编译原理实验报告

小组成员： 郑祎豪、潘宇轩、张涵智

2022/5/22

指导老师：冯雁

序言

概述

本小组实验基于C++语言完成了一个实现C语言部分功能的编译器，能够分析C语言的语法，并编译志LLVMIR。最后，借助LLVM的工具链，可以将LLVMIR文件编译成RISC-V汇编文件、arm汇编文件以及.o文件。最后使用GCC链接.o文件生成可执行文件。

为了进一步详细说明，我们首先采用了开源工具Lex来实施词法分析，此后我们使用Yacc工具执行语法分析，这一步将生成抽象语法树（AST）。当语法分析完成，我们将对AST进行语义分析，这一阶段将生成LLVM的中间表示。然后，我们可以借助LLVM工具链，将中间表示编译为目标系统所需的汇编代码。

开发文件

文件目录

```
2023_foc_compiler
├── doc
│   ├── 编译原理实验报告.pdf
│   └── 课堂展示.pdf
├── src
│   ├── ast.cpp
│   ├── ast.hpp
│   ├── CodeGenerator.cpp
│   ├── CodeGenerator.hpp
│   ├── lexer.l
│   ├── main.cpp
│   ├── Makefile
│   └── parser.y
└── test
    ├── test1.c
    ├── test2.c
    └── test3.c
```

提交文件说明

本次实验提交的文件及其说明如下：

- src：源代码文件夹
  - lexer.l：实现词法分析的flex源代码，生成token
  - parser.y：实现语法分析的yacc源代码，生成抽象语法树
  - CodeGenerator.cpp：中间代码生成器实现文件
  - CodeGenerator.hpp：中间代码生成器头文件，定义生成器环境
  - ast.hpp：抽象语法树头文件，定义所有抽象语法树节点类
  - ast.cpp：抽象语法树实现文件，主要包含 codeGen方法的实现
  - main.cpp：主函数所在文件，主要负责调用词法分析器、语法分析器、代码生成器
  - Makefile：定义编译链接规则
- doc：
  - 编译原理实验报告.pdf：报告文档
- 课堂展示.pdf：展示文档
- test:测试用例文件夹

组员分工

- 语法分析、词法分析：张涵智、郑祎豪
- 语义分析、中间代码生成、目标代码生成、测试：潘宇轩、郑祎豪
- 实验报告：郑祎豪

实现特性

- **C语言的基本语句。**`if`、`else`、`for`、`while`、`switch case`、`continue`、`break`、`return`
- **C语言表达式。**括号：`{}`、`[]`、`()`。取地址符`&`。运算符`&&`、`||`、`^`、`~`、`=`、`==`、`!=`、`<`、`<=`、`>`、`>=`、`+`、`-`、`*`、`/`
- **C语言内置类型。**`int`、`char`、`double`、`void`
- **数组。**`int a[num]`
- **类型转换。**主要实现了隐式类型转换，例如`int + double -> double`
- **可以调用C标准库里的printf函数和scanf函数。**可以在文件中直接调用`printf("%d",a)`
- **符号表作用域。**在如`if`、`for`、`while`语句块中重新定义变量会覆盖外层作用域中的同名变量，并且新定义的变量作用域只在语句内。

运行环境

操作系统

Ubuntu 22.04.2 LTS

依赖项

flex 2.6.4

bison 3.8.2

llvm 14.0.0

运行指南

1. 进入工程所在的目录，进入src文件夹
2. 运行`make`生成c_compiler可执行文件
3. 运行`c_compiler -i input_file -o output_file.ll`生成LLVM IR文件
4. 运行`c_compiler -i input_file -o output_file.ll -s output_file.s`生成riscv32汇编文件
5. 生成可执行文件

   `llvm-as -o=main.bc helloworld.ll`

   `llc -filetype=obj -o=main.o main.bc`

   `gcc -no-pie -o main main.o` 或 `clang -no-pie -o main main.o`

   生成riscv-32汇编代码

   `llc -march=riscv32 program.ll -o program.s`

词法分析

Lex

flex的全称是 fast lexical analyzer generator，用于产生词法分析器。根据脚本设定的规则，flex会生成lexer.cpp，后面会被编译成可执行文件。它会分析输入中可能存在的符合规则的内容，若找到会执行对应的代码。

flex的输入文件格式如下：

```
definition
%%
rules
%%
user's code
```

具体实现

定义部分

```
%{
#include <iostream>
#include <string>
#include "ast.hpp"
#include "parser.hpp"
#define SAVE_IDENTIFIER  yylval.sVal = new std::string(yytext, yyleng)
#define SAVE_DOUBLE  yylval.dVal = stod(std::string(yytext, yyleng))
#define SAVE_INTEGER  yylval.iVal = stoi(std::string(yytext, yyleng))
#define SAVE_CHAR     yylval.cVal = yytext[1]
#define SAVE_STRING     yylval.strVal = new std::string(yytext, yyleng)
#define TOKEN(t)    (yylval.token = t)
%}
```

- `SAVE_IDENTIFIER`：用于保存识别到的标识符。
- `SAVE_DOUBLE`：用于保存识别到的浮点数。
- `SAVE_INTEGER`：用于保存识别到的整数。
- `SAVE_CHAR`：用于保存识别到的字符。
- `SAVE_STRING`：用于保存识别到的字符串。
- `TOKEN(t)`：将识别到的标记赋给 `yylval.token`。

规则部分

```
%%

"extern"                return TOKEN(EXTERN);//c语言关键字
"return"                        return TOKEN(RETURN);
"for"                   return TOKEN(FOR);
"if"                    return TOKEN(IF);
"else"                  return TOKEN(ELSE);
"while"                 return TOKEN(WHILE);
"switch"                return TOKEN(SWITCH);
"case"                  return TOKEN(CASE);
"default"               return TOKEN(DEFAULT);
"array"                return TOKEN(ARRAY);
"continue"              return TOKEN(CONTINUE);
"until"                 return TOKEN(UNTIL);
"break"                  printf("break_lex\n"); return TOKEN(BREAK);

"int"                                                   {return TOKEN(INT); }
"char"                                                  {return TOKEN(CHAR); }
"double"                                                {return TOKEN(DOUBLE); }
"void"                                                  {return TOKEN(VOID); }
```

```
[a-zA-Z_][a-zA-Z0-9_]*  {SAVE_IDENTIFIER;  return IDENTIFIER;}//标识符
[0-9]+\.[0-9]*              {SAVE_DOUBLE; return REAL;}//浮点
[0-9]+                          {SAVE_INTEGER; return INTEGER;}//整型
\'"[^\\']"\'          {SAVE_CHAR; return CHARACTER;}//字符
\"(\\.|[^"\\])*\"       {SAVE_STRING; return STRING; }//字符串
"&&"                    return TOKEN(AND);//逻辑运算符
"||"                    return TOKEN(OR);
"^"                     return TOKEN(XOR);
"~"                     return TOKEN(NOT);


"="                            { return TOKEN(EQUAL);}//赋值和比较
"=="                            return TOKEN(CEQ);
"!="                            return TOKEN(CNE);
"<"                             return TOKEN(CLT);
"<="                            return TOKEN(CLE);
">"                             return TOKEN(CGT);
">="                              return TOKEN(CGE);


"("                             return TOKEN(LPAREN);//括号
")"                             return TOKEN(RPAREN);
"["                     return TOKEN(LBRACK);
"]"                     return TOKEN(RBRACK);
"{"                               return TOKEN(LBRACE);
"}"                             return TOKEN(RBRACE);



"+"                             return TOKEN(PLUS);//算术运算符
"-"                             return TOKEN(MINUS);
"*"                             return TOKEN(MUL);
"/"                             return TOKEN(DIV);


"."                                 return TOKEN(DOT);//其他
","                             return TOKEN(COMMA);
"&"                             return TOKEN(GAD);//取地址
";"                              return TOKEN(SEMI);
":"                     return TOKEN(COLON);
"->"                              return TOKEN(ARW);//箭头（暂未实现）

[ \t\n]                       //匹配回车
"//"[^\n]*                                  //匹配单行注释
"/*"[^*]*[*]+([^*/][^*]*[*]+)*"/"                           ; //匹配注释


.                       printf("Unknown token!\n"); yyterminate();

%%
```

在规则定义部分，我们设置了一系列的规则以匹配 C 语言中的各种关键字、符号以及注释等。

首先，我们识别了 C 语言的关键字，如"extern", "return", "for", "if", "else", "while", "switch", "case", "default", "array", "continue", "until", "break"等等。一旦匹配到这些关键字，我们就返回相应的标记。接下来，我们识别了一些基本的数据类型，包括"int", "char", "double" 和 "void"。

我们还设置了规则以匹配标识符，即以字母或下划线开始，后接字母、数字或下划线的字符串。以及识别了实数和整数，我们用正则表达式来匹配这些数字，并保存识别到的数值。字符和字符串的匹配也在我们的规则定义中，分别通过'"'[^\']'"'和'"'(.|[^"])*"'进行匹配。我们还定义了识别各种运算符的规则，包括逻辑运算符、赋值和比较运算符、括号符号、算术运算符以及其他一些符号。此外，我们还添加了规则来匹配空白字符（即空格、制表符和换行符），以及单行和多行注释。最后，如果输入的字符不匹配任何规则，我们就打印出"Unknown token!"的错误信息，并结束程序。

通过这些规则的定义，我们的词法分析器可以识别并处理大部分的 C 语言语法元素，这对于后续的语法分析和语义分析等步骤是非常有帮助的。

语法分析

Yacc

Yacc（Yet Another Compiler Compiler）和Bison是两个广泛使用的工具，用于生成语法分析器。它们能够将上下文无关文法转换为可用于解析编程语言或其他领域特定语言的解析器。

