浙江大学
ZHEJIANG UNIVERSITY

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

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
- GraphViz实现语法树可视化
- 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，AST可视化，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

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：

```
#define YY_USER_ACTION yylloc.first_line = yylloc.last_line = yylineno; \
    yylloc.first_column = yycolumn; yylloc.last_column = yycolumn + yyleng - 1; \
    yycolumn += yyleng;
```

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。

```
{definitions}
%%
{rules}
%%
{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。

```
KEY_AND and

...

TYPE_INT integer

...


SYM_ADD "+"

...
```

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。

```
LITERAL_INT [0-9]|[1-9][0-9]+
LITERAL_FLOAT ([0-9]+\.[0-9]+)|([0-9]+\.[0-9]+e{SIGN}?[0-9]+)|([0-9]+e{SIGN}?[0-9]+)
LITERAL_CHAR \'.\'
LITERAL_ESC_CHAR '\'#\'
LITERAL_STR \'([^']|{LITERAL_ESC_CHAR})*\'

COMMENT1 "{"[^\}]*"}"
COMMENT2 "//"[^\n]*
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yylval。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。

```
#ifdef DEBUG_LEXER
#define OUR_RETURN(X) cout << (#X) << endl;
#else
#define OUR_RETURN(X) return(X);
#endif

```

```
...

{KEY_AND} {
    OUR_RETURN(KEY_AND)
}

...

{LITERAL_INT} {
    yylval = strdup(yytext);
    OUR_RETURN(LITERAL_INT)
}

...

{COMMENT1} {
}

{COMMENT2} {
}

\n|(\r\n) {
    yycolumn = 1;
}
" "|\t {}

. {
    printf("Unknown Character %d\n", (int) yytext[0]);
    return yytext[0];
}
```

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。

```
declarations
%%
rules
%%
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。

```
#pragma once
#include <string>
#include <vector>
#include <iostream>
#include <string>
#include "../type/type.hpp"
#include <llvm/IR/Value.h>

class AST_BaseNode{
    private:
        // location for error information
        int column;
        int row;
    public:
        AST_BaseNode() = default; //default
        AST_BaseNode(int _column, int _row):column(_column), row(_row){}
        int GetColumn() const{
            return this->column;
        }
        int GetRow() const{
            return this->row;
        }
        std::string GetLocationString() const{
            return "(" + std::to_string(this->row) + "," +
std::to_string(this->column) + ")";
        }

        std::pair<int,int> GetLocation() const{
            return std::make_pair(this->column, this->row);
        }

        void SetColumn(int _column){
            this->column = _column;
        }
```

```
        void SetRow(int _row){
            this->row = _row;
        }
        void SetLocation(std::pair<int, int> location){
            this->column = location.first;
            this->row = location.second;
        }
        void SetLocation(int _column, int _row){
            this->column = _column;
            this->row = _row;
        }

        virtual void LevelOrder(){
            std::cout << "root" << std::endl;
        }

        virtual std::shared_ptr<Custom_Result> CodeGenerate() = 0;
};
```

3.2.2 AST_Expression类及其子类

AST_Expression 类及其子类种类如下：

```
#pragma once
#include "AST_BaseNode.hpp"
#include "AST_Value.hpp"

class AST_Expression;
class AST_Expression_List;
/* 下面的类作为AST_Expression的子类 */
class AST_Binary_Expression;
class AST_Unary_Expression;
class AST_Property_Expression;
class AST_Const_Value_Expression;
class AST_Function_Call;
class AST_Identifier_Expression;
class AST_Array_Expression;

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。

```
class AST_Expression : public AST_BaseNode
{
public:
    AST_Expression() = default;
```

```
};

class AST_Expression_List : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Expression_List() = default;
    void Add_Expression(AST_Expression *expr)
    {
        expr_list.push_back(expr);
    }

    std::vector<AST_Expression *> expr_list;
};

```

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

```
/**
 * @brief : 二元表达式
 *
 */
class AST_Binary_Expression : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Operation
    {
        GE,
        GT,
        LE,
        LT,
        EQUAL,
        UNEQUAL,
        PLUS,
        MINUS,
        OR,
        MUL,
        REALDIV,
        DIV,
        MOD,
        AND
```

```
    };
    AST_Binary_Expression(Operation _my_operation, AST_Expression *_left_expression, AST_Expression *_right_expression) : my_operation(_my_operation), left_expression(_left_expression), right_expression(_right_expression){};

    static std::string Get_Operation_Name(Operation my_op)
    {
        switch (my_op)
        {
        case Operation::GE:
            return "GE";
            break;
        case Operation::GT:
            return "GT";
            break;
        case Operation::LE:
            return "LE";
            break;
        case Operation::LT:
            return "LT";
            break;
        case Operation::EQUAL:
            return "EQUAL";
            break;
        case Operation::UNEQUAL:
            return "UNEQUAL";
            break;
        case Operation::PLUS:
            return "PLUS";
            break;
        case Operation::MINUS:
            return "MINUS";
            break;
        case Operation::OR:
            return "OR";
            break;
        case Operation::MUL:
            return "MUL";
            break;
        case Operation::DIV:
            return "DIV";
            break;
        case Operation::MOD:
            return "MOD";
            break;
        case Operation::AND:
            return "AND";
            break;
        }
        return "";
    }
```

```
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:
```

```
    AST_Property_Expression(std::string _id, std::string _prop_id) : id(_id), prop_id(_prop_id){};
};
```

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。

```
/**
 * @brief : 常数表达式
 *
 */
class AST_Const_Value_Expression : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

    AST_Const_Value *const_value;

public:
    AST_Const_Value_Expression(AST_Const_Value *_const_value) : const_value(_const_value){};
};
```

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

```
/**
 * @brief : 常数表达式
 *
 */
class AST_Const_Value_Expression : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

    AST_Const_Value *const_value;

public:
    AST_Const_Value_Expression(AST_Const_Value *_const_value) : const_value(_const_value){};
};

class AST_Function_Call : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

```

```
    /*func_id:函数名 ; args_list:参数列表*/
    std::string func_id;
    AST_Expression_List *args_list;

public:
    AST_Function_Call(std::string _func_id, AST_Expression_List
*_args_list) : func_id(_func_id), args_list(_args_list){};
};
```

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：

```
class AST_Identifier_Expression : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

    std::string id;

public:
    AST_Identifier_Expression(std::string _id) : id(_id){};
};

class AST_Array_Expression : public AST_Expression
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

    std::string id;
    AST_Expression *expression;

public:
    AST_Array_Expression(std::string _id, AST_Expression *_expression) :
id(_id), expression(_expression){};
};
```

3.2.3 AST_Program类及其相关类

AST_Program类及其子类种类如下：

```
#pragma once
#include "AST_BaseNode.hpp"
#include "AST_Type.hpp"
#include "AST_Expression.hpp"

class AST_Variable_Part;
```

```
class AST_Type_Part;
class AST_Const_Part;

class AST_Compound_Statement;
class AST_Program_Head;
class AST_Routine;
class AST_Routine_Body;
class AST_Routine_Head;

class AST_Routine_Part;
class AST_Parameters_Declaration_List;
class AST_Parameters;
class AST_Function_Declaration;
class AST_Procedure_Declaration;
class AST_Procedure_Head;
class AST_Parameters_Type_List;
class AST_Variable_Parameters_List;
class AST_Function_Head;

```

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。

```
/*
program:
    pro_head routine SYM_PERIOD
;
*/
class AST_Program : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Program_Head *program_head;
    AST_Routine *routine;

public:
    AST_Program(AST_Program_Head *_program_head, AST_Routine *_routine) :
program_head(_program_head), routine(_routine){};

    AST_Program_Head *Get_Program_Head() const
    {
        return this->program_head;
    }

    AST_Routine *Get_Routine() const
```

```
    {
        return this->routine;
    }
};

/*
pro_head:
    KEY_PROGRAM IDENTIFIER SYM_SEMICOLON{
    }
;
*/
class AST_Program_Head : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    std::string identifier;

public:
    AST_Program_Head(std::string _identifier) : identifier(_identifier){};

    std::string Get_Identifier() const
    {
        return identifier;
    }
};

/*
routine:
    routine_head routine_body{}
;
*/

class AST_Routine : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Routine_Head *routine_head;
    AST_Routine_Body *routine_body;

public:
    AST_Routine(AST_Routine_Head *_routine_head, AST_Routine_Body
*_routine_body) : routine_head(_routine_head), routine_body(_routine_body)
{};

    AST_Routine_Head *Get_Routine_Head() const
    {
        return this->routine_head;
```

```
    }

    AST_Routine_Body *Get_Routine_Body() const
    {
        return this->routine_body;
    }
};
```

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：

```
/*
routine_head:
    const_part type_part var_part routine_part{}
;
*/
class AST_Routine_Head : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Const_Part *const_part;
    AST_Type_Part *type_part;
    AST_Variable_Part *var_part;
    AST_Routine_Part *routine_part;

public:
    AST_Routine_Head(AST_Const_Part *_const_part, AST_Type_Part
*_type_part, AST_Variable_Part *_var_part, AST_Routine_Part
*_routine_part) : const_part(_const_part), type_part(_type_part),
var_part(_var_part), routine_part(_routine_part){};
    AST_Const_Part *Get_Const_Part() const
    {
        return this->const_part;
    }
    AST_Type_Part *Get_Type_Part() const
    {
        return this->type_part;
    }
    AST_Variable_Part *Get_Variable_Part() const
    {
        return this->var_part;
    }
    AST_Routine_Part *Get_Routine_Part() const
    {
        return this->routine_part;
```

```
    }
};


/*
routine_body:
    compound_stmt
;
*/

class AST_Routine_Body : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Compound_Statement *compound_statement;

public:
    AST_Routine_Body(AST_Compound_Statement *_compound_statement) :
compound_statement(_compound_statement){};
    AST_Compound_Statement *Get_Compound_Statement() const
    {
        return this->compound_statement;
    }
};

```

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：

```
routine_part:  {function_decl | procedure_decl}
    routine_part function_decl
    | routine_part procedure_decl
    | function_decl
    | procedure_decl
    |
;
```

`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：

```

/// for routine part list
class AST_Declaration_BaseClass : public AST_BaseNode
{
public:
    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
    *_function_declaration) : function_declaration(_function_declaration)
20      {
21          this->declaration_type =
    ENUM_Declaration_Type::FUNCTION_DECLARATION;
22      }
23      AST_Declaration_BaseClass(AST_Procedure_Declaration
    *_procedure_declaration) : procedure_declaration(_procedure_declaration)
24      {
25          this->declaration_type =
    ENUM_Declaration_Type::PROCEDURE_DECLARATION;
26      }
27
28      AST_Function_Declaration *Get_Function_Declaration() const
29      {
30          return this->function_declaration;
31      }
32      AST_Procedure_Declaration *Get_Procedure_Declaration() const
33      {
34          return this->procedure_declaration;
35      }
36      ENUM_Declaration_Type Get_Declaration_Type() const
37      {
38          return this->declaration_type;
39      }
40  };
41
42  /*
43  routine_part:  {function_decl | procedure_decl}
44      routine_part function_decl
45      | routine_part procedure_decl
46      | function_decl
47      | procedure_decl
48      |
49  ;
50  */
51  class AST_Routine_Part : public AST_BaseNode
52  {
53  public:
```

```
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    std::vector<AST_Declaration_BaseClass *> declaration_list;

public:
    void Add_Declaration(AST_Declaration_BaseClass *_declaration)
    {
        this->declaration_list.push_back(_declaration);
    }
    AST_Routine_Part(AST_Declaration_BaseClass *_declaration)
    {
        this->declaration_list.clear();
        Add_Declaration(_declaration);
    }
    std::vector<AST_Declaration_BaseClass *> Get_Declaration_List() const
    {
        return declaration_list;
    }
};
```

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

```
procedure_decl:
    procedure_head SYM_SEMICOLON routine SYM_SEMICOLON
;
function_decl:
    function_head SYM_SEMICOLON routine SYM_SEMICOLON
;

```

对应的定义函数也很简单：

```

/*
function_decl:
    function_head SYM_SEMICOLON routine SYM_SEMICOLON
;
*/
class AST_Function_Declaration : public AST_BaseNode
{
public:
    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,
    AST_Routine *_routine) : procedure_head(_procedure_head),
    routine(_routine){};
45      AST_Procedure_Head *Get_Procedure_Head() const
46      {
47          return this->procedure_head;
48      }
49      AST_Routine *Get_Routine() const
50      {
51          return this->routine;
52      }
53  };
54
```

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。

```
/*
function_head:
    KEY_FUNCTION IDENTIFIER parameters SYM_COLON simple_type_decl
;
*/
class AST_Function_Head : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    std::string identifier;
    AST_Parameters *parameters;
    AST_Simple_Type_Declaration *simple_type_declaration;

public:
    AST_Function_Head(std::string _identifier, AST_Parameters *_parameters, AST_Simple_Type_Declaration *_simple_type_declaration) : identifier(_identifier), parameters(_parameters), simple_type_declaration(_simple_type_declaration){};
    std::string Get_Identifier() const
    {
        return this->identifier;
    }
    AST_Parameters *Get_Parameters() const
    {
        return this->parameters;
    }
    AST_Simple_Type_Declaration *Get_Simple_Type_Declaration() const
    {
        return this->simple_type_declaration;
    }
};

/*
procedure_head:
    KEY_PROCEDURE IDENTIFIER parameters
;
*/
class AST_Procedure_Head : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    std::string identifier;
```

```
    AST_Parameters *parameters;

public:
    AST_Procedure_Head(std::string _identifier, AST_Parameters
*_parameters) : identifier(_identifier), parameters(_parameters){};
    std::string Get_Identifier() const
    {
        return this->identifier;
    }
    AST_Parameters *Get_Parameters() const
    {
        return this->parameters;
    }
};
```

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：

```
parameters:
    SYM_LPAREN para_decl_list SYM_RPAREN {}
    | {}
;
para_decl_list:
    para_decl_list SYM_SEMICOLON para_type_list
    | para_type_list
;
```

对应的代码如上分析所述：

```
/*
parameters:
    SYM_LPAREN para_decl_list SYM_RPAREN {}
    | {}
;
*/
class AST_Parameters : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Parameters_Declaration_List *parameters_declaration_list;

```

```
15  public:
16      AST_Parameters() = default;
17      AST_Parameters(AST_Parameters_Declaration_List
    *_parameters_declaration_list) :
    parameters_declaration_list(_parameters_declaration_list){};
18      AST_Parameters_Declaration_List *Get_Parameters_Declaration_List()
    const
19      {
20          return this->parameters_declaration_list;
21      }
22  };
23
24  /*
25  para_decl_list:
26      para_decl_list SYM_SEMICOLON para_type_list
27      | para_type_list
28  ;
29  */
30  class AST_Parameters_Declaration_List : public AST_BaseNode
31  {
32  public:
33      std::shared_ptr<Custom_Result>CodeGenerate() override;
34
35  private:
36      std::vector<AST_Parameters_Type_List *> parameters_type_list_list;
37
38  public:
39      void Add_Parameters_Type_List(AST_Parameters_Type_List
    *_parameters_type_list)
40      {
41          this->parameters_type_list_list.push_back(_parameters_type_list);
42      }
43      AST_Parameters_Declaration_List(AST_Parameters_Type_List
    *_parameters_type_list)
44      {
45          this->parameters_type_list_list.clear();
46          Add_Parameters_Type_List(_parameters_type_list);
47      }
48      AST_Parameters_Declaration_List()
49      {
50          this->parameters_type_list_list.clear();
51      }
52      std::vector<AST_Parameters_Type_List *> Get_Parameter_Type_List_List()
    const
53      {
54          return this->parameters_type_list_list;
55      }
56  };
```

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：

```
para_type_list:
    var_para_list SYM_COLON simple_type_decl
    | name_list SYM_COLON simple_type_decl
;
var_para_list:
    KEY_VAR name_list
;
```

据此我们很方便可以定义出两个AST构造函数类：

```
/*
para_type_list:
    var_para_list SYM_COLON simple_type_decl
    | name_list SYM_COLON simple_type_decl
;
*/
class AST_Parameters_Type_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    enum class List_Type
    {
        VARIABLE_PARAMETERS_LIST,
        NAME_LIST
    };
    AST_Variable_Parameters_List *variable_parameters_list;
    AST_Name_List *name_list;
    List_Type list_type;
    AST_Simple_Type_Declaration *simple_type_declaration;

public:
    AST_Parameters_Type_List(AST_Variable_Parameters_List
*_variable_parameters_list, AST_Simple_Type_Declaration
*_simple_type_declaration) :
variable_parameters_list(_variable_parameters_list),
simple_type_declaration(_simple_type_declaration),
list_type(List_Type::VARIABLE_PARAMETERS_LIST){};
    AST_Parameters_Type_List(AST_Name_List *_name_list,
AST_Simple_Type_Declaration *_simple_type_declaration) :
name_list(_name_list), simple_type_declaration(_simple_type_declaration),
list_type(List_Type::NAME_LIST){};
    AST_Variable_Parameters_List *Get_Variable_Parameters_List() const
    {
        return this->variable_parameters_list;
    }
```

```
    AST_Name_List *Get_Name_List() const
    {
        return this->name_list;
    }
    AST_Simple_Type_Declaration *Get_Simple_Type_Declaration() const
    {
        return this->simple_type_declaration;
    }
    List_Type Get_List_Type() const
    {
        return this->list_type;
    }
    bool isVar(){
        return this->list_type == List_Type::VARIABLE_PARAMETERS_LIST;
    }
};

/*
var_para_list:
    KEY_VAR name_list
;
*/
class AST_Variable_Parameters_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

private:
    AST_Name_List *name_list;

public:
    AST_Variable_Parameters_List(AST_Name_List *_name_list) :
name_list(_name_list){};
    AST_Name_List *Get_Name_List() const
    {
        return this->name_list;
    }
};

