



本科实验报告

课程名称 编译原理

成 员 李政 (3190105300)

王宜平 (3190102199)

郭帅 (3190105305)

学 院 竺可桢学院

专 业 混合班

指导老师 李莹

浙江大学

2022 年 5 月 22 日

第一章 序言

- 1.1 概述
- 1.2 开发环境
- 1.3 文件说明
- 1.4 组员分工

第二章 词法分析

- 2.1 Lex
- 2.2 正则表达式
- 2.3 具体实现
 - 2.3.1 定义区
 - 2.3.2 规则区

第三章 语法分析

- 3.1 Yacc
- 3.2 抽象语法树AST
 - 3.2.1 AST_BaseNode类
 - 3.2.2 AST_Expression类及其子类
 - 3.2.2.1 AST_Expression和AST_Expression_List
 - 3.2.2.2 二元和一元表达式: AST_Binary_Expression和AST_Unary_Expression
 - 3.2.2.4 Record表达式 (结构体) AST_Property_Expression
 - 3.2.2.5 常数表达式AST_Const_Value_Expression
 - 3.2.2.6 函数调用AST_Function_Call
 - 3.2.2.7 标识符表达式AST_Identifier_Expression和数组表达式AST_Array_Expression
 - 3.2.3 AST_Program类及其相关类
 - 3.2.3.1 AST_Program/AST_Program_Head/AST_Routine
 - 3.2.3.2 AST_Routine_Head/AST_Routine
 - 3.2.3.3 AST_Declaration_BaseClass/AST_Routine_Part
 - 3.2.3.4 AST_Function_Declaration/ AST_Procedure_Declaration
 - 3.2.3.5 AST_Function_Head/AST_Procedure_Head
 - 3.2.3.6 AST_Parameters/AST_Parameters_Declaration_List
 - 3.2.3.7 AST_Parameters_Type_List/AST_Variable_Parameters_List
 - 3.2.4 AST_Statement类及其相关类
 - 3.2.4.1 AST_Compound_Statement/ AST_Statement_List / AST_Statement
 - 3.2.4.2 AST_Label/AST_Non_Label_Statement/
 - 3.2.4.2 AST_Assign_Statement
 - 3.2.4.3 AST_Procedure_Statement
 - 3.2.4.4 AST_If_Statement/AST_Else_Clause
 - 3.2.4.5 AST_Repeat_Statement/AST_While_Statement/
 - 3.2.4.6 AST_For_Statement/AST_Direction/AST_Goto_Statement
 - 3.2.5 AST_Type类及其相关类
 - 3.2.5.1 AST_Type/AST_Type_Declaration/AST_Type_Declaration_List
 - 3.2.5.2 AST_Type_Definition/AST_Type_Part
 - 3.2.5.3 AST_Simple_Type_Declaration/AST_Array_Type_Declaration
 - 3.2.5.4 AST_Record_Type_Declaration/AST_Field_Declaration及name_list
 - 3.2.6 AST_Value类及其相关类
- 3.3 语法分析的具体实现
 - 3.3.1 声明类型
 - 3.3.2 根据CFG生成AST

第四章 语义分析

- 4.1 LLVM概述
- 4.2 LLVM IR
 - 4.2.1 IR布局
 - 4.2.2 IR上下文环境
 - 4.2.3 IR核心类
- 4.3 IR生成
 - 4.3.1 运行环境设计
 - 4.3.2 类型系统
 - 4.3.3 函数信息维护

4.3.4 自定义类型转换成LLVM类型

4.3.5 表达式生成

 4.3.5.1 标识符生成

 4.3.5.2 数组表达式生成

 4.3.5.3 记录表达式生成

4.3.6 二元操作

4.3.7 赋值语句

4.3.8 主函数入口

4.3.9 Routine

4.3.10 常量/变量声明

4.3.11 函数/过程声明

4.3.12 函数/过程调用

4.3.13 系统函数/过程

4.3.14 分支语句

 4.3.14.1 if语句

 4.3.14.2 case语句

4.3.15 循环语句

 4.3.15.1 for语句

 4.3.15.2 while语句

 4.3.15.3 repeat语句

第五章 优化考虑

 常量折叠

第六章 代码生成

 6.1 选择目标机器

 6.2 配置 Module

 6.3 生成目标代码

第七章 测试案例

 7.1 数据类型测试

 7.1.1 内置类型测试

 7.1.2 数组类型测试

 7.1.3 记录类型测试

 7.2 运算测试

 7.3 控制流测试

 7.3.1 分支测试

 7.3.2 循环测试

 7.4 函数测试

 7.4.1 简单函数测试

 7.5 快速排序测试

 7.6 矩阵乘法测试

 7.7 选课系统测试

第八章 总结

附录

 parser/pascal.y：语法分析器

第一章 序言

1.1 概述

本实验小组设计了一个Pascal语法子集的编译器，支持将Pascal语法规范的代码文本输入转为目标代码。整个流程包括词法分析、语法分析、语义分析、优化、生成目标代码、测试。其中运用的工具或语言有：

- Flex实现词法分析
- Yacc实现语法分析
- LLVM实现代码优化、中间代码生成、目标代码生成
- C++实现AST构建和代码生成过程定义，是主要开发语言

我们最终通过了全部三个测试点，并且设计了额外的结构体Record，并且通过了运行测试。

1.2 开发环境

- 操作系统：Linux
- 编译环境：
 - Flex 2.6.4
 - Bison (GNU Bison) 3.7.6
 - LLVM 12.0.0
- 开发方式：VSCode远程连接虚拟机开发
- 版本控制：利用git进行版本控制，在Github中提出issue进行沟通交流。

1.3 文件说明

本次实验提交的文件及其说明如下：

- src: 源代码文件夹
 - lexer: 词法分析
 - parser: 语法分析放yacc文件
 - ast: 抽象语法树
 - type: 额外定义返回类型
 - contents: 函数空间相关
- doc: 报告文档文件夹
 - report.pdf: 报告文档
- test: 测试用例文件夹

1.4 组员分工

组员	具体分工
李政	语法分析，语义分析，中间代码生成，运行环境设计，目标代码生成，运行测试
王宜平	语法分析，部分语义分析，中间代码生成，运行测试
郭帅	词法分析，部分AST可视化

第二章 词法分析

词法分析是计算机科学中将字符序列转换为标记 (token) 序列的过程。在词法分析阶段，编译器读入源程序字符串流，将字符流转换为标记序列，同时将所需要的信息存储，然后将结果交给语法分析器。

我们使用术语Token来表示由连续字符组成的每个有意义的单词。记号是词法分析的基本单位，通常用空格或制表符分隔。在lexer/pascal.l中，我们构建了几个规则来检测不同类型的标记，如关键字、标识符和文字。

2.1 Lex

我们使用Lex来定位每个token，该位置在语义分析中可以在发生错误时提供具体的错误位置。方法如下：

```
1 #define YY_USER_ACTION yyloc.first_line = yyloc.last_line = yylineno; \
2     yyloc.first_column = yycolumn; yyloc.last_column = yycolumn + yylen - \
3     1; \
4     yycolumn += yylen;
```

Lex的整体框架如下：

Lex的输入文件由以%%分割的三部分组成，分别是声明区、规则区和程序区。声明区用于声明变量类型，规则区用于提供正则表达式，编写词法规则，程序区用于提供一些其它功能，比如此处提供了一个错误检测功能。

```
1 {definitions}
2 %%
3 {rules}
4 %%
5 {user subroutines}
```

2.2 正则表达式

要给每个需要识别的token加上规则。本程序的主要token分为两类，一类是关键字的正则表达式。这类表达式直接以其名字作为一个类型。另一类是识别整数、小数、注释等的正则表达式，这类表达式通过基本符号的正确组合构建。

2.3 具体实现

2.3.1 定义区

一部分为每个关键字定义正则表达式，使用 KEY_xxx xxx 等形式为关键字、类型、运算符定义。

```
1 KEY_AND and
2 ...
3 ...
4 ...
5 TYPE_INT integer
6 ...
7 ...
8 ...
9 ...
10 SYM_ADD "+"
11 ...
12 ...
```

另一部分为识别整数、小数、字符串、注释、标识符等等的正则表达式。

```
1 LITERAL_INT [0-9] | [1-9][0-9]+
2 LITERAL_FLOAT ([0-9]+\.[0-9]+) | ([0-9]+\.[0-9]+e{SIGN}?[0-9]+) | ([0-9]+e{SIGN}?[0-9]+)
3 LITERAL_CHAR \'.\''
4 LITERAL_ESC_CHAR '\\#\\''
5 LITERAL_STR \\'([^\']|{\LITERAL_ESC_CHAR})*\'
6
7 COMMENT1 {"[^\\}]*"}"
8 COMMENT2 //[^\\n]*"
9 IDENTIFIER [a-zA-Z_][a-zA-Z0-9_]*
```

2.3.2 规则区

对于每个类型，也即识别到的每一个token，我们都用一个函数将其值返回。该函数在debug状态下提供一个输出告知该token的类型，在一般状态下直接返回。对于包含value的函数则将其先赋给yyval。识别到注释（包括 COMMENT1 和 COMMENT2 两种类型的注释），空格，制表符则不进行任何操作直接忽略。

对于换行符则更新yycolumn，对于识别到的其它未被定义的类型的token则输出报错信息。

```
1 #ifdef DEBUG_LEXER
2 #define OUR_RETURN(X) cout << (#X) << endl;
3 #else
4 #define OUR_RETURN(X) return(X);
5 #endif
6
7 ...
8
9 {KEY_AND} {
10     OUR_RETURN(KEY_AND)
11 }
12
13 ...
14
15 {LITERAL_INT} {
16     yyval = strdup(yytext);
17     OUR_RETURN(LITERAL_INT)
18 }
19
20 ...
21
22 {COMMENT1} {
```

```
23 }
24
25 {COMMENT2} {
26 }
27
28 \n|(\r\n) {
29     yycolumn = 1;
30 }
31 " "|\\t {}
32
33 . {
34     printf("Unknown Character %d\n", (int) yytext[0]);
35     return yytext[0];
36 }
```

第三章 语法分析

语法分析的过程是在前面词法分析的基础上，将产生的token文本进行分析，确定其语法结构的过程。我们最后通过建立抽象语法树（Abstract Syntax Tree）等方式来呈现这种结构。

3.1 Yacc

我们使用Yacc这一语法分析工具来和前面提到的lex工具一起完成语法分析的任务。他会匹配lex中定义的对应CFG规则，来调用通过用户自定义的处理函数进行语法分析，如建立AST等。

Yacc本身是遵循LALR(1)这一自底向上语法分析规则的自动语法生成器，与Lex类似，Yacc的输入文件通过%%分割成declarations、rules和programs三个区域。

```
1 declarations
2 %%
3 rules
4 %%
5 programs
```

declarations部分需要提前用`%token<token_type>`等语句声明非终结符和终结符，且rules区域使用CFG而不是要求更高的正则表达式。

3.2 抽象语法树AST

抽象语法树AST通过树这一数据结构来表示语法的结构。一般语法树会将非终结符全部展示，而抽象语法树会省略掉一般语法树中的一些细节，只保留最重要的结构，这也方便我们构建树的过程。如`if-else-statement`语句会直接保留三个子结点：`condition`、`then-statement`、`else-statement`等，以及如嵌套的括号等会直接通过树的层级结构表示优先级，而不会显式存储括号的位置。

我们AST的整体设计思路是，建立一个基结点类来记录语句块在代码中所处的位置，方便错误处理。然后其他的结点类都继承这个基类。类的内部主要是定义了子类的属性和构造函数，以及对应的代码生成函数。

3.2.1 AST_BaseNode类

AST_BaseNode 类是抽象语法树其他所有类的基类，其私有属性包括代码所在行号和列号。公共属性包括构造函数、获取行号和列号信息的功能函数以及CodeGenerate的代码生成函数。

```
1 #pragma once
2 #include <string>
3 #include <vector>
4 #include <iostream>
5 #include <string>
6 #include "../type/type.hpp"
7 #include <llvm/IR/Value.h>
8
9 class AST_BaseNode{
10     private:
11         // location for error information
12         int column;
13         int row;
14     public:
15         AST_BaseNode() = default; //default
16         AST_BaseNode(int _column, int _row):column(_column), row(_row){}
17         int GetColumn() const{
18             return this->column;
19         }
20         int GetRow() const{
21             return this->row;
22         }
23         std::string GetLocationString() const{
24             return "(" + std::to_string(this->row) + "," +
25             std::to_string(this->column) + ")";
26         }
27         std::pair<int,int> GetLocation() const{
28             return std::make_pair(this->column, this->row);
29         }
30
31         void SetColumn(int _column){
32             this->column = _column;
33         }
34         void SetRow(int _row){
35             this->row = _row;
36         }
37         void SetLocation(std::pair<int, int> location){
38             this->column = location.first;
39             this->row = location.second;
40         }
41         void SetLocation(int _column, int _row){
42             this->column = _column;
43             this->row = _row;
44         }
45
46         virtual void Levelorder(){
47             std::cout << "root" << std::endl;
48         }
49
50         virtual std::shared_ptr<Custom_Result> CodeGenerate() = 0;
51     };
```

3.2.2 AST_Expression类及其子类

AST_Expression类及其子类种类如下：

```
1 #pragma once
2 #include "AST_BaseNode.hpp"
3 #include "AST_Value.hpp"
4
5 class AST_Expression;
6 class AST_Expression_List;
7 /* 下面的类作为AST_Expression的子类 */
8 class AST_Binary_Expression;
9 class AST_Unary_Expression;
10 class AST_Property_Expression;
11 class AST_Const_Value_Expression;
12 class AST_Function_Call;
13 class AST_Identifier_Expression;
14 class AST_Array_Expression;
15
16 class AST_Const_value;
```

3.2.2.1 AST_Expression和AST_Expression_List

AST_Expression类作为表达式类的父类，便于其他函数定义表达式类时作为父类被继承。而AST_Expression_List类保留了一个AST_Expression的vector向量来作为表达式列表信息。

```
1 class AST_Expression : public AST_BaseNode
2 {
3     public:
4         AST_Expression() = default;
5     };
6
7 class AST_Expression_List : public AST_Expression
8 {
9     public:
10        std::shared_ptr<Custom_Result> CodeGenerate() override;
11
12    public:
13        AST_Expression_List() = default;
14        void Add_Expression(AST_Expression *expr)
15        {
16            expr_list.push_back(expr);
17        }
18
19        std::vector<AST_Expression *> expr_list;
20    };
21
```

3.2.2节的下列函数都默认为继承自AST_Expression类，不再赘述

3.2.2.2 二元和一元表达式：AST_Binary_Expression和AST_Unary_Expression

AST_Binary_Expression 定义了表达式的二元运算，支持比较、四则运算（整型和浮点）、逻辑运算等等，并且自定义了操作数类方便调用

```
1  /**
2  * @brief : 二元表达式
3  *
4  */
5  class AST_Binary_Expression : public AST_Expression
6  {
7  public:
8     std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10 public:
11    enum class Operation
12    {
13        GE,
14        GT,
15        LE,
16        LT,
17        EQUAL,
18        UNEQUAL,
19        PLUS,
20        MINUS,
21        OR,
22        MUL,
23        REALDIV,
24        DIV,
25        MOD,
26        AND
27    };
28    AST_Binary_Expression(Operation _my_operation, AST_Expression
*_left_expression, AST_Expression *_right_expression) :
29    my_operation(_my_operation), left_expression(_left_expression),
30    right_expression(_right_expression){};
31
32    static std::string Get_Operation_Name(Operation my_op)
33    {
34        switch (my_op)
35        {
36            case Operation::GE:
37                return "GE";
38                break;
39            case Operation::GT:
40                return "GT";
41                break;
42            case Operation::LE:
43                return "LE";
44                break;
45            case Operation::LT:
46                return "LT";
47                break;
48            case Operation::EQUAL:
49                return "EQUAL";
50                break;
51            case Operation::UNEQUAL:
```

```

50         return "UNEQUAL";
51     break;
52     case Operation::PLUS:
53         return "PLUS";
54     break;
55     case Operation::MINUS:
56         return "MINUS";
57     break;
58     case Operation::OR:
59         return "OR";
60     break;
61     case Operation::MUL:
62         return "MUL";
63     break;
64     case Operation::DIV:
65         return "DIV";
66     break;
67     case Operation::MOD:
68         return "MOD";
69     break;
70     case Operation::AND:
71         return "AND";
72     break;
73 }
74 return "";
75 }
76
77 Operation my_operation;
78 AST_Expression *left_expression, *right_expression;
79 };

```

`AST_Unary_Expression` 定义了表达式的一元运算，支持取非，取反等操作

```

1 /**
2  * @brief : 一元表达式
3 *
4 */
5 class AST_Unary_Expression : public AST_Expression
6 {
7 public:
8     std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10 public:
11     enum class Operation
12     {
13         NOT,
14         SUB,
15         ADD
16     };
17     AST_Unary_Expression(Operation _my_operation, AST_Expression
*_expression) : my_operation(_my_operation), expression(_expression){};
18
19     Operation my_operation;
20     AST_Expression *expression;
21 };

```

3.2.2.4 Record表达式 (结构体) AST_Property_Expression

AST_Property_Expression 定义了结构表达式的成员名以及Record对象本身的变量名，方便后续通过变量名访问变量空间调用对应变量的地址和值。

```
1  /**
2   * @brief 结构表达式 : Record的成员
3   *
4   */
5  class AST_Property_Expression : public AST_Expression
6  {
7  public:
8      std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10     /*id:Record变量名 ; prop_id:成员变量名*/
11     std::string id, prop_id;
12
13 public:
14     AST_Property_Expression(std::string _id, std::string _prop_id) :
15     id(_id), prop_id(_prop_id){};
16 }
```

3.2.2.5 常数表达式AST_Const_Value_Expression

AST_Const_Value_Expression 定义了常数值表达式，简单地储存了常数值对象 AST_Const_Value。

```
1  /**
2   * @brief : 常数表达式
3   *
4   */
5  class AST_Const_Value_Expression : public AST_Expression
6  {
7  public:
8      std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10     AST_Const_Value *const_value;
11
12 public:
13     AST_Const_Value_Expression(AST_Const_Value *_const_value) :
14     const_value(_const_value){};
15 }
```

3.2.2.6 函数调用AST_Function_Call

函数表达式也是作为一个表达式值进行返回

```
1  /**
2   * @brief : 常数表达式
3   *
4   */
5  class AST_Const_Value_Expression : public AST_Expression
```

```

6  {
7      public:
8          std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10     AST_Const_Value *const_value;
11
12     public:
13         AST_Const_Value_Expression(AST_Const_Value *_const_value) :
14             const_value(_const_value){};
15     };
16
17     class AST_Function_Call : public AST_Expression
18     {
19         public:
20             std::shared_ptr<Custom_Result> CodeGenerate() override;
21
22             /*func_id:函数名 ; args_list:参数列表*/
23             std::string func_id;
24             AST_Expression_List *args_list;
25
26             public:
27                 AST_Function_Call(std::string _func_id, AST_Expression_List *_args_list)
28                     : func_id(_func_id), args_list(_args_list){};
29     };

```

3.2.2.7 标识符表达式AST_Identifier_Expression和数组表达式AST_Array_Expression

这两个表达式主要是都储存了一个id作为变量调用的标识符，`AST_Array_Expression`的数组索引内部还可以储存其他的表达式类，故定义如下：

```

1  class AST_Identifier_Expression : public AST_Expression
2  {
3      public:
4          std::shared_ptr<Custom_Result> CodeGenerate() override;
5
6          std::string id;
7
8      public:
9          AST_Identifier_Expression(std::string _id) : id(_id){};
10     };
11
12     class AST_Array_Expression : public AST_Expression
13     {
14         public:
15             std::shared_ptr<Custom_Result> CodeGenerate() override;
16
17             std::string id;
18             AST_Expression *expression;
19
20         public:
21             AST_Array_Expression(std::string _id, AST_Expression *_expression) :
22                 id(_id), expression(_expression){};
23     };

```

3.2.3 AST_Program类及其相关类

AST_Program 类及其子类种类如下：

```
1 #pragma once
2 #include "AST_BaseNode.hpp"
3 #include "AST_Type.hpp"
4 #include "AST_Expression.hpp"
5
6 class AST_Variable_Part;
7 class AST_Type_Part;
8 class AST_Const_Part;
9
10 class AST_Compound_Statement;
11 class AST_Program_Head;
12 class AST_Routine;
13 class AST_Routine_Body;
14 class AST_Routine_Head;
15
16 class AST_Routine_Part;
17 class AST_Parameters_Declaration_List;
18 class AST_Parameters;
19 class AST_Function_Declaration;
20 class AST_Procedure_Declaration;
21 class AST_Procedure_Head;
22 class AST_Parameters_Type_List;
23 class AST_Variable_Parameters_List;
24 class AST_Function_Head;
25
```

3.2.3.1 AST_Program/AST_Program_Head/AST_Routine

AST_Program 是整个程序头对应的类，在yacc中定义的CFG中对应生成program_head 和 routine，其中 AST_Program_Head 主要匹配Pascal程序名， Routine 生成routine_head和routine_body，包含了程序的其余全部代码信息。这三个类对应的定义非常直接，主要包含了CodeGenerate和构造函数。

```
1 /*
2 program:
3     pro_head routine SYM_PERIOD
4 ;
5 */
6 class AST_Program : public AST_BaseNode
7 {
8 public:
9     std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 private:
12     AST_Program_Head *program_head;
13     AST_Routine *routine;
14
15 public:
16     AST_Program(AST_Program_Head *_program_head, AST_Routine *_routine) :
17         program_head(_program_head), routine(_routine){};
```

```

17     AST_Program_Head *Get_Program_Head() const
18     {
19         return this->program_head;
20     }
21
22     AST_Routine *Get_Routine() const
23     {
24         return this->routine;
25     }
26 }
27
28 /*
29 pro_head:
30     KEY_PROGRAM IDENTIFIER SYM_SEMICOLON{
31     }
32 ;
33 */
34
35 class AST_Program_Head : public AST_BaseNode
36 {
37 public:
38     std::shared_ptr<Custom_Result>CodeGenerate() override;
39
40 private:
41     std::string identifier;
42
43 public:
44     AST_Program_Head(std::string _identifier) : identifier(_identifier){};
45
46     std::string Get_Identifier() const
47     {
48         return identifier;
49     }
50 }
51
52 /*
53 routine:
54     routine_head routine_body{}
55 ;
56 */
57
58 class AST_Routine : public AST_BaseNode
59 {
60 public:
61     std::shared_ptr<Custom_Result>CodeGenerate() override;
62
63 private:
64     AST_Routine_Head *routine_head;
65     AST_Routine_Body *routine_body;
66
67 public:
68     AST_Routine(AST_Routine_Head *_routine_head, AST_Routine_Body
*_routine_body) : routine_head(_routine_head), routine_body(_routine_body)
{};;
69
70     AST_Routine_Head *Get_Routine_Head() const
71     {
72         return this->routine_head;

```

```

73     }
74
75     AST_Routine_Body *Get_Routine_Body() const
76     {
77         return this->routine_body;
78     }
79 };

```

3.2.3.2 AST_Routine_Head/AST_Routine

`AST_Routine_Head` 类对应的是Pascal程序的“头部”，包括了常数值定义，类型定义，变量定义和程序定义等等，而 `AST_Routine` 对应是直接生成代码块内部的复合表达式：

```

1  /*
2  routine_head:
3      const_part type_part var_part routine_part{}
4  ;
5  */
6  class AST_Routine_Head : public AST_BaseNode
7  {
8  public:
9      std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 private:
12     AST_Const_Part *const_part;
13     AST_Type_Part *type_part;
14     AST_Variable_Part *var_part;
15     AST_Routine_Part *routine_part;
16
17 public:
18     AST_Routine_Head(AST_Const_Part *_const_part, AST_Type_Part *_type_part,
19     AST_Variable_Part *_var_part, AST_Routine_Part *_routine_part) :
20     const_part(_const_part), type_part(_type_part), var_part(_var_part),
21     routine_part(_routine_part){};
22     AST_Const_Part *Get_Const_Part() const
23     {
24         return this->const_part;
25     }
26     AST_Type_Part *Get_Type_Part() const
27     {
28         return this->type_part;
29     }
30     AST_Variable_Part *Get_Variable_Part() const
31     {
32         return this->var_part;
33     }
34     AST_Routine_Part *Get_Routine_Part() const
35     {
36         return this->routine_part;
37     }
38 };

```

```

38  /*
39   routine_body:
40     compound_stmt
41 ;
42 */
43
44 class AST_Routine_Body : public AST_BaseNode
45 {
46 public:
47     std::shared_ptr<Custom_Result> CodeGenerate() override;
48
49 private:
50     AST_Compound_Statement *compound_statement;
51
52 public:
53     AST_Routine_Body(AST_Compound_Statement *_compound_statement) :
54         compound_statement(_compound_statement){};
55     AST_Compound_Statement *Get_Compound_Statement() const
56     {
57         return this->compound_statement;
58     }
59 };

```

3.2.3.3 AST_Declaration_BaseClass/AST_Routine_Part

`AST_Declaration_BaseClass` 是表示函数声明的类，包括了function类的声明或者procedure类的声明。`AST_Routine_Part` 部分实现了多个function和procedure的混合声明的CFG，即：

```

1 routine_part: {function_decl | procedure_decl}
2     routine_part function_decl
3     | routine_part procedure_decl
4     | function_decl
5     | procedure_decl
6     |
7 ;

```

`AST_Declaration_BaseClass` 可以通过C++的enum类定义比较方便地实现，而 `AST_Routine_Part` 可以表示多个 `AST_Declaration_BaseClass` 组成的vector，具体如下：

```

1
2 // for routine part list
3 class AST_Declaration_BaseClass : public AST_BaseNode
4 {
5 public:
6     std::shared_ptr<Custom_Result> CodeGenerate() override;
7
8 private:
9     enum class ENUM_Declaration_Type
10    {
11         FUNCTION_DECLARATION,
12         PROCEDURE_DECLARATION
13    };
14     AST_Function_Declaration *function_declaration;

```

```

15     AST_Procedure_Declaration *procedure_declaration;
16     ENUM_Declaration_Type declaration_type;
17
18 public:
19     AST_Declaration_BaseClass(AST_Function_Declaration
20     *_function_declaration) : function_declaration(_function_declaration)
21     {
22         this->declaration_type =
23             ENUM_Declaration_Type::FUNCTION_DECLARATION;
24     }
25     AST_Declaration_BaseClass(AST_Procedure_Declaration
26     *_procedure_declaration) : procedure_declaration(_procedure_declaration)
27     {
28         this->declaration_type =
29             ENUM_Declaration_Type::PROCEDURE_DECLARATION;
30     }
31
32     AST_Function_Declaration *Get_Function_Declaration() const
33     {
34         return this->function_declaration;
35     }
36     AST_Procedure_Declaration *Get_Procedure_Declaration() const
37     {
38         return this->procedure_declaration;
39     }
40
41     ENUM_Declaration_Type Get_Declaration_Type() const
42     {
43         return this->declaration_type;
44     }
45
46 };
47
48 /**
49 routine_part: {function_decl | procedure_decl}
50 routine_part function_decl
51 | routine_part procedure_decl
52 | function_decl
53 | procedure_decl
54 |
55 ;
56 */
57 class AST_Routine_Part : public AST_BaseNode
58 {
59 public:
60     std::shared_ptr<Custom_Result>CodeGenerate() override;
61
62 private:
63     std::vector<AST_Declaration_BaseClass *> declaration_list;
64
65 public:
66     void Add_Declaration(AST_Declaration_BaseClass *_declaration)
67     {
68         this->declaration_list.push_back(_declaration);
69     }
70     AST_Routine_Part(AST_Declaration_BaseClass *_declaration)
71     {
72         this->declaration_list.clear();
73         Add_Declaration(_declaration);
74     }
75 }
```

```
69     std::vector<AST_Declaration_BaseClass *> Get_Declaration_List() const
70     {
71         return declaration_list;
72     }
73 };
```

3.2.3.4 AST_Function_Declaration/ AST_Procedure_Declaration

function声明和procedure声明两部分是类似的，都是在函数的开头定义函数名和对应的参数列表，以及函数体的定义。这里后两部分归成routine部分，而用head来表示函数或例程名以及参数列表。对应的CFG是类似的

```
1 procedure_decl:
2     procedure_head SYM_SEMICOLON routine SYM_SEMICOLON
3 ;
4 function_decl:
5     function_head SYM_SEMICOLON routine SYM_SEMICOLON
6 ;
7
```

对应的定义函数也很简单：

```
1 /*
2  * 
3  * function_decl:
4  *     function_head SYM_SEMICOLON routine SYM_SEMICOLON
5  * ;
6  */
7 class AST_Function_Declaration : public AST_BaseNode
8 {
9 public:
10     std::shared_ptr<Custom_Result> CodeGenerate() override;
11 
12 private:
13     AST_Function_Head *function_head;
14     AST_Routine *routine;
15 
16 public:
17     AST_Function_Declaration(AST_Function_Head *_function_head, AST_Routine *_routine) : function_head(_function_head), routine(_routine){};
18     AST_Function_Head *Get_Function_Head() const
19     {
20         return this->function_head;
21     }
22     AST_Routine *Get_routine() const
23     {
24         return this->routine;
25     }
26 };
27 
28
```

```

29 /*
30 procedure_decl:
31     procedure_head SYM_SEMICOLON routine SYM_SEMICOLON
32 ;
33 */
34 class AST_Procedure_Declaration : public AST_BaseNode
35 {
36 public:
37     std::shared_ptr<Custom_Result> CodeGenerate() override;
38
39 private:
40     AST_Procedure_Head *procedure_head;
41     AST_Routine *routine;
42
43 public:
44     AST_Procedure_Declaration(AST_Procedure_Head *_procedure_head,
45     AST_Routine *_routine) : procedure_head(_procedure_head), routine(_routine)
46     {};
47     AST_Procedure_Head *Get_Procedure_Head() const
48     {
49         return this->procedure_head;
50     }
51     AST_Routine *Get_Routine() const
52     {
53         return this->routine;
54     }
55 };

