《编译技术》课程设计文 档

学号：\_\_\_\_\_\_10231023\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_吴俣\_\_\_\_\_\_\_\_\_\_\_\_\_\_

年 月 日

目录

[一．需求说明 2](#_Toc345090323)

[1．文法说明 2](#_Toc345090324)

[2．目标代码说明 4](#_Toc345090325)

[ 类 P-code 语言 4](#_Toc345090326)

[二．详细设计 5](#_Toc345090327)

[1．程序结构 5](#_Toc345090328)

[2．类/方法/函数功能 6](#_Toc345090329)

[3．调用依赖关系 6](#_Toc345090330)

[4．符号表结构 9](#_Toc345090331)

[5．运行栈结构\* 9](#_Toc345090332)

[6．错误信息描述 9](#_Toc345090333)

[三．操作说明 11](#_Toc345090334)

[1．运行环境 11](#_Toc345090335)

[2．操作步骤 11](#_Toc345090336)

[四．测试报告 13](#_Toc345090337)

[1．测试程序及测试结果 13](#_Toc345090338)

[Test 1: 13](#_Toc345090339)

[Test 2 16](#_Toc345090340)

[Test 3 18](#_Toc345090341)

[Test 4: 20](#_Toc345090342)

[Test 5: 22](#_Toc345090343)

[2．测试结果分析 23](#_Toc345090344)

[五．总结感想 24](#_Toc345090345)

## 一．需求说明

### 1．文法说明

＜加法运算符＞ ::= +｜-  
＜乘法运算符＞  ::= \*｜/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  
＜数字＞   ::= ０｜＜非零数字＞  
＜非零数字＞  ::= １｜．．．｜９  
＜字符＞    ::=   '＜加法运算符＞｜＜乘法运算符＞｜＜字母＞｜＜数字＞'  
＜字符串＞   ::=  "｛十进制编码为32-126的ASCII字符｝"                           
＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞  
＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}  
                             | float＜标识符＞＝＜实数＞{,＜标识符＞＝＜实数＞}  
                             | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}  
＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝  
＜整数＞        ::= ［＋｜－］＜无符号整数＞｜０  
＜小数部分＞    ::= ＜数字＞｛＜数字＞｝｜＜空＞  
＜实数＞        ::= ［＋｜－］＜整数＞[.＜小数部分＞]  
＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  
＜声明头部＞   ::=  int＜标识符＞ |float ＜标识符＞|char＜标识符＞  
＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  
＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ }  
＜可枚举常量＞  ::=  ＜整数＞| ＜字符＞  
＜类型标识符＞      ::=  int | float | char  
＜有返回值函数定义＞  ::=  ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’  
＜无返回值函数定义＞  ::= void＜标识符＞‘(’＜参数＞‘)’‘{’＜复合语句＞‘}’  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
＜参数＞    ::= ＜参数表＞  
＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}|＜空＞  
＜主函数＞    ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’  
＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
＜因子＞    ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’｜＜整数＞|＜实数＞|＜字符＞｜＜有返回值函数调用语句＞|‘(’＜表达式＞‘)’           
＜语句＞    ::= ＜条件语句＞｜＜循环语句＞| ‘{’＜语句列＞‘}’｜＜有返回值函数调用语句＞;   
                      | ＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞|＜情况语句＞｜＜返回语句＞;  
＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞  
＜条件语句＞  ::=  if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］  
＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真  
＜循环语句＞   ::=  while ‘(’＜条件＞‘)’＜语句＞  
＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞［＜缺省＞］‘}’  
＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}  
＜情况子语句＞  ::=  case＜可枚举常量＞：＜语句＞  
＜缺省＞   ::=  default : ＜语句＞  
＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’  
＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’  
＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞  
＜语句列＞   ::= ＜语句＞｛＜语句＞｝  
＜读语句＞    ::=  scanf ‘(’＜标识符＞{,＜标识符＞}‘)’  
＜写语句＞    ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’|printf ‘(’＜字符串＞ ‘)’|printf ‘(’＜表达式＞‘)’  
＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

### 2．目标代码说明

### 类 P-code 语言

一种栈式机的语言此类栈式机没有累加器和通用寄存器，有一个栈式存储器，有四个控制寄存器（指令寄存器 I，指令地址寄存器 P，栈顶寄存器 T和基址寄存器 B），算逻运算都在栈顶进行