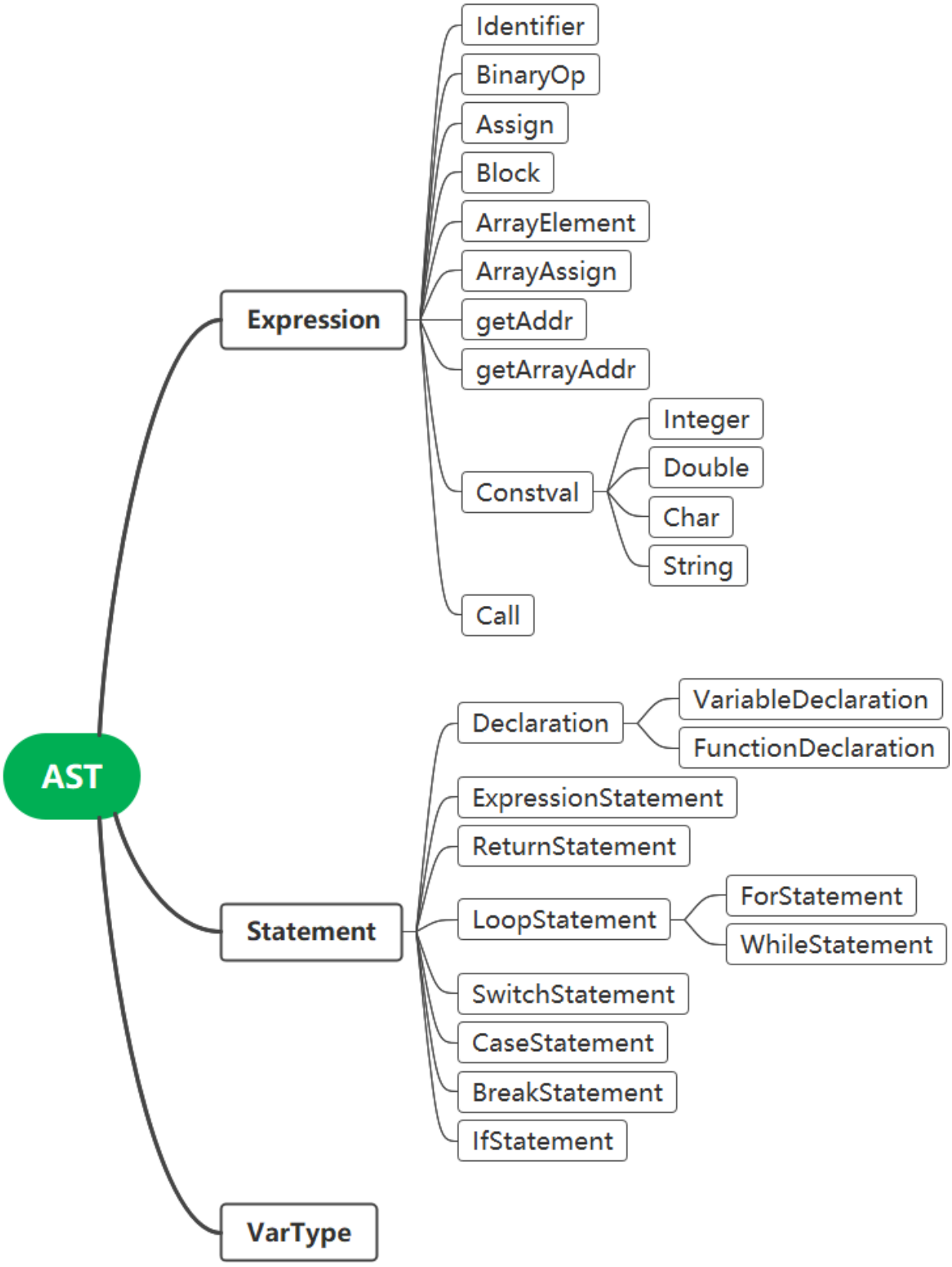
Yacc是最早的工具之一，它采用LALR(1)（Look-Ahead Left-to-Right, Rightmost Derivation）技术来生成语法分析器。通过定义文法规则和语义动作，Yacc能够根据输入的文本串进行解析，并执行相应的语义操作。Yacc生成的解析器主要使用C语言编写，通常需要与词法分析器Lex结合使用，形成完整的编译器前端。

Bison是对Yacc的扩展和改进，它提供了更多功能和灵活性。与Yacc类似，Bison也使用LALR(1)技术，可以生成解析器表用于语法分析。Bison支持多种目标语言，包括C、C++等，使开发人员能够根据自己的需求选择合适的编程语言。Bison还引入了一些实验性的功能，例如生成LR(1)解析器表和GLR（广义LR）解析器表，以扩展其适用范围和性能。

无论是Yacc还是Bison，它们都为开发者提供了强大的工具来构建各种语言解析器。它们被广泛用于编译器设计、解释器开发、领域特定语言的实现等领域，为程序语言的处理提供了基础框架和技术支持。对于熟悉Yacc的开发者来说，迁移到Bison是相对简单的，因为它们具有高度的兼容性和相似的工作原理。

AST设计

flex和bison将输入的文本解析为ast。我们ast类的继承关系如下图所示：


AST
Expression
Identifier
BinaryOp
Assign
Block
ArrayElement
ArrayAssign
getAddr
getArrayAddr
Constval
Integer
Double
Char
String
Call
Statement
Declaration
VariableDeclaration
FunctionDeclaration
ExpressionStatement
ReturnStatement
LoopStatement
ForStatement
WhileStatement
SwitchStatement
CaseStatement
BreakStatement
IfStatement
VarType

文法描述

1. **程序**：整个程序由多个语句（stmts）构成。

```
program:
      stmts {
          programBlock = $1;
      };
```

2. **语句**：语句可以是变量声明、函数声明、返回语句、条件语句、循环语句等。

```
stmt :
   var_decl SEMI  {}
   | func_decl {printf("func_1\n");} //
   | expression SEMI { $$ = new ExpressionStatement(*$1); }
   | RETURN expression SEMI { $$ = new ReturnStatement($2); }
   | RETURN SEMI { $$ = new ReturnStatement(); }
   | BreakStmt  SEMI { printf("break_y\n"); $$ = $1; }
   | ContinueStmt  SEMI { $$ = $1; }
   | SwitchStmt {  $$ = $1;}
   | IfStmt { $$ = $1; }
   | LoopStmt { $$ = $1; }
   ;
```

3. **关键字声明**：处理了各种关键字声明，包括条件语句、循环语句、数组语句等。

```
IfStmt:
   IF LPAREN expression RPAREN block {
    $$ = new IfStatement(*$3, *$5);
}
| IF LPAREN expression RPAREN block ELSE block {
    $$ = new IfStatement(*$3, *$5, $7);
}

LoopStmt:
ForStmt{ $$ = $1; }
|WhileStmt{ $$ = $1; }
;
```

4. **代码块**：处理由大括号包围的一组语句。代码块可能是空的，也可能包含多条语句。

```
block:
LBRACE stmts RBRACE {
    $$ = $2;
}
| LBRACE RBRACE {
    $$ = new Block();
};
```

5. **变量定义、函数定义**：处理变量和函数的定义。

```
var_decl :varType ident{$$ = new VariableDeclaration(*$1, *$2); }
  | varType ident EQUAL expression {  $$ = new VariableDeclaration(*$1, *$2,
$4);  }
  | varType ident LBRACK INTEGER RBRACK { // 定义数组
          $$ = new VariableDeclaration(*$1, *$2, $4);
}
  ;
```

6. **函数参数定义**：通过`func_decl_args`规则来处理。

```
func_decl_args : /*blank*/  { $$ = new VariableList(); }
| var_decl { $$ = new VariableList(); $$->push_back($<var_decl>1); }
| func_decl_args COMMA var_decl { $1->push_back($<var_decl>3); }
;
```

7. **标识符**：处理标识符。

```
ident :
IDENTIFIER {  $$ = new Identifier(*$1); delete $1; }
;
```

8. **表达式**：处理包括赋值运算、逻辑运算、函数调用、数组访问等在内的表达式。

```
expression : ident EQUAL expression { $$ = new Assign(*$<ident>1, *$3); }
| ident LPAREN call_args RPAREN { $$ = new Call(*$1, *$3); delete $3; }
| ident { $<ident>$ = $1; }
| const_value
```

9. **函数调用参数**：处理函数调用的参数。

```
call_args : /*blank*/  { $$ = new ExpressionList(); }//调用函数时的参数
      | expression { $$ = new ExpressionList(); $$->push_back($1); }
      | call_args COMMA expression  { $1->push_back($3); }
      ;
```

10. **常量值**：处理常量值。

```
const_value : INTEGER { $$ = new Integer($1); }//const值
    | REAL { $$ = new Double($1); }
    | CHARACTER { $$ = new Char($1);}
    | STRING { $$ = new String(*$1); }
    ;
```

总的来说，这个语法分析的定义是对C语言一部分语法的覆盖，能够处理大部分常见的C语言程序。

ast类定义

数据结构

在我们的编译器设计中，利用类（class）的继承关系定义了多种不同的节点，来表示不同的语法结构。所有的节点都继承自基类 `Node`，下面是主要的几种类型：

- **Expression**：表示一种表达式，如算术表达式、逻辑表达式等。这是一个抽象类，具体的表达式类型如 `Integer`、`Double`、`Char`、`String`、`Identifier`、`Call`、`BinaryOp`、`Assign` 等都继承自 `Expression` 类。
- **Statement**：表示一种语句，如声明语句、条件语句、循环语句等。这也是一个抽象类，具体的语句类型如 `Declaration`、`VariableDeclaration`、`FunctionDeclaration`、`ExpressionStatement`、`ReturnStatement`、`LoopStatement`、`ForStatement`、`WhileStatement`、`IfStatement` 等都继承自 `Statement` 类。
- **ConstVal**：表示常量值，如整数、浮点数、字符和字符串。这是一个抽象类，具体的常量类型如 `Integer`、`Double`、`Char`、`String` 都继承自 `ConstVal` 类。

原理

当编译器进行语法分析时，它会遍历源代码的每一个字符，根据预定的语法规则生成对应的节点，然后将这些节点链接起来，形成一棵抽象语法树。

在这个过程中，有两个关键的步骤：

1. **节点的生成**：根据源代码中的每一个语法结构，生成相应的节点。例如，对于一个整数常量，就生成一个 `Integer` 类的实例；对于一个赋值表达式，就生成一个 `Assign` 类的实例，然后将赋值运算符左边的表达式和右边的表达式作为子节点链接到这个 `Assign` 节点上。
2. **节点的链接**：将生成的节点按照它们在源代码中的关系链接起来，形成一棵树。例如，对于一个 `if` 语句，就将条件表达式的节点、`then` 代码块的节点和 `else` 代码块的节点（如果有的话）链接到一个 `IfStatement` 节点上。

实现

每个节点都有一个 `codeGen` 方法，该方法负责生成该节点对应的 LLVM IR 代码。例如，`Integer` 类的 `codeGen` 方法就生成一个表示整数常量的 LLVM IR 代码；`Assign` 类的 `codeGen` 方法则生成一个表示赋值操作的 LLVM IR 代码。

以下是一些关键节点的实现细节：

- `VarType`：表示一个变量的类型，如 `int`、`char`、`double`、`void`。
- `Integer`、`Double`、`Char`、`String`：表示整数、浮点数、字符和字符串常量。
- `Identifier`：表示一个变量名或函数名。
- `Call`：表示一个函数调用。包含函数名和实参列表。
- `BinaryOp`：表示一个二元操作。包含操作符和两个操作数。
- `Assign`：表示一个赋值操作。包含被赋值的变量和表达式。
- `ArrayElement` 和 `ArrayAssign`：表示数组元素的访问和赋值。
- `GetAddr` 和 `GetArrayAddr`：表示取地址操作。

- `Block`：表示一个代码块，包含多个语句。
- `Declaration`、`VariableDeclaration`、`FunctionDeclaration`：表示声明语句，包含变量声明和函数声明。
- `ExpressionStatement`、`ReturnStatement`：表示表达式语句和返回语句。
- `LoopStatement`、`ForStatement`、`WhileStatement`：表示循环语句，包含 for 循环和 while 循环。
- `IfStatement`：表示 if 条件语句。
- `SwitchStatement` 和 `CaseStatement`：表示 switch 语句和 case 语句。
- `BreakStatement` 和 `ContinueStatement`：表示 break 语句和 continue 语句。

