**《编译技术》课程设计**

**详细设计**

# 引言

## 编写目的

本详细设计文档旨在对整个c0编译器的设计做一说明，包括程序结构，各类和函数的功能描述和调用依赖关系，符号表结构，运行栈结构，出错信息编号及描述等，使开发者更加对编译器的设计有更清晰的认识。

## 参考文献

[1]Compilers Principles,Techniques, & Tools. V.Aho, S.Lam, Ravi Sethi, D.Ullman. 北京,机械工业出版社(第2版)

[2]编译原理及编译程序构造. 高仲仪, 金茂忠. 北京,北京航空航天大学出版社(第2版)

# 编译器结构

**词法分析器**

**语法语义分析器**

**中间代码**

**代码优化**

**代码生成器**

**符号表**

**X86汇编指令**

**源程序**

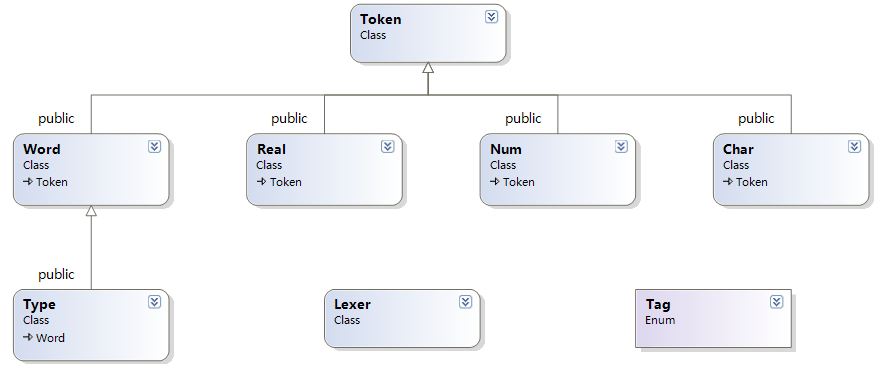
# 词法分析设计

## 1．功能

词法分析器读入组成源程序的字符流，并且将字符串形式的源程序分解成具有独立语法意义的单词符号。

## 2．程序结构

词法分析程序位于头文件lexical\_analyzer.h和源文件lexical\_analyzer.cpp中，类图如下：



## 3．C0文法关键字

const, int, float, char, void, if, else, while, for, return, main, printf, scanf

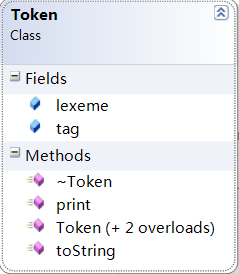
## 4．输出形式

|  |  |  |
| --- | --- | --- |
| 单词类别 | 单词值 | 枚举常量 |
| const | 1 | CONST |
| int | 2 | INT |
| float | 3 | FLOAT |
| char | 4 | CHAR |
| void | 5 | VOID |
| if | 6 | IF |
| else | 7 | ELSE |
| while | 8 | WHILE |
| for | 9 | FOR |
| return | 10 | RETURN |
| main | 11 | MAIN |
| printf | 12 | PRINTF |
| scanf | 13 | SCANF |
| 标识符 | 14 | ID |
| 字符 | 15 | CHARACTER |
| 字符串 | 16 | STRING |
| 整数 | 17 | INTEGER |
| 实数 | 18 | FLOAT |
| + | 19 | PLUS |
| - | 20 | MINUS |
| \* | 21 | STAR |
| / | 22 | DIVI |
| < | 23 | LT |
| > | 24 | GT |
| = | 25 | ASSIGN |
| ( | 26 | LPAR |
| ) | 27 | RPAR |
| { | 28 | LBRACE |
| } | 29 | RBRACE |
| ‘ | 30 | SQUOTE |
| “ | 31 | DQUOTE |
| . | 32 | DOT |
| , | 33 | COMMA |
| ; | 34 | SEMI |
| <= | 35 | LE |
| >= | 36 | GE |
| != | 37 | NEQU |
| == | 38 | EQU |
| 常量 | 39 | CONSTANT |
| 变量 | 40 | VAR |
| 参数 | 41 | PARA |
| 主函数 | 42 | MAINPROC |
| 无返回值函数 | 43 | VOIDPROC |
| 有返回值函数 | 44 | PROC |
| 取负 | 45 | NEG |
| 临时变量 | 46 | TEMP |

