《编译技术》
课程设计文档

学号：_____12061036_____

姓名：_____李明浩_____

2015 年 1 月 9 日

一、 需求说明

1. 文法说明

文法“扩充 C0 文法-高-数组-无实型”的特点如下：

1) 描述了非分程序结构的类 C 语言；
2) 支持常量、变量、函数定义；
3) 支持一维数组；
4) 变量种类支持整型和字符，含弱类型检查；
5) 支持 if 型条件分支语句；
6) 支持 while 和 for 型循环语句；
7) 允许函数内递归调用。

所给原文法中，由于公共部分交集过多、非终结符为中文，不方便程序的编写、含有部分冗余产生式（如<有返回值函数调用语句>和<无返回值函数调用语句>右部完全相同），同时部分划分不合理（过密和过疏的情况都有），因此在消除左递归、提取公共部分、简化等处理后，文法“扩充 C0 文法-高-数组-无实型”改写如下：

```
//词法分析相关
<Letter> ::= _|a|...|z|A|...|Z
<Digital> ::= 0|<NotZeroDigital>
<NotZeroDigital> ::= 1|...|9
<Character> ::= ‘(+|-|*|/|<Letter>|<Digital>)’
<String> ::= “{ 32,33,35-126 ASCII Char}”
<UnsignedInt> ::= <NotZeroDigital>{<Digital>}
<Integer> ::= [+|-]<UnsignedInt>|0
<Identifier> ::= <Letter>{<Letter>|<Digital>}
<TypeIden> ::= int|char

//常量定义相关
<ConstDec> ::= const<ConstDef>;{const<ConstDef>;}
<ConstDef> ::= int<Identifier>=<Integer>{,<Identifier>=<Integer>}|
                char<Identifier>=<Character>{,<Identifier>=<Character>}

//变量定义相关
<VarDec> ::= <VarDef>;{<VarDef>;}
<VarDef> ::=
<TypeIden>(<Identifier>(’[‘<UnsignedInt>’]’|<Null>)){,(<Identifier>(’[‘<UnsignedInt>’]’|<Null>))}

//函数定义相关
<ParaTable> ::= <TypeIden><Identifier>{,<TypeIden><Identifier>}|<Null>
<FuncDef> ::= (<TypeIden>|void) <Identifier>’(‘<ParaTable>’)’’{‘<CompStmt>’}’

//表达式相关
```

<Expression> ::= [+|-]<Term>{(+|-)<Term>}
<Term> ::= <Factor>{(*|/)<Factor>}
<Factor> ::= <Identifier>('['<Expression>']'|<Null>)|<Integer>|<Character>|<CallStmt>|'('<Expression>')'

//多种语句
<CallStmt> ::= <Identifier>'('(<Expression>{,<Expression>}|<Null>)')'
<AssignStmt> ::= <Identifier>(=<Expression>|'['<Expression>']'=<Expression>)
<ScanStmt> ::= scanf'('<Identifier>{,<Identifier>}')'
<PrintStmt> ::= printf'('(<String>,<Expression>|<String>|<Expression>)')'
<ReturnStmt> ::= return['('<Expression>')']

//条件语句相关
<RelOp> ::= <|<=|>|>=|!=|==
<Condition> ::= <Expression>(<RelOp><Expression>|<Null>)
<CondStmt> ::= if'('<Condition>')'<Statement>[else<Statement>]

//循环语句相关
<WhileStmt> ::= while'('<Condition>')'<Statement>
<ForStmt> ::=
for'('<Identifier>=<Expression>;<Condition>;<Identifier>=<Identifier>(+|-)<UnsignedInt>')'<Statement>

//高层产生式
<Statement> ::=
<CondStmt>|<WhileStmt>|<ForStmt>|'{'<StmtList>'}'|<CallStmt>;|<AssignStmt>;|<ScanStmt>;|<PrintStmt>;|<Null>;|<ReturnStmt>;
<StmtList> ::= {<Statement>}
<CompStmt> ::= [<ConstDec>][<VarDec>]<StmtList>
<Main> ::= void main '('')''{'<CompStmt>'}'
<Program> ::= [<ConstDec>][<VarDec>]{<FuncDef>}<Main>

原文法 G[<程序>] = G'[<Program>]，是合法的改写。各部分结构明晰，公共部分少，仅靠简单预读即能确定入口，方便词法及语法分析的进行。

2. 目标代码说明

目标代码为基于寄存器-寄存器指令集的 RISC 架构的 MIPS 指令。所使用的指令有：

a) 数字运算相关
   add、addu、addiu、sub、subu、mul、div
b) 存储相关
   lw、sw、li、la
c) 跳转相关
   条件跳转：beq、bne、bgt、bge、blt、ble
   无条件跳转：j、jal、jr
d) 其他

move、syscall

运行编译器后，目标代码将生成在工程目录下，名称为 mips.asm。

目标代码中分为两部分：数据段和代码段。数据段用来存放字符串和全局常量。代码段的开始部分含有主动调用 main 函数的 jal 指令，并在 main 函数执行结束后跳转至_EOF 标签，通过 syacall 正常结束程序。”#”后注释为辅助理解所用。

3. 优化方案

1) 划分基本块与建立流图

对四元式进行基本块的划分和流图的建立，明确程序中分支、路径等控制流信息，为之后进一步优化打下基础。要求所有基本块满足其定义，流图完整，不缺失前驱与后继。

2) 基本块内优化

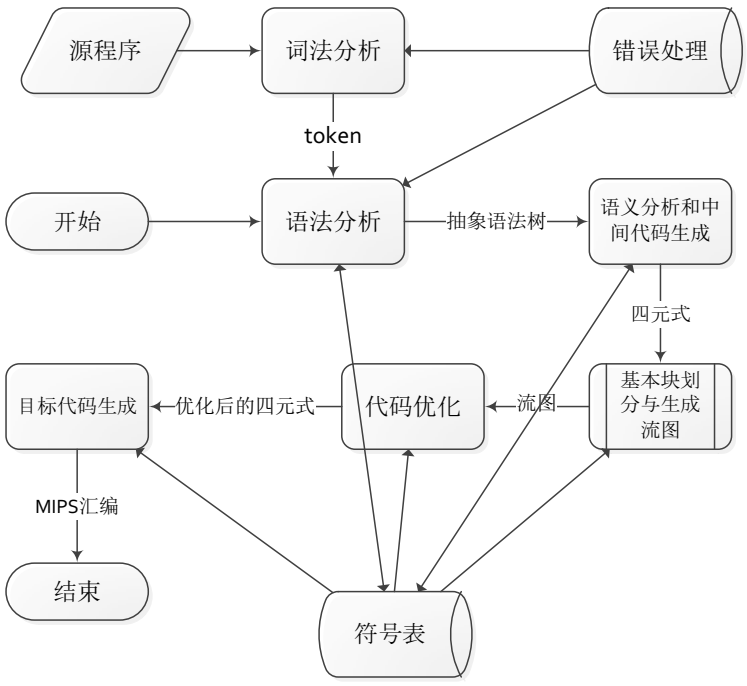
通过建立 DAG 图，达到消除局部公共子表达式、常数合并与传播、死代码删除的目的。

3) 全局优化

结合流图进行活跃变量数据流分析。要求得到每个函数的冲突图，以便代码生成时全局寄存器分配所用。

二、 详细设计

1. 程序结构



2. 类和方法及其调用关系

各类中设置、获取成员值的方法基本上以 Get、Set 或直接以成员名称命名，在这里将省略。另外使用默认构造、析构函数的也将一并省略。

类的成员具有”m_”前缀，类的静态成员具有”sm_”前缀。类的私有方法具有”__”前缀。

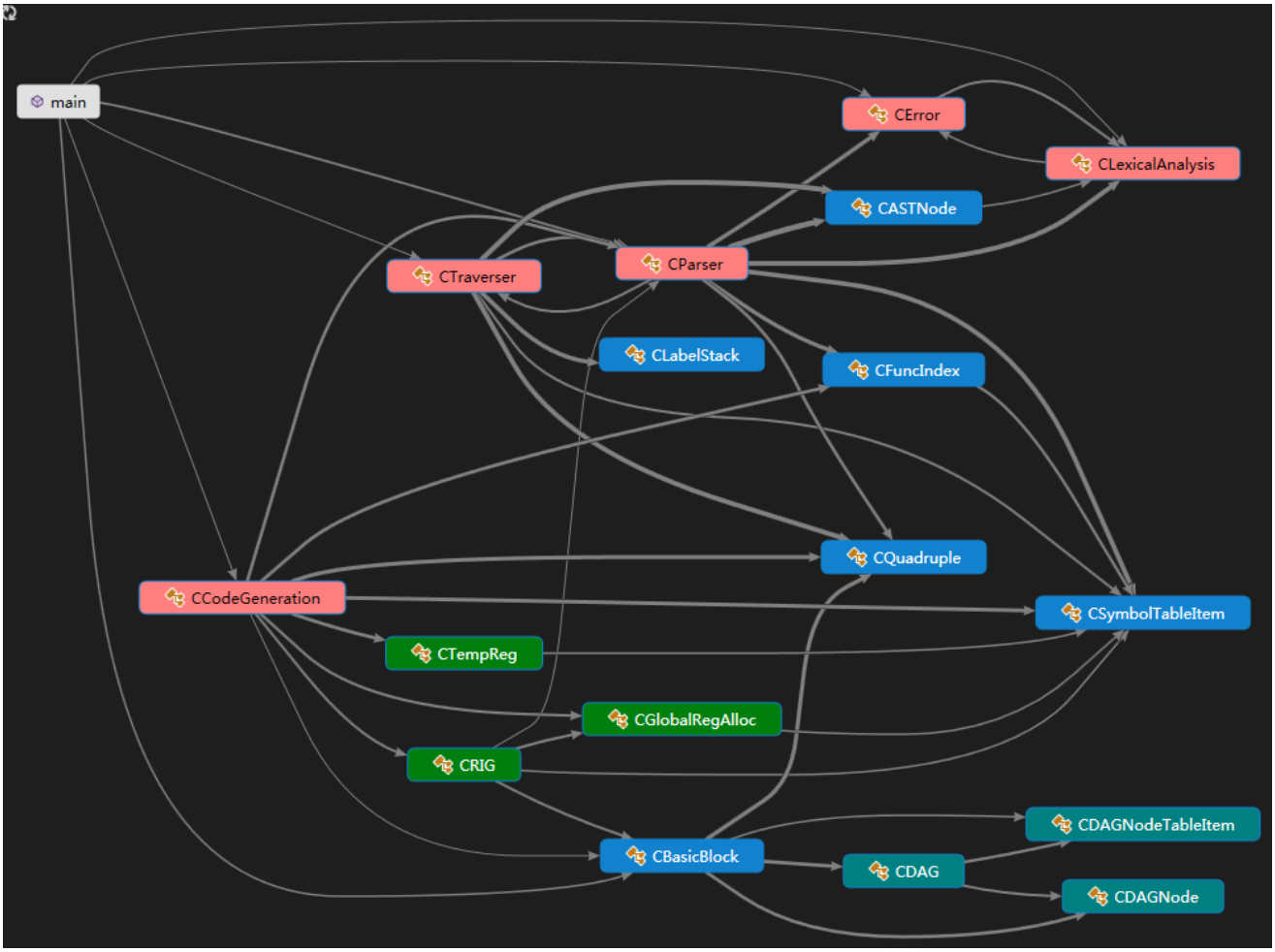
作为数据结构的类有：

CASTNode、CSymbolTableItem、CFuncIndex、CLabelStack、CQuadruple、CBasicBlock、CDAGNode、CDAGNodeTableItem、CDAG、CTempReg、CRIG、CGlobalRegAlloc。