```

3.2.3.5 AST_Function_Head/AST_Procedure_Head

这两者功能是类似的，定义了**标识符**、**参数列表**，对于函数不同的一点在于需要声明**返回类型**。这里我们限定返回值是simple_type，在后面部分会进行详述。

```

1 /*
2 function_head:
3     KEY_FUNCTION IDENTIFIER parameters SYM_COLON simple_type_decl
4 ;
5 */
6 class AST_Function_Head : public AST_BaseNode
7 {
8 public:
9     std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 private:
12     std::string identifier;
13     AST_Parameters *parameters;
14     AST_Simple_Type_Declaration *simple_type_declaration;
15
16 public:
17     AST_Function_Head(std::string _identifier, AST_Parameters *_parameters,
18     AST_Simple_Type_Declaration *_simple_type_declaration) :
19     identifier(_identifier), parameters(_parameters),
20     simple_type_declaration(_simple_type_declaration){};
21     std::string Get_Identifier() const

```

```

19     {
20         return this->identifier;
21     }
22     AST_Parameters *Get_Parameters() const
23     {
24         return this->parameters;
25     }
26     AST_Simple_Type_Declaration *Get_Simple_Type_Declaration() const
27     {
28         return this->simple_type_declaration;
29     }
30 };
31
32 /**
33 procedure_head:
34     KEY_PROCEDURE IDENTIFIER parameters
35 ;
36 */
37 class AST_Procedure_Head : public AST_BaseNode
38 {
39 public:
40     std::shared_ptr<Custom_Result>CodeGenerate() override;
41
42 private:
43     std::string identifier;
44     AST_Parameters *parameters;
45
46 public:
47     AST_Procedure_Head(std::string _identifier, AST_Parameters *_parameters)
48     : identifier(_identifier), parameters(_parameters){};
49     std::string Get_Identifier() const
50     {
51         return this->identifier;
52     }
53     AST_Parameters *Get_Parameters() const
54     {
55         return this->parameters;
56     }
57 };

```

3.2.3.6 AST_Parameters/AST_Parameters_Declaration_List

`AST_Parameters` 类主要用于生成参数化列表【也可以为空】，`AST_Parameters_Declaration_List`类有多个声明内容，可以自然地用vector表示每个声明的内容（CFG中指定为`para_type_list`）。这两部分对应的CFG如下：

```

1 parameters:
2     SYM_LPAREN para_decl_list SYM_RPAREN {}
3     | {}
4 ;
5 para_decl_list:
6     para_decl_list SYM_SEMICOLON para_type_list
7     | para_type_list
8 ;

```

对应的代码如上分析所述：

```

1 /*
2 parameters:
3     SYM_LPAREN para_decl_list SYM_RPAREN {}
4     | {}
5 ;
6 */
7 class AST_Parameters : public AST_BaseNode
8 {
9 public:
10     std::shared_ptr<Custom_Result> CodeGenerate() override;
11
12 private:
13     AST_Parameters_Declaration_List *parameters_declaration_list;
14
15 public:
16     AST_Parameters() = default;
17     AST_Parameters(AST_Parameters_Declaration_List
18 *parameters_declaration_list) :
19         parameters_declaration_list(_parameters_declaration_list){};
20     AST_Parameters_Declaration_List *Get_Parameters_Declaration_List() const
21     {
22         return this->parameters_declaration_list;
23     }
24 };
25 /*
26 para_decl_list:
27     para_decl_list SYM_SEMICOLON para_type_list
28     | para_type_list
29 ;
30 */
31 class AST_Parameters_Declaration_List : public AST_BaseNode
32 {
33 public:
34     std::shared_ptr<Custom_Result> CodeGenerate() override;
35
36 private:
37     std::vector<AST_Parameters_Type_List *> parameters_type_list_list;
38
39 public:
40     void Add_Parameters_Type_List(AST_Parameters_Type_List
41 *_parameters_type_list)
42     {
43         this->parameters_type_list_list.push_back(_parameters_type_list);
44     }

```

```

43     AST_Parameters_Declaration_List(AST_Parameters_Type_List
44     *_parameters_type_list)
45     {
46         this->parameters_type_list_list.clear();
47         Add_Parameters_Type_List(_parameters_type_list);
48     }
49     AST_Parameters_Declaration_List()
50     {
51         this->parameters_type_list_list.clear();
52     }
53     std::vector<AST_Parameters_Type_List *> Get_Parameter_Type_List_List()
54     const
55     {
56         return this->parameters_type_list_list;
57     }
58 };

```

3.2.3.7 AST_Parameters_Type_List/AST_Variable_Parameters_List

这两个类是Pascal参数化定义的具体实现，因为pascal定义参数传递的形式是多个变量可以对应一个类型，所以这里的CFG应该写成多个name_list可以对应一个simple_type_decl，CFG具体为：

```

1 para_type_list:
2     var_para_list SYM_COLON simple_type_decl
3     | name_list SYM_COLON simple_type_decl
4 ;
5 var_para_list:
6     KEY_VAR name_list
7 ;

```

据此我们很方便可以定义出两个AST构造函数类：

```

1 /*
2 para_type_list:
3     var_para_list SYM_COLON simple_type_decl
4     | name_list SYM_COLON simple_type_decl
5 ;
6 */
7 class AST_Parameters_Type_List : public AST_BaseNode
8 {
9 public:
10     std::shared_ptr<Custom_Result> CodeGenerate() override;
11
12 private:
13     enum class List_Type
14     {
15         VARIABLE_PARAMETERS_LIST,
16         NAME_LIST
17     };
18     AST_Variable_Parameters_List *variable_parameters_list;
19     AST_Name_List *name_list;
20     List_Type list_type;
21     AST_Simple_Type_Declaration *simple_type_declaration;
22
23 public:

```

```

24     AST_Parameters_Type_List(AST_Variable_Parameters_List
25     *_variable_parameters_list, AST_Simple_Type_Declaration
26     *_simple_type_declaration) :
27     variable_parameters_list(_variable_parameters_list),
28     simple_type_declaration(_simple_type_declaration),
29     list_type(List_Type::VARIABLE_PARAMETERS_LIST){};
30
31     AST_Parameters_Type_List(AST_Name_List * _name_list,
32     AST_Simple_Type_Declaration * _simple_type_declaration) :
33     name_list(_name_list), simple_type_declaration(_simple_type_declaration),
34     list_type(List_Type::NAME_LIST){};
35
36     AST_Variable_Parameters_List *Get_Variable_Parameters_List() const
37     {
38         return this->variable_parameters_list;
39     }
40
41     AST_Name_List *Get_Name_List() const
42     {
43         return this->name_list;
44     }
45
46     AST_Simple_Type_Declaration *Get_Simple_Type_Declaration() const
47     {
48         return this->simple_type_declaration;
49     }
50
51     List_Type Get_List_Type() const
52     {
53         return this->list_type;
54     }
55
56     bool isvar(){
57         return this->list_type == List_Type::VARIABLE_PARAMETERS_LIST;
58     }
59
60
61     /*
62     var_para_list:
63         KEY_VAR name_list
64     */
65
66     class AST_Variable_Parameters_List : public AST_BaseNode
67     {
68
69     public:
70         std::shared_ptr<custom_Result>CodeGenerate() override;
71
72     private:
73         AST_Name_List *name_list;
74
75     public:
76         AST_Variable_Parameters_List(AST_Name_List * _name_list) :
77             name_list(_name_list){};
78         AST_Name_List *Get_Name_List() const
79         {
80             return this->name_list;
81         }
82     };
83
84
85
86
87

```

3.2.4 AST_Statement类及其相关类

AST_Statement 类及其子类种类如下：

```
1 #pragma once
2 #include "AST_BaseNode.hpp"
3 #include "AST_Expression.hpp"
4 #include "AST_value.hpp"
5
6 class AST_Compound_Statement;
7 class AST_Statement_List;
8 class AST_Statement;
9 class AST_Label;
10 class AST_Non_Label_Statement;
11 class AST_Assign_Statement;
12 class AST_Procedure_Statement;
13
14 class AST_If_Statement;
15 class AST_Case_Statement;
16 class AST_Repeat_Statement;
17 class AST_While_Statement;
18 class AST_For_Statement;
19 class AST_Goto_Statement;
20 class AST_Else_Clause;
21 class AST_Break_Statement;
22
23 class AST_Case_Expression_List;
24 class AST_Case_Expression;
25 class AST_Direction;
```

3.2.4.1 AST_Compound_Statement/ AST_Statement_List / AST_Statement

AST_Compound_Statement 是Program类生成的所有statement的起始，是其他statement语句的祖先，负责产生语句列表stmt_list，语句列表包含若干语句stmt，用分号隔开。而语句stmt可以产生包含label的标记语句（便于goto）和不含label的非标记语句non_label_stmt。利用vector对象，根据上述分析，对应的CFG和AST构造函数如下直接得出。

```
1 /*
2 compound_stmt:
3     KEY_BEGIN stmt_list KEY_END
4 ;
5 */
6 class AST_Compound_Statement : public AST_BaseNode
7 {
8 public:
9     std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 public:
12     AST_Statement_List *statement_list;
13
14 public:
15     AST_Compound_Statement(AST_Statement_List *_statement_list) :
16         statement_list(_statement_list){};
```

```

16     AST_Statement_List *Get_Statement_List() const
17     {
18         return this->statement_list;
19     }
20 };
21
22 /**
23 stmt_list:
24     stmt_list stmt SYM_SEMICOLON
25     |
26 ;
27 */
28 class AST_Statement_List : public AST_BaseNode
29 {
30 public:
31     std::shared_ptr<Custom_Result>CodeGenerate() override;
32
33 public:
34     std::vector<AST_Statement *> statement_list;
35
36 public:
37     AST_Statement_List() = default;
38     void Add_Statement(AST_Statement *_statement)
39     {
40         this->statement_list.push_back(_statement);
41     }
42 };
43
44 /**
45 stmt:
46     label SYM_COLON non_label_stmt
47     | non_label_stmt
48 ;
49 */
50 class AST_Statement : public AST_BaseNode
51 {
52 public:
53     std::shared_ptr<Custom_Result>CodeGenerate() override;
54
55 public:
56     enum class Has_Label
57     {
58         NOT_HAS,
59         HAS
60     };
61     AST_Label *label;
62     AST_Non_Label_Statement *non_label_statement;
63     Has_Label has_label;
64
65 public:
66     AST_Statement() = default;
67     void Add_Label_and_Non_Label_Statement(AST_Label *_label,
68     AST_Non_Label_Statement *_non_label_statement)
69     {
70         this->label = _label;
71         this->non_label_statement = _non_label_statement;
72         this->has_label = Has_Label::HAS;
73     }

```

```

73     void Add_Non_Label_Statement(AST_Non_Label_Statement
74         *_non_label_statement)
75     {
76         this->non_label_statement = _non_label_statement;
77         this->has_label = Has_Label::NOT_HAS;
78     }
79     AST_Label *Get_Label() const
80     {
81         return this->label;
82     }
83     AST_Non_Label_Statement *Get_Non_Label_Statement() const
84     {
85         return this->non_label_statement;
86     }
87     Has_Label Get_Has_Label() const
88     {
89         return this->has_label;
90     }
91 };

```

3.2.4.2 AST_Label/AST_Non_Label_Statement/

我们的Pascal编译器支持字符串表示或者int类型表示的label作为goto语句的跳转目的地址，AST_Label中指明了对应的结点的label类型；non_label_stmt作为基本的单语句单元包含了所有常用的语句类型，如：赋值、调用函数、复合、分支、重复、跳转、break等，AST_Non_Label_Statement同样用enum类知名了statement对应的类型。

```

1  /*
2  Label:
3      LITERAL_INT
4      | IDENTIFIER
5  ;
6 */
7 class AST_Label : public AST_BaseNode
8 {
9 public:
10     std::shared_ptr<custom_Result> CodeGenerate() override;
11
12 public:
13     enum class Int_or_Identifier
14     {
15         LITERAL_INT,
16         IDENTIFIER
17     };
18     int literal_int;
19     std::string identifier;
20     Int_or_Identifier int_or_identifier;
21
22 public:
23     AST_Label(int _literal_int) : literal_int(_literal_int)
24     {
25         this->int_or_identifier = Int_or_Identifier::LITERAL_INT;
26     }
27     AST_Label(std::string _identifier) : identifier(_identifier)
28     {

```

```

29         this->int_or_identifier = Int_or_Identifier::IDENTIFIER;
30     }
31
32     int Get_Literal_Int() const
33     {
34         return this->literal_int;
35     }
36     std::string Get_Identifier() const
37     {
38         return this->identifier;
39     }
40     Int_or_Identifier Get_Int_or_Identifier() const
41     {
42         return this->int_or_identifier;
43     }
44     bool isIdentifier() const{
45         return int_or_identifier == Int_or_Identifier::IDENTIFIER;
46     }
47 };
48
49 /*
50 non_label_stmt:
51     assign_stmt
52     | proc_stmt
53     | compound_stmt
54     | if_stmt
55     | case_stmt
56     | repeat_stmt
57     | while_stmt
58     | for_stmt
59     | goto_stmt
60 ;
61 */
62 class AST_Non_Label_Statement : public AST_BaseNode
63 {
64 public:
65     std::shared_ptr<Custom_Result>CodeGenerate() override;
66
67 public:
68     enum class Statement_Type
69     {
70         ASSIGN,
71         PROCEDURE,
72         COMPOUND,
73         IF,
74         CASE,
75         REPEAT,
76         WHILE,
77         FOR,
78         GOTO,
79         BREAK
80     };
81     Statement_Type statement_type;
82
83     AST_Assign_Statement *assgin_statement;
84     AST_Procedure_Statement *procedure_statement;
85     AST_Compound_Statement *compound_statement;
86     AST_If_Statement *if_statement;

```

```

87     AST_Case_Statement *case_statement;
88     AST_Repeat_Statement *repeat_statement;
89     AST_While_Statement *while_statement;
90     AST_For_Statement *for_statement;
91     AST_Goto_Statement *goto_statement;
92     AST_Break_Statement *break_statement;
93
94 public:
95     AST_Non_Label_Statement(AST_Assign_Statement *_assgin_statement) :
96     assgin_statement(_assgin_statement)
97     {
98         this->statement_type = Statement_Type::ASSIGN;
99     }
100    AST_Non_Label_Statement(AST_Procedure_Statement *_procedure_statement)
101    : procedure_statement(_procedure_statement)
102    {
103        this->statement_type = Statement_Type::PROCEDURE;
104    }
105    AST_Non_Label_Statement(AST_Compound_Statement *_compound_statement) :
106    compound_statement(_compound_statement)
107    {
108        this->statement_type = Statement_Type::COMPOUND;
109    }
110    AST_Non_Label_Statement(AST_If_Statement *_if_statement) :
111    if_statement(_if_statement)
112    {
113        this->statement_type = Statement_Type::IF;
114    }
115    AST_Non_Label_Statement(AST_Case_Statement *_case_statement) :
116    case_statement(_case_statement)
117    {
118        this->statement_type = Statement_Type::CASE;
119    }
120    AST_Non_Label_Statement(AST_Repeat_Statement *_repeat_statement) :
121    repeat_statement(_repeat_statement)
122    {
123        this->statement_type = Statement_Type::REPEAT;
124    }
125    AST_Non_Label_Statement(AST_While_Statement *_while_statement) :
126    while_statement(_while_statement)
127    {
128        this->statement_type = Statement_Type::WHILE;
129    }
130    AST_Non_Label_Statement(AST_For_Statement *_for_statement) :
131    for_statement(_for_statement)
132    {
133        this->statement_type = Statement_Type::FOR;
134    }
135    AST_Non_Label_Statement(AST_Goto_Statement *_goto_statement) :
136    goto_statement(_goto_statement)
137    {
138        this->statement_type = Statement_Type::GOTO;
139    }
140    AST_Non_Label_Statement(AST_Break_Statement *_break_statement) :
141    break_statement(_break_statement)
142    {
143        this->statement_type = Statement_Type::BREAK;
144    }

```

```

135     Statement_Type Get_Statement_Type() const
136     {
137         return statement_type;
138     }
139     /*
140     ASSIGN,
141     PROCEDURE,
142     COMPOUND,
143     IF,
144     CASE,
145     REPEAT,
146     WHILE,
147     FOR,
148     GOTO,
149     BREAK
150 */
151     bool isAssign(){return statement_type == Statement_Type::ASSIGN;}
152     bool isProcedure(){return statement_type == Statement_Type::PROCEDURE;}
153     bool isCompound(){return statement_type == Statement_Type::COMPOUND;}
154     bool isIf(){return statement_type == Statement_Type::IF;}
155     bool isCase(){return statement_type == Statement_Type::CASE;}
156     bool isRepeat(){return statement_type == Statement_Type::REPEAT;}
157     bool iswhile(){return statement_type == Statement_Type::WHILE;}
158     bool isFor(){return statement_type == Statement_Type::FOR;}
159     bool isGoto(){return statement_type == Statement_Type::GOTO;}
160     bool isBreak(){return statement_type == Statement_Type::BREAK;}
161 };
162

```

3.2.4.2 AST_Assign_Statement

我们的赋值语句支持三种形式：直接赋值给变量、赋值给数组对象、赋值给结构体对象，所以对应的CFG有三种选择或在一起，c++代码用enum类选择。CFG和代码对应如下：

```

1  /*
2  assign_stmt:
3      IDENTIFIER SYM_ASSIGN expression
4      | IDENTIFIER SYM_LBRAC expression SYM_RBRAC SYM_ASSIGN expression
5      | IDENTIFIER SYM_PERIOD IDENTIFIER SYM_ASSIGN expression
6  ;
7  */
8
9  class AST_Assign_Statement : public AST_BaseNode
10 {
11 public:
12     std::shared_ptr<Custom_Result>CodeGenerate() override;
13
14     bool valueAssign(llvm::Value* left_ptr, Pascal_Type* left_type,
15                      llvm::Value* right_ptr, Pascal_Type* right_type){
16         if (left_type->isArray()){
17             //todo type transfer
18         }
19         return true;
20     }

```

```

21
22 public:
23     enum class Assign_Type
24     {
25         I_1_E_1,
26         I_1_E_2,
27         I_2_E_1
28     };
29
30     Assign_Type assign_type;
31
32     std::string identifier1;
33     std::string identifier2;
34     AST_Expression *expression1;
35     AST_Expression *expression2;
36
37 public:
38     AST_Assign_Statement(std::string _identifier1, AST_Expression
39     *_expression1) : identifier1(_identifier1), expression1(_expression1),
40     assign_type(Assign_Type::I_1_E_1) {}
41     AST_Assign_Statement(std::string _identifier1, AST_Expression
42     *_expression1, AST_Expression *_expression2) : identifier1(_identifier1),
43     expression1(_expression1), expression2(_expression2),
44     assign_type(Assign_Type::I_1_E_2) {}
45     AST_Assign_Statement(std::string _identifier1, std::string _identifier2,
46     AST_Expression *_expression1) : identifier1(_identifier1),
47     identifier2(_identifier2), expression1(_expression1),
48     assign_type(Assign_Type::I_2_E_1) {}
49     Assign_Type Get_Assign_Type() const
50     {
51         return assign_type;
52     }
53     bool isDirectAssign() const{
54         return assign_type == Assign_Type::I_1_E_1;
55     }
56     bool isArrayAssign() const{
57         return assign_type == Assign_Type::I_1_E_2;
58     }
59     bool isRecordAttrAssign() const{ // record attribute
60         return assign_type == Assign_Type::I_2_E_1;
61     }
62 };

```

3.2.4.3 AST_Procedure_Statement

AST_Procedure_Statement 是调用函数的语句，对应只有两种选择，直接调用函数名（Pascal中无参数调用可以不用括号）和加括号，输入表达式列表作为参数列表。对应CFG和C++代码如下：

```

1 /*
2 proc_stmt:
3     // SYS_PROC | SYS_PROC LP expression_list RP | READ LP factor RP
4     IDENTIFIER
5     | IDENTIFIER SYM_LPAREN expression_list SYM_RPAREN
6 ;

```

```

7  */
8  class AST_Procedure_Statement : public AST_BaseNode
9  {
10 public:
11     std::shared_ptr<Custom_Result> CodeGenerate() override;
12
13 public:
14     enum class Has_Expression
15     {
16         Not,
17         Yes
18     };
19     Has_Expression has_expression;
20
21     std::string identifier;
22     AST_Expression_List *expression_list;
23
24 public:
25     AST_Procedure_Statement(std::string _identifier) :
26         identifier(_identifier), has_expression(Has_Expression::Not) {}
27     AST_Procedure_Statement(std::string _identifier, AST_Expression_List
28     *_expression_list) : identifier(_identifier),
29         expression_list(_expression_list), has_expression(Has_Expression::Yes) {}
30     Has_Expression Get_Has_Expression()
31     {
32         return has_expression;
33     }
34 };

```

3.2.4.4 AST_If_Statement/AST_Else_Clause

分支语句有if和case两种，我们主要实现的是If语句，包含了if-then语句和else语句两个部分，后一部分为可选，CFG和c++代码同样非常直接：

```

1  /*
2  if_stmt:
3      KEY_IF expression KEY_THEN stmt else_clause
4  ;
5  */
6  class AST_If_Statement : public AST_BaseNode
7  {
8 public:
9     std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 public:
12     AST_Expression *expression;
13     AST_Statement *statement;
14     AST_Else_Clause *else_clause;
15
16     AST_If_Statement(AST_Expression *_expression, AST_Statement *_statement,
17     AST_Else_Clause *_else_clause) : expression(_expression),
18         statement(_statement), else_clause(_else_clause) {}
19 };

```

```

18 /*
19 else_clause:
20     KEY_ELSE stmt {}
21     | {}
22 ;
23 */
24 class AST_Else_Clause : public AST_BaseNode
25 {
26 public:
27     std::shared_ptr<Custom_Result>CodeGenerate() override;
28
29 public:
30     AST_Statement *statement;
31     AST_Else_Clause() = default;
32     AST_Else_Clause(AST_Statement *_statement) : statement(_statement) {}
33     bool isEmpty(){
34         return (this->statement == nullptr);
35     }
36 };
37
38

```

3.2.4.5 AST_Repeat_Statement/AST_While_Statement/

这2个语句都是重复语句，而且实现都很类似，都是判断一个表达式是否满足要求，从而判断要不要重复执行一些stmt_list语句列表，并且有独立的代码块，区别只在于判断在第一次执行前还是后。对应的CFG和AST代码如下：

```

1 /*
2 repeat_stmt:
3     KEY_REPEAT stmt_list KEY_UNTIL expression
4 ;
5 */
6 class AST_Repeat_Statement : public AST_BaseNode
7 {
8 public:
9     std::shared_ptr<Custom_Result>CodeGenerate() override;
10
11 public:
12     AST_Statement_List *statement_list;
13     AST_Expression *expression;
14
15     AST_Repeat_Statement(AST_Statement_List *_statement_list, AST_Expression
*_expression) : statement_list(_statement_list), expression(_expression) {}
16 };
17
18 /*
19 while_stmt:
20     KEY WHILE expression KEY DO stmt
21 */
22 class AST_While_Statement : public AST_BaseNode
23 {
24 public:

```

```

25     std::shared_ptr<Custom_Result> CodeGenerate() override;
26
27 public:
28     AST_Expression *expression;
29     AST_Statement *statement;
30
31     AST_While_Statement(AST_Expression *_expression, AST_Statement
32     *_statement) : expression(_expression), statement(_statement) {}
33 };

```

3.2.4.6 AST_For_Statement/AST_Direction/AST_Goto_Statement

direction语句是for语句中表明迭代变量的变化方向是递增还是递减，对应to或者downto。for语句的迭代语句同前用stmt或者stmt_list实现。而goto语句在语法分析阶段只用直接识别goto标识符和label即可。对应CFG和AST定义如下：

```

1  /*
2  for_stmt:
3      KEY_FOR IDENTIFIER SYM_ASSIGN expression my_direction expression KEY_DO
4      stmt
5  ;
6  */
7  class AST_For_Statement : public AST_BaseNode
8  {
9     public:
10        std::shared_ptr<Custom_Result> CodeGenerate() override;
11
12    public:
13        std::string identifier;
14        AST_Expression *expression1;
15        AST_Direction *my_direction;
16        AST_Expression *expression2;
17        AST_Statement *statement;
18
19        AST_For_Statement(std::string _identifier, AST_Expression *_expression1,
20        AST_Direction *_direction, AST_Expression *_expression2, AST_Statement
21        *_statement) : identifier(_identifier), expression1(_expression1),
22        my_direction(_direction), expression2(_expression2), statement(_statement)
23    {}
24    };
25
26    /*
27    my_direction:
28        KEY_TO
29        | KEY_DOWNT0
30    ;
31    */
32    class AST_Direction : public AST_BaseNode
33    {
34        public:
35            std::shared_ptr<Custom_Result> CodeGenerate() override;
36
37        public:
38            enum class To_or_DownTo

```

```

34     {
35         To,
36         DownTo
37     };
38     To_or_DownTo to_or_downto;
39     AST_Direction() = default;
40     void Set_To()
41     {
42         to_or_downto = To_or_DownTo::To;
43     }
44     void Set_Down_To()
45     {
46         to_or_downto = To_or_DownTo::DownTo;
47     }
48     bool isTo(){
49         return to_or_downto == To_or_DownTo::To;
50     }
51 };
52
53 /*
54 goto_stmt:
55     KEY_GOTO Label
56 ;
57 */
58
59 class AST_Goto_Statement : public AST_BaseNode
60 {
61 public:
62     std::shared_ptr<Custom_Result>CodeGenerate() override;
63
64 public:
65     AST_Label *label;
66     AST_Goto_Statement(AST_Label *_label) : label(_label) {}
67 };
68
69 class AST_Break_Statement : public AST_BaseNode{
70 public:
71     std::shared_ptr<Custom_Result>CodeGenerate() override;
72 public:
73     AST_Break_Statement() = default;
74 };

```

3.2.5 AST_Type类及其相关类

AST_Type 类及其相关类种类如下：

```

1 #pragma once
2 #include "AST_BaseNode.hpp"
3 #include "AST_Value.hpp"
4 #include <vector>
5
6 class AST_BaseNode;
7 class AST_Const_Value;
8
9 class AST_Type_Part;

```

```

10 class AST_Type_Definition;
11 class AST_Type_Declaration_List;
12
13 class AST_Type;
14 class AST_Type_Declaration;
15 class AST_Simple_Type_Declaration;
16 class AST_Array_Type_Declaration;
17 class AST_Record_Type_Declaration;
18 class AST_Field_Declaration_List;
19 class AST_Field_Declaration;
20 class AST_Name_List;
21

```

3.2.5.1 AST_Type/AST_Type_Declaration/AST_Type_Declaration_List

Type类定义了最简单的数据类型，包括int、char、boolean、float和string，用enum类分隔；
 AST_Type_Declaration作为类型声明可以转成简单类型声明、数组声明和结构体声明。
 AST_Type_Declaration_List同前用vector定义，对应实现如下

```

1 // easy_type -> int / char / boolean / float / string
2 class AST_Type : public AST_BaseNode
3 {
4     public:
5         std::shared_ptr<Custom_Result> CodeGenerate() override;
6
7     public:
8         enum class Type_Name
9         {
10             INT,
11             FLOAT,
12             BOOLEAN,
13             CHAR,
14             STRING
15         };
16         AST_Type(AST_Type::Type_Name _typename) : my_typename(_typename){};
17
18         AST_Type::Type_Name Get_Type_Name() const
19         {
20             return this->my_typename;
21         }
22
23         AST_Type::Type_Name my_typename;
24     };
25
26 // type_decl -> SimpleTypeDecl / ArrayTypeDecl / RecordTypeDecl
27 class AST_Type_Declaration : public AST_BaseNode
28 {
29     public:
30         virtual std::shared_ptr<Custom_Result> CodeGenerate()=0;
31     };
32
33
34 /**
35 *
36 * @type_decl_list->{type_definition}

```

```

37 */
38 class AST_Type_Declaration_List : public AST_BaseNode
39 {
40 public:
41     std::shared_ptr<Custom_Result> CodeGenerate() override;
42
43 public:
44     AST_Type_Declaration_List() = default;
45     void Add_Type_Definition(AST_Type_Definition *type_definition)
46     {
47         type_definition_list.push_back(type_definition);
48     }
49
50     std::vector<AST_Type_Definition *> type_definition_list;
51 };

