编译原理实验要求

构建S-ONNX模型的编译器S-ONNXCompiler

**目录**

[一、 背景简介 3](#_Toc20612)

[二、 任务一：撰写S-ONNXCompiler词法分析和语法分析程序 3](#_Toc25628)

[2.1 S-ONNX的词法规则 3](#_Toc24507)

[2.2 S-ONNX的语法规则 6](#_Toc31706)

[2.3 撰写S-ONNXCompiler的词法和词法分析程序 8](#_Toc9211)

[三、 任务二：撰写S-ONNXCompiler语义分析与代码生成程序 8](#_Toc31184)

[3.1 S-ONNX模型的语义分析需求 9](#_Toc661)

[3.2 S-ONNX模型的中间代码生成要求 11](#_Toc10915)

[3.3 S-ONNXCompiler的语义和代码生成任务 13](#_Toc2004)

[四、 任务三：撰写错误处理程序 13](#_Toc23264)

[五、 任务四：对S-ONNXComplier进行测试 13](#_Toc8656)

[六、 实验报告要求 28](#_Toc19705)

[七、 anltr4开源工具简介 29](#_Toc19203)

[八、 手工撰写词法分析程序参考案例 32](#_Toc5127)

[8.1 Tiny语言的词法规则定义 32](#_Toc27269)

[8.2 基于自动机模型的Tiny语言的词法程序 32](#_Toc8604)

[九、 手工撰写语法分析程序参考案例 42](#_Toc7954)

[9.1 Tiny语言语法规则定义 42](#_Toc1840)

[9.2 语法分析程序实现步骤 42](#_Toc4932)

ONNX（Open Neural Network Exchange）是一种开放的神经网络模型交换格式，广泛应用于不同深度学习框架之间的模型共享和迁移。ONNX通过标准化的模型表示，使得开发者能够在TensorFlow、PyTorch、Caffe等多种框架之间无缝地传递和部署神经网络模型，其核心是计算图，包括节点（NodeProto）、张量（TensorProto）和操作符（Operator）。

ONNX编译技术是将ONNX模型进行解析、优化，并生成适用于不同硬件平台的代码，以实现模型的高效推理和跨平台运行。本实验项目将对ONNX模型进行裁剪，形成S-ONNX（Simplified Open Neural Network Exchange），保留了ONNX模型的核心结构和语义。本实验项目要求研制一款编译器S-ONNXCompiler（**S-ONNXCompiler的编程语言不限**），S-ONNXCompiler编译器需要完成以下任务：

1. 依据S-ONNX的词法规则，把输入的S-ONNX源代码切分为词法单元，例如关键字、专用符号、标识符、常量等，为后续的语法分析提供基础；
2. 根据S-ONNX的语法规则，对词法分析得到的词法单元序列进行分析，判断其是否符合S-ONNX的语法结构，并生成抽象语法树；
3. 根据S-ONNX的语义要求，对S-ONNX模型进行语义检查；
4. 在完成上述分析的基础上，将S-ONNX模型转换为三地址码表示的中间代码。

在以上实验的基础上，按照实验要求撰写实验报告。

# 任务一：撰写S-ONNXCompiler词法和语法分析程序

## S-ONNX的词法规则

1. S-ONNX语言的关键字（31个）：

| 关键字 | 含义 |
| --- | --- |
| ModelProto | 定义了S-ONNX模型的整体结构。一个完整的S-ONNX模型通常由ModelProto消息表示，这个消息包含了模型的多个重要部分，像版本信息、图结构、算子集导入等。 |
| graph | 用于定义神经网络模型中的计算图，是S-ONNX模型的核心部分，在其内部组织节点、输入、输出等元素，构建完整的模型计算逻辑 |
| name | 用于唯一标识各种元素 |
| node | 代表计算图中的节点，每个节点执行特定的运算操作，如卷积、加法等，是构成计算图的基本单元 |
| input | 指定模型的输入张量信息，包括输入的名称、数据类型、形状等，用于明确模型的输入数据 |
| output | 表示模型的输出张量信息，定义了模型计算后的输出结果 |
| op\_type | 在定义节点时，用于指定节点所执行的操作类型，如“Conv”表示卷积操作、“Add”表示加法操作等 |
| attribute | 为节点添加额外的属性信息，例如卷积操作中的卷积核大小、步长等参数 |
| initializer | 当一个输入永不改变（例如线性回归的系数）时，将其转换为存储在图中的常量，即初始化器 |
| doc\_string | 模型的可读文档，允许使用Markdown格式书写 |
| domain | 一个反向DNS名称，用于指示模型命名空间或域，如‘org.onnx’ |
| model\_version | 模型本身的版本，以整数编码 |
| producer\_name | 生成模型的工具名称 |
| producer\_version | 生成工具的版本 |
| type | 一般用于指定张量、节点输入输出等的类型，比如tensor类型、sequence类型等，明确数据的结构和格式 |
| tensor\_type | 专门用于定义张量类型，包含数据类型（如float32、int64等）和形状信息，是描述张量特征的关键关键字 |
| ir\_version | 指示ONNX中间表示（IR）的版本，确保模型与不同版本的ONNX框架兼容 |
| elem\_type | 在ONNX中用于指定张量元素的数据类型。例如可以是float32、int64等，它决定了张量中每个元素存储的数据类型和精度。 |
| shape | 表示张量的形状，即各个维度的大小。它描述了张量在每个维度上的元素数量，对于确定张量的存储结构和参与计算的方式至关重要。例如一个二维张量的形状可能是[3,4]，表示它有3行4列。 |
| dim | 通常用来表示张量的一个维度，在描述张量形状时，dim可以用来指代每个具体的维度大小。比如在描述一个三维张量[2,3,4]时，每个数字都可以看作一个dim。 |
| dims | 是dim的复数形式，用于表示张量所有维度的集合，也就是完整描述张量形状的信息。例如dims=[2,3,4]完整地定义了一个三维张量的形状。 |
| raw\_data | 用于存储张量的实际数据，当张量数据以二进制形式存储时会使用这个关键字。通过raw\_data可以将数据直接嵌入到S-ONNX模型中，方便模型的传输和使用。 |
| opset\_import | 用于指定模型所使用的S-ONNX算子集版本。每个算子集版本定义了一组特定的算子及其行为，通过指定opset\_import可以确保模型在不同的S-ONNX运行环境中具有一致的计算结果。 |
| dim\_value | 指维度的值，即dim所代表的具体大小。例如在dim=5中，5就是dim\_value。 |
| data\_type | 与elem\_type类似，用于指定数据的类型，涵盖了各种数值类型（如整数、浮点数）、布尔类型、字符串类型等，明确数据的存储和处理方式。 |
| version | 一般用于表示模型、算子集或者S-ONNX版本的编号。例如模型的版本号可以用于标识模型的迭代更新，算子集版本号则决定了使用哪些算子定义。 |
| value | 是一个较为宽泛的概念，在不同的上下文中有不同的含义。它可以表示属性的值、张量的值等，用于携带具体的数据信息。 |
| int | 整型 |
| float | 浮点型 |
| string | 字符串类型 |
| bool | 布尔类型 |