作为功能性的类有：

CLexicalAnalysis、CParser、CTraverser、CCodeGeneration、CError。

全局调用关系如下图：



作为功能性的类为红色，作为数据结构的类为其他颜色，其中绿色为全局寄存器分配相关类，青色为 DAG 图相关类。

1) 词法分析类 CLexicalAnalysis

成员：

所有成员皆为静态变量。

sm_iCharPointer 为当前所读字符的位置。

sm_iColNumber 和 sm_iLineNumber 分别为当前所在位置的列与行数。

sm_iNumber 用来存放读到的整型数字。sm_sWord 用来存放当前读到的单词。

sm_mReservedWord 是保留字集。

方法：

__AddCharPointer 方法使得当前所读字符位置前进 1 位。

__DecCharPointer 方法使得当前所读字符位置后退 1 位。

__InitMap 方法初始化保留字集，即将 13 种保留字加入 map。

__Insert2Map 是__InitMap 方法中使用的向 map 增加项的子方法。

__isDigit 用于判断字符是否为数字。

__isLetter 用于判断字符是否为字母（包含下划线”_”）。

__Lowercase 用于将字符串中所有大写字母转换为小写，由于标识符不区分大小写，因此所有读到的单词将一律经过此方法转换为小写形式。

__PresentChar 方法返回代码在 sm_iCharPointer 位置上的字符。

isEOF 判断是否已经读到代码末尾。

词法分析关键方法：

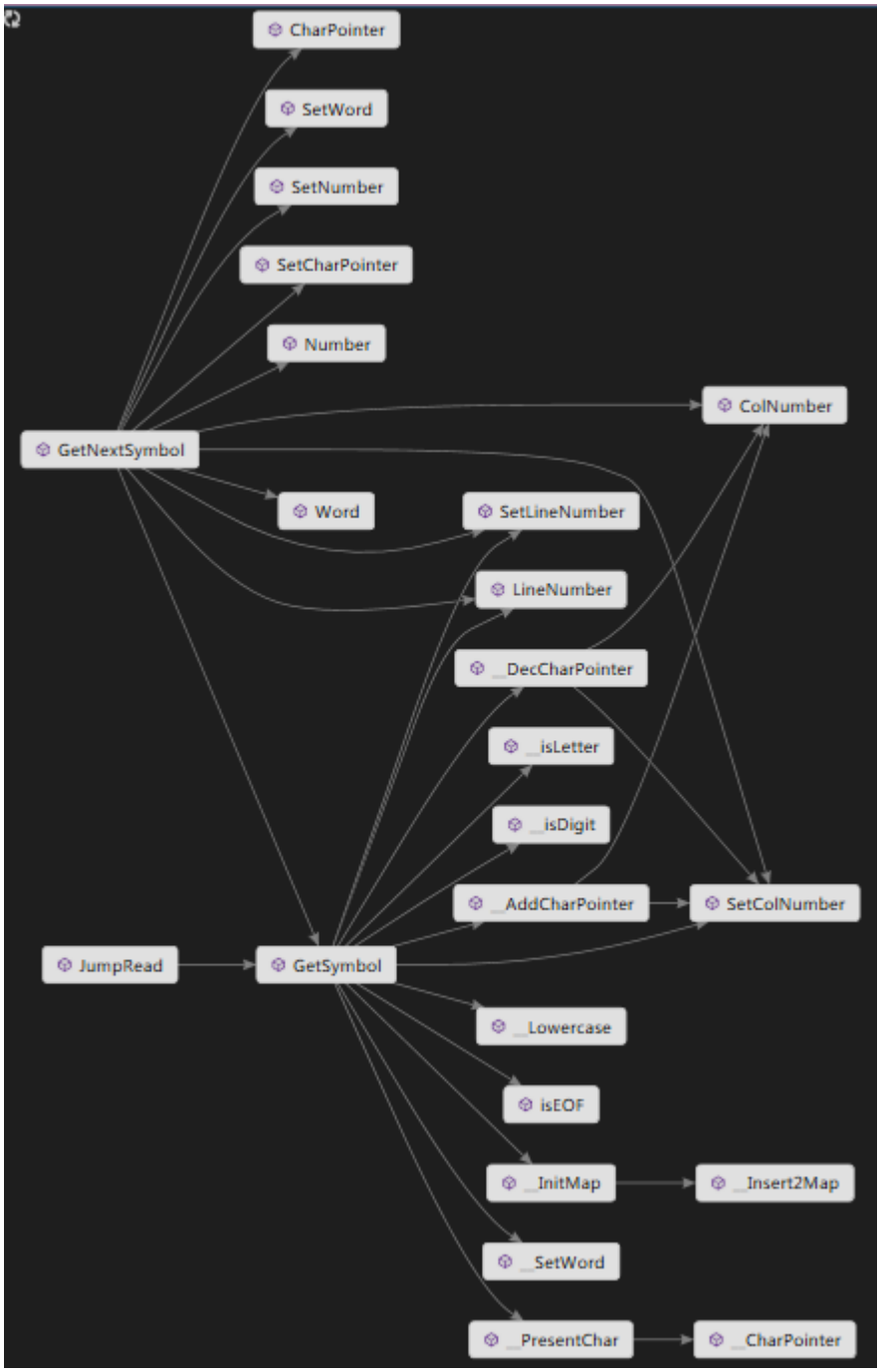
GetSymbol 为本类最主要的方法。作用是获得下一个 token 的 token 类型及其值。实现方法是每次根据__PresentChar 返回的字符和所有单词首字符集合判断分支，组合成单词后即可返回相应的类型与值。

GetNextSymbol 为实现超前读的方法。该方法主要通过循环调用 GetSymbol 方法，但在调用之前要保存当前的 sWord、INumber、行列号等信息，调用结束后进行恢复，这样超前读只是返回所需位置的单次种类与值信息，而不改变现有的词法分析进度与状态。

JumpRead 是另一主要的方法，主要用于错误处理的跳读。参数 sStopSet 是停止单词集合，当读到此集合内的单词时方法才结束。

调用关系：

本类不调用其他任何类的任何方法。可以明显地看出 GetSymbol 与 GetNextSymbol 方法的复杂性。



2) 抽象语法树结点类 CASTNode

成员：

成员 m_eNodeKind 是结点类型的枚举。

对于抽象语法树（AST）的结点类型设计如下：

- 计算型：PLUS, MINU, MULT, DIV
- 比较型：LSS, LEQ, GRE, GEQ, EQL, NEQ
- 参数表型：PARATABLE
- 定义声明型：CONSTDEC, CONSTDEF, VARDEC, VARDEF, FUNCDEF
- 语句型：COMPSTMT, CONDSTMT, WHILESTMT, FORSTMT, CALLSTMT, ASSIGNSTMT, SCANSTMT, PRINTSTMT, RETURNSTMT, STMTLIST
- 程序型：PROGRAM
- 其他：OTHER, TYPE, IDENTIFIER, INTEGER, ASSIGN_LEFT_ID, STRING, CHARACTER

m_iLine 代表此结点所在源程序的行数，记录此项是因为由源程序变为 AST 时不记录行数信息则会失，方便错误处理报出错误位置。

m_pChild 为儿子节点数组，最大值这里设为 6（for 语句会用到此最大值）。

m_pSibling 为相邻结点指针。

m_sValue 记录此结点的值。这里所有值全部使用 string 类型进行存储。

m_sQuadTemp 记录此结点作为临时变量时的标识符。

方法：

CASTNode 有两个重载，一种是指定结点类型（为空则默认为 OTHER 类型），另一种为指定结点值（自动设为 OTHER 类型）。构造器内儿子、相邻结点被初始化为 NULL，同时调用词法分析类的静态方法 LineNumber 设置行号。

AddChild 方法为对象加入指定的儿子节点，此结点将自动加在儿子节点数组的后部。

ChildNum 返回儿子节点的数目。

SiblingNum 返回指定儿子节点所含相邻结点链链长。

GetQuadTemp 方法在 m_sQuadTemp 成员为""时，即没有将其赋予临时变量时，将返回其值 m_sValue。

InsertSiblings 方法可以在对象指定的儿子结点上插入相邻结点，若儿子结点本身身为 NULL，则插入的结点直接作为儿子结点。

调试用方法：

当宏定义 DEBUG_TEST_TREE 时，TestTree 和 PrintTab 方法可用。TestTree 方法利用深度优先遍历访问指定头结点的 AST，并且通过控制 PrintTab 输出树结构至 TestTree.txt 中，方便调试的进行。

示例如下（不完整）：

```
<PROGRAM>
Child:
        <CONSTDEC>
        <VARDEC>
        Child:
                <VARDEF>
                Child:
                        [int]
                        [k]
                        [100]
                Child End -- VARDEF
        Child End -- VARDEC
        <FUNCDEF>
        Child:
                [void]
                [QuickSort]
                <PARATABLE>
                Child:
                        [int]
                        Sibling:
                                [int]
                        Sibling end -- OTHER int
                        [s]
                        Sibling:
                                [t]
                        Sibling end -- IDENTIFIER s
                Child End -- PARATABLE
                <COMPSTMT>
```

调用关系：

本类只调用了 CLexicalAnalysis 类中的方法，并且类本身属于递归定义。

3) 符号表项类 CSymbolTableItem

成员：

m_sName 为名称。

m_eCatagory 为种类的枚举，分为 CAT_CONST, CAT_VAR, CAT_ARRAY, CAT_FUNC, CAT_PARAM, CAT_STRING 六种。

m_sType 为类型。

m_nOffset 为相对地址。

m_sConstValue 为常量值。

m_nFuncParamNum 为函数中显式参数的个数。

m_nArraySize 为数组大小。

m_bLeafFunc 表示是否为叶子函数（不再调用其他函数的函数）。

sm_nStringIndex 记录字符串名称数。

方法：

CSymbolTableItem 构造器可以根据指定的 8 项表项构造对象。为空的部分有相应默认值。

GenStringIndex 静态方法可以生成一个字符串名称，前缀指定为 STRING_PREFIX（这里设定为"_str"）。

isString 结合 STRING_PREFIX 前缀判断给定字符串是否为“字符串名称”。

调试方法：

当宏定义 DEBUG_SYMBOL_TABLE 时，PrintSymbolTableItem 方法可用，输出表内 8 项内容。

调用关系：

本类不调用其他任何类的任何方法。

4) 函数附表项类 CFuncIndex

成员：

m_sFuncName 为函数名。

m_nIndex 记录此函数在符号表中的下标。

m_vTempVarTable 通过符号表表项对象指针的容器，记录此函数中所用的所有临时变量。

方法：

GetItemPtr 遍历 vector 返回具有给定名字的临时变量表项指针。

GetOffsetInTempVarTable 遍历 vector 返回具有给定名字的临时变量的偏移（相对地址）。

isInTempVarTable 遍历 vector 返回具有给定名字的临时变量是否在 vector 内。

PlusIndex 使得对象的 m_nIndex 加一。

调用关系：

本类使用了 CSymbolTableItem 类进行成员的定义以及方法的实现。

5) 错误处理类 CError

成员：

sm_gsErrorInfo 内部记录了所有错误的提示信息。

sm_nErrorSum 为错误数量累加值。

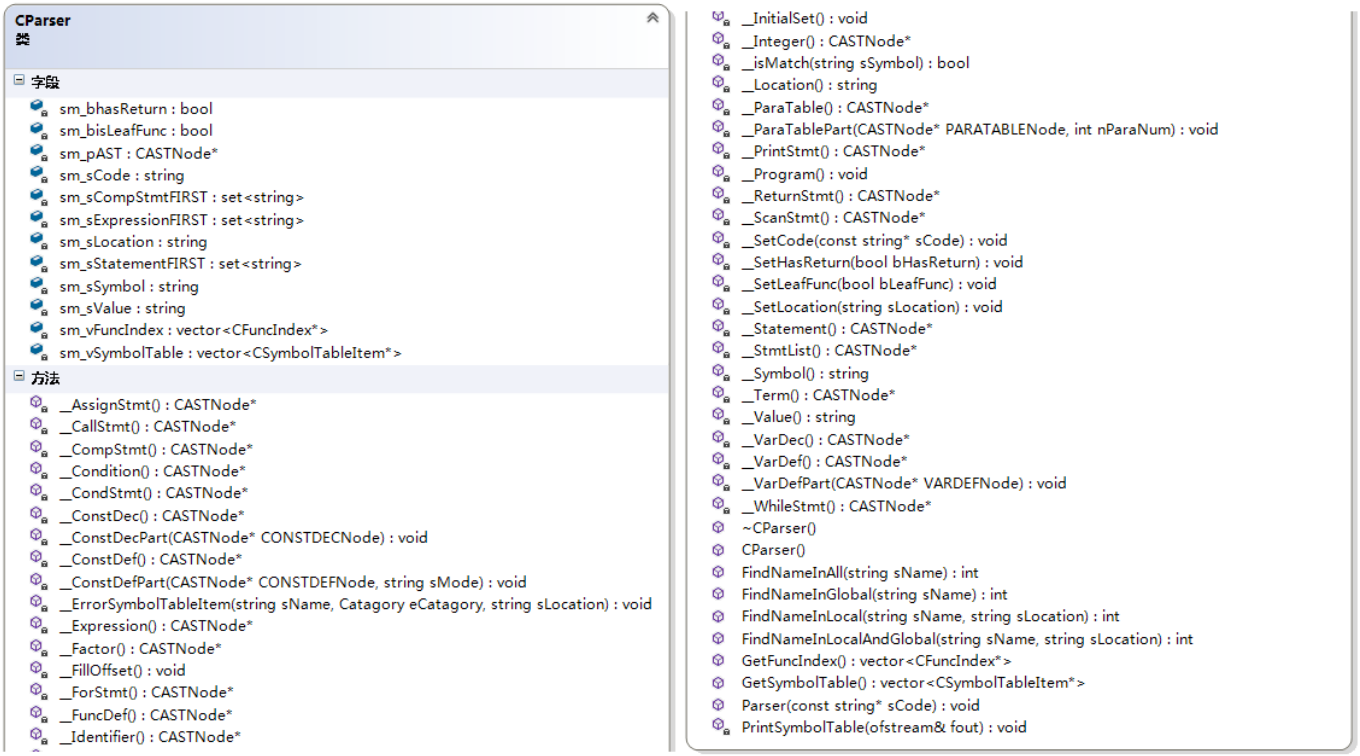
方法：

PrintErrorMsg 方法可以根据指定的错误编号打印错误信息。其中调用了语法分析类的获取行号、列号的方法，参数 sExtra 会在错误信息之后一并显示，可以输出跟错误现场相关的一些信息。

