编译原理课设需求分析

目录导航

1. 需求分析

1.1 目标

目的：设计并实现一个编译器，将Pascal代码转换成C代码。

用户：主要面向需要在C环境下运行Pascal程序的开发者，例如教育领域、遗留系统维护等。

1.2 功能需求

代码解析：能够解析标准Pascal代码，包括基本语法、结构、函数等。

类型转换：将Pascal数据类型转换为对应的C数据类型。

控制结构转换：将Pascal的控制结构（如 if, while, for, case 等）转换为C语言的等效结构。

错误处理：在源代码中检测到语法或语义错误时给出提示。

代码优化：生成的C代码应当是优化的，以提高执行效率。

1.3 性能需求

效率：转换过程应该高效，对于中等大小的程序（约1000行代码），在标准硬件上的处理时间应该在几秒内。

兼容性：生成的C代码应该能够在主流的C编译器上编译通过。

1.4 扩展需求

1. 对于字符串类型的支持
2. 对于注释格式的额外支持

2. 测试计划

2.1 测试环境

硬件：标准PC机，具有足够的内存和处理能力。

软件：Pascal编译器（用于生成测试数据）、C编译器（用于编译生成的C代码）、自动化测试工具等。

2.2 测试策略

单元测试：针对编译器中每个模块（如词法分析、语法分析、代码生成等）进行测试。

集成测试：将各个模块集成后进行的测试，确保模块间正确交互。

系统测试：在真实环境下对编译器进行的全面测试，包括功能测试、性能测试、稳定性测试等。

回归测试：在修改编译器后进行，确保修改没有引入新的错误。

2.2.1 单元测试

目的：确保编译器的每个独立部分（如词法分析器、语法分析器、代码生成器）按预期工作。

方法：为每个模块编写测试用例，覆盖各种输入场景，包括边界条件和异常情况。使用自动化测试框架来运行这些测试用例，以便于快速识别回归错误。

2.2.2 集成测试

目的：验证各个模块集成在一起后，编译器作为一个整体是否正常工作。

方法：设计测试用例，包含从简单到复杂的Pascal程序。这些程序应该涵盖Pascal语言的主要特性和构造。比较编译器输出的C代码和预期的结果，确保正确性。

2.2.3 系统测试

目的：模拟用户环境，确保编译器在实际使用中能够满足要求。

方法：从Pascal程序编写、编译到最终在目标环境中运行C程序，整个流程都需要被测试。包括对生成的C代码的编译、运行和输出结果的验证。

2.2.4 回归测试

目的：在对编译器修改之后再次对其进行测试，及时发现并解决新的bug。

方法：在每一阶段的更改之后，对更改部分的输入输出进行测试，确保其读取与输出合理合法，能够与其余部分形成联系。

2.3 维护和迭代

反馈收集：从用户那里收集反馈，了解编译器在实际使用中的表现，包括功能上的缺陷、性能问题以及用户体验的改进建议。

持续改进：根据用户反馈和测试结果，定期更新编译器，修复已知问题，引入新的功能和性能优化。

文档更新：随着编译器的更新，及时更新用户文档和开发者指南，确保文档反映了编译器的当前状态。

通过这种结构化和细化的方法，可以有效地设计和实现一个从Pascal到C语言的编译器，同时确保其质量和性能符合预期。

3. 测试用例

3.1 基础语法转换

测试用例1：基本变量声明和赋值。

测试用例2：控制结构转换，包括if-else、while循环等。

测试用例3：函数定义和调用。

3.2 复杂结构转换

测试用例4：数组和记录（record）的转换。

测试用例5：指针操作和动态内存管理。

测试用例6：模块和单元的转换。

3.3 测试用例生成器

测试用例生成器用以生成较长的测试样例，便于对编译器在多个维度进行测试，同时也能很方便地检验在较长输入的情况下，编译器的编译效率与准确性。此生成器由以下几个模块组成：

- 权值生成：根据输入的数字，来决定随机生成int、real、boolean、char类型的数据（0为int，1为real，2为boolean，3为char）。
- 变量及函数声明: 该模块实现变量的声明，随机生成变量名，然后随机为其赋为int、real、boolean、char。同时，函数名的生成也由这里实现。
- 结构生成: 此模块可以随机生成出最基本的Pascal函数。多次调用此代码，即可实现较长Pascal代码的构建，此模块中每次生成的函数，其函数名由变量及函数声明模块随机生成。
- 变量存储：此部分并非用于存储变量的值，而是作为结构生成模块的辅助。由于循环和 if 函数之中声明的函数无法被外界调用，所以此结构用于存储可以被当前层次代码调用的变量名。同时。已声明的函数名也记录在此，方便调用。
- 主体: 此模块用以连接其他模块，通过可以调控的调用次数，实现可控的多次随机调用结构生成模块部分的函数，从而在输出文件中得到完整的、可以执行的Pascal代码。

4. 总体设计

4.1 模块划分

编译器主要由以下几个模块组成：

- 词法分析：将源代码分解为一系列的记号（tokens）。

- 语法分析：根据记号构建抽象语法树（AST），并对可能存在的语法错误进行处理。
- 语义分析：根据AST完成符号表的建立，实现对符号表的操作、类型检查、转化、作用域识别的功能并且输出语义错误信息。
- 代码生成：将AST转换为C语言代码。
- 测试用例生成：生成Pascal-S测试用例

4.2 工具和技术

编程语言：推荐使用C或C++进行编译器开发，因为它们提供了良好的性能支持和底层控制能力。

开发工具：语法分析及语义分析利用Bison来实现语法分析程序。

开发环境：选择一个支持多语言编程和调试的集成开发环境（IDE），如Visual Studio或CLion，以便于开发和调试。

版本控制：使用Git进行版本控制，以管理代码的版本历史和协作开发。

4.3 关键技术点

词法分析技术：利用正则表达式匹配语言的词法规则，分解文本为记号流。

语法分析技术：采用如LL或LR解析器生成器来构建语法分析器，用于生成抽象语法树（AST）。

AST转换和优化：在AST上应用各种转换和优化技术，提高代码质量和执行效率。

目标代码生成：遍历AST，生成等价的C代码，需要考虑如何有效地映射Pascal的特性到C语言的特性上。

4.4 开发方法

1. 模块化设计

将编译器设计为若干独立、可互换的模块，如词法分析器、语法分析器、语义分析器、代码生成器等。每个模块负责处理编译过程中的一部分任务，并提供清晰定义的接口与其他模块通信。