```

3.2.5.2 AST_Type_Definition/AST_Type_Part

这是类型定义和类型声明列表的CFG和C++代码，功能很直接：

```

1 /**
2 type_definition->
3     IDENTIFIER SYM_EQ type_decl SYM_SEMICOLON
4 */
5 class AST_Type_Definition : public AST_BaseNode
6 {
7 public:
8     std::shared_ptr<Custom_Result> CodeGenerate() override;
9
10 public:
11     AST_Type_Definition(std::string id, AST_Type_Declaration *_type_decl) :
12         identifier(id), type_decl(_type_decl){};
13
14     std::string identifier;
15     AST_Type_Declaration *type_decl;
16 };
17 /**
18 type_part->
19     KEY_TYPE type_decl_list | %empty
20 */
21 class AST_Type_Part : public AST_BaseNode
22 {
23 public:
24     std::shared_ptr<Custom_Result> CodeGenerate() override;
25
26 public:
27     AST_Type_Part(AST_Type_Declaration_List *_type_decl_list) :
28         type_decl_list(_type_decl_list){};
29
30     AST_Type_Declaration_List *type_decl_list;
31 };

```

3.2.5.3 AST_Simple_Type_Declaration/AST_Array_Type_Declaration

简单数据类型包括了之前指的5种easy type和identifier、枚举、 subrange等等，array定义则指定 subrange和元素类型即可。对应CFG和C++代码如下：

```
1  /*
2   * simple_type_decl ->
3   *     easy_type
4   *     | IDENTIFIER
5   *     | SYM_LPAREN name_list SYM_RPAREN
6   *     | const_value  SYM_RANGE  const_value
7   *     | IDENTIFIER  SYM_RANGE  IDENTIFIER
8   */
9
10 class AST_Simple_Type_Declaration : public AST_Type_Declaration
11 {
12 public:
13     std::shared_ptr<Custom_Result> CodeGenerate() override;
14
15 public:
16     enum class My_Type
17     {
18         EASY_TYPE,
19         IDENTIFIER,
20         ENUMERATE,
21         VALUE_RANGE,
22         IDENTIFIER_RANGE
23     };
24
25     AST_Simple_Type_Declaration(AST_Type *_type) : my_typename(_type),
26     my_type(My_Type::EASY_TYPE){};
27
28     AST_Simple_Type_Declaration(std::string _define_id) :
29     define_id(_define_id), my_type(My_Type::IDENTIFIER){};
30
31     AST_Simple_Type_Declaration(AST_Name_List *_name_list) :
32     name_list(_name_list), my_type(My_Type::ENUMERATE){};
33
34     AST_Simple_Type_Declaration(AST_Const_Value *_low, AST_Const_Value
35     *_high) : low(_low), high(_high), my_type(My_Type::VALUE_RANGE){};
36
37     AST_Simple_Type_Declaration(std::string _low_name, std::string
38     _high_name) : low_name(_low_name), high_name(_high_name),
39     my_type(My_Type::IDENTIFIER_RANGE){};
40
41     My_Type Get_Type() const
42     {
43         return this->my_type;
44     }
45
46     My_Type my_type;
```

```

47    };
48
49 /**
50 array_type_decl->
51     KEY_ARRAY SYM_LBRAC simple_type_decl SYM_RBRAC KEY_OF type_decl
52 */
53 class AST_Array_Type_Declaration : public AST_Type_Declaration
54 {
55 public:
56     std::shared_ptr<Custom_Result> CodeGenerate() override;
57
58 public:
59     AST_Array_Type_Declaration(AST_Simple_Type_Declaration
60     *_simple_type_decl, AST_Type_Declaration *_type_decl) :
61         simple_type_decl(_simple_type_decl), type_decl(_type_decl){};
62
63     AST_Simple_Type_Declaration *simple_type_decl;
64     AST_Type_Declaration *type_decl;
65 };

```

3.2.5.4 AST_Record_Type_Declaration/AST_Field_Declaration及name_list

Record类型为拓展功能，结构体声明转成多个field声明对应的列表，而field声明包含了name_list和子类型。Name_list即为多个identifier的vector。对应CFG和C++代码如下：

```

1 /**
2  * @brief
3  * record_type_decl ->
4      KEY_RECORD field_decl_list KEY_END
5 */
6 class AST_Record_Type_Declaration : public AST_Type_Declaration
7 {
8 public:
9     std::shared_ptr<Custom_Result> CodeGenerate() override;
10
11 public:
12     AST_Record_Type_Declaration(AST_Field_Declaration_List
13     *_field_decl_list) : field_decl_list(_field_decl_list){};
14
15     AST_Field_Declaration_List *field_decl_list;
16 };
17 /**
18  * @brief
19  * field_decl_list -> {field_decl}
20 */
21 class AST_Field_Declaration_List : public AST_BaseNode
22 {
23 public:
24     std::shared_ptr<Custom_Result> CodeGenerate() override;
25
26 public:
27     AST_Field_Declaration_List(AST_Field_Declaration *_field_decl)
28     {

```

```

29         field_decl_list.clear();
30         field_decl_list.push_back(_field_decl);
31     }
32     void Add_Field_Declaration(AST_Field_Declaration *_field_decl)
33     {
34         field_decl_list.push_back(_field_decl);
35     }
36
37     std::vector<AST_Field_Declaration *> field_decl_list;
38 };
39
40 /**
41 * @brief
42 * field_decl->
43 * name_list SYM_COLON type_decl SYM_SEMICOLON
44 */
45 class AST_Field_Declaration : public AST_BaseNode
46 {
47 public:
48     std::shared_ptr<Custom_Result> CodeGenerate() override;
49
50 public:
51     AST_Field_Declaration(AST_Name_List *_name_list, AST_Type_Declaration
52     *_type_decl) : name_list(_name_list), type_decl(_type_decl){};
53
54     AST_Name_List *name_list;
55     AST_Type_Declaration *type_decl;
56 };
57
58 /**
59 * name_list->
60 * {IDENTIFIER}
61 */
62 class AST_Name_List : public AST_BaseNode
63 {
64 public:
65     std::shared_ptr<Custom_Result> CodeGenerate() override;
66
67 public:
68     AST_Name_List() = default;
69     void Add_Identifier(std::string id)
70     {
71         identifier_list.push_back(id);
72     }
73
74     std::vector<std::string> identifier_list;
75 };

```

3.2.6 AST_Value类及其相关类

AST_Value类及其相关类种类如下：

```
1 #pragma once
2 #include "AST_BaseNode.hpp"
3 #include "AST_Type.hpp"
4 #include "AST_Expression.hpp"
5
6 class AST_Expression;
7 class AST_Name_List;
8 class AST_Type_Declaration;
9
10 class AST_Const_Part;
11 class AST_Const_Expression_List;
12 class AST_Const_Expression;
13 class AST_Const_Value;
14
15 class AST_Variable_Part;
16 class AST_Variable_Declaration_List;
17 class AST_Variable_Declaration;
```

这一部分较为直接，我们直接一并阐述。其定义了多种值类型，包括各种基本变量的常数类型声明、常数列表、常数表达式、变量类型声明、变量声明列表等等，实现和前面部分原理基本一致，直接放上相关的CFG和代码：

```
1 /**
2 * @brief
3 * const_value:
4     LITERAL_INT
5     | LITERAL_FLOAT
6     | LITERAL_CHAR
7     | LITERAL_STR
8     | LITERAL_FALSE
9     | LITERAL_TRUE
10 */
11 class AST_Const_Value : public AST_BaseNode
12 {
13 public:
14     std::shared_ptr<Custom_Result> CodeGenerate() override;
15
16 public:
17     enum class value_Type
18     {
19         INT,
20         FLOAT,
21         CHAR,
22         STRING,
23         FALSE,
24         TRUE
25     };
26     AST_Const_Value(std::string _content, enum value_Type _value_type) :
27         content(_content), value_type(_value_type){};
28     std::string Get_Value_Type_Name(value_Type value_type)
```

```

29     {
30         std::string name;
31         switch (value_type)
32         {
33             case AST_Const_Value::Value_Type::INT:
34                 name = "INT";
35                 break;
36             case AST_Const_Value::Value_Type::FLOAT:
37                 name = "FLOAT";
38                 break;
39             case AST_Const_Value::Value_Type::CHAR:
40                 name = "CHAR";
41                 break;
42             case AST_Const_Value::Value_Type::STRING:
43                 name = "STRING";
44                 break;
45             case AST_Const_Value::Value_Type::FALSE:
46                 name = "FALSE";
47                 break;
48             case AST_Const_Value::Value_Type::TRUE:
49                 name = "TRUE";
50                 break;
51         }
52         return name;
53     }
54
55     AST_Const_Value::Value_Type value_type;
56     std::string content;
57 };
58
59 /**
60 const_expr_list->
61 {const_expr}
62 */
63 class AST_Const_Expression_List : public AST_BaseNode
64 {
65 public:
66     std::shared_ptr<Custom_Result> CodeGenerate() override;
67
68 public:
69     AST_Const_Expression_List() = default;
70     void Add_Const_Expression(AST_Const_Expression *const_expr)
71     {
72         const_expr_list.push_back(const_expr);
73     }
74
75     std::vector<AST_Const_Expression *> const_expr_list;
76 };
77
78 class AST_Const_Expression : public AST_BaseNode
79 {
80 public:
81     std::shared_ptr<Custom_Result> CodeGenerate() override;
82
83 public:
84     AST_Const_Expression(std::string _id, AST_Expression *_value) :
85         id(_id), value(_value){};

```

```
86     std::string id;
87     AST_Expression *value;
88 };
89
90 class AST_Const_Part : public AST_BaseNode
91 {
92 public:
93     std::shared_ptr<Custom_Result> CodeGenerate() override;
94
95 public:
96     AST_Const_Part(AST_Const_Expression_List *_const_expr_list) :
97         const_expr_list(_const_expr_list){};
98
99     AST_Const_Expression_List *const_expr_list;
100 };
101
102 class AST_Variable_Part : public AST_BaseNode
103 {
104 public:
105     std::shared_ptr<Custom_Result> CodeGenerate() override;
106
107 public:
108     AST_Variable_Part(AST_Variable_Declaration_List *_var_decl_list) :
109         var_decl_list(_var_decl_list){};
110
111     AST_Variable_Declaration_List *var_decl_list;
112 };
113
114 class AST_Variable_Declaration_List : public AST_BaseNode
115 {
116 public:
117     std::shared_ptr<Custom_Result> CodeGenerate() override;
118
119 public:
120     AST_Variable_Declaration_List() = default;
121     void Add_Variable_Declaration(AST_Variable_Declaration *var_decl)
122     {
123         var_decl_list.push_back(var_decl);
124     }
125
126     std::vector<AST_Variable_Declaration *> var_decl_list;
127 };
128
129 class AST_Variable_Declaration : public AST_BaseNode
130 {
131 public:
132     std::shared_ptr<Custom_Result> CodeGenerate() override;
133
134 public:
135     AST_Variable_Declaration(AST_Name_List *_name_list,
136                             AST_Type_Declaration *_type_decl) :
137         name_list(_name_list),
138         type_decl(_type_decl){};
139
140     AST_Name_List *name_list;
141     AST_Type_Declaration *type_decl;
142 };
143
```

3.3 语法分析的具体实现

3.3.1 声明类型

我们在Yacc文件的声明区声明终结、非终结符类型，便于后续调用

```
1 %union{
2     int token_type;
3     char* str;
4
5     //AST_Value.hpp
6     AST_Const_Part* ast_const_part;
7     AST_Const_Expression_List* ast_const_expression_list;
8     AST_Const_Expression* ast_const_expression;
9     AST_Const_value* ast_const_value;
10    AST_Variable_Part* ast_variable_part;
11    AST_Variable_Declaration_List* ast_variable_declaration_list;
12    AST_Variable_Declaration* ast_variable_declaration;
13    //AST_Type.hpp
14    AST_Type_Part* ast_type_part;
15    AST_Type_Definition* ast_type_definition;
16    AST_Type_Declaration_List* ast_type_declaration_list;
17    AST_Type* ast_type;
18    AST_Type_Declaration* ast_type_declaration;
19    AST_Simple_Type_Declaration* ast_simple_type_declaration;
20    AST_Array_Type_Declaration* ast_array_type_declaration;
21    AST_Record_Type_Declaration* ast_record_type_declaration;
22    AST_Field_Declaration_List* ast_field_declaration_list;
23    AST_Field_Declaration* ast_field_declaration;
24    AST_Name_List* ast_name_list;
25    //AST_Expression.hpp
26    AST_Expression* ast_expression;
27    AST_Expression_List* ast_expression_list;
28    //AST_Program.hpp
29    AST_Program* ast_program;
30    AST_Program_Head* ast_program_head;
31    AST_Routine* ast_routine;
32    AST_Routine_Head* ast_routine_head;
33    // AST_Declaration_BaseClass* ast_declaraction_baseclass;
34    AST_Routine_Part* ast_routine_part;
35    AST_Routine_Body* ast_routine_body;
36    AST_Function_Declaration* ast_function_declaration;
37    AST_Function_Head* ast_function_head;
38    AST_Procedure_Declaration* ast_procedure_declaration;
39    AST_Procedure_Head* ast_procedure_head;
40    AST_Parameters* ast_parameters;
41    AST_Parameters_Declaration_List* ast_parameters_declaration_list;
42    AST_Parameters_Type_List* ast_parameters_type_list;
43    AST_Variable_Parameters_List* ast_variable_parameters_list;
44    //AST_Statement.hpp
45    AST_Compound_Statement* ast_compound_statement;
46    AST_Statement_List* ast_statement_list;
47    AST_Statement* ast_statement;
```

```

48     AST_Label* ast_label;
49     AST_Non_Label_Statement* ast_non_label_statement;
50     AST_Assign_Statement* ast_assign_statement;
51     AST_Procedure_Statement* ast_procedure_statement;
52     AST_If_Statement* ast_if_statement;
53     AST_Else_Clause* ast_else_clause;
54     AST_Case_Statement* ast_case_statement;
55     AST_Case_Expression_List* ast_case_expression_list;
56     AST_Case_Expression* ast_case_expression;
57     AST_Repeat_Statement* ast_repeat_statement;
58     AST_While_Statement* ast_while_statement;
59     AST_For_Statement* ast_for_statement;
60     AST_Direction* ast_direction;
61     AST_Goto_Statement* ast_goto_statement;
62     AST_Break_Statement* ast_break_statement;
63 }
64
65 %token<token_type> KEY_BREAK KEY_EXIT
66 %token<token_type> TYPE_INT TYPE_INT_8 TYPE_INT_16 TYPE_INT_32 TYPE_INT_64
67 %token<token_type> TYPE_UNSIGNED_INT_8 TYPE_UNSIGNED_INT_16
TYPE_UNSIGNED_INT_32 TYPE_UNSIGNED_INT_64
68 %token<token_type> TYPE_BOOLEAN TYPE_FLOAT TYPE_FLOAT_16 TYPE_FLOAT_32
TYPE_CHAR TYPE_STRING
69 %token<str> LITERAL_INT LITERAL_FLOAT LITERAL_CHAR LITERAL_ESC_CHAR
LITERAL_STR LITERAL_TRUE LITERAL_FALSE IDENTIFIER
70 %token<token_type> SYM_ADD SYM_SUB SYM_MUL SYM_DIV SYM_EQ SYM_LT SYM_GT
SYM_LBRAC SYM_RBRAC SYM_PERIOD SYM_COMMA SYM_COLON
71 %token<token_type> SYM_SEMICOLON SYM_AT SYM_CARET SYM_LPAREN SYM_RPAREN
SYM_NE SYM_LE SYM_GE SYM_ASSIGN SYM_RANGE COMMENT
72 %token<token_type> KEY_AND KEY_ARRAY KEY_ASM KEY_BEGIN KEY_CASE KEY_CONST
KEY_CONSTRUCTOR KEY_DESTRUCTOR KEY_DIV
73 %token<token_type> KEY_DO KEY_DOWNT0 KEY_ELSE KEY_END KEY_FILE KEY_FOR
KEY_FUNCTION KEY_GOTO KEY_IF KEY_IMPLEMENTATION KEY_IN
74 %token<token_type> KEY_INHERITED KEY_INLINE KEY_INTERFACE KEY_LABEL KEY_MOD
KEY NIL KEY_NOT KEY_OBJECT KEY_OF KEY_OPERATOR KEY_OR
75 %token<token_type> KEY_PACKED KEY_PROCEDURE KEY_PROGRAM KEY_RECORD
KEY_REINTRODUCE KEY_REPEAT KEY_SELF KEY_SET KEY_SHL KEY_SHR
76 %token<token_type> KEY_THEN KEY_TO KEY_TYPE KEY_UNIT KEY_UNTIL KEYUSES
KEY_VAR KEY_WHILE KEY_WITH KEY_XOR
77 %token<token_type> SIGN
78
79 %type<ast_const_part> const_part
80 %type<ast_const_expression_list> const_expr_list
81 %type<ast_const_expression> const_expr
82 %type<ast_const_value> const_value
83 %type<ast_variable_part> var_part
84 %type<ast_variable_declaration_list> var_decl_list
85 %type<ast_variable_declaration> var_decl
86
87 %type<ast_type_part> type_part
88 %type<ast_type_definition> type_definition
89 %type<ast_type> easy_type
90 %type<ast_type_declaration> type_decl
91 %type<ast_type_declaration_list> type_decl_list
92 %type<ast_simple_type_declaration> simple_type_decl
93 %type<ast_array_type_declaration> array_type_decl
94 %type<ast_record_type_declaration> record_type_decl
95 %type<ast_field_declaration_list> field_decl_list

```

```

96 %type<ast_field_declaration> field_decl
97 %type<ast_name_list> name_list
98
99 %type<ast_expression> expression expr term factor
100 %type<ast_expression_list> expression_list
101
102
103 %type<ast_program> program;
104 %type<ast_program_head> pro_head;
105 %type<ast_routine> routine;
106 %type<ast_routine_head> routine_head;
107 %type<ast_routine_part> routine_part;
108 %type<ast_routine_body> routine_body;
109 %type<ast_function_declaration> function_decl;
110 %type<ast_function_head> function_head;
111 %type<ast_procedure_declaration> procedure_decl;
112 %type<ast_procedure_head> procedure_head;
113 %type<ast_parameters> parameters;
114 %type<ast_parameters_declaration_list> para_decl_list;
115 %type<ast_parameters_type_list> para_type_list;
116 %type<ast_variable_parameters_list> var_para_list;
117 %type<ast_compound_statement> compound_stmt;
118 %type<ast_statement_list> stmt_list;
119 %type<ast_statement> stmt;
120 %type<ast_label> label;
121 %type<ast_non_label_statement> non_label_stmt;
122 %type<ast_assign_statement> assign_stmt;
123 %type<ast_procedure_statement> proc_stmt;
124 %type<ast_if_statement> if_stmt;
125 %type<ast_else_clause> else_clause;
126 %type<ast_case_statement> case_stmt;
127 %type<ast_case_expression_list> case_expr_list;
128 %type<ast_case_expression> case_expr;
129 %type<ast_repeat_statement> repeat_stmt;
130 %type<ast_while_statement> while_stmt;
131 %type<ast_for_statement> for_stmt;
132 %type<ast_direction> direction;
133 %type<ast_goto_statement> goto_stmt;
134 %type<ast_break_statement> break_stmt;
135

```

3.3.2 根据CFG生成AST

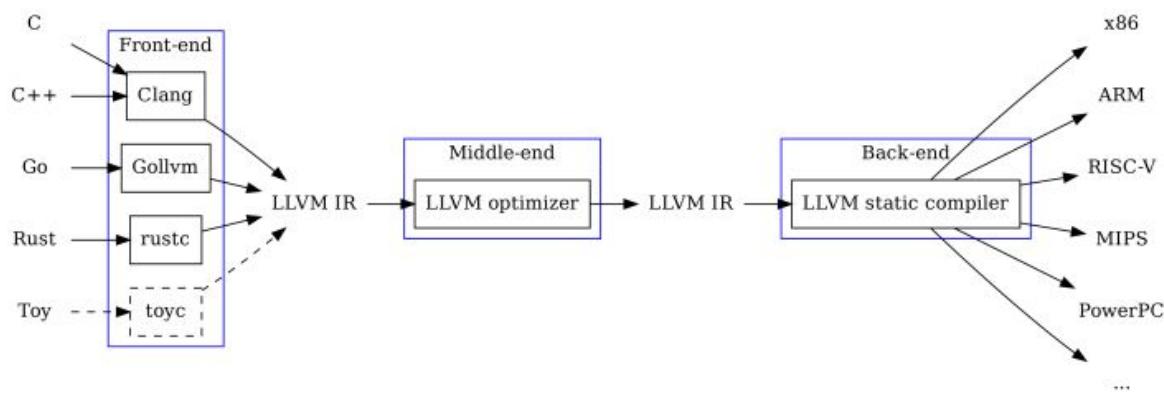
接着按从下往上的顺序构造语法树，文法见附录。其中我们还利用ast_root定位了节点的代码位置，方便输出错误信息。

第四章 语义分析

4.1 LLVM概述

LLVM(Low Level Virtual Machine)是以C++编写的编译器基础设施，包含一系列模块化的编译器组件和工具教练用俩开发编译器前端和后端。LLVM起源于2000年伊利诺伊大学Vikram Adve和Chris Lattner的研究，它是为了任意一种编程语言而写成的程序，利用虚拟技术创造出编译阶段、链接阶段、运行阶段以及闲置阶段的优化，目前支持Ada、D语言、Fortran、GLSL、Java字节码、Swift、Python、Ruby等十多种语言。

- 前端：LLVM最初被用来取代现有于GCC堆栈的代码产生器，许多GCC的前端已经可以与其运行，其中Clang是一个新的编译器，同时支持C、Objective-C以及C++。
- 中间端：LLVM IR是一种类似汇编的底层语言，一种强类型的精简指令集，并对目标指令集进行了抽象。LLVM支持C++中对象形式、序列化bitcode形式和汇编形式。
- 后端：LLVM支持x86、ARM、RISC-V、Qualcomm Hexagon、MIPS、Nvidia并行指令集等多种后端指令集。



4.2 LLVM IR

LLVM IR是**LLVM Intermediate Representation**，它是一种low-level language，是一个像**RISC**的指令集。

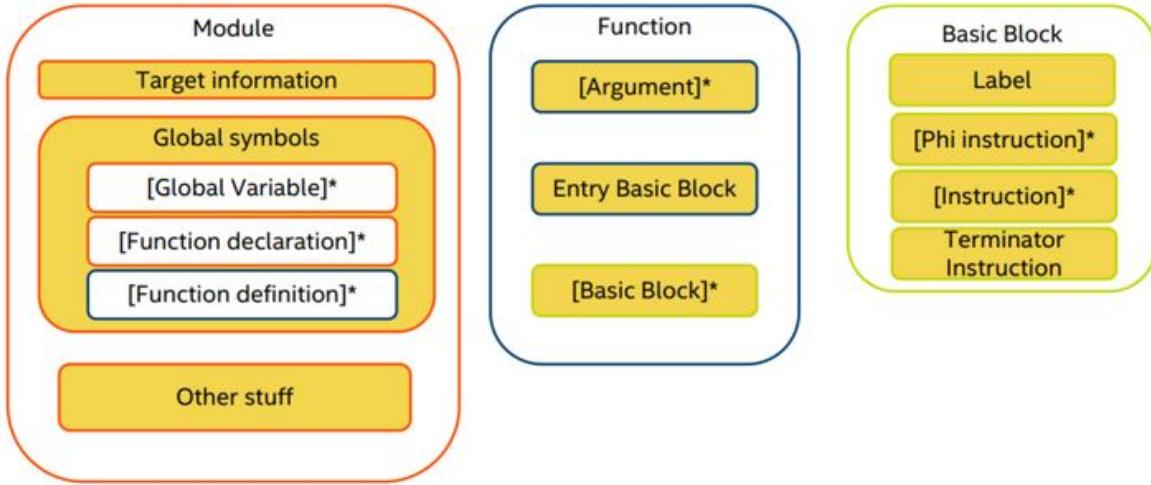
LLVM IR是**LLVM**的核心所在，通过将不同高级语言的前端转换成**LLVM IR**进行优化、链接后再传给不同的后端转换成为二进制代码，前端、优化、后端三个阶段互相解耦，这种模块化的设计使得**LLVM**优化不依赖于任何源码和目标机器。

4.2.1 IR布局

每个IR文件称为一个**Module**，它是其他所有IR对象的顶级容器，包含了目标信息、全局符号和所依赖的其他模块和符号表等对象的列表，其中全局符号又包括了全局变量、函数声明和函数定义。

函数**Function**由参数和多个基本块组成，其中第一个基本块称为entry基本块，这是函数开始执行的起点，另外**LLVM**的函数拥有独立的符号表，可以对标识符进行查询和搜索。

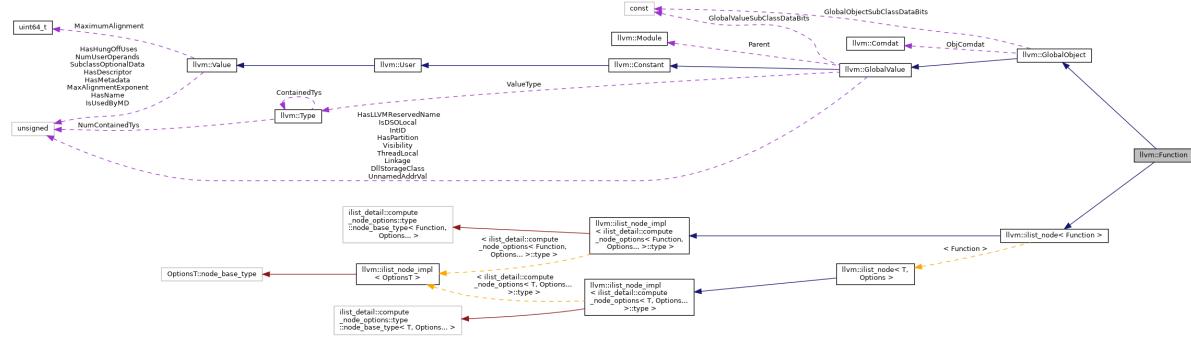
每一个基本块**Basic Block**包含了标签和各种指令的集合，标签作为指令的索引用于实现指令间的跳转，指令包含Phi指令、一般指令以及终止指令等。



4.2.2 IR上下文环境

- `LLVM::Context`: 提供用户创建变量等对象的上下文环境，尤其在多线程环境下至关重要。
- `LLVM::IRBuilder`: 提供创建LLVM指令并将其插入基础块的API。

4.2.3 IR核心类



- `llvm::Value`是程序计算的所有值的基类，可用作其他值的操作数。它表示一个类型的值，具有一个`llvm::Type*`成员和一个use list，前者指向值的类型类，后者跟踪使用了该值的其他对象，可以通过迭代器进行访问。
 - 值的存取分别可以通过`llvm::LoadInst`和`llvm::StoreInst`实现，也可以借助`IRBuilder`的`CreateLoad`和`CreateStore`实现。
- `llvm::Type`表示数据类型类，LLVM支持多种数据类型，可以通过Type ID判断类型：

Type ID	类型说明
HalfTyID	16-bit floating point type
BFloatTyID	16-bit floating point type (7-bit significand)
FloatTyID	32-bit floating point type
DoubleTyID	64-bit floating point type
X86_FP80TyID	80-bit floating point type (X87)
FP128TyID	128-bit floating point type (112-bit significand)
PPC_FP128TyID	128-bit floating point type (two 64-bits, PowerPC)
VoidTyID	type with no size
LabelTyID	Labels.
MetadataTyID	Metadata .
X86_MMXTyID	MMX vectors (64 bits, X86 specific)
X86_AMXTyID	AMX vectors (8192 bits, X86 specific)
TokenTyID	Tokens.
IntegerTyID	Arbitrary bit width integers.
FunctionTyID	Functions.
PointerTyID	Pointers.
StructTyID	Structures.
ArrayTyID	Arrays.
FixedVectorTyID	Fixed width SIMD vector type.
ScalableVectorTyID	Scalable SIMD vector type.
DXILPointerTyID	DXIL typed pointer used by DirectX target.