注：关键字是保留字，并且不区分大小写。

1. 语言的专用符号6个：

| 专用符号 | 含义 |
| --- | --- |
| [ ] | 通常用于表示列表或者数组。在描述张量的形状时，使用[dim1, dim2, ...]来表示各个维度的大小；在定义节点的输入输出列表时，也会用方括号，如inputs = [input1, input2]表示该节点有两个输入 |
| { } | 用于定义部分关键字的区域，比如node{}中{}表示该node的定义区域 |
| , | 作为元素之间的分隔符。在列表、元组、参数列表等场景中使用，如在定义多个输入时，input1, input2, input3用逗号分隔不同的输入；在描述张量形状时，[dim1, dim2, dim3]也是用逗号分隔各个维度的大小 |
| = | 用于赋值操作，将一个值或者对象赋给一个变量或者属性。例如attribute\_name = value表示将value赋给attribute\_name这个属性 |

1. 整数INTEGER、字符串STRING和字节数据BYTES的规则通过下述正则表达式定义：

INTEGER\_TYPE\_SUFFIX = l | L

INTEGER= (0 | [1-9] [0-9]\*) INTEGER\_TYPE\_SUFFIX?

ESCAPE\_SEQUENCE = \ ( b | t | n | f | r | " | ' | \ )

STRING = "(ESCAPE\_SEQUENCE | (~\ | ~") )\*"

BYTES= [0-9A-Fa-f]+b

其中，下表给出正则表达式中运算符的含义：

|  |  |
| --- | --- |
| ~\ | ~” | 表示匹配除了 \ 和 ” 外的所有字符都是合法的 |
| [a-z] | 与课件ppt中含义一致 |
| ? | 与课件ppt中含义一致 |
| + | 与课件ppt中含义一致 |
| \* | 与课件ppt中含义一致 |
| a | b | 与课件ppt中含义一致 |
| ( ) | 与课件ppt中含义一致 |

## S-ONNX的语法规则

S-ONNX的语法规则描述如下：

*G[model]*

*model* ->“**ModelProto**”**“{”***model\_body\_def* **“}”**

*model\_body\_def* -> *ir\_version\_def producer\_name\_def producer\_version\_def domain\_def model\_version\_def doc\_string\_def graph\_def opset\_import\_def*

*ir\_version\_def* ->“**ir\_version**”“**=**”**INTEGER**

*producer\_name\_def* ->“**producer\_name**”“**=**” **STRING**

*producer\_version\_def* ->“**producer\_version**”“**=**”**STRING**

*domain\_def* ->“**domain**”“**=**”**STRING**

*model\_version\_def* ->“**model\_version**”“**=**”**INTEGER**

*doc\_string\_def* ->“**doc\_string**”“**=**”**STRING**

*graph\_def* ->“**graph**”**“{”** *graph\_body\_def* **“}”**

*graph\_body\_def* -> *name\_def node\_list input\_list output\_list initializer\_list?*

*name\_def* ->“**name**”“**=**”**STRING**

*node\_list* ->“**node**”**“{”** *node\_def* **“}”** *node\_repeats*

*node\_repeats*-> *node\_list node\_repeats* **|****ε**

*input\_list* ->“**input**”**“{”** *value\_info\_def* **“}”***input\_repeats*

*input\_repeats*-> *input\_list input\_repeats* **|****ε**

*output\_list* ->“**output**”**“{”** *value\_info\_def* **“}”***output\_repeats*

*output\_repeats*->*output\_list output\_repeats* **|****ε**

*initializer\_list*->“**initializer**”**“{”***tensor\_def* **“}”***initializer\_repeats*

*initializer\_repeats*-> *initializer\_list initializer\_repeats* **|****ε**

*node\_def* -> *op\_type\_def name\_def* **(***input\_list* **|** *input\_arr* **) (***output\_list* **|** *output\_arr***)**  *attribute\_list?*

*op\_type\_def* ->“**op\_type**”“**=**”**STRING**

*input\_arr* **->** “**input**”“**=**”“[”**STRING** *id\_repeats*“]”

*output\_arr* **->** “**output**”“**=**”“[”**STRING** *id\_repeats*“]”

*id\_repeats* -> “,”**STRING** *id\_repeats* **|** **ε**

*attribute\_list*->“**attribute**”**“{”***attribute\_def* **“}”***attribute\_repeats*

*attribute\_repeats* -> *attribute\_list attribute\_repeats* **|****ε**

*attribute\_def* -> *name\_def value\_def*

*value\_def* ->**“value” “=” STRING**

*value\_info\_def* -> *name\_def type\_def*

*type\_def* ->“**type**”**“{”** *tensor\_type\_def* **“}”**

*tensor\_type\_def* ->“**tensor\_type**”**“{”** *elem\_type\_def shape\_def* **“}”**

*elem\_type\_def* ->“**elem\_type**”“**=**”“**int**” **|** “**float**” **|** “**string**” **|** “**bool**”

*shape\_def* ->“**shape**”**“{”** *dim\_list* **“}”**

*dim\_list* ->“**dim**”**“{”** *dim\_def* **“}”***dim\_repeats*

*dim\_repeats* -> *dim\_list dim\_repeats* **|****ε**

*dim\_def* ->“**dim\_value**”“**=**”**INTEGER |**“**dim\_param**”“**=**”**STRING**

*tensor\_def* -> *name\_def data\_type\_def dims\_def raw\_data\_def*

*data\_type\_def* ->“**data\_type**”“**=**”“**int**” **|** “**float**” **|** “**string**” **|** “**bool**”

*dims\_def* ->“**dims**”“**=**”**INTEGER** *dims\_repeats*

*dims\_repeats* ->**INTEGER** *dims\_repeats* **|****ε**

*raw\_data\_def* ->“**raw\_data**”“**=**”**BYTES**

*opset\_import\_def* ->“**opset\_import**”**“{”** *domain\_def version\_def* **“}”**