调用关系：

本类不调用其他任何类的任何方法。

6) 语法分析类 CParser



成员：

sm_sCode 存储了源代码，作为传给词法分析类 GetSymbol 方法的参数。

sm_sValue 和 sm_sSymbol 皆为词法分析返回的结果。

sm_sLocation 记录目前分析位置属于何函数。

sm_bhasReturn 表示到目前进度，该函数是否出现了返回语句。

sm_bisLeafFunc 表示当前函数中是否调用了其他函数。

sm_pAST 为 AST 头结点。

sm_vSymbolTable 为符号表，用 vector 储存符号表表项对象指针。

sm_vFuncIndex 为函数附表，用 vector 储存函数附表表项指针。

sm_sStatementFIRST、sm_sCompStmtFIRST 和 sm_sExpressionFIRST 分别为<Statement>、<CompStmt>和<Expression>的 FIRST 集，用于错误处理停止集的构建。

方法：

__ErrorSymbolTableItem 方法为出现未定义标识符时，为了放置重复报错，建立以未定义标识符名为名称的错误表项并加入符号表中。

__FillOffset 为符号表中所有项填写偏移量，包括函数附表中的临时变量。

__InitialSet 为 sm_sStatementFIRST、sm_sCompStmtFIRST 和 sm_sExpressionFIRST 加入相应的 token 类型。

__isMatch 判断当前 sm_sSymbol 是否为给定的 token 类型。

递归子程序分析关键方法：

非终结符	相关子程序
<Program>	__Program
<ConstDec>	__ConstDec、__ConstDecPart
<ConstDef>	__ConstDef、__ConstDefPart
<VarDec>	__VarDec
<VarDef>	__VarDef、__VarDefPart
<FuncDef>	__FuncDef
<ParaTable>	__ParaTable、__ParaTablePart
<Main>	__Program
<CompStmt>	__CompStmt
<StmtList>	__StmtList
<Statement>	__Statement
<CondStmt>	__CondStmt
<Condition>	__Condition
<WhileStmt>	__WhileStmt
<ForStmt>	__ForStmt
<CallStmt>	__CallStmt
<AssignStmt>	__AssignStmt
<ScanStmt>	__ScanStmt
<PrintStmt>	__PrintStmt
<ReturnStmt>	__ReturnStmt
<Expression>	__Expression
<Term>	__Term
<Factor>	__Factor

<Identifier>	__Identifier
<Integer>	__Integer

每个子程序（除了__Program）都返回一个子 AST 的树根结点指针。子程序通过调用词法分析的 GetSymbol 方法获得一个 token，根据其类型和值判断进入对应的子程序（有时需要超前读 1 到 2 个单词）当读到相应语法成分时，便加入树的相应位置，标识符需要写入符号表相应位置。子程序结尾将会把下一个单词读入至静态成员中，以便之后的子程序使用。

若出现错误，则调用错误处理类打印错误信息并进行跳读，跳读停止集由不同错误类型决定。

符号表查询方法：

FindNameInAll 方法从符号表表头顺序查找给定名称，返回其第一次出现的下标，若不存在则返回-1。

FindNameInGlobal 方法在全局部分查找给定名称，返回其第一次出现的下标，若不存在则返回-1。查找时利用函数附表首函数的下标，控制全局部分的范围。

FindNameInLocal 方法在局部部分查找给定名称，返回其第一次出现的下标，若不存在则返回-1。查找时利用函数附表，通过指定函数名称确定局部的范围。

FindNameInLocalAndGlobal 为上两个方法的结合。先进行局部查找，再进行全局查找，都不存在则返回-1。

GetFuncIndex 返回给定函数在符号表中的下标。

语法分析关键方法：

Parser 方法通过__InitialSet 初始化 FIRST 集后，调用词法分析类的 GetSymbol 方法获取第一个单词并进入子程序__Program。通过递归下降子程序，逐渐建立起以成员 sm_pAST 为头结点的 AST。

若有错误出现，则打印错误并停止后续工作。若无错误，则调用 CTraverser 类对 AST 进行递归遍历生成四元式，并调用__FillOffset 完善符号表中地址部分。

整个 Parser 方法完成了从源程序到未优化的四元式、符号表、函数附表的等价转化。

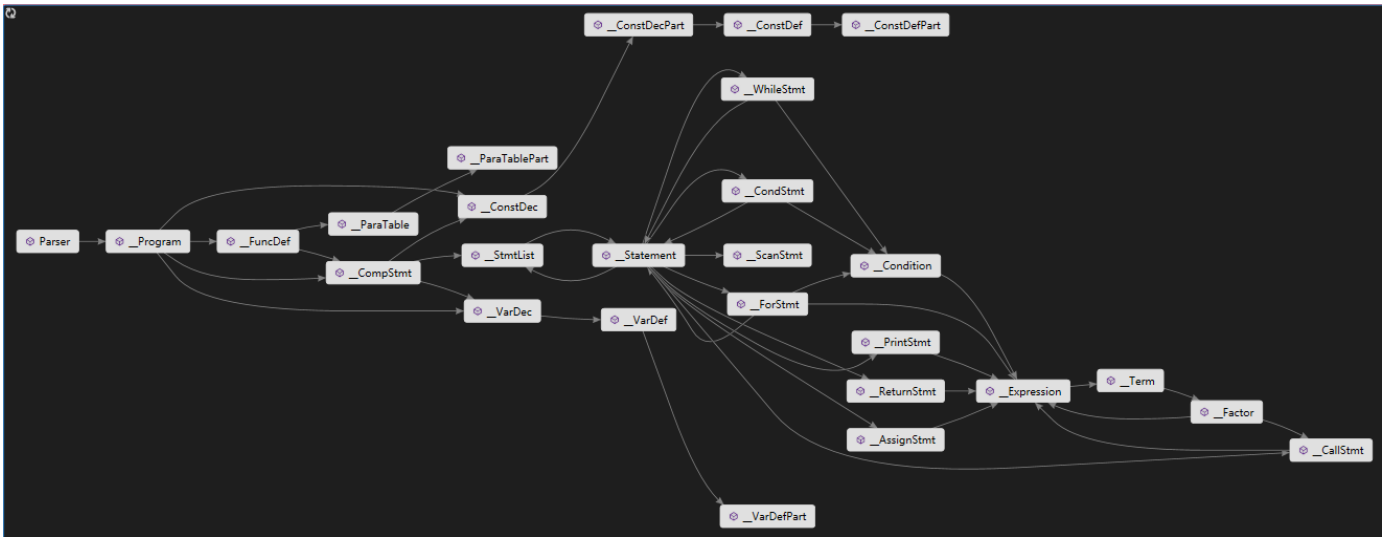
调试方法：

当宏定义 DEBUG_SYMOL_TABLE 时，PrintSymbolTable 打印表头并调用表项对象的 PrintSymbolTableItem 方法打印每一项的内容。

调用关系：

本类中，读取单词的方法需要 CLexicalAnalysis 类的参与；AST 的实现依赖于 CASTNode 类；语法分析时错误处理需要 CError 类；维护的符号表需要 CSymbolTableItem 类以及 CFuncIndex 类的参与；AST 的遍历涉及 CTraverser 类。

下图中很明显地显示了各子程序之间的调用关系。



7) 标签栈类 CLabelStack

成员：

m_gLabelStack 为标签栈，主要是语义分析生成跳转语句时遇到多个标签用到。

m_iStackTop 为栈顶指针。

方法：

方法皆为进出栈、判断空与满等基本方法，在此略过。

调用关系：

本类不调用其他任何类的任何方法。

8) 四元式类 CQuadruple

成员：

m_eQuadType 为四元式种类的枚举。

m_sOp 为四元式的运算符。

m_sArg1 为四元式的第一操作数。

m_sArg2 为四元式的第二操作数。

m_sResult 为四元式的结果。

m_nFuncIndex 为当前四元式所归属的函数在函数附表中的下标。

sm_gFuncTempDirty 数组代表某函数使用了哪些临时变量，此项为了计算 BeginFunc 时需要开辟空间的大小。

sm_gTempIndex 代表临时变量池内各变量的使用状态，有 TEMP_UNUSED 和 TEMP_USED 两种取值。

sm_iLabelIndex 为生成的标签数。

方法：

CQuadruple 构造方法只需提供四元式的四个参数即可，会根据运算符的不同自动为对象写入相应的四元式种类 m_eQuadType。

GenLabel 方法根据 sm_iLabelIndex 生成一个新标签。

GenTemp 方法根据 sm_gTempIndex 生成一个新临时变量。池中有空缺部分则将优先填补。

DropTemp 方法销毁一个临时变量。

isFunc、isLabel、isNum、isString、isTemp 分别判断一个字符串是否为函数标签名、标签名、数字、字符串（程序内的）、临时变量名。函数标签名、标签名、临时变量名的判断方式为根据相应的前缀。这里临时变量前缀 TEMP_PREFIX 设为”_tmp”，标签前缀 LABEL_PREFIX 设为”_L”，函数标签名前缀 FUNC_LABEL_PREFIX 设为”_”。

ResetFuncTempDirty 初始化函数使用的临时变量（全部为未使用）。

SumFuncTempDirty 求函数使用的临时变量总数。

TransQuad 可以将四元式转化为习惯书写形式下的字符串。

调试方法：

若宏定义 DEBUG_TEST_QUAD，则 PrintQuad 方法可用，将四元式对象输出习惯书写形式下的格式到文件流中。

调用关系：

本类只使用了 CSymbol 中宏定义 STRING_PREFIX，获取字符串变量前缀。

9) 遍历 AST 语义分析类 CTraverser

成员：

sm_iLabelStack 为标签栈。

sm_sLocation 为当前分析位置所属函数名。

sm_vQuads 容器存放产生的四元式对象指针。

方法：

__CountSiblings 返回下标为 1 的儿子节点的相邻结点链链长。

OptimizeLabels 将连续的标签名合并。

语义分析关键方法：

ASTTraversal 方法参数为 AST 某结点，作用为根据结点类型，进行遍历分析产生四元式。不同结点类型处理方法如下：

CONSTDEC 和 VARDEC 不产生任何四元式，因此不进行任何分析；

CONDSTMT、WHILESTMT、FORSTMT、FUNCDEF、CALLSTMT 将分别调用子方法__GenCondStmtQuads、__GenWhileStmtQuads、__GenForStmtQuads、__GenFuncDefQuads、__GenCallStmtQuads 进行分析，之后对其相邻结点调用 ASTTraversal 方法继续遍历。