好处：简化了问题的复杂性，提高了代码的可维护性和可测试性。

2. 测试驱动开发（TDD）

在编写实际的编译器代码之前，先编写测试用例。然后以通过这些测试用例为目标来开发功能，直至所有测试用例通过。

好处：确保代码的可靠性，及时发现并修正错误，同时也促进了模块化设计。

3. 持续集成（Continuous Integration, CI）

在开发过程中，频繁地将代码集成到主分支，并自动运行测试，确保新的更改不会破坏现有功能。

好处：提高软件质量，缩短反馈周期，加快发布速度。

通过采用这些设计模式和开发方法，Pascal到C语言编译器的设计将不仅能满足当前的需求，还能适应未来可能的扩展和变化，保证软件的长期可维护性和可靠性。

5. 实现细节

5.1 词法分析器

5.1.1 目标

将源代码文本分解为一系列的记号（tokens），如标识符、**保留字**（而非关键字）、常量、运算符等，并将识别到的记号按照一定的规则转换成记号流，并输出给语法分析器。每个记号包含了词法单元的类型和对应的属性信息。

> 在Pascal-S语言中，没有C中“关键字”的概念，取而代之的是“保留字”的概念。C中关键字的int, float所对应的integer, real在Pascal-S中是预定义标识符，也就是说，这些标识符也可以由用户来重新定义

5.1.2 实现方法

本模块使用Flex实现。Flex是一个自动化工具，旨在简化词法分析器的构建过程，通过预定义的正则表达式来匹配源代码中的词法单元。以下是使用Flex构建Pascal-S词法分析器的详细步骤：

5.1.2.1 定义记号

在 Flex 的 .l 文件中，首先需要定义记号的名称和对应的正则表达式。这些正则表达式描述了记号的模式，以及如何识别它们。例如：

```
DIGIT     [0-9]
LETTER    [a-zA-Z]
IDENTIFIER {LETTER}({LETTER}|{DIGIT})*
NUMBER     {DIGIT}+(\.{DIGIT}+)?
```

5.1.2.2 规则部分

这是.l文件的核心部分，定义了将应用于输入文本的正则表达式和相应的动作。每条规则由两部分组成：匹配模式（正则表达式）和执行的动作（C代码块）。例如：

```
{IDENTIFIER}   { printf("Identifier: %s\n", yytext); }
{NUMBER}       { printf("Number: %s\n", yytext); }
```

5.1.2.3 辅助代码

在 Flex 的 .l 文件中，可以包含一些辅助性的 C 代码，用于辅助词法分析器的工作。这些代码通常包括在花括号 `%{ ... %}` 中，并且可以包含一些声明、定义和其他辅助函数。例如：

```
%{
// 包含所需的头文件
#include <stdio.h>
#include <stdlib.h>

// 定义一个辅助函数
void report_error(const char *msg) {
    fprintf(stderr, "Error: %s\n", msg);
    exit(EXIT_FAILURE);
}
%}
```

5.1.3 关键任务

5.1.3.1 定义词法规则

首先，需要明确Pascal-S语言中的词法元素，包括：

- 保留字：and array begin case const div do downto else end for function if mod not of or procedure program record repeat then to type until var while := > < >= <= <> = + - * /
- 标识符：Pascal-S中不区分大小写

```
letter = a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
identifier = letter { letter | digit }
```

- **预定义标识符**：false true real char boolean integer **string** abs sqr odd chr ord succ pred round trunc sin cos exp ln sqrt crctan eof eoln read readln write writeln
- 数字：

```
 signed-number = signed-integer | signed-real
 signed-real = [ sign ] unsigned-real
 signed-integer = [ sign ] unsigned-integer
 unsigned-number = unsigned-integer | unsigned-real
 sign = '+' | '-'
 unsigned-real = digit-sequence '.' fractional-part [ 'e' scale-factor ] | digit-sequence 'e' scale-factor
```

```
unsigned-integer = digit-sequence
fractional-part = digit-sequence
scale-factor = [ sign ] digit-sequence
digit-sequence = digit { digit }
```

- 字符/字符串：

```
character-string = ''' string-element { string-element } '''
string-element = apostrophe-image | string-character
apostrophe-image = ''''
string-character = one-of-a-set-of-implementation-defined-characters
```

- 注释：

```
comment = ( '{' | '(*' ) commentary ( '*)' | '}' )
commentary可以是任何字符序列和行分隔符，但不包含{, }, (*, *)
```

- 分隔符：用于分隔记号，可以是空格符，行分隔符（回车、换行、分号），注释。

5.1.3.2 集成至编译器

生成的词法分析器代码需要与编译器的其他部分集成。具体来说，语法分析器将调用yylex函数来获取输入的记号，并根据这些记号进行语法分析。

5.1.3.3 处理词法错误

- 错误识别：
  在词法分析阶段，词法分析器需要能够识别并标识出输入文本中的词法错误。词法错误可能包括无效的字符、无法识别的词法单元、非法的字符串等。
- 错误报告：
  词法分析器应该能够报告识别到的词法错误，以便程序员能够及时发现并修复这些错误。错误报告通常包括错误的位置信息（行号、列号）、错误类型和相关的错误消息等。
- 恢复错误处理：
  当词法分析器遇到词法错误时，可以尝试进行一些恢复性的错误处理，以使词法分析器能够继续识别后续的输入文本。常见的错误恢复策略包括简单地跳过错误的字符或词法单元，直到找到一个合法的词法单元，或者使用特定的错误标记来标识错误的位置并继续分析。

