

C0 文法编译器设计文档

12231005 马宇辰

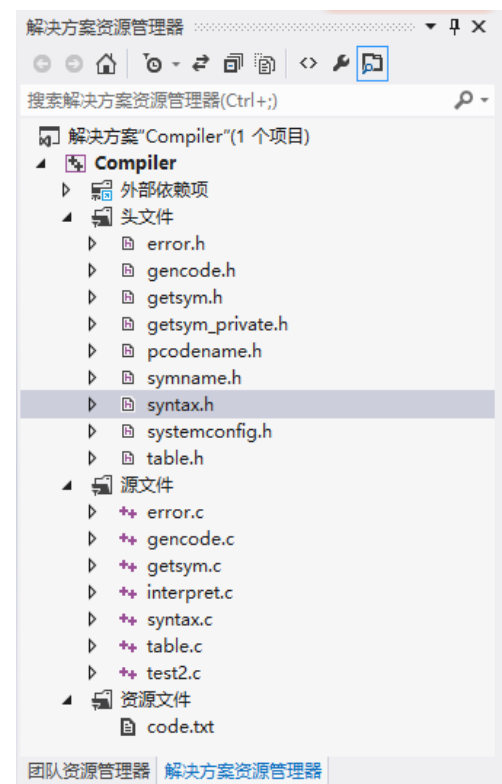
1 需求说明

本项目为针对扩充 C0 文法（数组）的编译及解释程序。扩充 C0 文法的内容请见附录部分。编译程序的输入应为使用扩充 C0 文法的 C 源程序（文件）。编译程序在源程序错误的情况下输出错误信息及其定位；在源程序正确的情况下向解释器输出 Pcode，之后由解释器对 Pcode 进行解释执行。本项目全部代码手工编程实现，不涉及代码优化内容。

2 详细设计

本项目编译器的词法分析部分利用了有限状态机的设计，语法分析则采用了递归子程序法。本项目的解释器在 C 环境下模拟了运行栈和全局堆区在执行 Pcode 时栈上和堆上数据的变化，以解释执行源程序编译后的目标码。

程序结构如下所示



重要头文件说明

error.h :	定义错误代号
pcodename.h :	定义 Pcode 指令代号
symname.h :	定义符号代号
syntax.h :	声明语法分析子程序
systemconfig.h :	定义系统常量

源文件说明

error.c :	错误处理
gencode.c :	代码生成
getsym.c :	词法分析
interpret.c :	Pcode 解释器
syntax.c :	语法分析
table.c :	符号表管理
test2.c :	主程序入口

Figure 1 程序代码结构

2.1 词法分析

词法分析被语法分析调用，其功能主要包括判断下一个符号类型，并对常量及标识符保留其值。实现思路利用了有限状态机的设计方式，与《编译原理及编译程序构造》一书 P55 页算法基本一致。

2.1.1 主要函数和接口：

类型	函数/接口名	作用
Int	readch()	从全局文件指针（待编译源文件）读入一个字符
Int	getsym(int *sym)	将最新的符号存入 sym，使用前必须外部调用 readch()
char[]	token	sym 对应的字符串（标识符、保留字、字符串常量）
Int	num	sym 整型常量或者浮点型常量的整数部分
Int	linenum	源文件当前行号

2.1.2 枚举型符号名

```
enum symbol{
    EOFIL=-1, //程序结束
    WORDERRORSYM,
    IFTK, //if
    ELSETK, //else
    DOTK, //do
    WHILETK, //while
    FORTK,
    RETURNTK, //return
    SCANFTK, //scanf
    PRINTFTK, //printf
    MAINTK, //
    CONSTTK, //const
    NOTESSYM, //注释
    INTTK, //int
    FLOATTK, //float
    CHARTK, //char
    VOIDTK,
    IDEN, //标示符
    INTCON, //整型
    FLOATCON, //浮点型
    CHARCON,
    STRCON,
    SPACE, //空格等
    PLUS, //加法运算符
    MINUS, //减法运算符
    MULT, //星号
    DIV, //除号
    LPARENT, //左括号
    RPARENT, //右括号
    LBRACE, //左花括号
    RBRACE, //右花括号
    LBRACK,
    RBRACK, //方括号
    COMMA, //逗号
    SEMICN, //分号
    COLON, //冒号
    ASSIGN, //赋值运算
    EQL, //等判断
    GRE, //大于号
    GEQ, //大于等于
    LSS, //小于号
    LEQ, //小于等于
    NEQ, //不等于号
    PERIOD, //点
    QMARK,
    DQMARK, //59
    EOLINE//};
```