- `llvm::Constant`表示各种常量的基类，包括`ConstantInt`整形常量、`ConstantFP`浮点型常量、`ConstantArray`数组常量、`ConstantStruct`结构体常量等。

4.3 IR生成

4.3.1 运行环境设计

LLVM IR的生成依赖上下文环境，我们构造了一个命名空间 `contents` 来保存环境，在递归遍历AST结点的时候使用 `Contents` 中的变量进行每个结点的IR生成。`Contents` 包括的环境配置有：

- 静态全局的上下文变量和构造器变量：

```
1 | LLVM::LLVMContext context;
2 | LLVM::IRBuilder<> builder(context);
```

- 公有的模块实例、全局常量表、代码块链表：

- 模块实例是中间代码顶级容器，用于包含所有变量、函数和指令。
- 全局常量表存储了全局常量名字与值的映射关系。
- 代码块链表存储了符号表信息。

```

1 std::unique_ptr<llvm::Module> module = std::make_unique<llvm::Module>
("pascal_module", context);
2 std::map<std::string, llvm::Constant *> names_2_constants; // global
constants
3 std::vector<CodeBlock *> codeblock_list;

```

- 符号表:

- 符号表是由编译器创建和维护的一种重要的数据结构，用来存储有关各种实体出现的信息，如变量名、函数名等。它通常只存在于翻译过程中的内存中。
- 因此，我们创建了一个名为 `CodeBlock` 的类来存储我们在语义分析阶段和代码生成阶段所需要的所有信息。

```

1 class CodeBlock{
2     std::map<std::string, llvm::Value *> names_2_values; //variables
3     std::map<std::string, Our_Type::Pascal_Type*> names_2_ourtype;
4     std::map<std::string, FuncSign*> names_2_funcsign; //function or
procedure
5     std::map<std::string, llvm::Function*> names_2_functions;
6     std::vector<llvm::BasicBlock*> loop_return_blocks; // for multiple loop
break
7 }

```

`names_2_values` 是变量名与 `llvm::Value` 的映射关系，可以通过 `llvm::Value` 直接访问到变量。

`names_2_ourtype` 是从变量名映射到类型的结构。这种结构在语义分析过程中是非常重要的，因为我们会根据它来检查语义类型，并进行实时更新。

为了在不同的上下文中维护变量和函数，我们随后在 `Contents` 命名空间中引入了一个名叫 `codeblock_list` 的变量，这个结构是函数嵌套的基础。

```

1 std::vector <CodeBlock *> codeblock_list;

```

`codeblock_list` 是一个全局堆栈，它存储了迄今为止的所有上下文，当我们进入到一个新的上下文环境（即进入一个新的函数体）时，将创建一个新的 `CodeBlock` 并将其压入 `codeblock_list` 中。当我们想调用一些变量时，只需从堆栈的顶部，以找到包含该变量名的第一个代码块。而当我们离开这个上下文环境时，将 `pop` 出相应的 `CodeBlock`。

4.3.2 类型系统

每个曾经使用LLVM自己构建过编译器的人都知道，我们设计变量类型的方法与我们维护符号表的方式高度相关。

- 如果使用LLVM支持的符号表，则可以直接使用它的API，但不能扩展其给定的数据类型。
- 如果构建自己的符号表，则可以有更广泛的数据类型。

这里我们已经构建了自己的符号表，下面将给出我们自定义的数据类型。

首先，我们需要构造一个基类，我们将它命名为 `Pascal_Type`。这个类的指针将存储在符号表中，因此它应该包含它属于哪种类型的信息。类型在枚举类类型组中清楚地解释。

```

1 enum class Type_Group
2 {
3     BUILT_IN,
4     ENUMERATE,
5     SUBRANGE,
6     ARRAY,
7     STRING,
8     RECORD
9 };

```

其中：

- 内置类型 `BUILT_IN` 包括 `INT`, `FLOAT`, `BOOLEAN`, `CHAR`, `VOID`。
- `ENUMERATE` 是枚举类型。
- `SUBRANGE` 是子界类型，使用 `lowest_value .. highest_value` 表示。
- `ARRAY` 是数组类型。
- `String` 是字符串类型，这里控制字符串的长度在256以内。
- `Void` 对应 `Void` 类型。

首先是内置类型 `Buildin_Type`，我们在这个子类中的使用一个枚举类来表示它为int、float、char、boolean或void，其中void仅用于函数和过程声明。然后，我们为每种类型定义一个常量指针，以便在使用语义类型时方便表示，因为它们经常被使用。

```

1 class Buildin_Type : public Pascal_Type
2 {
3     public:
4     enum class Buildin_Type_Name
5     {
6         INT,
7         FLOAT,
8         BOOLEAN,
9         CHAR,
10        VOID
11    };
12
13    Buildin_Type_Name buildin_type_name;
14
15    Buildin_Type (Buildin_Type_Name _buildin_type_name) ;
16 };
17 const Buildin_Type INT_TYPE_INST(Buildin_Type::Buildin_Type_Name::INT);
18 const Buildin_Type REAL_TYPE_INST(Buildin_Type::Buildin_Type_Name::FLOAT);
19 const Buildin_Type CHAR_TYPE_INST(Buildin_Type::Buildin_Type_Name::CHAR);
20 const Buildin_Type
21     BOOLEAN_TYPE_INST(Buildin_Type::Buildin_Type_Name::BOOLEAN);
22 const Buildin_Type VOID_TYPE_INST(Buildin_Type::Buildin_Type_Name::VOID);
23 Pascal_Type *const INT_TYPE = (Pascal_Type *)(&INT_TYPE_INST);
24 Pascal_Type *const REAL_TYPE = (Pascal_Type *)(&REAL_TYPE_INST);
25 Pascal_Type *const CHAR_TYPE = (Pascal_Type *)(&CHAR_TYPE_INST);
26 Pascal_Type *const BOOLEAN_TYPE = (Pascal_Type *)(&BOOLEAN_TYPE_INST);
27 Pascal_Type *const VOID_TYPE = (Pascal_Type *)(&VOID_TYPE_INST);

```

接下来是我们设计的基于 `Pascal` 的数据类型，如数组和记录。首先，我们的数组类型是以一种嵌套的方式实现的，这意味着一个数组的 `element_type` 可以是另一个数组。当我们需要将类型转换为 `11vm::Type` 时，我们将以递归的方式实现它。

另一方面，`Enumerate_Type` 的实现也有点棘手。事实上，我们可以考虑枚举的定义，因为我们首先命名一些int常量，然后每个具有枚举类型的变量实际上都是一个整数。我们的 `Enumerate` 设计直接反映了这一想法。

```
1 class Enumerate_Type : public Pascal_Type
2 {
3     std::vector<std::string> enum_list;
4     Enumerate_Type(std::vector<std::string> _enum_list) ;
5 };
6
7 class Subrange_Type : public Pascal_Type
8 {
9     std::pair<int, int> begin_2_end;
10
11    Subrange_Type(int _begin, int _end);
12 };
13
14 class Array_Type : public Pascal_Type
15 {
16     Subrange_Type subrange;
17     Pascal_Type *element_type;
18
19     Array_Type(Subrange_Type _subrange, Pascal_Type *_element_type);
20
21     LLVM::ConstantInt *GetLLVMLow;
22
23     LLVM::ConstantInt *GetLLVMHigh;
24 };
25
26 class String_Type : public Pascal_Type
27 {
28     int len; // string array
29
30     String_Type(int _len = 256);
31 };
32
33 class Record_Type : public Pascal_Type
34 {
35     std::vector<std::string> name_list;
36     std::vector<Pascal_Type *> type_list;
37
38     Record_Type(std::vector<std::string> _name_list, std::vector<Pascal_Type *> _type_list);
39 };
```

4.3.3 函数信息维护

函数维护是编译器中一个复杂的部分。为了更好地操作和存储功能，我们为每个功能头设计了以下结构。

```

1 // 定义函数特征类，包括函数的返回类型，参数，局部变量（名字+类型）
2 class FuncSign
3 {
4 public:
5     FuncSign(int n_local, std::vector<std::string>
6             _name_list, std::vector<Pascal_Type*> _type_list, std::vector<bool>
7             _is_var, Pascal_Type * _return_type);
8
9 private:
10    int num_local_variables; // the number of local var
11    std::vector<Pascal_Type *> type_list;
12    std::vector<std::string> name_list;
13    std::vector<bool> is_var;
14    Pascal_Type *return_type;
15 };

```

对于每个函数声明，我们应该存储其参数的所有信息，包括参数类型，参数名称，是否为变量以及返回信息。

- name_list: 表示这些参数的名称。当在某个地方调用这个函数时，所引入的参数值将被绑定到这些名称中。
- type_list: 表示这些参数的类型。我们可以根据它进行类型检查。
- is_var: 表示该参数是否为变量。值传递和引用传递都是支持的，我们将执行不同的操作来实现这两者。
- return_type: 表示函数的返回类型。

4.3.4 自定义类型转换成LLVM类型

在 LLVM 中，类型表示值的类型。类型与上下文实例有关联。上下文内部删除重复数据类型，因此一次只有一个特定类型的实例。换句话说，在一个上下文中的所有消费者之间共享一个唯一的类型。

因此，当我们开始使用 LLVM 后端生成中间代码时，我们应该使用 LLVM API 将我们的语义表示类型转换为 `llvm::Type`，这在做 Alloc、CreateFunc 等时经常需要作为参数。除了我们自己的类型之外，我们还需要提供在编译中使用的 LLVM context。

```

1 llvm::Type *GetLLVMTy(llvm::LLVMContext &context, Pascal_Type *const
p_type);

```

我们从内置类型开始匹配，对于这些类型，我们只需要返回一个 LLVM 预定义的常量指针。

```

1 if (p_type->type_group == Pascal_Type::Type_Group::BUILT_IN)
2 {
3     if (isEqual(p_type, INT_TYPE))
4         return llvm::Type::getInt32Ty(context);
5     else if (isEqual(p_type, REAL_TYPE))
6         return llvm::Type::getDoubleTy(context);
7     else if (isEqual(p_type, CHAR_TYPE))
8         return llvm::Type::getInt8Ty(context);
9     else if (isEqual(p_type, BOOLEAN_TYPE))
10        return llvm::Type::getInt1Ty(context);
11    else if (isEqual(p_type, VOID_TYPE))
12        return llvm::Type::getVoidTy(context);
13    else
14        return nullptr;
15 }

```

对于字符串和数组，核心工作都在于创建一个`llvm::Array`，唯一的区别是：字符串不能嵌套，而在生成一个数组类型时，我们必须首先递归地获得它的`element_type`，然后返回创建的`llvm::Array`。

```
1 | else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::STRING)
2 | {
3 |     Our_Type::String_Type *str = (Our_Type::String_Type *)p_type;
4 |     return llvm::ArrayType::get(GetLLVMTy(context, CHAR_TYPE), (uint64_t)
5 | (str->len));
6 | }
7 | else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::ARRAY) {
8 |     Our_Type::Array_Type *array = (Our_Type::Array_Type *) p_type;
9 |     llvm::ArrayType *ret = nullptr;
10 |    int len = array->subrange.begin_2_end.second - array-
11 | >subrange.begin_2_end.first + 1;
12 |    std::cout << "creating array of length " << len << std::endl;
13 |
14 |    ret = llvm::ArrayType::get(GetLLVMTy(context, array->element_type),
15 | (uint64_t) len);
16 |    return ret;
17 | }
```

对于记录，我们需要使用由`llvm::StructType`提供的静态函数。这个类型显示了为什么语义类型类是必要的。如代码中所示，当我们构造一个`llvm::StructType`时，我们只提供每个组件的`llvm::Type`，但不提供它们的名称。但是，如果我们想加载或存储一个成员值，我们就会使用它们的名称来访问它们。因此，在操作Pascal记录时，语义类型和`llvm::Type`都是必需的。

```
1 | else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::RECORD)
2 | {
3 |     Our_Type::Record_Type *record = (Our_Type::Record_Type *)p_type;
4 |     std::vector<llvm::Type *> llvm_type_vec;
5 |     for (auto t : record->type_list)
6 |     {
7 |         llvm_type_vec.push_back(GetLLVMTy(context, t));
8 |     }
9 |     return llvm::StructType::get(context, llvm_type_vec);
10 | }
```

最后，对于第4.2章中提到的枚举类型，我们将枚举类型实际上变成了整数类型。

```
1 | else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::ENUMERATE)
2 | {
3 |     return llvm::Type::getInt32Ty(context);
4 | }
```

4.3.5 表达式生成

4.3.5.1 标识符生成

基于变量取值、以及LLVM函数的符号表可以实现标识符到`llvm::Value*`的映射从而返回标识符的值：

```
1 | std::shared_ptr<Custom_Result> AST_Identifier_Expression::CodeGenerate()
2 | {
3 |     //判断是否在当前code_block
```

```
4     if(Contents::GetCurrentBlock()->isValue(this->id)){
5         llvm::Type * my_type
6 =Our_Type::GetLLVMTType(Contents::context,Contents::GetVarType(this->id));
7         llvm::Value *mem = Contents::GetCurrentBlock()->names_2_values[this-
8 >id];
9         llvm::Value *value =
Contents::builder.CreateLoad(/*type=*/my_type,mem,"load identifier value");
10        return std::make_shared<Value_Result>(Contents::GetVarType(this-
11 >id), value, mem);
12    }
13    //再判断是否在最外层code_block
14    else if(Contents::codeblock_list[0]->isValue(this->id)){
15        llvm::Type * my_type
16 =Our_Type::GetLLVMTType(Contents::context,Contents::codeblock_list[0]-
17 >names_2_ourtype[this->id]);
18        llvm::Value *mem = Contents::codeblock_list[0]->names_2_values[this-
19 >id];
20        llvm::Value *value =
Contents::builder.CreateLoad(/*type=*/my_type,mem,"load identifier value");
21        return std::make_shared<Value_Result>(Contents::codeblock_list[0]-
22 >names_2_ourtype[this->id], value, mem);
23    }
24    else{
25        //无参数的函数调用
26        AST_Function_Call *func_call = new AST_Function_Call(this->id,
27 nullptr);
28        auto ret = func_call->CodeGenerate();
29        if (ret != nullptr) return ret;
30        Record_and_Output_Error(true,this->id + " is neither a variable nor
31 a no-arg function. Cannot get named value: ",this->GetLocation());
32        return nullptr;
33    }
34 }
```

4.3.5.2 数组表达式生成

基于数组元素下标的引用过程：

- 根据数组标识符查询符号表得到数组地址。
 - 根据数组标识符查询得到数组的范围类型，分为常量范围类型和变量范围类型。
 - 根据范围类型的下限和索引下标的值计算地址偏移量。
 - 根据数组地址和地址偏移量获取数组元素地址。
 - 根据元素地址加载元素值。

以上过程通过 `AST_Array_Expression` 的 `CodeGenerate` 函数实现获取元素引用:

```
1 std::shared_ptr<Custom_Result> AST_Array_Expression::CodeGenerate()
2 {
3     auto index = std::static_pointer_cast<Value_Result> (expression-
4 >CodeGenerate());
5     // 这里直接从变量表里面找
6     Pascal_Type* _array_type = Contents::GetVarType(this->id);
7     auto array = (Contents::GetCurrentBlock()->isvalue(this->id))?
8                 std::make_shared<Value_Result>(
9                     _array_type,nullptr,Contents::GetCurrentBlock()->names_2_values[this->id]);
10    std::make_shared<Value_Result>
11        (_array_type,nullptr,Contents::codeblock_list[0]->names_2_values[this->id]);
12 }
```

```

9 //判断是否在当前code_block
10 //再判断是否在最外层code_block
11
12     bool isStr = array->GetType()->type_group ==
Pascal_Type::Type_Group::STRING;
13     bool isArr = array->GetType()->type_group ==
Pascal_Type::Type_Group::ARRAY;
14     if (array == nullptr || (!isArr && !isstr)) {
15         std::cout << "array exp fail" << std::endl;
16         return nullptr;
17     }
18
19     Array_Type* array_type = (Array_Type*)(array->GetType());
20     if (!isEqual(index->GetType(), Our_Type::INT_TYPE) && !isEqual(index-
>GetType(), Our_Type::CHAR_TYPE))
21         return nullptr;
22
23     llvm::value *base; //获取数组的基址
24     if (isArr) {
25         base = array_type->GetLLVMLow(Contents::context);
26     } else {
27         base =
28     llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context), 0, true);
29
30     //考虑不是从0开始编号
31     auto offset = Contents::builder.CreateSub(index-
>GetValue(), base, "subtmp");
32     std::vector<llvm::Value *> offset_vec = {
33         llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context), 0),
34         offset
35     };
36
37     if (isArr) {
38         llvm::Type *my_type =
39     Our_Type::GetLLVMTy(Contents::context, array_type->element_type);
40         llvm::Value *tmp_mem =
41     Contents::builder.CreateGEP(/*type=my_type, */array->GetMemory(), offset_vec,
42         "ArrayCall");
43         llvm::Value *value =
44     Contents::builder.CreateLoad(/*type= */my_type, tmp_mem, "load array value");
45         //直接从name2_to_value中取
46         return std::make_shared<Value_Result>(array_type->element_type,
47             value, tmp_mem);
48     } else {
49         llvm::Type *my_type =
50     Our_Type::GetLLVMTy(Contents::context, Our_Type::CHAR_TYPE);
51         llvm::Value *tmp_mem =
52     Contents::builder.CreateGEP(/*type=my_type, */array->GetMemory(), offset_vec,
53         "ArrayCall");
54         llvm::Value *value =
55     Contents::builder.CreateLoad(/*type= */my_type, tmp_mem, "load string value");
56         return std::make_shared<Value_Result>(Our_Type::CHAR_TYPE, value,
57             tmp_mem);
58     }
59 }
```

4.3.5.3 记录表达式生成

基于记录表达式的引用过程：

- 根据 record 标识符查询符号表得到记录地址。
- 根据 record 属性id查询得到属性的类型。
- 根据 record 属性id计算地址偏移量。
- 根据记录地址和地址偏移量获取数组元素地址。
- 根据元素地址加载元素值。

以上过程通过 AST_Array_Expression 的 CodeGenerate 函数实现获取元素引用：

```
1 std::shared_ptr<Custom_Result> AST_Property_Expression::CodeGenerate()
2 {
3     // 这里直接从变量表里面找
4     auto _record_type = Contents::GetVarType(this->id);
5     auto val = (Contents::GetCurrentBlock()->isValue(this->id))?
6         std::make_shared<Value_Result>
7         (_record_type,nullptr,Contents::GetCurrentBlock()->names_2_values[this-
8 >id]):                                std::make_shared<Value_Result>
9         (_record_type,nullptr,Contents::codeblock_list[0]->names_2_values[this-
10 >id]);                                //判断是否在当前code_block
11        //再判断是否在最外层code_block
12
13        if(!_record_type->isRecord()){
14            //report error
15            return nullptr;
16        }
17        auto record_type = (Record_Type*)_record_type;
18        auto name_list = record_type->name_list;
19        auto type_list = record_type->type_list;
20        int bias = -1;
21        for(int i=0;i<name_list.size();i++){
22            if(name_list[i] == this->prop_id){
23                bias = i;
24                break;
25            }
26        }
27        if(bias == -1){
28            //report error
29            return nullptr;
30        }
31        std::vector<llvm::Value *> gep_vec =
32 { llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context),0,true),
33   (llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context),bias,true)
34 )};
35
36        llvm::Type * my_type =
37 Our_Type::GetLLVMTy(Contents::context,type_list[bias]);
38        llvm::Value * mem = Contents::builder.CreateGEP(/*type = my_type,*/*val-
39 >GetMemory(),gep_vec,"record_field");
40        llvm::Value * ret = Contents::builder.CreateLoad(/*type =
41 /*/my_type,mem,"load record type");
42
43        return std::make_shared<Value_Result>(type_list[bias],ret,mem);
```

4.3.6 二元操作

LLVM的IRBuilder集成了丰富的二元操作接口，包括ADD, SUB, MUL, DIV, CMPGE, CMPLT, CMPEQ, CMPNE, AND, OR, SREM(MOD), XOR等，实验中在 `AST_Binary_Expression` 的 `CodeGenerate` 方法中根据操作符和两个操作数返回一个二元操作结果值（整型操作和浮点操作有区别）：

```

1 std::shared_ptr<Custom_Result> AST_Binary_Expression::CodeGenerate()
2 {
3     auto l = std::static_pointer_cast<Value_Result>(left_expression-
4 >CodeGenerate());
5     auto r = std::static_pointer_cast<Value_Result>(right_expression-
6 >CodeGenerate());
7     if (l == nullptr || r == nullptr)
8         return nullptr;
9
10    // semantic check
11
12    bool is_real = isEqual(l->GetType(), REAL_TYPE);
13    if (my_operation == Operation::REALDIV)
14        is_real = true;
15    auto L = l->GetValue(), R = r->GetValue();
16    if (is_real)
17    {
18        L = Contents::builder.CreateUIToFP(L, GetLLVMTyp(Contents::context,
19                                         REAL_TYPE));
20        R = Contents::builder.CreateUIToFP(R, GetLLVMTyp(Contents::context,
21                                         REAL_TYPE));
22    }
23    switch (my_operation)
24    {
25        case Operation::GE:
26            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
27 Contents::builder.CreateFCmpGE(L, R, "cmpftmp")
28                                         :
29 Contents::builder.CreateICmpSGE(L, R, "cmpftmp"));
30        case Operation::GT:
31            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
32 Contents::builder.CreateFCmpOGT(L, R, "cmpftmp")
33                                         :
34 Contents::builder.CreateICmpSGT(L, R, "cmpftmp"));
35        case Operation::LE:
36            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
37 Contents::builder.CreateFCmpOLE(L, R, "cmpftmp")
38                                         :
39 Contents::builder.CreateICmpSLE(L, R, "cmpftmp"));
40        case Operation::LT:
41            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
42 Contents::builder.CreateFCmpOLT(L, R, "cmpftmp")
43                                         :
44 Contents::builder.CreateICmpSLT(L, R, "cmpftmp"));
45        case Operation::EQUAL:
46            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
47 Contents::builder.CreateFCmpOEQ(L, R, "cmpftmp")
48                                         :
49 
```

```

35   :
36   Contents::builder.CreateICmpEQ(L, R, "cmptmp"));
37   case Operation::UNEQUAL:
38     return std::make_shared<value_Result>(BOOLEAN_TYPE, is_real ?
39     Contents::builder.CreateFCmpONE(L, R, "cmpftmp"))
40   :
41   Contents::builder.CreateICmpNE(L, R, "cmptmp"));
42   case Operation::PLUS:
43     return std::make_shared<value_Result>(BOOLEAN_TYPE, is_real ?
44     Contents::builder.CreateFAdd(L, R, "addftmp"))
45   :
46   Contents::builder.CreateAdd(L, R, "addtmp"));
47   case Operation::MINUS:
48     return std::make_shared<value_Result>(BOOLEAN_TYPE, is_real ?
49     Contents::builder.CreateFSub(L, R, "subftmp"))
50   :
51   Contents::builder.CreateSub(L, R, "subtmp"));
52   case Operation::MUL:
53     return std::make_shared<value_Result>(BOOLEAN_TYPE, is_real ?
54     Contents::builder.CreateFMul(L, R, "mulftmp"))
55   :
56   Contents::builder.CreateMul(L, R, "multmp"));
57   case Operation::DIV:
58     if (is_real)
59     {
60       // 报告错误
61     }
62     return std::make_shared<value_Result>(INT_TYPE,
63     Contents::builder.CreateSDiv(L, R, "divtmp"));
64   case Operation::MOD:
65     if (is_real)
66     {
67       // 报告错误
68     }
69     return std::make_shared<value_Result>(INT_TYPE,
70     Contents::builder.CreateSRem(L, R, "modtmp"));
71   case Operation::REALDIV:
72     return std::make_shared<value_Result>(REAL_TYPE,
73     Contents::builder.CreateFDiv(L, R, "divftmp"));
74   case Operation::OR:
75     return std::make_shared<value_Result>(BOOLEAN_TYPE,
76     Contents::builder.CreateOr(L, R, "ortmp"));
77   case Operation::AND:
78     return std::make_shared<value_Result>(BOOLEAN_TYPE,
79     Contents::builder.CreateAnd(L, R, "andtmp"));
80   default:
81     return nullptr;
82   }
83 }
```

4.3.7 赋值语句

与标识符引用相对的是赋值语句，需要计算等式右表达式的值并赋予左表达式，分为标识符赋值，数组赋值与记录类型赋值。这三者的区别就是标识符引用和数组/记录元素引用，前者直接在符号表查询地址，后者需要根据下标和下限计算偏移量再获取元素的地址（这一部分在 4.3.5 节进行了详细解释）。

```

1 std::shared_ptr<Custom_Result> AST_Assign_Statement::CodeGenerate()
2 {
3
4     if (isDirectAssign())
5     {
6         llvm::value *left_mem = Contents::GetCurrentBlock()-
>names_2_values[identifier1];
7         // Contents::codeBlock_list[0]
8
9         auto right = std::static_pointer_cast<Value_Result>(expression1-
>CodeGenerate());
10        Contents::builder.CreateStore(right->GetValue(), left_mem);
11    }
12    else if (isArrayAssign())
13    {
14        // llvm::value* left_mem = Contents::GetCurrentBlock()-
>names_2_values[identifier1];
15        // auto expidx = std::static_pointer_cast<Value_Result>(expression1-
>CodeGenerate());
16        auto ast_array_expression = std::make_shared<AST_Array_Expression>
(this->identifier1, this->expression1);
17        std::shared_ptr<Value_Result> left_array_ret =
std::static_pointer_cast<Value_Result>(ast_array_expression-
>CodeGenerate());
18        auto right = std::static_pointer_cast<Value_Result>(expression2-
>CodeGenerate());
19
20        Contents::builder.CreateStore(right->GetValue(), left_array_ret-
>GetMemory());
21    }
22    else if (isRecordAttrAssign())
23    {
24        //• IDENTIFIER '.' IDENTIFIER ':=' expression
25        // llvm::value* left_mem = Contents::GetCurrentBlock()-
>names_2_values[identifier1];
26        // auto expidx = std::static_pointer_cast<Value_Result>(expression1-
>CodeGenerate());
27        auto ast_record_expression =
std::make_shared<AST_Property_Expression>(this->identifier1, this-
>identifier2);
28        std::shared_ptr<Value_Result> left_array_ret =
std::static_pointer_cast<Value_Result>(ast_record_expression-
>CodeGenerate());
29        auto right = std::static_pointer_cast<Value_Result>(expression1-
>CodeGenerate());
30
31        Contents::builder.CreateStore(right->GetValue(), left_array_ret-
>GetMemory());
32    }
33    return nullptr;
34 }
```

4.3.8 主函数入口

AST_Program 相当于程序的 main 函数，需要构建 main 函数的函数类型并创建函数实例，之后将 main 函数作为参数构造一个基础块作为指令的插入点，递归调用 Routine 的代码生成。

```
1 std::shared_ptr<Custom_Result> AST_Program::CodeGenerate()
2 {
3     this->program_head->CodeGenerate();
4     Contents::codeblock_list.push_back(new CodeBlock());
5     //下面定义主函数: int main(void);
6     LLVM::FunctionType * function_type =
7         LLVM::FunctionType::get(GetLLVMTy(Contents::context, INT_TYPE), false);
8     LLVM::Function * main_function = LLVM::Function::Create(
9         /*函数类型:包括返回值和函数参数*/function_type,
10        /*ExternalLink表示能够模块外使用*/LLVM::Function::ExternalLinkage,
11        /*函数名*/"main",
12        &*(Contents::module));
13     LLVM::BasicBlock *entry =
14         LLVM::BasicBlock::Create(Contents::context, "entry", main_function);
15     Contents::builder.SetInsertPoint(entry);
16     //创建返回值,正常返回0
17     this->routine->CodeGenerate();
18     Contents::builder.CreateRet(LLVM::ConstantInt::get(LLVM::Type::getInt32Ty(Contents::context), 0, true));
19     return nullptr;
20 }
```

4.3.9 Routine

Routine 包含了两个部分，一个是 Routine_Head，一个是 Routine_Body。其中 Routine_Head 包含了常量声明、变量声明、类型声明、子例程声明。而 Routine_Body 包含了函数体声明，该结点只需要依次调用子结点的 CodeGenerate 方法即可。

```
1 std::shared_ptr<Custom_Result> AST_Routine::CodeGenerate()
2 {
3     this->routine_head->CodeGenerate();
4     this->routine_body->CodeGenerate();
5     return nullptr;
6 }
7 std::shared_ptr<Custom_Result> AST_Routine_Head::CodeGenerate()
8 {
9     if(this->const_part) this->const_part->CodeGenerate();
10    if(this->type_part) this->type_part->CodeGenerate();
11    if(this->var_part) this->var_part->CodeGenerate();
12    if(this->routine_part) this->routine_part->CodeGenerate();
13    return nullptr;
14 }
15 std::shared_ptr<Custom_Result> AST_Routine_Body::CodeGenerate()
16 {
17     return this->Get_Compound_Statement()->CodeGenerate();
18 }
```

4.3.10 常量/变量声明

常量声明和变量声明相似，只不过增加了初始化和常量属性声明。

- 全局常量/变量：调用`llvm::GlobalVariable`构造全局常量/变量，通过指定`isConstant`参数来区分常量和变量。此时的初始化可以通过传递初始值作为`llvm::GlobalVariable`的参数实现。

```
1 //全局常量
2 std::shared_ptr<Custom_Result> AST_Const_Expression::CodeGenerate()
3 {
4     std::shared_ptr<Value_Result> res =
5     std::static_pointer_cast<Value_Result>(this->value->CodeGenerate());
6     llvm::GlobalVariable *constant = new llvm::GlobalVariable(
7         /*Module=*/(Contents::module),
8         /*Type=*/GetLLVMTy(Contents::context, res->GetType()),
9         /*isConstant=*/true,
10        /*Linkage=*/llvm::GlobalValue::PrivateLinkage,
11        /*Initializer=*/(llvm::Constant *)res->GetValue(), // has
12        initializer, specified below
13        /*Name=*/this->id);
14     if (Contents::codeblock_list.back()->names_2_values.count(this->id) ||
15         Contents::names_2_constants.count(this->id)) {
16         //error
17     }
18     Contents::names_2_constants[this->id] = (llvm::Constant *) (res-
19     >GetValue());
20     Contents::codeblock_list[0]->names_2_values[this->id] = constant;
21     return nullptr;
22 }
```

```
1 //全局变量
2 std::shared_ptr<Custom_Result> AST_Variable_Declaration::CodeGenerate()
3 {
4     auto name_list = std::static_pointer_cast<Name_List>(this->name_list-
5     >CodeGenerate());
6     auto type_decl = std::static_pointer_cast<Type_Result>(this->type_decl-
7     >CodeGenerate());
8
9     if(type_decl == nullptr){
10         return nullptr; //error
11     }
12     for(auto identifier : name_list->GetNameList()){
13         llvm::Type *ty = GetLLVMTy(Contents::context, type_decl-
14         >GetType());
15         if(Contents::codeblock_list.size() == 1){
16             //是全局变量
17             llvm::Constant * init_const;
18             if(type_decl->GetType()->isSimple()){
19                 init_const = llvm::Constant::getNullValue(ty);
20             }else{
21                 init_const = llvm::ConstantAggregateZero::get(ty);
22             }
23             llvm::GlobalVariable * var = new llvm::GlobalVariable(
24                 /*Module=*/(Contents::module),
25                 /*Type=*/ty,
26                 /*isConstant=*/false,
27                 /*Linkage=*/llvm::GlobalValue::CommonLinkage,
```

```

25             /*Initializer=*/
26             /*Name=*/
27             if(Contents::codeblock_list.back()-
28 >names_2_values.count(identifier)){
29                 #ifdef GEN_DEBUG
30                     std::cout << "变量重复定义" <<std::endl;
31                 #endif
32             }
33             Contents::codeblock_list.back()->names_2_values[identifier] =
34 var;
35             Contents::codeblock_list.back()->names_2_ourtype[identifier] =
36 type_decl->GetType();
37             }else{
38                 //是局部变量
39                 llvm::AllocaInst *var = Contents::builder.CreateAlloca(
40                     ty,
41                     nullptr,
42                     identifier
43                 );
44                 if(Contents::codeblock_list.back()-
45 >names_2_values.count(identifier)){
46                     //error
47                 }
48                 Contents::codeblock_list.back()->names_2_values[identifier] =
49 var;
50                 Contents::codeblock_list.back()->names_2_ourtype[identifier] =
51 type_decl->GetType();
52             }
53         }
54     }
55     return nullptr;
56 }
```