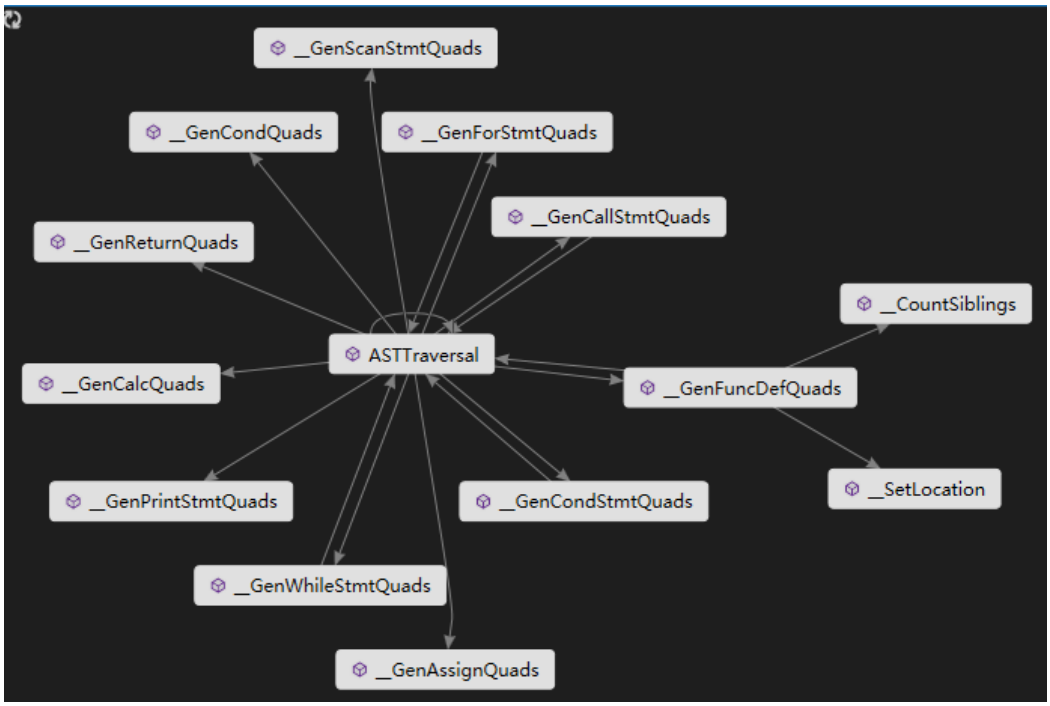
若非以上类型结点，则以深度优先遍历的原则，对其所有儿子结点依次进行访问。所有儿子结点访问结束之后，再进行本结点的类型判断：计算型、比较型、ASSIGNSTMT、RETURNSTMT、PRINTSTMT、SCANSTMT 分别调用子方法__GenCalcQuads、__GenCondQuads、__GenAssignQuads、__GenReturnQuads、__GenPrintStmtQuads、__GenScanStmtQuads 进行分析，这里如此做的原因是这些结点的儿子结点必须优先于本结点被访问，使用的是类似后续遍历的思想。之后对其相邻结点调用 ASTTraversal 方法继续遍历。

调试方法：

当宏定义 DEBUG_TEST_QUAD 时，PrintQuads 方法可用，作用为将 sm_vQuads 内所有的四元式调用对象自己的 PrintQuad 方法输出至文件流。

调用关系：

本类需要 CASTNode 类、CQuadruple 类以及 CParser 类的参与。可以明显地看出 ASTTraversal 方法对其他方法的调用，及其自身的递归。



10) 基本块类 CBasicBlock

注意：从这里起，优化相关成员与方法的使用需要宏定义 DEBUG_OPTIMIZE。

成员：

m_sBlockName 为基本块名。

m_sLabel 代表若基本块内第一条四元式为标签名，则其值为这个标签名。

m_vQuads 为此基本块内的四元式。

m_vSuccessor 为此基本块的后继。

m_sUseSet、m_sDefSet、m_vInSet、m_vOutSet 分别代表数据流分析中的 use 集、def 集、in 集与 out 集。

sm_nBlockNo 负责基本块的计数。

sm_vBasicBlocks 储存了一个函数内的所有基本块信息，即为流图。

方法：

__Calc 方法计算以 sOp 为运算符时，两个操作数的运算结果，用于常数合并优化。

__GenStartBlock、__GenEndBlock 分别用于产生起始块和终结块。

__ResetBlockNo 置 sm_nBlockNo 为 1，因为另一个函数的基本块要从 B1 开始。

CBasicBlock 构造器有两个重载：提供基本块名的（只用于产生开始和终结块）以及不提供基本块名的（由 sm_nBlockNo 自动生成）。

GenBasicBlocks 方法用于从四元式到整个源程序各函数流图的生成。

GenDAG 方法用于生成 DAG 图。

（从 DAG 图导出四元式的方法实现时遇到问题，因此未提供）。

__GenUseDefSet 生成 use 集和 def 集，__GenInOutSet 生成 in 集和 out 集。

以上涉及优化的方法，详细算法后述。

调用关系：

本类中涉及四元式，因此需要调用 CQuadruple 类；若需要优化，则需要 DAG.h 中与 DAG 有关的 3 个类参与。

11) DAG 图结点类 CDAGNode

成员：

m_pLChild、m_pRChild 分别代表左、右儿子结点。

m_vFather 代表父节点集合。

m_sOp 为结点代表的运算符，也用作结点标志。

m_nNumber 为结点号。

方法：

所有方法均为基本操作，在此略过。

调用关系：

本类自身为递归定义。

12) DAG 图结点表表项类 CDAGNodeTableItem

成员：

m_nNumber 为结点号。

m_sID 为结点所对应的变量。

方法：

皆为基本方法，在此省略。

调用关系：

本类不调用其他任何类的任何方法。

13) DAG 图类 CDAG

成员：

m_nNodeSum 为结点总数。

m_vDAG 为 DAG 图。

m_vNodeTable 为结点表。

方法：

FindDAGNode 返回 DAG 图中给定结点号的 DAG 结点。

FindInTable 在结点表中遍历查找给定名称的变量在结点表中的下标。

调用关系：

本类除了用到上述两个构建 DAG 图的必备类，还涉及了 CQuadruple 类。

14) 临时寄存器池单位类 CTempReg

成员：

m_pTempVar 为当前临时寄存器池中所存放的临时变量的符号表表项的指针。

m_nCounter 为计数器，辅助 LRU 算法决定使用的寄存器。

方法：

AddCounter 使 m_nCounter 加一。

调用关系：

本类只涉及到符号表，因此只与 CSymbolTableItem 类有关。

15) 冲突图类 CRIG

成员：

m_aRIG 用二维数组储存冲突图（RIG），并且使用对称矩阵。

m_vVars 对应二维数组各下标的变量名。

方法：

__GetIndex 获得指定变量名在 m_vVars 中的下标，也即 RIG 中的结点号。

__CalcDegree 计算某结点当前度数。

__SetEdge 为指定的两变量代表的结点之间生成一条无向边。

__GetNeighbor 返回与给定变量相连结点的结点号。

__isInVector 判断一个 string 变量是否在一个 vector<string>内。

__isRemoved 返回某变量代表的结点是否被从 RIG 中移走（不分配全局寄存器）。

__PopNode、__PushNode 代表将某变量出队、入队。

__RemoveNode 从 RIG 中移走给定变量代表的结点及其所有边。

__GenVars 从流图中获取待分配全局寄存器的所有变量。

__GenRIG 从流图根据活跃变量分析的结果生成冲突图。

__RIGColoring 应用图着色算法，返回全局寄存器分配结果。

GenGlobalVarAlloc 方法依次调用__GenVars、__GenRIG 和__RIGColoring，从流图中返回全局寄存器分配结果。

调用关系：

本类基于基本块的划分，也需要频繁访问符号表，因此涉及 CBasicBlock 类、CSymbolTableItem 类。

16) 全局寄存器分配单位类 CGlobalRegAlloc

成员：

m_pVariable 为要分配全局寄存器的变量名。

m_sRegister 为寄存器名。这里全局寄存器使用$s0、$s1 和$s2。

方法：

所有方法皆为基本方法，在此略过。

调用关系：

本类只涉及符号表，因此只与 CSymbolTableItem 有关。

17) 目标代码生成类 CCodeGeneration

成员：

sm_sLocation 为目前所分析的函数名。

sm_bHasTextInGlobal 代表全局部分生成目标代码时是否生成了 Text 段的内容。

sm_aTempVarPool 是临时寄存器池。这里其大小 TEMP_VAR_POOL_SIZE 为 5，即使用$t0~$t4 作为临时寄存器。

方法：

__itos 方法使整型转化为字符串类型。

__GenSegmentAnnotation 用于在 MIPS 汇编中产生段落注释。

__GenGlobalCode 用于处理全局常量、源程序中的字符串，他们将被定义在数据段上。同时根据全局变量的数目，在内存的全局指针指向部分分配空间。

目标代码生成关键方法：

GenCode 方法遍历整个基本块群中每个函数的每个基本块的每条四元式，根据其类型输出不同的 MIPS 汇编。

对于标签、函数标签，直接输出；

对于开始函数的标志，操作$sp，存储调用函数的$fp、$ra，设置新的$fp 并为被调用函数的局部变量和临时变量分配空间；

参数进栈、退栈需要操作$sp；

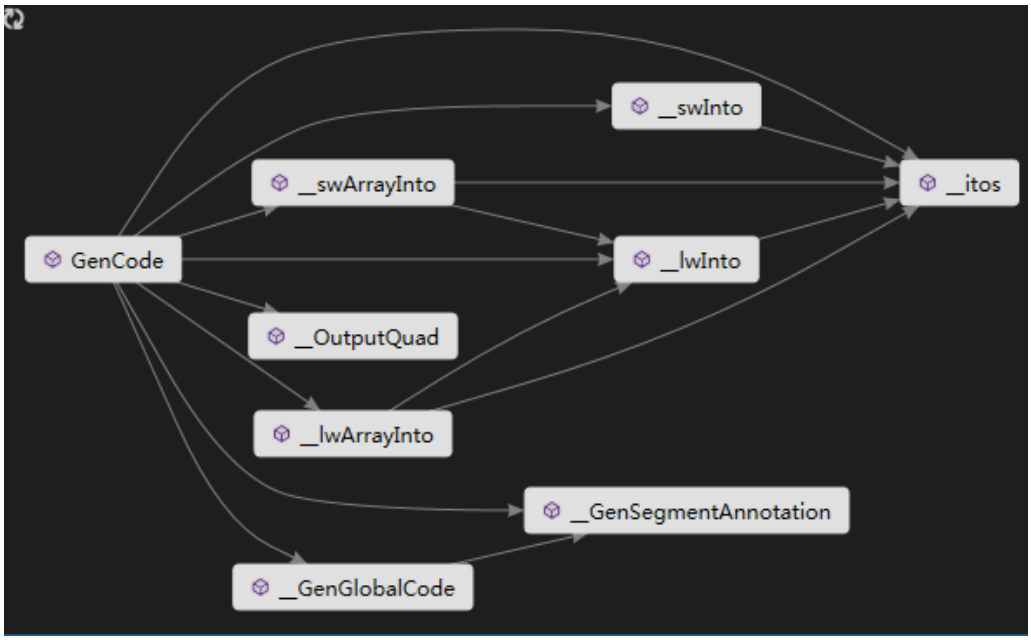
赋值语句先调用__lwInto 或__lwArrayInto 方法将赋值号右部内容从内存上读出。针对临时变量、经过全局寄存器分配的局部变量、不分配全局寄存器的局部变量、全局变量、常量，分别有不同的逻辑。之后调用__swInto 或__swArrayInto 方法写入赋值号左部，同样根据左部类型不同，会有不同的逻辑。

- 对于临时变量：读入时，先判断是否已在临时寄存器池内，若在，则不用产生 MIPS 汇编进行读取，直接返回所在的寄存器编号即可；若不在，需判断寄存器池是否有空位，若有则令该位置存放此临时变量，并返回这个寄存器编号；若无空位，则需根据 LRU(Least Recently Used, 近期最少使用)算法，找到临时寄存器池中计数器最大的一项，将这个寄存器中的临时变量存回相应位置后再把要求的临时变量读入。存入时，也要判断目标临时变量是否在临时寄存器池中。若在，直接用 move 指令即可；若不在，就按照该目标临时变量的偏移，sw 存入内存。
- 对于经过全局寄存器分配的局部变量：全局寄存器池由$s5~$s7 维护，因此需要生成查询全局寄存器池是否为目标变量的绝对地址的代码，若二者相同，即目前全局寄存器被目标局部变量所拥有，不需要 lw 指令；否则需要将该全局寄存器当前占据变量写回内存，再进行 lw 操作。存入操作同理。
- 对于不分配全局寄存器的局部变量、全局变量：一律使用$s3 进行读与存操作；
- 对于常量：直接用 li 指令取值放入$s3。
- 对于含数组的读与存，利用$t5 和$t6 计算要求下标的偏移进而计算目标变量的绝对地址，再生成