5.1.4 扩展任务

我们决定对Pascal-S语法进行扩展，按照ISO 7185:1990标准支持Pascal相同的字符串类型与注释格式。在定义词法规则一节中所呈现内容已经包含了扩展后的Pascal-S语法。

5.2 语法分析器

5.2.1 目标

根据记号流构建抽象语法树（AST），AST以树形结构表示程序的语法结构。

确保语法分析器能够从词法分析器接收记号流，并正确处理记号和任何相关的词法信息（如字面量的值）。

构建的AST不仅应该反映程序的语法结构，还应该包含足够的信息以供后续的语义分析阶段使用，比如类型信息、作用域信息等。

5.2.2 实现方法

使用Bison根据Pascal-S语言的文法规则自动生成语法分析器。Bison接受一个文法规则文件，并产生一个能够解析该文法的C代码。

5.2.3 关键任务

1. 定义文法规则

在学习Pascal-S官方语法结构后，综合实验指导上的相关表达式以及我们自己制定的规则，编写精确的文法规则来描述Pascal-S语言的语法结构，包括所有的语法结构如程序结构、变量声明、控制流语句（如if、while、for等）、过程和函数声明等。下面为对于部分语法结构的说明，可能在后续开发过程中添加新的语法结构。

主程序：（1）主程序由程序名称标识符、参数列表、分程序和复合语句组成。

（2）主程序的参数列表与子函数/过程不同，无需指出标识符的类型。然而如果程序主体中用到了该参数，就必须在变量定义区中再定义一次，指明具体类型。

```
programstruct -> program_head ; program_body.
program_head -> program id ( idlist ) | program id
program_body -> const_declarations
                var_declarations
                subprogram_declarations
                compound_statement
idlist -> id | idlist , id
```

常量：（1）常量用const关键字来声明。

（2）常量定义需要指定初值，而不必指定类型。

（3）常量的初值可以由整数、浮点数、字符常量、别的常量标识符等指明，可以是一个可以由编译器计算出结果的常量表达式。

（4）由于PASCAL-S的常量定义不包含类型，其类型需要编译器判断。其产生式如下：

```
const_declarations -> ε | const const_declaration ;
const_declaration -> id = const_value | const_declaration ; id = const_value
const_value -> + num | - num | num | ' letter '
```

变量：（1）变量用var关键字来声明。

（2）变量定义需要指明类型，无需指定初值。

（3）变量也可以是一维或多维数组，数组的各维下标由表达式组成。

（4）变量也可以是记录类型的变量，其产生式如下：

```
var_declarations -> ε | var var_declaration ;
var_declaration -> idlist : type | var_declaration ; idlist : type
```

类型：支持 Pascal-S 语言的常见数据类型，包括整数、实数、布尔值和字符，扩展实现了字符串，record等数据类型 。其产生式如下：

```
type -> basic_type | array [ period ] of basic_type | recordtype | stringtype
basic_type -> integer | real | boolean | char
period -> digits .. digits | period ,  digits .. digits
recordtype -> record
          var_declaration
          end;
stringtype -> string [ num ] | string
def-record -> type
              record-name = record
                  var_declaration
              end;
```

子函数/子过程：函数和过程的头部中都包含名称和参数表，此外函数头还需指明返回值的类型，然后是子函数/过程的主体。初步设定子函数/过程中不能再嵌套函数/过程，在后续开发过程中可能进行进一步拓展。产生式如下：

```
subprogram_declarations -> ε | subprogram_declarations subprogram ;
subprogram -> subprogram_head ; subprogram_body
subprogram_head -> procedure id formal_parameter | function id
formal_parameter : basic_type
subprogram_body -> const_declarations
                   var_declarations
                   compound_statement
```

参数表：（1）参数列表由一系列的标识符、类型关键字组成的参数变量定义组成。

（2）PASCAL_S的参数有两种，引用和传值，其中引用调用需要包含var关键字。产生式如下：

```
formal_parameter -> ε | ( parameter_list )
parameter_list -> parameter | parameter_list ; parameter
parameter -> var_parameter | value_parameter
var_parameter -> var value_parameter
value_parameter -> idlist : basic_type
```

语句：复合语句部分主要包含过程调用，函数调用，程序的三种结构，表达式。

```
compound_statement -> begin statement_list end
statement_list -> statement | statement_list ; statement
statement -> ε
| variable assignop expression
| func_id assignop expression
| procedure_call
| compound_statement
| if-statement
| loop-statement
```

（1）过程调用，过程调用结构较为简单，如下所示：

```
procedure_call -> id | id ( expression_list )
```

（2）函数调用，由于Pascal-S拥有大量的库函数，我们将满足除了read()，write()更多的基本的库函数，如writeln()，exit()等。

```
func -> read ( variable_list )
| write ( expression_list )
```

variable_listd的结构如下：

```
variable_list -> variable | variable_list , variable
variable -> id id_varparts
id_varparts -> ε | id_varparts id_varpart
```

```
id_varpart -> . id | [ expression_list ]
```

（3）程序主要拥有三个结构，顺序结构、分支结构和循环结构。

顺序结构是由begin和end关键字及其包括的递归定义的复合语句块。

分支结构我们只支持if条件语句。（我们对于if条件语句嵌套结构的处理方式为ELSE与最近的并且未被配对的IF配对）

循环结构我们支持FOR语句、WHILE语句和REPEAT语句。

```
分支结构
if_statement -> if expression then statement else_part
else_part -> ε | else statement

循环结构
loop-statement -> for id assignop(:=) expression to expression do statement
| for id assignop expression downto expression do statement
| while expression do statement
| repeat statement until expression
```

