《编译技术》课程 设计文档

学号：\_\_\_\_\_\_\_\_\_13061073\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_\_郝元桢\_\_\_\_\_\_\_\_\_\_\_

2015年 11月 日

## 一．需求说明

### 1．文法说明

<程序> ::= <分程序>.

语法分析：程序主体部分为分程序，以 . 结束。

句子实例：var x :integer;

begin

x=1;

end .

<分程序> ::= [<常量说明部分>][<变量说明部分>]{[<过程说明部分>]| [<函数说明部分>]}<复合语句>

语法分析：分程序按照顺序由常量说明部分(可有可无)，变量说明部分(可有可无)，过程说明部分或函数说明部分(0个或以上随意组成)，最后必须有一个复合语句。

句子实例：const x=1,y=2;

var a,b:integer;

begin

a:=x+y;

b:=x

end

<常量说明部分> ::= const<常量定义>{,<常量定义>};

语法分析：常量说明部分由const作为开始标志，后面必须有一个或多个常量定义，由逗号隔开，以分号结束。

句子实例：const x=1,y=2;

<常量定义> ::= <标识符>＝ <常量>

语法分析：常量定义必须按顺序由标识符，=号和常量组成，并且各有一个。

句子实例：x=0

<常量> ::= [+| -]<无符号整数>|<字符>

语法分析：常量由+或-（可有可无）与无符号整数组成，或者单由字符构成。

句子实例：+10 或 ‘a’等

<字符> ::= '<字母>' | '<数字>'

语法分析：字符由单引号括住的一个字母或一个数字构成。

句子实例： ‘b’ ‘2’

<字符串> ::= "{十进制编码为32,33,35-126的ASCII字符}"

语法分析：字符串由双引号括住的0个或多个十进制编码为32,33,35-126的ASCII字符构成。

句子实例： “I am here!”

<无符号整数> ::= <数字>{<数字>}

语法分析：无符号整数由一个或多个数字构成。

句子实例：21 或 098

<标识符> ::= <字母>{<字母>|<数字>}

语法分析：标识符由一个字母开头，后面可以有0个或多个字母或者数字的随意排列。

句子实例：ab 或 a1 ，又如0a就是错误的

<变量说明部分> ::= var <变量说明> ; {<变量说明>;}

语法分析：变量以var作为开始，后面有一个或多个以分号结尾的变量说明。

句子实例：var x :integer; c :char;

<变量说明> ::= <标识符>{, <标识符>} : <类型>

语法分析：变量说明由一个或多个标识符开始，中间用逗号隔开，再由一个冒号作为标识符与类型的分界，类型作为结尾只有一个。

句子实例：x,y,z :integer

<类型> ::= <基本类型>|array'['<无符号整数>']' of <基本类型>

语法分析：类型由基本类型或array[无符号整数]of基本类型(数组)构成。

句子实例：integer 或 array[3] of integer

<基本类型> ::= integer | char

语法分析：integer或char

句子实例：char

<过程说明部分> ::= <过程首部><分程序>{;<过程首部><分程序>};

语法分析：过程说明部分由一个或多个过程首部和分程序构成，以分号进行分割，并且以分号结束。

句子实例：procedure example;

分程序;

<函数说明部分> ::= <函数首部><分程序>{;<函数首部><分程序>};

语法分析：函数说明部分由一个或多个函数首部和分程序的组合组成，并以分号分割每个组合，最后必须有一个分号作为结束。

句子实例：function example : integer;

分程序;

<过程首部> ::= procedure<标识符>[<形式参数表>];

语法分析：过程首部必须由procedure开始，加一个标识符，可以有或没有形式参数表，最后以分号结束。

句子实例：procedure example(var x:integer);

<函数首部> ::= function <标识符>[<形式参数表>]: <基本类型>;

语法分析：函数首部必须以function开始，加一个标识符，可以有或没有形式参数表，再以一个冒号分割，后面有一个函数返回值的基本类型，最后以分号结束。

句子实例：function example(var y:char):integer;

<形式参数表> ::= '(' <形式参数段>{; <形式参数段>}')'

语法分析：形式参数表是由小括号括住的一个或多个形式参数段组成，每个形式参数段由分号分割，并且最后一个形式参数段后面没有分号。

句子实例：(var x:integer; c:char; var i,j:char)