常量/变量创建之后将自动加入到当前函数的符号表或者整个模块的符号表中。

4.3.11 函数/过程声明

函数声明和过程声明类似，函数声明只需要基于过程声明增加返回值处理即可，这里把二者合在一起实现。声明包括：

- 函数类型声明
 - 参数类型考虑引用传递，所以需要处理非指针类型和指针类型。
 - 函数声明的返回值类型为实际类型，过程声明的返回值类型为空。
- 函数实例和基础块：根据函数类型创建函数实例并构建基础块作为代码插入点。
- 函数入栈：将函数实例的指针推入函数栈。
- 函数实参获取：可以通过`llvm::Function::arg_iterator`迭代遍历函数的实际参数并获取参数的值。
 - 值传递：将获取的参数值直接存到局部变量中。
 - 引用传递：通过`TheBuilder.CreateGEP`获取参数地址，并指定新的变量名，无需存储值，此
外为了在函数调用时可以区分引用传递参数和值传递参数，我们利用LLVM函数的参数属性进
行标识。
- 函数返回值声明：函数声明需要创建函数返回值变量，同时以函数名称给其命名。
- 函数体生成：调用函数体子结点生成代码。
- 函数返回：创建返回实例，函数声明返回函数名的返回值，过程声明返回空值。
- 函数出栈：将函数指针弹出栈顶，并将当前函数指针重新指向栈顶。

```

1 std::shared_ptr<custom_Result> AST_Declaration_BaseClass::CodeGenerate()
2 {
3     bool is_function = (this->declaration_type ==
4         ENUM_Declaration_Type::FUNCTION_DECLARATION);
5     auto parameters = std::static_pointer_cast<Type_List_Result>(
6         is_function ? this->Get_Function_Declaration()-
7 >Get_Function_Head()->Get_Parameters()->CodeGenerate()-
8             : this->Get_Procedure_Declaration()-
9 >Get_Procedure_Head()->Get_Parameters()->CodeGenerate()
10        );
11
12
13
14     Pascal_Type * return_type = Our_Type::VOID_TYPE;
15     std::string function_name;
16     if(is_function){
17         //有返回值
18         function_name = this->Get_Function_Declaration()-
19 >Get_Function_Head()->Get_Identifier();
20         auto return_type_result = std::static_pointer_cast<Type_Result>(
21 (this->Get_Function_Declaration()->Get_Function_Head()-
22 >Get_Simple_Type_Declaration()->CodeGenerate());
23         if(return_type_result == nullptr){
24             Record_and_Output_Error(true,"Can not recognize the return type
25 for the function definition.",this->GetLocation());
26             return nullptr;
27         }
28         return_type = return_type_result->GetType();
29     }else{
30         function_name = this->Get_Procedure_Declaration()-
31 >Get_Procedure_Head()->Get_Identifier();
32     }
33     LLVM::Type *LLVM_return_type =
34     GetLLVMTyp(Contents::context,return_type);
35     auto name_list = parameters->GetNameList();
36     std::vector<std::shared_ptr<Type_Result>> type_var_list = parameters-
37 >GetTypeList();
38     std::vector<LLVM::Type*> LLVM_type_list;
39     std::vector<Pascal_Type*> type_list;
40     std::vector<bool> var_list;
41
42     //检查是否有重名变量
43     for(int i=0;i <name_list.size();i++){
44         for(int j=i+1;j<name_list.size();j++){
45             if(name_list[i] == name_list[j]){
46                 Record_and_Output_Error(true,"The parameters in the
47 function/procedure definition are duplicated.",this->GetLocation());
48                 return nullptr;
49             }
50         }
51     }
52
53
54     // Adding local variables

```

```

46 // we must put local variables first
47 // because after we create this function,
48 // we have to add the variables to the next CodeBlock
49 // in this step, we must add the function parameters later
50 // so as to overwrite the older local variables
51 auto local_vars = Contents::GetAllLocalVarNameType();
52 std::vector<std::string> local_name_list = local_vars.first;
53 std::vector<Pascal_Type*> local_type_list = local_vars.second;
54
55 for(int i=0;i<local_name_list.size();i++){
56     name_list.push_back(local_name_list[i]);
57     type_list.push_back(local_type_list[i]);
58     var_list.push_back(true);
59
60     llvm_type_list.push_back(llvm::PointerType::getUnqual(GetLLVMType(Contents
61         ::context,local_type_list[i])));
62 }
63
64 //adding function parameters
65 for(std::shared_ptr<Type_Result> type : type_var_list){
66     type_list.push_back(type->GetType());
67     var_list.push_back(type->GetIsVal());
68
69     FuncSign *funcsign = new
70     FuncSign((int)local_name_list.size(),name_list,type_list,var_list,return_ty
71     pe);
72     llvm::FunctionType *functiontype = llvm::FunctionType::get(
73         /*返回类型*/llvm_return_type,
74         /*参数类型列表*/llvm_type_list,
75         /*isVar*/false
76     );
77     llvm::Function *function =
78     llvm::Function::Create(functiontype,llvm::GlobalVariable::ExternalLinkage,f
79     unction_name,Contents::module.get());
80
81     Contents::GetCurrentBlock()-
82     >Set_Function(function_name,function,funcsign);
83
84     //MODIFY PARAMETERS PASSING
85     Contents::codeblock_list.push_back(new CodeBlock());
86     Contents::GetCurrentBlock()->block_name = function_name;
87     Contents::GetCurrentBlock()->is_function = is_function;
88     int j = 0;
89     llvm::Function::arg_iterator arg_id;
90     for(arg_id = function->arg_begin();arg_id != function-
91     >arg_end();arg_id++,j++){
92         if(var_list[j]){

```



```

135     this->Get_Procedure_Declaration()->Get_Routine()->CodeGenerate();
136     Contents::builder.CreateRetVoid();
137 }
138
139     Contents::builder.SetInsertPoint(oldblock);
140     Contents::codeblock_list.pop_back();
141     return nullptr;
142 }
```

4.3.12 函数/过程调用

函数调用和过程调用类似：

- 通过 `Find_Funcsign` 和 `Find_Function` 从符号表查找函数参数列表以及函数指针。
- 创建函数参数向量，逐个计算参数表达式的值，为了区分值传递和引用传递，需要通过 `Function::arg_iterator` 遍历函数的参数，并判断是否具有之前标记的属性：
 - 值传递：调用代码生成计算并加载值。
 - 引用传递：直接查询符号表传递地址。
- 通过 `IRBuilder` 的 `CreateCall` 构造函数调用。

```

1 std::shared_ptr<Custom_Result> AST_Function_Call::CodeGenerate()
2 {
3     std::shared_ptr<Value_List_Result> value_list;
4     std::vector<std::shared_ptr<Value_Result>> value_vec;
5     // 判断是否含有参数
6     bool has_args = true;
7     if(this->args_list == nullptr){
8         has_args = false;
9     }else{
10        value_list = std::static_pointer_cast<Value_List_Result> (this-
11 >args_list->CodeGenerate());
12        value_vec = value_list->GetValueList();
13    }
14
15    // 获取函数名称    this->func_id;
16    using namespace std;
17    for (int i = Contents::codeblock_list.size() - 1; i >= 0; i--){
18        FuncSign *funcsign = (Contents::codeblock_list[i])->Find_FuncSign(this->func_id);
19        if(funcsign == nullptr) {
20            continue;
21        }
22        // Note the function/procedure can not be overridden in pascal, so
23        // the function is matched iff the name is matched.
24        // NameList().size include all local variables that require to be
25        // passed
26        // we should compare NameList.size() - n_local
27        // which is the actual arg size
28        if(funcsign->GetNameList().size() - funcsign->GetLocalVariablesNum() != value_vec.size()){
29            Record_and_Output_Error(true,"Can't find function" + this->func_id + ": you have "+std::to_string(value_vec.size()) + "parameters,
30            but the defined one has "
31                +std::to_string(funcsign->GetNameList().size() - funcsign->GetLocalVariablesNum()) + "parameters.",this->GetLocation());
32            return nullptr;
33        }
34    }
35 }
```

```

29 }
30     auto name_list = funcsign->GetNameList();
31     auto type_list = funcsign->GetTypeList();
32     auto var_list = funcsign->GetVarList();
33     auto return_type = funcsign->GetReturnType();
34
35     llvm::Function *callee = (Contents::codeblock_list[i])->Find_Function(this->func_id);
36     if(callee == nullptr) {
37         continue;
38     }
39     std::vector<llvm::Value*> parameters;
40
41     // 添加局部变量
42     // in generator_program.cpp, we define all locals at the head of
43     // the para list
44     int cur;
45     int n_local = funcsign->GetLocalVariablesNum();
46     for(cur = 0; cur < n_local; cur++) {
47         std::string local_name = name_list[cur];
48         if (Contents::GetCurrentBlock()-
49             >names_2_values.find(local_name) == Contents::GetCurrentBlock()-
50             >names_2_values.end()) {
51
52             parameters.push_back(nullptr);
53         } else {
54             parameters.push_back(Contents::GetCurrentBlock()-
55             >names_2_values[local_name]);
56         }
57     }
58
59     // 函数传参
60     for (auto value: value_vec){
61         if (!isEqual(value->GetType(), type_list[cur])){
62             Record_and_Output_Error(true,"Type does not match on
63             function " + this->func_id + " calling.",this->GetLocation());
64             return nullptr;
65         }
66         if (value->GetMemory() != nullptr) {
67             parameters.push_back(value->GetMemory());
68         } else {
69             Contents::temp_variable_count++;      //不重复编码
70             // here we encounter a literally const value as a parameter
71             // we add a local variable to the IRBuilder
72             // but do not reflect it in current_CodeBlock->named_values
73             // thus we do not add abnormal local variables when we
74             declare another function/procedure
75             llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
76                 GetLLVMTy(Contents::context, type_list[cur]),
77                 nullptr,
78                 "0_" + std::to_string(Contents::temp_variable_count)
79             );
79             Contents::builder.CreateStore(value->GetValue(), mem);
80             parameters.push_back(mem);
81         }
82         cur++;
83     }
84     auto ret = Contents::builder.CreateCall(callee, parameters);

```

```

80
81     if (funcsign->GetReturnType() ->type_group ==
82         Our_Type::Pascal_Type::Type_Group::STRING) {
83         // to return a str type for writeln to print
84         // we have to use its pointer
85         // to achieve this, we add a never used variable here
86         Contents::temp_variable_count++;
87
88         llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
89             GetLLVMType(Contents::context, funcsign->GetReturnType()),
90             nullptr,
91             "0_" + this->func_id +
92             std::to_string(Contents::temp_variable_count)
93             );
94         Contents::builder.CreateStore(ret, mem);
95         llvm::Type *my_type
96         =Our_Type::GetLLVMType(Contents::context, funcsign->GetReturnType());
97         llvm::Value *value =
98         Contents::builder.CreateLoad(/*type=*/my_type, mem, "load return value");
99         return std::make_shared<Value_Result>(funcsign-
100 >GetReturnType(), value, mem); //, ret->getPointerOperand()); //, "call_"+
101 node->getFuncId()
102     } else {
103         return std::make_shared<Value_Result>(funcsign-
104 >GetReturnType(), ret);
105     }
106 }
107
108 using namespace Our_Type;
109
110 // To do list : 系统调用
111 // Currently, sys_function will use no local variables that has cascade
112 relation
113 // So we do not need to deal with the locals and do it simply
114 if (Contents::isSysFunc(this->func_id)) {
115     return std::make_shared<Value_Result>(VOID_TYPE,
116     Contents::GenSysFunc(this->func_id, value_vec));
117 }
118 Record_and_Output_Error(true, "Function " + this->func_id + " not
119 found.", this->GetLocation());
120 return nullptr;
121 }
```

4.3.13 系统函数/过程

本次实验基于C语言的printf和scanf设计了读写系统函数：read(readln)和writewriteln)。

在 `Contents.cpp` 中定义了 `isSysFunc` 和 `GenSysFunc`，分别负责检验函数调用是否为系统函数，以及创建函数实例：

```

1 // 系统调用函数：包括readln 和 writeln
2 bool isSysFunc(std::string id)
3 {
4     for (auto &ch : id)
5         ch = tolower(ch);
6     if (id == "write" || id == "writeln")
7         return true;
8     if (id == "read" || id == "readln")
```

```

9     return true;
10    return false;
11 }
12
13 llvm::Value *GenSysFunc(std::string id, const
14 std::vector<std::shared_ptr<ValueResult>> &args_list)
15 {
16     for (auto &ch : id)
17         ch = tolower(ch);
18     if (id == "write")
19         return GenSysWrite(args_list, false);
20     if (id == "writeln")
21         return GenSysWrite(args_list, true);
22     if (id == "read")
23         return GenSysRead(args_list, false);
24     if (id == "readln")
25         return GenSysRead(args_list, true);
26     // report error
27     return nullptr;
28 }
```

在函数调用时，检测是否出现 `read` 和 `write` 函数，出现则在生成代码时创建函数实例：

- 构造函数实参，根据参数类型指定输出的格式化字符，同时生成参数值到参数向量

类型	格式化字符	备注
integer	%d	
char	%c	
bool	%d	false对应0, true对应1
real	%lf	不能为%f, 因为llvm::Double类型无法用%f输入

- 判断是否换行，`writeln` 函数在格式字符串末尾增加换行符
- 拼接格式化字符串和参数向量
- 通过IRBuilder调用函数

```

1 llvm::Value *GenSyswrite(const std::vector<std::shared_ptr<ValueResult>>
2 &args_list, bool new_line)
3 {
4     static llvm::Function *llvm_printf = nullptr;
5     if (llvm_printf == nullptr)
6     {
7         // register printf
8         std::vector<llvm::Type *> arg_types =
9             {llvm::Type::getInt8PtrTy(Contents::context)};
10        llvm::FunctionType *func_type = llvm::FunctionType::get(
11            llvm::Type::getInt32Ty(Contents::context),
12            arg_types,
13            true);
14        llvm::Function *func = llvm::Function::Create(
15            func_type,
16            llvm::Function::ExternalLinkage,
17            "printf",
18            &*(Contents::module));
```

```

17     func->setCallingConv(llvm::CallingConv::C);
18     llvm_printf = func;
19 }
20 std::string format;
21 std::vector<llvm::Value *> printf_args;
22 printf_args.emplace_back(nullptr);
23 for (auto arg : args_list)
24 {
25     Our_Type::Pascal_Type *tp = arg->GetType();
26     if (tp->isIntegerTy())
27     {
28         format += "%d";
29         printf_args.emplace_back(arg->GetValue());
30     }
31     else if (tp->isFloatingPointTy())
32     {
33         format += "%lf";
34         printf_args.emplace_back(arg->GetValue());
35     }
36     else if (tp->isCharTy())
37     {
38         format += "%c";
39         printf_args.emplace_back(arg->GetValue());
40     }
41     else if (tp->isString())
42     {
43         format += "%s";
44         printf_args.emplace_back(arg->GetMemory());
45     }
46     else
47     {
48         std::cerr << "[write/writeLn] Unsupported type" << std::endl;
49         return nullptr;
50     }
51     if (new_line)
52     {
53         format += "\n";
54     }
55     printf_args[0] = Contents::builder.CreateGlobalStringPtr(format,
56 "printf_format");
57     }
58     return Contents::builder.CreateCall(llvm_printf, printf_args,
59 "call_printf");
60 }
```

```

1 llvm::Value *GenSysRead(const std::vector<std::shared_ptr<ValueResult>>
2 &args_list, bool new_line)
3 {
4     static llvm::Function *llvm_scanf = nullptr;
5     if (llvm_scanf == nullptr)
6     {
7         // register printf
8         std::vector<llvm::Type *> arg_types =
9         {llvm::Type::getInt8PtrTy(Contents::context)};
10        llvm::FunctionType *func_type = llvm::FunctionType::get(
11            llvm::Type::getInt32Ty(Contents::context),
12            arg_types,
```

```

11         true);
12     llvm::Function *func = llvm::Function::Create(
13         func_type,
14         llvm::Function::ExternalLinkage,
15         "scanf",
16         &*(Contents::module));
17     func->setCallingConv(llvm::callingConv::C);
18     llvm_scanf = func;
19 }
20 std::string format;
21 std::vector<llvm::Value *> scanf_args;
22 scanf_args.emplace_back(nullptr);
23 for (auto arg : args_list)
24 {
25     Our_Type::Pascal_Type *tp = arg->GetType();
26     if (tp->isIntegerTy())
27     {
28         format += "%d";
29     }
30     else if (tp->isFloatingPointTy())
31     {
32         format += "%lf";
33     }
34     else if (tp->isCharTy())
35     {
36         format += "%c";
37     }
38     else if (tp->isString())
39     {
40         format += "%s";
41     }
42     else
43     {
44         std::cerr << "[read/readln] Unsupported type" << std::endl;
45         return nullptr;
46     }
47     scanf_args.emplace_back(arg->GetMemory());
48     if (new_line)
49     {
50         format += "%*[^\n]";
51     }
52     scanf_args[0] = Contents::builder.CreateGlobalStringPtr(format,
53     "scanf_format");
54     }
55     llvm::Value *ret = Contents::builder.CreateCall(llvm_scanf, scanf_args,
56     "call_scanf");
57     if (new_line)
58     {
59         // consume \n
60         Contents::builder.CreateCall(llvm_scanf,
61         Contents::builder.CreateGlobalStringPtr("%*c", "scanf_newline"),
62         "call_scanf");
63     }
64     return ret;
65 }
```

4.3.14 分支语句

4.3.14.1 if语句

if语句可以被抽象为四个基础块：

- if_Cond: 条件判断基础块，计算条件表达式的值并创建跳转分支。
- if_then: 条件为真执行的基础块，结束之后跳转到if_continue基础块。
- if_else: 条件为假执行的基础块，结束之后跳转到if_continue基础块，对于没有else部分的分支语句我们创建一个空的else基础块来达到简化代码的目的。
- if_continue: 语句结束之后的基础块，作为后面代码的插入点。

```
1 std::shared_ptr<Custom_Result> AST_If_Statement::CodeGenerate(){
2     // std::cout << "hello" << std::endl;
3     llvm::Function *function = Contents::builder.GetInsertBlock()-
4         >getParent();
5
6     llvm::BasicBlock *then_stmt_block =
7         llvm::BasicBlock::Create(Contents::context, "if_then", function);
8     llvm::BasicBlock *else_stmt_block =
9         llvm::BasicBlock::Create(Contents::context, "if_else", function);
10
11    llvm::BasicBlock *continue_stmt_block =
12        llvm::BasicBlock::Create(Contents::context, "if_continue", function);
13        //continue
14
15    auto condition_ret = std::static_pointer_cast<Value_Result>(expression-
16        >CodeGenerate());
17    //TODO: no expression / expression type not bool error check
18    //contiuete br
19    Contents::builder.CreateCondBr(condition_ret->GetValue(),
20        then_stmt_block, else_stmt_block);
21    Contents::builder.SetInsertPoint(then_stmt_block); // then
22    this->statement->CodeGenerate(); // do then
23    Contents::builder.CreateBr(continue_stmt_block); // unconditional br
24    Label instruction
25    Contents::builder.SetInsertPoint(else_stmt_block); // else
26    this->else_clause->CodeGenerate();
27    Contents::builder.CreateBr(continue_stmt_block); // continue out
28    Contents::builder.SetInsertPoint(continue_stmt_block); // else
29
30    return nullptr;
31 }
```

4.3.14.2 case语句

case语句可以看作多个if语句，本实验将其抽象为多个基础块对和一个after基础块，事先计算好表达式的值之后，进入第一个基础块对，每个基础块对的第一块判断条件值是否与当前值相等，相等则跳转到第二块，第二块执行完毕后跳转到after基础块执行case语句之后的代码，否则跳转到下一对基础块对，下一个基础块对同样如此，直到最后一个基础块对时，不管是否相等都需要跳转到after基础块。

```
1 std::shared_ptr<Custom_Result> AST_Case_Statement::CodeGenerate(){
2     // std::cout << "hello" << std::endl;
3     llvm::Function* function = Contents::builder.GetInsertBlock()-
4         >getParent(); // for Create parent function
5     auto case_expression_vec = case_expression_list->case_expression_list;
6     int case_num = case_expression_vec.size();
```

```

6     std::vector<llvm::BasicBlock *> condition_block_vec;
7     std::vector<llvm::BasicBlock *> handle_block_vec;
8
9     for(int i=0; i<case_num; i++){
10
11         condition_block_vec.push_back(llvm::BasicBlock::Create(Contents::context,
12             "case_condition", function));
13
14         handle_block_vec.push_back(llvm::BasicBlock::Create(Contents::context,
15             "case_handle", function));
16         }
17         auto endcase_block = llvm::BasicBlock::Create(Contents::context,
18             "case_endcase", function);
19
20         for(int i=0; i<case_num; i++){
21             //use binary expression for convenience
22             //TODO: reconstruct
23             Contents::builder.SetInsertPoint(condition_block_vec[i]);
24             // compare
25             bool_cmp = new
26             AST_Binary_Expression(AST_Binary_Expression::Operation::EQUAL, this-
27             >expression, case_expression_vec[i]->e);
28         }
29         return nullptr;
30     }

```

4.3.15 循环语句

4.3.15.1 for语句

for语句可以被抽象为四个基础块：

- for_start: 定义一个常量，负责每次循环变量的自增或是自减。
- for_condition: 计算条件表达式的值，判断循环条件，为真进入for_handle块，否则进入for_end块。
- for_handle: 条件为真时执行的循环体基础块，完成循环变量的递增或递减，之后跳转到for_condition基础块。
- for_end: 跳出循环之后执行的基础块，作为之后代码的插入点。

```

1 std::shared_ptr<Custom_Result> AST_For_Statement::CodeGenerate(){
2     llvm::Function * function = Contents::builder.GetInsertBlock()-
3     >getParent();
4     llvm::BasicBlock* for_start_block =
5         llvm::BasicBlock::Create(Contents::context, "for_start", function);
6     llvm::BasicBlock* for_handle_block =
7         llvm::BasicBlock::Create(Contents::context, "for_handle", function);
8     llvm::BasicBlock* for_condition_block =
9         llvm::BasicBlock::Create(Contents::context, "for_condition", function);
10    llvm::BasicBlock* for_end_block =
11        llvm::BasicBlock::Create(Contents::context, "for_end", function);
12
13    // TODO: continue
14    Contents::GetCurrentBlock()-
15    >loop_return_blocks.push_back(for_end_block);
16
17    //start
18    Contents::builder.CreateBr(for_start_block);

```

```

13     Contents::builder.SetInsertPoint(for_start_block);
14
15     // definition once
16     // TODO enum class step add
17     std::cout << "begin const value" << std::endl;
18     auto ast_const_value = new AST_Const_Value(
19         this->my_direction->isTo() ? "1" : "-1",
20         AST_Const_Value::Value_Type::INT
21     );
22     auto const_value_ret = std::static_pointer_cast<Value_Result>
23     (ast_const_value->CodeGenerate());
24
25     // reuse ast code generator
26     std::cout << "begin for assign" << std::endl;
27     auto ast_assign_statement = new AST_Assign_Statement(this->identifier,
28     this->expression1);
29     ast_assign_statement->CodeGenerate(); // local varaible
30
31     std::cout << "begin first cmp" << std::endl;
32     // compare iteration assign
33     auto ast_cmp_statement = new AST_Binary_Expression(
34         this->my_direction->isTo() ?
35         AST_Binary_Expression::Operation::GT :
36         AST_Binary_Expression::Operation::LT,
37         this->expression1, this->expression2
38     );
39     auto ast_cmp_ret = std::static_pointer_cast<Value_Result>
40     (ast_cmp_statement->CodeGenerate());
41
42     //branch
43     std::cout << "begin first condition br" << std::endl;
44     Contents::builder.CreateCondBr(ast_cmp_ret->GetValue(), for_end_block,
45     for_handle_block);
46
47     // handle
48     std::cout << "begin handle" << std::endl;
49     Contents::builder.SetInsertPoint(for_handle_block);
50     this->statement->CodeGenerate();
51
52     // condition: step -> cmp ? contine(->handle) or end
53     std::cout << "begin condition" << std::endl;
54     Contents::builder.CreateBr(for_condition_block);
55     Contents::builder.SetInsertPoint(for_condition_block);
56     // step
57     std::cout << "begin step" << std::endl;
58
59     auto left_id = new AST_Identifier_Expression(this->identifier); // left
60     value for step
61     auto left_ret = std::static_pointer_cast<Value_Result>(left_id-
62     >CodeGenerate());
63
64     std::cout << "begin step add" << std::endl;
65     llvm::Value* add_ret = Contents::builder.CreateAdd(left_ret-
66     >GetValue(), const_value_ret->GetValue(), "tmpadd");
67
68     std::cout << "begin step store" << std::endl;
69     Contents::builder.CreateStore(add_ret, left_ret->GetMemory());

```

```

62     // auto ast_step_statement = new AST_Binary_Expression( // has symbol
63     //     AST_Binary_Expression::Operation::PLUS,
64     //     left_ret, ast_const_value_expression
65     // );
66
67
68     // compare (loop end )
69     std::cout << "begin compare" << std::endl;
70     left_id = new AST_Identifier_Expression(this->identifier);
71     ast_cmp_statement = new AST_Binary_Expression(
72         this->my_direction->isTo() ?
73             AST_Binary_Expression::Operation::GT :
74             AST_Binary_Expression::Operation::LT,
75             left_id, this->expression2
76         );
77     ast_cmp_ret = std::static_pointer_cast<Value_Result>(ast_cmp_statement-
78 >CodeGenerate());
79
80     Contents::builder.CreateCondBr(ast_cmp_ret->GetValue(), for_end_block,
81     for_handle_block);
82
83     Contents::builder.CreateBr(for_end_block);
84     Contents::builder.SetInsertPoint(for_end_block);
85     Contents::GetCurrentBlock()->loop_return_blocks.pop_back();
86
87     std::cout << "for success" << std::endl;
88     return nullptr;
89
90 }

```

4.3.15.2 while语句

while语句类似for语句，区别在于loop基础块中没有循环变量的递增、递减操作：

```

1 std::shared_ptr<Custom_Result> AST_While_Statement::CodeGenerate()
2 {
3     // 初始化各个BasicBlock
4     LLVM::Function *func = Contents::builder.GetInsertBlock()->getParent();
5     LLVM::BasicBlock *conditon_block =
6     LLVM::BasicBlock::Create(Contents::context, "while_condition", func);
7     LLVM::BasicBlock *body_block =
8     LLVM::BasicBlock::Create(Contents::context, "while_body", func);
9     LLVM::BasicBlock *end_block =
10    LLVM::BasicBlock::Create(Contents::context, "while_end", func);
11
12    Contents::GetCurrentBlock()->loop_return_blocks.push_back(end_block);
13
14    Contents::builder.CreateBr(conditon_block);
15    Contents::builder.SetInsertPoint(conditon_block);
16
17    auto cond_res = std::static_pointer_cast<Value_Result>(this->expression-
18 >CodeGenerate());
19
20    if (cond_res == nullptr || !isEqual(cond_res->GetType(), BOOLEAN_TYPE)){
21        Record_and_Output_Error(true, "Invalid condition in while
22 statement.", this->GetLocation());
23    }

```

```

18     return nullptr;
19 }
20
21 // 条件判断, 是否结束循环
22 Contents::builder.CreateCondBr(cond_res->GetValue(), body_block,
end_block);
23
24 // 在body_block生成需要执行的while主体
25 Contents::builder.SetInsertPoint(body_block);
26 this->statement->CodeGenerate();
27
28 // 无条件跳转回condition_block 进行结束条件的判断
29 Contents::builder.CreateBr(condition_block);
30 Contents::builder.SetInsertPoint(end_block);
31
32 Contents::GetCurrentBlock()->loop_return_blocks.pop_back();
33
34 return nullptr;
35 }
```

