**同济大学计算机系**

**编译原理课程实验报告**

****

**学 号 1953186**

**姓 名 龚乙骁**

**专 业 信息安全**

**授课老师 高秀芬**

**实验二 中间代码生成器设计与实现**

1. **运行和开发环境**

OS：Win10

开发语言：C++

IDE：Visual Studio 2019

开发库：MFC

注：代码中文可能乱码，因为项目采用的是多字节字符集，而不是Unicode字符集，需要手动调整编辑器编码

1. **能识别的单词、语法、扩充的功能**
2. **能识别的单词：**

关键字：int | void | if | else | while | return

标识符： 字母（字母|数字）\* （注：不与关键字相同，不能以数字开头）

数值：数字（数字）\*

赋值号： =

算符： + | - | \* | / | = | == | > | >= | < | <= | !=

界符： ；

分隔符： ，

注释号： /\* \*/ | //

左括号： （

右括号： ）

左大括号： {

右大括号： }

字母：| a |....| z | A |....| Z |

数字：0| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

结束符：#（宏定义和头文件引用中也可以用，本次实验中代码没有终结符）

1. **能识别的语法：**

* Program ::= <类型> < ID>’(‘ ‘)’<语句块>
* <类型>::=int | void
* <ID>::=字母(字母|数字)\*
* <语句块> ::= ‘{’ <内部声明> <语句串>‘}’
* <内部声明> ::= 空 | <内部变量声明>{; <内部变量声明>}
* <内部变量声明>::=int <ID> （注： { }中的项表示可重复若干次）
* <语句串> ::= <语句> { <语句> }
* <语句> ::= <if语句> |< while语句> | <return语句> | <赋值语句>
* <赋值语句> ::= <ID> =<表达式>;
* <return语句> ::= return [ <表达式> ] （注：[ ]中的项表示可选）
* <while语句> ::= while ‘(’ <表达式> ‘)’ <语句块>
* <if语句> ::= if ‘(‘<表达式>’)’ <语句块> [ else <语句块> ]（注：[ ]中的项表示可选）
* <表达式>::=<加法表达式>{ relop  <加法表达式> }  （注：relop-> <|<=|>|>=|==|!=）
* <加法表达式> ::= <项> {+ <项> | -<项>}
* <项> ::= <因子> {\* <因子> | /<因子>}
* <因子> ::=ID|num | ‘(’ <表达式>‘)’