*version\_def* ->“**version**”“**=**”**INTEGER**

其中，红色双引号括起来字符串，表示匹配该字符串本身。

## 撰写词法和词法分析程序

可以手工撰写或利用Antlr4工具（参考Antlr4开源工具）生成词法和语法分析程序。下面给出S-ONNXCompiler的词法和语法分析的任务：

1. 根据2.1的词法规则，**输入为“xxxx.txt”S-ONNX模型文件，**实现一个能够将输入的S-ONNX源代码分割成词法单元的词法分析程序，并提供getToken()接口输出单词，每个单词包括单词的类型和词值两部分。
2. 根据S-ONNX语法规则的文法设计抽象语法树结构，抽象语法树的定义尽量简洁易懂，必要时提供说明。
3. 依据2.2的语法规则，实现一个能够对词法单元序列进行自上而下或自下而上的语法分析，判断其是否符合S-ONNX语法的语法分析程序，输出格式化的抽象语法树。树的格式化形式可以用缩进来表示父子级关系，也可以可视化展示。

# 任务二：撰写语义分析与代码生成程序

## S-ONNX模型的语义分析需求

本次实验语义分析的内容有以下3条：

1. 命名冲突

|  |  |
| --- | --- |
| 示例 | 语义错误 |
| node{  op\_type = “Add”  name = “A”  ...  }  node{  op\_type = “Sub”  name = “A”  ...  } | 节点名称重复——同一个模型中，不能有同名节点。 |
| tensor {  name ="tensor1"  elem\_type= FLOAT  ...  }  tensor {  name="tensor1"  elem\_type=INT  ...  } | 张量名称重复——同一个模型中，不能有同名张量。 |
| node {  name = "AddNode"  op\_type = "Add"  input = ["input1", "input2"]  output = ["output1"]  }  node {  name = "SubNode"  op\_type = "Sub"  input = ["input3", "output1"]  output = ["output1"]  } | 输出命名冲突——输出张量的名称必须是唯一的，否则会使模型的计算图产生混淆。  输出张量output1被两个节点重复使用。 |

1. 未定义即使用

|  |  |
| --- | --- |
| 示例 | 语义错误 |
| node {  name = "B"  op\_type = "MatMul"  input = ["unknown\_tensor", "valid\_tensor"]  output = ["result\_tensor"]  ...  }  tensor {  name = "valid\_tensor"  elem\_type = FLOAT  ...  } | 输入张量必须先定义才能被引用——输入张量“unknown\_tensor”应该先定义才能被引用 |
| node {  name = "D"  op\_type = "Conv"  input = ["input\_image", "undefined\_initializer"]  output = ["output\_feature\_map"]  ...  }  tensor {  name: "input\_image"  elem\_type: FLOAT  ...  } | initializer必须先定义才能被引用——“undefined\_initializer”应该先定义才能被引用 |

1. 张量类型检查

|  |  |
| --- | --- |
| 示例 | 语义错误 |
| node {  name = "A"  op\_type = "Add"  input = ["tensor1", "tensor2"]  output = ["tensor3"]  ...  }  tensor {  name = "tensor1"  elem\_type= FLOAT  ...  }  tensor {  name = "tensor2"  elem\_type= INT  ...  } | 同一操作符的输入张量类型必须一致——Add操作符的输入张量类型不一致 |
| graph {  node {  name = "D"  op\_type = "Conv"  input = ["input\_tensor"]  output = ["output\_tensor"]  ...  }  input = ["input\_tensor"]  output = ["output\_tensor"]  }  tensor {  name = "input\_tensor"  elem\_type = FLOAT  ...  }  tensor {  name = "output\_tensor"  elem\_type = BOOL  ...  } | 模型整体输入输出不匹配——卷积操作的输入输出是浮点类型，该示例中模型输入为FLOAT类型，输出为BOOL类型 |

## S-ONNX模型的中间代码生成要求

在对S-ONNX模型进行编译、优化或推理等处理过程中，会产生不同形式的中间代码。例如，可以利用商业软件Intel OpenVINO Toolkit生成包括描述卷积层、池化层、全连接层等节点的连接关系和参数的.xml文件，保存各层权重数据的.bin文件等。

本次实验中间代码使用三地址码（Three-Address Code, TAC）表示，每条指令由以下部分组成：

<result> = <operation> <operand1>, <operand2>, ..., <attributes>

其中：

<result>：指令的输出变量（通常是一个张量）。

<operation>：操作符（如 `Input`, `Output`, `Conv`, `Relu` 等）。

<operand1>, <operand2>, ...：操作数，通常是输入张量或常量。

<attributes>：操作的属性。

表1 三地址码规范类

|  |  |  |  |
| --- | --- | --- | --- |
| **操作类型** | **三地址码规范** | **含义** | **示例** |
| 输入张量定义 | <result> = Input(<name>, <data\_type>, <shape> | <name>：输入张量的名称 | T1 = Input("input1", FLOAT, [1, 3, 224, 224]) |
| <data\_type>：张量的数据类型（如 `FLOAT`, `INT32`） |
| <shape>：张量的形状（如 `[1, 3, 224, 224]`） |
| 输出张量定义 | Output(<name>, <operand>) | <name>：输出张量的名称 | Output("output1", T2) |
| <operand>：输出张量的来源（通常是一个中间变量） |
| 通用算子 | <result> = <operation>(<operand1>, <operand2>, ..., <attributes>) | <operation>：操作符名称（如 `Conv`, `Relu`, `Add`） | 卷积操作：T3 = Conv(T1, T2, kernel\_shape=[3, 3], strides=[1, 1]) |
| <operand1>, <operand2>, ...：输入张量 |
| 激活函数：T4 = Relu(T3) |
| <attributes>：操作的属性（如 `kernel\_shape`, `strides`） |
| 常量定义 | <result> = Constant(<value>, <data\_type>, <shape>) | <value>：常量值（如 `[1, 2, 3]`） | T5 = Constant([1, 2, 3], INT32, [3]) |
| <data\_type>：常量的数据类型 |
| <shape>：常量的形状 |
| 权重初始化 | <result> = Initializer(<name>, <data\_type>, <shape>, <raw\_data>) | <name>：权重张量的名称 | W1 = Initializer("weight1", FLOAT, [64, 3, 3, 3], raw\_data=...) |
| <data\_type>：权重的数据类型 |
| <shape>：权重的形状 |
| <raw\_data>：权重的原始数据 |
| 张量维度操作 | <result> = Reshape(<operand>, <new\_shape>) | <operand>：需要调整形状的张量 | T6 = Reshape(T3, [1, 64, 112, 112]) |
| <new\_shape>：新的形状 |
| 条件判断 | If(<condition>) Goto <label> | <condition>：条件表达式 |  |
| <label>：跳转的目标标签 |
| 循环 | <label>: <body> If(<condition>) Goto <label> |  | If(T7 > 0) Goto L1 |
| 打印张量信息 | Print(<operand>) | <operand>：需要打印的张量 | Print(T3) |