4.3.15.3 repeat语句

repeat语句与while语句类似，区别在于先进入loop基础块，因此loop块一定会执行：

```

1 std::shared_ptr<Custom_Result> AST_Repeat_Statement::CodeGenerate()
2 {
3     // 初始化各个BasicBlock
4     LLVM::Function *func = Contents::builder.GetInsertBlock()->getParent();
5     LLVM::BasicBlock *condition_block =
6     LLVM::BasicBlock::Create(Contents::context, "repeat_condition", func);
7     LLVM::BasicBlock *body_block =
8     LLVM::BasicBlock::Create(Contents::context, "repeat_body", func);
9     LLVM::BasicBlock *continue_block =
10    LLVM::BasicBlock::Create(Contents::context, "repeat_continue", func);
11
12    Contents::GetCurrentBlock()-
13    >loop_return_blocks.push_back(continue_block);
14
15    // 进入body主体
16    Contents::builder.CreateBr(body_block);
17
18    // 在body_block生成需要执行的while主体
19    Contents::builder.SetInsertPoint(body_block);
20    this->statement_list->CodeGenerate();
21
22    // 无条件跳转回condition_block 进行结束条件的判断
23    Contents::builder.CreateBr(condition_block);
24    Contents::builder.SetInsertPoint(condition_block);
25
26    auto cond_res = std::static_pointer_cast<Value_Result> (this-
27    >expression->CodeGenerate());
28    if (cond_res == nullptr || !isEqual(cond_res->GetType(), BOOLEAN_TYPE)){
29        Record_and_Output_Error(true, "Invalid condition in repeat
30 statement.", this->GetLocation());
31        return nullptr;
32    }
33 }
```

```
28     Contents::builder.CreateCondBr(cond_res->GetValue(), continue_block, body_block);
29     Contents::builder.SetInsertPoint(continue_block);
30
31     Contents::GetCurrentBlock() -> loop_return_blocks.pop_back();
32     return nullptr;
33 }
```

第五章 优化考虑

常量折叠

常量折叠是一种非常常见和重要的优化，在LLVM生成IR的时候支持通过IRBuilder进行常量折叠优化，而不需要通过AST中的任何额外操作来提供支持。在调用IRBuilder时它会自动检查是否存在常量折叠的机会，如果有则直接返回常量而不是创建计算指令。以下是一个常量折叠的例子：

- 优化前

```
1 define double @test(double %x) {
2 entry:
3     %addtmp = fadd double 2.000000e+00, 1.000000e+00
4     %addtmp1 = fadd double %addtmp, %x
5     ret double %addtmp1
6 }
```

- 优化后

```
1 define double @test(double %x) {
2 entry:
3     %addtmp = fadd double 3.000000e+00, %x
4     ret double %addtmp
5 }
```

第六章 代码生成

6.1 选择目标机器

LLVM 支持本地交叉编译。我们可以将代码编译为当前计算机的体系结构，也可以像针对其他体系结构一样轻松地进行编译。LLVM 提供了 `sys::getDefaultTargetTriple`，它返回当前计算机的目标三元组：

```
1 auto TargetTriple = sys::getDefaultTargetTriple();
```

在获取Target前，初始化所有目标以发出目标代码：

```
1 InitializeAllTargetInfos();
2 InitializeAllTargets();
3 InitializeAllTargetMCs();
4 InitializeAllAsmParsers();
5 InitializeAllAsmPrinters();
```

使用目标三元组获得 Target：

```

1 std::string Error;
2 auto Target = TargetRegistry::lookupTarget(TargetTriple, Error);
3
4 // Print an error and exit if we couldn't find the requested target.
5 // This generally occurs if we've forgotten to initialise the
6 // TargetRegistry or we have a bogus target triple.
7 if (!Target) {
8     errs() << Error;
9     return 1;
10 }

```

`TargetMachine` 类提供了我们要定位的机器的完整机器描述：

```

1 auto CPU = "generic";
2 auto Features = "";
3
4 TargetOptions opt;
5 auto RM = Optional<Reloc::Model>();
6 auto TargetMachine = Target->createTargetMachine(TargetTriple, CPU, Features,
opt, RM);

```

6.2 配置 Module

配置模块，以指定目标和数据布局，可以方便了解目标和数据布局。

```

1 auto TheTargetMachine =
2     Target->createTargetMachine(TargetTriple, CPU, Features, opt, RM);

```

6.3 生成目标代码

1. 先定义要将文件写入的位置

```

1 auto Filename = input_file_name + ".o"; //current directory
2 if(system(("rm " +Filename).c_str()) == -1){
3     std::cout << "error in rm .o" << std::endl;
4 }
5 std::error_code EC;
6 raw_fd_ostream dest(Filename, EC, sys::fs::OF_None);
7
8 if (EC)
9 {
10     errs() << "Could not open file: " << EC.message();
11     return 1;
12 }

```

2. 定义一个发出目标代码的过程，然后运行该 pass

```

1 legacy::PassManager pass;
2 auto FileType = CGFT_ObjectFile;
3
4 if (TheTargetMachine->addPassesToEmitFile(pass, dest, nullptr, FileType))
5 {
6     errs() << "TheTargetMachine can't emit a file of this type";
7     return 1;
8 }
9
10 pass.run((*(Contents::module)));
11 dest.flush();

```

更简便的方法是使用LLVM自带的工具套件： llvm-as 和 llc

- llvm-as 可以将生成的IR文件编译为以.bc为后缀的字节码
- llc 可以将字节码编译成以.s为后缀的汇编代码

```

1 llvm-as test.ll -o test.bc
2 llc -march=x86_64 test.bc -o test.s

```

第七章 测试案例

7.1 数据类型测试

7.1.1 内置类型测试

- 测试代码

```

1 program buildIntTypeTest;
2 const
3     ic = 3;
4     rc = 3.14;
5     bc = false;
6     cc = 'b';
7 var
8     iv : integer;
9     rv : real;
10    bv : boolean;
11    cv : char;
12
13 begin
14     iv := 1;
15     rv := 99.9;
16     bv := true;
17     cv := 'a';
18     writeln(ic);
19     writeln(rc);
20     writeln(bc);
21     writeln(cc);
22     writeln(iv);
23     writeln(rv);
24     writeln(bv);
25     writeln(cv);
26 end
27 .

```

- IR

```

1 ; ModuleID = 'pascal_module'
2 source_filename = "pascal_module"
3
4 @ic = private constant i32 3
5 @rc = private constant double 3.140000e+00
6 @bc = private constant i1 false
7 @cc = private constant i8 98
8 @iv = common global i32 0
9 @rv = common global double 0.000000e+00
10 @bv = common global i1 false
11 @cv = common global i8 0
12 @printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
13 @printf_format.1 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
align 1
14 @printf_format.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
1
15 @printf_format.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align
1
16 @printf_format.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
1
17 @printf_format.5 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
align 1
18 @printf_format.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
1
19 @printf_format.7 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align
1
20
21 define i32 @main() {
22 entry:
23   store i32 1, i32* @iv, align 4
24   store double 9.990000e+01, double* @rv, align 8
25   store i1 true, i1* @bv, align 1
26   store i8 97, i8* @cv, align 1
27   %"load identifier value" = load i32, i32* @ic, align 4
28   %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load identifier
value")
29   %"load identifier value1" = load double, double* @rc, align 8
30   %call_printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5
x i8], [5 x i8]* @printf_format.1, i32 0, i32 0), double %"load identifier
value1")
31   %"load identifier value3" = load i1, i1* @bc, align 1
32   %call_printf4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format.2, i32 0, i32 0), i1 %"load identifier
value3")
33   %"load identifier value5" = load i8, i8* @cc, align 1
34   %call_printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format.3, i32 0, i32 0), i8 %"load identifier
value5")
35   %"load identifier value7" = load i32, i32* @iv, align 4
36   %call_printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format.4, i32 0, i32 0), i32 %"load identifier
value7")
37   %"load identifier value9" = load double, double* @rv, align 8

```

```

38  %call_printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
39    ([5 x i8], [5 x i8]* @printf_format.5, i32 0, i32 0), double %"load
40    identifier value9")
41  %"load identifier value11" = load i1, i1* @bv, align 1
42  %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
43    ([4 x i8], [4 x i8]* @printf_format.6, i32 0, i32 0), i1 %"load identifier
44    value11")
45  %"load identifier value13" = load i8, i8* @cv, align 1
46  %call_printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
47    ([4 x i8], [4 x i8]* @printf_format.7, i32 0, i32 0), i8 %"load identifier
      value13")
48  ret i32 0
49 }
50
51 declare i32 @printf(i8*, ...)
52

```

- 汇编指令

```

1   .text
2   .file  "pascal_module"
3   .globl main                                # -- Begin function main
4   .p2align 4, 0x90
5   .type  main,@function
6   main:                                     # @main
7     .cfi_startproc
8   # %bb.0:                                    # %entry
9     subl  $12, %esp
10    .cfi_def_cfa_offset 16
11    movl  $1, iv
12    movl  $1079572889, rv+4                  # imm = 0x4058F999
13    movl  $-1717986918, rv                   # imm = 0x9999999A
14    movb  $1, bv
15    movb  $97, cv
16    movl  .Lic, %eax
17    movl  %eax, 4(%esp)
18    movl  $.Lprintf_format, (%esp)
19    calll printf@PLT
20    fldl  .Lrc
21    fstpl 4(%esp)
22    movl  $.Lprintf_format.1, (%esp)
23    calll printf@PLT
24    movzb1 .Lbc, %eax
25    movl  %eax, 4(%esp)
26    movl  $.Lprintf_format.2, (%esp)
27    calll printf@PLT
28    movzb1 .Lcc, %eax
29    movl  %eax, 4(%esp)
30    movl  $.Lprintf_format.3, (%esp)
31    calll printf@PLT
32    movl  iv, %eax
33    movl  %eax, 4(%esp)
34    movl  $.Lprintf_format.4, (%esp)
35    calll printf@PLT
36    fldl  rv
37    fstpl 4(%esp)
38    movl  $.Lprintf_format.5, (%esp)

```

```
39    calll  printf@PLT
40    movzb1 bv, %eax
41    movl  %eax, 4(%esp)
42    movl  $.Lprintf_format.6, (%esp)
43    calll  printf@PLT
44    movzb1 cv, %eax
45    movl  %eax, 4(%esp)
46    movl  $.Lprintf_format.7, (%esp)
47    calll  printf@PLT
48    xorl  %eax, %eax
49    addl  $12, %esp
50    .cfi_def_cfa_offset 4
51    retl
52.Lfunc_end0:
53    .size   main, .Lfunc_end0-main
54    .cfi_endproc
55                                # -- End function
56    .type   .Lic,@object          # @ic
57    .section .rodata,"a",@progbits
58    .p2align 2
59.Lic:
60    .long   3                  # 0x3
61    .size   .Lic, 4
62
63    .type   .Lrc,@object          # @rc
64    .p2align 3
65.Lrc:
66    .quad   0x40091eb851eb851f      # double 3.1400000000000001
67    .size   .Lrc, 8
68
69    .type   .Lbc,@object          # @bc
70.Lbc:
71    .byte   0                  # 0x0
72    .size   .Lbc, 1
73
74    .type   .Lcc,@object          # @cc
75.Lcc:
76    .byte   98                 # 0x62
77    .size   .Lcc, 1
78
79    .type   iv,@object          # @iv
80    .comm   iv,4,4
81    .type   rv,@object          # @rv
82    .comm   rv,8,8
83    .type   bv,@object          # @bv
84    .comm   bv,1,1
85    .type   cv,@object          # @cv
86    .comm   cv,1,1
87    .type   .Lprintf_format,@object      # @printf_format
88    .section .rodata.str1.1,"aMS",@progbits,1
89.Lprintf_format:
90    .asciz  "%d\n"
91    .size   .Lprintf_format, 4
92
93    .type   .Lprintf_format.1,@object      # @printf_format.1
94.Lprintf_format.1:
95    .asciz  "%lf\n"
96    .size   .Lprintf_format.1, 5
```

```

97     .type    .Lprintf_format.2,@object      # @printf_format.2
98     .Lprintf_format.2:
99         .asciz  "%d\n"
100        .size   .Lprintf_format.2, 4
102
103        .type    .Lprintf_format.3,@object      # @printf_format.3
104     .Lprintf_format.3:
105         .asciz  "%c\n"
106        .size   .Lprintf_format.3, 4
107
108        .type    .Lprintf_format.4,@object      # @printf_format.4
109     .Lprintf_format.4:
110         .asciz  "%d\n"
111        .size   .Lprintf_format.4, 4
112
113        .type    .Lprintf_format.5,@object      # @printf_format.5
114     .Lprintf_format.5:
115         .asciz  "%lf\n"
116        .size   .Lprintf_format.5, 5
117
118        .type    .Lprintf_format.6,@object      # @printf_format.6
119     .Lprintf_format.6:
120         .asciz  "%d\n"
121        .size   .Lprintf_format.6, 4
122
123        .type    .Lprintf_format.7,@object      # @printf_format.7
124     .Lprintf_format.7:
125         .asciz  "%c\n"
126        .size   .Lprintf_format.7, 4
127
128     .section    ".note.GNU-stack","",@progbits
129

```

- 运行结果

```

lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli buildInTypeTest.pas.ll
3
3.140000
0
b
1
99.900000
1
a

```

7.1.2 数组类型测试

- 测试代码

```

1 program ArrayTest;
2 const
3     l = 1;
4     r = 6;
5 var
6     iv : integer;
7     a : array[l..r] of integer;
8
9 begin

```

```

10    iv := 0;
11    for iv := 1 to r do
12    begin
13        a[iv] := iv;
14    end;
15    for iv := 1 to r do
16    begin
17        writeln(a[iv]);
18    end;
19 end
20 .

```

- IR

```

1 ; ModuleID = 'pascal_module'
2 source_filename = "pascal_module"
3
4 @l = private constant i32 1
5 @r = private constant i32 6
6 @iv = common global i32 0
7 @a = common global [6 x i32] zeroinitializer
8 @printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
9
10 define i32 @main() {
11 entry:
12     store i32 0, i32* @iv, align 4
13     br label %for_start
14
15 for_start:                                ; preds = %entry
16     %"load identifier value" = load i32, i32* @l, align 4
17     store i32 %"load identifier value", i32* @iv, align 4
18     %"load identifier value1" = load i32, i32* @l, align 4
19     %"load identifier value2" = load i32, i32* @r, align 4
20     %cmptmp = icmp sgt i32 %"load identifier value1", %"load identifier
21     value2"
22     br i1 %cmptmp, label %for_end, label %for_handle
23
24 for_handle:                                ; preds = %for_condition,
25 %for_start
26     %"load identifier value3" = load i32, i32* @iv, align 4
27     %subtmp = sub i32 %"load identifier value3", 1
28     %ArrayCall = getelementptr [6 x i32], [6 x i32]* @a, i32 0, i32 %subtmp
29     %"load array value" = load i32, i32* %ArrayCall, align 4
30     %"load identifier value4" = load i32, i32* @iv, align 4
31     store i32 %"load identifier value4", i32* %ArrayCall, align 4
32     br label %for_condition
33
34 for_condition:                            ; preds = %for_handle
35     %"load identifier value5" = load i32, i32* @iv, align 4
36     %tmpadd = add i32 %"load identifier value5", 1
37     store i32 %tmpadd, i32* @iv, align 4
38     %"load identifier value6" = load i32, i32* @iv, align 4
39     %"load identifier value7" = load i32, i32* @r, align 4
40     %cmptmp8 = icmp sgt i32 %"load identifier value6", %"load identifier
41     value7"
42     br i1 %cmptmp8, label %for_end, label %for_handle
43     br label %for_end

```

```

41
42     for_end:                                ; preds = %for_condition,
43     %for_condition, %for_start
44         br label %for_start9
45
45     for_start9:                               ; preds = %for_end
46     %"load identifier value13" = load i32, i32* @l, align 4
47     store i32 %"load identifier value13", i32* @iv, align 4
48     %"load identifier value14" = load i32, i32* @l, align 4
49     %"load identifier value15" = load i32, i32* @r, align 4
50     %cmptmp16 = icmp sgt i32 %"load identifier value14", %"load identifier
51     value15"
52     br i1 %cmptmp16, label %for_end12, label %for_handle10
53
53     for_handle10:                            ; preds =
54     %for_condition11, %for_start9
55     %"load identifier value17" = load i32, i32* @iv, align 4
56     %subtmp18 = sub i32 %"load identifier value17", 1
57     %ArrayCall19 = getelementptr [6 x i32], [6 x i32]* @a, i32 0, i32
58     %subtmp18
59     %"load array value20" = load i32, i32* %ArrayCall19, align 4
60     %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
61     x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load array value20")
62     br label %for_condition11
63
61     for_condition11:                         ; preds = %for_handle10
62     %"load identifier value21" = load i32, i32* @iv, align 4
63     %tmpadd22 = add i32 %"load identifier value21", 1
64     store i32 %tmpadd22, i32* @iv, align 4
65     %"load identifier value23" = load i32, i32* @iv, align 4
66     %"load identifier value24" = load i32, i32* @r, align 4
67     %cmptmp25 = icmp sgt i32 %"load identifier value23", %"load identifier
68     value24"
69     br i1 %cmptmp25, label %for_end12, label %for_handle10
70     br label %for_end12
71
71     for_end12:                                ; preds =
72     %for_condition11, %for_condition11, %for_start9
73         ret i32 0
74     }
75
75     declare i32 @printf(i8*, ...)
76

```

- 汇编指令

```

1      .text
2      .file  "pascal_module"
3      .globl main                      # -- Begin function main
4      .p2align 4, 0x90
5      .type   main,@function
6      main:                           # @main
7          .cfi_startproc
8      # %bb.0:                      # %entry
9          pushl  %esi
10         .cfi_def_cfa_offset 8
11         subl   $8, %esp

```

```

12     .cfi_def_cfa_offset 16
13     .cfi_offset %esi, -8
14     movl    .L1, %eax
15     movl    %eax, iv
16     cmpl    .Lr, %eax
17     jg    .LBB0_3
18 # %bb.1:                                # %for_handle.preheader
19     movl    .Lr, %ecx
20     .p2align   4, 0x90
21 .LBB0_2:                                # %for_handle
22                                         # =>This Inner Loop Header: Depth=1
23     movl    iv, %edx
24     movl    %edx, a-4(%edx,4)
25     movl    iv, %edx
26     incl    %edx
27     movl    %edx, iv
28     cmpl    %ecx, %edx
29     jle   .LBB0_2
30 .LBB0_3:                                # %for_end
31     movl    %eax, iv
32     cmpl    .Lr, %eax
33     jg    .LBB0_6
34 # %bb.4:                                # %for_handle10.preheader
35     movl    .Lr, %esi
36     .p2align   4, 0x90
37 .LBB0_5:                                # %for_handle10
38                                         # =>This Inner Loop Header: Depth=1
39     movl    iv, %eax
40     subl    $8, %esp
41     .cfi_adjust_cfa_offset 8
42     pushl   a-4(%eax,4)
43     .cfi_adjust_cfa_offset 4
44     pushl   $.Lprintf_format
45     .cfi_adjust_cfa_offset 4
46     calll   printf@PLT
47     addl    $16, %esp
48     .cfi_adjust_cfa_offset -16
49     movl    iv, %eax
50     incl    %eax
51     movl    %eax, iv
52     cmpl    %esi, %eax
53     jle   .LBB0_5
54 .LBB0_6:                                # %for_end12
55     xorl    %eax, %eax
56     addl    $8, %esp
57     .cfi_def_cfa_offset 8
58     popl    %esi
59     .cfi_def_cfa_offset 4
60     retl
61 .Lfunc_end0:
62     .size   main, .Lfunc_end0-main
63     .cfi_endproc
64                                         # -- End function
65     .type   .L1,@object                 # @1
66     .section  .rodata,"a",@progbits
67     .p2align   2
68 .L1:
69     .long   1                           # 0x1

```

```

70     .size    .L1, 4
71
72     .type    .Lr,@object          # @r
73     .p2align   2
74 .Lr:
75     .long    6                  # 0x6
76     .size    .Lr, 4
77
78     .type    iv,@object         # @iv
79     .comm    iv,4,4
80     .type    a,@object          # @a
81     .comm    a,24,16
82     .type    .Lprintf_format,@object      # @printf_format
83     .section  .rodata.str1.1,"aMS",@progbits,1
84 .Lprintf_format:
85     .asciz   "%d\n"
86     .size    .Lprintf_format, 4
87
88     .section  ".note.GNU-stack","",@progbits
89

```

- 运行结果

```

lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli ArrayTest.pas.ll
1
2
3
4
5
6

```

7.1.3 记录类型测试

- 测试代码

```

1 program typerecord;
2 type
3     InfoType = record
4         MyName : String;
5         Age : integer;
6         City, State : integer;
7     end;
8 var
9     Book1, Book2: InfoType;
10 begin
11     Book1.MyName := 'AI';
12     Book1.Age := 2;
13     Book1.City := 4;
14     Book1.State := 5;
15     writeln(Book1.MyName, ' ', Book1.Age, ' ', Book1.City, ' ', Book1.State);
16     Book2.MyName := 'CS';
17     Book2.Age := 3;
18     Book2.City := 41;
19     Book2.State := 52;
20     writeln(Book2.MyName, ' ', Book2.Age, ' ', Book2.City, ' ', Book2.State);
21 end.

```

- IR代码

```

20    %"load record type3" = load i32, i32* getelementptr inbounds ({ [256 x
21        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 3),
22        align 4
23        store i32 5, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8],
24        i32, i32, i32 }* @Book1, i32 0, i32 3), align 4
25        %"load record type4" = load [256 x i8], [256 x i8]* getelementptr inbounds
26        ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 0),
27        align 1
28        %"load record type5" = load i32, i32* getelementptr inbounds ({ [256 x
29        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 1),
30        align 4
31        %"load record type6" = load i32, i32* getelementptr inbounds ({ [256 x
32        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 2),
33        align 4
34        %"load record type7" = load i32, i32* getelementptr inbounds ({ [256 x
35        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 3),
36        align 4
37        call printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16
38        x i8], [16 x i8]* @printf_format, i32 0, i32 0), [256 x i8]* getelementptr
39        inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }*
        @Book1, i32 0, i32 0), i8 32, i32 %"load record type5", i8 32, i32 %"load
        record type6", i8 32, i32 %"load record type7")
40        %"load record type8" = load [256 x i8], [256 x i8]* getelementptr inbounds
41        ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32
42        0, i32 0), align 1
43        %"load const string9" = load [256 x i8], [256 x i8]* @1, align 1
44        store [256 x i8] %"load const string9", [256 x i8]* getelementptr inbounds
45        ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32
46        0, i32 0), align 1
47        %"load record type10" = load i32, i32* getelementptr inbounds ({ [256 x
48        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1),
49        align 4
50        store i32 3, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
51        { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1), align 4
52        %"load record type11" = load i32, i32* getelementptr inbounds ({ [256 x
53        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2),
54        align 4
55        store i32 41, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
56        { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2), align 4
57        %"load record type12" = load i32, i32* getelementptr inbounds ({ [256 x
58        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3),
59        align 4
60        store i32 52, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
61        { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3), align 4
62        %"load record type13" = load [256 x i8], [256 x i8]* getelementptr
63        inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }*
64        @Book2, i32 0, i32 0), align 1
65        %"load record type14" = load i32, i32* getelementptr inbounds ({ [256 x
66        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1),
67        align 4
68        %"load record type15" = load i32, i32* getelementptr inbounds ({ [256 x
69        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2),
70        align 4
71        %"load record type16" = load i32, i32* getelementptr inbounds ({ [256 x
72        i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3),
73        align 4

```

```

40    %call_printf17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
41        ([16 x i8], [16 x i8]* @printf_format.1, i32 0, i32 0), [256 x i8]*
42        getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32,
43        i32, i32 }* @Book2, i32 0, i32 0), i8 32, i32 %"load record type14", i8 32,
44        i32 %"load record type15", i8 32, i32 %"load record type16")
45    ret i32 0
}
declare i32 @printf(i8*, ...)
```

- 测试结果

```
lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli typerecord.pas.ll
AI 2 4 5
CS 3 41 52
```

7.2 运算测试

- 测试代码

```

1 program ComputeTest;
2 const
3     ic = 1;
4     rc = 1.5;
5     jc = 8;
6     kc = 3;
7 var
8     add_rc : real;
9     sub_rc : real;
10    mul_rc : real;
11 begin
12     writeln(ic + ic);
13     add_rc := rc + rc;
14     writeln(add_rc);
15     writeln(ic - ic);
16     sub_rc := rc - rc;
17     writeln(sub_rc);
18     mul_rc := rc*rc;
19     writeln(mul_rc);
20     writeln(ic / (ic + 1));
21     writeln(jc mod kc);
22 end
23 .
```

- IR

```

1 ; ModuleID = 'pascal_module'
2 source_filename = "pascal_module"
3
4 @ic = private constant i32 1
5 @rc = private constant double 1.500000e+00
6 @jc = private constant i32 8
7 @kc = private constant i32 3
8 @add_rc = common global double 0.000000e+00
```

```

9 @sub_rc = common global double 0.000000e+00
10 @mul_rc = common global double 0.000000e+00
11 @printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
12 @printf_format.1 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
13 align 1
14 @printf_format.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
1
15 @printf_format.3 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
16 align 1
17 @printf_format.4 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
18 align 1
19 @printf_format.5 = private unnamed_addr constant [5 x i8] c"%lf\0A\00",
20 align 1
21 @printf_format.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
22 1
23
24 define i32 @main() {
25 entry:
26   %"load identifier value" = load i32, i32* @ic, align 4
27   %"load identifier value1" = load i32, i32* @ic, align 4
28   %addtmp = add i32 %"load identifier value", %"load identifier value1"
29   %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
30 x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %addtmp)
31   %"load identifier value2" = load double, double* @rc, align 8
32   %"load identifier value3" = load double, double* @rc, align 8
33   %addftmp = fadd double %"load identifier value2", %"load identifier
34   value3"
35   store double %addftmp, double* @add_rc, align 8
36   %"load identifier value4" = load double, double* @add_rc, align 8
37   %call_printf5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5
38 x i8], [5 x i8]* @printf_format.1, i32 0, i32 0), double %"load identifier
39   value4")
40   %"load identifier value6" = load i32, i32* @ic, align 4
41   %"load identifier value7" = load i32, i32* @ic, align 4
42   %subtmp = sub i32 %"load identifier value6", %"load identifier value7"
43   %call_printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
44 x i8], [4 x i8]* @printf_format.2, i32 0, i32 0), i32 %subtmp)
45   %"load identifier value9" = load double, double* @rc, align 8
46   %"load identifier value10" = load double, double* @rc, align 8
47   %subftmp = fsub double %"load identifier value9", %"load identifier
48   value10"
49   store double %subftmp, double* @sub_rc, align 8
50   %"load identifier value11" = load double, double* @sub_rc, align 8
51   %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
52   ([5 x i8], [5 x i8]* @printf_format.3, i32 0, i32 0), double %"load
53   identifier value11")
54   %"load identifier value13" = load double, double* @rc, align 8
55   %"load identifier value14" = load double, double* @rc, align 8
56   %mulftmp = fmul double %"load identifier value13", %"load identifier
57   value14"
58   store double %mulftmp, double* @mul_rc, align 8
59   %"load identifier value15" = load double, double* @mul_rc, align 8
60   %call_printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
61   ([5 x i8], [5 x i8]* @printf_format.4, i32 0, i32 0), double %"load
62   identifier value15")
63   %"load identifier value17" = load i32, i32* @ic, align 4
64   %"load identifier value18" = load i32, i32* @ic, align 4
65   %addtmp19 = add i32 %"load identifier value18", 1

```

```

50  %0 = uitofp i32 %"load identifier value17" to double
51  %1 = uitofp i32 %addtmp19 to double
52  %divftmp = fdiv double %0, %1
53  %call_printf20 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([5 x i8], [5 x i8]* @printf_format.5, i32 0, i32 0), double %divftmp)
54  %"load identifier value21" = load i32, i32* @jc, align 4
55  %"load identifier value22" = load i32, i32* @kc, align 4
56  %modtmp = srem i32 %"load identifier value21", %"load identifier value22"
57  %call_printf23 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @printf_format.6, i32 0, i32 0), i32 %modtmp)
58  ret i32 0
59 }
60
61 declare i32 @printf(i8*, ...)
62

```

- 汇编指令

```

1   .text
2   .file  "pascal_module"
3   .globl main                                # -- Begin function main
4   .p2align 4, 0x90
5   .type  main,@function
6   main:                                     # @main
7       .cfi_startproc
8   # %bb.0:                                    # %entry
9       pushl  %esi
10      .cfi_def_cfa_offset 8
11      subl    $40, %esp
12      .cfi_def_cfa_offset 48
13      .cfi_offset %esi, -8
14      movl    .Lic, %esi
15      leal    (%esi,%esi), %eax
16      movl    %eax, 4(%esp)
17      movl    $.Lprintf_format, (%esp)
18      calll  printf@PLT
19      fldl    .Lrc
20      fstl    32(%esp)                      # 8-byte Folded Spill
21      fadd    %st, %st(0)
22      fstl    add_rc
23      fstpl   4(%esp)
24      movl    $.Lprintf_format.1, (%esp)
25      calll  printf@PLT
26      movl    $0, 4(%esp)
27      movl    $.Lprintf_format.2, (%esp)
28      calll  printf@PLT
29      fldl    32(%esp)                      # 8-byte Folded Reload
30      fsub    %st, %st(0)
31      fstl    sub_rc
32      fstpl   4(%esp)
33      movl    $.Lprintf_format.3, (%esp)
34      calll  printf@PLT
35      fldl    32(%esp)                      # 8-byte Folded Reload
36      fmul    %st, %st(0)
37      fstl    mul_rc
38      fstpl   4(%esp)
39      movl    $.Lprintf_format.4, (%esp)