## 5．类和函数功能描述

### 5.1 Token类

**结构：**



**功能：**

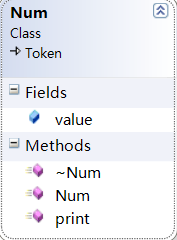
词法单元

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Token | 构造函数 |
| 2 | print | 输出 |
| 3 | toString | 转换为字符串 |

### 5.2 Num类

**结构：**



**功能：**

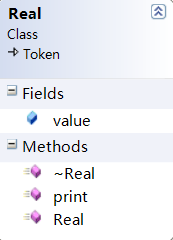
存放词法单元中的整数

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Num | 构造函数 |
| 2 | print | 输出 |

### 5.3 Real类

**结构：**

****

**功能：**

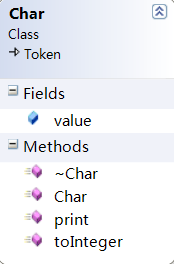
存放词法单元中的实数

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Real | 构造函数 |
| 2 | print | 输出 |

### 5.4 Char类

**结构：**

****

**功能：**

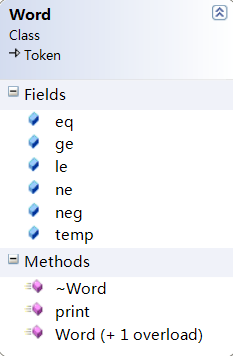
存放词法单元中的字符

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Real | 构造函数 |
| 2 | print | 输出 |
| 3 | toInteger | 转换为整型 |

### 5.5 Word类

**结构：**



**功能：**

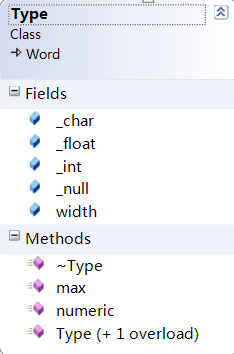
存放词法单元中的单词(包括关键字和标识符等)

**主要函数功能：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Word | 构造函数 |
| 2 | print | 输出 |

### 5.6 Type类

**结构：**

****

**功能：**

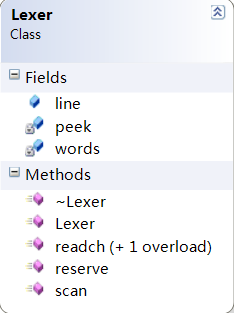
存放词法单元中的类型标识符

**主要函数描述：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Type | 构造函数 |
| 2 | max | 类型自动提升 |
| 3 | numeric | 判断是否为合法类型 |

### 5.7 Lexer类

**结构：**

****

**功能：**

词法分析器

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Lexer | 构造函数 |
| 2 | readch | 向前读一字符 |
| 3 | reserve | 保存关键字 |
| 4 | scan | 读取一词法单元并返回 |

# 语法、语义分析和中间代码生成设计

## 功能

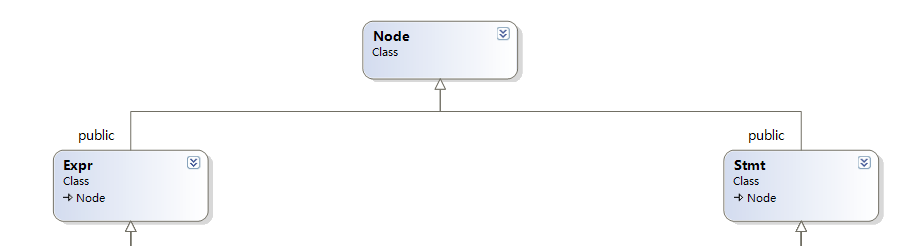
语法分析从词法分析程序取得单词串形式的源程序，根据文法分析并识别出各种语法成分，并进行语法正确性检查。

