《编译技术》课程设计文 档

2015年 1月 8日

目录

[一．需求说明 3](#_Toc440102381)

[1. 文法说明 3](#_Toc440102382)

[2. 目标代码说明 6](#_Toc440102383)

[二．详细设计 7](#_Toc440102384)

[1. 程序结构 7](#_Toc440102385)

[2. 类/方法/函数功能 8](#_Toc440102386)

[3. 调用依赖关系 10](#_Toc440102387)

[4. 符号表管理方案 11](#_Toc440102388)

[5. 存储分配方案 12](#_Toc440102389)

[6. 解释执行程序\* 13](#_Toc440102390)

[7. 出错处理 13](#_Toc440102391)

[三．操作说明 15](#_Toc440102392)

[1．运行环境 15](#_Toc440102393)

[2．操作步骤 15](#_Toc440102394)

[四．测试报告 17](#_Toc440102395)

[1．测试程序及测试结果 17](#_Toc440102396)

[2． 测试结果分析 18](#_Toc440102397)

[五．总结感想 19](#_Toc440102398)

## 一．需求说明

### 1. 文法说明

我选择的是中难度文法。文法如下：

1、<程序>                      ::=  <分程序>.

范例：

const a=4; .

分析：程序的主体是分程序，“.”相当于结束符号，告诉编译器需要编译的程序到此为止；

2、<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>

范例：

const a = 10;

var b :integer;

function fun1(c:integer);begin end;

begin end.

分析：

程序结构的顺序是确定的，必须先是常量说明部分，再是变量说明部分，然后过程说明部分和函数说明部分可以出现多次，且无先后顺序，最后必须有符合语句。

3、<常量说明部分>     ::=  const<常量定义>{,<常量定义>};  
<常量定义>              ::=  <标识符>＝ <常量>  
<常量>                       ::=  [+| -] (<无符号整数>| <无符号实数>)|<字符>  
<字符>                       ::=  '<字母>' | '<数字>'

范例：

常量说明部分：const a = -1,b = ‘a’,c = +12.000;

分析：多个声明之间用逗号隔开，常量可以是实数整数和字母。

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

范例：”hello world”

分析：字符串用双引号标示，和pascal不同。

5、<无符号整数>          ::=  <数字>{<数字>}  
<无符号实数>          ::=  <无符号整数>.<无符号整数>

范例：

无符号整数：1314

无符号实数：13.14

分析：无符号实数只允许小数点形式，而且必须出现小数点。

6、<标识符>                  ::=  <字母>{<字母>|<数字>}

范例：a1

分析：标识符只能是字母开头，后面可以是字母和数字。

7、<变量说明部分>     ::=  var <变量说明> ; {<变量说明>;}  
<变量说明>              ::=  <标识符>{, <标识符>} :  <类型>  
<类型>                      ::=   <基本类型>|array'['<无符号整数>']' of <基本类型>  
<基本类型>              ::=   integer | real | char

范例：变量说明部分：var a,b,c:integer;

Ch1,ch2:char;r1,r2:real;arr1:array [2] of integer;

分析：变量说明部分var只出现了一次。变量类型可以是基本类型，也可以是数组。

8、<过程说明部分>     ::=  <过程首部><分程序>{;<过程首部><分程序>};  
<函数说明部分>     ::=  <函数首部><分程序>{;<函数首部><分程序>};  
<过程首部>             ::=   procedure<标识符>'('[<形式参数表>]')';  
<函数首部>               ::=   function <标识符>'('[<形式参数表>]')': <基本类型>;  
<形式参数表>          ::= [var] <标识符>{, <标识符>}: <基本类型>{; <形式参数表>}

范例：produce pro1();

Var i:integer;

Begin

I:=i+1

End

分析：形式参数表可以为空，但是括号必须有。

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

分析：这里是所有的语句类型，还有空语句，这个使对分号处理变得简单了。

10、<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>  
<函数标识符>          ::=  <标识符>