## S-ONNXCompiler的语义和代码生成任务

1. 在实验一内容的基础上，对S-ONNX模型进行语义检查，完成语义分析。
2. 解析S-ONNX模型并生成相应的三地址码，按照表1给出的中间代码生成规范，生成相应的中间代码，实现方法可参考ppt中第六章。
3. 根据语义规则，给出抽象语法树各节点对应的中间代码生成方法；
4. 后序遍历抽象语法树，生成相应的中间代码。

# 任务三：撰写错误处理程序

任务三要求在S-ONNXComplier编译器中设计并实现错误处理模块，主要内容包括：

1. 错误分类
2. 词法错误：如非法字符、字符串未闭合等。
3. 语法错误：如缺少必要的符号（括号、分号等）、结构不匹配。
4. 语义错误：如命名冲突、未定义引用、类型不匹配等。
5. 错误报告机制
6. 错误定位：错误信息中需标明发生错误的文件、行号以及相应的代码片段。
7. 错误描述：错误提示需准确描述错误类型、原因。

# 任务四：对S-ONNXComplier进行测试

本节中给出了10个测试用例，作为最低要求，可以自行增加测试用例，对S-ONNXComplier进行充分测试。

1. 测试用例1

|  |
| --- |
| ModelProto{  ir\_version=8  producer\_name= "onnx-example"  producer\_version = "1.0"  domain = "example\_domain"  model\_version = 1  doc\_string = "This is an example ONNX model."  graph {  name= "test-model"  node {  op\_type= "Pad"  name= "test-node"  input=["X","pads","value"]  output=[ "Y" ]  attribute {  name= "mode"  value = "33"  }  }  input {  name= "X"  type {  tensor\_type {  elem\_type= int  shape {  dim {  dim\_value= 3  }  dim {  dim\_value= 2  }  }  }  }  }  input {  name="pads"  type {  tensor\_type {  elem\_type= int  shape {  dim {  dim\_value= 1  }  dim {  dim\_value= 4  }  }  }  }  }  input {  name= "value"  type {  tensor\_type {  elem\_type= int  shape {  dim {  dim\_value= 1  }  }  }  }  }  output {  name="Y"  type {  tensor\_type {  elem\_type= int  shape {  dim {  dim\_value=3  }  dim {  dim\_value=4  }  }  }  }  }  initializer {  name= "conv.bias"  data\_type=int  dims=1 2 3 4  raw\_data=000000000000b  }  }  opset\_import {  domain = "ex"  version=15  }  } |

1. 测试用例2

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel2."  graph{  name = "GraphWithInitializer"  node{  op\_type = "MatMul"  name = "MatMulNode"  input = ["input1", "initializer1"]  output = ["output1"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  dim{  dim\_value = 2  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  dim{  dim\_value = 1  }  }  }  }  }  initializer{  name = "initializer1"  data\_type = float  dims = 2 1  raw\_data = 01020304b  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |
|  |

1. 测试用例3

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel3."  graph{  name = "MultiNodeGraph"  node{  op\_type = "Mul"  name = "MulNode"  input = ["input1", "input2", "input3"]  output = ["output1"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input3"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

1. 测试用例4

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel4."  graph{  name = "ComplexShapeGraph"  node{  op\_type = "Conv"  name = "ConvNode"  input = ["input1", "initializer1"]  output = ["output1"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  dim{  dim\_value = 3  }  dim{  dim\_value = 224  }  dim{  dim\_value = 224  }  }  }  }  }  initializer{  name = "initializer1"  data\_type = float  dims = 64 3 3 3  raw\_data = 0102b  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  dim{  dim\_value = 64  }  dim{  dim\_value = 222  }  dim{  dim\_value = 222  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

1. 测试用例5

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel5."  graph{  name = "NodeWithStringAttrGraph"  node{  op\_type = "CustomOp"  name = "CustomNode"  input = ["input1"]  output = ["output1"]  attribute{  name = "customAttr"  value = "SomeStringValue"  }  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

1. 测试用例6

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel6."  graph{  name = "MultiIOGraph"  node{  op\_type = "Add"  name = "AddNode"  input = ["input1", "input2"]  output = ["output1", "output2"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

1. 测试用例7

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "Test"  graph{  node{  op\_type = "Add"  name = "Node1"  input = ["input1", "input2"]  output = ["output1"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  keyword = "value"  } |

1. 测试用例8

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "Test"  graph{  node{  op\_type = "Add"  name = "Node1"  input = ["input1", "input2"]  output = ["output1"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  }  } |

1. 测试用例9

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel9."  graph{  name = "DuplicateNodeNameGraph"  node{  op\_type = "Add"  name = "DuplicateNode"  input = ["input1"]  output = ["output1"]  }  node{  op\_type = "Mul"  name = "DuplicateNode"  input = ["input2"]  output = ["output2"]  }  input{  name = "input1"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  input{  name = "input2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  output{  name = "output1"  type{  tensor\_type{  elem\_type = int  shape{  dim{  dim\_value = 1  }  }  }  }  output{  name = "output2"  type{  tensor\_type{  elem\_type = float  shape{  dim{  dim\_value = 1  }  }  }  }  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

1. 测试用例10

|  |
| --- |
| ModelProto{  ir\_version = 1  producer\_name = "TestProducer"  producer\_version = "1.0"  domain = "test.onnx"  model\_version = 1  doc\_string = "This is testmodel10."  graph{  name = "SyntaxErrorGraph"  node{  op\_type = "Add"  name = "AddNode"  input = ["input1", "input2"]  output = ["output1"]  }  }  opset\_import{  domain = "ai.onnx"  version = 11  }  } |

# 实验报告要求

整合任务一至任务四的要求，构建一个完整的S-ONNX编译器，并依据实践内容撰写一篇实验报告，报告内容包括但不限于实验环境，实验内容，实验流程，测试说明，将这两次任务中的设计，实现，测试等内容讲述清楚即可。实验报告的要求如下：