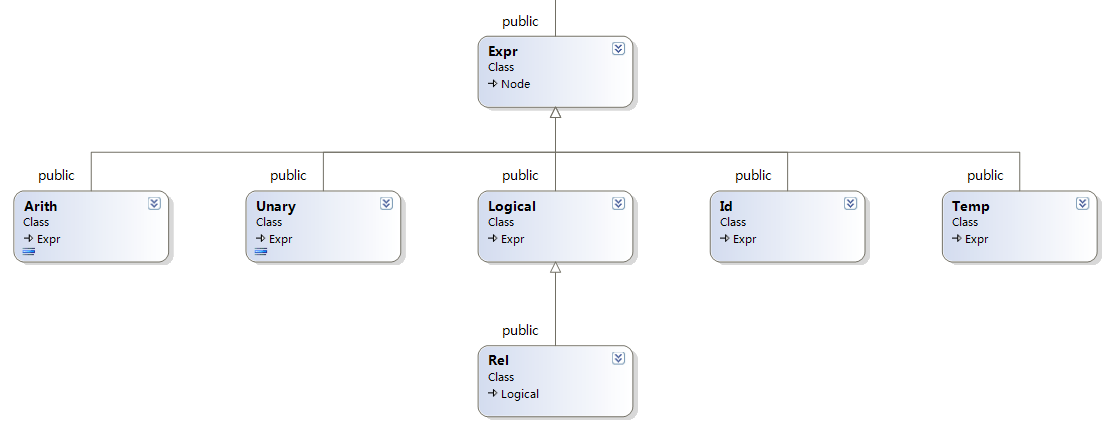
语义分析使用语法分析程序识别出的语法成分和符号表中的信息来检查源程序是否和文法定义的语义一致。