1. **为了方便语义分析，以上语法经过处理可以化为以下文法：**

**其中@表示空**

**S ::= Program**

**Program ::= FunRtType FunDec Block**

**VarType ::= int**

**FunRtType ::= void | int**

**FunDec ::= <ID> PreFunSign ( ParameterList )**

**PreFunSign ::= @**

**ParameterList ::= ParamDec , ParameterList | ParamDec | @**

**ParamDec ::= VarType <ID>**

**Block ::= { DefList StatementList }**

**DefList ::= Def DefList | @**

**Def ::= VarType <ID> ;**

**StatementList ::= Statement StatementList | @**

**Statement ::= AssignStatement ; | ReturnStatement ; | IfStatement | WhileStatement**

**AssignStatement ::= <ID> = Exp**

**Exp ::= ArithmeticExp | Exp Relop ArithmeticExp**

**ArithmeticExp ::= Item | Item + Item | Item - Item**

**Item ::= Factor | Factor \* Factor | Factor / Factor**

**Factor ::= <INT> | ( Exp ) | <ID>**

**ReturnStatement ::= return Exp | return**

**Relop ::= > | < | >= | <= | == | !=**

**IfStatement ::= if IfSign1 ( Exp ) IfSign2 Block ElseBlock**

**IfSign1 ::= @**

**IfSign2 ::= @**

**ElseBlock ::= @ | ElseSign else Block**

**ElseSign ::= @**

**WhileStatement ::= while WhileSign1 ( Exp ) WhileSign2 Block**

**WhileSign1 ::= @**

**WhileSign2 ::= @**

**4. 扩充的功能：**

1. 增加了出错处理

如果在预处理或者词法分析时出错，会弹出窗口进行提示出错类型以及出错行数（指预处理之后的行数），

语法分析出错时会提醒并给出出错行数，

语义分析能够提示的错误有：变量未定义，内部变量重定义，函数参数重定义，函数缺少返回值

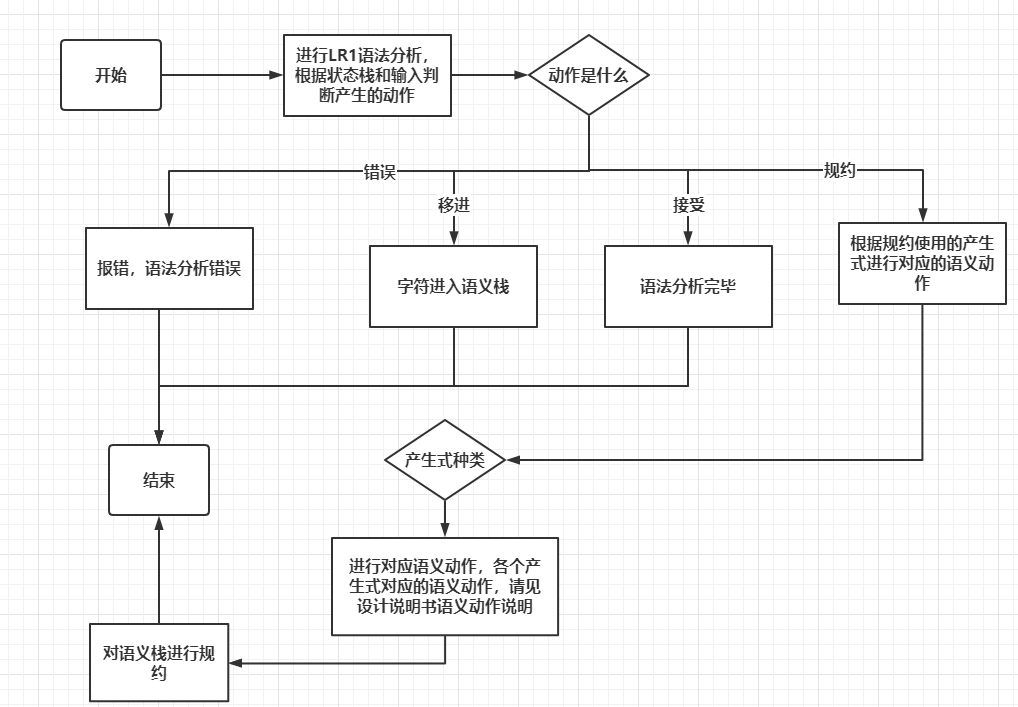
2. 语义分析考虑了符号表的作用域，能够适应更复杂的函数声明和定义，针对本次设计只考虑了一个函数定义，如需要定义多个函数只需要略微修改文法即可，并且对函数返回值进行了处理，为实现函数调用做准备, 同时可以查看符号表，并且可以看到各个符号的作用域

3 .预处理能够实现去掉多余的空格回车，注释，去掉头文件引用以及展开宏定义（为了便于阅读，预处理时没有去掉全部的回车）

4. 能够读取本地的代码文件到编辑框，也能将编辑框的文件保存到本地文件中（文件类型可以是txt，cpp，c）

1. **分析算法的主程序框图**

**程序框图：**



**语义动作说明：**

S ::= Program

无特殊动作，对语义栈进行规约

Program ::= FunRtType FunDec Block

记录Program函数到全局符号表，之后对语义栈进行规约

VarType ::= int

无特殊动作，对语义栈进行规约，并标明符号是int

FunRtType ::= void | int

无特殊动作，对语义栈进行规约，并标明符号是int或者是void

FunDec ::= <ID> PreFunSign ( ParameterList )

无特殊动作，对语义栈进行规约

PreFunSign ::= @

这个规约代表着一个函数即将被声明，因此需要做以下动作：

为这个函数创建一个符号表，并且让这个符号表入表栈，代表作用域切换到这个函数内

将函数名记录到全局符号表中

创建存储这个函数返回值的变量，将这个变量记录到这个函数的符号表中

产生该函数声明的四元式

对语义栈进行规约

ParameterList ::= ParamDec , ParameterList | ParamDec | @

无特殊动作，对语义栈进行规约

ParamDec ::= VarType <ID>

这个规约代表着一个函数参数即将被声明，因此需要做以下动作：

查找函数符号表，检查是否是重定义，是则报错

如果不是重定义，那么将这个符号加入函数符号表

对语义栈进行规约

Block ::= { DefList StatementList }

无特殊动作，对语义栈进行规约；

注：由于本次设计除去program外，Block只会在if和while语句中产生，本次实现Block作用域区分是在if和while语句的语义动作中实现的，但是最合理的设计是对Block相关文法进行修改并在语义动作中实现作用域区分。