```

3.2.4 AST_Statement类及其相关类

AST_Statement类及其子类种类如下：

```
#pragma once
#include "AST_BaseNode.hpp"
#include "AST_Expression.hpp"
```

```
#include "AST_Value.hpp"

class AST_Compound_Statement;
class AST_Statement_List;
class AST_Statement;
class AST_Label;
class AST_Non_Label_Statement;
class AST_Assign_Statement;
class AST_Procedure_Statement;

class AST_If_Statement;
class AST_Case_Statement;
class AST_Repeat_Statement;
class AST_While_Statement;
class AST_For_Statement;
class AST_Goto_Statement;
class AST_Else_Clause;
class AST_Break_Statement;

class AST_Case_Expression_List;
class AST_Case_Expression;
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。

```
/*
compound_stmt:
    KEY_BEGIN stmt_list KEY_END
;
*/
class AST_Compound_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result> CodeGenerate() override;

public:
    AST_Statement_List *statement_list;

public:
    AST_Compound_Statement(AST_Statement_List *_statement_list) :
statement_list(_statement_list){};
    AST_Statement_List *Get_Statement_List() const
    {
        return this->statement_list;
    }
```

```
};

/*
stmt_list:
    stmt_list stmt SYM_SEMICOLON
    |
;
*/
class AST_Statement_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    std::vector<AST_Statement *> statement_list;

public:
    AST_Statement_List() = default;
    void Add_Statement(AST_Statement *_statement)
    {
        this->statement_list.push_back(_statement);
    }
};

/*
stmt:
    label SYM_COLON non_label_stmt
    | non_label_stmt
;
*/
class AST_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Has_Label
    {
        NOT_HAS,
        HAS
    };
    AST_Label *label;
    AST_Non_Label_Statement *non_label_statement;
    Has_Label has_label;

public:
    AST_Statement() = default;
    void Add_Label_and_Non_Label_Statement(AST_Label *_label,
AST_Non_Label_Statement *_non_label_statement)
    {
        this->label = _label;
```

```
        this->non_label_statement = _non_label_statement;
        this->has_label = Has_Label::HAS;
    }
    void Add_Non_Label_Statement(AST_Non_Label_Statement
*_non_label_statement)
    {
        this->non_label_statement = _non_label_statement;
        this->has_label = Has_Label::NOT_HAS;
    }
    AST_Label *Get_Label() const
    {
        return this->label;
    }
    AST_Non_Label_Statement *Get_Non_Label_Statement() const
    {
        return this->non_label_statement;
    }
    Has_Label Get_Has_Label() const
    {
        return this->has_label;
    }
};
```

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。

```
/*
label:
    LITERAL_INT
    | IDENTIFIER
;
*/
class AST_Label : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Int_or_Identifier
    {
        LITERAL_INT,
        IDENTIFIER
    };
    int literal_int;
    std::string identifier;
    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) :
    assgin_statement(_assgin_statement)
96      {
97          this->statement_type = Statement_Type::ASSIGN;
98      }
99      AST_Non_Label_Statement(AST_Procedure_Statement
    *_procedure_statement) : procedure_statement(_procedure_statement)
100     {
101         this->statement_type = Statement_Type::PROCEDURE;
102     }
103     AST_Non_Label_Statement(AST_Compound_Statement *_compound_statement)
    : compound_statement(_compound_statement)
104     {
105         this->statement_type = Statement_Type::COMPOUND;
106     }
107     AST_Non_Label_Statement(AST_If_Statement *_if_statement) :
    if_statement(_if_statement)
108     {
109         this->statement_type = Statement_Type::IF;
110     }
111     AST_Non_Label_Statement(AST_Case_Statement *_case_statement) :
    case_statement(_case_statement)
112     {
113         this->statement_type = Statement_Type::CASE;
114     }
115     AST_Non_Label_Statement(AST_Repeat_Statement *_repeat_statement) :
    repeat_statement(_repeat_statement)
116     {
117         this->statement_type = Statement_Type::REPEAT;
```

```
118     }
119     AST_Non_Label_Statement(AST_While_Statement *_while_statement) :
    while_statement(_while_statement)
120     {
121         this->statement_type = Statement_Type::WHILE;
122     }
123     AST_Non_Label_Statement(AST_For_Statement *_for_statement) :
    for_statement(_for_statement)
124     {
125         this->statement_type = Statement_Type::FOR;
126     }
127     AST_Non_Label_Statement(AST_Goto_Statement *_goto_statement) :
    goto_statement(_goto_statement)
128     {
129         this->statement_type = Statement_Type::GOTO;
130     }
131     AST_Non_Label_Statement(AST_Break_Statement *_break_statement) :
    break_statement(_break_statement)
132     {
133         this->statement_type = Statement_Type::BREAK;
134     }
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：

```
/*
assign_stmt:
    IDENTIFIER SYM_ASSIGN expression
    | IDENTIFIER SYM_LBRAC expression SYM_RBRAC SYM_ASSIGN expression
    | IDENTIFIER SYM_PERIOD IDENTIFIER SYM_ASSIGN expression
;
*/

class AST_Assign_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

    bool ValueAssign(llvm::Value* left_ptr, Pascal_Type* left_type, llvm::Value* right_ptr, Pascal_Type* right_type){
        if (left_type->isArray()){
            //todo type transfer

        }
        return true;
    }

public:
    enum class Assign_Type
    {
        I_1_E_1,
        I_1_E_2,
        I_2_E_1
    };

    Assign_Type assign_type;

    std::string identifier1;
    std::string identifier2;
    AST_Expression *expression1;
    AST_Expression *expression2;

public:
    AST_Assign_Statement(std::string _identifier1, AST_Expression *_expression1) : identifier1(_identifier1), expression1(_expression1), assign_type(Assign_Type::I_1_E_1) {}
```

```
    AST_Assign_Statement(std::string _identifier1, AST_Expression *_expression1, AST_Expression *_expression2) : identifier1(_identifier1), expression1(_expression1), expression2(_expression2), assign_type(Assign_Type::I_1_E_2) {}
    AST_Assign_Statement(std::string _identifier1, std::string _identifier2, AST_Expression *_expression1) : identifier1(_identifier1), identifier2(_identifier2), expression1(_expression1), assign_type(Assign_Type::I_2_E_1) {}
    Assign_Type Get_Assign_Type() const
    {
        return assign_type;
    }
    bool isDirectAssign() const{
        return assign_type == Assign_Type::I_1_E_1;
    }
    bool isArrayAssign() const{
        return assign_type == Assign_Type::I_1_E_2;
    }
    bool isRecordAttrAssign() const{ // record attribute
        return assign_type == Assign_Type::I_2_E_1;
    }
};

```

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：

```
/*
proc_stmt:
    // SYS_PROC | SYS_PROC  LP  expression_list  RP |  READ  LP  factor  RP
    IDENTIFIER
    | IDENTIFIER SYM_LPAREN expression_list SYM_RPAREN
;
*/
class AST_Procedure_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Has_Expression
    {
        Not,
        Yes
    };
    Has_Expression has_expression;

```

```
    std::string identifier;
    AST_Expression_List *expresion_list;

public:
    AST_Procedure_Statement(std::string _identifier) : identifier(_identifier), has_expression(Has_Expression::Not) {}
    AST_Procedure_Statement(std::string _identifier, AST_Expression_List *_expresion_list) : identifier(_identifier), expresion_list(_expresion_list), has_expression(Has_Expression::Yes) {}
    Has_Expression Get_Has_Expression()
    {
        return has_expression;
    }
};

```

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：

```
/*
if_stmt:
    KEY_IF expression KEY_THEN stmt else_clause
;
*/
class AST_If_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Expression *expression;
    AST_Statement *statement;
    AST_Else_Clause *else_clause;

    AST_If_Statement(AST_Expression *_expression, AST_Statement *_statement, AST_Else_Clause *_else_clause) : expression(_expression), statement(_statement), else_clause(_else_clause) {}
};

/*
else_clause:
    KEY_ELSE stmt {}
    | {}
;
*/
class AST_Else_Clause : public AST_BaseNode
{
```

```
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Statement *statement;
    AST_Else_Clause() = default;
    AST_Else_Clause(AST_Statement *_statement) : statement(_statement) {}
    bool isEmpty(){
        return (this->statement == nullptr);
    }
};

```

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：

```
/*
repeat_stmt:
    KEY_REPEAT stmt_list KEY_UNTIL expression
;
*/
class AST_Repeat_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Statement_List *statement_list;
    AST_Expression *expression;

    AST_Repeat_Statement(AST_Statement_List *_statement_list, AST_Expression *_expression) : statement_list(_statement_list), expression(_expression) {}
};

/*
while_stmt:
    KEY_WHILE expression KEY_DO stmt
*/
class AST_While_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Expression *expression;
```

```
    AST_Statement *statement;

    AST_While_Statement(AST_Expression *_expression, AST_Statement
*_statement) : expression(_expression), statement(_statement) {}
};

```

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：

```
/*
for_stmt:
    KEY_FOR IDENTIFIER SYM_ASSIGN expression my_direction
expression KEY_DO stmt
;
*/
class AST_For_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    std::string identifier;
    AST_Expression *expression1;
    AST_Direction *my_direction;
    AST_Expression *expression2;
    AST_Statement *statement;

    AST_For_Statement(std::string _identifier, AST_Expression
*_expression1, AST_Direction *_direction, AST_Expression *_expression2,
AST_Statement *_statement) : identifier(_identifier),
expression1(_expression1), my_direction(_direction),
expression2(_expression2), statement(_statement) {}
};

/*
my_direction:
    KEY_TO
    | KEY_DOWNTO
;
*/
class AST_Direction : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
```

```
    enum class To_or_DownTo
    {
        To,
        DownTo
    };
    To_or_DownTo to_or_downto;
    AST_Direction() = default;
    void Set_To()
    {
        to_or_downto = To_or_DownTo::To;
    }
    void Set_Down_To()
    {
        to_or_downto = To_or_DownTo::DownTo;
    }
    bool isTo(){
        return to_or_downto == To_or_DownTo::To;
    }
};

/*
goto_stmt:
    KEY_GOTO label
;
*/

class AST_Goto_Statement : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Label *label;
    AST_Goto_Statement(AST_Label *_label) : label(_label) {}
};

class AST_Break_Statement : public AST_BaseNode{
    public:
        std::shared_ptr<Custom_Result>CodeGenerate() override;
    public:
        AST_Break_Statement() = default;
};
```

3.2.5 AST_Type类及其相关类

AST_Type类及其相关类种类如下：

```
#pragma once
#include "AST_BaseNode.hpp"
#include "AST_Value.hpp"
```

```
#include <vector>

class AST_BaseNode;
class AST_Const_Value;

class AST_Type_Part;
class AST_Type_Definition;
class AST_Type_Declaration_List;

class AST_Type;
class AST_Type_Declaration;
class AST_Simple_Type_Declaration;
class AST_Array_Type_Declaration;
class AST_Record_Type_Declaration;
class AST_Field_Declaration_List;
class AST_Field_Declaration;
class AST_Name_List;

```

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

```
// easy_type -> int / char / boolean / float / string
class AST_Type : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Type_Name
    {
        INT,
        FLOAT,
        BOOLEAN,
        CHAR,
        STRING
    };
    AST_Type(AST_Type::Type_Name _typename) : my_typename(_typename){};

    AST_Type::Type_Name Get_Type_Name() const
    {
        return this->my_typename;
    }

    AST_Type::Type_Name my_typename;
};

// type_decl -> SimpleTypeDecl / ArrayTypeDecl / RecordTypeDecl
```

```
class AST_Type_Declaration : public AST_BaseNode
{
public:
    virtual std::shared_ptr<Custom_Result> CodeGenerate()=0;
};


/**
 *
 *type_decl_list->{type_definition}
 */
class AST_Type_Declaration_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Type_Declaration_List() = default;
    void Add_Type_Definition(AST_Type_Definition *type_definition)
    {
        type_definition_list.push_back(type_definition);
    }

    std::vector<AST_Type_Definition *> type_definition_list;
};
```

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：

```
/**
type_definition->
    IDENTIFIER SYM_EQ type_decl SYM_SEMICOLON
*/
class AST_Type_Definition : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Type_Definition(std::string id, AST_Type_Declaration *_type_decl)
: identifier(id), type_decl(_type_decl){};

    std::string identifier;
    AST_Type_Declaration *type_decl;
};

/**
type_part->
    KEY_TYPE type_decl_list | %empty
```

```
*/
class AST_Type_Part : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Type_Part(AST_Type_Declaration_List *_type_decl_list) :
type_decl_list(_type_decl_list){};

    AST_Type_Declaration_List *type_decl_list;
};

```

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：

```

/*
simple_type_decl ->
    easy_type
    | IDENTIFIER
    | SYM_LPAREN name_list SYM_RPAREN
    |  const_value  SYM_RANGE  const_value
    |  IDENTIFIER  SYM_RANGE  IDENTIFIER
*/
class AST_Simple_Type_Declaration : public AST_Type_Declaration
{
public:
    std::shared_ptr<Custom_Result> CodeGenerate() override;

public:
    enum class My_Type
    {
        EASY_TYPE,
        IDENTIFIER,
        ENUMERATE,
        VALUE_RANGE,
        IDENTIFIER_RANGE
    };

    AST_Simple_Type_Declaration(AST_Type *_type) : my_typename(_type),
my_type(My_Type::EASY_TYPE){};

    AST_Simple_Type_Declaration(std::string _define_id) :
define_id(_define_id), my_type(My_Type::IDENTIFIER){};

```

```
29      AST_Simple_Type_Declaration(AST_Name_List *_name_list) :
    name_list(_name_list), my_type(My_Type::ENUMERATE){};
30
31      AST_Simple_Type_Declaration(AST_Const_Value *_low, AST_Const_Value
    *_high) : low(_low), high(_high), my_type(My_Type::VALUE_RANGE){};
32
33      AST_Simple_Type_Declaration(std::string _low_name, std::string
    _high_name) : low_name(_low_name), high_name(_high_name),
    my_type(My_Type::IDENTIFIER_RANGE){};
34
35      My_Type Get_Type() const
36      {
37          return this->my_type;
38      }
39
40      My_Type my_type;
41
42      AST_Type *my_typename;
43      std::string define_id;
44      AST_Name_List *name_list;
45      AST_Const_Value *low, *high;
46      std::string low_name, high_name;
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
    *_simple_type_decl, AST_Type_Declaration *_type_decl) :
    simple_type_decl(_simple_type_decl), type_decl(_type_decl){};
60
61      AST_Simple_Type_Declaration *simple_type_decl;
62      AST_Type_Declaration *type_decl;
63  };
```

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   /**
```

```
 * @brief
 * record_type_decl ->
    KEY_RECORD field_decl_list KEY_END
 */
class AST_Record_Type_Declaration : public AST_Type_Declaration
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Record_Type_Declaration(AST_Field_Declaration_List
*_field_decl_list) : field_decl_list(_field_decl_list){};

    AST_Field_Declaration_List *field_decl_list;
};

/**
 * @brief
 * field_decl_list -> {field_decl}
 */
class AST_Field_Declaration_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Field_Declaration_List(AST_Field_Declaration *_field_decl)
    {
        field_decl_list.clear();
        field_decl_list.push_back(_field_decl);
    }
    void Add_Field_Declaration(AST_Field_Declaration *_field_decl)
    {
        field_decl_list.push_back(_field_decl);
    }

    std::vector<AST_Field_Declaration *> field_decl_list;
};

/**
 * @brief
 * field_decl->
    name_list SYM_COLON type_decl SYM_SEMICOLON
 */
class AST_Field_Declaration : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
```

```
    AST_Field_Declaration(AST_Name_List *_name_list, AST_Type_Declaration *_type_decl) : name_list(_name_list), type_decl(_type_decl){};

    AST_Name_List *name_list;
    AST_Type_Declaration *type_decl;
};

/**
 * name_list->
    {IDENTIFIER}
*/
class AST_Name_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Name_List() = default;
    void Add_Identifier(std::string id)
    {
        identifier_list.push_back(id);
    }

    std::vector<std::string> identifier_list;
};
```

3.2.6 AST_Value类及其相关类

AST_Value类及其相关类种类如下：

```
#pragma once
#include "AST_BaseNode.hpp"
#include "AST_Type.hpp"
#include "AST_Expression.hpp"

class AST_Expression;
class AST_Name_List;
class AST_Type_Declaration;

class AST_Const_Part;
class AST_Const_Expression_List;
class AST_Const_Expression;
class AST_Const_Value;

class AST_Variable_Part;
class AST_Variable_Declaration_List;
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：

```
/**
 * @brief
 * const_value:
    LITERAL_INT
    | LITERAL_FLOAT
    | LITERAL_CHAR
    | LITERAL_STR
    | LITERAL_FALSE
    | LITERAL_TRUE
 */
class AST_Const_Value : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    enum class Value_Type
    {
        INT,
        FLOAT,
        CHAR,
        STRING,
        FALSE,
        TRUE
    };
    AST_Const_Value(std::string _content, enum Value_Type _value_type) :
content(_content), value_type(_value_type){};

    std::string Get_Value_Type_Name(Value_Type value_type)
    {
        std::string name;
        switch (value_type)
        {
        case AST_Const_Value::Value_Type::INT:
            name = "INT";
            break;
        case AST_Const_Value::Value_Type::FLOAT:
            name = "FLOAT";
            break;
        case AST_Const_Value::Value_Type::CHAR:
            name = "CHAR";
            break;
        case AST_Const_Value::Value_Type::STRING:
            name = "STRING";
            break;
        case AST_Const_Value::Value_Type::FALSE:
```

```
            name = "FALSE";
            break;
        case AST_Const_Value::Value_Type::TRUE:
            name = "TRUE";
            break;
        }
        return name;
    }

    AST_Const_Value::Value_Type value_type;
    std::string content;
};

/*
const_expr_list->
    {const_expr}
*/
class AST_Const_Expression_List : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Const_Expression_List() = default;
    void Add_Const_Expression(AST_Const_Expression *const_expr)
    {
        const_expr_list.push_back(const_expr);
    }

    std::vector<AST_Const_Expression *> const_expr_list;
};

class AST_Const_Expression : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
    AST_Const_Expression(std::string _id, AST_Expression *_value) :
id(_id), value(_value){};

    std::string id;
    AST_Expression *value;
};

class AST_Const_Part : public AST_BaseNode
{
public:
    std::shared_ptr<Custom_Result>CodeGenerate() override;

public:
```