* **指令格式**
* **f ： 操作码**
* **l ： 层次差 （标识符引用层减去定义层）**
* **a ： 不同的指令含义不同**
* **指令** “INT 0 A” ：在栈顶开辟A个储存单元
* **指令** “OPR 0 0” ：将栈顶第一个元素取负
* **指令** “CAL L A” ：调用层差为L的地址为A的函数
* **指令** “LIT 0 A” ：将立即数存入栈顶
* **指令** “LOD L A”：将地址为A层差为L的放到栈顶
* **指令** “STO L A”：将栈顶存到地址为A层差为L的寄存器
* **指令** “OPR 0 2”：栈顶和次栈顶相加，存到次栈顶
* **指令** “OPR 0 3” ：栈顶和次栈顶相减，存到次栈顶
* **指令** “OPR 0 4” ：栈顶和次栈顶相乘，存到次栈顶
* **指令** “OPR 0 5” ：栈顶和次栈顶相除，存到次栈顶
* **指令** “OPR 0 8”：比较栈顶与次栈顶是否相等，相等次栈顶置1
* **指令** “OPR 0 9” ：比较栈顶与次栈顶是否不相等，不相等次栈顶置1

* **指令** “OPR 0 10” ：比较栈顶是否小于，满足次栈顶置1
* **指令** “OPR 0 11” ：比较栈顶是否大于，满足次栈顶置1
* **指令** “OPR 0 12” ：比较栈顶是否大于等于，满足次栈顶置1
* **指令** “OPR 0 13” ：比较栈顶是否小于等于，满足次栈顶置1
* **指令** “JMP 0 A” ：无条件跳转到A地址
* **指令** “JPC 0 A” ： 若栈顶为0，则跳转到A
* **指令** “rei 0 0”：读取一整数
* **指令** “ref 0 0”：读取一浮点数
* **指令** “rec 0 0”：读取一字符
* **指令** “pri 0 0”：打印一整数
* **指令** “prf 0 0”：打印一浮点数
* **指令** “prc 0 0”：打印一字符
* **指令** “cas 0 0”：比较栈顶与次栈顶 若满足存在栈顶

注记：若为数组元素，则STO与LOD指令的第三个指令为相对偏移量取负，先从栈顶找到数组的首地址。

## 二．详细设计

### 1．程序结构

扩充C0语言编译程序采用以语法分析为核心、一遍扫描的编译方法。词法分析和代码生成作为独立的子程序供语法分析程序调用。语法分析的同时，提供了出错报告和出错恢复的功能。在源程序没有错误编译通过的情况下，调用类PCODE解释程序解释执行生成的类PCODE代码。  
 在程序中首先运行main函数，获得读入和输出的文件。之后函数调用program()这一主要的分析函数对程序进行分析。Program函数每次调用getsyms来对下一个字符串进行分析处理。而getsyms函数返回他所读到的内容以及类型。

当getsyms读到保留字时，如果为常量定义就会调用两个函数 constdeclaration ()进行测试去测试常量所赋值是否与常量类型一致。并且会调用函数find()来测试是否标识符被重复定义。在变量定义时，也会调用find()函数对其进行测试。如果均无问题，将变量或者常量标识符存入符号表，调用int enter(int lev,int type,int kind)函数

变量分析与常量分析类似。其中变量包括普通的整形，浮点型以及字符型变量。并且当读到函数时，也归为变量类型，加入符号表。之后我们会进行函数的语句列定义分析，调用int statement( )来处理每一个函数中的语句通过来判断语句列是否合法以及执行了什么样的功能。在每一句的调用中，我们可以归纳总结语句，来判断语句完成了什么样的功能。