范例：a := b + c

Func1 := a

Arr[a] := a

分析：以上是三种类型，1是把表达式赋值给标识符，2是把表达式赋值给函数标识符3是赋值给数组

11、<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>']'|<无符号整数>| <无符号实数>|'('<表达式>')' | <函数调用语句>  
<函数调用语句>      ::=   <标识符>'('[<实在参数表>]')'  
<实在参数表>          ::=  <实在参数> {, <实在参数>}  
<实在参数>              ::=  <表达式>

范例：因子：a,a[1],123,111.11,(a+q),func1(a,b)

项：a\*b,a/11.1

表达式：a+1-p

分析：函数参数为空时也要有括号，字符不能作为因子。

12、<加法运算符>          ::=  +|-  
<乘法运算符>          ::=  \*|/

分析：加法运算符也包括减法，乘法运算符也包括除法。

13、<条件>                       ::=  <表达式><关系运算符><表达式>  
<关系运算符>          ::=  <|<=|>|>= |=|<>

范例：a>2

分析：条件必须是两个表达式的比较。

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

范例：if a > 0 then a := a + 1 else a := a – 1

分析：同样的结构可以形成else if语句。

15、<repeat循环语句>  ::=  repeat <语句>until<条件>

范例：repeat a:=a-1 until a<0

16、<for循环语句>         ::=  for <标识符> := <表达式> (to|downto) <表达式> do <语句>

范例：for a:=1 to 100 do b:=b+1

分析：前后表达式应该都是相同型的。

17、<过程调用语句>      ::=   <标识符>'('[<实在参数表>]')'

分析：参数为空时也要有括号。

18、<复合语句>              ::=  begin<语句>{; <语句>}end

分析：如果有多条语句，要用分号分开。

19、<读语句>                  ::=  read'('<标识符>{,<标识符>}')'

范例：read(a,c,v)

分析：可以读多个参数，用逗号隔开

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

范例：write(“hello”,a)

Write(“aaa”)

Write(a)

分析：可以写字符串，可以写表达式。但是只有上面三种写法。

21、<字母>                      ::=   a|b|c|d…x|y|z |A|B…|Z  
<数字>                      ::=   0|1|2|3…8|9

分析：字母包括大小写。

附加说明：

（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或实型输出

（5）数组的下标从0开始

（6）for循环语句步长为1

1. 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

### 2. 目标代码说明

目标代码是生成PCODE指令，所有指令为PASCAL-S的子集。

指令的数据结构为：

typedef struct {

int f, x, y;

}order;

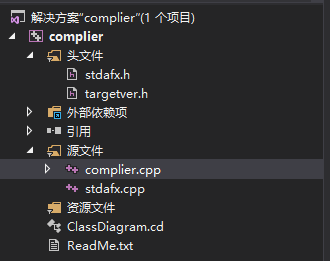
order code[cmax + 1];