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

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

```
%union{
    int token_type;
    char* str;

    //AST_Value.hpp
    AST_Const_Part* ast_const_part;
    AST_Const_Expression_List* ast_const_expression_list;
    AST_Const_Expression* ast_const_expression;
    AST_Const_Value* ast_const_value;
    AST_Variable_Part* ast_variable_part;
    AST_Variable_Declaration_List* ast_variable_declaration_list;
    AST_Variable_Declaration* ast_variable_declaration;
    //AST_Type.hpp
    AST_Type_Part* ast_type_part;
    AST_Type_Definition* ast_type_definition;
    AST_Type_Declaration_List* ast_type_declaration_list;
    AST_Type* ast_type;
    AST_Type_Declaration* ast_type_declaration;
    AST_Simple_Type_Declaration* ast_simple_type_declaration;
    AST_Array_Type_Declaration* ast_array_type_declaration;
    AST_Record_Type_Declaration* ast_record_type_declaration;
    AST_Field_Declaration_List* ast_field_declaration_list;
    AST_Field_Declaration* ast_field_declaration;
    AST_Name_List* ast_name_list;
    //AST_Expression.hpp
    AST_Expression* ast_expression;
    AST_Expression_List* ast_expression_list;
    //AST_Program.hpp
    AST_Program* ast_program;
    AST_Program_Head* ast_program_head;
    AST_Routine* ast_routine;
    AST_Routine_Head* ast_routine_head;
    // AST_Declaration_BaseClass* ast_declaration_baseclass;
    AST_Routine_Part* ast_routine_part;
    AST_Routine_Body* ast_routine_body;
    AST_Function_Declaration* ast_function_declaration;
    AST_Function_Head* ast_function_head;
    AST_Procedure_Declaration* ast_procedure_declaration;
    AST_Procedure_Head* ast_procedure_head;
    AST_Parameters* ast_parameters;
    AST_Parameters_Declaration_List* ast_parameters_declaration_list;
    AST_Parameters_Type_List* ast_parameters_type_list;
    AST_Variable_Parameters_List* ast_variable_parameters_list;
    //AST_Statement.hpp
    AST_Compound_Statement* ast_compound_statement;
    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_DOWNTO 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 KEY_USES
    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。

3.4 AST可视化

为了测试词汇分析和语法分析的正确性，以及更直观地查看程序结构，我们实现了AST节点的可视化功能。

为了充分利用Graphviz（可视化工具包），我们创建了一个名为GraphViz的类来直接生成Graphviz源代码，在这个类中我们维护了一个包含Graphviz中的节点和边的数据结构。以下是GraphViz中的私有变量。

```
class GraphViz
{
private:
    int id_cnt = 0;
    std::stack<int> stk;
    std::vector<std::string> nodes;
    std::vector<std::string> edges;
}
```

- stk是一个维护AST节点层次结构的堆栈。
- nodes和edges分别负责存储最终可视化结果的点与边。

为了便于维护GraphViz的节点信息，我们设计了以下几个方法：

```
void AddNode(std::string label, int line, int column)
{
    int id = id_cnt++;
    std::stringstream ostr("");
    ostr << "node_" << id << "[";
    ostr << "label=\"" << label << "\\n";
    ostr << "[" << line << ", " << column << "]";
    ostr << "\"];";
    std::string node_str = ostr.str();
    nodes.push_back(node_str);
    std::stringstream ostr1("");
    ostr1.clear();
    ostr1 << "node_" << stk.top() << "->"
        << "node_" << id << ";";
    std::string edge_str = ostr1.str();
    edges.push_back(edge_str);
    stk.push(id);
}

void AddIdentifier(std::string content)
{
    int id = id_cnt++;
    std::stringstream ostr("");
    ostr << "node_" << id << "[";
    ostr << "label=\"<ID>" << content << "\"";
```

```
    ostr << "];";
    std::string node_str = ostr.str();
    nodes.push_back(node_str);
    std::stringstream ostr1("");
    ostr1.clear();
    ostr1 << "node_" << stk.top() << "->"
        << "node_" << id << ";";
    std::string edge_str = ostr1.str();
    edges.push_back(edge_str);
}

void AddValue(std::string t, std::string content)
{
    int id = id_cnt++;
    std::stringstream ostr("");
    ostr << "node_" << id << "[";
    ostr << "label=\"<" << t << ">" << content << "\"";
    ostr << "];";
    std::string node_str = ostr.str();
    nodes.push_back(node_str);
    std::stringstream ostr1("");
    ostr1 << "node_" << stk.top() << "->"
        << "node_" << id << ";";
    std::string edge_str = ostr1.str();
    edges.push_back(edge_str);
}
```

之后我们在3.2章中介绍的所有AST节点中实现了一个PrintNode方法，分别对于每一类节点进行可视化。例如对于AST_Program，其PrintNode方法如下所示：

```
void AST_Program::PrintNode(GraphViz *g)
{
    g->AddNode("program", GetRow(), GetColumn());
    program_head->PrintNode(g);
    routine->PrintNode(g);
    g->Pop();
}
```

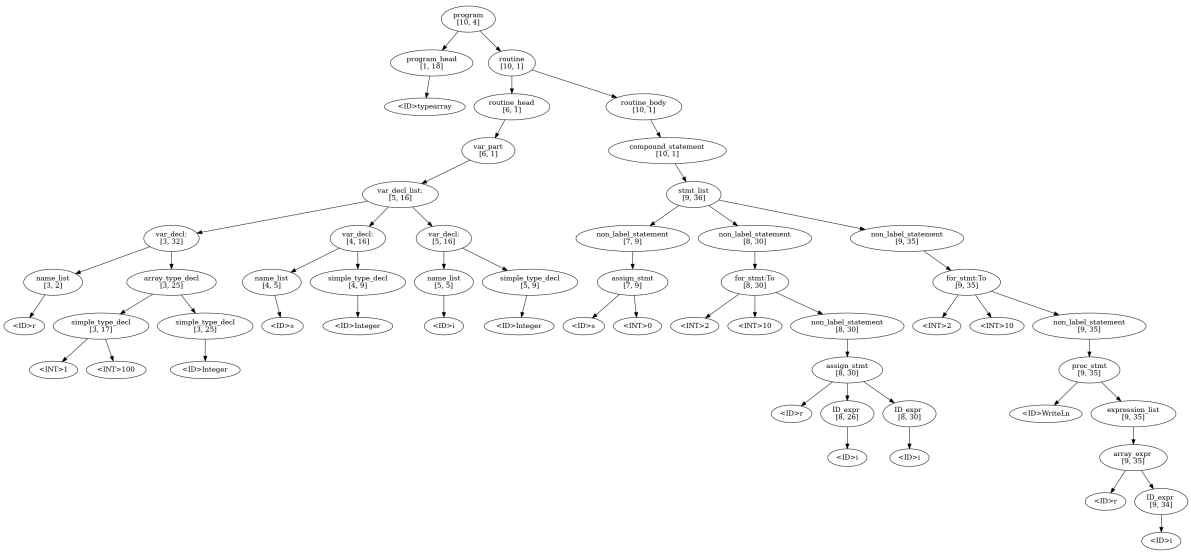


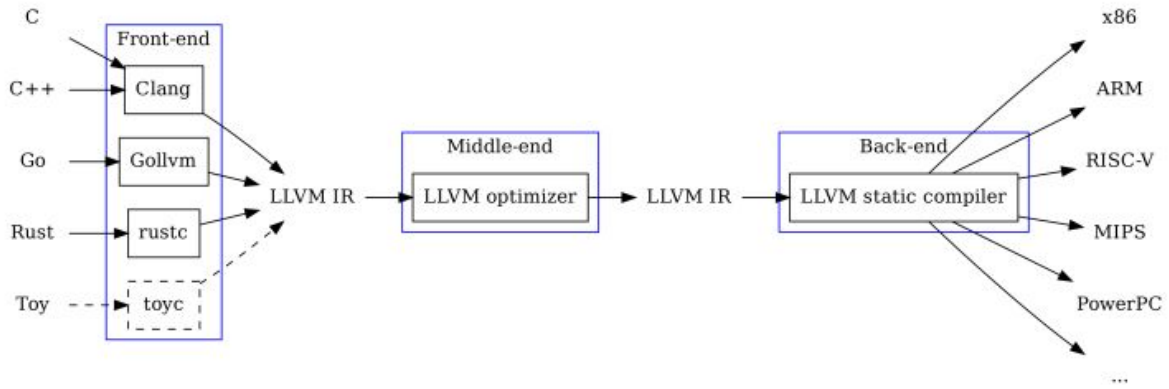
图3.4 for语句的可视化

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 languange，是一个像**RISC**的指令集。

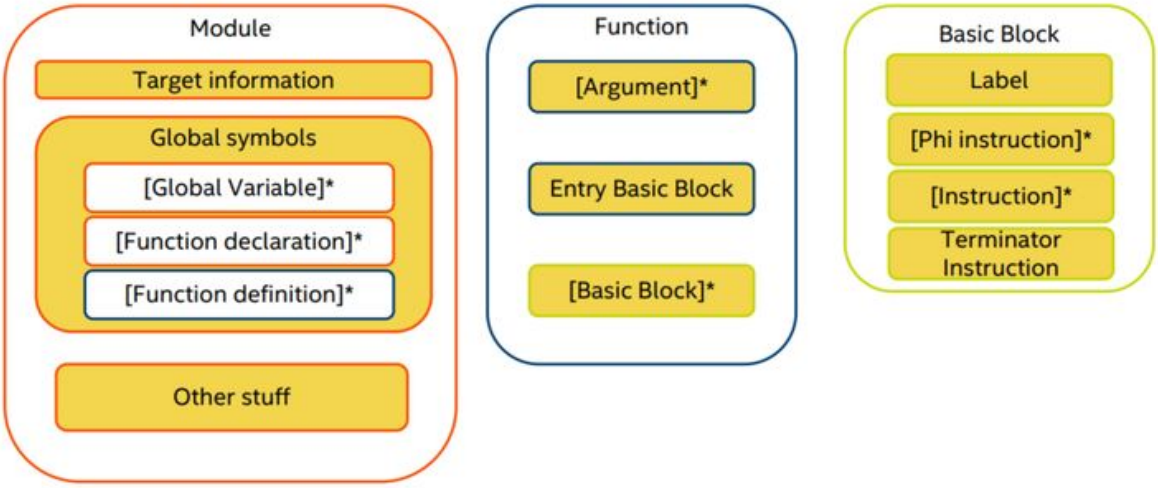
LLVM IR 是 LLVM的核心所在，通过将不同高级语言的前端变换成LLVM IR进行优化、链接后再传给不同目标的后端转换成为二进制代码，前端、优化、后端三个阶段互相解耦，这种模块化的设计使得LLVM优化不依赖于任何源码和目标机器。

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。

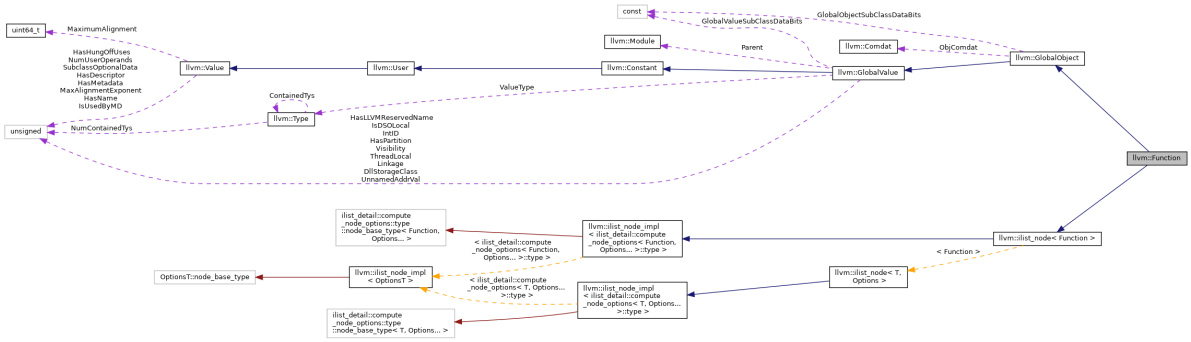
每一个基本块Basic Block包含了标签和各种指令的集合，标签作为指令的索引用于实现指令间的跳转，指令包含Phi指令、一般指令以及终止指令等。



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：

```
llvm::LLVMContext context;
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。

```
std::unique_ptr<llvm::Module> module = std::make_unique<llvm::Module>
("pascal_module", context);
std::map<std::string, llvm::Constant *> names_2_constants; // global
constants
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。

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

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。

```
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。

```
enum class Type_Group
{
    BUILT_IN,
    ENUMERATE,
    SUBRANGE,
    ARRAY,
    STRING,
    RECORD
};
```

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。

```
class Buildin_Type : public Pascal_Type
{
    public:
    enum class Buildin_Type_Name
    {
        INT,
        FLOAT,
        BOOLEAN,
        CHAR,
        VOID
    };

    Buildin_Type_Name buildin_type_name;

    Buildin_Type (Buildin_Type_Name _buildin_type_name) ;
};
const Buildin_Type INT_TYPE_INST(Buildin_Type::Buildin_Type_Name::INT);
const Buildin_Type REAL_TYPE_INST(Buildin_Type::Buildin_Type_Name::FLOAT);
const Buildin_Type CHAR_TYPE_INST(Buildin_Type::Buildin_Type_Name::CHAR);
const Buildin_Type
BOOLEAN_TYPE_INST(Buildin_Type::Buildin_Type_Name::BOOLEAN);
const Buildin_Type VOID_TYPE_INST(Buildin_Type::Buildin_Type_Name::VOID);
Pascal_Type *const INT_TYPE = (Pascal_Type *)(&INT_TYPE_INST);
Pascal_Type *const REAL_TYPE = (Pascal_Type *)(&REAL_TYPE_INST);
Pascal_Type *const CHAR_TYPE = (Pascal_Type *)(&CHAR_TYPE_INST);
Pascal_Type *const BOOLEAN_TYPE = (Pascal_Type *)(&BOOLEAN_TYPE_INST);
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llvm::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。

```
class Enumerate_Type : public Pascal_Type
{
    std::vector<std::string> enum_list;
    Enumerate_Type(std::vector<std::string> _enum_list) ;
};

class Subrange_Type : public Pascal_Type
{
    std::pair<int, int> begin_2_end;

    Subrange_Type(int _begin, int _end);
};

class Array_Type : public Pascal_Type
{
    Subrange_Type subrange;
    Pascal_Type *element_type;

    Array_Type(Subrange_Type _subrange, Pascal_Type *_element_type);

    llvm::ConstantInt *GetLLVMLow;

    llvm::ConstantInt *GetLLVMHigh;
};

class String_Type : public Pascal_Type
{
    int len; // string array

    String_Type(int _len = 256);
};

class Record_Type : public Pascal_Type
{
    std::vector<std::string> name_list;
    std::vector<Pascal_Type *> type_list;

    Record_Type(std::vector<std::string> _name_list,
std::vector<Pascal_Type *> _type_list);
};
```

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。

```
//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）
class FuncSign
{
public:
    FuncSign(int n_local,std::vector<std::string>
_name_list,std::vector<Pascal_Type*> _type_list,std::vector<bool>
_is_var,Pascal_Type * _return_type);

private:
    int num_local_variables; // the number of local var
    std::vector<Pascal_Type *> type_list;
    std::vector<std::string> name_list;
    std::vector<bool> is_var;
    Pascal_Type *return_type;
};
```

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、CleateFunc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LLVMcontext。

```
llvm::Type *GetLLVMType(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。

```
if (p_type->type_group == Pascal_Type::Type_Group::BUILT_IN)
{
    if (isEqual(p_type, INT_TYPE))
        return llvm::Type::getInt32Ty(context);
    else if (isEqual(p_type, REAL_TYPE))
        return llvm::Type::getDoubleTy(context);
    else if (isEqual(p_type, CHAR_TYPE))
        return llvm::Type::getInt8Ty(context);
    else if (isEqual(p_type, BOOLEAN_TYPE))
```

```
        return llvm::Type::getInt1Ty(context);
    else if (isEqual(p_type, VOID_TYPE))
        return llvm::Type::getVoidTy(context);
    else
        return nullptr;
}
```

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。

```
else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::STRING)
{
    Our_Type::String_Type *str = (Our_Type::String_Type *)p_type;
    return llvm::ArrayType::get(GetLLVMType(context, CHAR_TYPE),
(uint64_t)(str->len));
}
else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::ARRAY) {
    Our_Type::Array_Type *array = (Our_Type::Array_Type *) p_type;
    llvm::ArrayType *ret = nullptr;
    int len = array->subrange.begin_2_end.second - array-
>subrange.begin_2_end.first + 1;
    std::cout << "creating array of length " <<len << std::endl;

    ret = llvm::ArrayType::get(GetLLVMType(context, array->element_type),
(uint64_t) len);
    return ret;
}
```

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。

```
else if (p_type->type_group == Our_Type::Pascal_Type::Type_Group::RECORD)
{
    Our_Type::Record_Type *record = (Our_Type::Record_Type *)p_type;
    std::vector<llvm::Type *> llvm_type_vec;
    for (auto t : record->type_list)
    {
        llvm_type_vec.push_back(GetLLVMType(context, t));
    }
    return llvm::StructType::get(context, llvm_type_vec);
}
```