在程序的最后调用Interprete来虚拟执行PCODE

### 2．类/方法/函数功能

int interprete(); //运行栈

int getsyms(); //词法分析程序

int enter(char\* name,int type,int kind); //把变量填入符号表

int constdeclaration(); //分析常量说明部分

int program(); //分析程序

int charcheck(); //检查标识符是否重复定义

int constdefine( int type ); //分析常量定义

int vardefine( int type); //分析变量说明部分

int returnfunction( int type) ; //有返回值的函数

int voidfunction( ); //无返回值的函数定义语句

int statement( ); //语句

int ifstatement( ); //if条件语句

int whilestatement( ); //while循环语句

int switchstatement( ); //switch语句

int scanfstatement( ); //scanf读语句

int printfstatement( ); //printf写语句

int returnstatement( ); //return语句

int assignstatement(int a); //赋值语句

int mainfunction( ); //主函数

int call(int offset); //函数调用语句

int expression(); //表达式

int gen(char \*op,int lev,int x); //产生一条表达式

int term(); //项

int factor(); //因子

int find(int l);//在符号表中查找标识符是否存在

### 3．调用依赖关系

int program() 对int constdeclaration(); int vardefine( int type); 进行调用因为每个函数也属于一种变量的定义。

int constdeclaration(); 对int find(int l); 查找标识符 以及int enter(char\* name,int type,int kind);加入符号表进行调用。

int vardefine( int type) 对int find(int l); 查找标识符 以及int enter(char\* name,int type,int kind);加入符号表进行调用。当读到函数时候调用int voidfunction( ); int returnfunction( int type) ;int mainfunction( );

int voidfunction( ); int returnfunction( int type) ;int mainfunction( ); 对如下的几部分进行调用

int constdeclaration(); 定义函数常量

int vardefine( int type) 定义函数变量

int statement( ); 函数语句处理

int statement( ); 调用如下部分

int ifstatement( ); //if条件语句

int whilestatement( ); //while循环语句

int switchstatement( ); //switch语句

int scanfstatement( ); //scanf读语句

int printfstatement( ); //printf写语句

int returnstatement( ); //return语句

int assignstatement(int a); //赋值语句

int term(); //项

int factor(); //因子

int expression(); //表达式

在每次调用时候会进行gen产生新表达式。

**词法分析过程：**

词法分析子程序名为getsym，功能是从源程序中读出一个单词符号（token），把它的信息放入全局变量 sym、id和num中，语法分析器需要单词时，直接从这三个变量中获得。getsym过程通过反复调用系统函数fgetc()从源程序过获取字符，并把它们拼成单词

**语法分析子程序分析：**  
　　语法分析子程序采用了自顶向下的递归子程序法，语法分析同时也根 据程序的语意生成相应的代码，并提供了出错处理的机制。语法分析主要由函数调用分析过程（call）、常量定义分析过程 （constdeclaration）、变量定义分析过程（vardeclaration）、语句分析过程（statement）、表达式处理过程 （expression）、项处理过程（term）、因子处理过程（factor）和条件处理过程（condition）构成。这些过程在结构上构成一个嵌套的层次结构。除此之外，还有出错报告过程（error）、代码生成过程（gen）、符号表过程 （enter）

**函数处理过程分析：**

在调用函数时，首先利用一个指针，记录好此时程序的地址在哪儿。之后，生成jmp指令跳转，跳转到符号表中对应函数所指向的地址，并且传递参数，在符号表中复制一份所传递的参数，只更改这些参数所在层数。之后开始执行函数的语句。当遇到return语句或者程序走到结尾时候，可以返回相应的值，并且调用开始存的指针，执行jmp指令跳回原来的函数。

**常量定义过程**：  
　　通过循环，反复获得标识符和对应的值，存入符号表。符号表中记录下标识符的名字和它对应的值。  
  
**变量定义过程**：  
　　与常量定义类似，通过循环，反复获得标识符，存入符号表。符号表中记录下标识符的名字、它所在的层及它在所在层中的偏移地址。

**语句处理过程：**  
　 　语句处理过程是一个嵌套子程序，通过调用表达式处理、项处理、因子处理等过程及递归调用自己来实现对语句的分析。语句处理过程可以识别的语句包括赋值语句、scanf语句、printf语句、调用语句、if语句、while语句、switch语句。分析的同时生成相应的类PCODE指令。

**赋值语句的处理：**  
　　首先获取赋值号左边的标识符，从符号表中找到它的信息，并确认这个标识符确为变量名。然后通过调用表达式处理过程算得赋值号右部的表达式的值并生成相应的指令保证这个值放在运行期的数据栈顶。最后通过前面查到的左部变量的位置信息， 生成相应的sto指令，把栈顶值存入指定的变量的空间，实现了赋值操作。