1. 提交文件的命名方式为“**学号-姓名-编译原理实验报告**”，示例“2023123456-张三-编译原理实验报告”。
2. 提交实验过程中产生的所有源码。
3. 提交S-ONNXComplier的详细设计，写在.word文档中，命名为“**设计方案**”。包括：词法分析模块的设计方案，语法分析模块的设计方案，抽象语法树的设计方案，语义分析模块的设计方案，错误处理的设计方案，以及中间代码生成模块的设计方案。
4. 以**截图或图片形式**提交第五章中给出的10个测试用例的测试结果，写在“.word”文档中，命名为“**测试报告**”。包括词法分析模块的输出结果、语法分析模块的输出结果，语义分析模块的输出结果，中间代码模块的输出结果。

# 附件1：anltr4开源工具简介

Antlr4（Another Tool for Language Recognition）是一款基于Java开发的开源的语法分析器生成工具，能够根据语法规则文件生成对应的语法分析器，广泛应用于DSL构建，语言词法语法解析等领域。Antlr可以生成不同target的AST，包括Java、C++、JS、Python、C#等，可以满足不同语言的开发需求。下面是Antlr4相关网址：

1. 官方网址： <https://www.antlr.org/>
2. Github开源地址：<https://github.com/antlr/antlr4>
3. 官方文档：<https://github.com/antlr/antlr4/blob/master/doc/index.md>

关于如何使用antlr4生成词法，语法分析程序，有两种方法。

1. 继承listener方法，来完成对每类节点的监听。

首先，确保你在工程引入了org.antlr.v4.runtime 运行时库。

如何构建开发环境，查阅以下链接学习

<https://github.com/antlr/antlr4/blob/master/doc/java-target.md>

继承Listener来实现对每类节点的监听，查阅以下链接学习

<https://media.pragprog.com/titles/tpantlr2/listener.pdf>

这种方法的好处是用户无需自己遍历分析树，通过写监听函数即可实现对每类节点的监听。缺点是如果每类节点都有通用的操作，那么用户不得不为每类节点写监听函数，代码比较臃肿。

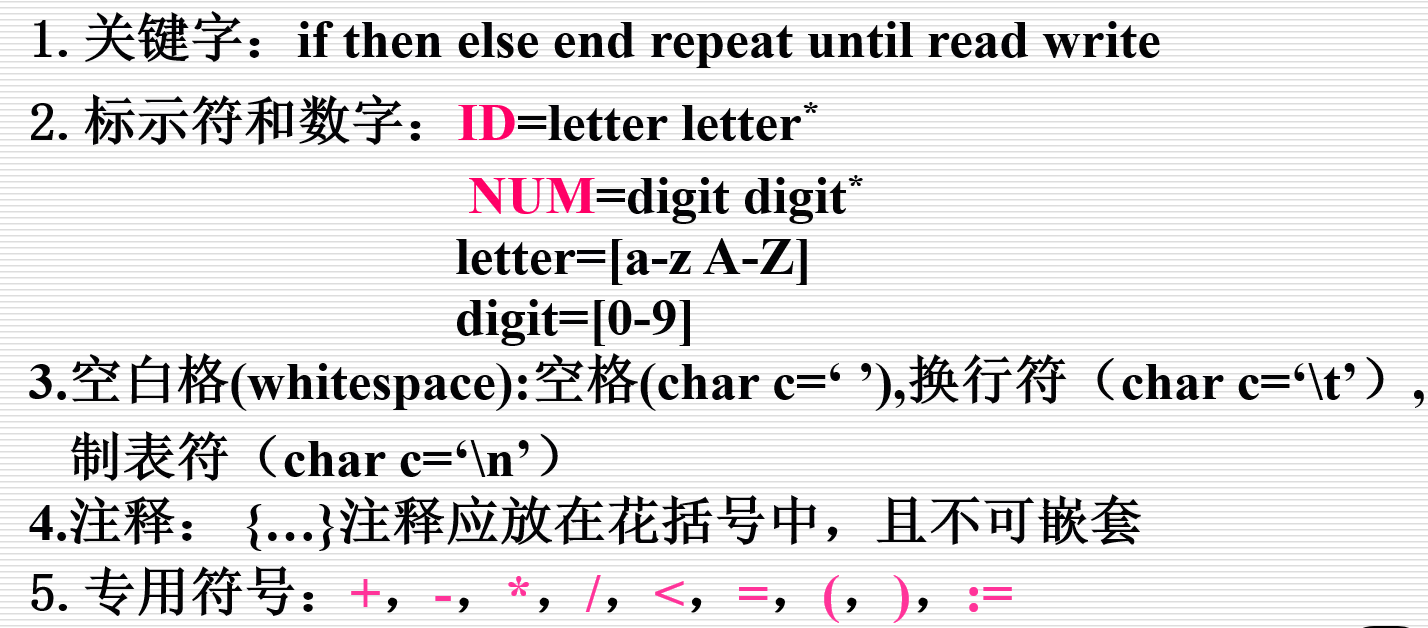
1. 自己遍历分析树。下面提供了一段Java示例程序供参考学习。这种方法的好处与缺点和第一种刚好相反。自由度更高，自行遍历可以获取到每个节点的属性，并进行一些通用的操作，代码会更精简。缺点是需要自己写递归函数来遍历分析树，对自身算法水平有要求。

|  |
| --- |
| IDLLexer idlL = null;  IDLParser parser = null;  CharStream input = CharStreams.fromString(Util.readFile(idlPath));  idlL = new IDLLexer(input);  CommonTokenStream tokens = new CommonTokenStream(idlL);  parser = new IDLParser(tokens);  ParseTree tree = parser.specification();  if (parser.getNumberOfSyntaxErrors()>0)  throw new SyntaxException(String.format("有 %s 处语法错误，请根据报错信息进行修改后再次尝试 !",parser.getNumberOfSyntaxErrors()));  /\* 渲染所有的stg模板并生成 \*/  context = buildIDLTree(tree,parser);  for (STGroup stg:  stgs) {  targetCodes.add(\_generateTargetCode(context,stg,semanticStg));  }  return targetCodes; |
| public static IDLTreeNode buildIDLTree(ParseTree t,Parser parser) throws NullPointerException{  /\* 错误节点 \*/  if ( t instanceof ErrorNode) {  return null;  }  /\* 终结符节点 \*/  else if ( t instanceof TerminalNode) {  String tokenType = tokensTypeCode.get(((TerminalNode) t).getSymbol().getType());  IDLTreeNode tNode = new IDLTreeNode(tokenType,"T",Trees.getNodeText(t,parser),null,null);  /\* 加入到上下文环境中去 \*/  tNode.setHashCode(tNode.hashCode());  contextMap.put(tNode.hashCode(),tNode);  return tNode;  }  /\* 非终结符节点 \*/  IDLTreeNode root = new IDLTreeNode(Trees.getNodeText(t,parser),"NONT",null,null,null);  root.setHashCode(root.hashCode());  /\* 加入到上下文环境中去 \*/  contextMap.put(root.hashCode(),root);  ArrayList<IDLTreeNode> childs = new ArrayList<IDLTreeNode>();  /\* 对分析树递归 \*/  RuleNode r = (RuleNode)t;  int n = r.getChildCount();  for (int i = 0; i<n; i++) {  ParseTree rc = r.getChild(i);  IDLTreeNode child = buildIDLTree(rc,parser);  childs.add(child);  /\* 为孩子节点设置父亲节点 \*/  child.setPnode(root);  }  /\* 为当前节点设置孩子节点列表 \*/  root.setNodes(childs);  return root;  } |

