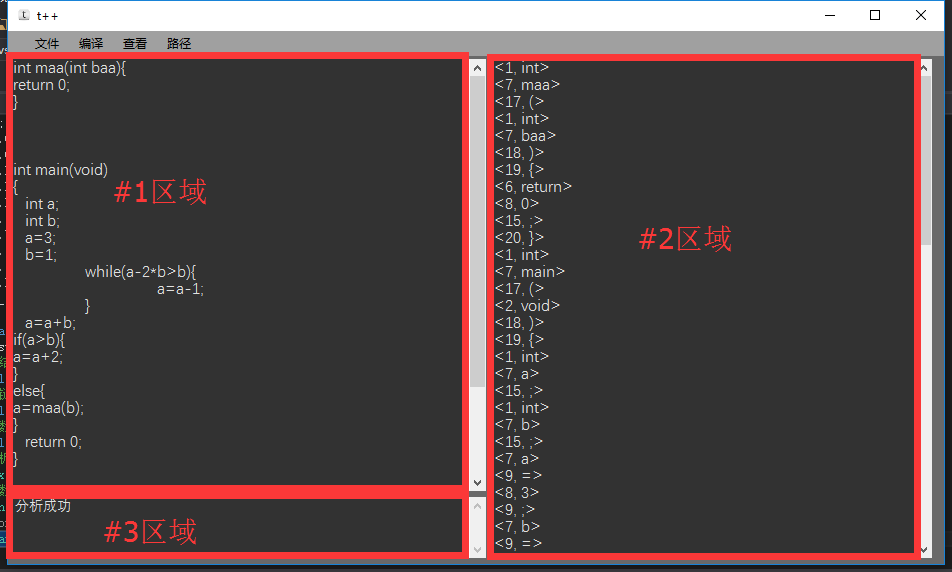
# 1 程序功能描述

程序主界面如下图所示：



程序界面分为三个区域。图中#1区域是代码编辑区，打开或新建的代码在该区域进行查看和编辑；#2区域是结果查看区，该区域可以显示词法分析、语法分析、中间代码和目标代码的生成结果；#3区域是编译提示区，该区域可以显示编译的结果，在编译错误的情况下显示相关的错误信息。

通过按键对程序功能的描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 选项 | | 功能 |
| 文件 | 打开 | 打开已有的.c文件 |
| 新建 | 新建源代码 |
| 退出 | 退出程序 |
| 保存 | 将源代码保存到当前路径 |
| 关闭 | 关闭打开的源代码 |
| 另存为 | 将源代码保存到另一指定路径 |
| 分析 | | 对源代码进行编译并生成相关文件 |
| 查看 | 词法分析 | 查看词法分析的结果 |
| 语法分析 | 查看语法分析的结果 |
| 中间代码 | 查看生成的中间代码 |
| 目标代码 | 查看生成的目标代码 |
| 路径 | | 修改程序生成的相关文件的保存路径 |

# 2 程序具体实现

## 2.1 主要算法

### 2.1.1 词法分析

1. 词法规则

|  |  |
| --- | --- |
| 词项 | 编码 |
| int | 1 |
| void | 2 |
| if | 3 |
| else | 4 |
| while | 5 |
| return | 6 |
| ID | 7 |
| NUM | 8 |
| = | 9 |
| + | 10 |
| - | 11 |
| \* | 12 |
| / | 13 |
| repo | 14 |
| ; | 15 |
| , | 16 |
| ( | 17 |
| ) | 18 |
| { | 19 |
| } | 20 |
| # | -1 |
| NONE | 0 |

1. 状态转换图



1. 核心算法流程图



### 2.1.2 语法分析

1. 消除左递归后的文法

Program ::= <声明串>

<声明串> ::=<声明>{<声明>}

<声明> ::=int <ID> <声明类型> | void <ID> <函数声明>

<声明类型>::=<变量声明> | <函数声明>

<变量声明> ::= ;

<函数声明> ::=’ (‘<形参>’) ‘<语句块>

<形参>::= <参数列表> | void

<参数列表> ::= <参数> {, <参数>}

<参数> ::= int <ID>

<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> ;