语义分析

llvm工具

在我们的实验中，我们将使用LLVM工具链来实现自定义语言的编译。首先，我们将设计并实现一个编译器前端，将源代码解析为抽象语法树（AST）。然后，我们将AST转换为LLVM的中间表示（IR）。我们会使用LLVM提供的API和工具来生成、优化并最终编译IR。

LLVM IR是一种低级别的类型化编程语言，可以被LLVM优化并最终转换为机器码。在我们的实验中，每个AST节点类都有一个`codeGen`方法，这个方法负责生成对应该节点的LLVM IR。

在设计AST时，我们主要关注节点类的结构和职责，以及它们如何相互关联。每个节点类都代表源代码中的一种结构，如表达式、语句、类型声明等。每个类都包含了对应该结构的所有信息，如操作数、操作类型等。在`codeGen`方法中，每个节点类都知道如何生成自己的LLVM IR。

运行环境

我们构造了CodeGenerator类来保存上下文环境，进行ast中每个节点的IR的生成。

上下文环境

我们在Codegenerator.cpp中定义上下文变量与构造器变量：

- `llvm::LLVMContext`：这个类用于在LLVM中承载全局信息。不同的`LLVMContext`实例之间是隔离的，这意味着你可以在同一个程序中同时使用多个`LLVMContext`，而它们之间不会相互干扰。每个`LLVMContext`都可以包含许多不同的模块（`llvm::Module`）。
- `llvm::IRBuilder<>`：这个类用于生成LLVM IR指令。它提供了创建和插入LLVM IR指令的接口，使得创建IR变得更加简单。例如，如果想要创建一个加法指令，只需要调用`IRBuilder`的`CreateAdd`方法。`IRBuilder`会生成合适的IR指令，并插入到指定的基本块（`llvm::BasicBlock`）中。

符号表

定义符号表可以识别出重名的变量，隔离变量作用域

我们的符号表主要储存局部变量，全局变量采用Module->getGlobalVariable()检查变量是否重定义和全局变量的取用

符号表定义：

```
using SymbolTable = std::map<std::string, llvm::Value*>;
```

CodeGenerator类的实现

类`CodeGenerator`的主要成员包括一个`llvm::Module`对象，一个符号表栈`SymbolTableStack`，一个当前函数指针`CurrFunction`，以及两个基本块栈`ConditionBlockStack`和`EndBlockStack`。`Module`对象被用来保存生成的IR代码。符号表栈用于在代码生成过程中追踪变量和函数的定义和作用域。`CurrFunction`指向当前正在生成代码的函数，而`ConditionBlockStack`和`EndBlockStack`分别用于处理`continue`和`break`语句的跳转目标。

```
class CodeGenerator{
    public:
        using SymbolTable = std::map<std::string, llvm::Value*>;
        llvm::Module* Module;
        std::vector<SymbolTable*> SymbolTableStack;
        llvm::Function* CurrFunction;
        // condition block, when "continue;" jump here
        std::vector<llvm::BasicBlock*> ConditionBlockStack;
        // end block, when "break;" and "continue;" jump here
        std::vector<llvm::BasicBlock*> EndBlockStack;
        llvm::BasicBlock* TmpBB;
        llvm::BasicBlock* GlobalBB;
        llvm::Function* GlobalFunc;
        llvm::Function *printf,*scanf;
```

为了管理作用域，`CodeGenerator`类提供了`PushSymbolTable`和`PopSymbolTable`方法。每当代码生成进入一个新的作用域时，就调用`PushSymbolTable`方法在栈顶创建一个新的符号表。反之，每当离开一个作用域时，就调用`PopSymbolTable`方法弹出栈顶的符号表。通过这种方式，我们可以确保在每个作用域中，变量和函数的名称都是唯一的，并且在作用域之外，它们都是不可见的。

```
// 创建并推入一个空的符号表
void CodeGenerator::PushSymbolTable(void) {
    this->SymbolTableStack.push_back(new SymbolTable);
}

// 移除最后一个符号表
void CodeGenerator::PopSymbolTable(void) {
    if (this->SymbolTableStack.size() == 0) return;
    delete this->SymbolTableStack.back();
    this->SymbolTableStack.pop_back();
}
```

`FindVariable`方法可以在符号表中查找给定名称的变量，而`AddVariable`方法则用于将新的变量添加到当前的符号表中。这两个方法在处理变量声明和引用时非常有用。

```
// 查找变量
llvm::Value* CodeGenerator::FindVariable(std::string varName) {
    if (this->SymbolTableStack.empty()) return nullptr;
    for (auto tableIter = this->SymbolTableStack.rbegin(); tableIter != this-
>SymbolTableStack.rend(); ++tableIter) {
        auto foundVarIter = (*tableIter)->find(varName);
        if (foundVarIter != (*tableIter)->end())
            return foundVarIter->second;
    }
    return this->Module->getGlobalVariable(varName, true);
}

// 添加变量到当前符号表
bool CodeGenerator::AddVariable(std::string varName, llvm::Value* varValue){
    if (this->SymbolTableStack.empty()) return false;
    auto& topTable = *(this->SymbolTableStack.back());
    auto foundVarIter = topTable.find(varName);
    if (foundVarIter != topTable.end())
        return false;
    topTable[varName] = varValue;
    return true;
}
```

`CodeGenerator`类也包含了用于处理函数和循环的方法。在进入一个新的函数或循环时，会调用相应的`EnterFunction`或`EnterLoop`方法，从而设置正确的当前函数和跳转目标。相应地，当离开一个函数或循环时，会调用`LeaveFunction`或`LeaveLoop`方法。

```
// 设置当前函数
void CodeGenerator::EnterFunction(llvm::Function* Func) {
    this->CurrFunction = Func;
}
// 移除当前函数
void CodeGenerator::LeaveFunction(void) {
    this->CurrFunction = NULL;
}
// 获取当前函数
llvm::Function* CodeGenerator::GetCurrentFunction(void){
    return this->CurrFunction;
}
// 进入循环
void CodeGenerator::EnterLoop(llvm::BasicBlock* ConditionBB, llvm::BasicBlock*
EndBB){
    this->ConditionBlockStack.push_back(ConditionBB);
    this->EndBlockStack.push_back(EndBB);
```

```
}
// 离开循环
void CodeGenerator::LeaveLoop(void){
    if (this->ConditionBlockStack.size() != 0)
        this->ConditionBlockStack.pop_back();
    if (this->EndBlockStack.size() != 0)
        this->EndBlockStack.pop_back();
}
```

`CodeGenerator`类还包括生成LLVM IR形式的`printf`函数与`scanf`函数，这两个函数的实现在执行`gcc -no-pie -o main main.o`时将从 C 标准库中链接到程序中。`printf` 函数的实现并不在 `main.o` 中，而是在 C 标准库中。在链接阶段，链接器将找到这个实现，并将其和 `main.o` 中的其他代码一起链接，生成最后的可执行文件。

```
llvm::Function* GenPrintf(); //得到llvm形式的printf函数

llvm::Function* GenScanf();  //得到llvm形式的scanf函数
```

最后，`CodeGenerator`类还包括了`GenerateCode`和`GenIR`两个方法，用于实际生成IR代码。`GenerateCode`方法接受一个抽象语法树的根节点，并遍历该树来生成相应的IR代码。生成的代码被保存在`Module`对象中。`GenIR`方法则接收一个文件名参数，将`Module`对象的内容写入到指定的文件中。

```
void CodeGenerator::GenIR(const string& filename ){
    llvm::verifyModule(*this->Module, &llvm::outs());

    std::error_code EC;
    llvm::raw_fd_ostream dest(filename, EC);
    if (EC) {
        llvm::errs() << "Could not open file: " << EC.message();
        return;
    }
    this->Module->print(dest, nullptr);
}
```

ast.cpp的实现

此文件主要实现每个Node节点中对应的codeGen(CodeGenerator& context)方法，用于为每个节点生成中间代码。

类型

不同的数据类型的codegen函数，获取VarType类型对应的LLVM类型

```
llvm::Type* VarType::getLLVMType(){
    // 创建一个llvm::Type指针LLVMType
    llvm::Type *LLVMType;
    // 判断变量的类型
    switch (type) {
        case _Int: LLVMType = IRBuilder.getInt32Ty(); break;
        case _Char: LLVMType = IRBuilder.getInt8Ty(); break;
        case _Double: LLVMType = IRBuilder.getDoubleTy(); break;
        case _Void: LLVMType = IRBuilder.getVoidTy(); break;
        default: break;
    }
    // 如果变量有大小（可能是数组）
    if(this->size > 0){
        // 使用llvm::ArrayType::get生成数组类型，并赋值给LLVMType
        LLVMType = llvm::ArrayType::get(LLVMType, this->size);
    }
    // 返回LLVMType
    return LLVMType;
}
```

我们还实现了类型转换，在不同的场景下，我们对数据进行不同类型的处理。函数 `getCastInst()` 通过源数据类型和目标数据类型确定适当的类型转换操作。根据传入的两个参数类型，函数会返回对应的转换操作指令。函数 `typeCast()` 是执行类型转换的函数。首先，它会打印出源数据类型和目标数据类型，然后根据这两种类型调用适当的转换函数。这个函数对各种类型进行了处理。当源类型等于目标类型时，直接返回源值。若目标类型是 1-bit 整型（对应于布尔类型），根据源值的类型进行特定的类型转换。其他情况下，函数将根据 `getCastInst()` 的返回结果创建并返回一个新的类型转换指令。

```
//根据源数据类型和目标数据类型确定适当的转换操作类型。它处理不同的类型之间的转换，例如，从 float 到 int32，从 int32 到 float 等等。
llvm::Instruction::CastOps getCastInst(llvm::Type* src, llvm::Type* dst) {
    if (src == llvm::Type::getFloatTy(Context) && dst ==
llvm::Type::getInt32Ty(Context)) {
        return llvm::Instruction::FPToSI;
    }
    else if (src == llvm::Type::getInt32Ty(Context) && dst ==
llvm::Type::getFloatTy(Context)) {
        return llvm::Instruction::SIToFP;
    }
    else if (src == llvm::Type::getInt8Ty(Context) && dst ==
llvm::Type::getFloatTy(Context)) {
        return llvm::Instruction::UIToFP;
```

```
    }
    else if (src == llvm::Type::getInt8Ty(Context) && dst ==
llvm::Type::getInt32Ty(Context)) {
        return llvm::Instruction::ZExt;
    }
    else if (src == llvm::Type::getInt32Ty(Context) && dst ==
llvm::Type::getInt8Ty(Context)) {
        return llvm::Instruction::Trunc;
    }
    else {
        throw logic_error("[ERROR] Wrong typecast");
    }
}

//执行类型转换。它首先打印出源数据类型和目标数据类型，然后根据源数据类型和目标数据类型调用适当的转换
函数。
llvm::Value* typeCast(llvm::Value* src, llvm::Type* dst) {
    llvm::Type* type = src->getType();
    llvm::outs() << "TypeCast: Type of value " << *src << " is: ";
    type->print(llvm::outs());
    llvm::outs() << ", casting to " << *dst;
    llvm::outs() << "\n";

    if(src->getType() == dst){
        return src;
    }
    //如果目标类型（dst）是 1-bit 整型（对应于布尔类型），执行相应的类型转换。
    if (dst == llvm::Type::getInt1Ty(Context)){
        if (src->getType() == IRBuilder.getInt1Ty()){
            return src;
        }
        else if (src->getType()->isIntegerTy()){
            return IRBuilder.CreateICmpNE(src, llvm::ConstantInt::get(src-
>getType(), 0, true));
        }
        else if (src->getType()->isFloatingPointTy()){
            return IRBuilder.CreateFCmpONE(src, llvm::ConstantFP::get(src-
>getType(), 0.0));
        }
        else if (src->getType()->isPointerTy()){
            return IRBuilder.CreateICmpNE(IRBuilder.CreatePtrToInt(src,
IRBuilder.getInt64Ty()), IRBuilder.getInt64(0));
        }
        else {
            throw std::logic_error("Cannot cast to bool type.");
            return NULL;
        }
    }
    llvm::Instruction::CastOps op = getCastInst(src->getType(), dst);
    return IRBuilder.CreateCast(op, src, dst, "tmptypecast");
}
```