（4）表达式

表达式是由一系列的操作符和操作数组成的，其定义一般是递归的。我们采用了实验指导中的分层方式。

最底层的因子概念，因子包括：单个常量，单个变量，由单目运算符与另一因子组成的新的因子，括号括起来的表达式，函数调用；

然后是项，项可以是一个因子，也可以是由*、/、div、mod、and等优先级较高的双目运算符和两个因子组成的

接下来是简单表达式的，简单表达式可以是一个项，也可以是由+、-、or等优先级较低的双目运算符和两个项组成。

最终是表达式，表达式可以是一个简单表达式，可以是由>、=、<、<=、>=、<>等优先级最低的关系双目运算符和两个简单表达式组成。

```
expression_list -> expression | expression_list , expression
expression -> simple_expression | simple_expression relop simple_expression
simple_expression -> term | simple_expression addop term
term -> factor | term mulop factor
factor -> num | variable
| ( expression )
| id ( expression_list )
| not factor
```

```
 9 | uminus factor
10 | record-id . record-member
11 | ' letter '
```

2. 构建AST

识别出语法结构后，语法分析器将根据这些结构构建AST。AST是代码逻辑和结构的抽象表示，每个节点对应于源代码中的一个构造，如一个表达式、一个语句或一个程序块。在后续的编译阶段，AST将被用于进一步的分析和最终的代码生成。

为了有效地构建和操作AST，并且我们采用C++进行程序编写，因此我们将定义AST节点类型，并为了相关接口的重复使用，以及规范化，我们将利用C++的继承特性完成不同节点类的定义。

初步AST设计如下：

- 一个Pascal-S的AST节点包括但不限于下列节点：

声明节点：表示变量、常量和类型的声明。

表达式节点：表示各种算术和逻辑表达式。

语句节点：表示赋值、控制流（如if、while、for）和过程调用等。

程序和模块节点：表示整个程序或模块的结构，是AST的根节点。

- 对于部分基本节点的属性定义如下：

2.1 Program

`program_head` 与 `program_body` :指针，指向主程序的头节点与主体节点。

2.2 ProgramHeader

`id` : 字符串，程序标识符。

`parameters` : 指针，指向 `Parameter` 节点，程序参数列表。

2.3 ProgramBody

`declarations` : 指针，指向 `Declaration` 节点，主程序中的所有声明。

`subprogram_declarations` ： 指针，指向子程序节点。

`statements` : 指针，指向 `Statement` 节点，主程序中的所有语句。

2.4 Declaration：

`kind` : 字符串，声明类型（ `const` , `var` , `type` , `function` , `procedure` ）。

`identifiers` :字符串，声明的标识符。

`type` : 字符串或 `TypeDeclaration` 节点，声明的类型。

`initialValue` : 表达式，可选，初始化值。

2.5 Statement：

`type`：字符串，语句类型（`assignment`，`procedureCall`，`functionCall`，`compound`，`if`，`loop`，`exit`）。

`target`：字符串或`Identifier`节点，赋值目标。

`expression`：`Expression`节点，表达式。

`sequence`：指针，指向包含`Statement`节点，语句序列。

2.6 Expression：

`type`：字符串，表达式类型（`simple`，`term`，`factor`，`binaryOperation`，`unaryOperation`）。

`operator`：字符串，运算符（如`+`，`-`，`*`，`/`，`div`，`mod`，`and`，`or`，`not`）。

`operands`：指针，指向包含`Expression`节点，操作数。

2.7 ControlFlow：

`type`：字符串，控制流类型（`if`，`for`，`while`，`repeat`）。

`condition`：指针，指向`Expression`节点，条件表达式。

`body`：指针，指向`Statement`节点，循环或条件体。

`elseBody`：指针，指向`Statement`节点，`else`分支体。

上述定义为部分基本节点的粗略定义，在实际开发过程中，我们将进行更详细的设计，由于节点数较多，暂时无法给出所有的定义。

3. 错误恢复

在语法分析过程中，处理和报告语法错误是一个重要方面。当输入的记号序列不能被文法规则所接受时，需要有机制来识别这些错误，并给出有意义的错误信息，帮助开发者定位和修正代码中的问题。

语法分析程序进行错误处理的基本目标如下：

1. 能够清楚而准确地报告发现的错误，如错误的位置和性质。
2. 能够从错误中恢复过来，以便继续诊断后面可能存在的错误。
3. 错误处理功能不应该明显地影响编译程序对正确程序的处理效率。

对于在语法分析阶段出现的错误大致的处理方案如下：

1. 程序识别失败，遇到该错误时报错并终止语法分析程序
2. 出现非法记号，遇到该错误时报错并跳过该非法记号继续分析。
3. 非终结符号识别失败，如不完整的函数头、过程头，不完整的参数列表，不完整的数组下表列表，错误的保留字等，遇到该错误时报错并继续分析。
4. 标点符号缺失，遇到该错误时报错并继续分析。

5. 运算符号缺失，遇到该错误时报错并继续分析。
6. 常数初始化右值缺失，遇到该错误时报错并继续分析。

对于语法错误的恢复功能暂时还未设计，在后续开发过程中可能会实现对简单的错误进行自动恢复的功能。

4. 测试和验证

通过一系列设计好的测试用例，验证语法分析器的正确性和健壮性。测试用例应该覆盖Pascal语法的各个方面，包括各种有效和无效的语法结构，以确保分析器能够正确识别合法代码并恰当地报告语法错误。

对于语法分析板块的单元测试，由于lex与yacc需要共同使用，将在词法分析模块完成之时进行共同的测试，我们将尽可能考虑各种情况来验证语法分析程序对于正确的语法以及错误的语法的正确识别。我们也将设计语法树的打印接口，更方便找到语法分析板块中的问题。

在测试用例生成板块完成之后我们能进行更全面的测试。

5.3 语义分析

5.3.1 目标

在语法分析的基础上，进一步检查程序的语义正确性。

输入是语法树，输出是语义错误信息（错误的性质和位置）和符号表，主要完成符号表的建立和操作、类型检查与转化、作用域识别等方面的内容。

5.3.2 实现方法

在语法分析建立AST的过程中同时建立符号表，随后遍历AST，利用符号表中的信息对每个节点进行语义规则的检查，并输出语义错误。

5.3.3 关键任务

为了正确判断出程序中出现的语义错误，必须深入理解目标编程语言（如Pascal）的语义规则。这包括但不限于变量和函数的作用域规则、类型系统、表达式的合法性、过程和函数调用的参数匹配等。