<形式参数段> ::= [var]<标识符>{, <标识符>}: <基本类型>

语法分析：形式参数段开头可以有一个或没有var，接着是用一个或多个标识符组成，用逗号隔开，以冒号结尾，冒号后面写一个基本类型。

句子实例：var x,y:integer

<语句> ::= <赋值语句>|<条件语句>|<情况语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<for循环语句>|<空>

语法分析：语句可以为赋值语句、条件语句、情况语句、过程调用语句、复合语句、读语句、写语句、for循环语句或空语句中的一个。

句子实例：x:=1

<赋值语句> ::= <标识符> := <表达式>| <函数标识符> := <表达式> | <标识符>'['<表达式>']':= <表达式>

语法分析：赋值语句左侧可以是标识符、函数标识符或标识符[表达式],赋值符号是‘:=’,右侧是一个表达式结束。

句子实例：x:=y+1

<函数标识符> ::= <标识符>

语法分析：函数标识符就是标识符。

句子实例：x

<表达式> ::= [+|-]<项>{<加法运算符><项>}

语法分析：表达式开始部分可以有或没有一个+或-号，然后必须有一个项，之后按照加法运算符和项的组合重复0或多次。

句子实例：-x+1-y

<项> ::= <因子>{<乘法运算符><因子>}

语法分析：项由一个因子开始，之后按照乘法运算符和因子的组合重复0或多次。

句子实例：x\*y\*z

<因子> ::= <标识符>|<无符号整数>|'('<表达式>')' | <函数调用语句>|<标识符>'['<表达式>']'

语法分析：因子可以为一个标识符、无符号整数、(表达式)、函数调用语句或标识符[表达式]。

句子实例：a[1]

<函数调用语句> ::= <标识符>[<实在参数表>]

语法分析：函数调用语句由一个标识符开始，可以有一个或没有实在参数表。

句子实例：func(x, y)

<实在参数表> ::= '('<实在参数> {, <实在参数>}')'

语法分析：实在参数表由小括号括住的多个实在参数组成，实参由逗号隔开，最后一个实参末尾没有逗号。

句子实例：(x,y,z)

<实在参数> ::= <表达式>

语法分析：实参是一个表达式。

句子实例：-x

<加法运算符> ::= +|-

语法分析：加法运算符是一个+或-。

句子实例：+

<乘法运算符> ::= \*|/

语法分析：乘法运算符是一个\*或/。

句子实例：\*

<条件> ::= <表达式><关系运算符><表达式>

语法分析：条件是由一个表达式、一个关系运算符、一个表达式按照顺序组成的。

句子实例：x<y

<关系运算符> ::= <|<=|>|>= |=|<>

语法分析：关系运算符可以是大于号、小于号、小于等于号、等于号、大于等于号或不等号中的一个。

句子实例：=

<条件语句> ::= if<条件>then<语句> | if<条件>then<语句>else<语句>

语法分析：条件语句由if开始，加一个条件，再用then隔开，最后加一个语句，或者在then语句之后还可以加一个else和语句。

句子实例：if a=b then

a=1

<情况语句> ::= case <表达式> of <情况表元素>{; <情况表元素>}end

语法分析：情况语句由case开始，加一个表达式，再以of隔开，加一个或多个情况表元素，并用分号隔开，最后一个情况表后面没有分号，最后以end结束。

句子实例：case i of

1: i=i+1 ;

2: i=i+2

end

<情况表元素> ::= <常量> : <语句>

语法分析：情况表元素由一个常量、一个冒号和一个语句按顺序组成。

句子实例：1 : x=1

<for循环语句> ::= for <标识符> := <表达式> （downto | to） <表达式> do <语句> //步长为1

语法分析：for循环语句由for作为开始标志，跟一个标识符，在用一个:=作为赋值符，右侧跟一个表达式，再用一个downto或to分割，再加一个表达式，再用do引出最后的语句。

句子实例：for x:=1 to 10 do

x:=x\*x

<过程调用语句> ::= <标识符>[<实在参数表>]

语法分析：过程调用语句由一个标识符开始，后面可以有一个或没有实在参数表。

句子实例：proc(x,y)

<复合语句> ::= begin<语句>{; <语句>}end

语法分析：复合语句以begin开始，后面有一个或多个语句，并用分号分割，最后一个语句后面没有分号，最后用end结束。

句子实例：begin

x:=1;

y:=2

end

<读语句> ::= read'('<标识符>{,<标识符>}')'