**scanf语句的处理：**  
　　确定scanf语句语法合理的前提下（否则报错），生成相应的指令：第一条是rei指令，实现从标准输入设备上读一个整数值，放在数据栈顶。第二条是sto指令，把栈顶的值存入scanf语句括号中的变量所在的单元。

**printf语句的处理：**  
　　在语法正确的前提下，生成指令：通过循环调用表达式处理过程分析printf语句括号中的每一个表达式，生成相应指令保证把表达式的值算出并放到数据栈顶并生成pri指令，输出表达式的值。

**if语句的处理：**  
　　按if语句的语法，首先调用逻辑表达式处理过程处理if语句的条件，把相应的真假值放到数据栈顶。接下去记录下代码段分配位置（即下面生成的jpc指令的位置），然后生成条件转移jpc指令（遇0或遇假转移），转移地址未知暂时填0。然后调用语句处理语句后面的语句或语句块。语句处理完后，当前代码段分配指针的位置就应该是上面的jpc指令的转移位置。通过前面记录下的jpc指令的位置，把它的跳转位置改成当前的代码段指针位置。

**while语句的处理：**  
　 　首先用cx1变量记下当前代码段分配位置，作为循环的开始位置。然后处理while语句中的条件表达式生成相应代码把结果放在数据栈顶，再用cx2变量 记下当前位置，生成条件转移指令，转移位置未知，填0。通过递归调用语句分析过程分析语句或语句列并生成相应代码。最后生成一条无条件跳转指 令jmp，跳转到cx1所指位置，并把cx2所指的条件跳转指令的跳转位置改成当前代码段分配位置。

**表达式、项、因子处理：**  
　 　根据语法可知，表达式应该是由正负号或无符号开头、由若干个项以加减号连接而成。而项是由若干个因子以乘除号连接而成，因子则可能是一个标识符 或一个数字，或是一个以括号括起来的子表达式。根据这样的结构，构造出相应的过程，递归调用就完成了表达式的处理。把项和因子独立开处理解决了加减号与乘 除号的优先级问题。

### 4．符号表结构

union MyUnion

{

int valint;

float valfloat;

char valchar;

MyStruct valfunc;

};

struct Character

{

char \*name; //符号表的变量名称

int lev; //符号表的层数（在C0文法中为函数中的局部变量和全局变量）

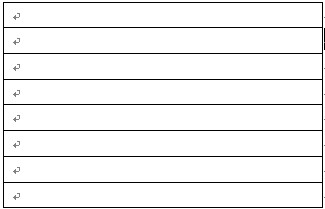
int type; //符号的类型 int float char

int kind; //是常量 变量 还是函数名称

MyUnion f; //具体的符号表符号内容

};

### 5．运行栈结构\*

 栈顶指针

局部变量

栈底指针记录上个块地址

全局变量

每次进行pcode的执行时，做好栈顶指针的移动，并且执行函数调用时，记录此时的栈底位置，将这个栈底位置存储到下一个函数的栈底地址中。每次操作在栈顶进行，lev=0时进行全局变量的赋值，否则进行局部变量的赋值

### 6．错误信息描述

case 1:

printf("没有找到该变量""%s\n",word);

break;

case 2:

printf("此行或上一行缺少分号\n");

return 0;

case 3:

printf("缺少返回值\n");

return 0;

case 4:

printf("void不能有返回值\n");

return 0;

case 5:

printf("返回值类型有误\n");

return 0;

case 6:

printf("缺少(\n");

return 0;

case 7:

printf("参数个数不符合\n");

return 0;

case 8:

printf("变量重复定义\n");

return 0;

case 9:

printf("常量无法赋值\n");

return 0;

case 10:

printf("非法字符\n");

return 0;

case 11:

printf("缺少类型标识符\n");

return 0;

case 12:

printf("缺少变量\n");

return 0;

case 13:

printf("缺少逗号\n");

return 0;

case 14:

printf("缺少单引号\n");

return 0;

case 15:

printf("缺少双引号\n");

return 0;

case 16:

printf("赋值语句赋值类型出错\n");

return 0;

case 17:

printf("数组定义错误\n");

return 0;

case 18:

printf("数组使用错误\n");

return 0;

case 19:

printf("数组缺少]");

return 0;

case 20:

printf("缺少)\n");

return 0;

case 21:

printf("缺少}\n");

return 0;

case 22:

printf("缺少:\n");

return 0;

case 24:

printf("case之后不是可枚举常量\n");

return 0;