2.1.3 功能和注意事项

- 1) 需要在主函数所在源文件定义全局量 charin(FILE*), 函数 charin 为待编译文件的文件指针
- 2) 在调用 getsym (int*) 函数前, 必须在外部先调用 readch()
- 3) 调用 getsym (int*) 后, 当前符号存在 sym 指向的地址空间
- 4) 变量名在 symname.h 中定义声明, 保留字在函数 reserver 中判断。
- 5) 如果符号是整型常量, 数值大小会存在 num 和 real 中; 如果符号是浮点型常量, 数值大小会存在 real 中, 整数部分会存在 num 中; 如果符号是字符型常量, ASCII 码值会存在 num 和 real 中
- 6) 如果符号是标识符, 标识符内容将存在 token 中

2.2 解释器及 P-CODE 指令设计

由于涉及浮点型的运算, 本程序运行栈的原子单位没有直接使用整型或者浮点型变量, 而是使用了结构体。结构体包含三个成员变量: int type, int i, int f 分别标识栈内该原子单位的类型, 数值, 和浮点型数值。所有 P-code 指令都是针对栈上的结构体们进行操作的。

2.2.1 Pcode 指令设计

而 P-code 指令的结构体包含四个成员变量: int codename, int l, int i, float f 。分别标识指令码, 操作类型和操作数。具体 P-code 设计表格见下:

Table 1 Pcode 指令及其功能

Funname	l	i	作用
LIT	0	Int	将立即数加载到运行栈顶
LITC	0	Int	将立即数对应 ACSII 码值的字符加载到运行栈顶
LITF	0	Float	将浮点型立即数加载到运行栈顶
OPR	0	RET	无返回值函数返回, 基址和指令数寄存器取保留值
		RTS	有返回值函数返回栈顶元素, 基址和指令数寄存器取保留值
		OPP	栈顶元素取相反数
		ADD	次栈顶元素加栈顶元素, 类型为次栈顶元素类型
		MNS	次栈顶元素减栈顶元素, 类型为次栈顶元素类型

		MUL	次栈顶元素乘栈顶元素，类型为次栈顶元素类型
		DIV	次栈顶元素除栈顶元素，类型为次栈顶元素类型
		PEQU	若次栈顶元素等于栈顶元素，栈顶存 0，否则存 1
		PNEQ	若次栈顶元素不等于栈顶元素，栈顶存 0，否则存 1
		BIG	若次栈顶元素大于栈顶元素，栈顶存 0，否则存 1
		BEQ	若次栈顶元素大于等于栈顶元素，栈顶存 0，否则存 1
		SMA	若次栈顶元素小于于栈顶元素，栈顶存 0，否则存 1
		SEQ	若次栈顶元素小于等于栈顶元素，栈顶存 0，否则存 1
LOD	0	0	读取块内栈顶元素大小偏移位置的元素
	1	Int	读取块内立即数大小偏移位置的元素
	2	0	读取全局栈顶元素大小偏移位置的元素
	3	Int	读取全局立即数大小偏移位置的元素
STO	0	0	将栈顶元素存到块内次栈顶偏移位置
	1	Int	将栈顶元素存到块内立即数偏移位置
	2	0	将栈顶元素存到全局次栈顶偏移位置
	3	Int	将栈顶元素存到全局立即数偏移位置
CAL	0	Int	调用入口地址为立即数的函数
INT	0	Int	将栈指针上移立即数大小
JMP	0	Int	跳转到立即数地址
JPC	0	Int	如果栈顶元素为 0 则跳转到立即数地址
RED	I	0	从标准输入流读入一个整型到栈顶
	C	0	从标准输入流读入一个字符型到栈顶
	F	0	从标准输入流读入一个浮点型到栈顶
WRT	0	0	按照栈顶元素类型将其输出到标准输出流
WRTS	0	int	将字符串表中立即数位的字符串输出到标准输出流