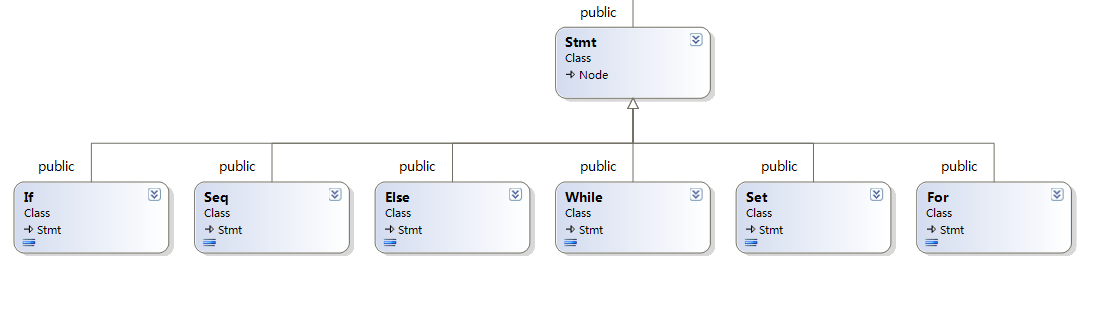
语法和语义分析程序相互结合，产生中间代码。

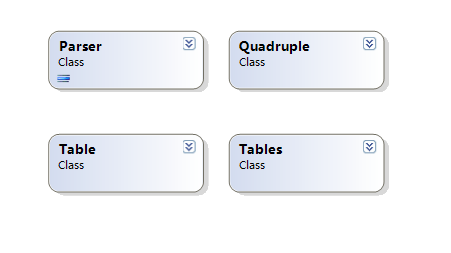
## 程序结构

词法分析程序位于头文件syntax\_parser.h和源文件syntax\_parser.cpp中，类图如下：









## 中间代码定义

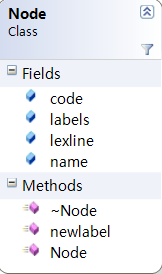
中间代码为四元式，数据结构和定义如下：

|  |  |
| --- | --- |
| 四元式 | 说明 |
| res = expr1 + expr2 | 两数相加 |
| res = expr1 – expr2 | 两数相减 |
| res = expr1 \* expr2 | 两数相乘 |
| res = expr1 / expr2 | 两数相除 |
| res = minus expr1 | 操作数取负 |
| label: | 标号 |
| goto jumpTo | 无条件跳转 |
| if expr1 > expr2 goto jumpTo | 大于则跳转 |
| if expr1 >= expr2 goto jumpTo | 大于或等于则跳转 |
| if expr1 < expr2 goto jumpTo | 小于则跳转 |
| if expr1 <= expr2 goto jumpTo | 小于或等于则跳转 |
| if expr1 == expr2 goto jumpTo | 等于则跳转 |
| if expr1 != expr2 goto jumpTo | 不等于则跳转 |
| res = expr1 | 赋值 |
| call expr1 | 无返回值函数调用 |
| call expr1 res | 有返回值函数调用 |
| param expr1 | 传参 |
| return expr1 | 函数返回值 |
| scanf expr1 | 输入 |
| printf expr1 | 输出字符串或表达式 |
| printf expr1 expr2 | 输出字符串和表达式 |

## 类和函数功能描述

### 4.1 Node类

**结构：**

****

**功能：**

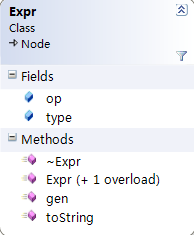
语法树节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Node | 构造函数 |
| 2 | newlabel | 产生新标号 |

### 4.2 Expr类

**结构：**



**功能：**

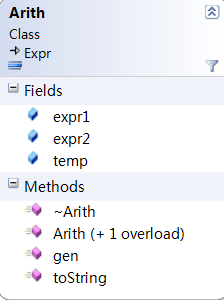
表达式语法节点

**主要函数功能：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Expr | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.3 Arith类

**结构：**

****

**功能：**

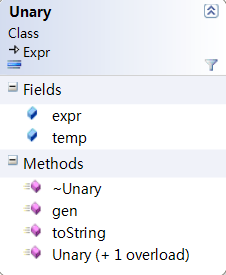
双目运算符语法节点，包括+、-、\*、/运算

**主要函数功能：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Arith | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.4 Unary类

**结构：**

****

**功能：**

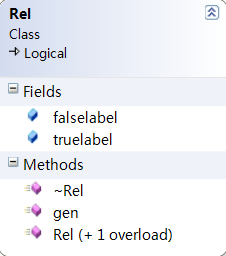
单目运算符语法节点，包括取负运算

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Unary | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.5 Rel类

**结构：**

****

**功能：**

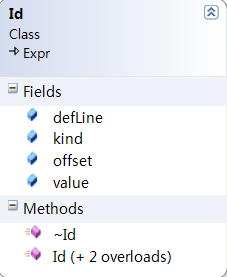
关系运算符语法节点，包括<、<=、>、>=、==、!=运算

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Rel | 构造函数 |
| 2 | gen | 产生四元式 |

### 4.6 Id类

**结构：**

****

**功能：**

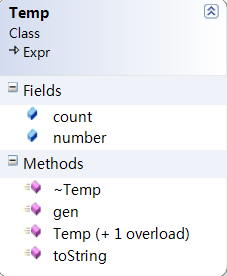
标识符语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Id | 构造函数 |

### 4.7 Temp类

**结构：**

****

**功能：**

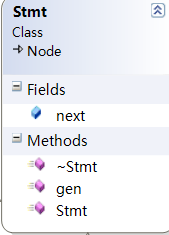
临时变量语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Temp | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.8 Stmt类

**结构：**

****

**功能：**

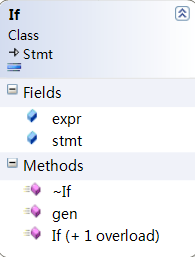
语句节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Stmt | 构造函数 |
| 2 | gen | 产生四元式 |

### 4.9 If类

**结构：**

****

**功能：**

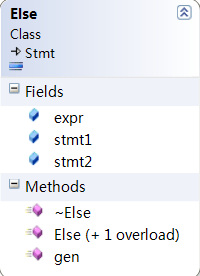
if语句语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | If | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.10 Else类