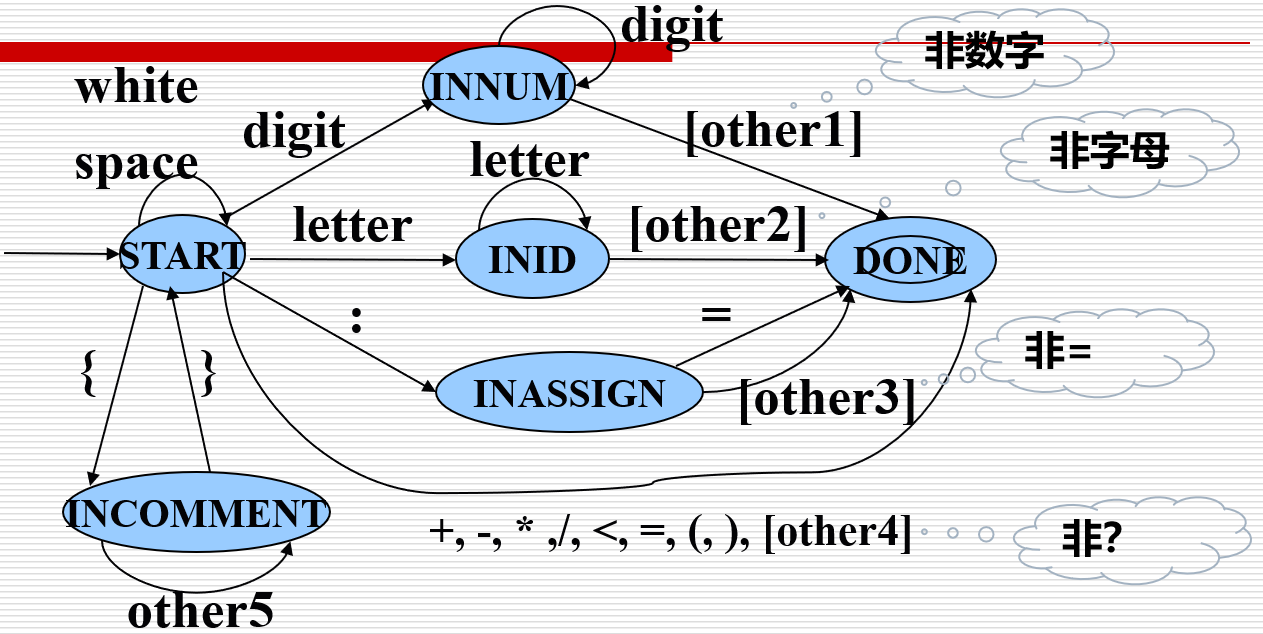
其中对于tokensTypeCode存的是antlr4生成的IDL.tokens 文件中的token与token id的字典。

# 附件2：手工撰写词法分析程序参考案例

## 1、Tiny语言的词法规则定义



Tiny语言的词法规则对应的确定性有穷自动机模型：



## 2、基于自动机模型的Tiny语言的词法程序

Scan.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File: scan.h \*/

/\* The scanner interface for the TINY compiler \*/

/\* Compiler Construction: Principles and Practice \*/

/\* Kenneth C. Louden \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_SCAN\_H\_

#define \_SCAN\_H\_

/\* MAXTOKENLEN is the maximum size of a token \*/

#define MAXTOKENLEN 40

/\* tokenString array stores the lexeme of each token \*/

extern char tokenString[MAXTOKENLEN+1];

/\* function getToken returns the

\* next token in source file

\*/

TokenType getToken(void);

#endif

Scan.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File: scan.c \*/

/\* The scanner implementation for the TINY compiler \*/

/\* Compiler Construction: Principles and Practice \*/

/\* Kenneth C. Louden \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "globals.h"

#include "util.h"

#include "scan.h"

/\* states in scanner DFA \*/

typedef enum

{ START,INASSIGN,INCOMMENT,INNUM,INID,DONE }

StateType;

/\* lexeme of identifier or reserved word \*/

char tokenString[MAXTOKENLEN+1];

/\* BUFLEN = length of the input buffer for

source code lines \*/

#define BUFLEN 256

static char lineBuf[BUFLEN]; /\* holds the current line \*/

static int linepos = 0; /\* current position in LineBuf \*/

static int bufsize = 0; /\* current size of buffer string \*/

static int EOF\_flag = FALSE; /\* corrects ungetNextChar behavior on EOF \*/

/\* getNextChar fetches the next non-blank character

from lineBuf, reading in a new line if lineBuf is

exhausted \*/

static int getNextChar(void)

{ if (!(linepos < bufsize))

{ lineno++;

if (fgets(lineBuf,BUFLEN-1,source))

{ if (EchoSource) fprintf(listing,"%4d: %s",lineno,lineBuf);

bufsize = strlen(lineBuf);

linepos = 0;

return lineBuf[linepos++];

}

else

{ EOF\_flag = TRUE;

return EOF;

}

}

else return lineBuf[linepos++];

}

/\* ungetNextChar backtracks one character

in lineBuf \*/

static void ungetNextChar(void)

{ if (!EOF\_flag) linepos-- ;}

/\* lookup table of reserved words \*/

static struct

{ char\* str;

TokenType tok;

} reservedWords[MAXRESERVED]

= {{"if",IF},{"then",THEN},{"else",ELSE},{"end",END},

{"repeat",REPEAT},{"until",UNTIL},{"read",READ},

{"write",WRITE}};