2.2.2 解释器设计

解释器的设计使用两个较大大数组分别模拟了运行栈和全局变量所在的堆区（栈式实现），并设计了栈指针寄存器、基址寄存器、指令数寄存器（PC）和返回值寄存器。栈指针寄存器记录栈顶元素当前位置，基址寄存器存储当前基址位置，指令数寄存器记录当前执行到的 Pcode 数，返回值寄存器在函数返回时存储该函数的返回值。

具体指令的识别和执行利用了有限状态机模型。

2.3 语法分析及语法制导

2.3.1 语法分析程序设计

本编译器使用递归子程序法对给定文法的源程序进行分析。

给定的文法除了少数之外，绝大多数规则都满足 FIRST 集不相交。为了使用递归子程序法对该文法的语句进行语法分析，需要将该文法改写为 LL（1）文法；或者采用假读的方式在 FIRST 集相交的多重入口判断需要调用的规则；或者采用回溯的方式返回到多重入口，调用另一规则。本编译器使用了第二种方式，在多重入口的情况时，假读后面的符号，判断应该调用的规则。

递归子程序的划分和说明如下表

函数	作用
program()	语法分析母函数
constdeclaration(int *sym, char area[])	常量声明，传入当前符号指针和当前函数名
vardeclaration(int *sym, char area[], int *addr)	变量声明，传入当前符号指针和当前函数名
expression(int *sym, char FunID[])	表达式制导，传入当前符号指针和当前函数名
term(int *sym, char FunID[])	项制导，传入当前符号指针和当前函数名
factor(int *sym, char FunID[])	因子制导，传入当前符号指针和当前函数名
cal_function(int *sym, char funID[])	函数调用，传入当前符号指针和当前函数名
sentence(int *sym, char name[])	语句制导，传入当前符号指针和当前函数名

<code>function_body(int *sym, char name[], int funptr)</code>	函数声明（函数体），传入当前符号指针，函数名，函数在函数表中位置
<code>read(int *sym, char funid[])</code>	读语句制导，传入当前符号指针和当前函数名
<code>write(int *sym, char funid[])</code>	写语句制导，传入当前符号指针和当前函数名
<code>return_sentence(int *sym, char funID[])</code>	返回语句制导，传入符号指针和当前函数名
<code>if_sentence(int *sym, char funid[])</code>	条件语句制导，传入当前符号指针和当前函数名
<code>assign_sentence(int *sym, char funID[])</code>	赋值语句制导，传入当前符号指针和当前函数名
<code>do_sentence(int *sym, char funid[])</code>	Do-While 循环语句制导，传入当前符号指针和当前函数名
<code>for_sentence(int *sym, char funid[])</code>	For-循环语句制导，传入当前符号指针和当前函数名

2.4 错误处理

编译器错误处理部分将编译过程中的报错信息输出到标准输出（如果需要输出到文件需要修改 `test2` 文件中对 `err` 文件指针的初始化）错误代码和其对应的意义如下所示。

错误代码	错误内容
NO_ERROR	Default 内容，无错误
UNKNOWN_SYMBOL	不能识别的符号
STRING_TOO_LONG	标识符或字符串常量超长
WRONG_FORMAT_REAL_NUMBER	错误形式的浮点数
WRONG_FORMAT_INTEGER	错误形式的整数
UNFINISHED_STRING	字符串常量引号未匹配
UNFINISHED_PROGRAM	程序结构未完成
MISSING_SEMICN	遗留分号

