq $a0,$t1,Label5
Label4:
        j Label7
Label5:
        move $v0,$a0
        jr $ra
Label6:
        j Label14
Label7:
        li $t1,1
Label8:
        sub $t2,$a0,$t1
Label9:
        move $t0,$a0
        move $a0,$t2
Label10:
        addi $sp,$sp,-24
        sw $t0,0($sp)
        sw $ra,4($sp)
        sw $t1,8($sp)
        sw $t2,12($sp)
        sw $t3,16($sp)
        sw $t4,20($sp)
        jal fact
        lw $a0,0($sp)
        lw $ra,4($sp)
```

运行结果：

```
function.c asm result:
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter an integer:4
24
```

（4）wrong.c

运行结果：

```
wrong.c :
参数个数不匹配!
```

五、 成员分工

词法分析：合作完成

语法设计：合作完成

中间代码生成：合作完成

中间代码转汇编：合作完成

六、总结及心得体会

通过此次实验，我们将课上所学应用于实践，对于编译器构造过程，诸如词法分析、语法分析、语义构造、中间代码生成等有了充分的了解，对于汇编语言进行了相应学习，了解了有关寄存器分配、底层语言逻辑构造等，有了很大的收获！

七、源码

见附件

八、参考源码

汇编程序参考源码地址如下

https://github.com/sagark/MARS-app