常量

通过IRBuilder获取llvm::Constant

```
// Integer 类型的代码生成函数。在控制台输出 "INT: 值"，并返回一个 int32 常量值。
llvm::Value* Integer::codeGen(CodeGenerator& context) {
    cout << "INT: " << value << endl;
    return IRBuilder.getInt32(value);
}

// Double 类型的代码生成函数。在控制台输出 "DOUBLE: 值"，并返回一个 double 常量值。
llvm::Value* Double::codeGen(CodeGenerator& context) {
    cout << "DOUBLE: " << value << endl;
    return llvm::ConstantFP::get(IRBuilder.getDoubleTy(), value);
}

// Char 类型的代码生成函数。在控制台输出 "CHAR: 值"，并返回一个 int8（字符型）常量值。
llvm::Value* Char::codeGen(CodeGenerator& context) {
    cout << "CHAR: " << value << endl;
    return IRBuilder.getInt8(value);
}

// String 类型的代码生成函数。首先在控制台输出 "STRING: 值"，然后处理字符串中的 "\n"（换行符）。
// 创建一个 LLVM 常量数据数组来存储处理后的字符串，并将其设置为全局变量。
// 最后，创建一个全局变量指针，并返回该指针。
llvm::Value* String::codeGen(CodeGenerator& context) {
    std::cout << "STRING: " << value << std::endl;

    std::string processedStr = value.substr(1, value.length() - 2);
    const std::string newLine = "\n";
    size_t newlinePos = processedStr.find("\\n");

    while(newlinePos != std::string::npos) {
        processedStr.replace(newlinePos, 2, newLine);
        newlinePos = processedStr.find("\\n");
    }

    llvm::Constant *constStr = llvm::ConstantDataArray::getString(Context,
processedStr);

    llvm::GlobalVariable *globalStrVar = new llvm::GlobalVariable(*(context.Module),
                                                                  constStr-
>getType(),
                                                                  true,

llvm::GlobalValue::PrivateLinkage,
                                                                  constStr,

"_Const_String_");
```

```
    std::vector<llvm::Value*> indices = {IRBuilder.getInt32(0),
IRBuilder.getInt32(0)};
    llvm::PointerType* ptrType = llvm::cast<llvm::PointerType>(globalStrVar-
>getType());
    llvm::Type* elementType = ptrType->getPointerElementType();

    llvm::Value *strPointer = IRBuilder.CreateInBoundsGEP(elementType, globalStrVar,
indices, "tmpstring");

    return strPointer;
}
```

标识符

`Identifier::codeGen()` 用于处理变量名的查找和对应值的加载。

我们调用 `context.FindVariable(name)` 来尝试在当前上下文中找到这个变量。如果找不到（返回 nullptr），则报告一个未声明的变量错误。

下面的部分处理了两种情况：变量是数组类型和变量是非数组类型。

1. 如果变量是数组类型，我们会创建一个指向数组的第一个元素的指针。这个指针由 `CreateConstGEP2_32` 函数创建，这个函数用于获取数组元素的地址。创建的指针被存储在 `res` 中。
2. 如果变量不是数组类型，我们会从内存中加载其值。这通过 `CreateLoad` 函数完成，它创建一个用于加载内存地址中的值的指令。加载的值被存储在 `res` 中。

最后，函数返回 `res`，这是一个指向我们获取或加载的值的 LLVM IR 值。

```
// 此函数通过名字查找变量并返回其 LLVM IR 代码。
// 如果变量是数组类型，会获取数组元素的指针。
// 如果变量不是数组，会从内存中加载其值。
llvm::Value* Identifier::codeGen(CodeGenerator& context) {
    cout << "IDENTIFIER: " << name << endl;
    llvm::Value* var = context.FindVariable(name);
    if(var == nullptr){
        std::cerr << "undeclared variable " << name << endl;
        return nullptr;
    }
    llvm::Type* tp = var->getType()->getPointerElementType();
    llvm::outs() << "identifier type:" << *tp << "\n";
    llvm::Value* res = nullptr;
    if(tp->isArrayTy()) {
        std::vector<llvm::Value*> indices = {IRBuilder.getInt32(0),
IRBuilder.getInt32(0)};
        res = IRBuilder.CreateConstGEP2_32(tp, var, 0, 0, "arrayPtr");
    }
    else {
        res = IRBuilder.CreateLoad(tp, var, "LoadInst");
    }
```

```
    return res;
}
```

对于数组成员的访问又有所不同，通过`FindVariable`函数查找数组，并获取其值。如果找不到该数组，则报错。调用`codeGen`函数生成索引的代码，并获取其值。检查索引值是否为整数类型。如果索引值不是整数类型，则报错。打印数组变量的类型信息。如果数组值的类型是指针类型，那么就加载指针所指向的值。否则，如果数组值的类型是数组类型，那么就添加索引值。如果数组值的类型既不是指针类型也不是数组类型，那么就报错。创建一个GEP（GetElementPointer）指令来计算元素的地址。最后，加载并返回元素的值。

```
// 对应如 a[2] 的数组元素访问
llvm::Value* ArrayElement::codeGen(CodeGenerator &context){
    std::cout << "正在访问数组元素，数组名为：" << identifier.name << "[]" << std::endl;

    // 查找数组并获取其值，如果找不到该数组则报错
    llvm::Value* arrayValue = context.FindVariable(identifier.name);
    if(arrayValue == nullptr){
        std::cerr << "尝试访问未声明的数组 " << identifier.name << std::endl;
        return nullptr;
    }

    std::cout << "访问的数组索引为：" << std::endl;
    llvm::Value* indexValue = index.codeGen(context);

    // 检查索引值是否为整数类型
    if (!indexValue->getType()->isIntegerTy()) {
        std::cerr << "数组索引值不是整数类型" << std::endl;
        return nullptr;
    }

    vector<llvm::Value*> indexList;

    // 打印数组变量的类型信息
    typePrint(arrayValue);

    // 如果 arrayValue 的类型是指针类型，则载入指针所指向的值
    if(arrayValue->getType()->getPointerElementType()->isPointerTy()) {
        arrayValue = IRBuilder.CreateLoad(arrayValue->getType()-
>getPointerElementType(), arrayValue);
        indexList = {indexValue};
    }
    // 如果 arrayValue 的类型是数组类型，则添加索引值
    else if(arrayValue->getType()->getPointerElementType()->isArrayTy()) {
        indexList = {IRBuilder.getInt32(0),indexValue};
    }
    // 如果 arrayValue 的类型既不是指针类型也不是数组类型，则报错
    else{
        std::cerr << "[] 只能用于指针或数组" << std::endl;
        return nullptr;
    }
```

```
    // 创建一个 GEP 指令来计算元素的地址
    llvm::Value* elePtr = IRBuilder.CreateInBoundsGEP(arrayValue->getType()-
>getPointerElementType(), arrayValue, llvm::ArrayRef<llvm::Value*>(indexList),
"tmparray");

    // 载入并返回元素的值
    return IRBuilder.CreateLoad(elePtr->getType()->getPointerElementType(), elePtr,
"tmpvar");
}
```

二元操作

核心部分是BinaryOp的codeGen函数，它处理二元运算并生成对应的LLVM IR代码。

函数主要分为以下几个部分：

1. 处理逻辑与（AND）和逻辑或（OR）运算：先检查操作数类型是否为布尔型，然后使用`llvm::IRBuilder::CreateAnd`或`llvm::IRBuilder::CreateOr`生成对应的LLVM IR指令。
2. 处理其他类型的运算：首先，确保操作数的类型一致，如果不一致则使用`unifyOperandTypes`函数进行类型转换。然后根据操作符类型（加、减、乘、除或比较操作）生成对应的LLVM IR指令。算术运算使用`determineBinOpInstruction`函数确定对应的LLVM IR指令，比较运算使用`generateComparisonInstruction`函数生成对应的LLVM IR指令。

```
// BinaryOp 的 codeGen 函数用于生成二元运算的 LLVM IR 代码
llvm::Value* BinaryOp::codeGen(CodeGenerator& context){
    std::cout << "Processing Binary Operation: " << op << std::endl;
    llvm::Value* leftOperand = lhs.codeGen(context);
    llvm::Value* rightOperand = rhs.codeGen(context);
    llvm::Instruction::BinaryOps binOpInstr;

    // 如果是逻辑与或逻辑或运算
    if(op == AND || op == OR){
        // 检查类型是否为布尔型
        if (leftOperand->getType() != llvm::Type::getInt1Ty(Context) ||
rightOperand->getType() != llvm::Type::getInt1Ty(Context)) {
            throw std::logic_error("Both operands should be of type bool for
'AND'/'OR' operations.");
        }
        return (op == AND) ? IRBuilder.CreateAnd(leftOperand, rightOperand,
"tmpAnd")
                           : IRBuilder.CreateOr(leftOperand, rightOperand, "tmpOR");
    }
    // 否则，处理其他类型的运算
    else{
        // 保证左右操作数类型一致
        if (leftOperand->getType() != rightOperand->getType()) {
            unifyOperandTypes(leftOperand, rightOperand);
```

```
        }
        // 处理算术运算
        if(op == PLUS || op == MINUS || op == MUL || op == DIV){
            binOpInstr = determineBinOpInstruction(op, leftOperand->getType());
            return llvm::BinaryOperator::Create(binOpInstr, leftOperand,
rightOperand, "", IRBuilder.GetInsertBlock());
        }
        // 处理比较运算
        else {
            return generateComparisonInstruction(op, leftOperand, rightOperand);
        }
    }
}
```