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。

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

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：

```
std::shared_ptr<Custom_Result> AST_Identifier_Expression::CodeGenerate()
{
    //判断是否在当前code_block
    if(Contents::GetCurrentBlock()->isValue(this->id)){
        llvm::Type * my_type
=Our_Type::GetLLVMType(Contents::context,Contents::GetVarType(this->id));
        llvm::Value *mem = Contents::GetCurrentBlock()-
>names_2_values[this->id];
        llvm::Value *value =
Contents::builder.CreateLoad(/*type=*/my_type,mem,"load identifier
value");
        return std::make_shared<Value_Result>(Contents::GetVarType(this-
>id), value, mem);
    }
    //再判断是否在最外层code_block
    else if(Contents::codeblock_list[0]->isValue(this->id)){
        llvm::Type * my_type
=Our_Type::GetLLVMType(Contents::context,Contents::codeblock_list[0]-
>names_2_ourtype[this->id]);
        llvm::Value *mem = Contents::codeblock_list[0]-
>names_2_values[this->id];
        llvm::Value *value =
Contents::builder.CreateLoad(/*type=*/my_type,mem,"load identifier
value");
        return std::make_shared<Value_Result>(Contents::codeblock_list[0]-
>names_2_ourtype[this->id], value, mem);
    }
    else{
        //无参数的函数调用
        AST_Function_Call *func_call = new AST_Function_Call(this->id,
nullptr);
        auto ret = func_call->CodeGenerate();
        if (ret != nullptr) return ret;
        Record_and_Output_Error(true,this->id + " is neither a variable
nor a no-arg function. Cannot get named value: ",this->GetLocation());
        return nullptr;
    }
}
```

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：

```
std::shared_ptr<Custom_Result> AST_Array_Expression::CodeGenerate()
{
    auto index = std::static_pointer_cast<Value_Result> (expression->CodeGenerate());
    // 这里直接从变量表里面找
    Pascal_Type* _array_type  = Contents::GetVarType(this->id);
    auto array =  (Contents::GetCurrentBlock()->isValue(this->id))?
                    std::make_shared<Value_Result>(_array_type,nullptr,Contents::GetCurrentBlock()->names_2_values[this->id]):
                    std::make_shared<Value_Result>(_array_type,nullptr,Contents::codeblock_list[0]->names_2_values[this->id]);
    //判断是否在当前code_block
    //再判断是否在最外层code_block

    bool isStr = array->GetType()->type_group == Pascal_Type::Type_Group::STRING;
    bool isArr = array->GetType()->type_group == Pascal_Type::Type_Group::ARRAY;
    if (array == nullptr || (!isArr && !isStr)) {
        std::cout << "array exp fail" << std::endl;
        return nullptr;
    }

    Array_Type* array_type = (Array_Type*)(array->GetType());
    if (!isEqual(index->GetType(), Our_Type::INT_TYPE) && !isEqual(index->GetType(), Our_Type::CHAR_TYPE))
        return nullptr;

    llvm::Value *base;  //获取数组的基址
    if (isArr) {
        base = array_type->GetLLVMLow(Contents::context);
    } else {
        base = llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context), 0, true);
    }

    //考虑不是从0开始编号
    auto offset = Contents::builder.CreateSub(index->GetValue(),base,"subtmp");
    std::vector<llvm::Value *> offset_vec = {

     llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context),0),
        offset
    };
```

```
36
37      if (isArr) {
38          llvm::Type *my_type =
    Our_Type::GetLLVMType(Contents::context,array_type->element_type);
39          llvm::Value *tmp_mem =
    Contents::builder.CreateGEP(/*type=my_type,*/array->GetMemory(),
    offset_vec, "ArrayCall");
40          llvm::Value *value =
    Contents::builder.CreateLoad(/*type=*/my_type,tmp_mem,"load array
    value");  //直接从name2_to_value中取
41          return std::make_shared<Value_Result>(array_type->element_type,
    value, tmp_mem);
42      } else {
43          llvm::Type *my_type =
    Our_Type::GetLLVMType(Contents::context,Our_Type::CHAR_TYPE);
44          llvm::Value *tmp_mem =
    Contents::builder.CreateGEP(/*type=my_type,*/array->GetMemory(),
    offset_vec, "ArrayCall");
45          llvm::Value *value =
    Contents::builder.CreateLoad(/*type=*/my_type,tmp_mem,"load string
    value");
46          return std::make_shared<Value_Result>(Our_Type::CHAR_TYPE, value,
    tmp_mem);
47      }
48  }
```

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>
    (_record_type,nullptr,Contents::GetCurrentBlock()->names_2_values[this-
    >id]):
7                       std::make_shared<Value_Result>
    (_record_type,nullptr,Contents::codeblock_list[0]->names_2_values[this-
    >id]);
8       //判断是否在当前code_block
9       //再判断是否在最外层code_block
10
11      if(!_record_type->isRecord()){
```

```
        //report error
        return nullptr;
    }
    auto record_type = (Record_Type*)_record_type;
    auto name_list = record_type->name_list;
    auto type_list = record_type->type_list;
    int bias = -1;
    for(int i=0;i<name_list.size();i++){
        if(name_list[i] == this->prop_id){
            bias = i;
            break;
        }
    }
    if(bias == -1){
        //report error
        return nullptr;
    }
    std::vector<llvm::Value *> gep_vec =
{llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context),0,true),

(llvm::ConstantInt::get(llvm::Type::getInt32Ty(Contents::context),bias,tru
e))};

    llvm::Type * my_type =
Our_Type::GetLLVMType(Contents::context,type_list[bias]);
    llvm::Value * mem = Contents::builder.CreateGEP(/*type =
my_type,*/val->GetMemory(),gep_vec,"record_field");
    llvm::Value * ret = Contents::builder.CreateLoad(/*type =
*/my_type,mem,"load record type");
    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，CMPLE，CMPGT，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）：

```
std::shared_ptr<Custom_Result> AST_Binary_Expression::CodeGenerate()
{
    auto l = std::static_pointer_cast<Value_Result>(left_expression-
>CodeGenerate());
    auto r = std::static_pointer_cast<Value_Result>(right_expression-
>CodeGenerate());
    if (l == nullptr || r == nullptr)
        return nullptr;

    // semantic check

    bool is_real = isEqual(l->GetType(), REAL_TYPE);
    if (my_operation == Operation::REALDIV)
```

```
12          is_real = true;
13      auto L = l->GetValue(), R = r->GetValue();
14      if (is_real)
15      {
16          L = Contents::builder.CreateUIToFP(L,
    GetLLVMType(Contents::context, REAL_TYPE));
17          R = Contents::builder.CreateUIToFP(R,
    GetLLVMType(Contents::context, REAL_TYPE));
18      }
19      switch (my_operation)
20      {
21          case Operation::GE:
22              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpOGE(L, R, "cmpftmp")
23                                                  :
    Contents::builder.CreateICmpSGE(L, R, "cmptmp"));
24          case Operation::GT:
25              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpOGT(L, R, "cmpftmp")
26                                                  :
    Contents::builder.CreateICmpSGT(L, R, "cmptmp"));
27          case Operation::LE:
28              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpOLE(L, R, "cmpftmp")
29                                                  :
    Contents::builder.CreateICmpSLE(L, R, "cmptmp"));
30          case Operation::LT:
31              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpOLT(L, R, "cmpftmp")
32                                                  :
    Contents::builder.CreateICmpSLT(L, R, "cmptmp"));
33          case Operation::EQUAL:
34              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpOEQ(L, R, "cmpftmp")
35                                                  :
    Contents::builder.CreateICmpEQ(L, R, "cmptmp"));
36          case Operation::UNEQUAL:
37              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFCmpONE(L, R, "cmpftmp")
38                                                  :
    Contents::builder.CreateICmpNE(L, R, "cmptmp"));
39          case Operation::PLUS:
40              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFAdd(L, R, "addftmp")
41                                                  :
    Contents::builder.CreateAdd(L, R, "addtmp"));
42          case Operation::MINUS:
43              return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
    Contents::builder.CreateFSub(L, R, "subftmp")
44                                                  :
    Contents::builder.CreateSub(L, R, "subtmp"));
45          case Operation::MUL:
```

```
            return std::make_shared<Value_Result>(BOOLEAN_TYPE, is_real ?
Contents::builder.CreateFMul(L, R, "mulftmp")
                                                  :
Contents::builder.CreateMul(L, R, "multmp"));
        case Operation::DIV:
            if (is_real)
            {
                // 报告错误
            }
            return std::make_shared<Value_Result>(INT_TYPE,
Contents::builder.CreateSDiv(L, R, "divtmp"));
        case Operation::MOD:
            if (is_real)
            {
                // 报告错误
            }
            return std::make_shared<Value_Result>(INT_TYPE,
Contents::builder.CreateSRem(L, R, "modtmp"));
        case Operation::REALDIV:
            return std::make_shared<Value_Result>(REAL_TYPE,
Contents::builder.CreateFDiv(L, R, "divftmp"));
        case Operation::OR:
            return std::make_shared<Value_Result>(BOOLEAN_TYPE,
Contents::builder.CreateOr(L, R, "ortmp"));
        case Operation::AND:
            return std::make_shared<Value_Result>(BOOLEAN_TYPE,
Contents::builder.CreateAnd(L, R, "andtmp"));

        default:
            return nullptr;
    }
}
```

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）。

```
std::shared_ptr<Custom_Result> AST_Assign_Statement::CodeGenerate()
{

    if (isDirectAssign())
    {
        llvm::Value *left_mem = Contents::GetCurrentBlock()-
>names_2_values[identifier1];
        // Contents::codeblock_list[0]

        auto right = std::static_pointer_cast<Value_Result>(expression1-
>CodeGenerate());
        Contents::builder.CreateStore(right->GetValue(), left_mem);
    }
```

```
    else if (isArrayAssign())
    {
        // llvm::Value* left_mem = Contents::GetCurrentBlock()->names_2_values[identifier1];
        // auto expidx = std::static_pointer_cast<Value_Result>(expression1->CodeGenerate());
        auto ast_array_expression = std::make_shared<AST_Array_Expression>(this->identifier1, this->expression1);
        std::shared_ptr<Value_Result> left_array_ret = std::static_pointer_cast<Value_Result>(ast_array_expression->CodeGenerate());
        auto right = std::static_pointer_cast<Value_Result>(expression2->CodeGenerate());

        Contents::builder.CreateStore(right->GetValue(), left_array_ret->GetMemory());
    }
    else if (isRecordAttrAssign())
    {
        //• IDENTIFIER '.' IDENTIFIER ':=' expression
        // llvm::Value* left_mem = Contents::GetCurrentBlock()->names_2_values[identifier1];
        // auto expidx = std::static_pointer_cast<Value_Result>(expression1->CodeGenerate());
        auto ast_record_expression = std::make_shared<AST_Property_Expression>(this->identifier1, this->identifier2);
        std::shared_ptr<Value_Result> left_array_ret = std::static_pointer_cast<Value_Result>(ast_record_expression->CodeGenerate());
        auto right = std::static_pointer_cast<Value_Result>(expression1->CodeGenerate());

        Contents::builder.CreateStore(right->GetValue(), left_array_ret->GetMemory());
    }
    return nullptr;
}
```

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。

```
std::shared_ptr<Custom_Result> AST_Program::CodeGenerate()
{
    this->program_head->CodeGenerate();
    Contents::codeblock_list.push_back(new CodeBlock());
    //下面定义主函数:  int main(void);
    llvm::FunctionType * function_type  = llvm::FunctionType::get(GetLLVMType(Contents::context,INT_TYPE),false);
    llvm::Function * main_function = llvm::Function::Create(
```

```
        /*函数类型:包括返回值和函数参数*/function_type,
        /*ExternalLink表示能够模块外使用
*/llvm::Function::ExternalLinkage,
        /*函数名*/"main",
        &*(Contents::module));
    llvm::BasicBlock *entry =
llvm::BasicBlock::Create(Contents::context,"entry",main_function);
    Contents::builder.SetInsertPoint(entry);
    //创建返回值,正常返回0
   this->routine->CodeGenerate();
Contents::builder.CreateRet(llvm::ConstantInt::get(llvm::Type::getInt32Ty(
Contents::context),0,true));
    return nullptr;
}
```

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，该结点只需要依此调用子结点的CodeGenerate方法即可。

```
std::shared_ptr<Custom_Result> AST_Routine::CodeGenerate()
{
    this->routine_head->CodeGenerate();
    this->routine_body->CodeGenerate();
    return nullptr;
}
std::shared_ptr<Custom_Result> AST_Routine_Head::CodeGenerate()
{
    if(this->const_part) this->const_part->CodeGenerate();
    if(this->type_part) this->type_part->CodeGenerate();
    if(this->var_part) this->var_part->CodeGenerate();
    if(this->routine_part) this->routine_part->CodeGenerate();
    return nullptr;
}
std::shared_ptr<Custom_Result> AST_Routine_Body::CodeGenerate()
{
    return this->Get_Compound_Statement()->CodeGenerate();
}
```

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。

```
//全局常量
std::shared_ptr<Custom_Result> AST_Const_Expression::CodeGenerate()
{
    std::shared_ptr<Value_Result> res =
std::static_pointer_cast<Value_Result>(this->value->CodeGenerate());
    llvm::GlobalVariable *constant = new llvm::GlobalVariable(
```

```
            /*Module=*/*(Contents::module),
            /*Type=*/GetLLVMType(Contents::context, res->GetType()),
            /*isConstant=*/true,
            /*Linkage=*/llvm::GlobalValue::PrivateLinkage,
            /*Initializer=*/(llvm::Constant *) res->GetValue(), // has initializer, specified below
            /*Name=*/this->id );
    if (Contents::codeblock_list.back()->names_2_values.count(this->id) || Contents::names_2_constants.count(this->id)) {
        //error
    }
    Contents::names_2_constants[this->id] = (llvm::Constant *) (res->GetValue());
    Contents::codeblock_list[0]->names_2_values[this->id] = constant;
    return nullptr;
}
```

```
//全局变量
std::shared_ptr<Custom_Result> AST_Variable_Declaration::CodeGenerate()
{
    auto name_list = std::static_pointer_cast<Name_List>(this->name_list->CodeGenerate());
    auto type_decl = std::static_pointer_cast<Type_Result>(this->type_decl->CodeGenerate());

    if(type_decl == nullptr){
        return nullptr;    //error
    }
    for(auto identifier : name_list->GetNameList()){
        llvm::Type *ty = GetLLVMType(Contents::context,type_decl->GetType());
        if(Contents::codeblock_list.size() == 1){
            //是全局变量
            llvm::Constant * init_const;
            if(type_decl->GetType()->isSimple()){
                init_const = llvm::Constant::getNullValue(ty);
            }else{
                init_const = llvm::ConstantAggregateZero::get(ty);
            }
            llvm::GlobalVariable * var = new llvm::GlobalVariable(
                /*Module=*/*(Contents::module),
                /*Type=*/ty,
                /*isConstant=*/false,
                /*Linkage=*/llvm::GlobalValue::CommonLinkage,
                /*Initializer=*/init_const, // has initializer, specified below
                /*Name=*/identifier);
            if(Contents::codeblock_list.back()->names_2_values.count(identifier)){
                #ifdef GEN_DEBUG
                std::cout << "变量重复定义" <<std::endl;
```

```
                #endif
            }
            Contents::codeblock_list.back()->names_2_values[identifier] =
var;
            Contents::codeblock_list.back()->names_2_ourtype[identifier] =
type_decl->GetType();
        }else{
            //是局部变量
            llvm::AllocaInst *var = Contents::builder.CreateAlloca(
                ty,
                nullptr,
                identifier
            );
            if(Contents::codeblock_list.back()-
>names_2_values.count(identifier)){
                //error
            }
            Contents::codeblock_list.back()->names_2_values[identifier] =
var;
            Contents::codeblock_list.back()->names_2_ourtype[identifier] =
type_decl->GetType();
        }
    }
    return nullptr;
}
```