**结构：**

****

**功能：**

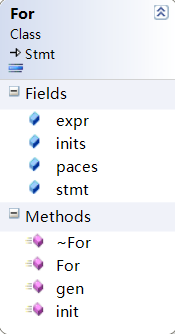
if-else语句语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Else | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.11 For类

**结构：**

****

**功能：**

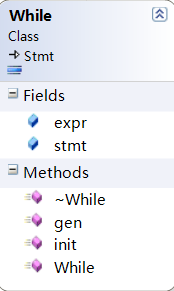
for语句语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | For | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.12 While类

**结构：**

****

**功能：**

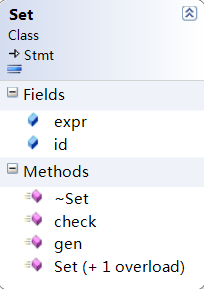
while语句语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | While | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.13 Set类

**结构：**

****

**功能：**

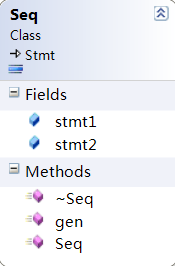
赋值语句语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Set | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.14 Seq类

**结构：**

****

**功能：**

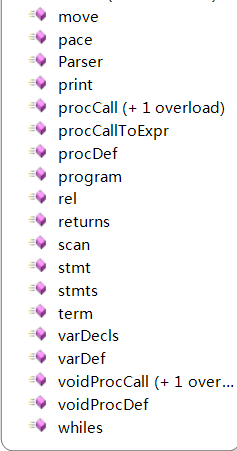
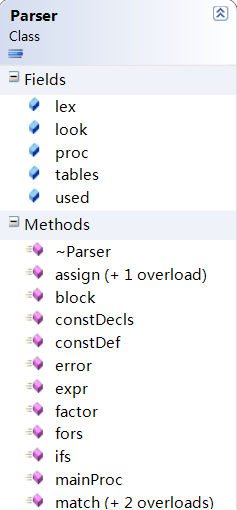
语句列语法节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Seq | 构造函数 |
| 2 | gen | 产生四元式 |
| 3 | toString | 转换为字符串 |

### 4.15 Parser类

**结构：**

****

**功能：**

语法分析器，同时进行语义分析，并产生中间代码

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Parser | 构造函数 |
| 2 | move | 向前读取词法单元 |
| 3 | match | 读取词法单元并与指定词法单元匹配 |
| 4 | program | 分析<程序> |
| 5 | mainProc | 分析<主函数> |
| 6 | block | 分析<复合语句> |
| 7 | stmts | 分析<语句列> |
| 8 | stmt | 分析<语句> |
| 9 | ifs | 分析<条件语句> |
| 10 | whiles | 分析<循环语句>中的while语句 |
| 11 | fors | 分析<循环语句>中的for语句 |
| 12 | assign | 分析<赋值语句> |
| 13 | scan | 分析<读语句> |
| 14 | print | 分析<写语句> |
| 15 | returns | 分析<返回语句> |
| 16 | procDef | 分析<有返回值函数定义部分> |
| 17 | voidProcDef | 分析<无返回值函数定义部分> |
| 18 | procCall | 分析<有返回值函数调用语句>，返回结果不作表达式 |
| 19 | procCallToExpr | 分析<有返回值函数调用语句>，返回结果作为表达式 |
| 20 | voidProcCall | 分析<无返回值函数调用语句> |
| 21 | rel | 分析<条件> |
| 22 | expr | 分析<表达式> |
| 23 | term | 分析<项> |
| 24 | factor | 分析<因子> |
| 25 | pace | 分析<步长> |
| 26 | varDecls | 分析<变量说明部分> |
| 27 | varDef | 分析<变量定义> |
| 28 | constDecls | 分析<常量说明部分> |
| 29 | constDef | 分析<常量定义> |