<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> ::=num | ‘(‘<表达式>’)’ |<ID> FTYPE

FTYPE ::= <call> | 空

<call> ::=’(’<实参> ’) ’

<实参> ::=<实参列表> | 空

<实参列表> ::= <表达式>{, <表达式>}

1. LL(1)语法分析方法
2. 自顶向下建树，试图生成一个和所给符号串（输入符号串）相一致的终结符号串；
3. 在建树过程中，按照给定的文法选择生成规则，展开为向下生长的树。每一步都试图将生成的尚未消除的最左叶片与输入符号匹配；
4. 匹配失败后，退回到出错点，选择其他可能的规则，直到生长出与读入符号串完全一致的树形结构；
5. 如果按上述方法构造成功，说明读入的符号串是文法的一个句子；反之，如果穷尽所有的途径都没有办法成功，则说明读入的符号串不是文法的句子。
6. 错误行号计算
7. 定义变量errorline记录错误行号；
8. 读入符号串的过程中每读到’\n’，则将errorline加1，进行回溯时errorline随符号串的读取位置变化进行对应修改；
9. 当分析到错误时，符号串不再被读取，errorline不再变化，此时errorline记录了错误行号。

### 2.1.3 生成中间代码

1. 递归下降翻译
2. 对每个非终结符A构造一个函数过程，对A的每个继承属性设置一个形式参数，函数的返回值为A的综合属性（作为记录，或指向记录的一个指针，记录中有若干域，每个属性对应一个域）。A对应的函数过程中，为出现在A的产生式中的每一个文法符号的每一个属性都设置一个局部变量；
3. 非终结符A对应的函数过程中根据当前的输入符号决定使用哪个产生式候选；
4. 每个产生式对应的程序代码中，按照从左到右的次序，对于单词符（终结符）、非终结符和语义动作分别做以下工作：

对于带有综合属性x的终结符X，把x的值存入为X.x设置的变量中，然后产生一个匹配X的调用，并继续读入一个输入符号；

对于每个非终结符B，产生一个右边带有函数调用的赋值语句c=B(b1, b2, …, bk)，其中，b1, b2, …, bk是为B的继承属性设置的变量，c是为B的综合属性设置的变量；

对于语义动作，把动作的代码抄进分析器中，用代表属性的变量来代替对属性的每一次调用。

### 2.1.4 生成目标代码

1. 生成算法

对于每一个中间代码，将它依据以下方法转化成目标代码，其中每一次使用到的寄存器在使用前进行压栈，使用后进行出栈。

## 2.2 基本框图



## 2.3 主要模块

程序使用C#语言，面向对象进行编程，因此整个程序的模块通过类进行划分。程序的类的结构如下图所示：



### 2.3.1 Program类

包含了应用程序的初始化的相关函数。

### 2.3.2 Form类

包含了界面初始化和控件动作定义的相关函数，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| filename | string | 当前打开文件路径 |
| sourcecode | string | 待编译源代码 |
| resultpath | string | 生成文件保存路径 |
| tra | Translator | 编译器实例 |

### 2.3.3 Node类

作为符号表结点，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| symbol | public string | 符号名 |
| scope | public string | 作用域 |
| next | public Node | 下一结点 |

### 2.3.4 LinkList类

符号表，保存函数名、变量和参数等信息，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| head | public Node | 符号表的头结点 |

### 2.3.5 FuncName类

函数表，保存所有的函数信息，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| isreturn | public bool | 返回值标志 |
| start | public int | 中间代码的起始行 |
| sentenceline | public int | 中间代码的行数 |
| paranum | public int | 参数个数 |
| varnum | public int | 变量个数 |
| name | public string | 函数名 |
| sentenceblock | public string | 中间代码 |
| lastcode | public string | 目标代码 |
| paralist | public string[] | 参数表 |
| varlist | public string[] | 变量表 |

### 2.3.6 lexical类

词法分析器，读取字符，返回有效的符号，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| sourcecode | private string | 待编译源代码 |
| ch | private char | 预读字符 |
| strToken | private string | 当前获得的字符串 |
| scope | public string | 作用域 |
| cur | public int | 当前结点所在语法树的位置游标 |
| code | public int | 词法类型号 |
| line | public int | 当前字符所在行号 |
| info | public string | 当前词的具体信息 |
| end | public bool | 结束标志 |
| sblt | public LinkList | 符号表实例 |