1. 建立符号表

符号表是在语义分析过程中用于跟踪标识符声明的数据结构。

当进入一个新的作用域（如函数体）时，会创建一个新的符号表或一个符号表的层级，用于存储当前作用域内的所有声明。每当遇到一个标识符声明时，就将其添加到当前符号表中；每当遇到一个标识符引用时，就在符号表中查找以确认其声明。也就是说在每个程序块的入口建立一个符号表的子表，将该块的声明的所有标识符属性记录到该表中。在每个程序块的出口返回到主符号表，不再引用已经执行完的块中声明的局部变量。

对于符号表的基本设计如下：

主符号表要求记录：

- 种类标志
- 标识符名字
- 行号
- 列号
- 类型
- 常量取值
- 参数个数/数组维数/字符串字符数
- 指向record成员的指针
- 数组各维上下界
- 指向函数/过程子符号表的指针

由于C语言程序不支持函数和过程的嵌套定义，我们暂定我们所编译的Pascal-S程序也不支持函数和过程的嵌套定义，因此子符号表由于不支持函数和过程的嵌套定义，比起主符号表少了函数/过程相关的域。子符号表要求记录：（可能在后续开发过程中进行拓展）

- 种类标志
- 标识符名字
- 行号
- 列号
- 类型
- 常量取值
- 数组维数/字符串字符数
- 指向record成员的指针
- 数组各维上下界
- 指向原函数符号表的指针

上面各个属性的基本作用如下：

- 标识符名字：主要用于添加、查找、修改等操作。
- 行号/列号：记录下每个符号的行号/列号，用于报错时需要的位置信息。
- 类型：对于变量和常量来说，该域记录着变量或常量类型；

  对于数组来说，该域记录着数组元素的类型；

  对于函数来说，该域记录着函数返回值类型。
- 常量取值：保存常量的取值，以便后续计算、常数传播、检查除0错误、检查数组下标越界。
- 参数个数/数组维数：对于数组类型的变量，我们将存储其维数；

  对于函数类型，我们将存储其参数个数；

  对于字符串类型，我们将记录其长度。
- 数组各维上下界：对于数组，需要记录其各维上下界，用于判断是否存在越界。
- 指向record成员的指针：用于record类型变量，该指针指向该记录所包含的全部成员。
- 指向函数/过程子符号表的指针：在该指针域中保存该符号表中指向函数/过程子符号表的指针，便于进行定位和重定位处理。
- 指向原函数符号表的指针：在该指针域中保存该符号表中指向父函数符号表的指针，便于进行作用于的判断。
- 符号表的具体数据结构暂时还未设计，由于未来可能支持嵌套作用域，我们的主符号表与子符号表之间采用栈结构。

2. 类型检查及转换

类型检查是语义分析中的一个核心任务，确保所有的操作和表达式都是类型安全的。例如，不应该允许将整型变量赋值给字符串类型的变量，除非定义了明确的类型转换规则。类型检查过程需要遍历AST，对每个节点应用类型规则，确保操作符的操作数类型正确，赋值操作的左右两边类型一致等。

对于类型转化，我们仅支持隐式类型转化，例如从integer类型到real类型的隐式类型转化。

对于表达式类型检查，其可能涉及到基本类型的检查（如整数、浮点数、布尔值等）、复合类型的兼容性检查（如数组、记录等）。

根据每个运算符对于操作数的类型的不同要求，我们会对于各运算符的操作数类型进行检查，若遇到类型不合要求的操作数会进行警告。例如relop运算符要求两个操作数类型一致，在两个操作数无法隐式转换为相同类型的时候会进行警告。

```
1.1 < 2 (可以进行隐式类型转换)
1.1 < 'string' (无法进行类型转换，报错)
```

函数调用时实参和形参类型的匹配也是类型检查的重点，我们同样支持一定程度的隐式类型转换。

对于语句类型检查，由于不同的语句有不同的要求，需要具体分析。比如赋值语句就要求左值和右值的类型一致，if语句的条件表达式的类型为”bool”，而不能是其它类型的条件表达式。

```
var num : real;
num = 'letter' (无法将字符串赋值给实数变量)

if 2 then statement (条件表达式的类型为integer，不是bool，会进行报错)
```

3. 作用域和生命周期

根据Pascal的作用域规则，检查变量和函数的声明和使用，确保它们在适当的作用域内。作用域解析和绑定过程负责确定程序中每个标识符的作用域，以及将每个标识符引用与其对应的声明绑定。实现方法为在遍历AST时，维护一个符号表来记录当前作用域内所有声明的标识符及其属性。

我们暂定我们设计的PASCAL-S语法不支持函数/过程的嵌套定义，因此定义在PASCAL程序一开始的常量、变量就可以是视作是全局作用域，可以被主程序体和子程序体引用。（开发过程中可能进行调整）

按照我们当前的设定每一个子函数/过程中定义的常量、变量的作用域就局限在当前函数/过程。

检测标识符未定义错误时，需退回到全局作用域中。

检测标识符重定义错误时，只需要在局部作用域中检查。

4. 错误报告以及处理

语法分析最终会提供清晰准确的错误和警告信息，会尽可能详细，能够保证提供错误的类型、位置（行号和列号）、尽力提供可能的修正建议。

基本的错误类型如下所示：

- 类型错误：包括类型不匹配的赋值、错误的操作数类型（如将整型用于逻辑运算）、函数调用时实参和形参类型不匹配等情况。
- 作用域和声明错误：尝试访问未声明的变量、重复声明变量、在不正确的作用域内访问变量等。
- 函数和过程调用错误：包括错误数量的参数、类型不匹配的参数等。