赋值

通过`FindVariable`函数查找变量，并获取其值。如果找不到该变量，则抛出错误。打印变量的类型信息，这是通过`typePrint`函数实现的。调用`codeGen`函数，生成表达式的代码，并获取其值。检查表达式的值与目标变量的类型是否相同。如果类型不同，则调用`typeCast`函数进行类型转换。如果无法进行类型转换，则抛出错误。最后，创建一个LLVM IR的store指令，将表达式的值存储到目标变量中。

```
// 利用yacc中的EQUAL来代表赋值操作
// 形式如：变量 = 表达式
llvm::Value* Assign::codeGen(CodeGenerator &context){
    std::cout << "处理赋值操作，变量名为：" << ident.name << std::endl;

    // 查找变量并获取其值，如果找不到该变量则抛出错误
    llvm::Value* targetValue = context.FindVariable(ident.name);
    if(targetValue == nullptr){
        std::cerr << "尝试赋值给未声明的变量 " << ident.name << std::endl;
        return nullptr;
    }

    // 打印变量的类型信息
    llvm::Type* targetType = targetValue->getType();
    typePrint(targetValue);

    // 生成表达式的代码并获取其值
    llvm::Value* exprValue = expr.codeGen(context);
    std::cout << "表达式的值为：";
    typePrint(exprValue);

    // 获取当前的代码块
    auto CurrentBlock = IRBuilder.GetInsertBlock();

    // 如果表达式的值与目标变量的类型不同，则进行类型转换
    if (exprValue->getType() != targetValue->getType()->getPointerElementType())
```

```
        exprValue = typeCast(exprValue, targetValue->getType()-
>getPointerElementType());

    // 如果无法进行类型转换，则抛出错误
    if (exprValue == nullptr) {
        throw std::domain_error("无法将表达式的值转换为目标变量的类型。");
        return nullptr;
    }

    // 在 LLVM IR 中创建一个 store 指令，将表达式的值存储到目标变量中
    IRBuilder.CreateStore(exprValue, targetValue);

    return exprValue;
}
```

数组的赋值有所不同，首先，它会尝试在上下文中寻找与数组名称对应的LLVM值。如果找不到，将会输出错误消息并返回`nullptr`。然后，它调用`index.codeGen`方法来为数组索引生成LLVM值。随后，它会检查数组值的类型。如果数组值是指针类型，那么就加载指针所指向的值。如果数组值的类型是数组类型，那么就使用索引值和0作为GEP（GetElementPointer）指令的索引参数。之后，使用GEP指令来计算待赋值元素的地址。接着，它调用`rhs.codeGen`方法来为赋值表达式生成LLVM值。再然后，如果待赋值元素的类型与赋值表达式的类型不同，那么就进行类型转换。最后，使用Store指令将赋值表达式的值存储到待赋值元素的地址中，并返回结果。

```
// ArrayAssign 函数用于对数组元素进行赋值操作，例如 a[2] = 5;
llvm::Value* ArrayAssign::codeGen(CodeGenerator &context){
    std::cout << "正在进行数组元素赋值操作，目标数组：" << identifier.name << "[]" <<
std::endl;

    // 在上下文中寻找数组变量，如果找不到，返回错误
    llvm::Value* arrVal = context.FindVariable(identifier.name);
    if(!arrVal){
        std::cerr << "无法找到数组 " << identifier.name << std::endl;
        return nullptr;
    }

    // 生成数组索引
    llvm::Value* idxVal = index.codeGen(context);
    vector<llvm::Value*> idxVec;

    // 打印目标数组的类型
    llvm::errs() << "目标数组的类型为：";
    arrVal->getType()->print(llvm::errs());
    llvm::errs() << '\n';

    // 如果 arrVal 是指针类型
    if(arrVal->getType()->getPointerElementType()->isPointerTy()) {
        arrVal = IRBuilder.CreateLoad(arrVal->getType()->getPointerElementType(),
arrVal);
        idxVec = {idxVal};
    }
    // 如果 arrVal 是数组类型
```

```
    else {
        idxVec = {IRBuilder.getInt32(0),idxVal};
    }

    // 使用 GEP 指令计算左值（待赋值元素）的地址
    llvm::Value* left =  IRBuilder.CreateInBoundsGEP(arrVal->getType()-
>getPointerElementType(), arrVal, llvm::ArrayRef<llvm::Value*>(idxVec), "tmpvar");

    // 生成右值（即赋值表达式）
    llvm::Value* right = rhs.codeGen(context);

    // 检查类型，并在需要时进行类型转换
    if (right->getType() != left->getType()->getPointerElementType())
        right = typeCast(right, left->getType()->getPointerElementType());

    // 执行赋值操作并返回结果
    return IRBuilder.CreateStore(right, left);
}
```

取地址

`GetAddr::codeGen` 和 `GetArrayAddr::codeGen` 用于生成 LLVM IR 代码以获取变量地址的。

在 `GetAddr::codeGen` 中：首先，它会在上下文的符号表和全局变量中查找给定名称的变量。如果找不到，将会输出错误消息并返回 `nullptr`。如果找到了变量，那么它将返回该变量的地址。

```
llvm::Value* GetAddr::codeGen(CodeGenerator &context){
    cout << "GetAddr : " << rhs.name << endl;
    // 在符号表和全局变量中查找
    llvm::Value* result = context.FindVariable(rhs.name);
    if(result == nullptr){
        cerr << "undeclared variable " << rhs.name << endl;
        return nullptr;
    }
    return result;
}
```

在 `GetArrayAddr::codeGen` 中：首先，它会在上下文中查找给定名称的数组。如果找不到，将会输出错误消息并返回 `nullptr`。然后，它会生成数组索引的 LLVM 值。根据数组值的类型，如果数组值是指针类型，那么就加载指针所指向的值并将索引值加入索引列表。如果数组值的类型是数组类型，那么就使用索引值和0作为 GEP（GetElementPointer）指令的索引参数。最后，使用 GEP 指令来计算元素的地址并返回。

```
llvm::Value* GetArrayAddr::codeGen(CodeGenerator &context){
    std::cout << "正在取得数组元素地址：" << rhs.name << "[]" << std::endl;

    llvm::Value* arrayVal = context.FindVariable(rhs.name);
```

```
    if(arrayVal == nullptr){
        std::cerr << "未声明的数组 " << rhs.name << std::endl;
        return nullptr;
    }

    llvm::Value* indexVal = index.codeGen(context);
    std::vector<llvm::Value*> idxList;

    // 如果目标是指针
    if(arrayVal->getType()->getPointerElementType()->isPointerTy()) {
        arrayVal = IRBuilder.CreateLoad(arrayVal->getType()-
>getPointerElementType(), arrayVal);
        idxList = {indexVal};
    }
    // 如果目标是数组
    else {
        idxList = {IRBuilder.getInt32(0),indexVal};
    }

    // 获取元素指针
    llvm::Value* elePtr = IRBuilder.CreateInBoundsGEP(arrayVal->getType()-
>getPointerElementType(), arrayVal, llvm::ArrayRef<llvm::Value*>(idxList),
"elePtr");
    return elePtr;
}
```

变量定义

VariableDeclaration类的codeGen方法负责生成 LLVM IR 代码以声明变量，包括全局变量和局部变量。在 `VariableDeclaration::codeGen` 函数中：它首先从变量声明中获取对应的 LLVM 类型，然后打印类型信息。判断当前是否在函数中，即判断声明的变量是全局变量还是局部变量。如果不在函数中，那么它会声明一个全局变量，首先检查该全局变量是否已经定义过，如果定义过则抛出异常。然后创建一个全局变量并为其设置初始值，初始值的创建过程是通过调用 `createArrayInitializer` 或 `createVariableInitializer` 函数实现的。如果在函数中，那么它会声明一个局部变量。首先在函数的入口块创建一个 alloca 指令以分配变量的存储空间，然后检查该变量是否在当前作用域内重复定义，如果重复定义则抛出异常。如果存在赋值表达式，那么就生成对应的赋值代码。最后，返回变量的地址。

```
// VariableDeclaration 的 codeGen 方法负责生成变量声明的 LLVM IR 代码
llvm::Value* VariableDeclaration::codeGen(CodeGenerator &context){
    // 首先从变量声明中获取对应的 LLVM 类型
    llvm::Type* VarType = type.getLLVMType();

    llvm::outs() << "变量声明：类型 ";
    VarType->print(llvm::outs());
    llvm::outs() << "\n";
    if(context.CurrFunction == NULL){
        // 全局变量
        std::cout << "声明全局变量 " << id.name << std::endl;
        context.ChangeToGlobalBB();
```

```
        // 检查全局变量是否重复定义
        if(context.Module->getGlobalVariable(id.name, true) != nullptr){
            throw std::logic_error("全局变量重复定义: " + id.name);
            return NULL;
        }
        // 创建全局变量
        auto Alloc = new llvm::GlobalVariable(*(context.Module), VarType, false,
llvm::Function::ExternalLinkage, 0, id.name);

        llvm::Constant* Initializer;
        // 如果是数组类型
        if(VarType->isArrayTy()){
            Initializer = createArrayInitializer(VarType);
        }
        // 不是数组类型
        else{
            Initializer = createVariableInitializer(VarType, context,
assignmentExpr);
        }
        Alloc->setInitializer(Initializer);
        context.ChangeToTmpBB();

        return Alloc;
    }
    else{
        // 局部变量
        std::cout << "声明局部变量 " << id.name << std::endl;
        llvm::Function *Func = context.CurrFunction;

        // 在函数的入口块创建 alloca 指令来为变量分配空间
        llvm::IRBuilder<> TmpB(&Func->getEntryBlock(), Func-
>getEntryBlock().begin());
        llvm::AllocaInst* Alloc = TmpB.CreateAlloca(VarType, 0, id.name);
        // 检查变量是否在当前作用域内重复定义
        if(!context.AddVariable(id.name, Alloc)) {
            Alloc->eraseFromParent();
            throw std::logic_error("局部变量重复定义: " + id.name);
            return NULL;
        }
        // 如果存在赋值表达式，那么就生成对应的赋值代码
        if (assignmentExpr) {
            Assign ass(id, *assignmentExpr);
            ass.codeGen(context);
        }
        return Alloc;
    }
}
```

`createArrayInitializer` 和 `createVariableInitializer` 函数

```
// 创建数组的初始化常量
```

```
llvm::Constant* VariableDeclaration::createArrayInitializer(llvm::Type* VarType){
    llvm::Type *eleType = VarType->getArrayElementType();

    std::vector<llvm::Constant*> constArrayEle;
    llvm::Constant* constEle = llvm::ConstantInt::get(eleType, 0);
    int size = VarType->getArrayNumElements();
    for (int i = 0; i < size; i++) {
        constArrayEle.push_back(constEle);
    }

    return llvm::ConstantArray::get(llvm::ArrayType::get(eleType, size),
constArrayEle);
}
// 创建变量的初始化常量
llvm::Constant* VariableDeclaration::createVariableInitializer(llvm::Type* VarType,
CodeGenerator& context, Expression* assignmentExpr){
    llvm::Constant* Initializer = llvm::ConstantInt::get(VarType, 0);
    if (assignmentExpr) {
        Assign ass(id, *assignmentExpr);
        Initializer = static_cast<llvm::Constant *>(typeCast(static_cast<llvm::Value
*>(ass.codeGen(context)), VarType));
    }

    return Initializer;
}
```