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。

```
std::shared_ptr<Custom_Result> AST_Declaration_BaseClass::CodeGenerate()
{
```

```
    bool is_function = (this->declaration_type == ENUM_Declaration_Type::FUNCTION_DECLARATION);
    auto parameters = std::static_pointer_cast<Type_List_Result>(
        is_function ? this->Get_Function_Declaration()->Get_Function_Head()->Get_Parameters()->CodeGenerate()
                    : this->Get_Procedure_Declaration()->Get_Procedure_Head()->Get_Parameters()->CodeGenerate()
    );

    if(parameters == nullptr){
        Record_and_Output_Error(true,"Can not recognize the parameters for function/procedure definition.",this->GetLocation());
        return nullptr;
    }

    Pascal_Type * return_type = Our_Type::VOID_TYPE;
    std::string function_name;
    if(is_function){
        //有返回值
        function_name = this->Get_Function_Declaration()->Get_Function_Head()->Get_Identifier();
        auto return_type_result = std::static_pointer_cast<Type_Result>(this->Get_Function_Declaration()->Get_Function_Head()->Get_Simple_Type_Declaration()->CodeGenerate());
        if(return_type_result == nullptr){
            Record_and_Output_Error(true,"Can not recognize the return type for the function definition.",this->GetLocation());
            return nullptr;
        }
        return_type = return_type_result->GetType();
    }else{
        function_name = this->Get_Procedure_Declaration()->Get_Procedure_Head()->Get_Identifier();
    }
    llvm::Type *llvm_return_type = GetLLVMType(Contents::context,return_type);
    auto name_list = parameters->GetNameList();
    std::vector<std::shared_ptr<Type_Result>> type_var_list = parameters->GetTypeList();
    std::vector<llvm::Type*> llvm_type_list;
    std::vector<Pascal_Type*> type_list;
    std::vector<bool> var_list;

    //检查是否有重名变量
    for(int i=0;i <name_list.size();i++){
        for(int j=i+1;j<name_list.size();j++){
            if(name_list[i] == name_list[j]){
                Record_and_Output_Error(true,"The parameters in the function/procedure definition are duplicated.",this->GetLocation());
                return nullptr;
            }
        }
```

```
43      }
44
45      // 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
    llvm_type_list.push_back(llvm::PointerType::getUnqual(GetLLVMType(Conten
    ts::context,local_type_list[i])));
60      }
61
62      //adding function parameters
63      for(std::shared_ptr<Type_Result> type : type_var_list){
64          type_list.push_back(type->GetType());
65          var_list.push_back(type->GetIsVal());
66
    llvm_type_list.push_back(llvm::PointerType::getUnqual(GetLLVMType(Conten
    ts::context,type->GetType())));
67      }
68
69      FuncSign *funcsign = new
    FuncSign((int)local_name_list.size(),name_list,type_list,var_list,return_
    type);
70      llvm::FunctionType *functiontype = llvm::FunctionType::get(
71          /*返回类型*/llvm_return_type,
72          /*参数类型列表*/llvm_type_list,
73          /*isVar*/false
74      );
75      llvm::Function *function =
    llvm::Function::Create(functiontype,llvm::GlobalVariable::ExternalLinkage
    ,function_name,Contents::module.get());
76
77      Contents::GetCurrentBlock()-
    >Set_Function(function_name,function,funcsign);
78
79      llvm::BasicBlock* oldBlock = Contents::builder.GetInsertBlock();
80      llvm::BasicBlock* basicBlock =
    llvm::BasicBlock::Create(Contents::context,"entry",function,nullptr);
81      Contents::builder.SetInsertPoint(basicBlock);
82
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-
    >arg_end();arg_id++,j++){
91          if(var_list[j]){
92              Contents::GetCurrentBlock()->names_2_values[name_list[j]] =
    (llvm::Value *) arg_id;
93              if(j >= local_name_list.size()){
94                  Contents::GetCurrentBlock()-
    >names_2_ourtype[name_list[j]] = type_list[j];
95              }
96              std::cout << "Inserted var param " << name_list[j] <<
    std::endl;
97          }else{
98              llvm::Type * my_type =
    Our_Type::GetLLVMType(Contents::context,type_list[j]);
99              llvm::Value *value =
    Contents::builder.CreateLoad(/*type=*/my_type,
    (llvm::Value*)arg_id,"load var param");
100             llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
101                 GetLLVMType(Contents::context,type_list[j]),
102                 nullptr,
103                 name_list[j]
104             );
105             Contents::builder.CreateStore(value,mem);
106             Contents::GetCurrentBlock()->names_2_values[name_list[j]] =
    mem;
107             if(j >= local_name_list.size())
108                 Contents::GetCurrentBlock()-
    >names_2_ourtype[name_list[j]] = type_list[j];
109             std::cout << "Inserted val param " << name_list[j] <<
    std::endl;
110         }
111     }
112
113
114     if(is_function){
115         //add function to named_value for itself
116         llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
117                 GetLLVMType(Contents::context,return_type),
118                 nullptr,
119                 function_name
120         );
121         Contents::GetCurrentBlock()->names_2_values[function_name] = mem;
122         Contents::GetCurrentBlock()->names_2_ourtype[function_name] =
    return_type;
123         std::cout << "Inserted val param " << function_name << std::endl;
124
```

```
125         //add return mechanism
126         this->Get_Function_Declaration()->Get_routine()->CodeGenerate();
127         if(Contents::codeblock_list.size() == 1){
128 
 Contents::builder.CreateRet(llvm::ConstantInt::get(llvm::Type::getInt32T
y(Contents::context),0,true));
129         }else {
130             llvm::Type *my_type =
Our_Type::GetLLVMType(Contents::context,return_type);
131             llvm::Value * ret =
Contents::builder.CreateLoad(/*type=*/my_type,Contents::GetCurrentBlock
()->names_2_values[function_name],"load ret param");
132             Contents::builder.CreateRet(ret);
133         }
134     }else{
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-
>args_list->CodeGenerate());
11         value_vec = value_list->GetValueList();
12     }
13 
14     //获取函数名称     this->func_id;
15     using namespace std;
16     for (int i = Contents::codeblock_list.size() - 1; i>= 0; i--){
```

```
17          FuncSign *funcsign = (Contents::codeblock_list[i])->Find_FuncSign(this->func_id);
18          if(funcsign == nullptr) {
19              continue;
20          }
21          // Note the function/procedure can not be overridden in pascal, so the function is matched iff the name is matched.
22          // NameList().size include all local variables that require to be passed
23          // we should compare NameList.size() - n_local
24          // which is the actual arg size
25          if(funcsign->GetNameList().size() - funcsign->GetLocalVariablesNum() != value_vec.size()){
26              Record_and_Output_Error(true,"Can't find function" + this->func_id + ": you have "+std::to_string(value_vec.size()) + "parameters, but the defined one has "
27              +std::to_string(funcsign->GetNameList().size() - funcsign->GetLocalVariablesNum()) + "parameters.",this->GetLocation());
28              return nullptr;
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 the para list
43          int cur;
44          int n_local = funcsign->GetLocalVariablesNum();
45          for(cur = 0; cur < n_local; cur++) {
46              std::string local_name = name_list[cur];
47              if (Contents::GetCurrentBlock()->names_2_values.find(local_name) == Contents::GetCurrentBlock()->names_2_values.end()) {
48
49                  parameters.push_back(nullptr);
50              } else {
51                  parameters.push_back(Contents::GetCurrentBlock()->names_2_values[local_name]);
52              }
53          }
54
55          // 函数传参
```

```
56          for (auto value: value_vec){
57              if (!isEqual(value->GetType(), type_list[cur])){
58                  Record_and_Output_Error(true,"Type does not match on
    function " + this->func_id + " calling.",this->GetLocation());
59                  return nullptr;
60              }
61              if (value->GetMemory() != nullptr) {
62                  parameters.push_back(value->GetMemory());
63              } else {
64                  Contents::temp_variable_count++;    //不重复编码
65                  // here we encounter a literally const value as a
    parameter
66                  // we add a local variable to the IRBuilder
67                  // but do not reflect it in Current_CodeBlock-
    >named_values
68                  // thus we do not add abnormal local variables
    when we declare another function/procedure
69                  llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
70                      GetLLVMType(Contents::context, type_list[cur]),
71                      nullptr,
72                      "0_" + std::to_string(Contents::temp_variable_count)
73                  );
74                  Contents::builder.CreateStore(value->GetValue(), mem);
75                  parameters.push_back(mem);
76              }
77              cur++;
78          }
79          auto ret = Contents::builder.CreateCall(callee, parameters);
80
81          if (funcsign->GetReturnType()->type_group ==
    Our_Type::Pascal_Type::Type_Group::STRING) {
82              // to return a str type for writeln to print
83              // we have to use its pointer
84              // to achieve this, we add a never used variable here
85              Contents::temp_variable_count++;
86
87              llvm::AllocaInst *mem = Contents::builder.CreateAlloca(
88                  GetLLVMType(Contents::context, funcsign-
    >GetReturnType()),
89                  nullptr,
90                  "0_" + this->func_id +
    std::to_string(Contents::temp_variable_count)
91              );
92              Contents::builder.CreateStore(ret, mem);
93              llvm::Type * my_type
    =Our_Type::GetLLVMType(Contents::context,funcsign->GetReturnType());
94              llvm::Value *value =
    Contents::builder.CreateLoad(/*type=*/my_type,mem,"load return value");
95              return std::make_shared<Value_Result>(funcsign-
    >GetReturnType(), value, mem); //, ret->getPointerOperand()); //,
    "call_"+ node->getFuncId()
```

```
96          } else {
97              return std::make_shared<Value_Result>(funcsign-
    >GetReturnType(), ret);
98          }
99      }
100     using namespace Our_Type;
101
102     // To do list : 系统调用
103     // Currently, sys_function will use no local variables that
    has cascade relation
104     // So we do not need to deal with the locals and do it
    simply
105     if (Contents::isSysFunc(this->func_id)) {
106         return std::make_shared<Value_Result>(VOID_TYPE,
    Contents::GenSysFunc(this->func_id, value_vec));
107     }
108     Record_and_Output_Error(true,"Function " + this->func_id + " not
    found.",this->GetLocation());
109     return nullptr;
110 }
```

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write(writeln)。

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
    std::vector<std::shared_ptr<Value_Result>> &args_list)
14 {
15     for (auto &ch : id)
16         ch = tolower(ch);
17     if (id == "write")
18         return GenSysWrite(args_list, false);
19     if (id == "writeln")
20         return GenSysWrite(args_list, true);
21     if (id == "read")
22         return GenSysRead(args_list, false);
23     if (id == "readln")
```

```
        return GenSysRead(args_list, true);
    // report error
    return nullptr;
}
```

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

```
llvm::Value *GenSysWrite(const std::vector<std::shared_ptr<Value_Result>> &args_list, bool new_line)
{
    static llvm::Function *llvm_printf = nullptr;
    if (llvm_printf == nullptr)
    {
        // register printf
        std::vector<llvm::Type *> arg_types = {llvm::Type::getInt8PtrTy(Contents::context)};
        llvm::FunctionType *func_type = llvm::FunctionType::get(
            llvm::Type::getInt32Ty(Contents::context),
            arg_types,
            true);
        llvm::Function *func = llvm::Function::Create(
            func_type,
            llvm::Function::ExternalLinkage,
            "printf",
            &*(Contents::module));
        func->setCallingConv(llvm::CallingConv::C);
        llvm_printf = func;
    }
    std::string format;
    std::vector<llvm::Value *> printf_args;
    printf_args.emplace_back(nullptr);
    for (auto arg : args_list)
    {
        Our_Type::Pascal_Type *tp = arg->GetType();
        if (tp->isIntegerTy())
        {
            format += "%d";
```

```
            printf_args.emplace_back(arg->GetValue());
        }
        else if (tp->isFloatingPointTy())
        {
            format += "%lf";
            printf_args.emplace_back(arg->GetValue());
        }
        else if (tp->isCharTy())
        {
            format += "%c";
            printf_args.emplace_back(arg->GetValue());
        }
        else if (tp->isString())
        {
            format += "%s";
            printf_args.emplace_back(arg->GetMemory());
        }
        else
        {
            std::cerr << "[write/writeln] Unsupported type" << std::endl;
            return nullptr;
        }
        if (new_line)
        {
            format += "\n";
        }
        printf_args[0] = Contents::builder.CreateGlobalStringPtr(format,
"printf_format");
    }
    return Contents::builder.CreateCall(llvm_printf, printf_args,
"call_printf");
}
```

```
llvm::Value *GenSysRead(const std::vector<std::shared_ptr<Value_Result>>
&args_list, bool new_line)
{
    static llvm::Function *llvm_scanf = nullptr;
    if (llvm_scanf == nullptr)
    {
        // register printf
        std::vector<llvm::Type *> arg_types =
{llvm::Type::getInt8PtrTy(Contents::context)};
        llvm::FunctionType *func_type = llvm::FunctionType::get(
            llvm::Type::getInt32Ty(Contents::context),
            arg_types,
            true);
        llvm::Function *func = llvm::Function::Create(
            func_type,
            llvm::Function::ExternalLinkage,
            "scanf",
            &*(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,
    "scanf_format");
53      }
54      llvm::Value *ret = Contents::builder.CreateCall(llvm_scanf,
    scanf_args, "call_scanf");
55      if (new_line)
56      {
57          // consume \n
58          Contents::builder.CreateCall(llvm_scanf,
    Contents::builder.CreateGlobalStringPtr("%*c", "scanf_newline"),
    "call_scanf");
59      }
60      return ret;
61  }
```

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。

```
std::shared_ptr<Custom_Result> AST_If_Statement::CodeGenerate(){
    // std::cout << "hello" << std::endl;
    llvm::Function *function = Contents::builder.GetInsertBlock()->getParent();

    llvm::BasicBlock *then_stmt_block = llvm::BasicBlock::Create(Contents::context, "if_then", function);
    llvm::BasicBlock *else_stmt_block = llvm::BasicBlock::Create(Contents::context, "if_else", function);

    llvm::BasicBlock *continue_stmt_block = llvm::BasicBlock::Create(Contents::context, "if_continue", function); //continue

    auto condition_ret = std::static_pointer_cast<Value_Result>(expression->CodeGenerate());
    //TODO: no expression / expression type not bool error check
    //contiute br
    Contents::builder.CreateCondBr(condition_ret->GetValue(), then_stmt_block, else_stmt_block);
    Contents::builder.SetInsertPoint(then_stmt_block); // then
    this->statement->CodeGenerate(); // do then
    Contents::builder.CreateBr(continue_stmt_block); // unconditional br label instruction
    Contents::builder.SetInsertPoint(else_stmt_block); // else
    this->else_clause->CodeGenerate();
    Contents::builder.CreateBr(continue_stmt_block); //continue out
    Contents::builder.SetInsertPoint(continue_stmt_block); // else

    return nullptr;
}
```

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.

```
std::shared_ptr<Custom_Result> AST_Case_Statement::CodeGenerate(){
    // std::cout << "hello" << std::endl;
```

```
    llvm::Function* function = Contents::builder.GetInsertBlock()->getParent(); // for Create parent function
    auto case_expression_vec = case_expression_list->case_expression_list;
    int case_num = case_expression_vec.size();
    std::vector<llvm::BasicBlock *> condition_block_vec;
    std::vector<llvm::BasicBlock *> handle_block_vec;

    for(int i=0; i<case_num; i++){

     condition_block_vec.push_back(llvm::BasicBlock::Create(Contents::context, "case_condition", function));

     handle_block_vec.push_back(llvm::BasicBlock::Create(Contents::context, "case_handle", function));
    }
    auto endcase_block = llvm::BasicBlock::Create(Contents::context, "case_endcase", function);

    for(int i=0; i<case_num; i++){
        //use binary expression for convenience
        //TODO: reconstruct
        Contents::builder.SetInsertPoint(condition_block_vec[i]);
        // compare
        bool_cmp = new AST_Binary_Expression(AST_Binary_Expression::Operation::EQUAL, this->expression, case_expression_vec[i]->e);
    }
    return nullptr;
}
```

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。

```
std::shared_ptr<Custom_Result> AST_For_Statement::CodeGenerate(){
    llvm::Function * function = Contents::builder.GetInsertBlock()->getParent();
    llvm::BasicBlock* for_start_block = llvm::BasicBlock::Create(Contents::context, "for_start", function);
    llvm::BasicBlock* for_handle_block = llvm::BasicBlock::Create(Contents::context, "for_handle", function);
    llvm::BasicBlock* for_condition_block = llvm::BasicBlock::Create(Contents::context, "for_condition", function);
```

```
6       llvm::BasicBlock* for_end_block =
    llvm::BasicBlock::Create(Contents::context, "for_end", function);
7
8       // TODO: continue
9       Contents::GetCurrentBlock()-
    >loop_return_blocks.push_back(for_end_block);
10
11      //start
12      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>
    (ast_const_value->CodeGenerate());
23
24      // reuse ast code generator
25      std::cout << "begin for assign" << std::endl;
26      auto ast_assign_statement = new AST_Assign_Statement(this->identifier,
    this->expression1);
27      ast_assign_statement->CodeGenerate(); // local varaibla
28
29      std::cout << "begin first cmp" << std::endl;
30      // compare iteration assign
31      auto ast_cmp_statement =  new AST_Binary_Expression(
32                  this->my_direction->isTo() ?
    AST_Binary_Expression::Operation::GT :
33
    AST_Binary_Expression::Operation::LT,
34                  this->expression1, this->expression2
35      );
36      auto ast_cmp_ret = std::static_pointer_cast<Value_Result>
    (ast_cmp_statement->CodeGenerate());
37
38      //branch
39      std::cout << "begin first condition br" << std::endl;
40      Contents::builder.CreateCondBr(ast_cmp_ret->GetValue(), for_end_block,
    for_handle_block);
41
42      // handle
43      std::cout << "begin handle" << std::endl;
44      Contents::builder.SetInsertPoint(for_handle_block);
45      this->statement->CodeGenerate();
46
47      // condition: step -> cmp ? contine(->handle) or end
48      std::cout << "begin condition" << std::endl;
```

```
49      Contents::builder.CreateBr(for_condition_block);
50      Contents::builder.SetInsertPoint(for_condition_block);
51      // step
52      std::cout << "begin step" << std::endl;
53
54      auto left_id = new AST_Identifier_Expression(this->identifier); //
    left value for step
55      auto left_ret = std::static_pointer_cast<Value_Result>(left_id-
    >CodeGenerate());
56
57      std::cout << "begin step add" << std::endl;
58      llvm::Value* add_ret = Contents::builder.CreateAdd(left_ret-
    >GetValue(),const_value_ret->GetValue(),"tmpadd");
59
60      std::cout << "begin step store" << std::endl;
61      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() ?
    AST_Binary_Expression::Operation::GT :
73
    AST_Binary_Expression::Operation::LT,
74                  left_id, this->expression2
75      );
76      ast_cmp_ret = std::static_pointer_cast<Value_Result>
    (ast_cmp_statement->CodeGenerate());
77
78      Contents::builder.CreateCondBr(ast_cmp_ret->GetValue(), for_end_block,
    for_handle_block);
79
80      Contents::builder.CreateBr(for_end_block);
81      Contents::builder.SetInsertPoint(for_end_block);
82      Contents::GetCurrentBlock()->loop_return_blocks.pop_back();
83
84      std::cout << "for success" << std::endl;
85      return nullptr;
86
87  }
```

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：