5. 测试和验证

测试语义分析器的正确性和健壮性涉及到编写一系列的测试程序，这些程序应该包括合法的代码示例以确保它们被正确分析，以及故意包含各种语义错误的代码示例来验证错误检测和报告机制的有效性。

对于建立符号表，我们会定义符号表输出接口，用于直观的检查符号表的正确性。

对于语义错误报告，测试用例会至少包含上述错误类型，对于上述基本错误类型我们会在开发过程中进行小型单元功能测试，没有必要提前进行编写，而最终的对于整个程序的测试将会进一步测试语义错误输出的功能。

5.4 代码生成

5.4.1 设计模式

访问者模式（Visitor Pattern）

- **应用场景：**对AST的遍历和操作。由于AST包含多种不同类型的节点，对这些节点进行操作（如类型检查、代码生成等）时，如果采用传统的方法，将需要在每个节点类中实现多种操作，这会导致代码的重复和难以维护。
- **好处：**通过将操作逻辑从对象结构中分离出来，使得在不修改对象结构的情况下，能够向对象结构中添加新的操作，提高了系统的扩展性。

5.4.2 目标

- 将AST转换为等效的C代码，这是编译过程的最终阶段，目的是产生一个可以被C编译器进一步处理的源代码。

5.4.3 实现方法

- 实现一个遍历AST的访问者，对每种类型的节点应用特定的代码生成逻辑。

5.4.4 核心

- **遍历AST**

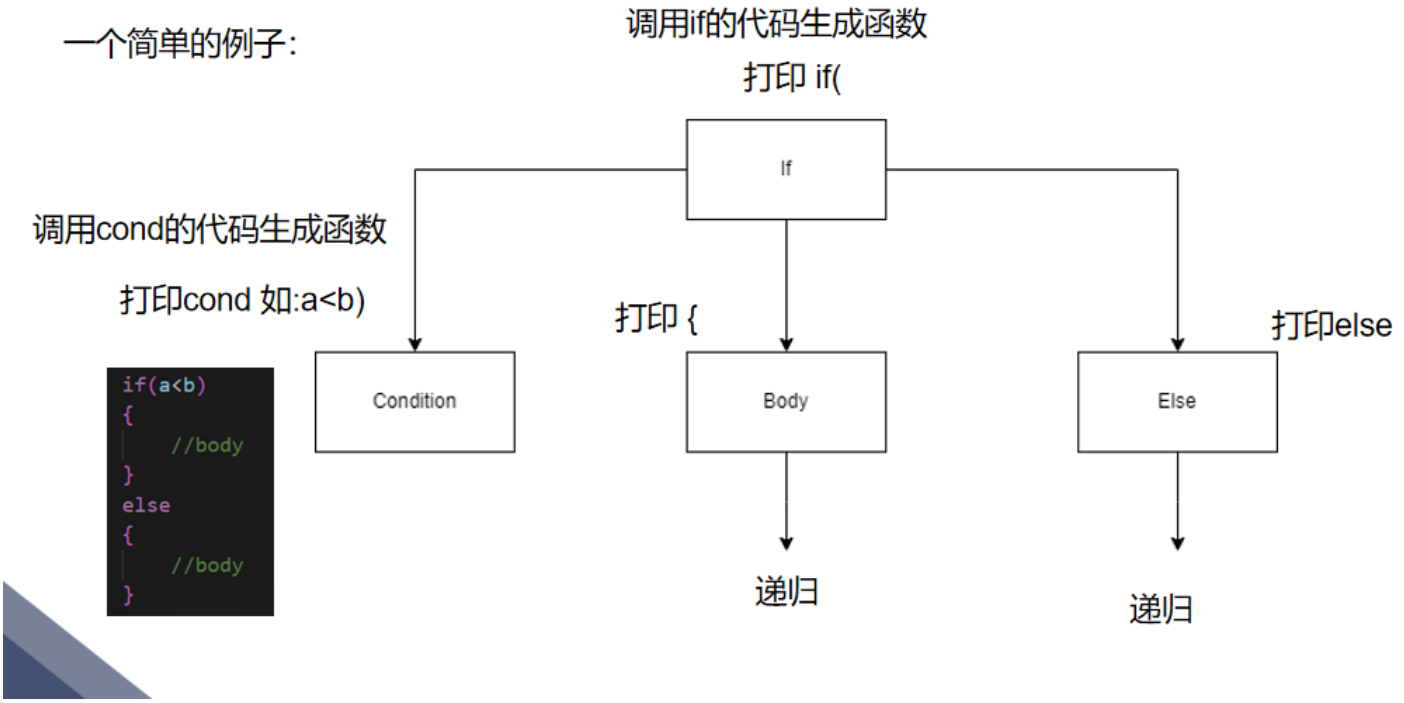
对于AST的遍历可以采用**深度优先遍历（DFS）**算法。

DFS可以确保按照源代码的结构顺序生成目标代码，比较符合代码生成的需求。

在深度优先遍历中，可以按照以下步骤进行：

1. 从根节点开始遍历AST。
2. 对于每个节点，先处理子节点，再处理自身节点。
3. 遍历过程中，根据节点的类型生成相应的目标代码。

这种遍历方式可以保证生成的目标代码结构正确，并且可以较好地处理复杂的语法结构。在遍历过程中，可以根据需要进行一些优化，比如记录变量的作用域信息、避免重复生成相同的代码等。



5.4.5 关键任务：

1. 理解源语言和目标语言的特性

深入理解源语言和目标语言（Pascal-S和C）的特性是代码生成的前提。这包括数据类型、控制结构（如循环、条件判断）、函数调用约定等。了解这些特性有助于生成符合目标语言习惯且高效的代码。