语法分析：读语句由read开始，后面是用小括号括住的一个或多个标识符，标识符用逗号隔开，最后一个标识符后面没有逗号。

句子实例：read(a, b)

<写语句> ::= write '(' <字符串>,<表达式> ')'|write'(' <字符串> ')'|write'('<表达式>')'

语法分析：写语句以write开始，后面用小括号括住的一个字符串或一个表达式或一个字符串,表达式的组合。

句子实例：write(“abc”)

<字母> ::= a|b|c|d…x|y|z |A|B…|Z

语法分析：字母可以是26个小写或大写英文字母中的一个。

句子实例：a

<数字> ::= 0|1|2|3…8|9

语法分析：数字可以是0到9的阿拉伯数字中的一个

句子实例：8

**说明：**

（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算

（2）标识符不区分大小写字母

（3）赋值语句中<函数标识符> := <表达式> 作为函数的返回值，其类型应与返回类型一致，此语句后面的语句可继续执行

（4）写语句中的字符串原样输出，表达式只有单个字符类型的变量或常量按字符输出，其他表达式均按整型输出

（5）情况语句中，case后面的表达式和情况表元素里面的常量只允许出现integer和char类型

（6）数组的下标从0开始

（7）带var的形式参数为变量形参，实参与该类形参传递数据时是传地址

本文法与获取文法完全一致，没有进行任何的修改与扩充。

### 2．目标代码说明

目标代码选取32位mips指令集。

### 3. 优化方案\*

基本块内部的公共子表达式删除（DAG图）；

全局寄存器分配（引用计数或着色算法）；

数据流分析（通过活跃变量分析，或利用定义-使用链建网等方法建立冲突图）；

## 二．详细设计

### 1．程序结构

源程序

语法分析

符号表

出错管理

中间代码生成

四元式

优化

编译程序

目标代码

### 2．类/方法/函数功能

error(int eno):

错误处理函数，处理相关的词法语法错误，由其他函数在判断之后调用，并输出相应的错误代号和错误出现的位置。没有特殊的算法。

**词法分析部分：**

void getch():

读取字符函数，用于从源代码中读取一个字符，主要在词法分析中调用，没有特殊算法。

void nextsym():

词法分析函数，主要用于对源程序进行分词分析，为语法分析提供相应的token，并进行相应的单词正确性判断。

**语法分析部分：**

void entertable(string name, enum objecttype k):

查找符号表，判断新定义的标识符是否在该分程序段已经有定义，如果有则报错，没有则把新定义的标识符登陆进入符号表。

int locate(string name):

在符号表里查找标识符，返回一个标识符在符号表里的位置。

void entervar():

将变量登陆到符号表的函数。

void constant():

取常量值的函数。

int parameterlist():

循环处理过程、函数声明时的形参表。

void constdeclare():

常数声明处理函数。

void variabledeclare():

变量声明处理函数。

void procdeclare():

过程及函数声明处理函数。

string arrayselector():

数组元素的地址计算选取函数。

string factor():

因子分析处理函数。

string term():

项分析处理函数。处理乘除运算的表达式。

string simpleexp():

简单表达式处理函数。及处理加减运算式的表达式。

string conditionexp():

条件判断表达式处理函数。

void call(int btabrefer, string procname):

过程及函数调用函数。

void statement():

语句处理函数。

void assignment(string first):

赋值语句处理函数。

void compoundstatement():

复合语句处理函数。

void ifstatement():

if语句处理函数。

void casestatement():

case语句处理函数。

void forstatement():

for循环语句处理函数。

void writeproc():

写语句处理函数。

void readproc():

读语句处理函数。

void block():

分程序处理函数。

**四元式生成部分：**

string gen\_code(enum four\_code\_option op, string t1, string t2, string t3):

根据传入的参数生成四元式的函数。

void print\_four\_code():

打印四元式内容到文件的函数。

**目标代码生成部分：**

int locate(string name):

查找符号表函数。

void entermips(string mipsop, string mipsrs, string mipsrt, string mipsrd):

将mips指令填入指令表的函数。

void gen\_mips():

由四元式翻译生成目标代码的函数。

void print\_mips():

打印目标代码到文件。

**主程序部分：**

setup():

初始化函数，主要对程序开始前需要赋值的数组进行统一的赋值操作。

main():