### 2.3.7 SynTree类

语法分析树结点，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| parentCur | public int | 父结点游标 |
| childNum | public int | 子结点数量 |
| child | public int[] | 子结点游标保存数组 |
| info | public string | 当前结点信息 |
| place | public string | 当前结点的变量名 |
| line | public int | 当前结点首行 |
| linenum | public int | 当前结点包含行数 |
| midsen | public string | 当前结点包含的中间代码 |
| lastsen | public string | 当前结点包含的目标代码 |

### 2.3.8 Translator类

编译器，数据成员如下：

|  |  |  |
| --- | --- | --- |
| 数据名 | 类型 | 含义 |
| cur | public int | 语法分析树结点游标 |
| errorline | public int | 错误发生行 |
| supervarnum | public int | 全局变量数 |
| funcnamecount | public int | 函数的数量 |
| isreturn | public bool | 函数是否有返回值 |
| iscontrol | public bool | 是否是控制语句 |
| supervar | public string[] | 全局变量表 |
| errorinfo | public string | 错误返回信息 |
| lexpath | public string | 词法二元式保存路径 |
| midpath | public string | 中间代码的保存路径 |
| goalpath | public string | 目标代码的保存路径 |
| lexic | public lexical | 词法分析器实例化 |
| syntree | public SynTree[] | 语法分析树结点存储表 |
| funcname | public FuncName[] | 函数表 |
| codeline | private int | 中间代码表的下一打印行数 |
| midline | private int | 中间代码下一打印行所在行数 |
| count | private int | 中间变量的数量 |
| ifnum | private int | 函数体内if语句的数量 |
| whilenum | private int | 函数体内while语句的数量 |
| scope | private string | 当前位置所在域 |
| wdata | private string | 待写入文件数据区 |
| cachepath | private string | 缓存文件存放路径 |
| lexfs | private FileStream | 词法分析结果保存文件的文件描述符 |
| midfs | private FileStream | 中间代码保存文件的文件描述符 |
| goalfs | private FileStream | 目标代码保存文件的文件描述符 |
| cachefs | private FileStream | 缓存信息保存文件的文件描述符 |
| lexsw | private StreamWriter | 词法分析结果流式写句柄 |
| midsw | private StreamWriter | 中间代码流式写句柄 |
| goalsw | private StreamWriter | 目标代码流式写句柄 |
| cachesw | private StreamWriter | 缓存信息流式写句柄 |
| dgv1 | private DataGridView | 数据表控件的实例化 |

## 2.4 功能函数

### 2.4.1 Form类

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 说明 |
| 打开ToolStripMenuItem\_Click | private void | 打开源代码文件 |
| 新建ToolStripMenuItem\_Click | private void | 新建源代码 |
| 退出ToolStripMenuItem\_Click | private void | 退出程序 |
| 保存ToolStripMenuItem\_Click | private void | 保存源代码 |
| 关闭ToolStripMenuItem\_Click | private void | 关闭源代码 |
| 另存为ToolStripMenuItem\_Click | private void | 将源代码保存到其他文件 |
| 分析ToolStripMenuItem\_Click | private void | 编译 |
| 词法分析ToolStripMenuItem\_Click | private void | 查看词法分析结果 |
| 语法分析ToolStripMenuItem\_Click | private void | 查看语法分析结果 |
| button1\_Click | private void | 展开语法树 |
| button2\_Click | private void | 收起语法树 |
| 中间代码ToolStripMenuItem\_Click | private void | 查看中间代码 |
| 目标代码ToolStripMenuItem\_Click | private void | 查看目标代码 |
| 路径ToolStripMenuItem\_Click | private void | 修改保存文件路径 |
| textBox1\_TextChanged | private void | 修改源代码时重置界面 |
| treeView1\_VisibleChanged | private void | 同步语法树的按钮显示 |
| ShowSyntaxTree | private void | 显示语法分析树 |
| MakeSyntaxTree | private void | 同步树的结点 |
| ShowState | private void | 显示运行状态信息 |