例如：

- C语言中的不等于用!=，而PASCAL中用<>。
- pascal在声明变量时，除了要说明类型，还要再前面加上var保留字，C语言中没有这样的关键字，只需要指明类型即可。
- pascal中的const保留字作用域较大，不局限于下一个分号，而C语言中，每一分号隔出的部分都要单独使用一个const关键字。且pascal中的常量在定义时不需要指明类型，但是C语言需要。
- C语言中数组各维下标默认从0开始，而pascal中的范围可以任意指定，因此需要对数组下标进行相应的变换（在目标代码中需要新增定义临时变量以指明偏移量）
- pascal中如果函数或者过程没有参数，则无需包含空内容的括号，C语言中则需要，且PASCAL中函数返回值用 函数标识符:=表达式 表示，而C语言使用return语句。
- pascal语句块使用begin和end，而C语言使用{}。

2. 代码生成过程

变量和函数声明

- Pascal中的变量和函数声明需要转换为C语言的等价声明。这涉及到类型转换和初始化值的生成。

表达式转换

- 算术和逻辑表达式在两种语言中有着类似的表示，但仍需要注意操作符的差异和类型转换规则。

控制流结构的转换

- Pascal的控制流结构（如if, while, for等）需要转换为C语言的等价结构。这通常是直接映射的过程，但要注意循环控制和条件表达式的语法差异。

函数和过程的处理

- Pascal的函数和过程调用需要转换为C语言的函数调用。这包括参数列表的转换和返回值的处理。对于Pascal的按引用（var参数）调用，需要生成指针参数和相应的地址传递代码。

优化和清理

- 在生成目标代码的过程中，可以实施一些简单的优化，如消除冗余的临时变量、简化表达式等。同时，需要确保生成的代码是整洁和可读的，这对于后续的调试和维护非常重要。

标准库的使用

- Pascal的一些标准库功能（如输入/输出处理）需要转换为C语言的标准库调用。这可能需要一个运行时支持库，以提供那些在C标准库中没有直接等价物的Pascal功能。

5.4.6 对应关系具体分析

1. 头文件添加

- 头文件的需求来自于库程序的调用，PASCAL-S仅支持read、write、writeln、exit四种库程序的调用，其中exit对应了返回值语句，所以剩下的三个过程对应到C程序就是printf和scanf这两个语句，这两个语句包含在**stdio.h**头文件中。我们将在所有生成函数调用、过程调用代码的位置调用一个函数专门来检查所调用和函数和过程是否是库程序，如果是库程序，则需标记相应的头文件。最后输出代码的时候，输出被标记了的头文件即可。
- 这样设计而不是对read、write、writeln进行特判，保证了我们编译器的可扩展性，后续如果要添加对别的库程序的支持，对于代码的修改将会特别容易进行。
- 另外，在C99中C语言加入了对bool类型的支持，但仍需加入**stdbool.h**头文件，考虑到boolean类型是PASCAL-S程序的基本类型，生成的C语言代码中，将自动加入stdbool.h头文件。

2. 程序结构转换

- PASCAL-S程序的主程序名在C语言中没有对应的内容，因为在C语言中，入口函数固定名称为main。我们需要在C程序中专门设置一个程序，取名为PASCAL程序的主程序名。然后main函数只要简单的调用该函数即可。
- PASCAL-S程序中，定义在主程序部分的常量和变量可以被所有的子程序引用，这相当于C语言中的全局变量和全局常量。

- 结构良好的C程序需要将所有程序的声明放在main函数之前，这些程序的声明中，除了包括PASCAL-S程序中声明的所有子程序，还应包括我们在上文提到的PASCAL-S主程序对应到C程序的声明。

3. 类型保留/关键字

类型	PASCAL-S预定义标识符	C关键字
整型	integer	int
浮点型	real	float
字符型	char	char
布尔型	boolean	bool
记录/结构体	record	struct
字符串	string[n]/string	char[n]/char[255]

4. 运算符

运算符	PASCAL-S保留字	C符号
加法	+	+
减法	-	-
乘法	*	*
除法	/	/
整除	div	/
取余	mod	%
与	and	&&
或	or	||
非	not	!
大于	>	>
大于等于	>=	>=
小于	<	<

小于等于	<=	<=
等于	=	==
不等于	<>	!=
赋值	常量初始化时为= 赋值语句中为:=	=

5. 引用参数与指针

- PASCAL-S支持引用参数，C不支持，所以在C中只能用指针来代替参数引用。
- 首先在调用程序时，形参变量前需加入取地址符；其次在程序定义时，需将引用参数的定义改为对应类型的指针；在程序体内，所有涉及到原引用参数的，都需要加上解引用符；还需特殊考虑引用参数又作为实参调用，同时对应的形参也是引用参数的情形，此时实参直接为该指针变量即可，不需要解引用符，当然也可以解引用后再取地址。

6. 主程序参数列表

- pascal主程序头中包含了一个无类型的标识符列表。这个标识符列表类似于c语言中main函数的参数列表，但由于pascal-s缺少相应的语法支持，导致主程序参数没有任何实际用途，所以我们在代码生成阶段可以将其忽略。

7. 返回语句

类型	PASCAL-S的返回语句	C的返回语句
函数返回值语句	函数名:=表达式;	return (表达式);
	exit(表达式);	return (表达式);
过程返回语句	exit;	return;

8. 语句块

- PASCAL-S中复合语句块由BEGIN和END保留字包括，而C语言中则由一对大括号包括。
- PASCAL-S中，复合语句块的最后一条语句后没有分号，所以在PASCAL-S中，即使看上去只有一条语句，如果这条语句后面出现了一个分号，那这一条语句也必须用BEGIN和END包括为一个复合语句块，if语句的then语句部分就是一个典型的例子；**但是C程序的每一条语句都必须包含分号。**

9. 循环语句

PASCAL-S循环语句	C循环语句

for 循环变量 := 初值表达式 to 终值表达式 do 循环语句体	for(循环变量 = 初值表达式 ; 循环变量 <= 终值表达式 ; 循环变量++)循环语句体
while 条件表达式 do循环语句体	while(条件表达式) do循环语句体
repeat循环语句体until 条件表达式	do循环语句体while(条件表达式)