主函数。

### 3．调用依赖关系

函数调用关系：

`

main

block

variabledeclare

 arraytype

 parameterlist

constdeclare

 statement

 assignment

 standproc

compoundstatement

 ifstatement

 casestatement

 forstatement

 simpleexp

conditionexp

 term

 constant

procdeclare

 arrayselector

 factor

call

setup

### 4．符号表管理方案

符号表使用一个结构数组存储，主要用于存储变量、常量、过程、函数的标识符的相关信息。符号表采用栈式存储结构，按照标识符的声明顺序填充符号表，查找则通过栈顶开始逆序查找。

结构名为table，结构中的成员变量分别为：

name(string),记录标识符名字，有长度限定。

link(int),记录同一个分程序中上一个标识符在符号表中的位置，每个分程序第一个标识符link值为0.

obj(objecttype),记录标识符种类，分为常量、变量、过程和函数。

typ(valuetype),记录标识符类型，分为整形、字符型和数组型。过程没有类型，值为notype。

refer(int),当标识符为数组时，指向数组信息表中的位置，当为过程名或函数名时，指向分程序表中的位置。

level(int),记录该标识符所在分程序的静态层次。主程序层次为1，嵌套程序逐层加一。

ifvar(int),记录标识符是否为变量形参，变量形参为1，其余为0。

addr(int).对于变量（形参）名，填入其在运行栈中的存储相对地址。对于过程名或函数名，应填入其相应的目标代码的入口地址。对于整数或字符常量，应填入其相应的整数值或ascii代码值。

### 5．存储分配方案

1. 代码区：

用于存储运行的代码序列的内存区域。

1. 参数传递区：

用于存储函数调用时传递的参数值或地址的信息，作为跳转到被调用函数后取参数信息的数据区。

1. 运行栈：

用于记录程序在运行时的动态数据。

运行栈在程序每一次进入一个程序模块时都会在栈顶创建一个该模块的活动记录，活动记录包括局部数据区、参数区和display区。

局部数据区：存放当前运行的模块定义的变量、临时变量。

参数区：包含一个返回地址、一个指向前一个活动记录基的指针和一个返回值。

display区：display区包含一系列的指针，分别指向一个活动记录的开始位置，而这些活动记录所属的模块对于当前模块来说是全局的。

### 6. 四元式设计\*

1. add t1, t2, t3

t的可能取值有普通的变量名、以“#”开头的临时变量、以“%”开头的数组地址索引临时变量、数字常量。

将rs和t2的值相加赋值给t3。

1. sub t1, t2, t3

将t1和t2的值相减赋值给t3。

1. mul t1, t2, t3

将t1和t2的值相乘赋值给t3。

1. div t1, t2, t3

将t1和t2的值相乘赋值给t3。

1. beq t1, t2, label

跳转指令，当t1等于t2时，跳转到label。

1. bne t1, t2, label

跳转指令，当t1不等于t2时，跳转到label。

1. bgez t1, t2, label

跳转指令，当t1大于等于t2时，跳转到label。

1. bgtz t1, t2, label

跳转指令，当t1大于t2时，跳转到label。

1. blez t1, t2, label

跳转指令，当t1小于等于t2时，跳转到label。

1. bltz t1, t2, label

跳转指令，当t1小于t2时，跳转到label。

1. j 空 空 label

无条件跳转至label。

1. jal 空 空 label

无条件跳转到label，并把当前代码地址加4存入栈顶，主要用于函数调用。

1. jr 空 空 空

无条件跳转到栈顶指向的地址，主要用于函数返回。

14. dec f/p 空 name

声明函数和过程。

1. par v/n 空 name

声明参数，分为变量形参合普通形参，并会从传参栈中逐一取出相应的实参值。

16. call 空 空 name

调用函数和过程，转入调用函数入口。

17. label 空 空 name

标记，if表示if语句判错之后跳转地址；else表示if语句判对之后执行结束跳转的地址；forloop表示for循环的循环入口地址；forend表示for循环结束后的地址；case表示case语句每个分支判断错误后跳到的下一个分支的地址；caseend表示case语句分支命中执行之后跳转的结尾地址；begin表示跳转至函数本体的标志。