函数定义

`FunctionDeclaration` 类的 `codeGen` 负责生成 LLVM IR 代码以声明函数。

在 `FunctionDeclaration::codeGen` 函数中：它首先从函数声明中获取参数类型，然后获取返回类型，并据此生成函数类型。接着，根据函数类型创建 LLVM 函数并设置其链接类型和名称。创建一个新的基本块作为函数体的入口，并在此处插入指令。创建函数参数对应的存储空间，并生成函数体的代码。最后返回 `NULL`。

```
// FunctionDeclaration的codeGen方法负责生成函数声明的LLVM IR代码
llvm::Value* FunctionDeclaration::codeGen(CodeGenerator& context) {
    // 从函数声明中获取参数类型
    std::vector<llvm::Type*> ArgTypes = getArgTypes(context);

    // 获取返回类型
    llvm::Type* RetType = this->type.getLLVMType();

    // 获取函数类型
    llvm::FunctionType* FuncType = llvm::FunctionType::get(RetType, ArgTypes,
false);

    // 创建函数
    llvm::Function* Func = llvm::Function::Create(FuncType,
llvm::GlobalValue::ExternalLinkage, this->id.name, context.Module);
```

```
    // 创建一个新的基本块作为函数体的入口
    llvm::BasicBlock* FuncBlock = llvm::BasicBlock::Create(Context, "entry", Func);
    IRBuilder.SetInsertPoint(FuncBlock);

    // 创建参数对应的存储空间
    createArgAllocs(context, Func, ArgTypes);

    // 生成函数体的代码
    generateFuncBody(context, Func);

    return NULL;
}
```

在 `FunctionDeclaration::getArgTypes` 函数中，它从函数声明中获取参数类型，将参数类型保存在一个 `vector` 中，并返回该 `vector`。

```
// 从函数声明中获取参数类型
std::vector<llvm::Type*> FunctionDeclaration::getArgTypes(CodeGenerator& context) {
    std::vector<llvm::Type*> ArgTypes;
    for(auto i: this->arguments) {
        llvm::Type* tp = i->type.getLLVMType();
        if (!tp) {
            throw std::logic_error("使用未知类型定义函数 " + i->id.name);
            return {};
        }
        if (tp->isVoidTy()){
            if(arguments.size() > 1){
                throw std::logic_error("函数有多个参数类型为void.");
                return {};
            }
            else{
                break;
            }
        }
        // 如果函数参数类型为数组类型，则传递指向其元素的指针，而不是整个数组
        if (tp->isArrayTy()){
            tp = tp->getArrayElementType()->getPointerTo();
        }
        ArgTypes.push_back(tp);
    }
    return ArgTypes;
}
```

在 `FunctionDeclaration::createArgAllocs` 函数中，它创建函数参数对应的存储空间。首先，将符号表推到新的层级。然后，对于函数的每个参数，创建一条 alloca 指令以在函数的入口块中分配存储空间，并将参数存储到该位置。

```
// 创建函数参数对应的存储空间
void FunctionDeclaration::createArgAllocs(CodeGenerator& context, llvm::Function*
Func, std::vector<llvm::Type*>& ArgTypes) {
    context.PushSymbolTable();
    size_t Index = 0;
    for (auto ArgIter = Func->arg_begin(); ArgIter < Func->arg_end(); ArgIter++,
Index++) {
        // 在函数的入口块创建 alloca 指令来为参数分配空间
        llvm::IRBuilder<> TmpB(&Func->getEntryBlock(), Func-
>getEntryBlock().begin());
        auto Alloc = TmpB.CreateAlloca(ArgTypes[Index], 0, this->arguments[Index]-
>id.name);

        IRBuilder.CreateStore(ArgIter, Alloc);

        context.AddVariable(this->arguments[Index]->id.name, Alloc);
    }
}
```

在 `FunctionDeclaration::generateFuncBody` 函数中，它生成函数体的代码。首先，进入函数，将符号表推到新的层级。然后，生成函数体的代码。最后，从符号表中弹出当前层级并离开函数。

```
// 生成函数体的代码
void FunctionDeclaration::generateFuncBody(CodeGenerator& context, llvm::Function*
Func) {
    context.EnterFunction(Func);
    context.PushSymbolTable();
    this->block.codeGen(context);
    context.PopSymbolTable();
    context.LeaveFunction();
}
```

函数调用

`Call` 类的 `codeGen` 方法生成调用函数的 LLVM IR 代码。该方法首先检查是否在调用 `printf` 或 `scanf` 函数，这两个函数有特殊的处理方式。如果正在调用这两个函数之一，它将生成相应的调用代码并将结果存储在 `result` 变量中。如果 `result` 仍然为 `nullptr`，表示该函数既不是 `printf`，也不是 `scanf`，那么它将在 LLVM 模块中查找此函数。如果找不到该函数，将返回 `nullptr` 并打印错误消息。

如果找到了函数，那么它将生成对应的函数调用代码：它首先生成所有参数的代码并将结果存储在 `argValues` 向量中，然后使用这些参数创建函数调用指令。

```
// 这是 Call 类的 codeGen 函数，它负责生成调用函数的 LLVM IR 代码
llvm::Value* Call::codeGen(CodeGenerator& context){
    llvm::Value* result = nullptr;
```

```
    // 特别处理 printf 和 scanf 函数
    if(id.name == "printf"){
        result = generatePrintfCall(context, arguments);
    } else if(id.name == "scanf"){
        result = generateScanfCall(context, arguments);
    }

    // 如果不是 printf 或 scanf，就在模块中查找函数
    if (result == nullptr) {
        llvm::Function* function = context.Module->getFunction(id.name.c_str());
        if (function == nullptr) {
            std::cerr << "Function not found: " << id.name << std::endl;
            return nullptr;
        }

        std::cout << "Generating call to function: " << id.name << std::endl;

        std::vector<llvm::Value*> argValues;
        for(auto& arg : arguments){
            argValues.push_back(arg->codeGen(context));
        }

        result = llvm::CallInst::Create(function, llvm::makeArrayRef(argValues), "",
IRBuilder.GetInsertBlock());
        std::cout << "Created call to function: " << id.name << std::endl;
    }

    return result;
}
```

`ReturnStatement` 类的 `codeGen` 方法负责处理函数返回语句的 LLVM IR 代码生成。首先，我们在控制台打印出当前正在生成的语句类型。这个步骤主要是帮助我们进行调试和跟踪代码生成过程。然后，我们获取当前的函数上下文。如果没有函数上下文，就意味着我们遇到了一个没有函数主体的返回语句，这在语义上是错误的，所以我们抛出一个逻辑错误。接下来的步骤取决于返回语句是否包含一个表达式。如果没有表达式（即没有返回值），我们就生成返回 void 的代码。然而，如果函数的返回类型不是 void 类型，这就意味着我们应该有一个返回值，但是没有提供，所以我们再次抛出一个逻辑错误。如果返回语句有一个表达式，我们生成该表达式的代码，并进行类型转换，以确保返回的类型与函数声明的返回类型匹配。如果转换后的类型不匹配，我们会抛出一个逻辑错误。否则，我们将生成返回该值的 LLVM IR 代码。

```
// ReturnStatement的codeGen方法负责生成返回语句的LLVM IR代码
llvm::Value* ReturnStatement::codeGen(CodeGenerator &context){
    // 在控制台打印出当前正在生成的语句类型，帮助调试
    cout << "Generate Return Statement" << endl;

    // 获取当前的函数上下文，如果没有就报错
    llvm::Function* Func = context.GetCurrentFunction();
    if(Func == nullptr){
        throw std::logic_error("Return statement with no function body");
        return NULL;
```

```
    }

    // 如果没有表达式（即没有返回值），则生成返回void的代码；如果有表达式，则生成返回该表达式值的代
码
    if(expression == nullptr){
        if (Func->getReturnType()->isVoidTy()){
            return IRBuilder.CreateRetVoid();
        }else{
            throw std::logic_error("should return void");
            return NULL;
        }
    }else{
        // 生成表达式的代码，并进行类型转换，确保返回的类型与函数声明的返回类型匹配
        llvm::Value *ret = this->expression->codeGen(context);
        llvm::Value *RetVal = typeCast(ret, Func->getReturnType());
        if (RetVal == NULL) {
            throw std::logic_error("The type of return value doesn't match");
            return NULL;
        }else{
            return IRBuilder.CreateRet(RetVal);
        }
    }
}
```

分支语句

if语句

`IfStatement::codeGen` 方法用于生成 `if` 语句的 LLVM IR 代码。下面是对这个方法的详细解释：

首先，这个函数用 `this->condition.codeGen(context)` 生成条件表达式的代码，然后通过 `typeCast` 函数将结果转换为布尔类型。如果无法转换，就抛出一个异常。

然后，它创建三个基本块（`llvm::BasicBlock`）：`ThenBB`，`ElseBB` 和 `MergeBB`。这些基本块对应于 `if` 语句的 `then` 部分、`else` 部分以及 `if` 语句后面的代码。

接着，使用 `CreateCondBr` 创建一个条件分支指令：如果条件为真，就跳转到 `ThenBB`；否则跳转到 `ElseBB`。

然后在 `ThenBB` 中生成 `then` 部分的代码。在生成代码前后，使用 `context.PushSymbolTable()` 和 `context.PopSymbolTable()` 来维护符号表的状态。最后，如果 `ThenBB` 没有结束符（也就是说，没有 `return` 语句或者 `break`/`continue` 语句），那么就用 `TerminateBlockByBr(MergeBB)` 在 `ThenBB` 末尾添加一个到 `MergeBB` 的无条件分支。

在 `ElseBB` 中生成 `else` 部分的代码，方法与 `ThenBB` 类似。如果没有 `else` 部分，那么 `ElseBB` 就直接添加一个到 `MergeBB` 的无条件分支。

最后，如果 `MergeBB` 有至少一个前驱（也就是说，`ThenBB` 和 `ElseBB` 至少有一个执行到了末尾并跳转到了 `MergeBB`），那么就需要生成 `MergeBB` 的代码。这时将插入点设置为 `MergeBB`，在此之后的代码将被添加到 `MergeBB` 中。

```
// IfStatement::codeGen函数用于生成LLVM中间代码，表示if语句
// 其首先对条件进行求值，并创建对应的Then、Else和Merge基本块
// 然后在Then和Else基本块中分别生成相应的代码
llvm::Value* IfStatement::codeGen(CodeGenerator &context){
    // 打印一条消息，说明正在生成if语句
    cout << "create if statement " << endl;
    // 在这里，我们首先计算条件的值
    llvm::Value* Condition = this->condition.codeGen(context);
    // 然后我们检查条件是否可以转换为布尔值
    // 如果不能转换，我们抛出一个异常并返回NULL
    if (( Condition = typeCast(Condition, IRBuilder.getInt1Ty()) ) == NULL) {
        throw std::logic_error("condition type can not cast to bool");
        return NULL;
    }
    // 然后我们创建三个基本块：ThenBB，ElseBB和MergeBB
    // 这三个基本块分别用于存放if语句中的then部分，else部分，以及if语句结束后的代码
    llvm::Function* CurrentFunc = context.GetCurrentFunction();
    llvm::BasicBlock* ThenBB = llvm::BasicBlock::Create(Context, "Then");
    llvm::BasicBlock* ElseBB = llvm::BasicBlock::Create(Context, "Else");
    llvm::BasicBlock* MergeBB = llvm::BasicBlock::Create(Context, "Merge");

    // 然后我们创建一个条件分支指令，如果条件为真，就跳转到ThenBB，否则跳转到ElseBB
    IRBuilder.CreateCondBr(Condition, ThenBB, ElseBB);
    // 在ThenBB中生成then部分的代码
    CurrentFunc->getBasicBlockList().push_back(ThenBB);
    IRBuilder.SetInsertPoint(ThenBB);
    context.PushSymbolTable();
    this->next.codeGen(context);
    context.PopSymbolTable();
    TerminateBlockByBr(MergeBB);

    // 在ElseBB中生成else部分的代码，如果有的话
    CurrentFunc->getBasicBlockList().push_back(ElseBB);
    IRBuilder.SetInsertPoint(ElseBB);
    if(else_next){
        context.PushSymbolTable();
        this->else_next->codeGen(context);
        context.PopSymbolTable();
    }
    TerminateBlockByBr(MergeBB);

    // 如果MergeBB有至少一个前驱（即，它被使用了），我们就需要生成MergeBB的代码
    if (MergeBB->hasNPredecessorsOrMore(1)) {
        CurrentFunc->getBasicBlockList().push_back(MergeBB);
        IRBuilder.SetInsertPoint(MergeBB);
    }
    return NULL;
}
```

case语句

`SwitchStatement::codeGen` 和 `CaseStatement::codeGen` 函数用于生成 switch-case 语句的 LLVM IR 代码。