# 符号表设计

## 功能

记录在源程序中所使用的标识符并收集每个标识符的各种属性信息，如关于标识符的名称、种类、类型等，关于函数的名称、参数个数和类型、返回值类型等。

## 结构

全局符号表

函数符号表1

函数符号表n

**…**

**符号表字典**

全局变量

全局常量

参数

参数个数和类型

局部变量、常量

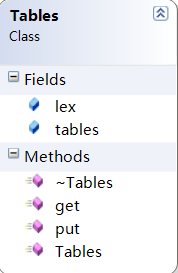
临时变量

返回值类型

## 类和函数功能描述

### 3.1 Tables类

**结构：**

****

**功能：**

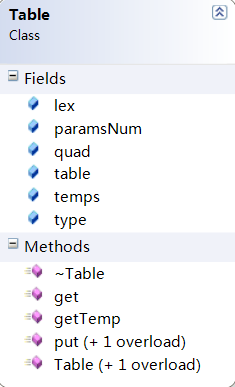
符号表字典，存放全局符号表和各函数符号表

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Tables | 构造函数 |
| 2 | get | 从符号表字典中获取符号表 |
| 3 | put | 向符号表字典中插入符号表 |

### 3.2 Table类

**结构：**

****

**功能：**

函数符号表

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Table | 构造函数 |
| 2 | get | 从符号表中取出变量、常量和参数 |
| 3 | getemp | 从符号表中取出临时变量 |
| 4 | put | 向符号表中插入变量、常量和参数 |
| 5 | putTemp | 向符号表中插入临时变量 |

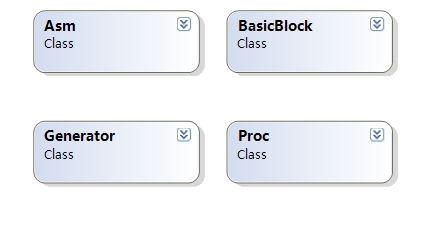
# 代码生成设计

## 功能

利用语法分析器产生的中间代码以及符号表信息，生成X86汇编码。

## 程序结构

代码生成程序位于头文件code\_generator.h和源文件code\_generator.cpp中，类图如下：



## 运行栈设计

运行栈如下图所示：

|  |
| --- |
| 临时变量区 |
| 局部变量区 |
| 返回地址 |
| 参数区 |

**函数声明和调用时运行栈的变化：**

通过esp寄存器移动栈顶指针，来达到申请和释放栈空间的目的。

每个函数定义时，都会先为该函数分配参数区，再为函数的中的临时变量和局部变量分配临时变量区和局部变量区。

在函数调用时，首先将所要传的参数压入被调用函数的参数区，然后执行该函数，该函数会从参数区中获得传入的参数，执行完函数后将返回值送入eax并返回。

## 寄存器分配

### 4.1 全局寄存器分配

通过数据流分析中的活跃变量分析，构建冲突图，并根据冲突图的图着色算法进行全局寄存器分配。但在图着色算法中选取不分配全局寄存器的节点时，不同的选取方法会对分配结果带来较大影响。

### 4.2 临时寄存器分配

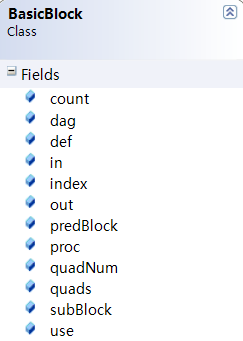
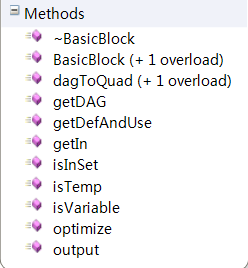
使用临时寄存器池对临时寄存器进行分配和管理：

1. 进入基本块时，清空临时寄存器池；
2. 为当前中间代码生成目标代码时，无论是临时变量还是局部变量，如需使用临时寄存器，都可以向临时寄存器池申请；
3. 临时寄存器池接到申请后，如寄存器池中有空闲寄存器，则可将该寄存器标识为该申请变量占用，并返回该空闲寄存器；
4. 临时寄存器池接到申请后，如寄存器池中没有空闲寄存器，则将在即将生成代码中不会被使用的寄存器写回相应的内存空间(根据活跃变量分析的结果)，标识该寄存器被新的变量占用，返回该空闲寄存器；
5. 在基本块结尾或函数调用前，将寄存器池中所有被占用的临时寄存器写回相应的内存空间，并清空临时寄存器池。