```
std::shared_ptr<Custom_Result> AST_While_Statement::CodeGenerate()
{
    // 初始化各个BasicBlock
    llvm::Function *func = Contents::builder.GetInsertBlock()->getParent();
    llvm::BasicBlock *conditon_block = llvm::BasicBlock::Create(Contents::context, "while_condition", func);
    llvm::BasicBlock *body_block = llvm::BasicBlock::Create(Contents::context, "while_body", func);
    llvm::BasicBlock *end_block = llvm::BasicBlock::Create(Contents::context, "while_end", func);

    Contents::GetCurrentBlock()->loop_return_blocks.push_back(end_block);

    Contents::builder.CreateBr(conditon_block);
    Contents::builder.SetInsertPoint(conditon_block);

    auto cond_res = std::static_pointer_cast<Value_Result>(this->expression->CodeGenerate());

    if (cond_res == nullptr || !isEqual(cond_res->GetType(), BOOLEAN_TYPE)){
        Record_and_Output_Error(true,"Invalid condition in while statement.",this->GetLocation());
        return nullptr;
    }

    // 条件判断，是否结束循环
    Contents::builder.CreateCondBr(cond_res->GetValue(), body_block, end_block);

    // 在body_block生成需要执行的while主体
    Contents::builder.SetInsertPoint(body_block);
    this->statement->CodeGenerate();

    // 无条件跳转回condition_block 进行结束条件的判断
    Contents::builder.CreateBr(conditon_block);
    Contents::builder.SetInsertPoint(end_block);

    Contents::GetCurrentBlock()->loop_return_blocks.pop_back();

    return nullptr;
}
```

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：

```
std::shared_ptr<Custom_Result> AST_Repeat_Statement::CodeGenerate()
{
    // 初始化各个BasicBlock
    llvm::Function *func = Contents::builder.GetInsertBlock()->getParent();
    llvm::BasicBlock *conditon_block = llvm::BasicBlock::Create(Contents::context, "repeat_condition", func);
    llvm::BasicBlock *body_block = llvm::BasicBlock::Create(Contents::context, "repeat_body", func);
    llvm::BasicBlock *continue_block = llvm::BasicBlock::Create(Contents::context, "repeat_continue", func);

    Contents::GetCurrentBlock()->loop_return_blocks.push_back(continue_block);

    // 进入body主体
    Contents::builder.CreateBr(body_block);

    // 在body_block生成需要执行的while主体
    Contents::builder.SetInsertPoint(body_block);
    this->statement_list->CodeGenerate();

    // 无条件跳转回condition_block 进行结束条件的判断
    Contents::builder.CreateBr(conditon_block);
    Contents::builder.SetInsertPoint(conditon_block);

    auto cond_res = std::static_pointer_cast<Value_Result> (this->expression->CodeGenerate());
    if (cond_res == nullptr || !isEqual(cond_res->GetType(), BOOLEAN_TYPE)){
        Record_and_Output_Error(true,"Invalid condition in repeat statement.",this->GetLocation());
        return nullptr;
    }

    Contents::builder.CreateCondBr(cond_res->GetValue(),continue_block,body_block);
    Contents::builder.SetInsertPoint(continue_block);

    Contents::GetCurrentBlock()->loop_return_blocks.pop_back();
    return nullptr;
}
```

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

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

- 优化后

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

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：

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

在获取Target前，初始化所有目标以发出目标代码：

```
InitializeAllTargetInfos();
InitializeAllTargets();
InitializeAllTargetMCs();
InitializeAllAsmParsers();
InitializeAllAsmPrinters();
```

使用目标三元组获得 Target：

```
std::string Error;
auto Target = TargetRegistry::lookupTarget(TargetTriple, Error);

// Print an error and exit if we couldn't find the requested
target.
// This generally occurs if we've forgotten to initialise the
// TargetRegistry or we have a bogus target triple.
if (!Target) {
  errs() << Error;
  return 1;
}
```

TargetMachine 类提供了我们要定位的机器的完整机器描述：

```
auto CPU = "generic";
auto Features = "";

TargetOptions opt;
auto RM = Optional<Reloc::Model>();
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。

```
auto TheTargetMachine =
       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

```
auto Filename = input_file_name + ".o"; //current directory
if(system(("rm " + Filename).c_str()) == -1){
    std::cout << "error in rm .o" << std::endl;
}
std::error_code EC;
raw_fd_ostream dest(Filename, EC, sys::fs::OF_None);

if (EC)
{
    errs() << "Could not open file: " << EC.message();
    return 1;
}
```

2. 定义一个发出目标代码的过程，然后运行该 pass

```
legacy::PassManager pass;
auto FileType = CGFT_ObjectFile;

if (TheTargetMachine->addPassesToEmitFile(pass, dest, nullptr, FileType))
{
    errs() << "TheTargetMachine can't emit a file of this type";
    return 1;
}

pass.run((*(Contents::module)));
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

```
llvm-as test.ll -o test.bc
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

```
program buildInTypeTest;
const
    ic = 3;
    rc = 3.14;
    bc = false;
    cc = 'b';
var
    iv : integer;
    rv : real;
    bv : boolean;
    cv : char;

begin
    iv := 1;
    rv := 99.9;
    bv := true;
    cv := 'a';
    WriteLn(ic);
    WriteLn(rc);
    WriteLn(bc);
    WriteLn(cc);
    writeln(iv);
```

```
    writeln(rv);
    writeln(bv);
    writeln(cv);
end
.
```

- IR

```
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@ic = private constant i32 3
@rc = private constant double 3.140000e+00
@bc = private constant i1 false
@cc = private constant i8 98
@iv = common global i32 0
@rv = common global double 0.000000e+00
@bv = common global i1 false
@cv = common global i8 0
@printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.1 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1
@printf_format.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.5 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.7 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1

define i32 @main() {
entry:
  store i32 1, i32* @iv, align 4
  store double 9.990000e+01, double* @rv, align 8
  store i1 true, i1* @bv, align 1
  store i8 97, i8* @cv, align 1
  %"load identifier value" = load i32, i32* @ic, align 4
  %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")
  %"load identifier value1" = load double, double* @rc, align 8
  %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")
```

```
  %"load identifier value3" = load i1, i1* @bc, align 1
  %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")
  %"load identifier value5" = load i8, i8* @cc, align 1
  %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")
  %"load identifier value7" = load i32, i32* @iv, align 4
  %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")
  %"load identifier value9" = load double, double* @rv, align 8
  %call_printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @printf_format.5, i32 0, i32 0), double %"load identifier value9")
  %"load identifier value11" = load i1, i1* @bv, align 1
  %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format.6, i32 0, i32 0), i1 %"load identifier value11")
  %"load identifier value13" = load i8, i8* @cv, align 1
  %call_printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format.7, i32 0, i32 0), i8 %"load identifier value13")
  ret i32 0
}

declare i32 @printf(i8*, ...)

```

- 汇编指令

```
    .text
    .file   "pascal_module"
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    subl    $12, %esp
    .cfi_def_cfa_offset 16
    movl    $1, iv
    movl    $1079572889, rv+4               # imm = 0x4058F999
    movl    $-1717986918, rv                # imm = 0x9999999A
    movb    $1, bv
    movb    $97, cv
    movl    .Lic, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format, (%esp)
```

```
    calll   printf@PLT
    fldl    .Lrc
    fstpl   4(%esp)
    movl    $.Lprintf_format.1, (%esp)
    calll   printf@PLT
    movzbl  .Lbc, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format.2, (%esp)
    calll   printf@PLT
    movzbl  .Lcc, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format.3, (%esp)
    calll   printf@PLT
    movl    iv, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format.4, (%esp)
    calll   printf@PLT
    fldl    rv
    fstpl   4(%esp)
    movl    $.Lprintf_format.5, (%esp)
    calll   printf@PLT
    movzbl  bv, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format.6, (%esp)
    calll   printf@PLT
    movzbl  cv, %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format.7, (%esp)
    calll   printf@PLT
    xorl    %eax, %eax
    addl    $12, %esp
    .cfi_def_cfa_offset 4
    retl
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .Lic,@object                    # @ic
    .section    .rodata,"a",@progbits
    .p2align    2
.Lic:
    .long   3                           # 0x3
    .size   .Lic, 4

    .type   .Lrc,@object                    # @rc
    .p2align    3
.Lrc:
    .quad   0x40091eb851eb851f              # double
3.1400000000000001
    .size   .Lrc, 8

```

```
    .type   .Lbc,@object                    # @bc
.Lbc:
    .byte   0                               # 0x0
    .size   .Lbc, 1

    .type   .Lcc,@object                    # @cc
.Lcc:
    .byte   98                              # 0x62
    .size   .Lcc, 1

    .type   iv,@object                      # @iv
    .comm   iv,4,4
    .type   rv,@object                      # @rv
    .comm   rv,8,8
    .type   bv,@object                      # @bv
    .comm   bv,1,1
    .type   cv,@object                      # @cv
    .comm   cv,1,1
    .type   .Lprintf_format,@object         # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%d\n"
    .size   .Lprintf_format, 4

    .type   .Lprintf_format.1,@object       # @printf_format.1
.Lprintf_format.1:
    .asciz  "%lf\n"
    .size   .Lprintf_format.1, 5

    .type   .Lprintf_format.2,@object       # @printf_format.2
.Lprintf_format.2:
    .asciz  "%d\n"
    .size   .Lprintf_format.2, 4

    .type   .Lprintf_format.3,@object       # @printf_format.3
.Lprintf_format.3:
    .asciz  "%c\n"
    .size   .Lprintf_format.3, 4

    .type   .Lprintf_format.4,@object       # @printf_format.4
.Lprintf_format.4:
    .asciz  "%d\n"
    .size   .Lprintf_format.4, 4

    .type   .Lprintf_format.5,@object       # @printf_format.5
.Lprintf_format.5:
    .asciz  "%lf\n"
    .size   .Lprintf_format.5, 5

    .type   .Lprintf_format.6,@object       # @printf_format.6
.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 := l to r do
12     begin
13         a[iv] := iv;
14     end;
15     for iv := l 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
```

```
@iv = common global i32 0
@a = common global [6 x i32] zeroinitializer
@printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  store i32 0, i32* @iv, align 4
  br label %for_start

for_start:                                        ; preds = %entry
  %"load identifier value" = load i32, i32* @l, align 4
  store i32 %"load identifier value", i32* @iv, align 4
  %"load identifier value1" = load i32, i32* @l, align 4
  %"load identifier value2" = load i32, i32* @r, align 4
  %cmptmp = icmp sgt i32 %"load identifier value1", %"load identifier value2"
  br i1 %cmptmp, label %for_end, label %for_handle

for_handle:                                       ; preds = %for_condition, %for_start
  %"load identifier value3" = load i32, i32* @iv, align 4
  %subtmp = sub i32 %"load identifier value3", 1
  %ArrayCall = getelementptr [6 x i32], [6 x i32]* @a, i32 0, i32 %subtmp
  %"load array value" = load i32, i32* %ArrayCall, align 4
  %"load identifier value4" = load i32, i32* @iv, align 4
  store i32 %"load identifier value4", i32* %ArrayCall, align 4
  br label %for_condition

for_condition:                                    ; preds = %for_handle
  %"load identifier value5" = load i32, i32* @iv, align 4
  %tmpadd = add i32 %"load identifier value5", 1
  store i32 %tmpadd, i32* @iv, align 4
  %"load identifier value6" = load i32, i32* @iv, align 4
  %"load identifier value7" = load i32, i32* @r, align 4
  %cmptmp8 = icmp sgt i32 %"load identifier value6", %"load identifier value7"
  br i1 %cmptmp8, label %for_end, label %for_handle
  br label %for_end

for_end:                                          ; preds = %for_condition, %for_condition, %for_start
  br label %for_start9

for_start9:                                       ; preds = %for_end
  %"load identifier value13" = load i32, i32* @l, align 4
  store i32 %"load identifier value13", i32* @iv, align 4
  %"load identifier value14" = load i32, i32* @l, align 4
  %"load identifier value15" = load i32, i32* @r, align 4
```

```
  %cmptmp16 = icmp sgt i32 %"load identifier value14", %"load identifier value15"
  br i1 %cmptmp16, label %for_end12, label %for_handle10

for_handle10:                                     ; preds = %for_condition11, %for_start9
  %"load identifier value17" = load i32, i32* @iv, align 4
  %subtmp18 = sub i32 %"load identifier value17", 1
  %ArrayCall19 = getelementptr [6 x i32], [6 x i32]* @a, i32 0, i32 %subtmp18
  %"load array value20" = load i32, i32* %ArrayCall19, align 4
  %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load array value20")
  br label %for_condition11

for_condition11:                                  ; preds = %for_handle10
  %"load identifier value21" = load i32, i32* @iv, align 4
  %tmpadd22 = add i32 %"load identifier value21", 1
  store i32 %tmpadd22, i32* @iv, align 4
  %"load identifier value23" = load i32, i32* @iv, align 4
  %"load identifier value24" = load i32, i32* @r, align 4
  %cmptmp25 = icmp sgt i32 %"load identifier value23", %"load identifier value24"
  br i1 %cmptmp25, label %for_end12, label %for_handle10
  br label %for_end12

for_end12:                                        ; preds = %for_condition11, %for_condition11, %for_start9
  ret i32 0
}

declare i32 @printf(i8*, ...)

```

- 汇编指令

```
    .text
    .file   "pascal_module"
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    pushl   %esi
    .cfi_def_cfa_offset 8
    subl    $8, %esp
    .cfi_def_cfa_offset 16
    .cfi_offset %esi, -8
```

```
14      movl    .Ll, %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:
```

```
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .Ll,@object                 # @l
    .section    .rodata,"a",@progbits
    .p2align    2
.Ll:
    .long   1                           # 0x1
    .size   .Ll, 4

    .type   .Lr,@object                 # @r
    .p2align    2
.Lr:
    .long   6                           # 0x6
    .size   .Lr, 4

    .type   iv,@object                  # @iv
    .comm   iv,4,4
    .type   a,@object                   # @a
    .comm   a,24,16
    .type   .Lprintf_format,@object     # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%d\n"
    .size   .Lprintf_format, 4

    .section    ".note.GNU-stack","",@progbits

```

- 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

```
program typerecord;
type
  InfoType = record
    MyName : String;
    Age : integer;
    City, State : integer;
  end;
var
  Book1, Book2: InfoType;
```

```pascal
begin
  Book1.MyName := 'AI';
  Book1.Age := 2;
  Book1.City := 4;
  Book1.State := 5;
  Writeln(Book1.MyName,' ',Book1.Age,' ',Book1.City,' ',Book1.State);
  Book2.MyName := 'CS';
  Book2.Age := 3;
  Book2.City := 41;
  Book2.State := 52;
  Writeln(Book2.MyName,' ',Book2.Age,' ',Book2.City,' ',Book2.State);
end.
```

- IR代码

```llvm
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@Book1 = common global { [256 x i8], i32, i32, i32 } zeroinitializer
@Book2 = common global { [256 x i8], i32, i32, i32 } zeroinitializer
@0 = private unnamed_addr constant [256 x i8] c"AI\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00", align 1
@printf_format = private unnamed_addr constant [16 x i8] c"%s%c%d%c%d%c%d\0A\00", align 1
```

```
@1 = private unnamed_addr constant [256 x i8]
c"CS\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00",
align 1
@printf_format.1 = private unnamed_addr constant [16 x i8]
c"%s%c%d%c%d%c%d\0A\00", align 1

define i32 @main() {
entry:
  %"load record type" = load [256 x i8], [256 x i8]* getelementptr
inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }*
@Book1, i32 0, i32 0), align 1
  %"load const string" = load [256 x i8], [256 x i8]* @0, align 1
  store [256 x i8] %"load const string", [256 x i8]* getelementptr
inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }*
@Book1, i32 0, i32 0), align 1
  %"load record type1" = load i32, i32* getelementptr inbounds ({ [256 x
i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32
1), align 4
  store i32 2, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
{ [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 1), align 4
  %"load record type2" = load i32, i32* getelementptr inbounds ({ [256 x
i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32
2), align 4
  store i32 4, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
{ [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 2), align 4
  %"load record type3" = load i32, i32* getelementptr inbounds ({ [256 x
i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32
3), align 4
  store i32 5, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 },
{ [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 3), align 4
  %"load record type4" = load [256 x i8], [256 x i8]* getelementptr
inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }*
@Book1, i32 0, i32 0), align 1
  %"load record type5" = load i32, i32* getelementptr inbounds ({ [256 x
i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32
1), align 4
  %"load record type6" = load i32, i32* getelementptr inbounds ({ [256 x
i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32
2), align 4
```

```
25    %"load record type7" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book1, i32 0, i32 3), align 4
26    %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16 x i8], [16 x i8]* @printf_format, i32 0, i32 0), [256 x i8]* getelementptr 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")
27    %"load record type8" = load [256 x i8], [256 x i8]* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 0), align 1
28    %"load const string9" = load [256 x i8], [256 x i8]* @1, align 1
29    store [256 x i8] %"load const string9", [256 x i8]* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 0), align 1
30    %"load record type10" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1), align 4
31    store i32 3, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1), align 4
32    %"load record type11" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2), align 4
33    store i32 41, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2), align 4
34    %"load record type12" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3), align 4
35    store i32 52, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3), align 4
36    %"load record type13" = load [256 x i8], [256 x i8]* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 0), align 1
37    %"load record type14" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 1), align 4
38    %"load record type15" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 2), align 4
39    %"load record type16" = load i32, i32* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 3), align 4
40    %call_printf17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16 x i8], [16 x i8]* @printf_format.1, i32 0, i32 0), [256 x i8]* getelementptr inbounds ({ [256 x i8], i32, i32, i32 }, { [256 x i8], i32, i32, i32 }* @Book2, i32 0, i32 0), i8 32, i32 %"load record type14", i8 32, i32 %"load record type15", i8 32, i32 %"load record type16")
41    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

```
program ComputeTest;
const
    ic = 1;
    rc = 1.5;
    jc = 8;
    kc = 3;
var
    add_rc : real;
    sub_rc : real;
    mul_rc : real;
begin
    writeln(ic + ic);
    add_rc := rc + rc;
    writeln(add_rc);
    writeln(ic - ic);
    sub_rc := rc - rc;
    writeln(sub_rc);
    mul_rc := rc*rc;
    writeln(mul_rc);
    writeln(ic / (ic + 1));
    writeln(jc mod kc);
end
.
```