```

```

40    calll  printf@PLT
41    movl  %esi, 16(%esp)
42    incl  %esi
43    movl  $0, 20(%esp)
44    fildll 16(%esp)
45    movl  %esi, 24(%esp)
46    movl  $0, 28(%esp)
47    fildll 24(%esp)
48    fdivrp %st, %st(1)
49    fstpl 4(%esp)
50    movl  $.Lprintf_format.5, (%esp)
51    calll  printf@PLT
52    movl  .Ljc, %eax
53    cltd
54    idivl .Lkc
55    movl  %edx, 4(%esp)
56    movl  $.Lprintf_format.6, (%esp)
57    calll  printf@PLT
58    xorl  %eax, %eax
59    addl  $40, %esp
60    .cfi_def_cfa_offset 8
61    popl  %esi
62    .cfi_def_cfa_offset 4
63    retl
64 .Lfunc_end0:
65     .size   main, .Lfunc_end0-main
66     .cfi_endproc
67                                     # -- End function
68     .type   .Lic,@object          # @ic
69     .section .rodata,"a",@progbits
70     .p2align 2
71 .Lic:
72     .long   1                  # 0x1
73     .size   .Lic, 4
74
75     .type   .Lrc,@object          # @rc
76     .p2align 3
77 .Lrc:
78     .quad   0x3ff8000000000000 # double 1.5
79     .size   .Lrc, 8
80
81     .type   .Ljc,@object          # @jc
82     .p2align 2
83 .Ljc:
84     .long   8                  # 0x8
85     .size   .Ljc, 4
86
87     .type   .Lkc,@object          # @kc
88     .p2align 2
89 .Lkc:
90     .long   3                  # 0x3
91     .size   .Lkc, 4
92
93     .type   add_rc,@object        # @add_rc
94     .comm   add_rc,8,8
95     .type   sub_rc,@object        # @sub_rc
96     .comm   sub_rc,8,8
97     .type   mul_rc,@object        # @mul_rc

```

```

98     .comm    mul_rc,8,8
99     .type    .Lprintf_format,@object          # @printf_format
100    .section   .rodata.str1.1,"aMS",@progbits,1
101    .Lprintf_format:
102        .asciz   "%d\n"
103        .size    .Lprintf_format, 4
104
105        .type    .Lprintf_format.1,@object      # @printf_format.1
106    .Lprintf_format.1:
107        .asciz   "%lf\n"
108        .size    .Lprintf_format.1, 5
109
110        .type    .Lprintf_format.2,@object      # @printf_format.2
111    .Lprintf_format.2:
112        .asciz   "%d\n"
113        .size    .Lprintf_format.2, 4
114
115        .type    .Lprintf_format.3,@object      # @printf_format.3
116    .Lprintf_format.3:
117        .asciz   "%lf\n"
118        .size    .Lprintf_format.3, 5
119
120        .type    .Lprintf_format.4,@object      # @printf_format.4
121    .Lprintf_format.4:
122        .asciz   "%lf\n"
123        .size    .Lprintf_format.4, 5
124
125        .type    .Lprintf_format.5,@object      # @printf_format.5
126    .Lprintf_format.5:
127        .asciz   "%lf\n"
128        .size    .Lprintf_format.5, 5
129
130        .type    .Lprintf_format.6,@object      # @printf_format.6
131    .Lprintf_format.6:
132        .asciz   "%d\n"
133        .size    .Lprintf_format.6, 4
134
135    .section   ".note.GNU-stack","",@progbits
136

```

- 运行结果

```

lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli ComputeTest.pas.ll
2
3.000000
0
0.000000
2.250000
0.500000
2

```

7.3 控制流测试

7.3.1 分支测试

- 测试代码

```
1 program ifstmt;
2 Const
3     constTerm1 = 3;
4     constTerm2 = 3.14;
5 var
6     curTerm1 :integer;
7     curTerm2 :integer;
8 begin
9     curTerm1 := constTerm1;
10    curTerm1 := 1 + 2*(2 + 3);
11
12    if curTerm1 = 11 then
13        curTerm2 := 5
14    else
15        if curTerm1 = 12 then
16            curTerm2 := 6
17        else
18            curTerm2 := 7;
19
20    writeln(curTerm1);
21    writeln(curTerm2);
22 end.
```

- IR

```
1 ; ModuleID = 'pascal_module'
2 source_filename = "pascal_module"
3
4 @constTerm1 = private constant i32 3
5 @constTerm2 = private constant double 3.140000e+00
6 @curTerm1 = common global i32 0
7 @curTerm2 = common global i32 0
8 @printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
9 @printf_format.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
10 1
11 define i32 @main() {
12 entry:
13 %"load identifier value" = load i32, i32* @constTerm1, align 4
14     store i32 %"load identifier value", i32* @curTerm1, align 4
15     store i32 11, i32* @curTerm1, align 4
16     %"load identifier value1" = load i32, i32* @curTerm1, align 4
17     %cmptmp = icmp eq i32 %"load identifier value1", 11
18     br i1 %cmptmp, label %if_then, label %if_else
19
20 if_then:                                     ; preds = %entry
21     store i32 5, i32* @curTerm2, align 4
22     br label %if_continue
23
24 if_else:                                     ; preds = %entry
25     %"load identifier value5" = load i32, i32* @curTerm1, align 4
26     %cmptmp6 = icmp eq i32 %"load identifier value5", 12
27     br i1 %cmptmp6, label %if_then2, label %if_else3
```

```

28
29 if_continue:                                ; preds = %if_continue4,
%if_then
30   %"load identifier value7" = load i32, i32* @curTerm1, align 4
31   %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load identifier
value7")
32   %"load identifier value8" = load i32, i32* @curTerm2, align 4
33   %call_printf9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8], [4 x i8]* @printf_format.1, i32 0, i32 0), i32 %"load identifier
value8")
34   ret i32 0
35
36 if_then2:                                    ; preds = %if_else
37   store i32 6, i32* @curTerm2, align 4
38   br label %if_continue4
39
40 if_else3:                                    ; preds = %if_else
41   store i32 7, i32* @curTerm2, align 4
42   br label %if_continue4
43
44 if_continue4:                               ; preds = %if_else3,
%if_then2
45   br label %if_continue
46 }
47
48 declare i32 @printf(i8*, ...)
49

```

- 汇编指令

```

1 .text
2 .file  "pascal_module"
3 .globl main                                # -- Begin function main
4 .p2align 4, 0x90
5 .type  main,@function
6 main:                                     # @main
7   .cfi_startproc
8 # %bb.0:                                    # %entry
9   subl  $12, %esp
10  .cfi_def_cfa_offset 16
11  movl  $11, curTerm1
12  xorl  %eax, %eax
13  testb %al, %al
14  jne .LBB0_4
15 # %bb.1:                                   # %if_then
16  movl  $5, curTerm2
17  jmp .LBB0_2
18 .LBB0_4:                                   # %if_else
19  cmpl  $12, curTerm1
20  jne .LBB0_5
21 # %bb.3:                                   # %if_then2
22  movl  $6, curTerm2
23  jmp .LBB0_2
24 .LBB0_5:                                   # %if_else3
25  movl  $7, curTerm2
26 .LBB0_2:                                   # %if_continue

```

```

27    subl    $8, %esp
28    .cfi_adjust_cfa_offset 8
29    pushl   curTerm1
30    .cfi_adjust_cfa_offset 4
31    pushl   $.Lprintf_format
32    .cfi_adjust_cfa_offset 4
33    calll   printf@PLT
34    addl    $8, %esp
35    .cfi_adjust_cfa_offset -8
36    pushl   curTerm2
37    .cfi_adjust_cfa_offset 4
38    pushl   $.Lprintf_format.1
39    .cfi_adjust_cfa_offset 4
40    calll   printf@PLT
41    addl    $16, %esp
42    .cfi_adjust_cfa_offset -16
43    xorl    %eax, %eax
44    addl    $12, %esp
45    .cfi_def_cfa_offset 4
46    retl
47 .Lfunc_end0:
48     .size   main, .Lfunc_end0-main
49     .cfi_endproc
50                                     # -- End function
51     .type   .LconstTerm1,@object          # @constTerm1
52     .section .rodata, "a",@progbits
53     .p2align 2
54 .LconstTerm1:
55     .long   3                           # 0x3
56     .size   .LconstTerm1, 4
57
58     .type   .LconstTerm2,@object          # @constTerm2
59     .p2align 3
60 .LconstTerm2:
61     .quad   0x40091eb851eb851f        # double 3.1400000000000001
62     .size   .LconstTerm2, 8
63
64     .type   curTerm1,@object           # @curTerm1
65     .comm   curTerm1,4,4
66     .type   curTerm2,@object           # @curTerm2
67     .comm   curTerm2,4,4
68     .type   .Lprintf_format,@object      # @printf_format
69     .section .rodata.str1.1,"aMS",@progbits,1
70 .Lprintf_format:
71     .asciz  "%d\n"
72     .size   .Lprintf_format, 4
73
74     .type   .Lprintf_format.1,@object      # @printf_format.1
75 .Lprintf_format.1:
76     .asciz  "%d\n"
77     .size   .Lprintf_format.1, 4
78
79     .section ".note.GNU-stack","",@progbits
80

```

- 运行结果

```
lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli ifstmt.pas.ll
```

11

5

7.3.2 循环测试

- 测试代码

```
1 program loopstmt;
2 var
3     array_a : array[0 .. 625] of integer;
4     i : Integer;
5     sum : Integer;
6 begin
7     writeln('test for for');
8     for i:=2 to 10 do array_a[i]:=i;
9     for i:=2 to 10 do writeln(array_a[i]);
10    writeln('test for while: 1+...10');
11    i:=0;
12    while i<10 do begin
13        sum:=sum+i;
14        i:=i+1;
15    end;
16    writeln(sum);
17 end.
```

- |R


```

51 %subtmp9 = sub i32 %"load identifier value8", 0
52 %ArrayCall10 = getelementptr [626 x i32], [626 x i32]* @array_a, i32 0,
53 i32 %subtmp9
54 %"load array value11" = load i32, i32* %ArrayCall10, align 4
55 %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
56 ([4 x i8], [4 x i8]* @printf_format.1, i32 0, i32 0), i32 %"load array
57 value11")
58 br label %for_condition6
59
60
61 for_condition6: ; preds = %for_handle5
62 %"load identifier value13" = load i32, i32* @i, align 4
63 %tmpadd14 = add i32 %"load identifier value13", 1
64 store i32 %tmpadd14, i32* @i, align 4
65 %"load identifier value15" = load i32, i32* @i, align 4
66 %cmptmp16 = icmp sgt i32 %"load identifier value15", 10
67 br i1 %cmptmp16, label %for_end7, label %for_handle5
68 br label %for_end7
69
70 for_end7: ; preds = %for_condition6,
71 %for_condition6, %for_start4
72 %"load const string17" = load [256 x i8], [256 x i8]* @1, align 1
73 %call_printf18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
74 ([4 x i8], [4 x i8]* @printf_format.2, i32 0, i32 0), [256 x i8]* @1)
75 store i32 0, i32* @i, align 4
76 br label %while_condition
77
78 while_condition: ; preds = %while_body,
79 %for_end7
80 %"load identifier value19" = load i32, i32* @i, align 4
81 %cmptmp20 = icmp slt i32 %"load identifier value19", 10
82 br i1 %cmptmp20, label %while_body, label %while_end
83
84 while_body: ; preds = %while_condition
85 %"load identifier value21" = load i32, i32* @sum, align 4
86 %"load identifier value22" = load i32, i32* @i, align 4
87 %addtmp = add i32 %"load identifier value21", %"load identifier value22"
88 store i32 %addtmp, i32* @sum, align 4
89 %"load identifier value23" = load i32, i32* @i, align 4
90 %addtmp24 = add i32 %"load identifier value23", 1
91 store i32 %addtmp24, i32* @i, align 4
92 br label %while_condition
93
94 while_end: ; preds = %while_condition
95 %"load identifier value25" = load i32, i32* @sum, align 4
96 %call_printf26 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
97 ([4 x i8], [4 x i8]* @printf_format.3, i32 0, i32 0), i32 %"load identifier
98 value25")
99 ret i32 0
100 }
```

- 汇编指令

```

1 .text
2 .file "pascal_module"
```

```

3     .globl  main                                # -- Begin function main
4     .p2align   4, 0x90
5     .type   main,@function
6 main:                                     # @main
7     .cfi_startproc
8 # %bb.0:                                     # %entry
9     subl   $20, %esp
10    .cfi_adjust_cfa_offset 20
11    pushl   $.L__unnamed_1
12    .cfi_adjust_cfa_offset 4
13    pushl   $.Lprintf_format
14    .cfi_adjust_cfa_offset 4
15    calll   printf@PLT
16    addl   $16, %esp
17    .cfi_adjust_cfa_offset -16
18    movl   $2, i
19    .p2align   4, 0x90
20 .LBB0_1:                                     # %for_handle
21                                         # =>This Inner Loop Header: Depth=1
22    movl   i, %eax
23    movl   %eax, array_a(%eax,4)
24    movl   i, %eax
25    incl   %eax
26    movl   %eax, i
27    cmpl   $11, %eax
28    j1  .LBB0_1
29 # %bb.2:                                     # %for_end
30    movl   $2, i
31    .p2align   4, 0x90
32 .LBB0_3:                                     # %for_handle5
33                                         # =>This Inner Loop Header: Depth=1
34    movl   i, %eax
35    subl   $8, %esp
36    .cfi_adjust_cfa_offset 8
37    pushl   array_a(%eax,4)
38    .cfi_adjust_cfa_offset 4
39    pushl   $.Lprintf_format.1
40    .cfi_adjust_cfa_offset 4
41    calll   printf@PLT
42    addl   $16, %esp
43    .cfi_adjust_cfa_offset -16
44    movl   i, %eax
45    incl   %eax
46    movl   %eax, i
47    cmpl   $11, %eax
48    j1  .LBB0_3
49 # %bb.4:                                     # %for_end7
50    subl   $8, %esp
51    .cfi_adjust_cfa_offset 8
52    pushl   $.L__unnamed_2
53    .cfi_adjust_cfa_offset 4
54    pushl   $.Lprintf_format.2
55    .cfi_adjust_cfa_offset 4
56    calll   printf@PLT
57    addl   $16, %esp
58    .cfi_adjust_cfa_offset -16
59    movl   $0, i
60    cmpl   $9, i

```


- 运行结果

```
lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli whilstmt.pas.ll
test for for
2
3
4
5
6
7
8
9
10
test for while: 1+...10
45
```

7.4 函数测试

7.4.1 简单函数测试

- 测试代码

```
1 program declfunc;
2 const
3     CurTerm1 = 3;
4 var
5     CurTerm2 : integer;
6 function myLine(a : integer): integer;
7 begin
8     myLine := a + a;
9 end;
10 begin
11     CurTerm2 := myLine(CurTerm1);
12     writeln(CurTerm2);
13 end.
```

- IR

```
1 ; ModuleID = 'pascal_module'
2 source_filename = "pascal_module"
3
4 @CurTerm1 = private constant i32 3
5 @CurTerm2 = common global i32 0
6 @printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
7
8 define i32 @main() {
9 entry:
10     %"load identifier value" = load i32, i32* @CurTerm1, align 4
11     %0 = call i32 @myLine(i32* @CurTerm1)
12     store i32 %0, i32* @CurTerm2, align 4
13     %"load identifier value1" = load i32, i32* @CurTerm2, align 4
14     %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
15 x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load identifier
16     value1")
17     ret i32 0
18 }
19
20 define i32 @myLine(i32* %0) {
21 entry:
22     %"load var param" = load i32, i32* %0, align 4
23     %a = alloca i32, align 4
24     store i32 %"load var param", i32* %a, align 4
25     %myLine = alloca i32, align 4
26     %"load identifier value" = load i32, i32* %a, align 4
27     %"load identifier value1" = load i32, i32* %a, align 4
28     %addtmp = add i32 %"load identifier value", %"load identifier value1"
29     store i32 %addtmp, i32* %myLine, align 4
30     %"load ret param" = load i32, i32* %myLine, align 4
31     ret i32 %"load ret param"
32 }
33 declare i32 @printf(i8*, ...)
```

• 汇编指令

```

1      .text
2      .file  "pascal_module"
3      .globl main                                # -- Begin function main
4      .p2align 4, 0x90
5      .type  main,@function
6  main:                                     # @main
7      .cfi_startproc
8  # %bb.0:                                    # %entry
9      subl    $12, %esp
10     .cfi_def_cfa_offset 16
11     movl    $.LCurTerm1, (%esp)
12     calll   myLine@PLT
13     movl    %eax, CurTerm2
14     movl    %eax, 4(%esp)
15     movl    $.Lprintf_format, (%esp)
16     calll   printf@PLT
17     xorl    %eax, %eax
18     addl    $12, %esp
19     .cfi_def_cfa_offset 4
20     retl
21 .Lfunc_end0:
22     .size   main, .Lfunc_end0-main
23     .cfi_endproc                                # -- End function
24
25     .globl  myLine                            # -- Begin function myLine
26     .p2align 4, 0x90
27     .type   myLine,@function
28 myLine:                                     # @myLine
29     .cfi_startproc
30 # %bb.0:                                    # %entry
31     subl    $8, %esp
32     .cfi_def_cfa_offset 12
33     movl    12(%esp), %eax
34     movl    (%eax), %eax
35     movl    %eax, 4(%esp)
36     addl    %eax, %eax
37     movl    %eax, (%esp)
38     addl    $8, %esp
39     .cfi_def_cfa_offset 4
40     retl
41 .Lfunc_end1:
42     .size   myLine, .Lfunc_end1-myLine
43     .cfi_endproc                                # -- End function
44
45     .type   .LCurTerm1,@object                # @CurTerm1
46     .section .rodata,"a",@progbits
47     .p2align 2
48 .LCurTerm1:
49     .long   3                                  # 0x3
50     .size   .LCurTerm1, 4
51
52     .type   CurTerm2,@object                  # @CurTerm2
53     .comm   CurTerm2,4,4

```

```

54     .type    .Lprintf_format,@object          # @printf_format
55     .section   .rodata.str1.1,"ams",@progbits,1
56 .Lprintf_format:
57     .asciz   "%d\n"
58     .size    .Lprintf_format, 4
59
60     .section   ".note.GNU-stack","",@progbits
61

```

- 运行结果

```

lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli declfunc.pas.ll
6

```

7.5 快速排序测试

- 测试代码

```

1
2 Program quicksort;
3
4 Var
5     a : array[1 .. 10001] of integer;
6     s : integer;
7     num : Integer;
8     flag : Integer;
9     s1 : Integer;
10
11 Procedure qsort(first,last:integer);
12
13 Var
14     temp : Integer;
15     mid : Integer;
16     pivot : Integer;
17     i : Integer;
18     j : Integer;
19 Begin
20     i := first;
21     j := last;
22     pivot := (first+last) div 2;
23     //select first element be the pivot
24     mid := a[pivot];
25     while i<=j do
26         Begin
27             While a[i] < mid Do begin i := i+1; end;
28             // find the first one bigger than a[pivot]
29             While a[j] > mid Do begin j := j-1; end;
30             // find the last one smaller than a[pivot]
31             If i<=j Then // swap a[i],a[j]
32                 Begin
33                     temp := a[i];
34                     a[i] := a[j];
35                     a[j] := temp;
36                     i := i+1;
37                     j := j-1;
38                 End;
39             End;

```

```

40     If first < j Then qsort(first,j);
41     If i < last Then qsort(i,last);
42 End;
43
44 Begin
45     Read(num);
46     // num := num -1;
47     For s:=1 To num Do
48         Read(a[s]);
49     qsort(1,num);
50     For s:=1 To num Do
51         writeln(a[s]);
52 End.
53

```

- IR (因代码过长, 不再贴出, 改为放在测试文件中上传)
- 运行结果

```

lz@ubuntu:~/Documents/Our_Pascal_Compiler/test/quicksort$ ./linux-amd64 ./qsort
fixed case 0 (size 0)...pass!
fixed case 1 (size 1)...pass!
fixed case 2 (size 2)...pass!
fixed case 3 (size 2)...pass!
fixed case 4 (size 3)...pass!
fixed case 5 (size 3)...pass!
fixed case 6 (size 3)...pass!
fixed case 7 (size 3)...pass!
fixed case 8 (size 3)...pass!
fixed case 9 (size 4)...pass!
fixed case 10 (size 9)...pass!
fixed case 11 (size 9)...pass!
fixed case 12 (size 10000)...pass!
fixed case 13 (size 10000)...pass!
fixed case 14 (size 4096)...pass!
randomly generated case 0 (size 10000)...pass!
randomly generated case 1 (size 10000)...pass!
randomly generated case 2 (size 10000)...pass!
randomly generated case 3 (size 10000)...pass!
randomly generated case 4 (size 10000)...pass!
randomly generated case 5 (size 10000)...pass!
randomly generated case 6 (size 10000)...pass!
randomly generated case 7 (size 10000)...pass!
randomly generated case 8 (size 10000)...pass!
randomly generated case 9 (size 10000)...pass!
-----
2022-05-21 02:45:57.178

```

7.6 矩阵乘法测试

- 测试代码

```

1 program arraymul;
2 var
3     array_a : array[0 .. 625] of integer;
4     array_b : array[0 .. 625] of integer;
5     array_c : array[0 .. 625] of integer;
6     m_a : integer;
7     n_a : integer;
8     s_a : integer;
9     m_b : integer;
10    n_b : integer;
11    s_b : integer;
12    s_c : integer;
13    i : integer;
14    j : integer;

```

```

15     k : integer;
16     tmp_a : Integer;
17     tmp_b : Integer;
18     tmp_c : Integer;
19 begin
20     //读入矩阵A
21     ReadLn(m_a,n_a);
22     // writeln('m=',m_a,',n=',n_a);
23     s_a:=m_a*n_a;
24     // writeln('m * n = ',s_a);
25
26     for i:=0 to s_a-1 do Read(array_a[i]);
27     //读入矩阵B
28     ReadLn(m_b,n_b);
29     // writeln('m=',m_b,',n=',n_b);
30     s_b:=m_b*n_b;
31     // writeln('m * n = ',s_b);
32
33     for i:=0 to s_b-1 do Read(array_b[i]);
34
35     if n_a <> m_b then
36         writeln('Incompatible Dimensions')
37     else begin
38         //进行矩阵乘法
39         for i:=0 to m_a-1 do
40             begin
41                 for j:=0 to n_b-1 do
42                     begin
43                         for k:=0 to m_b-1 do
44                             begin
45                                 tmp_c := i*n_b+j;
46                                 tmp_b := k*n_b+j;
47                                 tmp_a := i*m_b+k;
48                                 array_c[tmp_c] := array_c[tmp_c] + array_b[tmp_b] *
array_a[tmp_a];
49                                 end;
50                         end;
51                     end;
52
53                     //测试输出
54                     s_c := m_a*n_b;
55                     for i:=0 to m_a-1 do
56                         begin
57                             for j:=0 to n_b-1 do
58                                 begin
59                                     k:= i*n_b+j;
60                                     printf(array_c[k]);
61                                 end;
62                                 writeln;
63                             end;
64                         end;
65                     end.

```

- IR (因代码过长, 不再贴出, 改为放在测试文件中上传)
- 运行结果

```
lz@ubuntu:~/Documents/Our_Pascal_Compiler/test/matrix-multiplication$ ./linux-amd64 ./matrix
fixed case 0 (size [1x1]x[1x1])...pass!
fixed case 1 (size [1x1]x[2x1])...pass!
fixed case 2 (size [1x4]x[4x1])...pass!
fixed case 3 (size [4x1]x[1x4])...pass!
fixed case 4 (size [1x25]x[25x1])...pass!
randomly generated case 0 (size [20x20]x[20x20])...pass!
randomly generated case 1 (size [20x20]x[20x20])...pass!
randomly generated case 2 (size [20x20]x[20x20])...pass!
randomly generated case 3 (size [20x20]x[20x20])...pass!
randomly generated case 4 (size [20x20]x[20x20])...pass!
randomly generated case 5 (size [20x20]x[20x20])...pass!
randomly generated case 6 (size [20x20]x[20x20])...pass!
randomly generated case 7 (size [20x20]x[20x20])...pass!
randomly generated case 8 (size [20x20]x[20x20])...pass!
randomly generated case 9 (size [20x20]x[20x20])...pass!
-----
2022-05-21 02:55:56.298
```

7.7 选课系统测试

- 测试代码

```
1 Program advisor;
2
3 Var
4   courses : array[0 .. 39999] of char;
5   map: array[0..99] of integer;
6   is_passed: array[0..99] of integer;
7   course_name: array[0..4] of char;
8
9   // main variable
10  ch: char;
11  num: integer;
12  index: integer;
13  total_credits: integer;
14  attempted_credits: integer;
15  passed_credits: integer;
16  remained_credits: integer;
17  total_score: integer;
18  field: integer;
19  credit: integer;
20  key: integer;
21  i: integer;
22  GPA: real;
23  value: integer;
24  flag: integer;
25  j: integer;
26  k: integer;
27  tmp_idx: integer;
28
29  myeof: char;
30  c_tmp1 : integer;
31  c_tmp2 : integer;
32  tmp_value : char;
33  tmp_int : Integer;
34  zero : char;
35  split : char;
36  A : char;
37  B : char;
38  C : char;
39  D : char;
40  E : char;
```

```
41   F : char;
42
43   tmp_string : string;
44
45   intzero: Integer;
46   intnine : Integer;
47
48
49
50 function char2int(a: char):integer;
51 begin
52   if a = '0' then begin
53     char2int := 0;
54   end else if a = '1' then begin
55     char2int := 1;
56   end else if a = '2' then begin
57     char2int := 2;
58   end else if a = '3' then begin
59     char2int := 3;
60   end else if a = '4' then begin
61     char2int := 4;
62   end else if a = '5' then begin
63     char2int := 5;
64   end else if a = '6' then begin
65     char2int := 6;
66   end else if a = '7' then begin
67     char2int := 7;
68   end else if a = '8' then begin
69     char2int := 8;
70   end else if a = '9' then begin
71     char2int := 9;
72   end else begin
73     char2int := -1;
74   end;
75 end;
76
77 Begin
78   myeof := '~';
79   zero := '0';
80   split := '|';
81   A := 'A';
82   B := 'B';
83   C := 'C';
84   D := 'D';
85   E := 'E';
86   F := 'F';
87   intzero := 0;
88   intnine := 9;
89
90
91 // initial
92
93   num := 0;
94   index := 0;
95   total_credits := 0;
96   passed_credits := 0;
97   total_score := 0;
98   field := 0;
```

```

99    credit := 0;
100   key := 0;
101   i := 0;
102   GPA := 0.0;
103   value := 0;
104   flag := 0;
105   j := 0;
106   k := 0;
107
108  while True do begin
109    Read(ch);
110    if ch = '\n' then
111      break;
112    index := 0;
113    tmp_idx := num * 400 + index;
114    courses[tmp_idx] := ch;
115    index := index + 1;
116
117    while True do begin
118      Read(ch);
119      if ch = '\n' then
120        break;
121      tmp_idx := num * 400 + index;
122      courses[tmp_idx] := ch;
123      index := index + 1;
124    end;
125
126    tmp_idx := num * 400 + index;
127    courses[tmp_idx] := myeof;
128
129    is_passed[num] := 0;
130    num := num + 1;
131  end;
132
133 for i:=0 to (num-1) do begin
134   index := 0;
135   field := 0;
136   credit := 0;
137   key := 0;
138
139   tmp_idx := i * 400 + index;
140
141   tmp_value := courses[tmp_idx] ;
142
143   while (tmp_value <> myeof) do begin
144
145     tmp_idx := i * 400 + index;
146     tmp_value := courses[tmp_idx] ;
147
148     if (tmp_value = split) then begin // not such class
149       field := field + 1;
150       index := index + 1;
151     end;
152
153     if (field = 0) then begin
154       tmp_idx := i * 400 + index;
155       tmp_int := courses[tmp_idx];
156       tmp_int := char2int(tmp_value);

```

```

157
158         if (( tmp_int <= intnine) and (tmp_int >= intzero)) then
159             begin
160                 tmp_value := courses[tmp_idx];
161                 c_tmp1 := char2int(tmp_value);
162                 tmp_value := zero;
163                 c_tmp2 := char2int(tmp_value);
164                 key := key * 10 + c_tmp1 - c_tmp2;
165                 // write('key = ');
166                 // write(key);
167                 // write('\n');
168             end;
169             index := index + 1;
170         end else if (field = 1) then begin
171             tmp_idx := i * 400 + index;
172             tmp_value := courses[tmp_idx];
173             c_tmp1 := char2int(tmp_value);
174             tmp_value := zero;
175             c_tmp2 := char2int(tmp_value);

176             credit := c_tmp1 - c_tmp2;
177             total_credits := total_credits + credit;

178             map[key] := i;
179             index := index + 1;
180         end else if (field = 2) then begin
181             tmp_idx := i * 400 + index;
182             tmp_value := courses[tmp_idx];
183             if (courses[tmp_idx] <> '|') then begin
184                 index := index + 1;
185             end;
186         end else if (field = 3) then begin
187             tmp_idx := i * 400 + index;
188             tmp_value := courses[tmp_idx];

189             if (tmp_value <> myeof) then begin
190                 if (tmp_value = 'A') then begin
191                     attempted_credits := attempted_credits + credit;
192                     passed_credits := passed_credits + credit;
193                     total_score := total_score + 4 * credit;
194                     is_passed[i] := 1;
195                 end;
196                 if (tmp_value = 'B') then begin
197                     attempted_credits := attempted_credits + credit;
198                     passed_credits := passed_credits + credit;
199                     total_score := total_score + 3 * credit;
200                     is_passed[i] := 1;
201                 end;
202                 if (tmp_value = 'C') then begin
203                     attempted_credits := attempted_credits + credit;
204                     passed_credits := passed_credits + credit;
205                     total_score := total_score + 2 * credit;
206                     is_passed[i] := 1;
207                 end;
208                 if (tmp_value = 'D') then begin
209                     attempted_credits := attempted_credits + credit;
210                     passed_credits := passed_credits + credit;
211                     total_score := total_score + credit;
212                 end;
213             end;