## 三．操作说明

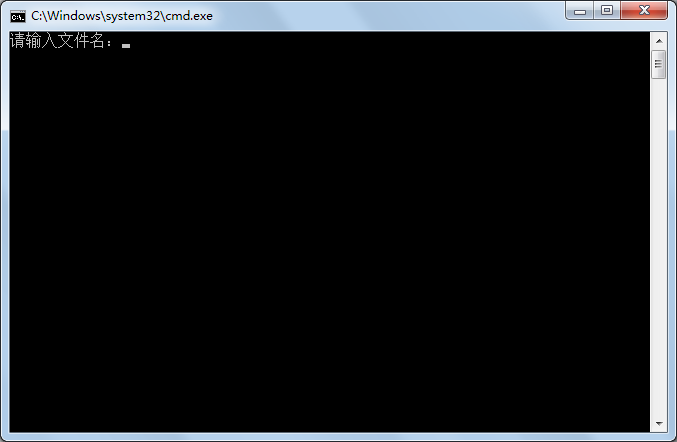
### 1．运行环境

操作系统：WindowsXP 及以上

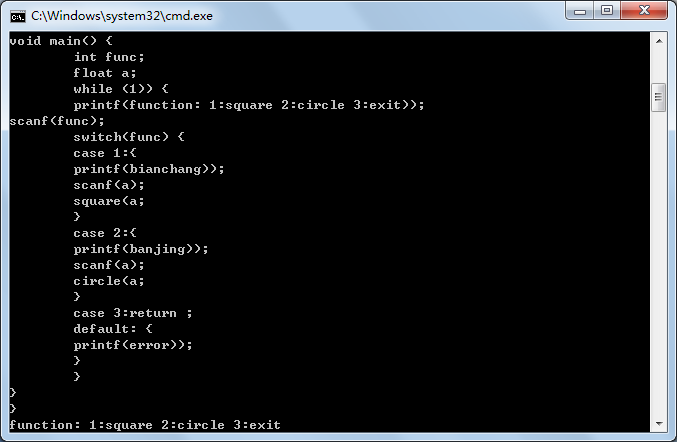
开发环境：VC6.0以上编译器

### 2．操作步骤

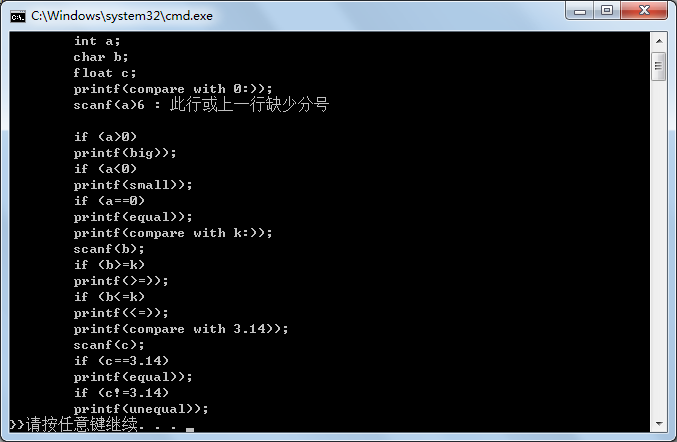
打开exe可执行文件，弹出对话框



之后输入文件名，文件名中不可含有空格，之后系统自动对代码进行分析。如果编译成功则显示运行结果，如下图



否则报错，打印语法错误



## 四．测试报告

### 1．测试程序及测试结果

### Test 1:

const int a = 1,b = -2;

const char e = 'a';

const float c = 3.14,d = -0.28;

int f,g;

float h,i5;

char j,k5;

int f1(int var1,char var2,float var3) {

int i;

i=0;

printf("output a number >3");

while (i<3) {

i=i+var1;

}

printf(i);

i=var2\*var3;

printf("output var2\*var3",i);

i=100.;

return (i);

}

void f2() {

int i;

printf("please input a number");

scanf(i);

switch(i) {

case +1:printf("1");

case +2:printf("2");

default:printf("error");

}

}

void main() {

const int c1=3;

int i1;

char i2;

float i3;

int k;

printf("please input three number");

scanf(i1,i2,i3);

k=f1(i1,i2,i3);

printf(k);

f2();

printf("const1:",c1);

printf("a:",a);