P-code指令集如下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 助记符 | f | x | y | 功能 |
| LDA | 0 | x | y | 把变量地址装入栈顶（x为层次，y为变量相对地址） |
| LOD | 1 | x | y | 装入值 |
| LDI | 2 | x | y | 间接装入 |
| DIS | 3 | x | y | 更新display |
| INT | 9 |  | y | 将栈顶元素加上y |
| JMP | 10 |  | y | 无条件转移到y |
| JPC | 11 |  | y | 如栈顶内容为假，转移到y |
| F1U | 14 |  | y | 增量步长型for循环体的入口测试 |
| F2U | 15 |  | y | 增量步长型for循环体的再入口测试 |
| F1D | 16 |  | y | 减量步长型for循环体的入口测试 |
| F2D | 17 |  | y | 减量步长型for循环体的再入口测试 |
| MKS | 18 |  | y | 标记栈 |
| CAL | 19 |  | y | 调用过程和函数 |
| IDX | 20 |  | y | 去下标变量地址（元素长度为1） |
| LDB | 22 |  | y | 装入块 |
| CPB | 23 |  | y | 复制块 |
| LDC | 24 |  | y | 装入字面常量 |
| LDR | 25 |  | y | 装入实数 |
| FLT | 26 |  | y | 转为浮点数 |
| RED | 27 |  | y | 读（y表示类型，1：整形，2：实型，3：字符型） |
| WRS | 28 |  | y | 写字符 |
| WRW | 29 |  | y | 写——隐含域宽 |
| WRU | 30 |  | y | 写——给定域宽 |
| HLT | 31 |  |  | 停止 |
| EXP | 32 |  |  | 退出过程 |
| EXF | 33 |  |  | 退出函数 |
| LDT | 34 |  |  | 取栈顶单元内容为地址的单元内容 |
| MUS | 36 |  |  | 求负 |
| WRR | 37 |  |  | 写实数——给定域宽 |
| STO | 38 |  |  | 将栈顶内容存入以栈顶次高元为地址的单元 |
| EQR | 39 |  |  | 实数等于比较 |
| NER | 40 |  |  | 实数不等比较 |
| LSR | 41 |  |  | 实型小于比较 |
| LER | 42 |  |  | 实型小于等于比较 |
| GTR | 43 |  |  | 实型大于比较 |
| GER | 44 |  |  | 实型大于等于比较 |
| EQL | 45 |  |  | 整型相等比较 |
| NEQ | 46 |  |  | 整型不等比较 |
| LSS | 47 |  |  | 整型小于比较 |
| LER | 48 |  |  | 整型小于等于比较 |
| GRT | 49 |  |  | 整型大于比较 |
| GEQ | 50 |  |  | 整型大于等于比较 |
| ADD | 52 |  |  | 整型加 |
| SUB | 53 |  |  | 整型减 |
| ADR | 54 |  |  | 实型加 |
| SUR | 55 |  |  | 实型减 |
| MUL | 57 |  |  | 整型乘 |
| DIV | 58 |  |  | 整型除 |
| MUR | 60 |  |  | 实型乘 |
| DIR | 61 |  |  | 实型除 |

## 二．详细设计

### 1. 程序结构



程序中只有两个源文件，其中stdafx.cpp是vs自带的，所有我写的代码都在complier.cpp中。

### 2. 类/方法/函数功能

本程序一共由46个函数组成（不包括主函数）。下面对这些函数的主要功能做简要说明。

set<T> \_union(const set<T> &s1, const set<T> &s2)

求两个集合的并集。

void errormsg()

打印被编译源程序中出错信息。

void endskip()

源程序出错后在被跳读的部分下面印出下划线标志，只在错误当行打印。

void nextch()

读取下一个字符，处理航结束符，打印被编译源程序行。

void error(int n)

打印出错位置和出错编号。

void fatal(int n)

打印表格溢出的信息。

void insymbol()

读取下一个单词符号

void setup()

程序开始是执行，初始化一些变量。

void enter(alfa x0, objecttyp x1, types x2, int x3)

把标准类型、过程和函数的名字登录到符号表tab中

void enterarray(int h)

登录数组信息向量表atab

void enterblock()

登录分程序表btab

void enterreal(double x)

登录实常数表rconst

void emit2(int fct, int a, int b)

void emit1(int fct, int b)

void emit(int fct)

以上三个为生成PCODE指令

void printtables()

打印编译生成的符号表、分程序表、实常数表、字符串表、PCODE指令

void enterids()

在符号表中登录基本类型、函数和过程的名字及相应信息。

void skip(symset fsys, int n)

跳读源程序，直至取来的符号属于给出的符号集为止，并打印出错信息。

void test(symset s1, symset s2, int n)

测试当前符号是否合法，若不合法，打印出错信息并进行跳读。

void testsemicolon(symset fsys)

测试当前符号是否为分号。

void enter(alfa id, objecttyp k, int level)

在符号表中登录分程序说明部分出现的名字。

int loc(alfa id, int level)

查找标识符在符号表中的位置。

void entervariable(int level)

将变量名登录到符号表中。

void constant(symset fsys, conrec \* c, int level)

处理程序中出现的常量，并有参数c返回该常量的类型和数值。

void typ(symset fsys, types \* tp, int \* rf, int \* sz, int level)