## 5．类和函数功能描述

### 5.1 BasicBlock类

**结构：**

** **

**功能：**

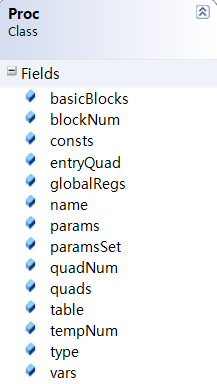
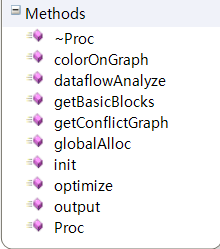
存放对函数划分得到的基本块和流图信息

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | BasicBlock | 构造函数 |
| 2 | isInSet | 判断元素是否在集合中 |
| 3 | isTemp | 判断是否为临时变量 |
| 4 | isVariable | 判断是否为局部变量 |
| 5 | getDAG | 创建DAG图并删除局部公共子表达式 |
| 6 | dagToQuad | 由DAG导出四元式 |
| 7 | getDefAndUse | 计算基本块的def集和use集 |
| 8 | getIn | 计算基本块的in集 |
| 9 | optimize | 基本块内优化 |
| 10 | output | 输出基本块信息 |

### 5.2 Proc类

**结构：**

** **

**功能：**

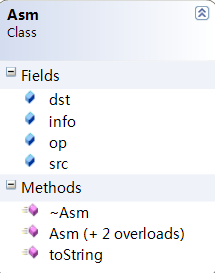
综合各函数的符号表、基本块、中间代码等信息

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Proc | 构造函数 |
| 2 | init | 初始化，获得函数符号表，划分基本块和流图 |
| 3 | getBasicBlocks | 划分基本块和流图 |
| 4 | dataflowAnalyze | 数据流分析 |
| 5 | globalAlloc | 全局寄存器分配 |
| 6 | getConflictGraph | 建立冲突图 |
| 7 | colorOnGraph | 冲突图着色算法 |
| 8 | optimize | 函数优化数据流分析-建立冲突图 |
| 9 | output | 输出函数信息 |

### 5.3 Asm类

**结构：**

****

**功能：**

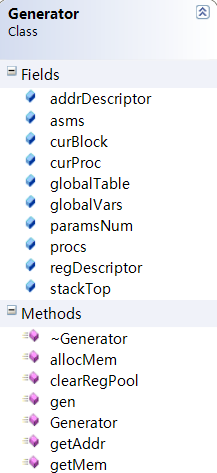
X86汇编指令

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Asm | 构造函数 |
| 2 | toString | 转换为字符串 |

### 5.4 Generator类

**结构：**

** **

**功能：**

代码生成器

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Generator | 构造函数 |
| 2 | init | 初始化，获取各函数信息和全局符号表信息 |
| 3 | gen | 生成X86汇编指令 |
| 4 | getReg | 为一条四元式分配寄存器 |
| 5 | getAddr | 获得变量的存储地址 |
| 6 | getMem | 获得变量在内存中的地址 |
| 7 | getype | 获得变量的类型 |
| 8 | isImmediate | 判断是否为立即数并进行类型转换 |
| 9 | isConst | 判断是否为局部常量 |
| 10 | isUse | 判断变量在当前基本块当前位置以后是否被使用 |
| 11 | isLive | 判断变量是否为活跃变量 |
| 12 | isInMem | 判断变量是否在内存中 |
| 13 | allocMem | 为变量分配地址 |
| 14 | updateRegDescriptor | 更新寄存器描述符 |
| 15 | updateAddrDescriptor | 更新地址描述符 |
| 16 | specifyReg | 为除法运算分配指定寄存器 |
| 17 | releaseRegPool | 释放不使用的寄存器 |
| 18 | cleaRegPool | 清空临时寄存器池 |
| 19 | saveLiveVar | 在基本块结尾处存储活跃变量 |
| 20 | saveRegVar | 调用函数前保存临时寄存器中的变量 |
| 21 | output | 输出X86汇编指令 |