DECLARATION_IS_NOT_START_WITH_TYPE	声明变量或常量时未标明数据类型
DECLARATION_HAVE_NO_EQ,	常量声明未赋值
DECLARATION_SHOULD_HAVE_A_ID,	声明未定义标识符
WRONG_ASSIGN_SYNTAX,	赋值语句的错误用法
WRONG_EXPRESSION,	表达式内部出现错误
ARRAY_OVERFLOW,	(可静态检测的) 数组越界
ARRAY_SUBVALUE_SHOULD_BE_INTEGER,	数组下标应为整数或表达式
USING_AN_ICON_WITHOUT_DECLARATION,	使用未声明的标识符
ERROR_VARIABLEDECLARATION,	错误的变量声明(可能由于不支持初始化)
ERROR_DATA_TYPE,	错误的数据类型
ERROR_FUNCTIONDECLARATION,	错误的函数定义
RE_DECLARATION,	重定义
ERROR_SENTENCE,///不够明确	其他句法错误
ERROR_PROGRAM_STRUCTURE,	错误的程序结构
ERROR_RETURN_TYPE,	错误的返回语句类型
PARENT_DISMATCH,	括号未匹配
ERROR_FOR_SENTENCE,	错误的 for 语句句法
ERROR_IN_WRITE_SENTENCE,	错误的写语句句法
ERROR_IN_READ_SENTENCE,	错误的读语句句法
MISSING_WHILE,	Do-While 语句遗漏 while 句
ERROR_CONDITION,	错误的判断条件
TOO_MANY_CODE_AFTER_MAIN_FUNCTION,	主程序结束后还有代码
MISSING_SENTENCE	遗漏句子

错误输出接口：**FILE*err** 错误信息会逐行输出在 **err** 指向的文本文件中（在该版本中逐行输出到标准输出）。值得注意的是，由于错误的种类层出不穷，本编译器不能保证正确识别所有的错误类型并完成编译。但绝大多数常见的错误都可以正确识别并继续编译。

2.5 函数调用关系

函数调用关系图请见附件“调用函数关系图.xps”