```

```

214             is_passed[i] := 1;
215         end;
216         if (tmp_value = 'F') then begin
217             attempted_credits := attempted_credits + credit;
218         end;
219         index := index + 1;//cannot out of this!!!
220     end;
221     tmp_idx := i * 400 + index;
222     tmp_value := courses[tmp_idx] ;
223 end;
224 tmp_idx := i * 400 + index;
225 tmp_value := courses[tmp_idx] ;
226 end;
227 tmp_idx := i * 400 + index;
228 tmp_value := courses[tmp_idx] ;
229 end;
230
231 if (attempted_credits <> 0) then begin
232     GPA := 1.0 * total_score / attempted_credits;
233 end;
234 remained_credits := total_credits - passed_credits;
235 write('GPA: ');
236 write(GPA);
237 write('\n');
238
239 write('Hours Attempted: ');
240 write(attempted_credits);
241 write('\n');
242
243 write('Hours Completed: ');
244 write(passed_credits);
245 write('\n');
246
247 write('Credits Remaining: ');
248 write(remained_credits);
249 write('\n');
250
251 write('\n');
252 write('Possible courses to Take Next');
253 write('\n');
254
255 if (remained_credits = 0) then begin
256     writeln(' None - Congratulations!');
257 end
258 else begin // find the class need to take
259     value := 0;
260     flag := 1;
261     j := 0;
262
263     for i:=0 to (num-1) do begin
264
265         if (is_passed[i] = 0) then begin // not even passed, refer
266             index := 0;
267             j := 0;
268             tmp_idx := i * 400 + index;
269             while (courses[tmp_idx] <> split) do begin // left courses
name
270                 course_name[j] := courses[tmp_idx];

```

```

271
272         j := j + 1;
273         index := index + 1;
274         tmp_idx := i * 400 + index;
275
276     end;
277
278     course_name[j] := myeof; // store courses name
279
280     index := index + 1;
281     tmp_idx := i * 400 + index;
282     while (courses[tmp_idx] <> split) do begin
283         index := index + 1;
284         tmp_idx := i * 400 + index;
285     end;
286     index := index + 1; // skip '|'
287
288     tmp_idx := i * 400 + index;
289
290     if (courses[tmp_idx] = split)then begin // output left
291         courses name, because there is no other dependent classes
292             write(' ');
293             for k:=0 to (j-1) do begin
294                 write(course_name[k]);
295             end;
296             write('\n');
297         end
298         else begin
299
300             flag := 1;
301             key := 0;
302
303             tmp_idx := i * 400 + index;
304             while (courses[tmp_idx] <> split) do begin //there is
305                 other class name
306                     tmp_value := courses[tmp_idx];
307                     tmp_int := char2int(tmp_value);
308
309                     if ((tmp_int <= intnine) and (tmp_int >= intzero))
310             then begin
311                 tmp_value := courses[tmp_idx];
312                 c_tmp1 := char2int(tmp_value);
313                 tmp_value := zero;
314                 c_tmp2 := char2int(tmp_value);
315                 key := key * 10 + c_tmp1 - c_tmp2; // class
316
317                 name enum key
318
319             ignore this one
320
321             tmp_idx := i * 400 + index;
322             if (courses[tmp_idx] = ',') then begin
323                 value := map[key];
324                 if (is_passed[value] = 0) then begin
325                     flag := 0; //dependent class not pass,
326
327                 end;
328                 key := 0; //clear key
329             end;
330         end;
331     end;
332
333 
```

```

324         tmp_idx := i * 400 + index;
325         if (courses[tmp_idx] = ';') then begin
326             value := map[key];
327             if (is_passed[value] = 0) then begin
328                 flag := 0;
329             end;
330             key := 0;
331             if (flag = 1) then begin
332                 break;
333             end
334             else begin
335                 flag := 1;
336             end;
337         end;
338         index := index + 1;
339         tmp_idx := i * 400 + index;
340     end;
341
342     tmp_idx := i * 400 + index;
343     if (courses[tmp_idx] = split) then begin
344         value := map[key];
345         if (is_passed[value] = 0) then begin
346             flag := 0;
347         end;
348     end;
349
350     if (flag = 1) then begin
351         write(' ');
352         for k:=0 to (j-1) do begin
353             write(course_name[k]);
354         end;
355         write('\n');
356     end;
357     tmp_idx := i * 400 + index;
358 end;
359     tmp_idx := i * 400 + index;
360 end;
361     tmp_idx := i * 400 + index;
362 end;
363     tmp_idx := i * 400 + index;
364 end;
365     tmp_idx := i * 400 + index;
366
367 // write('\n');
368 End.
```

- IR (因代码过长，不再贴出，改为放在测试文件中上传)
- 运行结果

```
alicewyp@alicewyp-virtual-machine:~/cp/project/Our_Pascal_Compiler/test/auto-advisor$ ./linux-amd64 ../../build/main
fixed case 0...pass!
fixed case 1...pass!
fixed case 2...pass!
fixed case 3...pass!
randomly generated case 0...pass!
randomly generated case 1...pass!
randomly generated case 2...pass!
randomly generated case 3...pass!
randomly generated case 4...pass!
randomly generated case 5...pass!
randomly generated case 6...pass!
randomly generated case 7...pass!
randomly generated case 8...pass!
randomly generated case 9...pass!
-----
2022-05-21 18:50:16.24
```

第八章 总结

本次实验我们完成了一个基于Pascal语言的编译器，目前我们支持的功能如下：

- 数据类型
 - 内置类型int, char, bool, real, string
 - 范围类型
 - 常量范围类型
 - 变量范围类型
 - 数组类型
 - 结构体类型Record
- 表达式
 - 一元表达式: NOT/SUB/ADD
 - 二元表达式: GE/GT/LE/LT/EQUAL/UNEQUAL/PLUS/MINUS/OR/MUL/REALDIV/DIV/MOD/AND
 - 常数表达式
 - 数组表达式
 - 结构体表达式
 - 字符串表达式
 - 函数调用
- 语句
 - 赋值语句
 - 函数调用语句
 - 复合语句
 - 嵌套语句
 - 流程控制语句
 - 分支语句: if-then-else
 - 循环语句: while, repeat, for
 - 跳转语句: goto
 - 退出语句: break
- 函数或过程
 - 函数或者过程的声明、定义、调用
 - 系统IO函数: read, write, writeln, readln
- 优化: 常量折叠

我们设计的是Pascal语法的一个子集，但通过了所有测试点并且完成了附加任务结构体，是对编译原理课程所学知识的实践，同时也运用了flex, yacc, llvm等工具，认识到了他们的强大力量。实验仍有许多需要完善的地方，希望后续有时间能够深入学习。

附录

parser/pascal.y: 语法分析器

```
1 program:
2     pro_head routine SYM_PERIOD {
3         //root of the ast, a global variable;
4         ast_root = new AST_Program($1,$2);
5         SET_LOCATION(ast_root);
6     }
7 ;
8
9 pro_head:
10    KEY_PROGRAM IDENTIFIER SYM_SEMICOLON{
11        $$ = new AST_Program_Head($2);
12        SET_LOCATION($$);
13    }
14 ;
15
16 routine:
17    routine_head routine_body{
18        // function_decl and procedure_decl
19        $$ = new AST_Routine($1,$2);
20        SET_LOCATION($$);
21    }
22 ;
23
24 routine_head:
25    const_part type_part var_part routine_part{
26        $$ = new AST_Routine_Head($1,$2,$3,$4);
27        SET_LOCATION($$);
28    }
29 ;
30
31 const_part:
32    KEY_CONST const_expr_list {
33        $$ = new AST_Const_Part($2);
34        SET_LOCATION($$);
35    }
36    | {
37        $$ = nullptr;
38    }
39 ;
40
41 type_part:
42    KEY_TYPE type_decl_list {
43        $$ = new AST_Type_Part($2);
44        SET_LOCATION($$);
45    }
46    | {
47        $$ = nullptr;
48    }
49 ;
50
51 var_part:
52    KEY_VAR var_decl_list {
```

```

53     $$ = new AST_Variable_Part($2);
54     SET_LOCATION($$);
55 }
56 | {
57     $$ = nullptr;
58 }
59 ;
60
61 routine_part:
62     routine_part function_decl {
63         AST_Declaration_BaseClass* tmp1 = new
AST_Declaration_BaseClass($2);
64         ($1) -> Add_Declaration(tmp1);
65         $$ = $1;
66         SET_LOCATION($$);
67     }
68     | routine_part procedure_decl {
69         AST_Declaration_BaseClass* tmp2 = new
AST_Declaration_BaseClass($2);
70         ($1) -> Add_Declaration(tmp2);
71         $$ = $1;
72         SET_LOCATION($$);
73     }
74     | function_decl {
75         AST_Declaration_BaseClass* tmp3 = new
AST_Declaration_BaseClass($1);
76         $$ = new AST_Routine_Part(tmp3);
77         SET_LOCATION($$);
78     }
79     | procedure_decl {
80         AST_Declaration_BaseClass* tmp4 = new
AST_Declaration_BaseClass($1);
81         $$ = new AST_Routine_Part(tmp4);
82         SET_LOCATION($$);
83     }
84     | {
85         $$ = nullptr;
86     }
87 ;
88
89 routine_body:
90     compound_stmt {
91         $$ = new AST_Routine_Body($1);
92         SET_LOCATION($$);
93     }
94 ;
95
96 compound_stmt:
97     KEY_BEGIN stmt_list KEY_END{
98         $$ = new AST_Compound_Statement($2);
99         SET_LOCATION($$);
100    }
101 ;
102
103 const_expr_list:
104     const_expr_list const_expr {
105         ($1)->Add_Const_Expression($2);
106         $$ = $1;

```

```

107     SET_LOCATION($$);
108 }
109 | const_expr {
110     $$ = new AST_Const_Expression_List();
111     ($$)->Add_Const_Expression($1);
112     SET_LOCATION($$);
113 }
114 ;
115
116 const_expr:
117     IDENTIFIER SYM_EQ expression SYM_SEMICOLON {
118         $$ = new AST_Const_Expression($1,$3);
119         SET_LOCATION($$);
120     }
121 ;
122
123 const_value:
124     LITERAL_INT {
125         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::INT);
126         SET_LOCATION($$);
127     }
128 | LITERAL_FLOAT {
129         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::FLOAT);
130         SET_LOCATION($$);
131     }
132 | LITERAL_CHAR {
133         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::CHAR);
134         SET_LOCATION($$);
135     }
136 | LITERAL_STR {
137         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::STRING);
138         SET_LOCATION($$);
139     }
140 | LITERAL_FALSE {
141         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::FALSE);
142         SET_LOCATION($$);
143     }
144 | LITERAL_TRUE {
145         $$ = new AST_Const_Value($1,AST_Const_Value::value_Type::TRUE);
146         SET_LOCATION($$);
147     }
148 ;
149
150 type_decl_list:
151     type_decl_list type_definition {
152         ($1) -> Add_Type_Definition($2);
153         $$ = $1;
154         SET_LOCATION($$);
155     }
156 | type_definition {
157         $$ = new AST_Type_Declaration_List();
158         ($$) -> Add_Type_Definition($1);
159         #ifdef GEN_DEBUG
160         std::cout << "yacc Add_Type_Definition" <<std::endl;
161         #endif
162         SET_LOCATION($$);
163     }
164 ;

```

```

165
166 type_definition:
167     IDENTIFIER SYM_EQ type_decl SYM_SEMICOLON {
168         $$ = new AST_Type_Definition($1,$3);
169         #ifdef GEN_DEBUG
170             std::cout << "yacc new AST_Type_Definition" <<std::endl;
171         #endif
172         SET_LOCATION($$);
173     }
174 ;
175
176 type_decl:
177     simple_type_decl {
178         $$ = $1;
179         #ifdef GEN_DEBUG
180             std::cout << "yacc simple_type_decl" <<std::endl;
181         #endif
182         SET_LOCATION($$);
183     }
184     | array_type_decl {
185         $$ = $1;
186         #ifdef GEN_DEBUG
187             std::cout << "yacc array_type_decl" <<std::endl;
188         #endif
189         SET_LOCATION($$);
190     }
191     | record_type_decl {
192         $$ = $1;
193         #ifdef GEN_DEBUG
194             std::cout << "yacc record_type_decl" <<std::endl;
195         #endif
196         SET_LOCATION($$);
197     }
198 ;
199
200 simple_type_decl:
201     easy_type {
202         $$ = new AST_Simple_Type_Declaration($1);
203         SET_LOCATION($$);
204     }
205     | IDENTIFIER {
206         $$ = new AST_Simple_Type_Declaration($1);
207         SET_LOCATION($$);
208     }
209     | SYM_LPAREN name_list SYM_RPAREN {
210         $$ = new AST_Simple_Type_Declaration($2);
211         SET_LOCATION($$);
212     }
213     | const_value SYM_RANGE const_value {
214         $$ = new AST_Simple_Type_Declaration($1,$3);
215         SET_LOCATION($$);
216     }
217     | IDENTIFIER SYM_RANGE IDENTIFIER {
218         $$ = new AST_Simple_Type_Declaration($1,$3);
219         SET_LOCATION($$);
220     }
221 ;
222

```

```

223 easy_type:
224     TYPE_BOOLEAN {
225         $$ = new AST_Type(AST_Type::Type_Name::BOOLEAN);
226         SET_LOCATION($$);
227     }
228     | TYPE_CHAR {
229         $$ = new AST_Type(AST_Type::Type_Name::CHAR);
230         SET_LOCATION($$);
231     }
232     | TYPE_INT {
233         $$ = new AST_Type(AST_Type::Type_Name::INT);
234         SET_LOCATION($$);
235     }
236     | TYPE_FLOAT {
237         $$ = new AST_Type(AST_Type::Type_Name::FLOAT);
238         SET_LOCATION($$);
239     }
240     | TYPE_STRING {
241         $$ = new AST_Type(AST_Type::Type_Name::STRING);
242         SET_LOCATION($$);
243     }
244 ;
245
246 name_list:
247     name_list SYM_COMMA IDENTIFIER {
248         ($1) -> Add_Identifier($3);
249         $$ = $1;
250         SET_LOCATION($$);
251     }
252     | IDENTIFIER {
253         $$ = new AST_Name_List();
254         ($$) -> Add_Identifier($1);
255         SET_LOCATION($$);
256     }
257 ;
258
259 array_type_decl:
260     KEY_ARRAY SYM_LBRAC simple_type_decl SYM_RBRAC KEY_OF type_decl {
261         $$ = new AST_Array_Type_Declaration($3,$6);
262         SET_LOCATION($$);
263     }
264 ;
265
266 record_type_decl:
267     KEY_RECORD field_decl_list KEY_END {
268         $$ = new AST_Record_Type_Declaration($2);
269         SET_LOCATION($$);
270     }
271 ;
272
273 field_decl_list:
274     field_decl_list field_decl {
275         ($1) -> Add_Field_Declaration($2);
276         $$ = $1;
277         SET_LOCATION($$);
278     }
279     | field_decl {
280         $$ = new AST_Field_Declaration_List($1);

```

```

281     SET_LOCATION($$);
282 }
283 ;
284
285 field_decl:
286   name_list SYM_COLON type_decl SYM_SEMICOLON {
287     $$ = new AST_Field_Declaration($1,$3);
288     SET_LOCATION($$);
289   }
290 ;
291
292 var_decl_list:
293   var_decl_list var_decl {
294     ($1) -> Add_Variable_Declaration($2);
295     $$ = $1;
296     SET_LOCATION($$);
297   }
298   | var_decl {
299     $$ = new AST_Variable_Declaration_List();
300     ($$) -> Add_Variable_Declaration($1);
301     SET_LOCATION($$);
302   }
303 ;
304
305 var_decl:
306   name_list SYM_COLON type_decl SYM_SEMICOLON {
307     $$ = new AST_Variable_Declaration($1,$3);
308     SET_LOCATION($$);
309   }
310 ;
311
312 function_decl:
313   function_head SYM_SEMICOLON routine SYM_SEMICOLON {
314     $$ = new AST_Function_Declaration($1, $3);
315     SET_LOCATION($$);
316   }
317 ;
318
319 function_head:
320   KEY_FUNCTION IDENTIFIER parameters SYM_COLON simple_type_decl {
321     $$ = new AST_Function_Head($2, $3, $5);
322     SET_LOCATION($$);
323   }
324 ;
325
326 procedure_decl:
327   procedure_head SYM_SEMICOLON routine SYM_SEMICOLON {
328     $$ = new AST_Procedure_Declaration($1, $3);
329     SET_LOCATION($$);
330   }
331 ;
332
333 procedure_head:
334   KEY_PROCEDURE IDENTIFIER parameters {
335     $$ = new AST_Procedure_Head($2, $3);
336     SET_LOCATION($$);
337   }
338 ;

```

```

339
340 parameters:
341     SYM_LPAREN para_decl_list SYM_RPAREN {
342         $$ = new AST_Parameters($2);
343         SET_LOCATION($$);
344     }
345     | {
346         $$ = new AST_Parameters();
347         SET_LOCATION($$);
348     }
349 ;
350
351 para_decl_list:
352     para_decl_list SYM_SEMICOLON para_type_list {
353         ($1) -> Add_Parameters_Type_List($3);
354         $$ = $1;
355         SET_LOCATION($$);
356     }
357     | para_type_list {
358         $$ = new AST_Parameters_Declaration_List($1);
359         SET_LOCATION($$);
360     }
361 ;
362
363 para_type_list:
364     var_para_list SYM_COLON simple_type_decl {
365         $$ = new AST_Parameters_Type_List($1,$3);
366         SET_LOCATION($$);
367     }
368     | name_list SYM_COLON simple_type_decl {
369         $$ = new AST_Parameters_Type_List($1,$3);
370         SET_LOCATION($$);
371     }
372 ;
373
374 var_para_list:
375     KEY_VAR name_list {
376         $$ = new AST_Variable_Parameters_List($2);
377         SET_LOCATION($$);
378     }
379 ;
380
381 stmt_list:
382     stmt_list stmt SYM_SEMICOLON {
383         std::cout << "yacc add stmt" << std::endl;
384         ($1) -> Add_Statement($2);
385         $$ = $1;
386         SET_LOCATION($$);
387     }
388     | {
389         std::cout << "yacc empty stmt" << std::endl;
390
391         $$ = new AST_Statement_List();
392         SET_LOCATION($$);
393     }
394 ;
395
396 stmt:

```

```

397     label SYM_COLON non_label_stmt {
398         $$ = new AST_Statement();
399         ($$) -> Add_Label_and_Non_Label_Statement($1,$3);
400         SET_LOCATION($$);
401     }
402     | non_label_stmt {
403         $$ = new AST_Statement();
404         ($$) -> Add_Non_Label_Statement($1);
405         SET_LOCATION($$);
406     }
407 ;
408
409 label:
410     LITERAL_INT {
411         $$ = new AST_Label($1);
412         SET_LOCATION($$);
413     }
414     | IDENTIFIER {
415         $$ = new AST_Label($1);
416         SET_LOCATION($$);
417     }
418 ;
419
420 non_label_stmt:
421     assign_stmt {
422         $$ = new AST_Non_Label_Statement($1);
423         SET_LOCATION($$);
424     }
425     | proc_stmt {
426         $$ = new AST_Non_Label_Statement($1);
427         SET_LOCATION($$);
428     }
429     | compound_stmt {
430         $$ = new AST_Non_Label_Statement($1);
431         SET_LOCATION($$);
432     }
433     | if_stmt {
434         $$ = new AST_Non_Label_Statement($1);
435         SET_LOCATION($$);
436     }
437     | case_stmt {
438         $$ = new AST_Non_Label_Statement($1);
439         SET_LOCATION($$);
440     }
441     | repeat_stmt {
442         $$ = new AST_Non_Label_Statement($1);
443         SET_LOCATION($$);
444     }
445     | while_stmt {
446         $$ = new AST_Non_Label_Statement($1);
447         SET_LOCATION($$);
448     }
449     | for_stmt {
450         $$ = new AST_Non_Label_Statement($1);
451         SET_LOCATION($$);
452     }
453     | goto_stmt {
454         $$ = new AST_Non_Label_Statement($1);

```

```

455     SET_LOCATION($$);
456 }
457 | break_stmt {
458     $$ = new AST_Non_Label_Statement($1);
459     SET_LOCATION($$);
460 }
461 ;
462
463 assign_stmt:
464     IDENTIFIER SYM_ASSIGN expression {
465         $$ = new AST_Assign_Statement($1, $3);
466         SET_LOCATION($$);
467     }
468 | IDENTIFIER SYM_LBRAC expression SYM_RBRAC SYM_ASSIGN expression {
469     $$ = new AST_Assign_Statement($1, $3, $6);
470     SET_LOCATION($$);
471 }
472 | IDENTIFIER SYM_PERIOD IDENTIFIER SYM_ASSIGN expression {
473     $$ = new AST_Assign_Statement($1, $3, $5);
474     SET_LOCATION($$);
475 }
476 ;
477
478 proc_stmt:
479 /* SYS_PROC | SYS_PROC LP expression_list RP | READ LP factor RP */
480 */
481     IDENTIFIER {
482         $$ = new AST_Procedure_Statement($1);
483         SET_LOCATION($$);
484     }
485 | IDENTIFIER SYM_LPAREN expression_list SYM_RPAREN {
486     $$ = new AST_Procedure_Statement($1,$3);
487     SET_LOCATION($$);
488 }
489 ;
490
491 if_stmt:
492     KEY_IF expression KEY_THEN stmt else_clause {
493         $$ = new AST_If_Statement($2,$4,$5);
494         SET_LOCATION($$);
495     }
496 ;
497
498 else_clause:
499     KEY_ELSE stmt {
500         $$ = new AST_Else_Clause($2);
501         SET_LOCATION($$);
502     }
503 | {
504     $$ = nullptr;
505 }
506 ;
507
508 case_stmt:
509     KEY_CASE expression KEY_OF case_expr_list KEY_END {
510         $$ = new AST_Case_Statement($2,$4);

```

```

512     SET_LOCATION($$);
513 }
514 ;
515
516 case_expr_list:
517     case_expr_list case_expr {
518         ($1)->Add_Case_Expression($2);
519         $$ = $1;
520         SET_LOCATION($$);
521     }
522     | case_expr {
523         $$ = new AST_Case_Expression_List();
524         ($$)->Add_Case_Expression($1);
525         SET_LOCATION($$);
526     }
527 ;
528
529 case_expr:
530     const_value SYM_COLON stmt SYM_SEMICOLON {
531         $$ = new AST_Case_Expression($1,$3);
532         SET_LOCATION($$);
533     }
534     | IDENTIFIER SYM_COLON stmt SYM_SEMICOLON {
535         $$ = new AST_Case_Expression($1,$3);
536         SET_LOCATION($$);
537     }
538 ;
539
540 repeat_stmt:
541     KEY_REPEAT stmt_list KEY_UNTIL expression {
542         $$ = new AST_Repeat_Statement($2,$4);
543         SET_LOCATION($$);
544     }
545 ;
546
547 while_stmt:
548     KEY_WHILE expression KEY_DO stmt {
549         $$ = new AST_While_Statement($2,$4);
550         SET_LOCATION($$);
551     }
552 ;
553
554 for_stmt:
555     KEY_FOR IDENTIFIER SYM_ASSIGN expression direction expression KEY_DO
stmt {
556         $$ = new AST_For_Statement($2, $4, $5, $6, $8);
557         SET_LOCATION($$);
558     }
559 ;
560
561 direction:
562     KEY_TO {
563         $$ = new AST_Direction();
564         ($$) -> Set_To();
565         SET_LOCATION($$);
566     }
567     | KEY_DOWNTO {
568         $$ = new AST_Direction();

```

```

569         ($$) -> Set_Down_To();
570         SET_LOCATION($$);
571     }
572 ;
573
574 goto_stmt:
575     KEY_GOTO Label {
576         $$ = new AST_Goto_Statement($2);
577         SET_LOCATION($$);
578     }
579 ;
580
581 break_stmt:
582     KEY_BREAK {
583         $$ = new AST_Break_Statement();
584         SET_LOCATION($$);
585     }
586 ;
587
588 expression_list:
589     expression_list SYM_COMMA expression{
590         $$ = $1;
591         $$ -> Add_Expression($3);
592         SET_LOCATION($$);
593     }
594     | expression {
595         $$ = new AST_Expression_List();
596         $$ -> Add_Expression($1);
597         SET_LOCATION($$);
598     }
599 ;
600
601 expression:
602     expression SYM_GE expr{
603         $$ = new
604             AST_Binary_Expression(AST_Binary_Expression::Operation::GE, $1, $3);
605             SET_LOCATION($$);
606     }
607     | expression SYM_GT expr {
608         $$ = new
609             AST_Binary_Expression(AST_Binary_Expression::Operation::GT, $1, $3);
610             SET_LOCATION($$);
611     }
612     | expression SYM_LE expr {
613         $$ = new
614             AST_Binary_Expression(AST_Binary_Expression::Operation::LE, $1, $3);
615             SET_LOCATION($$);
616     }
617     | expression SYM_LT expr {
618         $$ = new
619             AST_Binary_Expression(AST_Binary_Expression::Operation::LT, $1, $3);
620             SET_LOCATION($$);
621     }
622     | expression SYM_EQ expr {
623         $$ = new
624             AST_Binary_Expression(AST_Binary_Expression::Operation::EQUAL, $1, $3);
625             SET_LOCATION($$);
626     }

```

```

622     | expression SYM_NE expr{
623         $$ = new
624         AST_Binary_Expression(AST_Binary_Expression::Operation::UNEQUAL, $1, $3);
625         SET_LOCATION($$);
626     }
627     | expr{
628         $$ = $1;
629         SET_LOCATION($$);
630     }
631 ;
632
633 expr:
634     expr SYM_ADD term {
635         $$ = new
636         AST_Binary_Expression(AST_Binary_Expression::Operation::PLUS,$1,$3);
637         SET_LOCATION($$);
638     }
639     | expr SYM_SUB term {
640         $$ = new
641         AST_Binary_Expression(AST_Binary_Expression::Operation::MINUS,$1,$3);
642         SET_LOCATION($$);
643     }
644     | expr KEY_OR term {
645         $$ = new
646         AST_Binary_Expression(AST_Binary_Expression::Operation::OR,$1,$3);
647         SET_LOCATION($$);
648     }
649 ;
650
651 term:
652     term SYM_MUL factor{
653         $$ = new
654         AST_Binary_Expression(AST_Binary_Expression::Operation::MUL,$1,$3);
655         SET_LOCATION($$);
656     }
657     | term SYM_DIV factor{
658         $$ = new
659         AST_Binary_Expression(AST_Binary_Expression::Operation::REALDIV,$1,$3);
660         SET_LOCATION($$);
661     }
662     | term KEY_DIV factor {
663         $$ = new
664         AST_Binary_Expression(AST_Binary_Expression::Operation::DIV,$1,$3);
665         SET_LOCATION($$);
666     }
667     | term KEY_MOD factor {
668         $$ = new
669         AST_Binary_Expression(AST_Binary_Expression::Operation::MOD,$1,$3);
670         SET_LOCATION($$);
671     }
672     | term KEY_AND factor{
673         $$ = new
674         AST_Binary_Expression(AST_Binary_Expression::Operation::AND,$1,$3);
675         SET_LOCATION($$);
676     }

```

```

671    }
672    | factor {
673        $$ = $1;
674        SET_LOCATION($$);
675    }
676    ;
677
678 factor:
679     IDENTIFIER{
680         $$ = new AST_Identifier_Expression($1);
681         SET_LOCATION($$);
682     }
683     | IDENTIFIER SYM_LPAREN expression_list SYM_RPAREN {
684         $$ = new AST_Function_Call($1,$3);
685         SET_LOCATION($$);
686     }
687     | const_value{
688         $$ = new AST_Const_Value_Expression($1);
689         SET_LOCATION($$);
690     }
691     | SYM_LPAREN expression SYM_RPAREN{
692         $$ = $2;
693         SET_LOCATION($$);
694     }
695     | KEY_NOT factor{
696         $$ = new
697             AST_Unary_Expression(AST_Unary_Expression::Operation::NOT,$2);
698             SET_LOCATION($$);
699     }
700     | SYM_SUB factor{
701         $$ = new
702             AST_Unary_Expression(AST_Unary_Expression::Operation::SUB,$2);
703             SET_LOCATION($$);
704     }
705     | SYM_ADD factor{
706         $$ = new
707             AST_Unary_Expression(AST_Unary_Expression::Operation::ADD,$2);
708             SET_LOCATION($$);
709     }
710     | IDENTIFIER SYM_LBRAC expression SYM_RBRAC{
711         $$ = new AST_Array_Expression($1,$3);
712         SET_LOCATION($$);
713     }
714     | IDENTIFIER SYM_PERIOD IDENTIFIER{
715         $$ = new AST_Property_Expression($1,$3);
716         SET_LOCATION($$);
717     }
718     ;

```