/\* lookup an identifier to see if it is a reserved word \*/

/\* uses linear search \*/

static TokenType reservedLookup (char \* s)

{ int i;

for (i=0;i<MAXRESERVED;i++)

if (!strcmp(s,reservedWords[i].str))

return reservedWords[i].tok;

return ID;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* the primary function of the scanner \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* function getToken returns the

\* next token in source file

\*/

TokenType getToken(void)

{ /\* index for storing into tokenString \*/

int tokenStringIndex = 0;

/\* holds current token to be returned \*/

TokenType currentToken;

/\* current state - always begins at START \*/

StateType state = START;

/\* flag to indicate save to tokenString \*/

int save;

while (state != DONE)

{ int c = getNextChar();

save = TRUE;

switch (state)

{ case START:

if (isdigit(c))

state = INNUM;

else if (isalpha(c))

state = INID;

else if (c == ':')

state = INASSIGN;

else if ((c == ' ') || (c == '\t') || (c == '\n'))

save = FALSE;

else if (c == '{')

{ save = FALSE;

state = INCOMMENT;

}

else

{ state = DONE;

switch (c)

{ case EOF:

save = FALSE;

currentToken = ENDFILE;

break;

case '=':

currentToken = EQ;

break;

case '<':

currentToken = LT;

break;

case '+':

currentToken = PLUS;

break;

case '-':

currentToken = MINUS;

break;

case '\*':

currentToken = TIMES;

break;

case '/':

currentToken = OVER;

break;

case '(':

currentToken = LPAREN;

break;

case ')':

currentToken = RPAREN;

break;

case ';':

currentToken = SEMI;

break;

default:

currentToken = ERROR;

break;

}

}

break;

case INCOMMENT:

save = FALSE;

if (c == EOF)

{ state = DONE;

currentToken = ENDFILE;

}

else if (c == '}') state = START;

break;

case INASSIGN:

state = DONE;

if (c == '=')

currentToken = ASSIGN;

else

{ /\* backup in the input \*/

ungetNextChar();

save = FALSE;

currentToken = ERROR;

}

break;

case INNUM:

if (!isdigit(c))

{ /\* backup in the input \*/

ungetNextChar();

save = FALSE;

state = DONE;

currentToken = NUM;

}

break;

case INID:

if (!isalpha(c))

{ /\* backup in the input \*/

ungetNextChar();

save = FALSE;

state = DONE;

currentToken = ID;

}

break;

case DONE:

default: /\* should never happen \*/

fprintf(listing,"Scanner Bug: state= %d\n",state);

state = DONE;

currentToken = ERROR;

break;

}

if ((save) && (tokenStringIndex <= MAXTOKENLEN))

tokenString[tokenStringIndex++] = (char) c;

if (state == DONE)

{ tokenString[tokenStringIndex] = '\0';

if (currentToken == ID)

currentToken = reservedLookup(tokenString);

}

}

if (TraceScan) {

fprintf(listing,"\t%d: ",lineno);

printToken(currentToken,tokenString);

}

return currentToken;

} /\* end getToken \*/

Global.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File: globals.h \*/

/\* Global types and vars for TINY compiler \*/

/\* must come before other include files \*/

/\* Compiler Construction: Principles and Practice \*/

/\* Kenneth C. Louden \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_GLOBALS\_H\_

#define \_GLOBALS\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#ifndef FALSE

#define FALSE 0

#endif

#ifndef TRUE

#define TRUE 1

#endif

/\* MAXRESERVED = the number of reserved words \*/

#define MAXRESERVED 8

typedef enum

/\* book-keeping tokens \*/

{ENDFILE,ERROR,

/\* reserved words \*/

IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,

/\* multicharacter tokens \*/

ID,NUM,

/\* special symbols \*/

ASSIGN,EQ,LT,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,SEMI

} TokenType;

extern FILE\* source; /\* source code text file \*/

extern FILE\* listing; /\* listing output text file \*/

extern FILE\* code; /\* code text file for TM simulator \*/

extern int lineno; /\* source line number for listing \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* Syntax tree for parsing \*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef enum {StmtK,ExpK} NodeKind;

typedef enum {IfK,RepeatK,AssignK,ReadK,WriteK} StmtKind;

typedef enum {OpK,ConstK,IdK} ExpKind;

/\* ExpType is used for type checking \*/

typedef enum {Void,Integer,Boolean} ExpType;

#define MAXCHILDREN 3

typedef struct treeNode

{ struct treeNode \* child[MAXCHILDREN];

struct treeNode \* sibling;

int lineno;

NodeKind nodekind;

union { StmtKind stmt; ExpKind exp;} kind;

union { TokenType op;

int val;

char \* name; } attr;

ExpType type; /\* for type checking of exps \*/

} TreeNode;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* Flags for tracing \*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* EchoSource = TRUE causes the source program to

\* be echoed to the listing file with line numbers

\* during parsing

\*/

extern int EchoSource;

/\* TraceScan = TRUE causes token information to be

\* printed to the listing file as each token is

\* recognized by the scanner

\*/

extern int TraceScan;

/\* TraceParse = TRUE causes the syntax tree to be

\* printed to the listing file in linearized form

\* (using indents for children)

\*/

extern int TraceParse;

/\* TraceAnalyze = TRUE causes symbol table inserts

\* and lookups to be reported to the listing file

\*/

extern int TraceAnalyze;

/\* TraceCode = TRUE causes comments to be written

\* to the TM code file as code is generated

\*/

extern int TraceCode;

/\* Error = TRUE prevents further passes if an error occurs \*/

extern int Error;

#endif

Util.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File: util.h \*/

/\* Utility functions for the TINY compiler \*/

/\* Compiler Construction: Principles and Practice \*/

/\* Kenneth C. Louden \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_UTIL\_H\_

#define \_UTIL\_H\_

/\* Procedure printToken prints a token

\* and its lexeme to the listing file

\*/

void printToken( TokenType, const char\* );

/\* Function newStmtNode creates a new statement

\* node for syntax tree construction

\*/

TreeNode \* newStmtNode(StmtKind);

/\* Function newExpNode creates a new expression

\* node for syntax tree construction

\*/

TreeNode \* newExpNode(ExpKind);

/\* Function copyString allocates and makes a new

\* copy of an existing string

\*/

char \* copyString( char \* );

/\* procedure printTree prints a syntax tree to the

\* listing file using indentation to indicate subtrees

\*/

void printTree( TreeNode \* );