`SwitchStatement::codeGen` 首先获取当前的函数上下文和 switch 语句的比较值。然后它为每个 case 语句和相应的比较创建两个基本块的数组，同时为结束的基本块创建一个额外的位置。

然后，对于每个 case 语句，它首先设置了条件分支。如果比较值与当前 case 的值相等，就跳转到当前 case 的基本块，否则跳转到下一个比较基本块。然后在当前 case 的基本块中生成代码。为了处理 `break` 和 `continue` 语句，它使用 `context.EnterLoop` 和 `context.LeaveLoop` 函数将当前 case 块和结束块放入堆栈中。

`CaseStatement::codeGen` 函数只是生成 case 语句体的代码，然后在语句体结束后添加一个无条件分支到条件块。

`SwitchStatement::codeGen` 函数最后检查结束块是否有至少一个前驱，如果有，就生成结束块的代码。这是因为如果所有的 case 语句都有 `break` 语句，那么结束块可能没有前驱。

```
// 函数功能：生成Switch语句的代码
llvm::Value* SwitchStatement::codeGen(CodeGenerator &context){
    // 初始化
    llvm::Function* CurrentFunc = context.GetCurrentFunction();
    llvm::Value* Matches = this->matches.codeGen(context);

    // 为每个case和对比准备blocks
    std::vector<llvm::BasicBlock*> CaseBB(this->caseList.size() + 1);
    std::vector<llvm::BasicBlock*> ComparisonBB(this->caseList.size() + 1);
    for (int i = 0; i < this->caseList.size(); i++) {
        CaseBB[i] = llvm::BasicBlock::Create(Context, "Case" + std::to_string(i));
        ComparisonBB[i] = i ? llvm::BasicBlock::Create(Context, "Comparison" +
std::to_string(i)) : IRBuilder.GetInsertBlock();
    }
    CaseBB.back() = llvm::BasicBlock::Create(Context, "SwitchEnd");
    ComparisonBB.back() = CaseBB.back();

    // 为每个case生成代码
    context.PushSymbolTable();
    for (int i = 0; i < this->caseList.size(); i++) {
        // 准备分支
        CurrentFunc->getBasicBlockList().push_back(ComparisonBB[i]);
        IRBuilder.SetInsertPoint(ComparisonBB[i]);
        IRBuilder.CreateCondBr(
            CreateCmpEQ(Matches, this->caseList[i]->condition->codeGen(context)),
            CaseBB[i],
            ComparisonBB[i + 1]
        );

        // 生成case代码
        CurrentFunc->getBasicBlockList().push_back(CaseBB[i]);
        IRBuilder.SetInsertPoint(CaseBB[i]);
        context.EnterLoop(CaseBB[i + 1], CaseBB.back());
```

```
        this->caseList[i]->codeGen(context);
        context.LeaveLoop();
    }
    context.PopSymbolTable();

    // 完成"SwitchEnd"块
    if (CaseBB.back()->hasNPredecessorsOrMore(1)) {
        CurrentFunc->getBasicBlockList().push_back(CaseBB.back());
        IRBuilder.SetInsertPoint(CaseBB.back());
    }
    return NULL;
}

// 函数功能：生成Case语句的代码
llvm::Value* CaseStatement::codeGen(CodeGenerator &context){
    this->body.codeGen(context);
    TerminateBlockByBr(context.GetConditionBlock());
    return NULL;
}
```

循环语句

break 与 continue

`BreakStatement::codeGen` 函数首先获取当前的结束块，这是一个 `break` 语句应该跳转到的位置，然后在当前块中创建一个无条件分支到结束块。如果获取不到结束块（即 `break` 语句不在循环或 `switch` 语句中），它将抛出一个逻辑错误。因为在这种上下文中，`break` 语句是没有意义的。

```
// 函数功能：生成Break语句的代码
llvm::Value* BreakStatement::codeGen(CodeGenerator &context){
    cout << "break,codegen" << endl;
    llvm::BasicBlock* BreakPoint = context.GetEndBlock();
    if (BreakPoint)
        IRBuilder.CreateBr(BreakPoint);
    else
        throw std::logic_error("Break statement should only be used in loops or
switch statements.");
    return NULL;
}
```

`ContinueStatement::codeGen` 函数的功能和 `BreakStatement::codeGen` 类似，只是 `continue` 语句跳转的位置不同。`continue` 语句应该跳转到当前循环的条件块，所以它首先获取当前的条件块，然后在当前块中创建一个无条件分支到条件块。如果获取不到条件块（即 `continue` 语句不在循环中），它也将抛出一个逻辑错误。

```
// 函数功能：生成Continue语句的代码
llvm::Value* ContinueStatement::codeGen(CodeGenerator &context){
        llvm::BasicBlock* ContinuePoint = context.GetEndBlock();
        if (ContinuePoint)
            IRBuilder.CreateBr(ContinuePoint);
        else
            throw std::logic_error("Continue statement should only be used in loops
or switch statements.");
        return NULL;
}
```

for语句

在 `ForStatement::codeGen` 函数中：

- 创建四个基本块：一个用于检查循环条件（`ForCondBB`），一个用于执行循环体（`ForLoopBB`），一个用于执行每次循环结束后的操作（`ForTailBB`），最后一个用于在循环结束后执行（`ForEndBB`）。
- 在 `ForCondBB` 中生成初始化代码和条件代码，如果条件为真，将创建一个条件分支跳转到 `ForLoopBB`，否则跳转到 `ForEndBB`。
- 在 `ForLoopBB` 中生成循环体的代码，然后如果当前块未终止，将创建一个无条件分支跳转到 `ForTailBB`。
- 在 `ForTailBB` 中生成循环尾部的代码，并创建一个无条件分支跳转回 `ForCondBB`。
- 最后，将 `ForEndBB` 添加到函数的基本块列表中，并设置 `IRBuilder` 的插入点为 `ForEndBB`。

```
// ForStatement::codeGen方法用于将for语句转换为LLVM IR代码。
// 首先，它会创建一个条件检查块（ForCondBB）、一个循环体块（ForLoopBB）、一个循环尾部块
（ForTailBB）和一个循环结束块（ForEndBB）。
// 然后，它会在ForCondBB中生成对应于循环条件的代码，在ForLoopBB中生成对应于循环体的代码，以及在
ForTailBB中生成对应于循环尾部的代码。
llvm::Value* ForStatement::codeGen(CodeGenerator &context){
    // 获得当前函数
    llvm::Function* CurrentFunc = context.GetCurrentFunction();

    // 创建ForCondBB、ForLoopBB、ForTailBB和ForEndBB基本块
    llvm::BasicBlock* ForCondBB = llvm::BasicBlock::Create(Context, "ForCond");
    llvm::BasicBlock* ForLoopBB = llvm::BasicBlock::Create(Context, "ForLoop");
    llvm::BasicBlock* ForTailBB = llvm::BasicBlock::Create(Context, "ForTail");
    llvm::BasicBlock* ForEndBB = llvm::BasicBlock::Create(Context, "ForEnd");

    // 在ForCondBB中生成初始化代码
    context.PushSymbolTable();
    this->Initial.codeGen(context);
    TerminateBlockByBr(ForCondBB);

    // 在ForCondBB中生成条件代码
    CurrentFunc->getBasicBlockList().push_back(ForCondBB);
    IRBuilder.SetInsertPoint(ForCondBB);
```

```
    llvm::Value* Condition = this->condition.codeGen(context);

    // 检查条件是否可以转换为布尔值
    if (( Condition = typeCast(Condition, IRBuilder.getInt1Ty()) ) == NULL) {
        throw std::logic_error("condition type can not cast to bool");
        return NULL;
    }

    // 创建一个条件分支，如果条件为真，则跳转至ForLoopBB，否则跳转至ForEndBB
    IRBuilder.CreateCondBr(Condition, ForLoopBB, ForEndBB);

    // 在ForLoopBB中生成循环体的代码
    CurrentFunc->getBasicBlockList().push_back(ForLoopBB);
    IRBuilder.SetInsertPoint(ForLoopBB);
    context.EnterLoop(ForCondBB, ForEndBB);
    context.PushSymbolTable();
    this->LoopBody.codeGen(context);
    context.PopSymbolTable();
    context.LeaveLoop();

    // 如果当前块未终止，则创建一个无条件跳转至ForTailBB的分支
    TerminateBlockByBr(ForTailBB);

    // 在ForTailBB中生成循环尾部的代码，并跳转至ForCondBB
    CurrentFunc->getBasicBlockList().push_back(ForTailBB);
    IRBuilder.SetInsertPoint(ForTailBB);
    this->Tail.codeGen(context);
    IRBuilder.CreateBr(ForCondBB);

    // 在ForEndBB中生成后续代码
    CurrentFunc->getBasicBlockList().push_back(ForEndBB);
    IRBuilder.SetInsertPoint(ForEndBB);
    context.PopSymbolTable();

    return NULL;
}
```