18.push 空 空 name

将实际参数压入传参栈中。

19. array t1 t2 t3

将数组t1取第t2个元素存入t3中。

20. vart 0/1 空 number

声明临时变量，0状态表示初始化，1状态表示输入临时变量的个数number。

### 7. 目标代码生成方案\*

根据mips汇编指令集将四元式的代码逐一进行对照翻译，另外要加入对寄存器数据的存储操作和一些对运行栈进行操作的代码段，以便将生成的汇编代码正常运行。

### 8. 优化方案\*

基本块内部的公共子表达式删除（DAG图）；

数据结构：DAG图(有向无环图)，结点表(记录变量名和常量值以及它们当前对应的DAG图中的结点序号)，中间结点序列队列。

算法：

构建DAG图算法：

1. 建立结点表。
2. 对于如z = x op y的中间代码，其中z为记录计算结果的变量名，x为左操作数，行为有操作数，op为操作符，首先在结点表中寻找x，如果找到，记录下x结点号i，如果没有找到，在DAG图中新建一个叶节点i，再将x与新建叶节点号记录入结点表，y与x同理，结点号j。
3. 在DAG图中寻找中间结点，标记为op，左右操作数分别为分别为i和j，如果没有找到，则新建一个中间节点，并将i和j分别作为其左右子节点。
4. 在结点表中寻找z，如果找到则更新其结点号，如果没有找到，则在结点表中为z新建一项。
5. 对中间代码序列一次重复上述2~5。

启发式算法从DAG图中到处中间代码：

1. 初始化一个放置DAG图中间结点的队列。
2. 如果DAG图中还有中间结点没有进入队列，则执行步骤3，否则执行步骤5.
3. 选取一个尚未进入队列，但其所有父节点都进入队列的中间结点n，将其加入队列，或选取没有父节点的中间节点加入队列。
4. 如果n的最左子节点符合步骤3条件，将其加入队列，并沿着当前节点的最左边循环访问其最左子节点，将符合步骤3的中间节点加入队列，如果出现不符合条件的节点，则执行步骤2.
5. 将中间结点序列逆序输出，便得到中间结点的计算顺序，再整理得到中间代码序列。

全局寄存器分配（引用着色算法）；

图着色算法

算法：

1. 在冲突图中找到第一个连接边数数目小于全局寄存器数量K的节点，将它从图G中移走，并记录下来；

2. 重复1，记录节点的移走次序，直到无法再从冲突图中移走任何节点；

3. 在图中选取一个适当的节点，将它标志为“不分配全局寄存器”的节点，并从图中移走；

4. 重复1~3，直到图中只剩1个节点；

5. 给剩余的最后一个节点任意选取一种颜色（寄存器），然后按照节点被移走的顺序，逆序将节点和边添进去，并依次给新加入的节点选取颜色。着色过程中，保证连接边的节点须有不同颜色。

临时寄存器分配

算法：

1. 进入基本块时清空临时寄存器；

2. 为当前中间代码生成目标代码时，无论临时变量还是局部变量，需使用临时寄存器，都可以向临时寄存器池申请；

3. 临时寄存器池接到申请后，若有空闲寄存器，则标识为该申请变量占用，并返回该空闲寄存器；若没有，则将不会被使用的寄存器写回相应的内存空间，标志该寄存器被新的变量占用，返回该空闲寄存器。