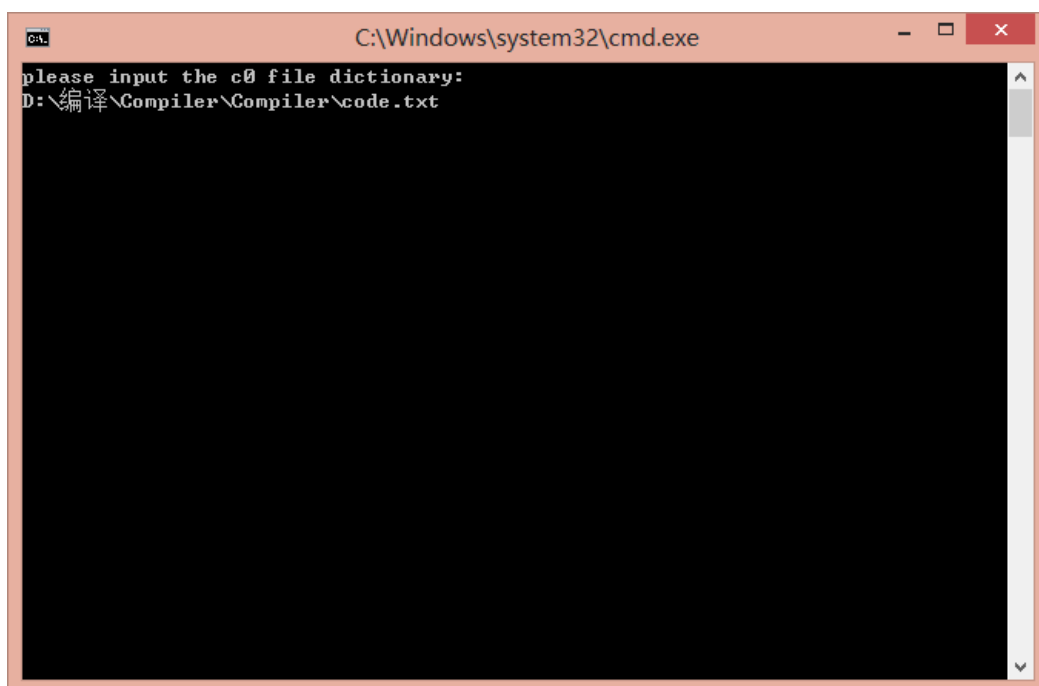
3 操作说明

3.1 运行环境

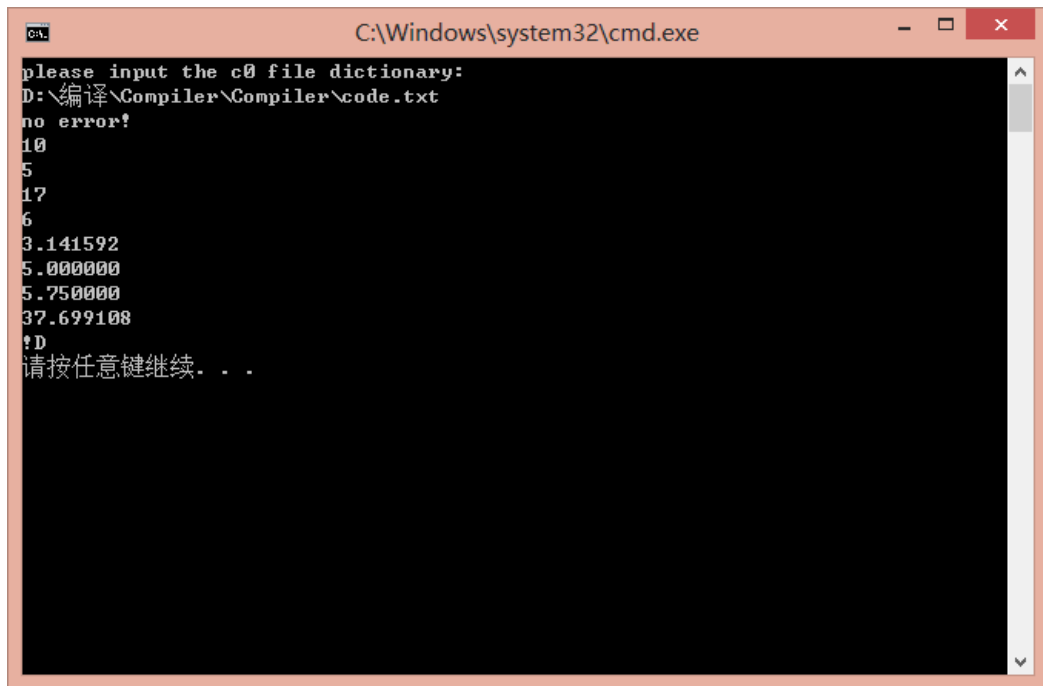
Visual Studio 2012。

3.2 操作说明

根据提示输入源文件文件路径，回车开始编译。



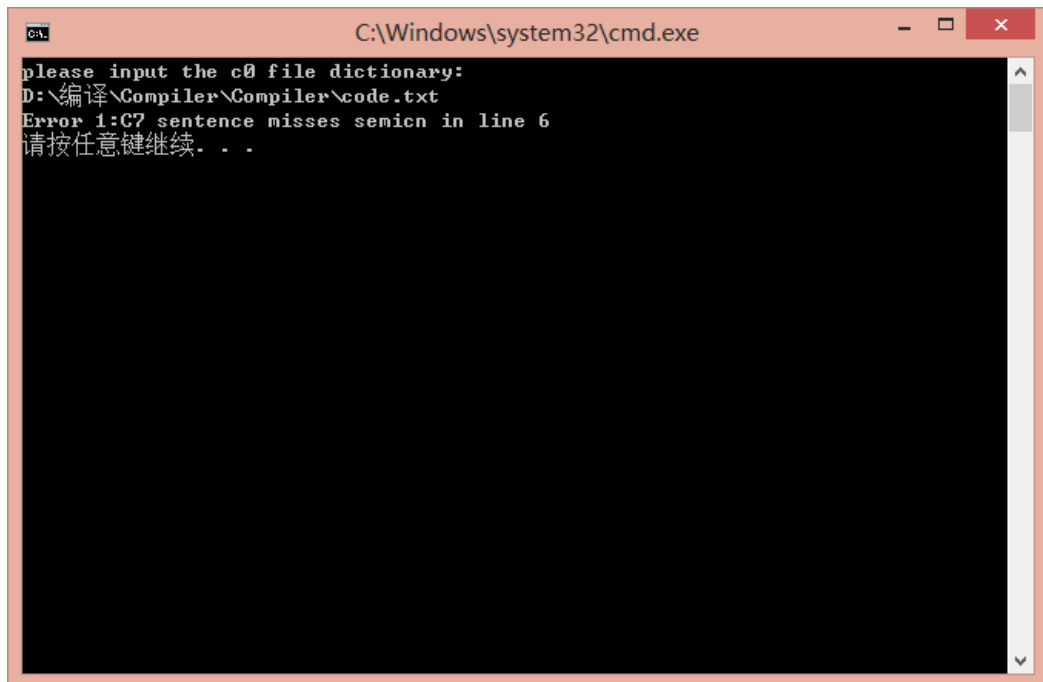
程序正确时输出“no error!” 并继续输出程序运行结果。



```
C:\Windows\system32\cmd.exe

please input the c0 file dictionary:
D:\编译\Compiler\Compiler\code.txt
no error!
10
5
17
6
3.141592
5.000000
5.750000
37.699108
?D
请按任意键继续. . .
```