相应的 lw 与 sw 指令。

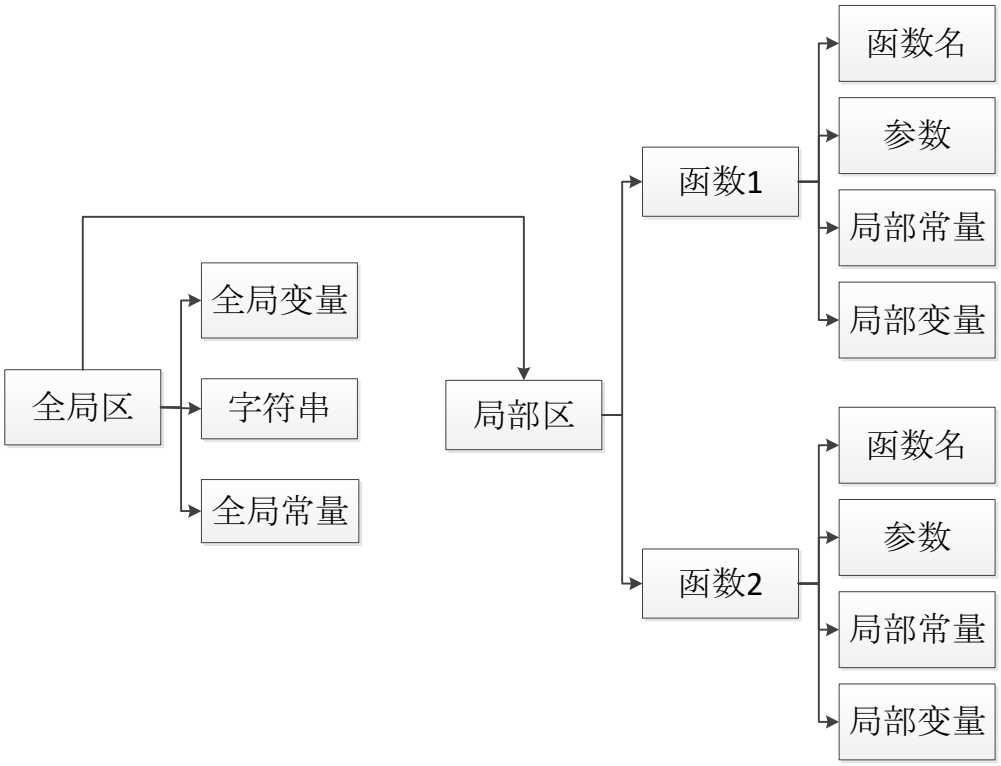
调用关系：

本类涉及四元式、符号表（查询方法在语法分析类中）、基本块，因此本类涉及与这些相关的所有类。



3. 符号表管理方案

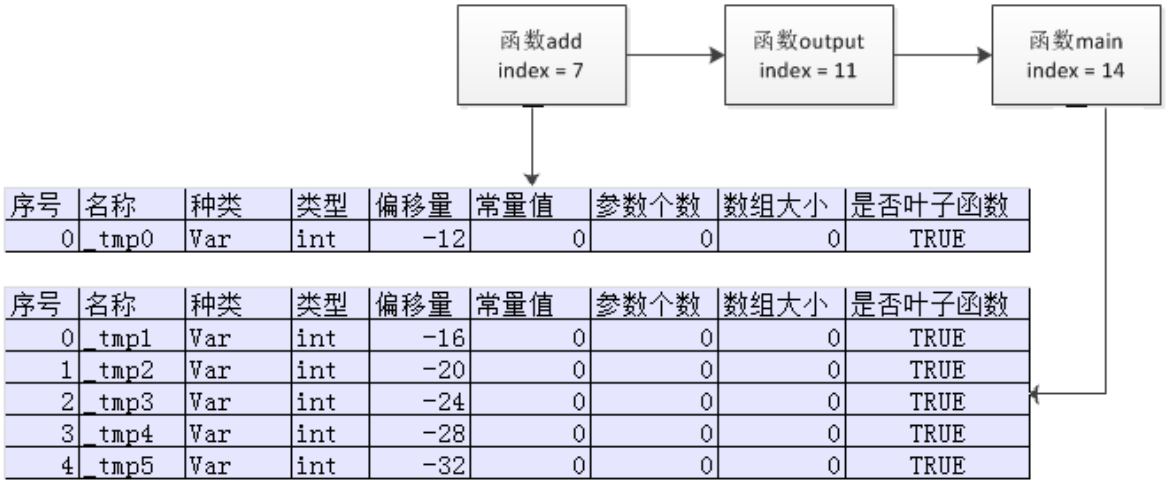
本编译器采用无序符号表。由上至下，符号表的组成为先全局区后局部区，全局区依次为全局常量、全局变量和字符串；局部区依次为各函数的子区域，每个子区域依次由函数名、参数、局部常量和局部变量组成。如图：



示例如下（对应程序见 AST 设计部分）：

序号	名称	种类	类型	偏移量	常量值	参数个数	数组大小	是否叶子函数
0	gc_x	Const	int	0	0	0	0	TRUE
1	gc_c1	Const	char	0	'a'	0	0	TRUE
2	gc_c2	Const	int	0	'z'	0	0	TRUE
3	g_x	Var	int	0	0	0	0	TRUE
4	g_y	Array	int	0	0	0	5	TRUE
5	_str0	String	int	0	=	0	0	TRUE
6	_str1	String	int	0	good!\n	0	0	TRUE
7	add	Func	int	0	0	2	0	TRUE
8	a	Param	int	4	0	0	0	TRUE
9	b	Param	int	8	0	0	0	TRUE
10	t	Var	int	-8	0	0	0	TRUE
11	output	Func	void	0	0	2	0	TRUE
12	n	Param	int	4	0	0	0	TRUE
13	c	Param	char	8	0	0	0	TRUE
14	main	Func	void	0	0	0	0	FALSE
15	i	Var	int	-8	0	0	0	TRUE
16	s	Var	int	-12	0	0	0	TRUE

对应还有函数附表，主要作用是记录临时变量信息，同时方便索引。示例如下：



序号	名称	种类	类型	偏移量	常量值	参数个数	数组大小	是否叶子函数
0	_tmp0	Var	int	-12	0	0	0	TRUE

序号	名称	种类	类型	偏移量	常量值	参数个数	数组大小	是否叶子函数
0	_tmp1	Var	int	-16	0	0	0	TRUE
1	_tmp2	Var	int	-20	0	0	0	TRUE
2	_tmp3	Var	int	-24	0	0	0	TRUE
3	_tmp4	Var	int	-28	0	0	0	TRUE
4	_tmp5	Var	int	-32	0	0	0	TRUE

建表的方法主要通过 CSymbolTableItem 的构造器结合若干 Set 方法进行，建立表项后插入到容器中即可。关于两个表中偏移量的确定，全部靠 CParser 类的__FillOffset 方法进行。该方法先填写主符号表，全局区的 Offset 由代码生成时 CCodeGeneration 类的__GenGlobalCode 方法生成并直接输出，不在符号表中填写；局部区的参数部分已在插入符号表时确定偏移量，常量与变量部分从-8 起依次减 4 填写，遇到数组则用数组大小乘 4 算出总共所需空间并分配相应大小。此方法第二部分为函数附表从四元式中提取属于这个函数的临时变量并插入表中，临时变量偏移量的起始值为这个函数在主表中局部变量偏移最小值再减 4。

查表主要通过 CParser 类内方法进行。其算法与功能在上一部分已述。

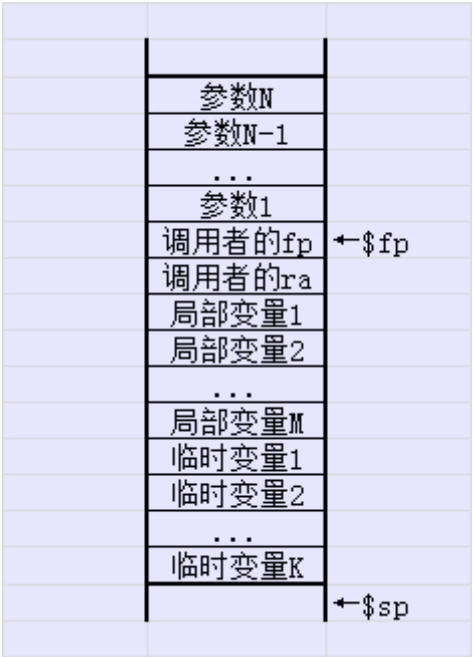
4. 存储分配方案

1) 寄存器功能

MIPS 提供 32 个 32 位寄存器，本编译器所产生的汇编代码对部分编译器的功能进行了限定。
$v0：用于存放函数返回值，以及用作 syscall 的参数；
$a0：用于存放”Print”系统调用的参数（要输出的数据的地址）；
$t0~$t4：用作临时寄存器；
$t5：用于暂时存放从数组读入的值、存放四则运算结果；
$t6：辅助运行时数组变量偏移的计算；
$t7：用于和全局寄存器值的比较；
$s0~$s2：用作全局寄存器；
$s3：不分配全局寄存器的局部变量等所存放的寄存器；
$s5~$s7：全局寄存器池；
$t8、$t9：用作四则运算、条件跳转两操作数的存放；
$gp：全局指针，用于存放全局变量；
$sp：栈顶指针（运行栈负增长）；
$fp：栈桢指针；
$ra：函数返回地址。

2) 运行栈设计

栈桢(frame)设计为如下图所示：（上部为高地址，下部为低地址）



调用一个函数前，调用者先将参数（1 至 N）倒序压栈，之后控制权转移到被调用的函数，先将$sp（栈顶指针）减 8，即开辟两个 int 空间，分别存放调用者的 fp 和 ra，以便函数结束时可以使运行栈恢复到调用前的状态。再根据函数局部变量和临时变量总数，$sp 减去 4 乘这个数目的大小，即为局部变量和临时变量开辟空间。此时，符号表内对应的参数和局部变量、临时变量的偏移即为相对$fp 的相对地址，只需用$fp 的值加上位移量，即可获得其在内存中的地址（这也是为什么首参数偏移起于 4，首局部变量偏移起于-8）。

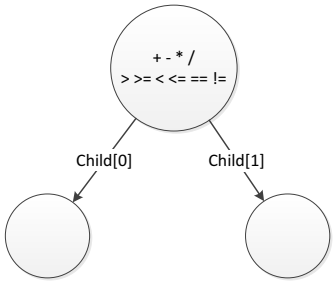
函数返回前，直接将$fp 赋给$sp，$fp-4 代表的地址上的值赋给$ra（恢复返回地址），$fp 代表的地址上的值赋给$fp（恢复栈桢指针），这样临时变量、局部变量全部退栈，$fp、$sp、$ra 也全部恢复至调用函数