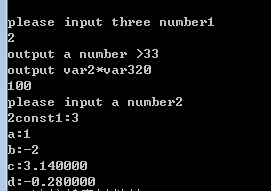
printf("b:",b);

printf("c:",c);

printf("d:",d);

printf("e:",e);

}



Error:当printf少了）后：const int a = 1,b = -2;

const char e = 'a';

const float c = 3.14,d = -0.28;

int f,g;

float h,i5;

char j,k5;

int f1(int var1,char var2,float var3) {

int i;

i=0;

printf("output a number >3");

while (i<3) {

i=i+var1;

}

printf(i;

i=var2\*var3;

printf("output var2\*var3",i);

i=100.;

return (i);

}

void f2() {

int i;

printf("please input a number");

scanf(i);

switch(i) {

case +1:printf("1");

case +2:printf("2");

default:printf("error");

}

printf("qwer");

}

void main() {

const int c1=3;

int i1;

char i2;

float i3;

int k;

printf("please input three number");

scanf(i1,i2,i3);

k=f1(i1,i2,i3);

printf(k);

f2();

printf("const1:",c1);

printf("a:",a);

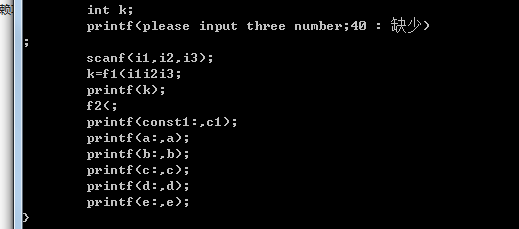
printf("b:",b);

printf("c:",c);

printf("d:",d);

printf("e:",e);

}



### Test 2

void main() {

int a;

char b;

float c;

printf("compare with 0:");

scanf(a);

if (a>0)

printf("big");

if (a<0)

printf("small");

if (a==0)

printf("equal");

printf("compare with k:");

scanf(b);

if (b>='k')

printf(">=");

if (b<='k')

printf("<=");

printf("compare with 3.14");

scanf(c);

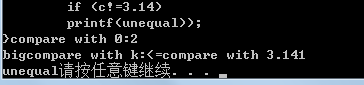
if (c==3.14)

printf("equal");

if (c!=3.14)

printf("unequal");

}



Error:去掉一个分号后：

void main() {

int a;

char b;

float c;

printf("compare with 0:");

scanf(a)

if (a>0)

printf("big");

if (a<0)

printf("small");

if (a==0)

printf("equal");

printf("compare with k:");

scanf(b);

if (b>='k')

printf(">=");

if (b<='k')

printf("<=");

printf("compare with 3.14");

scanf(c);

if (c==3.14)

printf("equal");

if (c!=3.14)

printf("unequal");

}



### Test 3

const float pi=3.14;

void square(float a) {

float result;

int func;

result=0;

printf("function C\_square S\_square");

scanf(func);

switch(func) {

case 1:result=4\*a;

case 2:result=a\*a;

}

printf("result:",result);

}

void circle(float r) {

float result;

int func;

result=0;

printf("function C\_circle S\_circle");

scanf(func);

if (func==1) result=2\*pi\*r;

if (func==2) result=pi\*(r\*r);

printf("result:",result);

}

void main() {

int func;

float a;

while (1) {

printf("function: 1:square 2:circle 3:exit");

scanf(func);

switch(func) {

case 1:{

printf("bianchang");

scanf(a);

square(a);

}

case 2:{

printf("banjing");

scanf(a);

circle(a);

}

case 3:return ;

default: {

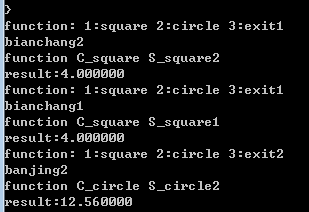
printf("error");

}

}

}

}



Error:引入未定义的元素

const float pi=3.14;

void square(float a) {

float result;

int func;

result=0;

printf("function C\_square S\_square");

scanf(func);

switch(func) {

case 1:result=4\*a;

case 2:result=a\*a;

}

printf("result:",result);

}

void circle(float r) {

float result;

int func;

result=0;

printf("function C\_circle S\_circle");

scanf(func);

if (func==1) result=2\*pi\*r;

if (func==2) result=pi\*(r\*r);

printf("result:",result);