处理类型描述，有参数得到它的类型（tp），指向类型详细信息表的指针（rf）和该类型的大小（sz）。

void constdec(symset fsys, int level)

处理常量定义，将常量名及相应信息填入符号表。

void variabledeclaration(symset fsys, int level, int \*dx)

处理变量说明，并将变量名及其相应信息填入符号表。

void procedeclaration(symset fsys, int level)

处理过程或函数声明，将过程（函数）名填入符号表，递归调用block分析处理程序，层次+1.

void parameterlist(symset fsys, int level, int \* dx)

处理过程或函数声明中的形参表，将形参及其有关信息登录到符号表中。

void block(symset fsys, bool isfun, int level)、

处理分程序。

void startblock(symset fsys, bool isfun, int level)

处理程序的入口，只调用一次，与block相似。

void selector(symset fsys, int level, item \* v)

处理数组下标变量。

void assignment(int lv, int ad, int i, symset fsys, int level)

处理赋值语句。

void compountstatement(symset fsys, int level)

处理复合语句。

void ifstatement(symset fsys, int level)

处理if语句。

void repeatstatement(symset fsys, int level)

处理repeat语句。

void forstatement(symset fsys, int level)

处理for语句。

void standproc(int n, symset fsys, int level)

处理read和write语句。

void call(symset fsys, int i, int level)

处理过程和函数调用。

void statement(symset fsys, int level)

处理各种语句。

types resulttype(types a, types b)

处理整型、实型、字符型变量运算时的类型转换。

void factor(symset fsys, int level, item \*x)

处理因子，由参数返回结果类型。

void term(symset fsys, int level, item \*x)

处理项，由参数返回结果类型。

void simpleexpression(symset fsys, int level, item \* x)

处理表达式，由参数返回结果类型。

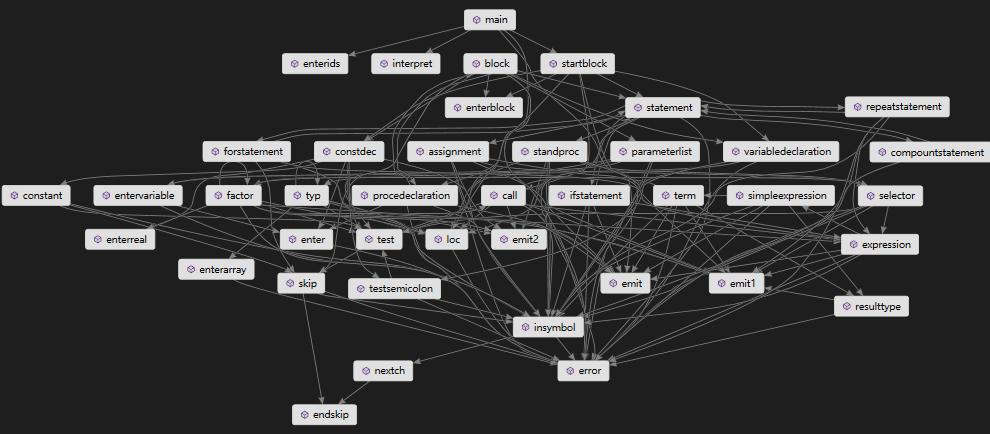
void expression(symset fsys, int level, item \* x)

处理条件，有参数返回结果类型。

void interpret()

PCODE指令解释执行。

### 3. 调用依赖关系



### 4. 符号表管理方案

（1）符号表tab

符号表索引变量：t

数据结构：

struct {

alfa name;

index link;

objecttyp obj;

types typ;

index \_ref;

bool normal;

int lev;

int adr;

}tab[tmax];

其中，alfa，index，objecttyp的定义如下：

typedef char alfa[alng + 1];

typedef int index;

typedef enum { konstant, vvariable, typel, prozedure, functionn }objecttyp;