#endif

# 附件3：手工撰写语法分析程序参考案例

## 1、Tiny语言语法规则定义

*program -> stmt-sequence*

*stmt-sequence ->  stmt-sequence;statement|statement*

*statement -> if-stmt|repeat-stmt;assign-stmt;read-stmt;write-stmt*

*if-stmt -> if exp then stmt- sequence end*

*| if exp then stmt- sequence else stmt-sequence end*

*repeat-stmt -> repeat stmt-sequence until exp*

*assign-stmt -> identifier := exp*

*read-stmt -> read identifier*

*write-stmt -> write exp*

*exp -> simple-exp comparison-op simple-exp | simple-exp*

*comparison-op -> <| =*

*simple-exp -> simple-exp addop term |term*

*addop -> +|-*

*term -> term mulop factor |factor*

*mulop -> \*|/*

*factor -> (exp)|number | identifier*

## 2、语法分析程序实现步骤

1. 将文法用EBNF表示

*program*→*stmt-sequence*

*stmt-sequence*→*statement*{ **；***statement* }

*statement*→*if-stmt | repeat-stmt | assign-stmt | read-stmt | write-stmt*

*if-stmt*→**if** *exp* **then** *stmt-sequence* **[ else** *stmt-squence* **] end**

*repeat-stmt*→**repeat** *stmt-sequence* **until** *exp*

*assign-stmt*→**identifier :=** *exp*

*read-stmt*→**read** **identifier**

*write-stmt*→**write** *exp*

*exp*→*simple-exp* **[** *comparison-op simple-exp* **]**

*comparison-op*→**<** | **=**

*simple-exp*→*term* { *addop term* }

*addop*→ **+**| **-**

*term*→*factor* { *mulop factor* }

*mulop*→ **\*** | **/**

*factor*→**(** *exp* **)** | **number** |**identifier**

1. 抽象语法树设计

*program*→*stmt-sequence*

stmt-sequence

*stmt-sequence→statement{ ；statement }*

statement

statement

statement

……

*if-stmt→if exp then stmt-sequence [ else stmt-squence ] end*

If

exp

stmt-sequence

stmt-sequence

*repeat-stmt*→**repeat** *stmt-sequence* **until** *exp*

*repeat*

*stmt-sequence*

*exp*

*assign-stmt*→**identifier :=** *exp*

*Assign（identifier）*

*exp*

*read-stmt*→**read** **identifier**

*read（identifier）*

*exp*

*write-stmt*→**write** *exp*

*write*

*exp*

*exp→simple-exp [ comparison-op simple-exp ]*

*comparison-op→****<*** *|* ***=***

*comparison-op(****<*** *|* ***=)***

*simple-exp*

*simple-exp*

*simple-exp→term { addop term }*

*addop→* ***+****|* ***-***

*Term*

*op(+|-)*

*(+|-*

*Term*

……

*term→factor { mulop factor }*

*mulop→* ***\**** *|* ***/***

*factor*

*mulop(\*|/)*

*(+|-*

*factor*

……

*factor*→**(** *exp* **)** | **number** |**identifier**

*exp*| **number** |**identifier**

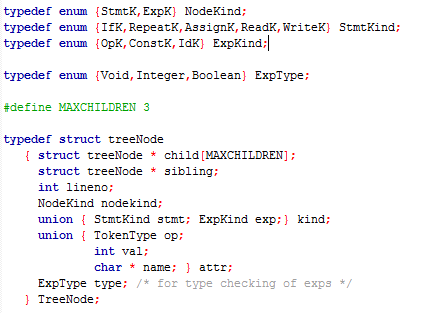
1. 定义数据结构

TINY有两种基本的结构类型：语句和表达式。

语句共有5类：（if语句、repeat语句、assign语句、read语句和read语句），

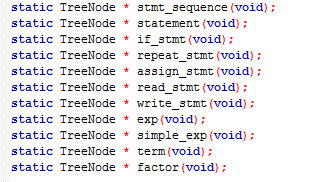
表达式共有3类（算符表达式、常量表达式和标识符表达式）。

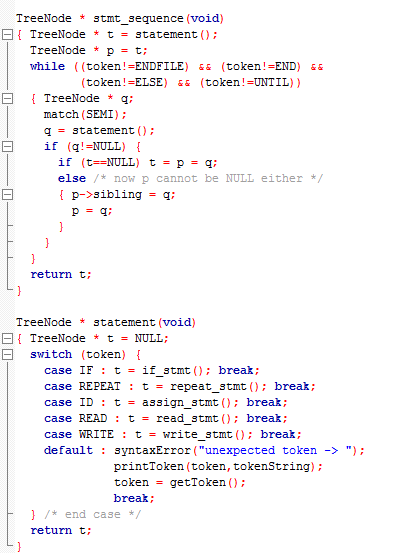
以C代码为例：

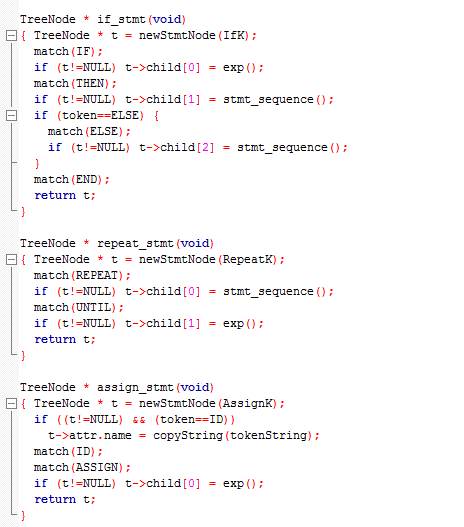


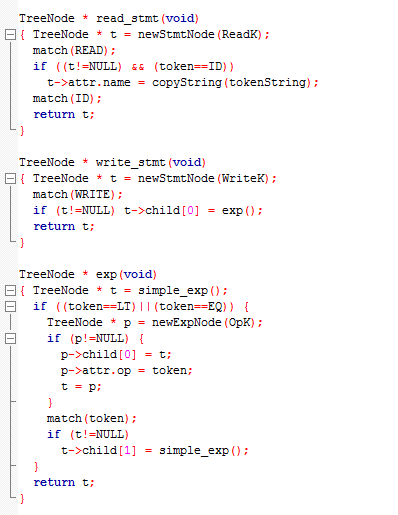
1. 实现递归下降分析程序

语法分析递归下降核心代码如下：









*simple-exp*→*term* { *addop term* }*3+4-5*