前的状态，此时控制权转移回原调用者。调用者再将$sp 加回，使所有参数退栈。

5. 抽象语法树设计

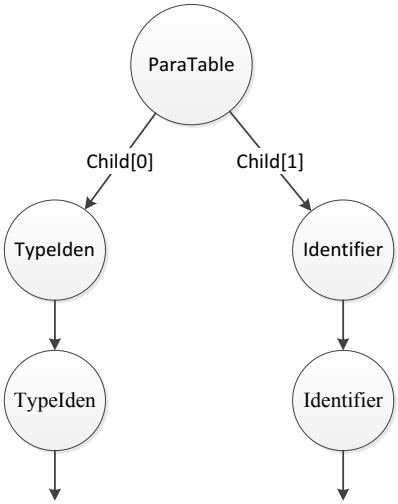
各类型结点 AST 示意图如下：

i. 表达式语句



ii. 参数表

<ParaTable> ::= <TypeIden><Identifier>{,<TypeIden><Identifier>}|<Null>

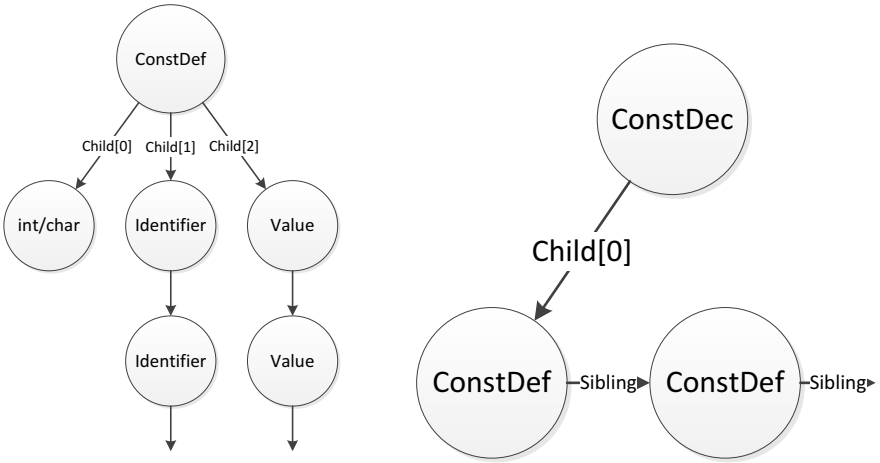


iii. 常量声明与定义

<ConstDec> ::= const<ConstDef>;{const<ConstDef>;}

<ConstDef> ::= int<Identifier>=<Integer>{,<Identifier>=<Integer>}|

char<Identifier>=<Character>{,<Identifier>=<Character>}

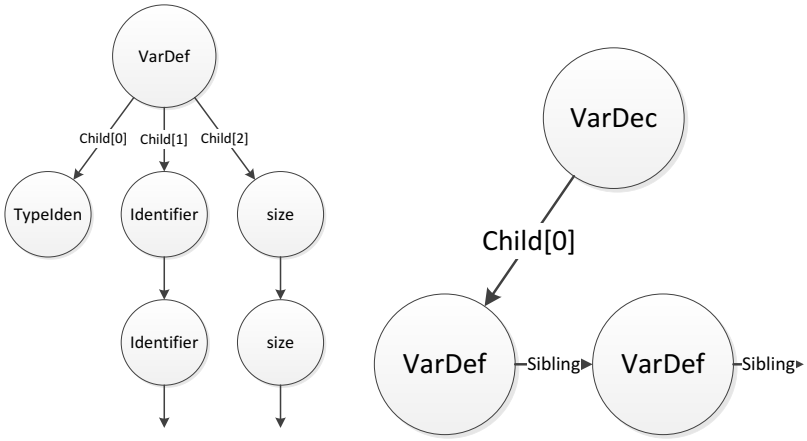


iv. 变量声明与定义

<VarDec> ::= <VarDef>;{<VarDef>;}

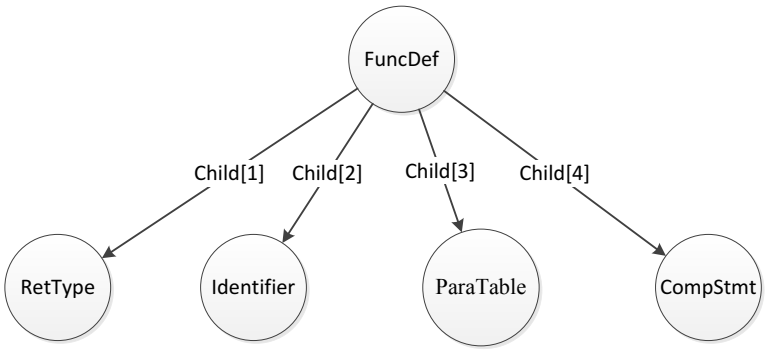
<VarDef> ::=

<TypeIden>(<Identifier>(’[‘<UnsignedInt>’]’|<Null>)){,(<Identifier>(’[‘<UnsignedInt>’]’|<Null>))}



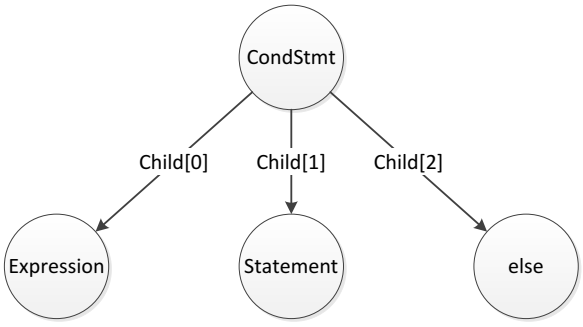
v. 函数定义

<FuncDef> ::= (<TypeIden>|void) <Identifier>’(‘<ParaTable>’)’’{‘<CompStmt>’}’



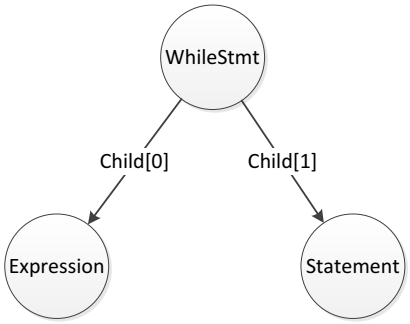
vi. if 语句

<CondStmt> ::= if’(‘<Condition>’)’<Statement>[else<Statement>]



vii. while 语句

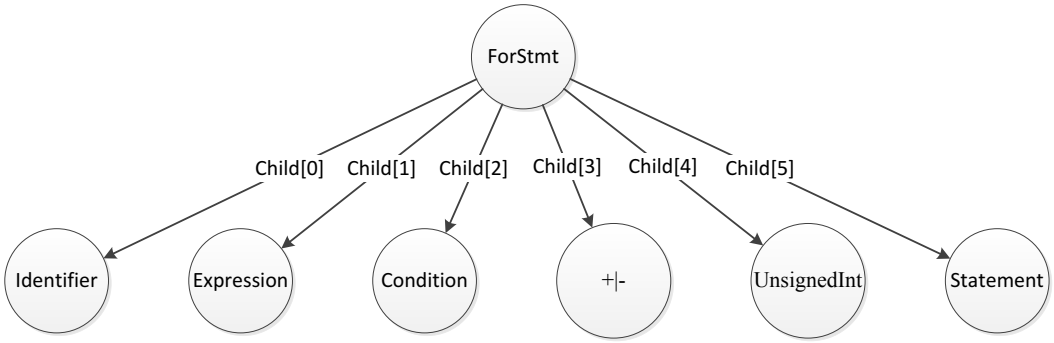
<WhileStmt> ::= while’(‘<Condition>’)’<Statement>



viii. for 语句

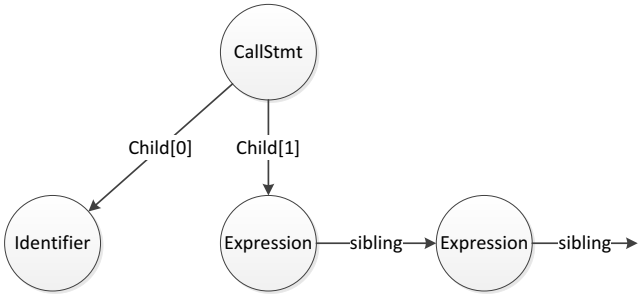
<ForStmt> ::=

for’(‘<Identifier>=<Expression>;<Condition>;<Identifier>=<Identifier>(+|-)<UnsignedInt>’)’<Statement>



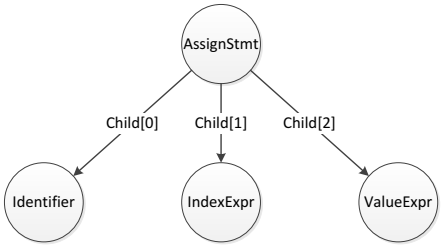
ix. 调用函数语句

<CallStmt> ::= <Identifier>’(‘(<Expression>{,<Expression>}|<Null>)’)’



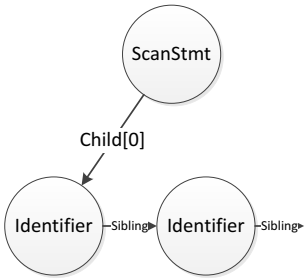
x. 赋值语句

<AssignStmt> ::= <Identifier>(=<Expression>|’[‘<Expression>’]’=<Expression>)



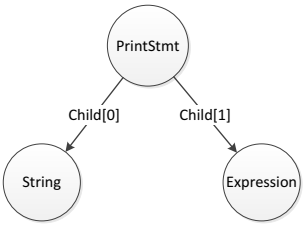
xi. 读语句

<ScanStmt> ::= scanf’(‘<Identifier>{,<Identifier>}’)’



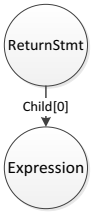
xii. 写语句

<PrintStmt> ::= printf’(‘(<String>,<Expression>|<String>|<Expression>)’)’



xiii. 返回语句

<ReturnStmt> ::= return[‘(‘<Expression>’)’]



完整示例：

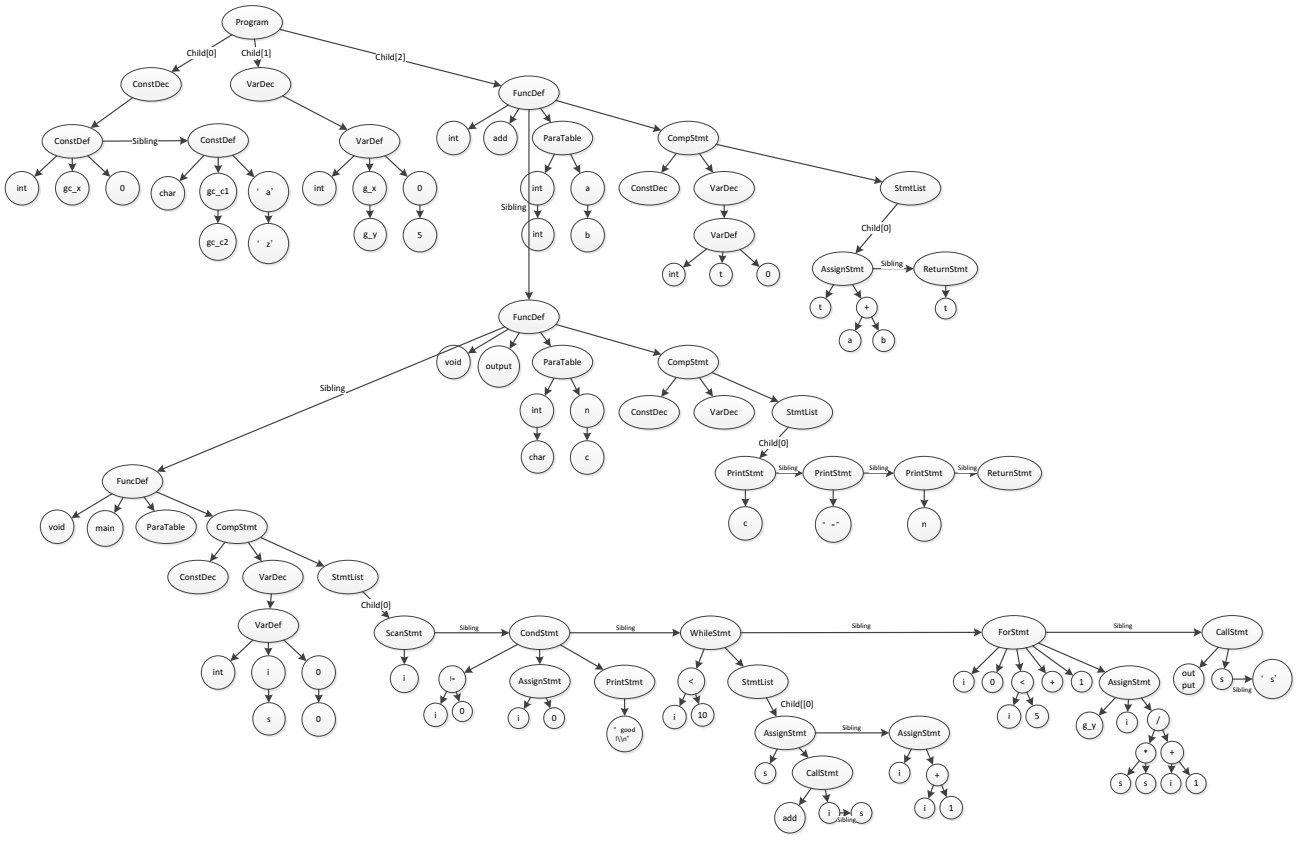
```
const int gc_x = 0;
const char gc_c1 = 'a', gc_c2 = 'z';
int g_x, g_y[5];

int add(int a, int b)
{
	int t;
	t = a + b;
	return (t);
}

void output(int n, char c)
{
	printf(c);
	printf(" = ");
	printf(n);
	return;
}

void main()
```

```
{
    int i, s;
    scanf(i);
    if (i != 0)
        i = 0;
    else
        printf("good!\n");
    while (i < 10){
        s = add(i, s);
        i = i + 1;
    }
    for (i = 0;i < 5;i = i + 1)
        g_y[i] = s * s    / (i + 1);
    output(s, 's');
}
```



6. 四元式设计

四元式标准格式为(Op, Arg1, Arg2, Result)。根据 Op，决定四元式的类型。

四元式分为如下 16 种枚举类型：

QUAD_TYPE_CALC, QUAD_TYPE_ASSIGN, QUAD_TYPE_JUMP, QUAD_TYPE_CONDJUMP, QUAD_TYPE_LABEL, QUAD_TYPE_FUNCLABEL, QUAD_TYPE_LCALL, QUAD_TYPE_SYSCALL, QUAD_TYPE_POP, QUAD_TYPE_PUSH, QUAD_TYPE_BEGINFUNC, QUAD_TYPE_ENDFUNC,