被编译的程序中每个标识符都应在符号表中登录一个登记项，该登记项由8个域组成，它们的意义说明如下：

|  |  |
| --- | --- |
| Name | 标识符名字 |
| Link | 指向同一分程序中上一个标识符在tab表中的位置，每个分程序在tab中登录的第一个标识符的link值为0. |
| Obj | 标识符种类，可以是常量（constant）、变量（variable）、类型（typel）、过程（procedure）和函数（function） |
| Typ | 标识符类型，可以是整型（ints）、实型（reals）、布尔型（bools）、字符型（chars）、数组型（arrays） |
| Ref | 当标识符为数组类型名或数组变量名时，ref为指向该数组在数组信息向量表（atab）中登录的位置；当标识符为过程名或函数名时，ref指向它们在btab表中的指针值；其他情况时，ref一律为0. |
| Lev | 表示该标识符所在分程序的静态层次。主程序层次为1，嵌套分程序的层次数逐一递增。 |
| Normal | 如果标识符为变量形参时，此值为false，值形参或其他变量，此值为true |
| Adr | 对于变量名，填入该变量在运行栈中分配的存储单元的相对地址；对于整数、字符常量名，应分别填入其相应的整数值、ASCII码值。对于实常量名，则在adr域中填入它在实常量表中登录的位置。 |

（2）分程序表btab

数据结构：

struct {

index last, lastpar, psize, vsize;

}btab[bmax];

Btab每个域的含义为：

|  |  |
| --- | --- |
| Last | 它始终指向该分程序中最后一个标识符在tab表中的位置。 |
| Lastpar | 指向过程或函数的最后一个参数在tab表中的位置。 |
| Psize | 参数及该分程序在运行栈中的内务信息区所占的存储单元数。 |
| Vsize | 局部变量、参数及内务信息区在运行栈中所占的存储单元总数。 |

（3）分程序索引表display

int display[lmax + 1];

display的索引变量是level，即该分程序的静态层次，display表项的内容就是指向分程序表btab的指针。

（4）数组信息向量表atab

struct {

types inxtyp, eltyp;

index low, high, elsize, \_size;

}atab[amax];

|  |  |
| --- | --- |
| Intxtp | 此处恒为ints，表示数组下标类型。 |
| Eltyp | 数组元素类型，可为任意类型。 |
| Low,high | 数组的上下界值。Low恒为0. |
| Elsize,size | 分别为数组元素的大小和该数组的大小。 |

（5）实常量表rconst

索引变量c2

double rconst[c2max];

被编译程序出现的所有字面时常量都要登录到实常量表中。同样的实常量只存一次。

### 5. 存储分配方案

P-code指令运行时，每一个分程序的数据区都是按照过程或函数的调用关系在数据堆栈中进行动态存储分配的。所有的运算操作都在栈顶单元之间进行，而且计算结果也留在栈顶。

Block中利用dx作为变量存储分配的索引，dx的初值是无，即每个分程序在运行栈S中的数据区开头应预留5个单元，作为内务信息区，用来存放函数结果（如果是过程则空着不用）、返回地址（RA）、静态链（SL）、动态链（DL）以及该过程（或函数）名在tab表中登记项的位置。在内务信息区上面将分配参数和局部变量存储区。

运行栈的数据结构是：

struct {

int i;

double r;

bool b;

}s[stacksize];

解释程序开始时，把s[1],s[2]置0，s[3]置成-1，s[4]置成btab[1].last，并让栈顶指针t指向全局变量区的最后一个单元。

每当目标代码运行到要调用一个过程或函数时，首先执行一条“标记栈”指令。该指令先把栈顶指针加5，即留出5个单元的“内务信息区”，并把过程（或函数）在tab中的位置填入内务信息区的第5个单元中，把本程序在数据栈中将要占据的数据区长度减1记录到内务信息区的第4个单元中。