}

void main() {

int func;

float a;

while (1) {

printf("function: 1:square 2:circle 3:exit");

scanf(func);

switch(func) {

case 1:{

printf("bianchang");

scanf(a);

square(bc);

}

case 2:{

printf("banjing");

scanf(a);

circle(a);

}

case 3:return ;

default: {

printf("error");

}

}

}

}



### Test 4:

int a[10];

void main()

{

int i,j,t;

i=0;

j=0;

while(i<10)

{

a[i]=10-i;

i=i+1;

}

i=0;

while(i<10)

{

j=i;

while(j<10)

{

if(a[j]<a[i])

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

j=j+1;

}

i=i+1;

}

j=0;

while(j<10)

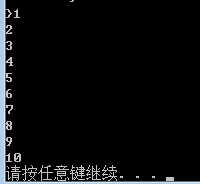
{

printf(a[j]);

j=j+1;

}

}



成功对数组元素进行了排序

Error:变量重复定义

int a[10];

void main()

{

int i,j,t;

float i;

i=0;

j=0;

while(i<10)

{

a[i]=10-i;

i=i+1;

}

i=0;

a[2]=3;

a[3]=4;

a[8]=2;

while(i<10)

{

j=i;

while(j<10)

{

if(a[j]<a[i])

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

j=j+1;

}

i=i+1;

}

j=0;

while(j<10)

{

printf(a[j]);

j=j+1;

}

}



### Test 5:

int digui(int n) {

if (n==1) {

return (1);

}

return (n\*digui(n-1));

}

void main() {

int n;

int result;

printf("please input a number");

scanf(n);

result=digui(n);

printf("n！result",result);

}



成功输出了5的阶乘

Error：当函数调用传的参数个数不一样时：

int digui(int n) {

if (n==1) {

return (1);

}

return (n\*digui(n-1));

}

void main() {

int n;

int result;

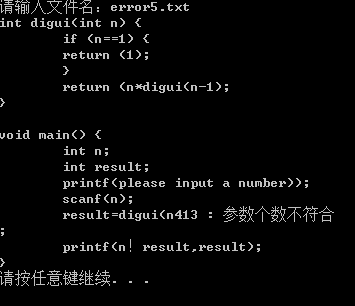
printf("please input a number");

scanf(n);

result=digui(n,4);

printf("n！result",result);

}



### 2．测试结果分析

|  |  |  |
| --- | --- | --- |
| 测试程序编号 | 正确程序覆盖点 | 错误程序覆盖点 |
| 1 | 全局三种类型常量 变量  有返回值调用，无返回值调用  输入，输出语句，switch与while | 语句缺少’)’ |
| 2 | 局部变量赋值  If语句的各种情况 | 缺少分号 |
| 3 | While与switch语句调用  无返回值有返回值函数调用 | 变量未定义 |
| 4 | 数组的使用  If与while相互嵌套 | 变量重复定义 |
| 5 | 递归函数调用 | 函数调用传参个数不对 |

## 五．总结感想

编译课程设计是我上大学以来碰到的最好的也是严格的课程设计。首先编译课程设计不像其他的课程设计以文档来决定大部分分数。而是最后高难度的编码。记得数字逻辑的平时成绩都是文档写的好不好还占特别大比重。我觉得这样就失去了实验课程的真正意义。然而编译则是真正计算机科学上的测试点评分，来评价你的编译器是否合格。

编译课设从一开始的一点都不会，到最后很有成就感的可以编译好多好多行的程序，这个进步是我一点一滴的汗水所浇筑的，记得有个星期每天晚上都是编译器程序的代码编写，一点点debug，一点点思考一点点前进。然后到最后进行测试的时候从基本的输入输出都有错误，到可以进行简单程序的编写，再到最后可以过老师的测试点。这一切的一切都让我领略到写一个系统多么难。由于前期对错误处理的认识不够，导致没有设计好架构，然后一个星期用来处理错误处理部分。最终也没有很好的处理完成。

不过一切的一切都过去了，编译实验让我的提升太多太多。我觉得中国的大学就应多一些这种课程，少一些水课。每学期几门硬课足够。而且这次让我感觉到软件工程中，随着代码量的增加 ，软件的维护性越来越弱，这也让我们有更好的编程风格