QUAD_TYPE_RETURN, QUAD_TYPE_PRINTPUSH, QUAD_TYPE_SCANPUSH, QUAD_TYPE_SCANPOP

1）QUAD_TYPE_CALC

示例：(+, a, b, t)
等价表示：t = a + b
注：Op 可以为’+’、’-‘、’*’、’/’，当且仅当 Op 为’-‘时，Arg2 可以为空。

2）QUAD_TYPE_ASSIGN

示例：(=, a, b, _)
等价表示：a = b
示例：([]=, b, 1, a)
等价表示：a[1] = b
示例：(=[], b, 1, a)
等价表示：a = b[1]

3）QUAD_TYPE_JUMP

示例：(j, _, _, Label)
表示无条件跳转。

4）QUAD_TYPE_CONDJUMP

示例：(j=, a, b, Label)
等价表示：If (a==b) Goto Label
注：Op 可以为 j=、j!=、j<、j<=、j>、j>=

5）QUAD_TYPE_LABEL

示例：(Label, _, _, _)

6）QUAD_TYPE_FUNCLABEL

示例：(Function, _, _, _)

7）QUAD_TYPE_LCALL

示例：(LCall, Function, _, _)
等价表示：Function(params)

8）QUAD_TYPE_SYSCALL

示例：(SysCall, _PrintInt, _, _)

9) QUAD_TYPE_POP

示例：(PopParams, 8, _, _)

10) QUAD_TYPE_PUSH

示例：(PushParam, a, _, _)

11) QUAD_TYPE_BEGINFUNC

示例：(BeginFunc, 16, _, _)

12) QUAD_TYPE_ENDFUNC

示例：(EndFunc, _, _, _)

13) QUAD_TYPE_RETURN

示例：(Return, _, _, _)或(Return, a, _, _)

14) QUAD_TYPE_PRINTPUSH

示例：(PrintPushParam, _str0, _, _)

15) QUAD_TYPE_SCANPUSH

示例：(ScanPushParam, a, _, _)

16) QUAD_TYPE_SCANPOP

示例：(ScanPopParam, a, _, _)

17) 综合示例

AST 所用程序生成未优化的四元式如下（已进行等价转换）：

```
_add:
    BeginFunc 8;
    _tmp0 = a + b;
    t = _tmp0;
    Return t;
    EndFunc;
_output:
    BeginFunc 0;
    PrintPushParam c;
    SysCall _PrintChar;
    PrintPushParam _str0;
```

```
        SysCall _PrintString;
        PrintPushParam n;
        SysCall _PrintInt;
        Return;
        EndFunc;
main:
        BeginFunc 28;
        SysCall _ScanInt;
        ScanPopParam i;
        If i = 0 Goto _L1;
        i = 0;
        Goto: _L0;
_L1:
        PrintPushParam _str1;
        SysCall _PrintString;
_L0:
        If i >= 10 Goto _L3;
        PushParam s;
        PushParam i;
        _tmp1 = LCall _add;
        PopParams 8;
        s = _tmp1;
        _tmp2 = i + 1;
        i = _tmp2;
        Goto: _L0;
_L3:
        i = 0;
_L4:
        If i >= 5 Goto _L5;
        _tmp3 = s * s;
        _tmp4 = i + 1;
        _tmp5 = _tmp3 / _tmp4;
        g_y[i] = _tmp5;
        _tmp3 = i + 1;
        i = _tmp3;
        Goto: _L4;
_L5:
        _tmp3 = 's';
        PushParam _tmp3;
        PushParam s;
        LCall _output;
        PopParams 8;
        EndFunc;
```

7. 优化方案

1) 划分基本块、生成流图

GenBasicBlocks 方法用于对所有四元式进行分析，生成各个函数对应的流图（vector<CBasicBlock*>类型），将这些流图再置于一个容器中，最终返回源代码对应的基本块群（vector<vector<CBasicBlock*>>类型）。

方法内部先对所有四元式进行扫描，将符合入口语句的四元式进行标记。入口语句分为 3 种：第一条四元式、标签、函数标签、进栈、退栈、函数开始、数组赋值四元式本身，以及无条件跳转、条件跳转、调用语句、系统调用、返回语句、函数结束的下一条四元式。产生入口语句后划分各基本块存储至相应数据结构中。

基本块如此细分的原因是方便 DAG 图的生成。为了保证生成 DAG 图前后某些功能性四元式的位置不会发生混乱。

之后是产生流图的过程。条件跳转语句会导致除了基本块之间首尾相接之外的另一通路；无条件跳转语句强制规定了直接后继块。只需要找到对这两种四元式为结尾的基本块进行特殊设置后继即可。

2) 生成 DAG 图、消除公共子表达式

这一部分通过使用书中算法，仅生成了无视数组运算的 DAG 图，由于有其他类型的四元式混在其中，导致 DAG 导出四元式有不小的麻烦，启发式算法中间还要涉及恢复的顺序，这里未能实现。

3) 活跃变量分析

先通过__GenUseDefSet 方法，对每个基本块的四元式进行分析，从中提取局部变量，设定 use 和 def 集。def 集有且仅有赋值语句的左侧符合；计算型、条件跳转型、赋值语句右值（这里一律不考虑数组变量）、进栈、返回语句中出现的局部变量可以进入 use 集。因为要求“首先被使用”以及“首先被定义”的要求，所以只有不在另一个集合中存在时，才可加入这一集合。

之后通过__GenInOutSet 方法，根据活跃变量分析对 in、out 集的定义进行循环计算。在这里设置了一个变量 bChanged 标志是否此次循环使得 in、out 集发生了改变（事实上只监测 out 集，因为 out 集的变动会影响到 in 集），只有不改变时才退出循环。

4) 生成冲突图(RIG)

通过__GenVars 和__GenRIG 方法共同进行。这里冲突图是依据每个基本块入口处活跃的变量两两之间不能共用全局寄存器。__GenVars 获取冲突图的结点、__GenRIG 根据每个 in 集建立冲突图的边。

5) 图着色算法

通过__RIGColoring 方法进行全局寄存器的分配。由于需要移走结点后再放回，因此将二维数组相应的值从 1 改为 0 会丢失边的信息。因此这里加入队列的操作是与这个结点有关的边全由 1 改为-1。而不分配全局寄存器的变量则由 1 改为 0。恢复时，再由-1 改回 1，根据__GetNeighbor 获得移回结点的相邻结点，他们之间不能共用全局寄存器，由此来确定移回的结点所分配的寄存器名称。

8. 出错处理

1) 错误处理方案

错误处理涵盖词法分析类与语法分析类（其中生成 AST 的过程涵盖部分语义分析的内容），若出现错误则程序在生成 AST 后即停止，即程序只能对完全符合规则的 AST 进行语义分析递归遍历。遇到错误后除了报出相应错误信息，还会根据错误现场情况进行单词的跳读。跳读的停止集一般为分号、右侧三种括号、逗号，或者后面非终结符的 FIRST 集

2) 错误详细信息

错误编号	错误类型	错误信息
0	语法错误	‘!’操作符无意义，是否希望使用’!=’？
1	语法错误	缺少字符后的单引号
2	语法错误	合法字符只能为四则运算符、数字或者字母（含下划线）
3	语法错误	合法字符串只能使用 ASCII 位于 32、33、35~126 之间的字符
4	语法错误	缺少字符串后的双引号
5	语法错误	缺少’)’
6	语法错误	缺少’]’
7	语法错误	未定义的标识符
8	语法错误	标识符不是函数名
9	语法错误	标识符非数组名
10	语法错误	缺少数组的下标
11	语义错误	void 函数不能返回值
12	语义错误	非 void 函数必须含有返回值
13	语法错误	缺少’(‘
14	语法错误	缺少”{‘
15	语法错误	缺少’}’
16	语法错误	程序应该结束于此
17	语法错误	缺少’;’
18	语义错误	重定义
19	语法错误	缺少’=’
20	语法错误	缺少’+’或’-‘
21	语法错误	程序必须包含 main 函数
22	语法错误	缺少”void”
23	语法错误	不完整的赋值语句
24	语法错误	数字 0 前不能有+或-
25	语义错误	scanf 语句中的标识符必须是简单变量
26	语义错误	表达式必须有可修改的左值
27	语义错误	表达式的右值必须是变量或常量
28	语义错误	需要正整数
29	语法错误	不合法的语句
30	语法错误	常量定义中缺少标识符
31	语法错误	不完整的常量定义

32	语法错误	不合法的常量初始值
33	语法错误	常量定义必须在变量定义之前
34	语法错误	不完整的变量定义
35	语义错误	数组长度必须大于 0
36	语法错误	不合法的数组下标
37	语法错误	main 函数必须为 void 类型
38	语法错误	main 函数不能拥有显式参数
39	语法错误	函数缺少名称
40	语义错误	函数必须有返回值
41	语法错误	缺少类型说明（int 或 char）
42	语法错误	缺少参数名
43	语义错误	scanf 语句中的标识符必须是简单变量
44	语法错误	缺少关系运算符
45	语法错误	缺少’,’
46	语义错误	此处标识符必须为简单变量
47	语义错误	显式参数数量与定义不符
48	语法错误	不完整的 for 语句
49	语法错误	不合法的表达式
50	语法错误	此处不能使用常量
51	语法错误	定义位置有误
52	语义错误	数组越界

三、 操作说明

1. 运行环境

打开 Visual Studio（推荐 2012 或以上版本），新建 C++的 Win32 控制台应用程序。在“项目”中选择“添加现有项”，将所有.cpp 和.h 文件导入，编译成功后生成可执行文件。

需配备 MARS (MIPS Assembler and Runtime Simulator)及 Java 环境。

2. 操作步骤

执行生成的可执行文件，提示”Please enter the path of the file:”，在控制台输入代码文件的绝对地址或者相对于工程文件的相对地址，回车即可。若所输入为非法路径，则会提示”Cannot find the file.”，请重新运行可执行文件并检查后重新输入。

若代码含错误，则会在控制台窗口显示相关错误信息。若完全正确，工程文件目录下会生成 MIPS 汇编代码 mips.asm，用 MARS 打开此文件后，按 F3 装配，成功后可按 F5 运行。

四、 测试报告

1. 正确测试程序及测试结果

1) 前文 AST 部分所用程序

```
const int gc_x = 0;
const char gc_c1 = 'a', gc_c2 = 'z';
int g_x, g_y[5];

int add(int a, int b)
{
	int t;
	t = a + b;
	return (t);
}

char fun()
{
	return ('a' + 5);
}

void empty() {}

void output(int n, char c)
{
	printf(c);
	printf(" = ");
	printf(n);
	return;
}

void main()
{
	int i, s;
	{}
	;;;
	empty();
	scanf(i);
	if (i != 0)
		i = 0;
	else
		printF("good!\n");
	while (i < 10){
		s = AdD(i, s);
```

```
            i = i + 1;
        }
        for (i = 0;i < 5;i = i + 1)
            g_y[i] = s * s    / (i + 1);
        output(s, 's');
        printf(fun());
}
```

运行结果：

```
**** user input : 0
good!
s = 45102
-- program is finished running --
```

2) 杨辉三角

```
const int min = 3, max = 34;
int a[150], b,CR,i;
void YHSJ(int CR)
{
        b = 3;
        a[1] = 1;
        a[2] = 1;
        printf(a[1]);
        printf("\n", a[1]);
        printf(" ", a[2]);
        printf("\n");
        while(b<=CR){
            for(i=b;i>=2;i=i-1)
                a[i]=a[i]+a[i-1];
            for(i=1;i<=b;i=i+1){
                printf(a[i]);
                printf(" ");
            }
            printf("\n");
            b = b + 1;
        }
}

void main()
{
        printf("Please enter a num between 3 and 34:\n");
        scanf(CR);
        if (CR>=min){
            if (CR<=max)
```

```
                YHSJ(CR);
        else
                printf("Too big!");
    }
    else
        printf("Too small!");
}
```

运行结果：

```
Please enter a num between 3 and 34:
**** user input : 8
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

-- program is finished running --
```

3) 负斐波那契数列正倒序输出

```
int fibo(int t)
{
    int output;
    if (t==1)
        return (-1);
    if (t==2)
        return (-1);
    else{
        output = fibo(t-1)+fibo(t-2);
        return (output);
    }
    return (0);
}

void main()
{
    int a, i, b[10], t;
    for (i=1; i<=10; i=i+1){
        t = fibo(i);
        printf("\n", t);
        b[i-1] = t;
```

```
    }
    printf("\nReverse:");
    for (i=10; i>=1; i=i-1)
        printf("\n", b[i-1]);
}
```

运行结果：

```
-1
-1
-2
-3
-5
-8
-13
-21
-34
-55
Reverse:
-55
-34
-21
-13
-8
-5
-3
-2
-1
-1
-- program is finished running --
```