DefList ::= Def DefList | @

无特殊动作，对语义栈进行规约

Def ::= VarType <ID> ;

这个规约代表着一个内部变量即将被声明，因此需要做以下动作：

查找函数符号表，检查是否是重定义，是则报错

如果不是重定义，那么将这个符号加入函数符号表

对语义栈进行规约

StatementList ::= Statement StatementList | @

无特殊动作，对语义栈进行规约

Statement ::= AssignStatement ; | ReturnStatement ; | IfStatement | WhileStatement

无特殊动作，对语义栈进行规约

AssignStatement ::= <ID> = Exp

这个规约代表即将进行一次赋值，因此需要做以下动作：

查找本层以及所有上层符号表，确定变量被声明，否则报错

如果变量已经被声明，则生成赋值语句的四元式

对语义栈进行规约

Exp ::= ArithmeticExp | Exp Relop ArithmeticExp

如果规约的是算术表达式，则直接进行语义栈规约

如果规约的是关系表达式，则将关系运算当成控制语句，生成2个跳转四元式和2个赋值四元式，并生成临时变量，在赋值四元式中存储关系运算结果

之后对语义栈进行规约

ArithmeticExp ::= Item | Item + Item | Item – Item

如果规约项是单独一个，那么直接对语义栈进行规约

如果规约的是项的加减，那么生成对应的运算四元式，并生成临时变量存储结果，之后对语义栈进行规约

Item ::= Factor | Factor \* Factor | Factor / Factor

如果规约因子是单独一个，那么直接对语义栈进行规约

如果规约的是因子的乘除，那么生成对应的运算四元式，并生成临时变量存储结果，之后对语义栈进行规约

Factor ::= <INT> | ( Exp ) | <ID>

如果因子是常量或者是表达式，那么直接对语义栈进行规约

如果因子是标志符，那么需要查找符号表，判断是否是未定义变量，是则报错

ReturnStatement ::= return Exp | return

如果规约有返回值，那么生成两个赋值四元式，第一个是将返回表达式的值赋值给函数的返回值变量，第二个是将函数返回值变量的值赋值给函数名

如果规约没有返回值那么先查找符号表确定该函数是否需要返回值，如果需要那么报错，如果不需要就生成返回的四元式

之后对语义栈进行规约

Relop ::= > | < | >= | <= | == | !=

无特殊动作，对语义栈进行规约

IfStatement ::= if IfSign1 ( Exp ) IfSign2 Block ElseBlock

这个规约代表着一个if语句即将生成，此时需要回填先前在规约IfSign等标志时产生的跳转四元式；如果无else只需要回填真、假出口；如果有else则需要回填真、假出口以及真语句块结束时的if出口