程序有误时输出报错信息。



```
C:\Windows\system32\cmd.exe

please input the c0 file dictionary:
D:\编译\Compiler\Compiler\code.txt
Error 1:C7 sentence misses semicn in line 6
请按任意键继续. . .
```

4 测试报告

测试程序 1-6 主要测试编译器和解释器各个功能、句型用法等。

测试程序 7-10 主要是考察源程序有错误的情况下报错信息。

4.1 测试程序 1

```
void main()
{
    const int i0=10;
    const float f0=3.1415926;
    const char c0='A';
    int i1, i2, i3;
    float f1, f2, f3;
    char c1, c2;
    i1=(-3+10*i0-5)/16;
    i2=c1+10+3.5*2;
    i3=f0*2;
    printf(i0, '\n', i1, '\n', i2, '\n', i3, '\n'); /*10 5 17 6*/
    f1=(-3+10*i0-5)/16;
    f2=(-3.0+10*i0-5)/16;
    f3=(i0+2)*f0;
    printf(f0, '\n', f1, '\n', f2, '\n', f3, '\n'); /*5.0000 5.7500 37....*/
    c1=33;
    c2=c0+3;
    printf(c1, c2, '\n'); /*! D*/
}
```

测试重点：计算，类型，写语句

测试结果：

no error!	3.141592
10	5.000000
5	5.750000
17	37.699108
6	!D

4.2 测试程序 2

```
const char c='!';
int i[10], ii;
int iii;
int fun(int a, int b)
{
    int s;
    i[a]=b;
    return a+b;
}
```

```

}                                printf("this is a
void main()                      string", ii, i[1], i[2], c); /*this is a
{                                string 326!*/
    ii=fun(1, 2);                }
    fun(2, 6);

```

测试重点：常量变量定义，函数调用，赋值语句，写语句，存数组

测试结果：this is a string 326!

4.3 测试程序 3

```

void main()
{
    int limit, n, i;
    limit=3;
    i=0;
    do
    {printf("haha");
      i=i+1;
    } while(i<limit);
    /*  haha
    haha
    haha*/
}

```

测试重点：do-循环语句

测试结果：haha haha haha

4.4 测试程序 4

```

int factorial(int n)
{
    if(n==1)
        return 1;
    else
        return factorial(n-1)*n;
}
void main( )
{
    printf (factorial(5) ); /*120*/
}

```

测试重点：运行栈，函数调用。

测试结果：120;

4.5 测试程序 5

```

int number[20];
int max(int n)
{
    int i,m;
    m=0;
    i=0;
    do{
        if(number[i]>m)
            m=number[i];
        i=i+1;
    }while(i<n)
    return m;
}

int sum (int n)
{
    int i,s;
    s=0;
    for(i=0;i<n;i=i+1)
        s=s+number[i];
    return s;
}

float average(int n)
{
    float m;
    m=n;
    return sum(n)/m;
}

float var(int n)
{
    float s,m,ave;
    int i;
    m=n;
    ave=average(n);
    for(i=0;i<n;i=i+1)
        s=s+(number[i]-ave)*(number[i]-ave);
    return s/m;
}

void draw(int n)
{
    int m,i,j;
    char nl;
    nl=10;
    m=max(n);
    for(i=m;i>0;i=i-1)
    {
        for(j=0;j<n;j=j+1)
        {
            if(number[j]>=i)
                printf("*");
            else
                printf(" ");
        }
    }
}

void main()
{
    int n,i;
    char nl;
    nl=10;
    printf("please input
    how many numbers");
    scanf(n);
    printf("please input
    numbers",nl);
    for(i=0;i<n;i=i+1)
    {
        scanf(number[i]);
    }
    printf("the max
    number is:",max(n),nl);
    printf("the sum
    is:",sum(n),nl);
    printf("the average
    is:",average(n),nl);
    printf("the var
    is:",var(n),nl);
    draw(n);
}