接下来是求实参表达式的指令，并把实参表达式的值或实参变量的地址放到预留的参数单元中。

接下来，执行过程或函数调用指令（CAL）。该指令把内务信息区的RA,SL,DL单元内容都填上。

### 6. 解释执行程序\*

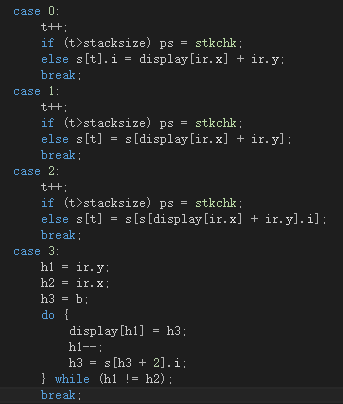
解释程序开始时，把s[1],s[2]置0，s[3]置成-1，s[4]置成btab[1].last，并让栈顶指针t指向全局变量区的最后一个单元。

每当目标代码运行到要调用一个过程或函数时，首先执行一条“标记栈”指令。该指令先把栈顶指针加5，即留出5个单元的“内务信息区”，并把过程（或函数）在tab中的位置填入内务信息区的第5个单元中，把本程序在数据栈中将要占据的数据区长度减1记录到内务信息区的第4个单元中。

接下来是求实参表达式的指令，并把实参表达式的值或实参变量的地址放到预留的参数单元中。

接下来，执行过程或函数调用指令（CAL）。该指令把内务信息区的RA,SL,DL单元内容都填上。

下面附上部分指令的解释程序，分别是f为0,1,2,3的指令。



### 7. 出错处理

本程序能够处理所有不符合文法的错误，能够处理标识符未定义或重复定义的错误。

对于数据类型一致性，本程序要求传递参数类型必须完全一致，赋值时可以从字符到整型，从整型到实型。

在编译过程中，如果各种表格有一个发生溢出，便会立即停止编译，打印出该表格溢出的信息（表格的容量设置的都很大，应该不会发生溢出）。

本程序遏制了某些变异宠重复出现的冗余错误信息，并在每行源程序下面仅打印出一行出错标志。在最后输出所有错误的详细信息。

在查出错误时，在被跳读的源程序下面打印下划线，提示用户注意（仅当被跳读的长度大于3时，不然没有空间打印下划线）。

下面是错误信息表。

|  |  |
| --- | --- |
| 序号 | 内容 |
| 0 | 该标识符未定义 |
| 1 | 该标识符重复定义 |
| 2 | 应是标识符 |
| 4 | 应是’)’ |
| 5 | 应是’:’ |
| 6 | 非法符号 |
| 7 | 形式参数表中，形参声明应该以标识符或var开头 |
| 8 | 应是of |
| 9 | 应是’(‘ |
| 10 | 类型定义必须以标识符、array开头（本程序中基本类型都是标识符） |
| 11 | 应是’[‘ |
| 12 | 应是’]’ |
| 14 | 应是’;’ |
| 15 | 函数结果应是integer、real或char型 |
| 16 | 应是’=’ |
| 17 | 条件表达式不是bool型 |
| 18 | 在for后面的循环变量只能是integer、char型 |
| 19 | For语句中初值和终值表达式必须与循环变量类型相同 |
| 21 | 数太大 |
| 23 | 形式参数表要有’(‘ |
| 24 | 非法字符 |
| 26 | 下表表达式类型必须是integer |
| 27 | 数组说明中，上界比须大于零，且类型必须为integer |
| 28 | 没有这样的数组 |
| 29 | 应是类型标识符 |
| 30 | 该类型未定义 |
| 32 | 调用时参数表应有’(‘ |
| 33 | 算数表达式类型不合法（例如array整体不能参加运算） |
| 35 | 相比较的对象类型必须相同 |
| 36 | 实参和形参类型必须相同 |
| 37 | 应是变量 |
| 39 | 实参个数与形参个数不等 |
| 40 | 小数点后没有数字 |
| 41 | Read或write参数类型不正确 |
| 42 | 该表达式应为实型 |
| 44 | 表达式中不能出现过程标识符 |
| 45 | 应是变量或过程/函数标识符 |
| 46 | 在赋值语句中被赋值变量应该与表达式类型相同 |
| 47 | 表达式中不应出现字符 |
| 49 | 表溢出 |
| 50 | 应是常量 |
| 51 | 应是’:=’ |
| 52 | 应是then |
| 53 | 应是until |
| 54 | 应是do |
| 55 | 应是to或downto |
| 56 | 应是begin |
| 57 | 应是end |
| 58 | 因子必须以标识符、常量、’(‘开头 |