### 2.4.2 LinkList类

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 说明 |
| GetLength | public int | 求单链表的长度 |
| Clear | public void | 清空单链表 |
| IsEmpty | public bool | 判断单链表是否为空 |
| Add | public int | 在单链表的末尾添加新元素 |
| Delete | public void | 删除单链表的第i个结点 |
| Locate | public int | 在单链表中查找symbol结点 |
| update | public void | 删除域为scope的结点 |

### 2.4.3 lexical类

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 说明 |
| Initial | public void | 初始化各变量 |
| GetChar | private void | 读取一个符号 |
| GetBC | private void | 跳过空格、制表符和换行符 |
| Concat | private void | 将符号添加到符号串尾部 |
| Reserve | private void | 检查符号串是否是关键字 |
| Retract | private void | 回退一个符号 |
| InsertId | public int | 将符号串插入到符号表 |
| CheckId | private int | 检查符号串是否在符号表中 |
| lex | public bool | 读取字符，匹配字符串 |

### 2.4.4 Translator类

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 说明 |
| Initial | public void | 初始化各变量 |
| CheckProgram | public bool | 检查当前符号是否符合Program的下推文法 |
| CheckDeclareList | private bool | 检查当前符号是否符合声明串的下推文法 |
| CheckDeclare | private bool | 检查当前符号是否符合声明的下推文法 |
| CheckDeclareType | private bool | 检查当前符号是否符合生命类型的下推文法 |
| CheckVariableDeclare | private bool | 检查当前符号是否符合变量声明的下推文法 |
| CheckFunctionDeclare | private bool | 检查当前符号是否符合函数声明的下推文法 |
| CheckFormalParameter | private bool | 检查当前符号是否符合形参的下推文法 |
| CheckParameterList | private bool | 检查当前符号是否符合参数列表的下推文法 |
| CheckParameter | private bool | 检查当前符号是否符合参数的下推文法 |
| CheckSentenceBlock | private bool | 检查当前符号是否符合语句块的下推文法 |
| CheckInternalDeclare | private bool | 检查当前符号是否符合内部声明的下推文法 |
| CheckInternalVariableDeclare | private bool | 检查当前符号是否符合内部变量声明的下推文法 |
| CheckSentenceList | private bool | 检查当前符号是否符合语句串的下推文法 |
| CheckSentence | private bool | 检查当前符号是否符合语句的下推文法 |
| CheckAssignmentSentence | private bool | 检查当前符号是否符合赋值语句的下推文法 |
| CheckReturnSentence | private bool | 检查当前符号是否符合return语句的下推文法 |
| CheckWhileSentence | private bool | 检查当前符号是否符合while语句的下推文法 |
| CheckIfSentence | private bool | 检查当前符号是否符合if语句的下推文法 |
| CheckExpression | private bool | 检查当前符号是否符合表达式的下推文法 |
| CheckAdditiveExpression | private bool | 检查当前符号是否符合加法表达式的下推文法 |
| CheckItem | private bool | 检查当前符号是否符合项的下推文法 |
| CheckFactor | private bool | 检查当前符号是否符合因子的下推文法 |
| CheckFtype | private bool | 检查当前符号是否符合Ftype的下推文法 |
| CheckCall | private bool | 检查当前符号是否符合Call的下推文法 |
| CheckActualParameter | private bool | 检查当前符号是否符合实参的下推文法 |
| CheckActualParameterList | private bool | 检查当前符号是否符合实参列表的下推文法 |
| GenerateIntermediateVariable | private string | 产生一个新的中间变量 |
| GererateWhileLable | private string | 产生一个while语句中的标签 |
| GererateIfLable | private string | 产生一个if语句中的标签 |
| PrintFormula | private string | 产生四元式 |
| EmitFormula | private void | 将四元式输出到文件 |
| FindFuncName | private int | 查找函数名 |
| AddFuncName | private void | 添加函数名 |
| DelFuncName | private void | 删除函数名 |
| GenerateLastCode | private void | 产生目标代码 |