```

测试重点：实型运算;

测试结果：统计结果

4.6 测试程序 6

```
void main()
{
    int i;
    for(i=0;i<3;i=i+1)
        printf("haha");
    printf("gaga");
}
```

错误：括号不匹配

4.7 测试程序 7

```
const int num=10;
void main()
{
    int ;
    printf(num);
}
```

错误：变量定义不完整

4.8 测试程序 8

```
int a;

const int a=3;

void main()

{

    printf("hello");

}
```

错误：程序结构有误

4.9 测试程序 9

```
void main()
{
```

```
int i;
for(i=0;i<3;i=i+1)
    printf("haha");
printf("gaga")
}
```

错误：遗失分号

4.10 测试程序 10

```
int a[3];

void main()

{

    a[4]=1;

}
```

错误：数组越界（静态）

5 总结感想

编译课程设计是我第一次完成千行级别的代码项目，也给我留下了很多感触。一开始和大多数同学一样，我对这么庞大的项目有些不知所措，不知道怎么开始这么庞大的项目。但是随着一步步的深入，我逐渐感受到了软件工程的厉害之处：如同庖丁解牛一般划分模块、定义功能和接口、单元测试，直到最后组装成为一个完整的编译器。之后一周一周地调试错误，不断完善尚未实现的功能。最后提交项目时有一种造物一般的感动。

之前写的代码绝大部分都不是 C 语言的（面向对象语言为主），考虑到锻炼的目的，这次的语言我选择使用 C 语言来实现。一开始很多函数、变量的封装没有办法实现导致前期的工作就遇到了很多困难。后来阅读了一些代码之后改变了自己非常不好的编码风格（虽然现在依然还不是很好，但是逐步有了一些进步哈~）现在对于一个软件工程代码的组织有了更深的体会。

当然整个编译课程设计也留给了我很多遗憾，最大的莫过于一开始没有敢于直接选高级的编译（包括优化部分），后来想改选高级时由于很多已经完成的工作可能要完全放弃也没有下定决心。实在是太可惜了！在十二周时，我的编译器就已经基本完成了，但是因为对于项目难度的错误估计让我没能实现编译过程中相当重要的优化部分，实在是个很大的遗憾。

另外整个编译课程设计的过程中尤其感谢史晓华老师、杨海燕老师，以及众多不辞辛苦的助教们。没有这些老师和助教的帮助我们在课程设计的过程中的困难可能大上几倍甚至十几倍，他们辛苦的付出值得我们真诚的感谢！

6 附录

6.1 C0 文法（数组-有实型）

<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 32,33,35-126 的 ASCII 字符 } "

<程序> ::= [<常量说明部分>] [<变量说明部分>] { <有返回值函数定义部分> | <无返回值函数定义部分> } <主函数>

<常量说明部分> ::= 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>

<变量说明部分> ::= <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> '] ' }

<类型标识符> ::= int | float | char

<有返回值函数定义部分> ::= <声明头部> '(' <参数> ')' '{' <复合语句> '}'

<无返回值函数定义部分> ::= void<标识符> '(' <参数> ')' '{' <复合语句> '}'

<复合语句> ::= [<常量说明部分>] [<变量说明部分>] <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>; | <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> ::= do<语句>while '(' <条件> ')' | for '(' <标识符> = <表达式>; <条件>; <标识符> = <标识符> (+|-)<步长> ')' <语句>

<步长> ::= <非零数字> { <数字> }

<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> ')']