同时表示该if语句的作用域完成，弹出该if的符号表，切换作用域到上一层

之后对语义栈进行规约

IfSign1 ::= @

这个规约代表着之后将进入一个if语句

生成该if的符号表，并且入表栈，切换作用域

之后对语义栈进行规约

IfSign2 ::= @

这个规约代表着即将进入if的第一个语句块

此时需要产生两个跳转四元式，第一个对应if为真跳转位置，第二个对应if为假跳转位置，同时将这个两个四元式地址入回填栈，在对整个if语句进行规约时进行回填

由于这个规约产生时，下一个四元式的地址刚好是if的真地址，因此将这个真地址入语义栈，为回填做准备

ElseBlock ::= @ | ElseSign else Block

如果规约中没有else块，那么产生一个跳转四元式，跳转地址为if的出口，并将该四元式入回填栈，等待回填；此时下一个四元式地址即为if出口，将其入语义栈等待回填

如果规约中有else，那么代表着一个else块已经规约完毕，此时需要将else的符号表出栈，切换作用域到上一层，之后对语义栈进行规约

ElseSign ::= @

这个规约代表着if有else块

首先创建else的符号表并入栈，切换作用域到该语句块

之后产生一个跳转四元式，这个跳转是if的真语句块的出口，跳转位置就是if语句的出口，将这个四元式入回填栈等待回填

此时下一个四元式地址就是if语句else语句块的入口，将这个地址入语义栈，为回填做准备

最后对语义栈进行规约

WhileStatement ::= while WhileSign1 ( Exp ) WhileSign2 Block

这个规约代表着一个while语句即将规约完毕

首先生成无条件跳转四元式，跳转地址就是while语句判断跳转四元式的地址

之后回填由WhileSign产生的待回填的真出口和假出口

然后将while的符号表出栈，切换作用域到上一层