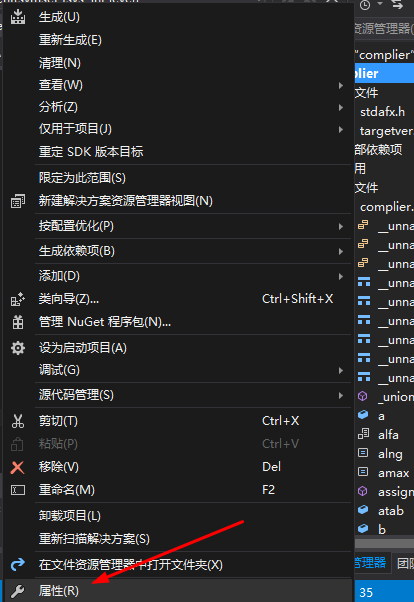
## 三．操作说明

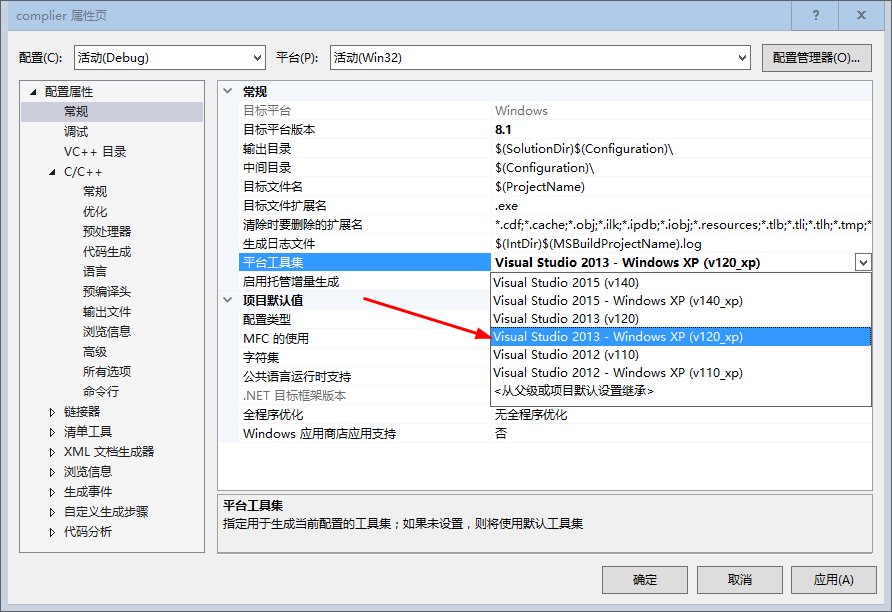
### 1．运行环境

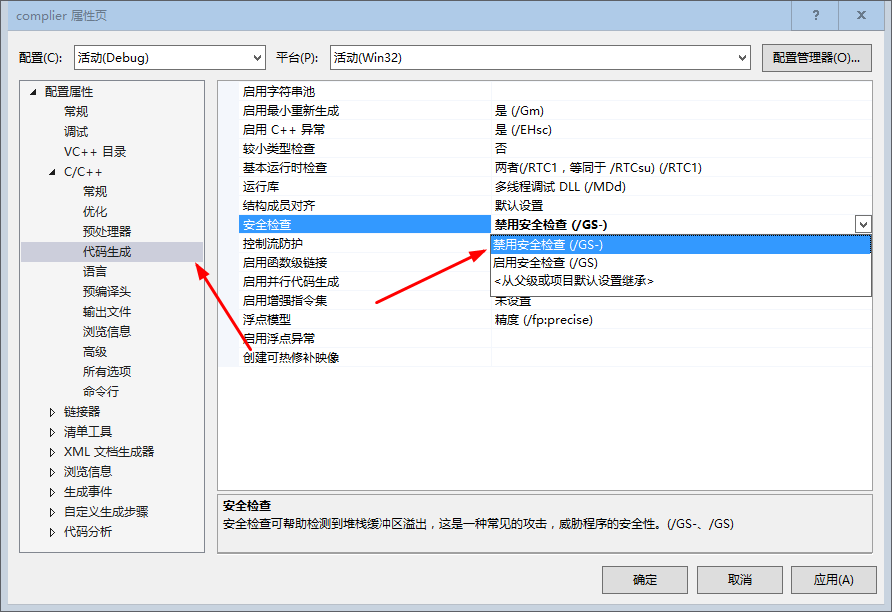
WIN7 及以上操作系统，Visual studio 2013 express。