# 代码优化设计

## 功能

通过冲突图进行全局寄存器分配以及通过DAG图进行基本块内公共子表达式删除来进行优化，以获得更高效的目标程序。

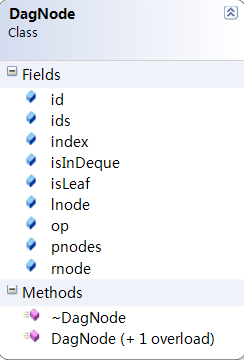
## 程序结构



## 3．类和函数功能描述

### 3.1 DagNode类

**结构：**

****

**功能：**

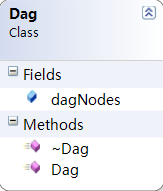
DAG图中的节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | DagNode | 构造函数 |

### 3.2 Dag类

**结构：**

****

**功能：**

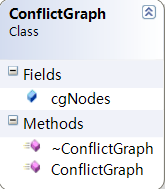
DAG图

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | Dag | 构造函数 |

### 3.3 ConflictGraphNode类

**结构：**

****

**功能：**

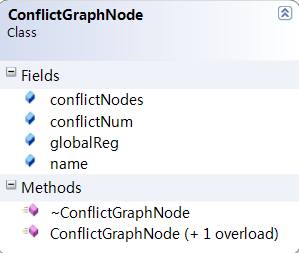
冲突图中的节点

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | ConflictGraphNode | 构造函数 |

### 3.4 ConflictGraph类

**结构：**

****

**功能：**

冲突图

**主要函数说明：**

|  |  |  |
| --- | --- | --- |
| 编号 | 函数名 | 函数说明 |
| 1 | ConflictGraph | 构造函数 |

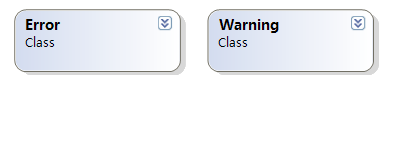
# 错误处理设计

## 功能

检测源程序中的语法错误并给出提示信息。

## 程序结构

错误处理程序位于头文件error.h和源文件error.cpp中，类图如下：



## 错误列表

### 3.1 错误信息列表

|  |  |  |
| --- | --- | --- |
| 编号 | 错误说明 | 提示语句 |
| 0 | 非法字符 | illegal character. |
| 1 | 缺少'\'' | missing '\''. |
| 2 | 缺少'\"' | missing '\"'. |
| 3 | 缺少'(' | missing '('. |
| 4 | 缺少')' | missing ')'. |
| 5 | 缺少',' | missing ','. |
| 6 | 缺少';' | missing ';'. |
| 7 | 缺少'{' | missing '{'. |
| 8 | 缺少'}' | missing '}'. |
| 9 | 非法符号 | illegal token. |
| 10 | 标识符未定义 | identifier has not been definited. |
| 11 | 标识符重复定义 | identifier has already been definited. |
| 12 | 应是标识符 | missing identifier. |
| 13 | 应是类型标识符 | missing type. |
| 14 | 函数未定义 | function has not been definited. |
| 15 | 函数重复定义 | function has already been definited. |
| 16 | 缺少'=' | missing '='. |
| 17 | 函数调用时参数过少 | too few arguments for call. |
| 18 | 函数调用时参数过多 | too many actual parameters. |
| 19 | 函数调用时参数类型不匹配 | proc: type conversion error. |
| 20 | 函数没有返回值 | must return a value. |
| 21 | void函数有返回值 | ‘void’ function returning a value. |
| 22 | 返回值类型不匹配 | return: type error. |
| 23 | 类型不匹配 | type conversion error. |
| 24 | 常量被修改 | cannot assign to a variable that is const. |

### 3.2 警告信息列表

|  |  |  |
| --- | --- | --- |
| 编号 | 错误说明 | 提示语句 |
| 0 | float类型转换至int类型 | conversion from 'float' to 'int', possible loss of data. |
| 1 | float类型转换至char类型 | conversion from 'float' to 'char', possible loss of data. |