- IR

```
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@ic = private constant i32 1
@rc = private constant double 1.500000e+00
@jc = private constant i32 8
@kc = private constant i32 3
@add_rc = common global double 0.000000e+00
@sub_rc = common global double 0.000000e+00
@mul_rc = common global double 0.000000e+00
```

```
@printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.1 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.3 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.4 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.5 = private unnamed_addr constant [5 x i8] c"%lf\0A\00", align 1
@printf_format.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  %"load identifier value" = load i32, i32* @ic, align 4
  %"load identifier value1" = load i32, i32* @ic, align 4
  %addtmp = add i32 %"load identifier value", %"load identifier value1"
  %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %addtmp)
  %"load identifier value2" = load double, double* @rc, align 8
  %"load identifier value3" = load double, double* @rc, align 8
  %addftmp = fadd double %"load identifier value2", %"load identifier value3"
  store double %addftmp, double* @add_rc, align 8
  %"load identifier value4" = load double, double* @add_rc, align 8
  %call_printf5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @printf_format.1, i32 0, i32 0), double %"load identifier value4")
  %"load identifier value6" = load i32, i32* @ic, align 4
  %"load identifier value7" = load i32, i32* @ic, align 4
  %subtmp = sub i32 %"load identifier value6", %"load identifier value7"
  %call_printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format.2, i32 0, i32 0), i32 %subtmp)
  %"load identifier value9" = load double, double* @rc, align 8
  %"load identifier value10" = load double, double* @rc, align 8
  %subftmp = fsub double %"load identifier value9", %"load identifier value10"
  store double %subftmp, double* @sub_rc, align 8
  %"load identifier value11" = load double, double* @sub_rc, align 8
  %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @printf_format.3, i32 0, i32 0), double %"load identifier value11")
  %"load identifier value13" = load double, double* @rc, align 8
  %"load identifier value14" = load double, double* @rc, align 8
```

```
  %mulftmp = fmul double %"load identifier value13", %"load identifier value14"
  store double %mulftmp, double* @mul_rc, align 8
  %"load identifier value15" = load double, double* @mul_rc, align 8
  %call_printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @printf_format.4, i32 0, i32 0), double %"load identifier value15")
  %"load identifier value17" = load i32, i32* @ic, align 4
  %"load identifier value18" = load i32, i32* @ic, align 4
  %addtmp19 = add i32 %"load identifier value18", 1
  %0 = uitofp i32 %"load identifier value17" to double
  %1 = uitofp i32 %addtmp19 to double
  %divftmp = fdiv double %0, %1
  %call_printf20 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @printf_format.5, i32 0, i32 0), double %divftmp)
  %"load identifier value21" = load i32, i32* @jc, align 4
  %"load identifier value22" = load i32, i32* @kc, align 4
  %modtmp = srem i32 %"load identifier value21", %"load identifier value22"
  %call_printf23 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format.6, i32 0, i32 0), i32 %modtmp)
  ret i32 0
}

declare i32 @printf(i8*, ...)

```

- 汇编指令

```
    .text
    .file   "pascal_module"
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    pushl   %esi
    .cfi_def_cfa_offset 8
    subl    $40, %esp
    .cfi_def_cfa_offset 48
    .cfi_offset %esi, -8
    movl    .Lic, %esi
    leal    (%esi,%esi), %eax
    movl    %eax, 4(%esp)
    movl    $.Lprintf_format, (%esp)
    calll   printf@PLT
    fldl    .Lrc
    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:
```

```
    .long   1                               # 0x1
    .size   .Lic, 4

    .type   .Lrc,@object                    # @rc
    .p2align    3
.Lrc:
    .quad   0x3ff8000000000000              # double 1.5
    .size   .Lrc, 8

    .type   .Ljc,@object                    # @jc
    .p2align    2
.Ljc:
    .long   8                               # 0x8
    .size   .Ljc, 4

    .type   .Lkc,@object                    # @kc
    .p2align    2
.Lkc:
    .long   3                               # 0x3
    .size   .Lkc, 4

    .type   add_rc,@object                  # @add_rc
    .comm   add_rc,8,8
    .type   sub_rc,@object                  # @sub_rc
    .comm   sub_rc,8,8
    .type   mul_rc,@object                  # @mul_rc
    .comm   mul_rc,8,8
    .type   .Lprintf_format,@object         # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%d\n"
    .size   .Lprintf_format, 4

    .type   .Lprintf_format.1,@object       # @printf_format.1
.Lprintf_format.1:
    .asciz  "%lf\n"
    .size   .Lprintf_format.1, 5

    .type   .Lprintf_format.2,@object       # @printf_format.2
.Lprintf_format.2:
    .asciz  "%d\n"
    .size   .Lprintf_format.2, 4

    .type   .Lprintf_format.3,@object       # @printf_format.3
.Lprintf_format.3:
    .asciz  "%lf\n"
    .size   .Lprintf_format.3, 5

    .type   .Lprintf_format.4,@object       # @printf_format.4
.Lprintf_format.4:
    .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

```
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@constTerm1 = private constant i32 3
@constTerm2 = private constant double 3.140000e+00
@curTerm1 = common global i32 0
@curTerm2 = common global i32 0
@printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@printf_format.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  %"load identifier value" = load i32, i32* @constTerm1, align 4
  store i32 %"load identifier value", i32* @curTerm1, align 4
  store i32 11, i32* @curTerm1, align 4
  %"load identifier value1" = load i32, i32* @curTerm1, align 4
  %cmptmp = icmp eq i32 %"load identifier value1", 11
  br i1 %cmptmp, label %if_then, label %if_else

if_then:                                          ; preds = %entry
  store i32 5, i32* @curTerm2, align 4
  br label %if_continue

if_else:                                          ; preds = %entry
  %"load identifier value5" = load i32, i32* @curTerm1, align 4
  %cmptmp6 = icmp eq i32 %"load identifier value5", 12
  br i1 %cmptmp6, label %if_then2, label %if_else3

if_continue:                                      ; preds = %if_continue4, %if_then
  %"load identifier value7" = load i32, i32* @curTerm1, align 4
  %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")
  %"load identifier value8" = load i32, i32* @curTerm2, align 4
  %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")
  ret i32 0

if_then2:                                         ; preds = %if_else
  store i32 6, i32* @curTerm2, align 4
  br label %if_continue4

if_else3:                                         ; preds = %if_else
```

```
  store i32 7, i32* @curTerm2, align 4
  br label %if_continue4

if_continue4:                                     ; preds = %if_else3, %if_then2
  br label %if_continue
}

declare i32 @printf(i8*, ...)

```

- 汇编指令

```
    .text
    .file   "pascal_module"
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    subl    $12, %esp
    .cfi_def_cfa_offset 16
    movl    $11, curTerm1
    xorl    %eax, %eax
    testb   %al, %al
    jne .LBB0_4
# %bb.1:                                # %if_then
    movl    $5, curTerm2
    jmp .LBB0_2
.LBB0_4:                                # %if_else
    cmpl    $12, curTerm1
    jne .LBB0_5
# %bb.3:                                # %if_then2
    movl    $6, curTerm2
    jmp .LBB0_2
.LBB0_5:                                # %if_else3
    movl    $7, curTerm2
.LBB0_2:                                # %if_continue
    subl    $8, %esp
    .cfi_adjust_cfa_offset 8
    pushl   curTerm1
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $8, %esp
    .cfi_adjust_cfa_offset -8
    pushl   curTerm2
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format.1
```

```
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $16, %esp
    .cfi_adjust_cfa_offset -16
    xorl    %eax, %eax
    addl    $12, %esp
    .cfi_def_cfa_offset 4
    retl
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .LconstTerm1,@object            # @constTerm1
    .section    .rodata,"a",@progbits
    .p2align    2
.LconstTerm1:
    .long   3                               # 0x3
    .size   .LconstTerm1, 4

    .type   .LconstTerm2,@object            # @constTerm2
    .p2align    3
.LconstTerm2:
    .quad   0x40091eb851eb851f              # double
3.1400000000000001
    .size   .LconstTerm2, 8

    .type   curTerm1,@object                # @curTerm1
    .comm   curTerm1,4,4
    .type   curTerm2,@object                # @curTerm2
    .comm   curTerm2,4,4
    .type   .Lprintf_format,@object         # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%d\n"
    .size   .Lprintf_format, 4

    .type   .Lprintf_format.1,@object       # @printf_format.1
.Lprintf_format.1:
    .asciz  "%d\n"
    .size   .Lprintf_format.1, 4

    .section    ".note.GNU-stack","",@progbits

```

- 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

```
program loopstmt;
var
    array_a : array[0 .. 625] of integer;
    i : Integer;
    sum : Integer;
begin
    WriteLn('test for for');
    for i:=2 to 10 do array_a[i]:=i;
    for i:=2 to 10 do WriteLn(array_a[i]);
    WriteLn('test for while: 1+...10');
    i:=0;
    while i<10 do begin
        sum:=sum+i;
        i:=i+1;
    end;
    Writeln(sum);
end.
```

- IR

```
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@array_a = common global [626 x i32] zeroinitializer
@i = common global i32 0
@sum = common global i32 0
@0 = private unnamed_addr constant [256 x i8] c"test for
for\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00", align 1
@printf_format = private unnamed_addr constant [4 x i8]
c"%s\0A\00", align 1
@printf_format.1 = private unnamed_addr constant [4 x i8]
c"%d\0A\00", align 1
```

```
@1 = private unnamed_addr constant [256 x i8] c"test for while: 1+...10\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00", align 1
@printf_format.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@printf_format.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  %"load const string" = load [256 x i8], [256 x i8]* @0, align 1
  %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format, i32 0, i32 0), [256 x i8]* @0)
  br label %for_start

for_start:                                        ; preds = %entry
  store i32 2, i32* @i, align 4
  br i1 false, label %for_end, label %for_handle

for_handle:                                       ; preds = %for_condition, %for_start
  %"load identifier value" = load i32, i32* @i, align 4
  %subtmp = sub i32 %"load identifier value", 0
  %ArrayCall = getelementptr [626 x i32], [626 x i32]* @array_a, i32 0, i32 %subtmp
  %"load array value" = load i32, i32* %ArrayCall, align 4
  %"load identifier value1" = load i32, i32* @i, align 4
  store i32 %"load identifier value1", i32* %ArrayCall, align 4
  br label %for_condition

for_condition:                                    ; preds = %for_handle
  %"load identifier value2" = load i32, i32* @i, align 4
  %tmpadd = add i32 %"load identifier value2", 1
  store i32 %tmpadd, i32* @i, align 4
  %"load identifier value3" = load i32, i32* @i, align 4
  %cmptmp = icmp sgt i32 %"load identifier value3", 10
  br i1 %cmptmp, label %for_end, label %for_handle
  br label %for_end

```

```
42  for_end:                                          ; preds =
    %for_condition, %for_condition, %for_start
43    br label %for_start4
44
45  for_start4:                                       ; preds = %for_end
46    store i32 2, i32* @i, align 4
47    br i1 false, label %for_end7, label %for_handle5
48
49  for_handle5:                                      ; preds =
    %for_condition6, %for_start4
50    %"load identifier value8" = load i32, i32* @i, align 4
51    %subtmp9 = sub i32 %"load identifier value8", 0
52    %ArrayCall10 = getelementptr [626 x i32], [626 x i32]* @array_a, i32
    0, i32 %subtmp9
53    %"load array value11" = load i32, i32* %ArrayCall10, align 4
54    %call_printf12 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @printf_format.1, i32 0, i32 0),
    i32 %"load array value11")
55    br label %for_condition6
56
57  for_condition6:                                   ; preds =
    %for_handle5
58    %"load identifier value13" = load i32, i32* @i, align 4
59    %tmpadd14 = add i32 %"load identifier value13", 1
60    store i32 %tmpadd14, i32* @i, align 4
61    %"load identifier value15" = load i32, i32* @i, align 4
62    %cmptmp16 = icmp sgt i32 %"load identifier value15", 10
63    br i1 %cmptmp16, label %for_end7, label %for_handle5
64    br label %for_end7
65
66  for_end7:                                         ; preds =
    %for_condition6, %for_condition6, %for_start4
67    %"load const string17" = load [256 x i8], [256 x i8]* @1, align 1
68    %call_printf18 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @printf_format.2, i32 0, i32 0),
    [256 x i8]* @1)
69    store i32 0, i32* @i, align 4
70    br label %while_condition
71
72  while_condition:                                  ; preds =
    %while_body, %for_end7
73    %"load identifier value19" = load i32, i32* @i, align 4
74    %cmptmp20 = icmp slt i32 %"load identifier value19", 10
75    br i1 %cmptmp20, label %while_body, label %while_end
76
77  while_body:                                       ; preds =
    %while_condition
78    %"load identifier value21" = load i32, i32* @sum, align 4
79    %"load identifier value22" = load i32, i32* @i, align 4
80    %addtmp = add i32 %"load identifier value21", %"load identifier value22"
```

```
  store i32 %addtmp, i32* @sum, align 4
  %"load identifier value23" = load i32, i32* @i, align 4
  %addtmp24 = add i32 %"load identifier value23", 1
  store i32 %addtmp24, i32* @i, align 4
  br label %while_condition

while_end:                                        ; preds =
%while_condition
  %"load identifier value25" = load i32, i32* @sum, align 4
  %call_printf26 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([4 x i8], [4 x i8]* @printf_format.3, i32 0, i32 0),
i32 %"load identifier value25")
  ret i32 0
}

declare i32 @printf(i8*, ...)

```

- 汇编指令

```
    .text
    .file   "pascal_module"
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    subl    $20, %esp
    .cfi_adjust_cfa_offset 20
    pushl   $.L__unnamed_1
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $16, %esp
    .cfi_adjust_cfa_offset -16
    movl    $2, i
    .p2align    4, 0x90
.LBB0_1:                                # %for_handle
                                        # =>This Inner Loop Header:
Depth=1
    movl    i, %eax
    movl    %eax, array_a(,%eax,4)
    movl    i, %eax
    incl    %eax
    movl    %eax, i
    cmpl    $11, %eax
    jl  .LBB0_1
# %bb.2:                                # %for_end
    movl    $2, i
```

```
    .p2align    4, 0x90
.LBB0_3:                                # %for_handle5
                                        # =>This Inner Loop Header:
Depth=1
    movl    i, %eax
    subl    $8, %esp
    .cfi_adjust_cfa_offset 8
    pushl   array_a(,%eax,4)
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format.1
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $16, %esp
    .cfi_adjust_cfa_offset -16
    movl    i, %eax
    incl    %eax
    movl    %eax, i
    cmpl    $11, %eax
    jl  .LBB0_3
# %bb.4:                                # %for_end7
    subl    $8, %esp
    .cfi_adjust_cfa_offset 8
    pushl   $.L__unnamed_2
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format.2
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $16, %esp
    .cfi_adjust_cfa_offset -16
    movl    $0, i
    cmpl    $9, i
    jg  .LBB0_7
    .p2align    4, 0x90
.LBB0_6:                                # %while_body
                                        # =>This Inner Loop Header:
Depth=1
    movl    i, %eax
    addl    %eax, sum
    incl    %eax
    movl    %eax, i
    cmpl    $9, i
    jle .LBB0_6
.LBB0_7:                                # %while_end
    subl    $8, %esp
    .cfi_adjust_cfa_offset 8
    pushl   sum
    .cfi_adjust_cfa_offset 4
    pushl   $.Lprintf_format.3
    .cfi_adjust_cfa_offset 4
    calll   printf@PLT
    addl    $16, %esp
```

```
    .cfi_adjust_cfa_offset -16
    xorl    %eax, %eax
    addl    $12, %esp
    .cfi_def_cfa_offset 4
    retl
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   array_a,@object             # @array_a
    .comm   array_a,2504,16
    .type   i,@object                   # @i
    .comm   i,4,4
    .type   sum,@object                 # @sum
    .comm   sum,4,4
    .type   .L__unnamed_1,@object       # @0
    .section    .rodata,"a",@progbits
.L__unnamed_1:
    .asciz  "test for
for\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000"
    .size   .L__unnamed_1, 256

    .type   .Lprintf_format,@object     # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%s\n"
    .size   .Lprintf_format, 4

    .type   .Lprintf_format.1,@object   # @printf_format.1
.Lprintf_format.1:
    .asciz  "%d\n"
    .size   .Lprintf_format.1, 4

    .type   .L__unnamed_2,@object       # @1
    .section    .rodata,"a",@progbits
.L__unnamed_2:
```

```
    .asciz  "test for while: 
1+...10\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000"
    .size   .L__unnamed_2, 256

    .type   .Lprintf_format.2,@object       # @printf_format.2
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format.2:
    .asciz  "%s\n"
    .size   .Lprintf_format.2, 4

    .type   .Lprintf_format.3,@object       # @printf_format.3
.Lprintf_format.3:
    .asciz  "%d\n"
    .size   .Lprintf_format.3, 4

    .section    ".note.GNU-stack","",@progbits

```

- 运行结果

```
lz@ubuntu:~/Documents/Our_Pascal_Compiler$ lli whilestmt.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

```
program declfunc;
const
    CurTerm1 = 3;
var
    CurTerm2 : integer;
function myLine(a : integer): integer;
begin
    myLine := a + a;
end;
begin
    CurTerm2 := myLine(CurTerm1);
    WriteLn(CurTerm2);
end.
```