### 2．操作步骤

打开打开visual studio，在工程名上点右键点击属性，配置属性->常规->平台工作集 选择Visual Studio 2013 WindowsXP（v120\_xp）；配置属性->c/c++->代码生成->安全检查 选择禁用安全检查。然后点击运行键即可。







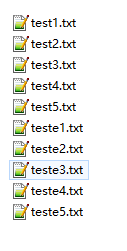
## 四．测试报告

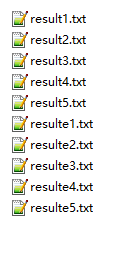
### 1．测试程序及测试结果

如果把代码粘到这里，会非常影响视觉的，所以我都放到外边的一个文件夹了。

测试程序见测试程序文件夹，共有10个测试程序，5个正确的程序命名为test[1-5].txt,5个错误的程序命名为teste[1-5].txt

测试结果见测试结果文件夹，共有10个测试结果，5个正确的结果被命名为result[1-5].txt,5个错误的程序的结果被命名为resulte[1-5].txt





### 测试结果分析

测试程序1：这个程序能够覆盖绝大多数文法，前一部分如果结果正确就输出right，如果结果错误就输出WRONG，可以看出测试结果是正确的。后一部分测试输入，分别输入实数、字母，并进行输出。

测试程序2：这个程序主要测试函数的嵌套，命名，传值，返回值。结果输出11,10，是正确的。

测试程序3：这个程序主要测试读语句与包括实数、字符、整型的四则运算。通过结算得到结果是正确的。

测试程序4：这个程序主要测试if语句和for循环。If根据不同的情况有不同的输出，for循环打印斐波那契数列。结果是正确的。

测试程序5：这个程序主要测试读语句和比较。如果是小写字母且小于m则输出自己，如果大于m则输出after m，如果是大写字母，则输出Big，其他输出invalid。结果是正确的。

错误测试程序1：这个有个传引用调用，程序传的是一个const常量，能够正确报错。

错误测试程序2：这个程序有两个错误，1是直接调用函数，2是函数内部有与函数名同名的变量。程序都能够识别并正确报错。

错误测试程序3：这个程序有外层引用里层变量的错误，测试结果能够识别出这个错误。

错误测试程序4：这个程序主要测试赋值时类型的错误。测试结果显示，不允许实型赋值给整型，不允许整型赋值给字符型，测试结果是正确的。

错误测试程序5：表达式中出现字符，测试结果能够正确报错。

## 五．总结感想

编译实验课程让我更加深刻的理解了编译器的构造，通过实验得到的知识比单单从课本上学来的知识要更加印象深刻。最初我是选的高难度，打算做一个生成x86的编译器，但是由于7班学习任务和我自己担任一门助教的任务过于繁重，我改成了中难度。我是完全仿照PASCAL-S来写的，本想写一个完全翻译版本，但是后来发现PASCAL-S也有很多不完善的地方，比如取反符号只能给整数取反，各种表的容量太小，因子中有个循环，而实际因子不会连续出现，形式参数表中没有括号等等，所以我开始了自己的改变，并最终正确实现了所有的功能。虽然难度是中，但也很大地提高了我读代码，写代码的能力。