4) 快速排序

```
int k[100];

void swap(int a, int b)
{
    int t;
    t = k[a];
    k[a] = k[b];
    k[b] = t;
}

void QuickSort(int s, int t)
{
```

```
    int i, j, Whileflag, WhileFlag2;
    Whileflag = 1;
    if (s<t){
        i = s;
        j = t + 1;
        while(Whileflag==1){
            i = i + 1;
            WhileFlag2 = 0;
            if (k[s]>k[i]){
                if (i!=t)
                    WhileFlag2 = 1;
            }
            while (WhileFlag2==1){
                i = i + 1;
                WhileFlag2 = 0;
                if (k[s]>k[i]){
                    if (i!=t)
                        WhileFlag2 = 1;
                }
            }
            j = j - 1;
            WhileFlag2 = 0;
            if (k[s]<k[j]){
                if (j!=s)
                    WhileFlag2 = 1;
            }
            while (WhileFlag2==1){
                j = j - 1;
                WhileFlag2 = 0;
                if (k[s]<k[j]){
                    if (j!=s)
                        WhileFlag2 = 1;
                }
            }
            if (i<j)
                swap(i, j);
            else
                Whileflag = 0;
        }
        swap(s, j);
        QuickSort(s, j - 1);
        QuickSort(j + 1, t);
    }
}
```

```
void main()
{
	int n, i, t;
	printf("Please input n(1<=n<=100):\n");
	scanf(n);
	if (n<1) return ;
	if (n>100) return ;
	for (i=0;i<n;i=i+1){
		printf("Please input k[", i);
		printf("]:\n");
		scanf(t);
		k[i] = t;
	}
	QuickSort(0, n - 1);
	for (i=0;i<n;i=i+1){
		printf("k[", i);
		printf("] = ", k[i]);
		printf("\n");
	}
}
```

运行结果：

```
Please input n(1<=n<=100):
**** user input : 6
Please input k[0]:
**** user input : 12
Please input k[1]:
**** user input : 0
Please input k[2]:
**** user input : -450
Please input k[3]:
**** user input : 12
Please input k[4]:
**** user input : 520
Please input k[5]:
**** user input : -2
k[0] = -450
k[1] = -2
k[2] = 0
k[3] = 12
k[4] = 12
k[5] = 520

-- program is finished running --
```

5) 求树的最小带权路径长度(Weighted Path Length of Tree, WPL)

```
int a[100];

void sort(int n)
{
	int i, j, temp;
	j = n - 1;
	while (j>=1){
		for (i=0;i<j;i=i+1)
			if (a[i]>a[i+1]){
				temp = a[i];
				a[i] = a[i+1];
				a[i+1] = temp;
			}
		j = j - 1;
	}
}

int CalcWPL(int n)
{
	int wpl, i, j;
	wpl = 0;
	for (i=n;i>=2;i=i-1){
		sort(i);
		a[0] = a[0] + a[1];
		wpl = wpl + a[0];
		for (j=2;j<i;j=j+1)
			a[j-1] = a[j];
	}
	return (wpl);
}

void main()
{
	int i, n, t;
	printf("Please enter num of nodes (1<=n<=100) : \n");
	scanf(n);
	printf("Please enter those nums:\n");
	for (i=0;i<n;i=i+1){
		scanf(t);
		a[i] = t;
	}
	printf("min WPL = ", CalcWPL(n));
```

```
}
```

运行结果：

```
Please enter num of nodes (1<=n<=100) :
**** user input : 5
Please enter those nums:
**** user input : 1
**** user input : 4
**** user input : 5
**** user input : 3
**** user input : 2
min WPL = 33
-- program is finished running --
```

2. 错误测试程序及错误信息

1) 前文 AST 所用程序改

```
const int gc_x = a;
const char gc_c1 = 'a, gc_c2 = 'z';
int g_x, g_y[5];
const int gc_z = 2;

int add(int a, int b)
{
	int t;
	t = a + b
}

void output(int n, char c)
{
	printf(a);
	printf(" = );
	printf(n);
}

void main()
{
	int i, s;
	scanf(i);
	if (i != 0
		i = 0;
	else
		printf("good!\n");
	while (i <> 10){
```

```
            s = add(i, s);
            i = i + 1;
        }
        for (i = 0;i < 5;i = i + 1)
            g_y[5] = s * s    / (i + 1);
        output(s, 's');
}
```

错误信息：

	Line	Col	No.	Error Info
1	1	19	32	invalid initial const value: a
2	2	22	1	missing single quotation after a char
3	4	6	33	const definition must be before var definition
4	10	2	17	missing ';'
5	14	11	7	undefined identifier: a
6	15	15	4	missing double quotation after a string
7	16	8	45	missing ','
8	24	4	5	missing ')'
9	27	13	49	invalid expression
10	32	9	52	array index out of bounds

Error num: 10

2) 杨辉三角改

```
const int min = 'n', max = -34;
int a[150], b,CR,i;
void YHSJ(int CR)
{
        b = ;
        a[-1] = 0;
        a[2] = -0;
        printf(a[1]);
        printf("\n", a[1]);
        printf(" ", a[2);
        printf("\n");
        while(b<=CR){
            for(i=b;i>=2;i=i-1)
                a[i]=a[i]+a[i-1];
            for(i=1;i<=b;i=i+1){
                printf(a[i]);
                printf(" ");
            }
            printf("\n");
            b = b + 1;
        }
```

```
}

void main()
{
    printf("Please enter a num between 3 and 34:\n");
    scanf CR);
    if (CR>=min){
        if (CR<=max)
            YHSJ();
        else
            printf("Too big!");
    }
    else
        printf("Too small!");
}
```

错误信息：

	Line	Col	No.	Error Info
1	1	20	32	invalid initial const value: n
2	5	7	49	invalid expression
3	6	10	52	array index out of bounds
4	10	18	6	missing ']'
5	27	13	13	missing '('
6	28	14	7	undefined identifier: min
7	30	11	47	mismatched number of parameters

Error num: 7

3) 斐波那契数列改

```
char fib(int t)
{
    int output;
    if (t==1)
        return (-1);
    if (t==2)
        return (-1);
    else{
        output = fibo(t-1)+fibo(t-2);
        return (output);
    }
    return (0);
}

main()
{
```

```
    const int s = 1;
    int a, i, b[10], t, s;
    for (i=1; i<=10; i++){
        s = fibo(i);
        printf("\n", t);
        b[b] = t;

    printf("\nReverse:");
    for (i=10; i>=1; i=i-1)
        printf(b[10], "\n");
}
```

错误信息：

	Line	Col	No.	Error Info
1	9	16	7	undefined identifier: fibo
2	9	16	17	missing ';'
3	9	16	7	undefined identifier: fibo
4	15	5	22	missing "void"
5	18	24	18	multiply defined symbol found: s in function "main"
6	19	21	48	incomplete for statement
7	20	6	20	missing '+' or '-'
8	20	6	28	a positive integer is required
9	22	7	10	missing index of array:
10	26	15	52	array index out of bounds
11	26	16	5	missing ')'

Error num: 11

4) 快速排序改

```
int k[100];

void QuickSort(int s, int t)
{
    int i, j, Whileflag, WhileFlag2;
    Whileflag = 1;
    if (s<t){
        i = s;
        j = t + 1;
        while(Whileflag==1){
            i = i + 1;
            WhileFlag2 = 0;
            if (k[s]>k[i]){
                if (i!=t)
                    WhileFlag2 = 1;
            }
```

```
                while (WhileFlag2==1){
                    i = i + 1;
                    WhileFlag2 = 0;
                    if (k[s]>k[i]){
                        if (i!=t)
                            WhileFlag2 = 1;
                    }
                }
                j = j - 1;
                WhileFlag2 = 0;
                if (k[s]<k[j]){
                    if (j!=s)
                        WhileFlag2 = 1;
                }
                while (WhileFlag2==1){
                    j = j - 1;
                    WhileFlag2 = 0;
                    if (k[s]<k[j]){
                        if (j!=s)
                            WhileFlag2 = 1;
                    }
                }
                if (i<j)
                    swap(i, j);
                else
                    Whileflag = 0;
            }
            swap(s, j);
            QuickSort(s, j - 1);
            QuickSort(j + 1, t);
        }
}

void main()
{
    int n, i, t;
    printf("Please input n(1<=n<=100):\n");
    scanf(n, t);
    if (n<1) return ;
    int m;
    if (n>100) return (2);
    for (i=0;i<n;i=i+1){
        printf("Please input k[", i);
        printf("]:\n");
```

```
            scanf(t);
            k[i] = t;
        }
        QuickSort(0, n - 1) = 2;
        for (i=0;i<n;i=i+1){
            printf("k[", i);
            printf("] = ", k[i]);
            printf("\n");
        }
}
```

错误信息：

	Line	Col	No.	Error Info
1	40	21	7	undefined identifier: swap
2	44	13	7	undefined identifier: swap
3	56	5	51	definition is misplaced
4	57	24	11	void function cannot return a value
5	64	26	17	missing ';'
6	64	26	29	invalid statement

Error num: 6

5) 求 minWPL 改

```
const int c;
int a[100];

void sort(int n)
{
    int i, j, temp;
    j = n - 1;
    while (j>=1){
        for (i=0;i<j;i=i+1)
            if (a[i]>a[i+1]){
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }
        j = j - 1;
    }
}

int CalcWPL(int n)
{
    int wpl, i, j;
    const int m = 2;
```

```
    wpl = 0;
    for (i=n;i>=2;i=i-1){
        sort(i);
        a[0] = a[0] + a[1];
        wpl = wpl + a[0];
        for (j=2;j<i;j=j+1)
            a[j-1] = a[j];
    }
    return wpl;
}

void main()
{
    int i, n, t;
    printf("Please enter num of nodes (1<=n<=100) : \n");
    scanf(n);
    n[2] = 5;
    printf("Please enter those nums:\n");
    for (0=i;i<n;i=i+1){
        scanf(t);
        a[i] = t(i);
    }
    printf("啊 min WPL = " CalcWPL(n));
}
}
```

错误信息：

	Line	Col	No.	Error Info
1	1	13	31	incomplete in const definition
2	22	7	33	const definition must be before var definition
3	31	12	12	non-void function must return a value
4	31	12	17	missing ';'
5	31	12	29	invalid statement
6	39	4	9	the identifier is not an array: n
7	41	8	48	incomplete for statement
8	43	11	9	the identifier is not an array: t
9	43	11	17	missing ';'
10	43	11	8	the identifier is not a name of function: t
11	45	10	3	string must use characters among ASCII 32,33,35-126
12	45	11	3	string must use characters among ASCII 32,33,35-126
13	45	12	3	string must use characters among ASCII 32,33,35-126
14	45	32	45	missing ','
15	45	36	17	missing ';'
16	45	36	29	invalid statement
17	47	2	16	program should end

Error num: 17

3. 测试结果分析

5 个正确的测试程序已几乎覆盖全部语法成分。

五、 总结感想

历时 2 个多月完成了个人目前所写的最大规模、最完整的一个系统，在这其中我收获了许多。

首先，我体会了编译的完整过程，知道 7 个逻辑部分各自的实现方式、作用。书本上的知识终究是理论，把所学的东西用在实验中才能真正地融会贯通。书本上的部分知识与实验要求有差异，这就需要我们能举一反三，学会变通；同时利用各种手段去查相关资料，在动手写一个编译器时不断地巩固已有知识，吸收新的知识。

其次，在上学期学完 C++课后，这是我第一个用 C++写的完整的程序。在选定代码种类之前我犹豫了一阵，但是因为这是一次绝佳的锻炼使用面向对象编程语言的机会，而且在做规划时，脑中自然而然产生了面向对象的想法，因此就选定了用 C++实现。因为这是一个庞大的系统，所以我希望尽可能每步做到标准化，比如严格保持代码格式、使用匈牙利命名法。虽然不一定完全正确，但这次系统化的尝试让我有了经验上的积累。确实，在书写代码、调试时，面对标准化的代码更能提高工作效率。

编译器的调试让我变得能静下心来分析问题、锁定根源、寻求解法。在做优化的过程中，经常碰到运算结果不正确的情况，只能在 MARS 上一步一步观察运行栈、寄存器，根据出错的指令反推代码生成部分的错误，而真正的错误可能又出现在前端……总之每一步都熬了过来，能够得到一个较为完善的编译器还是很有成就感的。

综上所述，这次课程设计无论是在学术上，还是在意志品质上，都对我有了一定提升。相信这份经验能够化为今后可以为我所用的利器。