- IR

```
; ModuleID = 'pascal_module'
source_filename = "pascal_module"

@CurTerm1 = private constant i32 3
@CurTerm2 = common global i32 0
@printf_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  %"load identifier value" = load i32, i32* @CurTerm1, align 4
  %0 = call i32 @myLine(i32* @CurTerm1)
  store i32 %0, i32* @CurTerm2, align 4
  %"load identifier value1" = load i32, i32* @CurTerm2, align 4
  %call_printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @printf_format, i32 0, i32 0), i32 %"load identifier value1")
  ret i32 0
}

define i32 @myLine(i32* %0) {
entry:
  %"load var param" = load i32, i32* %0, align 4
  %a = alloca i32, align 4
  store i32 %"load var param", i32* %a, align 4
  %myLine = alloca i32, align 4
  %"load identifier value" = load i32, i32* %a, align 4
  %"load identifier value1" = load i32, i32* %a, align 4
  %addtmp = add i32 %"load identifier value", %"load identifier value1"
  store i32 %addtmp, i32* %myLine, align 4
  %"load ret param" = load i32, i32* %myLine, align 4
  ret i32 %"load ret param"
}

declare i32 @printf(i8*, ...)
```

```
33
```

- 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
24                                          # -- End function
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
44                                          # -- End function
45      .type   .LCurTerm1,@object              # @CurTerm1
46      .section    .rodata,"a",@progbits
47      .p2align    2
```

```
.LCurTerm1:
    .long   3                               # 0x3
    .size   .LCurTerm1, 4

    .type   CurTerm2,@object                # @CurTerm2
    .comm   CurTerm2,4,4
    .type   .Lprintf_format,@object         # @printf_format
    .section    .rodata.str1.1,"aMS",@progbits,1
.Lprintf_format:
    .asciz  "%d\n"
    .size   .Lprintf_format, 4

    .section    ".note.GNU-stack","",@progbits

```

- 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

```
program arraymul;
var
    array_a : array[0 .. 625] of integer;
    array_b : array[0 .. 625] of integer;
    array_c : array[0 .. 625] of integer;
    m_a : integer;
    n_a : integer;
    s_a : integer;
    m_b : integer;
    n_b : integer;
    s_b : integer;
    s_c : integer;
    i : integer;
    j : integer;
    k : integer;
    tmp_a : Integer;
    tmp_b : Integer;
    tmp_c : Integer;
begin
    //读入矩阵A
    ReadLn(m_a,n_a);
    // WriteLn('m=',m_a,',n=',n_a);
    s_a:=m_a*n_a;
    // WriteLn('m * n = ',s_a);

    for i:=0 to s_a-1 do Read(array_a[i]);
    //读入矩阵B
    ReadLn(m_b,n_b);
    // WriteLn('m=',m_b,',n=',n_b);
    s_b:=m_b*n_b;
    // WriteLn('m * n = ',s_b);

    for i:=0 to s_b-1 do Read(array_b[i]);

    if n_a <> m_b then
        WriteLn('Incompatible Dimensions')
    else begin
        //进行矩阵乘法
        for i:=0 to m_a-1 do
        begin
            for j:=0 to n_b-1 do
            begin
                for k:=0 to m_b-1 do
                begin
                    tmp_c := i*n_b+j;
                    tmp_b := k*n_b+j;
                    tmp_a := i*m_b+k;
```

```
                    array_c[tmp_c] := array_c[tmp_c] + array_b[tmp_b] *
    array_a[tmp_a];
                end;
            end;
        end;

        //测试输出
        s_c := m_a*n_b;
        for i:=0 to m_a-1 do
        begin
            for j:=0 to n_b-1 do
            begin
                k:= i*n_b+j;
                printf(array_c[k]);
            end;
            WriteLn;
        end;
    end;
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

```
Program advisor;

Var
    courses :   array[0 .. 39999] of char;
    map: array[0..99] of integer;
    is_passed: array[0..99] of integer;
    course_name: array[0..4] of char;

    // main variable
    ch: char;
    num: integer;
    index: integer;
    total_credits: integer;
```

```
    attempted_credits: integer;
    passed_credits: integer;
    remained_credits: integer;
    total_score: integer;
    field: integer;
    credit: integer;
    key: integer;
    i: integer;
    GPA: real;
    value: integer;
    flag: integer;
    j: integer;
    k: integer;
    tmp_idx: integer;

    myeof: char;
    c_tmp1 : integer;
    c_tmp2 : integer;
    tmp_value : char;
    tmp_int : Integer;
    zero : char;
    split : char;
    A : char;
    B : char;
    C : char;
    D : char;
    E : char;
    F : char;

    tmp_string : string;

    intzero: Integer;
    intnine : Integer;



function char2int(a: char):integer;
begin
    if a = '0' then begin
        char2int := 0;
    end else if a = '1' then begin
        char2int := 1;
    end else if a = '2' then begin
        char2int := 2;
    end else if a = '3' then begin
        char2int := 3;
    end else if a = '4' then begin
        char2int := 4;
    end else if a = '5' then begin
        char2int := 5;
    end else if a = '6' then begin
        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
    begin
159                     tmp_value := courses[tmp_idx];
160                     c_tmp1 := char2int(tmp_value);
161                     tmp_value := zero;
162                     c_tmp2 := char2int(tmp_value);
163                     key := key * 10 + c_tmp1 - c_tmp2;
164                     // write('key = ');
165                     // write(key);
166                     // write('\n');
167                 end;
168                 index := index + 1;
```

```
169             end else if (field = 1) then begin
170                 tmp_idx := i * 400 + index;
171                 tmp_value := courses[tmp_idx];
172                 c_tmp1 := char2int(tmp_value);
173                 tmp_value := zero;
174                 c_tmp2 := char2int(tmp_value);
175 
176                 credit := c_tmp1 - c_tmp2;
177                 total_credits := total_credits + credit;
178 
179                 map[key] := i;
180                 index := index + 1;
181             end else if (field = 2) then begin
182                 tmp_idx := i * 400 + index;
183                 tmp_value := courses[tmp_idx];
184                 if ( courses[tmp_idx] <> '|') then begin
185                     index := index + 1;
186                 end;
187             end else if (field = 3) then begin
188                 tmp_idx := i * 400 + index;
189                 tmp_value := courses[tmp_idx];
190 
191                 if (tmp_value <> myeof) then begin
192                     if (tmp_value = 'A') then begin
193                         attempted_credits := attempted_credits + credit;
194                         passed_credits := passed_credits + credit;
195                         total_score := total_score + 4 * credit;
196                         is_passed[i] := 1;
197                     end;
198                     if (tmp_value = 'B') then begin
199                         attempted_credits := attempted_credits + credit;
200                         passed_credits := passed_credits + credit;
201                         total_score := total_score + 3 * credit;
202                         is_passed[i] := 1;
203                     end;
204                     if (tmp_value = 'C') then begin
205                         attempted_credits := attempted_credits + credit;
206                         passed_credits := passed_credits + credit;
207                         total_score := total_score + 2 * credit;
208                         is_passed[i] := 1;
209                     end;
210                     if (tmp_value  = 'D') then begin
211                         attempted_credits := attempted_credits + credit;
212                         passed_credits := passed_credits + credit;
213                         total_score := total_score + credit;
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;
```

```
                tmp_idx := i * 400 + index;
                tmp_value := courses[tmp_idx] ;
            end;
            tmp_idx := i * 400 + index;
            tmp_value := courses[tmp_idx] ;
        end;
        tmp_idx := i * 400 + index;
        tmp_value := courses[tmp_idx] ;
    end;

    if (attempted_credits <> 0) then begin
        GPA := 1.0 * total_score / attempted_credits;
    end;
    remained_credits := total_credits - passed_credits;
    write('GPA: ');
    write(GPA);
    write('\n');

    write('Hours Attempted: ');
    write(attempted_credits);
    write('\n');

    write('Hours Completed: ');
    write(passed_credits);
    write('\n');

    write('Credits Remaining: ');
    write(remained_credits);
    write('\n');

    write('\n');
    write('Possible Courses to Take Next');
    write('\n');

    if (remained_credits = 0) then begin
        writeln('  None - Congratulations!');
    end
    else begin // find the class need to take
        value := 0;
        flag := 1;
        j := 0;

        for i:=0 to (num-1) do begin

            if (is_passed[i] = 0) then begin // not even passed,
refer
                index := 0;
                j := 0;
                tmp_idx := i * 400 + index;
                while (courses[tmp_idx] <> split) do begin // left
courses name
                    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
    courses name, because there is no other dependent classes
291                     write('  ');
292                     for k:=0 to (j-1) do begin
293                         write(course_name[k]);
294                     end;
295                     write('\n');
296                 end
297                 else begin
298
299                     flag := 1;
300                     key := 0;
301
302                     tmp_idx := i * 400 + index;
303                     while (courses[tmp_idx] <> split) do begin //there
    is other class name
304                         tmp_value := courses[tmp_idx];
305                         tmp_int := char2int(tmp_value);
306
307                         if (( tmp_int <= intnine) and (tmp_int >=
    intzero)) then begin
308                             tmp_value := courses[tmp_idx];
309                             c_tmp1 := char2int(tmp_value);
310                             tmp_value := zero;
311                             c_tmp2 := char2int(tmp_value);
312                             key := key * 10 + c_tmp1 - c_tmp2; // class
    name enum key
313                         end;
314
315                         tmp_idx := i * 400 + index;
316                         if (courses[tmp_idx] = ',') then begin
317                             value := map[key];
```

```
318                             if (is_passed[value] = 0) then begin
319                                 flag := 0; //dependent class not
    pass, ignore this one
320                             end;
321                             key := 0; //clear key
322                         end;
323 
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

```
program:
    pro_head routine SYM_PERIOD {
        //root of the ast, a global variable;
        ast_root = new AST_Program($1,$2);
        SET_LOCATION(ast_root);
    }
;

pro_head:
    KEY_PROGRAM IDENTIFIER SYM_SEMICOLON{
        $$ = new AST_Program_Head($2);
        SET_LOCATION($$);
    }
;

routine:
    routine_head routine_body{
        // function_decl and procedure_decl
        $$ = new AST_Routine($1,$2);
        SET_LOCATION($$);
    }
;

routine_head:
    const_part type_part var_part routine_part{
        $$ = new AST_Routine_Head($1,$2,$3,$4);
        SET_LOCATION($$);
    }
;

const_part:
    KEY_CONST const_expr_list {
        $$ = new AST_Const_Part($2);
        SET_LOCATION($$);
    }
    | {
        $$ = nullptr;
    }
```

```
;

type_part:
    KEY_TYPE type_decl_list {
        $$ = new AST_Type_Part($2);
        SET_LOCATION($$);
    }
    | {
        $$ = nullptr;
    }
;

var_part:
    KEY_VAR var_decl_list {
        $$ = new AST_Variable_Part($2);
        SET_LOCATION($$);
    }
    | {
        $$ = nullptr;
    }
;

routine_part:
    routine_part function_decl {
        AST_Declaration_BaseClass* tmp1 = new
AST_Declaration_BaseClass($2);
        ($1) -> Add_Declaration(tmp1);
        $$ = $1;
        SET_LOCATION($$);
    }
    | routine_part procedure_decl {
        AST_Declaration_BaseClass* tmp2 = new
AST_Declaration_BaseClass($2);
        ($1) -> Add_Declaration(tmp2);
        $$ = $1;
        SET_LOCATION($$);
    }
    | function_decl {
        AST_Declaration_BaseClass* tmp3 = new
AST_Declaration_BaseClass($1);
        $$ = new AST_Routine_Part(tmp3);
        SET_LOCATION($$);
    }
    | procedure_decl {
        AST_Declaration_BaseClass* tmp4 = new
AST_Declaration_BaseClass($1);
        $$ = new AST_Routine_Part(tmp4);
        SET_LOCATION($$);
    }
    | {
        $$ = nullptr;
    }
```

```
;

routine_body:
    compound_stmt {
        $$ = new AST_Routine_Body($1);
        SET_LOCATION($$);
    }
;

compound_stmt:
    KEY_BEGIN stmt_list KEY_END{
        $$ = new AST_Compound_Statement($2);
        SET_LOCATION($$);
    }
;

const_expr_list:
    const_expr_list const_expr {
        ($1)->Add_Const_Expression($2);
        $$ = $1;
        SET_LOCATION($$);
    }
    | const_expr {
        $$ = new AST_Const_Expression_List();
        ($$)->Add_Const_Expression($1);
        SET_LOCATION($$);
    }
;

const_expr:
    IDENTIFIER SYM_EQ expression SYM_SEMICOLON {
        $$ = new AST_Const_Expression($1,$3);
        SET_LOCATION($$);
    }
;

const_value:
    LITERAL_INT {
        $$ = new AST_Const_Value($1,AST_Const_Value::Value_Type::INT);
        SET_LOCATION($$);
    }
    | LITERAL_FLOAT {
        $$ = new AST_Const_Value($1,AST_Const_Value::Value_Type::FLOAT);
        SET_LOCATION($$);
    }
    | LITERAL_CHAR {
        $$ = new AST_Const_Value($1,AST_Const_Value::Value_Type::CHAR);
        SET_LOCATION($$);
    }
    | LITERAL_STR {
        $$ = new AST_Const_Value($1,AST_Const_Value::Value_Type::STRING);
        SET_LOCATION($$);
```

```
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($$);
475         }
476     ;
477
478
479     proc_stmt:
480         /* SYS_PROC | SYS_PROC  LP  expression_list  RP |  READ  LP
        factor  RP */
481         IDENTIFIER {
482             $$ = 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($$);
488         }
489     ;
490
491
492     if_stmt:
493         KEY_IF expression KEY_THEN stmt else_clause {
494             $$ = new AST_If_Statement($2,$4,$5);
495             SET_LOCATION($$);
496         }
497     ;
498
499     else_clause:
500         KEY_ELSE stmt {
501             $$ = new AST_Else_Clause($2);
```

```
502             SET_LOCATION($$);
503         }
504         | {
505             $$ = nullptr;
506         }
507     ;
508
509     case_stmt:
510         KEY_CASE expression KEY_OF case_expr_list KEY_END {
511             $$ = new AST_Case_Statement($2,$4);
512             SET_LOCATION($$);
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
    AST_Binary_Expression(AST_Binary_Expression::Operation::GE, $1, $3);
```

```
604             SET_LOCATION($$);
605         }
606         | expression SYM_GT expr {
607             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::GT, $1, $3);
608             SET_LOCATION($$);
609         }
610         | expression SYM_LE expr {
611             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::LE, $1, $3);
612             SET_LOCATION($$);
613         }
614         | expression SYM_LT expr {
615             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::LT, $1, $3);
616             SET_LOCATION($$);
617         }
618         | expression SYM_EQ expr {
619             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::EQUAL, $1, $3);
620             SET_LOCATION($$);
621         }
622         | expression SYM_NE expr{
623             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::UNEQUAL, $1, $3);
624             SET_LOCATION($$);
625         }
626         | expr{
627             $$ = $1;
628             SET_LOCATION($$);
629         }
630     ;
631
632     expr:
633         expr SYM_ADD term {
634             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::PLUS,$1,$3);
635             SET_LOCATION($$);
636         }
637         | expr SYM_SUB term {
638             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::MINUS,$1,$3);
639             SET_LOCATION($$);
640         }
641         | expr KEY_OR term {
642             $$ = new
    AST_Binary_Expression(AST_Binary_Expression::Operation::OR,$1,$3);
643             SET_LOCATION($$);
644         }
645         | term {
646             $$ = $1;
647             SET_LOCATION($$);
```

```
    }
;

term:
    term  SYM_MUL factor{
        $$ = new
AST_Binary_Expression(AST_Binary_Expression::Operation::MUL,$1,$3);
        SET_LOCATION($$);
    }
    | term  SYM_DIV factor{
        $$ = new
AST_Binary_Expression(AST_Binary_Expression::Operation::REALDIV,$1,$3);
        SET_LOCATION($$);
    }
    | term  KEY_DIV factor {
        $$ = new
AST_Binary_Expression(AST_Binary_Expression::Operation::DIV,$1,$3);
        SET_LOCATION($$);
    }
    | term  KEY_MOD factor  {
        $$ = new
AST_Binary_Expression(AST_Binary_Expression::Operation::MOD,$1,$3);
        SET_LOCATION($$);
    }
    | term  KEY_AND factor{
        $$ = new
AST_Binary_Expression(AST_Binary_Expression::Operation::AND,$1,$3);
        SET_LOCATION($$);
    }
    | factor {
        $$ = $1;
        SET_LOCATION($$);
    }
;

factor:
    IDENTIFIER{
        $$ = new AST_Identifier_Expression($1);
        SET_LOCATION($$);
    }
    | IDENTIFIER SYM_LPAREN expression_list SYM_RPAREN {
        $$ = new AST_Function_Call($1,$3);
        SET_LOCATION($$);
    }
    | const_value{
        $$ = new AST_Const_Value_Expression($1);
        SET_LOCATION($$);
    }
    | SYM_LPAREN expression SYM_RPAREN{
        $$ = $2;
        SET_LOCATION($$);
    }
```

```
695       | KEY_NOT factor{
696           $$ = new
      AST_Unary_Expression(AST_Unary_Expression::Operation::NOT,$2);
697           SET_LOCATION($$);
698       }
699       | SYM_SUB factor{
700           $$ = new
      AST_Unary_Expression(AST_Unary_Expression::Operation::SUB,$2);
701           SET_LOCATION($$);
702       }
703       | SYM_ADD factor{
704           $$ = new
      AST_Unary_Expression(AST_Unary_Expression::Operation::ADD,$2);
705           SET_LOCATION($$);
706       }
707       | IDENTIFIER SYM_LBRAC expression SYM_RBRAC{
708           $$ = new AST_Array_Expression($1,$3);
709           SET_LOCATION($$);
710       }
711       | IDENTIFIER SYM_PERIOD IDENTIFIER{
712           $$ = new AST_Property_Expression($1,$3);
713           SET_LOCATION($$);
714       }
715   ;
```