最后对语义栈进行规约

WhileSign1 ::= @

这个规约代表即将进行一个while语句的规约

首先生成while的符号表并入栈，切换作用域到while的语句块

此时下一个四元式的地址就是while判断四元式的地址，将这个地址入语义栈，为之后无条件回跳四元式做准备

最后对语义栈进行规约

WhileSign2 ::= @

这个规约代表着即将进行while语句块的规约

首先根据while的判断表达式生成两个条件跳转四元式，分别对应真出口和假出口，将这两个四元式入回填栈，在之后while语句规约时会回填

此时下一个四元式地址就是while的真出口，将这个地址入语义栈，为之后回填做准备

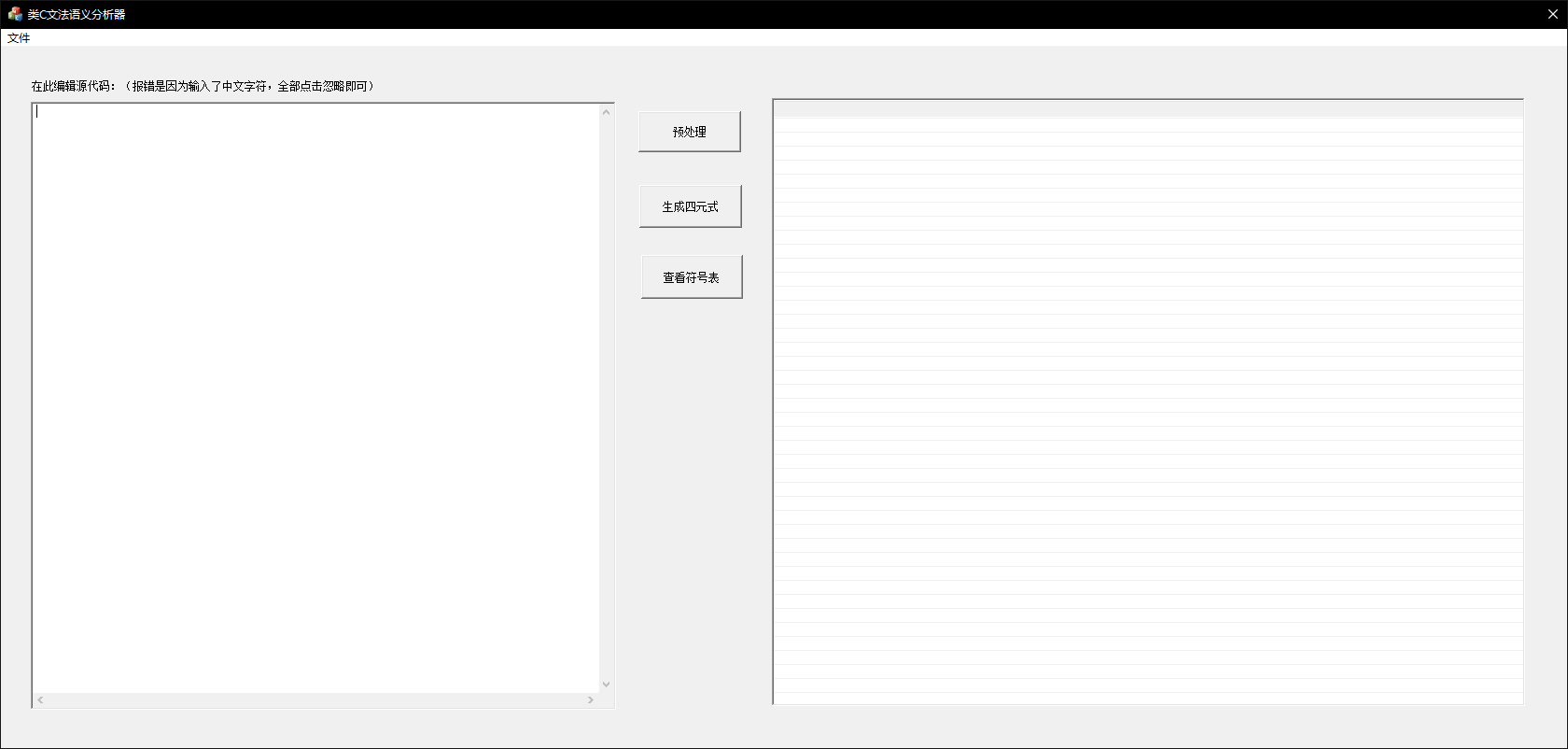
最后对语义栈进行规约

1. **运行结果**

可执行文件：



**初始界面**：



可以进行代码编辑、导入、保存操作，以及预处理，生成四元式操作，生成四元式时会自动进行预处理；生成四元式后可以查看符号表，直接查看没有内容

接下来开始测试中间代码生成器功能（注意代码结尾不要添加#号）

**测试代码：**



int program(int a,int b,int c)

{

int i;

int j;

i=0;

if(a>(b+c))

{

int j;

int h;

j=a+(b\*c+1);

if(a>(b+c))

{

int j;

}

}

else

{

int j;

j=a;

}

while(i<=100)

{

int k;

int a;

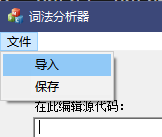
a = i + j;

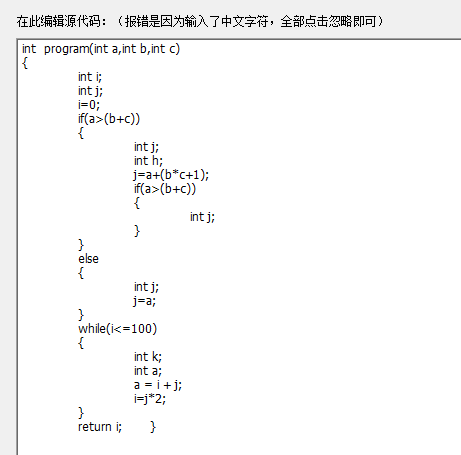
i=j\*2;