4. 在基本块结尾或者函数调用发生前，将寄存器池中的所有临时寄存器写回内存空间，清空。

### 9. 出错处理

|  |  |
| --- | --- |
| 错误代号 | 错误信息 |
| 0 | 程序不完整 |
| 1 | 标识符过长 |
| 2 | 整数过长 |
| 3 | 缺少右单引号 |
| 4 | 非法字符常量 |
| 5 | 多余的单引号 |
| 6 | 缺少右双引号 |
| 7 | 一个非法的字符 |
| 8 | 符号表溢出 |
| 9 | 标识符重定义 |
| 10 | 未定义的标识符 |
| 11 | 非法的常量值 |
| 12 | 不是一个合法的标识符名 |
| 13 | 应该有冒号，定义参数类型 |
| 14 | 非法参数类型 |
| 15 | 分隔符应该为分号或右括号结束 |
| 16 | 缺少右括号 |
| 17 | 常量赋值符应该为等号 |
| 18 | 应该有等号为常量赋值 |
| 19 | 每个常量之间应该有逗号分隔 |
| 20 | 常量声明结尾应该有分号 |
| 21 | 逗号后面必须有一个标识符 |
| 22 | 变量声明后面必须有冒号，声明类型 |
| 23 | 缺少左中括号 |
| 24 | 缺少右中括号 |
| 25 | 缺少of |
| 26 | 非法的数组类型 |
| 27 | 数组表溢出 |
| 28 | 非法的变量类型 |
| 29 | 应该有分号 |
| 30 | 没有函数或过程名 |
| 31 | 函数要有返回值类型 |
| 32 | 非法函数返回值类型 |
| 33 | 过程和函数声明结尾应该有分号 |
| 34 | 嵌套层次过多 |
| 35 | 分程序表溢出 |
| 36 | 每个分程序结束应该有分号 |
| 37 | 过程不能参与运算 |
| 38 | 非法的因子 |
| 39 | 缺少赋值符号 |
| 40 | 复合语句中每条语句后面应该有分号 |
| 41 | 缺少end |
| 42 | 缺少of |
| 43 | 情况语句每个常量后面应该有冒号 |
| 44 | 情况语句每个分支结尾应该有分号 |
| 45 | 应该有to或downto |
| 46 | 应该有do |
| 47 | 缺少左括号 |
| 48 | 应该是一个标识符 |
| 49 | 不合法的语句起始符 |
| 50 | 分程序必须有一个复合语句部分 |
| 51 | 常量不能被赋值 |
| 52 | For循环变量不合法 |
| 53 | 数组定义必须大于0 |
| 54 | 数组定义必须用正整数 |
| 55 | Read中实参必须是变量 |
| 56 | 参数个数不对 |
| 57 | 参数表中至少有一个参数 |
| 58 | 程序结尾缺少句号 |
|  |  |

## 三．操作说明

### 1．运行环境

1. 首先安装codeblocks13.12。
2. 为codeblocks13.12配置GNU GCC编译器。

### 2．操作步骤

1. 将编译需要的工程文件导入codeblocks。
2. 执行p\_compiler\_lang工程，进行编译的语法分析到生成四元式。

## 四．测试报告

### 1．测试程序及测试结果

**正确程序：**

**Test：**

const listlength = 19;

var x, y, p, searchr:integer; c1, c2 :char; list:array[20]of integer;

procedure sort;

var i,j,tmp:integer;

begin

for i := 0 to listlength-1 do

begin

for j := i+1 to listlength do

begin

if list[i] > list[j] then

begin

tmp := list[i];

list[i] := list[j];

list[j] :=tmp

end

end

end

end;

function halfsearch (i,j,no:integer; var place:integer):integer;

var half:integer;

begin

if i >= j then

begin

if list[i] = no then

begin

place := i;

halfsearch := place

end

else

begin

place := -1;

halfsearch := -1

end

end

else

begin

half := (i+j)/2;

if list[half] = no then

begin

place := half;

halfsearch := place

end

else

begin

if list[half] > no then

begin

j := half - 1;

searchr := halfsearch(i,j,no,place)

end

else

searchr := halfsearch(half+1,j,no,place)

end

end

end;

procedure testcase;

var rc:char; ri:integer;

begin

read(rc);

ri := 10;

case rc of

'1': begin

write(rc);

ri := 1

end;

'2': begin

write("Itis2");

ri := 2

end;

'3': begin

write("It is 3");

ri := 3

end

end;

y := ri\*2

end;

begin

y := 0;

testcase;

for x := 19 downto 0 do

list[x] := x\*2

;

sort;

searchr := halfsearch(1,20,y,p);

for x := 0 to 19 do

write(" ", list[x]);

write(" ");

if p>=0 then

write(" ", list[p]);

write(" ");

write(y);

x := 0

end.

**结果：**

输入：1 输出：1 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 2 2

输入：2 输出：Itis2 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 4 4

输入：3 输出：It is 3 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 6 6

**TestCase：**

const ca = 'a',cb = 'b',cc = 'c';

var vc:char; vi:integer;

procedure testcase1(c:char);

begin

case c of

'a' : write("a ");

'b' : write("b ");

'c' : write("c ");

end

end;

procedure testcase2(i:integer);

begin

case i of

1 : write("1 ");

2 : write("2 ");

3 : write("3 ");

end

end;

begin

vc := ca;

testcase1(vc);

vc := cb;

testcase1(vc);

vc := cc;

testcase1(vc);

vi := 1;

testcase2(vi);

vi := 2;

testcase2(vi);

vi := 3;

testcase2(vi);

end.