10. 分支语句

在if分支语句上，两者并没有多大的差别。

PASCAL-S中的结构为：

```
1   if 条件表达式 then 语句;
2   if 条件表达式 then 语句1 else 语句2;
```

C中的结构为：

```
1  if(条件表达式) 语句;
2  if(条件表达式) 语句1 else 语句2;
```

11. 输入输出方式

- PASCAL-S调用read进行输入，C调用scanf进行输入；
- read在调用时，只需要提供用逗号分隔的普通变量（或数组元素）列表即可，而scanf不仅需要变量列表，还需一个格式控制符字符串。

类型与其格式控制符的对应关系如下：

类型	格式控制符
int	%d
float	%f
char	%c

PASCAL-S不支持boolean类型变量的输入。

- PASCAL-S调用write或writeln进行输出，C调用printf进行输出；
- write在调用时，只需要提供用逗号分隔的表达式列表即可，而printf不仅需要表达式列表，还需一个格式控制符字符串。

- writeln除了需要在格式控制符字符串的最后加入一个换行符，与write没有任何区别。

另外还需强调输出对于boolean变量的支持。**PASCAL-S支持boolean表达式的输出，在PASCAL-S中，输出布尔表达式的值时，输出的就是true或者false**

c语言使用的库函数要首先#include 所以要先遍历一遍pascal代码找出哪些库函数需要

5.5 测试用例生成

1. **权值生成：**
   - 目标：根据输入的整数决定随机生成对应数据类型（int, real, boolean, char）的随机数据。
   - 方法：接受一个整数输入，对4取模，结果决定数据类型。使用一个简单的映射逻辑：{0: 'int', 1: 'real', 2: 'boolean', 3: 'char'}，然后根据类型，为其生成合理的随机的数据。

```
std::string generateType(int input) {
    std::vector<std::string> types = {"int", "real", "boolean", "char"};
    int index = input % 4;

    if(types[index]为int)
        return 随机生成的int类型数据（以字符串的形式）;
    else if(types[index]为real)
        return 随机生成的real类型数据（以字符串的形式）;
    else if(types[index]为boolean)
        return 随机生成的boolean类型数据（以字符串的形式）;
    else
        return 随机生成的char类型数据（以字符串的形式）;
}

//调用后需要手动转换为对应的int，real，boolean，char类型
```

2. **变量及函数声明：**
   - 目标：随机生成变量名或函数名，然后随机为其赋为int、real、boolean、char类型。
   - 方法：利用权值生成模块多次生成char类型数据，来组成字符串作为函数名或变量名。在生成随机数对4取模，作为该变量或函数的类型，若为函数，则为之生成随机数量的参数。将之Pascal-S变量或函数声明语句输出在输出文件上，同时对应在函数表中存储该函数，在当前级变量表中存储该变量。

```
std::string generateName() {    //生成名字
    std::string name;
    int length = rand() % 5 + 3;      //生成长度为3到7的名字，可自行修改
```

```
    for (int i = 0; i < length; ++i) {
        name += generateType(3);
    }
    return name;
}

std::string generatefunvar(当前级函数表, 当前级变量表) {     //函数或变量简单声明
    int 随机生成ty;
    string name = generateName();
    if(ty % 2 == 0){     //函数
        int 随机生成num % 5;
        string type;
        if(num == 0) type = "void";
        else if(num == 1) type = "int";
        else if(num == 2) type = "real";
        else if(num == 3) type = "boolean";
        else type = "char";
        随机生成若干输入条件;
        调用结构生成模块，为此函数生成若干代码结构;
        函数存入当前级函数表;
        函数写入输出文件;
    }
    else{     //变量
        int 随机生成num % 4;
        string type;
        string s = generateType(int input);
        if(num == 0) type = "int"; 转换s为int;
        else if(num == 1) type = "real"; 转换s为real;
        else if(num == 2) type = "boolean"; 转换s为boolean;
        else type = "char"; 转换s为char;
        变量存入当前级变量表;
        变量写入输出文件;
    }
}
```

3. 结构生成：

- 目的：随机生成基本的Pascal函数，包括但不限于赋值、控制结构（如if, while）等。
- 方法：为每种Pascal-S结构定义一个生成函数。定义一个结构体（FunctionWithArgs），由函数指针及其参数类型的数组组成，然后将所有Pascal-S结构生成函数对应的结构体存储在数组中，方便对其进行随机调用。

```
template<typename ReturnType, typename... Args>
struct FunctionWithArgs {
    std::function<ReturnType(Args...)> function;
```

```
    std::tuple<Args...> args;
};

std::vector<FunctionWithArgs> functionsWithArgs = {存入所有Pascal-S结构生成函数所对应的结构体};

void generate(){
    int 随机生成num % functionsWithArgs内结构体数;
    if(functionsWithArgs[num]不需要参数) 直接生成;
    else 随机生成参数或者在当前变量列表中寻找合理参数，生成;
}
```

4. **变量存储：**
   - 目的：存储可被当前层次代码调用的变量名和函数名，方便结构生成模块访问和使用。
   - 方法：使用数据结构（如列表或字典）来存储变量和函数。当生成新的变量或函数时，将其名称添加到存储结构中。对于嵌套结构，实现作用域管理，以模拟Pascal的作用域规则。对于变量，应该存储：级数（在调用变量与函数时，可以调用当前级数及其之下的变量与函数）、变量名、值。对于函数，应该存储：级数、函数名、类别、参数表、内容。
5. **主体：**
   - 目的：连接其他模块，通过可控的多次随机调用结构生成模块的函数，生成完整的、可执行的Pascal-S代码。
   - 方法：初始化：设置初始调用次数，这决定了生成代码的长度。循环生成：根据设定的次数，循环调用结构生成模块，每次迭代可能根据概率调用不同的代码结构生成子程序。输出：将生成的Pascal代码输出到文件或标准输出。