}

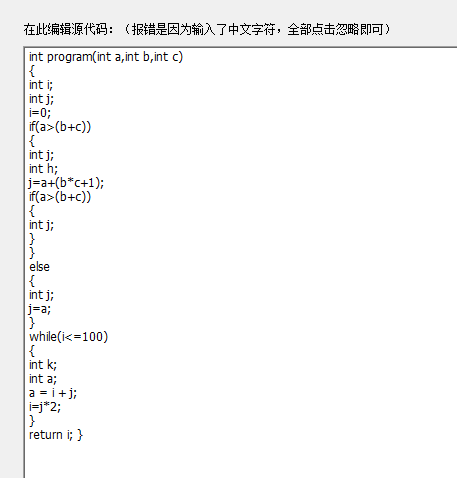
return i; }

将测试代码导入中间代码生成器（也可以手动在编辑框输入）：

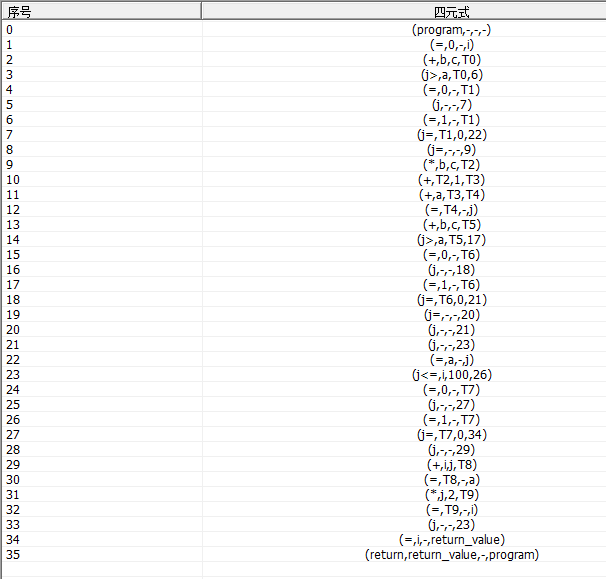




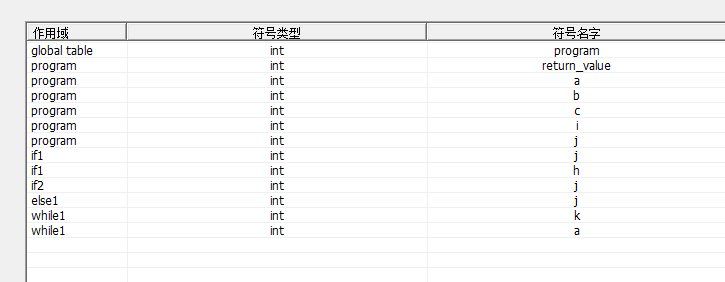
**预处理结果：**



**生成四元式：**

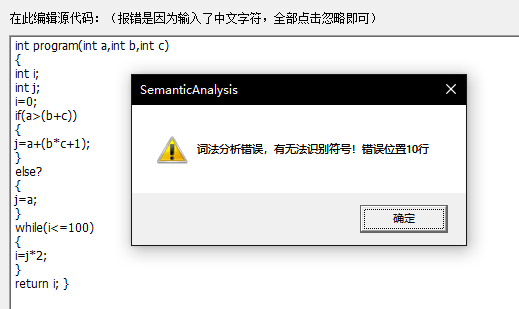


**查看符号表**

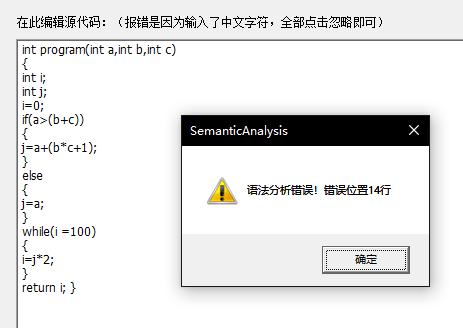


**错误示例**

词法错误（第十行输入了一个’?’）：

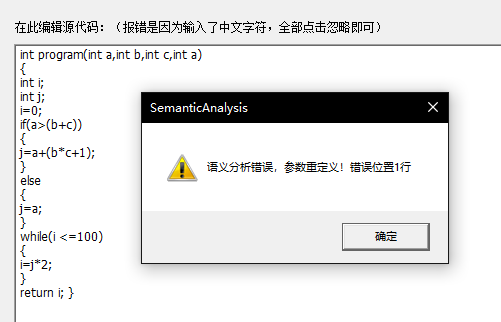


语法错误（第14行while的条件表达式改为了赋值语句）：

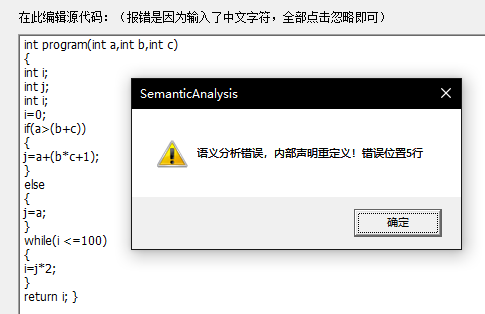


语义错误

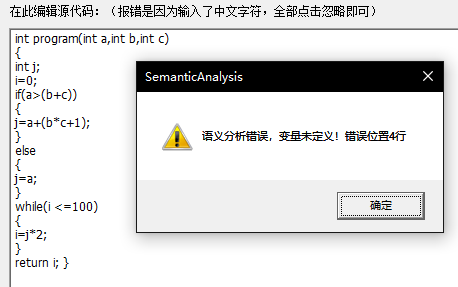
参数重定义：



内部变量重定义

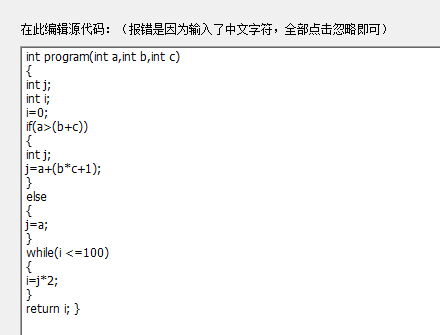


变量未声明：



可以看到在不同的语句块中，相同的变量定义不会冲突：

第9行在if语句块中加入了int j，并没有报错说j重定义



缺少返回值：