**结果：**

a b c 1 2 3

**TestFor：**

const first = 1, last = 10;

var i:integer;

procedure fortest;

var testresult:integer;

begin

for i:= first to last do

begin

testresult := i;

write(testresult);

end;

for i:= 10 downto 1 do

begin

testresult := i;

write(testresult);

end;

end;

begin

fortest;

end.

**结果：**

1234567891010987654321

**TestIf：**

const ca = 'a',cb = 'b',cc = 'c';

var a,b,c:char;

function testif(testchar:char):integer;

begin

if testchar = ca then

testif := 1

else if testchar = cb then

testif := 2;

if testchar = cc then

testif := 3

end;

begin

a := ca;

b := cb;

c := cc;

if testif(a) = 1 then

begin

write("good ", a);

write(" ");

end

else

write("false ");

if testif(b) = 2 then

begin

write("good ", b);

write(" ");

end

else

write("false ");

if testif(c) = 3 then

begin

write("good ", c);

write(" ");

end

else

write("false ");

end.

**结果：**

good a good b good c

**TestProcedureLevel：**

var p:integer;

procedure test1(p:integer);

procedure test2(var p:integer);

procedure test3(var p:integer);

procedure test4(var p:integer);

procedure test5(var p:integer);

procedure test6(var p:integer);

procedure test7(var p:integer);

procedure test8(var p:integer);

procedure test9(var p:integer);

begin

write(p);

p := 9;

end;

begin

test9(p);

write(p);

p := 8;

end;

begin

test8(p);

write(p);

p := 7;

end;

begin

test7(p);

write(p);

p := 6;

end;

begin

test6(p);

write(p);

p := 5;

end;

begin

test5(p);

write(p);

p := 4;

end;

begin

test4(p);

write(p);

p := 3;

end;

begin

test3(p);

write(p);

p := 2;

end;

begin

test2(p);

write(p);

p := 1;

end;

begin

p := 10;

test1(p);

write(p);

end.

**结果：**

109876543210

**错误程序：**

**\_TestError1：**

const a = 1;

begin

a = 2;

end.

**错误信息：**

常量不能被赋值。

**\_TestError2：**

const a= 1;

var b:integer;

.

**错误信息：**

每个分程序块结束必须有一个复合语句。

**\_TestError3：**

procedure test;

var a:integer;

begin

a := 1;

end;

begin

a := 1;

end.

**错误信息：**

标识符未定义。

**\_TestError4：**

var i:integer;

begin

for i := 1 do

i = i+1;

end.

**错误信息：**

for循环缺少to或downto。

**\_TestError5：**

var i:integer;

begin

if i = 1

i := 2;

end.

**错误信息：**

if语句缺少then。

### 2．测试结果分析

综上所有的正确程序已经覆盖了文法要求的所有语法成分。

Test基本覆盖了所有的语法成分。

TestCase主要覆盖了声明语句、过程调用和case等语法成分。

TestFor 主要覆盖了声明语句、过程调用和for循环语句等语法成分。

TestIf 主要覆盖了声明语句、函数调用和if语句相关的语法成分。

TestProcedureLevel主要覆盖了过程嵌套相关的语法成分。

错误程序分别覆盖了一个相关的语法或语义错误点：

\_TestError1检测了常量不能被赋值。

\_TestError2检测了每个分程序块结束必须有一个复合语句。

\_TestError3检测了标识符未定义。

\_TestError4检测了for循环缺少to或downto。

\_TestError5检测了if语句缺少then。

## 五．总结感想

在这次课程设计中我通过自己设计实现一个简单的编译程序了解了编译的主要步骤和工作，加深了对编译的理解，也通过编译程序的设计实现了一个千行代码的程序设计，间接的提高了我的代码编写能力，也从编写过程中学到了一些设计大型程序方法和编程的技巧。

对这次课程设计的感想主要还是感觉到工作量很大，在处理程序的bug与调试过程中花费了大量的时间，每周一次的检查感觉在工作量分配上还是有不妥。语法分析和目标代码生成相对于之前的词法分析和源代码阅读的工作量差距十分的明显，这也使得我对这次程序设计的时间安排上产生问题。在语法分析程序的设计中错误估计了时间，导致程序设计产生了许多影响后续工作的遗留问题。希望之后的课程设计在分配每一步工作时也同样合理的分配时间，给予一定的时间安排上的提醒，这样也许有助于同学合理的完成课程设计。