while语句

在 `WhileStatement::codeGen` 函数中：

- 创建三个基本块：一个用于检查循环条件（`WhileCondBB`），一个用于执行循环体（`WhileLoopBB`），最后一个用于在循环结束后执行（`WhileEndBB`）。
- 创建一个无条件分支跳转到 `WhileCondBB`，并在 `WhileCondBB` 中生成条件代码，如果条件为真，将创建一个条件分支跳转到 `WhileLoopBB`，否则跳转到 `WhileEndBB`。
- 在 `WhileLoopBB` 中生成循环体的代码，然后如果当前块未终止，将创建一个无条件分支跳转回 `WhileCondBB`。
- 最后，将 `WhileEndBB` 添加到函数的基本块列表中，并设置 `IRBuilder` 的插入点为 `WhileEndBB`。

```
// WhileStatement::codeGen方法用于将while语句转换为LLVM IR代码。
// 首先，它会创建一个条件检查块（WhileCondBB）、一个循环体块（WhileLoopBB）和一个循环结束块（WhileEndBB）。
// 然后，它会在WhileCondBB中生成对应于循环条件的代码，并在WhileLoopBB中生成对应于循环体的代码。
llvm::Value* WhileStatement::codeGen(CodeGenerator &context){
    // 获得当前函数
    llvm::Function* CurrentFunc = context.GetCurrentFunction();

    // 创建WhileCondBB、WhileLoopBB和WhileEndBB基本块
    llvm::BasicBlock* WhileCondBB = llvm::BasicBlock::Create(Context, "WhileCond");
    llvm::BasicBlock* WhileLoopBB = llvm::BasicBlock::Create(Context, "WhileLoop");
    llvm::BasicBlock* WhileEndBB = llvm::BasicBlock::Create(Context, "WhileEnd");

    // 创建一个无条件跳转至WhileCondBB的分支
    IRBuilder.CreateBr(WhileCondBB);

    // 在WhileCondBB中生成条件代码
    CurrentFunc->getBasicBlockList().push_back(WhileCondBB);
    IRBuilder.SetInsertPoint(WhileCondBB);
    llvm::Value* Condition = this->condition.codeGen(context);

    // 检查条件是否可以转换为布尔值
    if (( Condition = typeCast(Condition, IRBuilder.getInt1Ty()) ) == NULL) {
        throw std::logic_error("condition type can not cast to bool");
        return NULL;
    }

    // 创建一个条件分支，如果条件为真，则跳转至WhileLoopBB，否则跳转至WhileEndBB
    IRBuilder.CreateCondBr(Condition, WhileLoopBB, WhileEndBB);

    // 在WhileLoopBB中生成循环体的代码
    CurrentFunc->getBasicBlockList().push_back(WhileLoopBB);
    IRBuilder.SetInsertPoint(WhileLoopBB);
    context.EnterLoop(WhileCondBB, WhileEndBB);
    context.PushSymbolTable();
    this->LoopBody.codeGen(context);
    context.PopSymbolTable();
    context.LeaveLoop();

    // 如果当前块未终止，则创建一个无条件跳转至WhileCondBB的分支
    TerminateBlockByBr(WhileCondBB);

    // 在WhileEndBB中生成后续代码
    CurrentFunc->getBasicBlockList().push_back(WhileEndBB);
    IRBuilder.SetInsertPoint(WhileEndBB);

    return NULL;
}
```

代码优化

我们首先根据用户输入的参数判断是否需要进行代码优化。如果输入参数中包含优化选项，如"-O0", "-O1", "-O2", "-O3", "-Os", 或"-Oz"，我们就从其中提取出优化级别并根据此进行相应的优化操作。

在我们的实验中，我们创建了一个LLVM的优化器实例，并设置其优化级别。之后，我们将这个优化器应用到我们生成的代码模块上，从而达到了优化代码的目标。这种方式的优化是全局的，能够从整体上提升代码的执行效率。

我们创建了一个LLVM的优化器实例，并设置其优化级别。之后，我们将这个优化器应用到我们生成的代码模块上，从而达到了优化代码的目标。

```
auto it_O = argMap.find("O");
if(it_O != argMap.end()){
    unsigned optLevel = stoi(it_O->second);
    // 创建优化器
    llvm::legacy::PassManager pm;
    llvm::PassManagerBuilder pmb;
    pmb.OptLevel = optLevel;
    pmb.populateModulePassManager(pm);
    // 运行优化器
    pm.run(*(generator->Module));
}
```

代码生成

本节将详细解释编译器的代码生成阶段，特别是如何生成RISC-V架构的汇编代码。代码生成阶段是编译过程中的最后一步，该阶段负责将中间表示（IR）转换为目标机器语言。在我们的编译器中，这是通过使用LLVM的llc命令实现的。

在实现代码中，首先处理了"-s"选项。这个选项允许用户指定输出的汇编文件的名称。如果用户没有在文件名的结尾添加".s"，编译器将自动添加。这个设计是为了确保生成的汇编文件总是有正确的扩展名。

然后，通过将文件名插入到llc命令模板中，生成了一个新的命令。llc是LLVM中的一个工具，它可以将LLVM的中间表示转换为汇编代码。"-march=riscv32"选项指定了目标架构，即RISC-V。"-filetype=asm"选项指定了输出的文件类型，即汇编代码。"-o"选项后面跟着的是输出文件的名称。

最后，使用system函数执行了生成的llc命令。这样，就完成了从LLVM IR到RISC-V汇编代码的转换。

```
//处理-s选项
    auto it_s = argMap.find("s");
    if(it_s != argMap.end()){
        std::string OutputAssemblyFile = it_s->second;
        if(OutputAssemblyFile.length() <= 2 ||
OutputAssemblyFile.substr(OutputAssemblyFile.length() - 2) != ".s")
        {
            OutputAssemblyFile = OutputAssemblyFile + ".s";
        }

        // Convert LLVM IR to RISC-V assembly code
        std::string command = "llc -march=riscv32 " + OutputObjectFile + " -
filetype=asm -o " + OutputAssemblyFile;
        system(command.c_str());
    }
```

测试案例

数据类型测试

内置类型测试

- 测试代码

```
int main() {
    int i = 10;
    char c = 'a';
    double d = 3.14;

    printf("Integer: %d\n", i);
    printf("Char: %c\n", c);
    printf("Double: %lf\n", d);

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [13 x i8] c"Integer: %d\0A\00"
@_Const_String_.1 = private constant [10 x i8] c"Char: %c\0A\00"
@_Const_String_.2 = private constant [13 x i8] c"Double: %lf\0A\00"

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

```
declare i32 @scanf(...)

define i32 @main() {
entry:
  %d = alloca double, align 8
  %c = alloca i8, align 1
  %i = alloca i32, align 4
  store i32 10, i32* %i, align 4
  store i8 97, i8* %c, align 1
  store double 3.140000e+00, double* %d, align 8
  %LoadInst = load i32, i32* %i, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x i8], [13 x
i8]* @_Const_String_, i32 0, i32 0), i32 %LoadInst)
  %LoadInst1 = load i8, i8* %c, align 1
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10
x i8]* @_Const_String_.1, i32 0, i32 0), i8 %LoadInst1)
  %LoadInst3 = load double, double* %d, align 8
  %printf4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x i8], [13
x i8]* @_Const_String_.2, i32 0, i32 0), double %LoadInst3)
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -32
    .cfi_def_cfa_offset 32
    sw  ra, 28(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    li  a0, 10
    sw  a0, 8(sp)
    li  a0, 97
    sb  a0, 15(sp)
    lui a0, 262290
    addi    a0, a0, -328
    sw  a0, 20(sp)
    lui a0, 335544
    addi    a0, a0, 1311
    sw  a0, 16(sp)
    lui a0, %hi(.L_Const_String_)
```

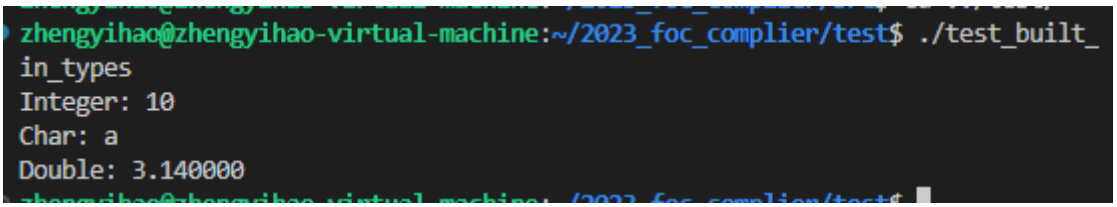
```
    addi    a0, a0, %lo(.L_Const_String_)
    li  a1, 10
    call    printf@plt
    lb  a1, 15(sp)
    lui a0, %hi(.L_Const_String_.1)
    addi    a0, a0, %lo(.L_Const_String_.1)
    call    printf@plt
    lw  a3, 20(sp)
    lw  a2, 16(sp)
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
    call    printf@plt
    li  a0, 0
    lw  ra, 28(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 32
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
.L_Const_String_:
    .asciz  "Integer: %d\n"
    .size   .L_Const_String_, 13

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
.L_Const_String_.1:
    .asciz  "Char: %c\n"
    .size   .L_Const_String_.1, 10

    .type   .L_Const_String_.2,@object      # @_Const_String_.2
.L_Const_String_.2:
    .asciz  "Double: %lf\n"
    .size   .L_Const_String_.2, 13

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

- 运行结果



数组类型测试

- 测试代码

```
int main() {
    int i;
    int intArr[5];
    char charArr[5];
    double doubleArr[5];
    for(i = 0; i < 5; i=i+1) {
        intArr[i] = i + 1;
    }
    charArr[0] = 'a';
    charArr[1] = 'b';
    charArr[2] = 'c';
    charArr[3] = 'd';
    charArr[4] = 'e';

    doubleArr[0] = 1.1;
    doubleArr[1] = 2.2;
    doubleArr[2] = 3.3;
    doubleArr[3] = 4.4;
    doubleArr[4] = 5.5;

    printf("Integer array:\n");
    for(i=0; i<5;  i=i+1){
        printf("%d ", intArr[i]);
    }
    printf("\n");

    printf("Character array:\n");
    for( i=0; i<5;  i=i+1){
        printf("%c ", charArr[i]);
    }
    printf("\n");

    printf("Double array:\n");
    for( i=0; i<5;  i=i+1){
        printf("%lf ", doubleArr[i]);
    }
    printf("\n");

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"
```

```
@_Const_String_ = private constant [16 x i8] c"Integer array:\0A\00"
@_Const_String_.1 = private constant [4 x i8] c"%d \00"
@_Const_String_.2 = private constant [2 x i8] c"\0A\00"
@_Const_String_.3 = private constant [18 x i8] c"Character array:\0A\00"
@_Const_String_.4 = private constant [4 x i8] c"%c \00"
@_Const_String_.5 = private constant [2 x i8] c"\0A\00"
@_Const_String_.6 = private constant [15 x i8] c"Double array:\0A\00"
@_Const_String_.7 = private constant [5 x i8] c"%lf \00"
@_Const_String_.8 = private constant [2 x i8] c"\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %doubleArr = alloca [5 x double], align 8
  %charArr = alloca [5 x i8], align 1
  %intArr = alloca [5 x i32], align 4
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  br label %ForCond

ForCond:                                          ; preds = %ForTail, %entry
  %LoadInst = load i32, i32* %i, align 4
  %icmptmp = icmp slt i32 %LoadInst, 5
  br i1 %icmptmp, label %ForLoop, label %ForEnd

ForLoop:                                          ; preds = %ForCond
  %LoadInst1 = load i32, i32* %i, align 4
  %tmpvar = getelementptr inbounds [5 x i32], [5 x i32]* %intArr, i32 0, i32
%LoadInst1
  %LoadInst2 = load i32, i32* %i, align 4
  %0 = add i32 %LoadInst2, 1
  store i32 %0, i32* %tmpvar, align 4
  br label %ForTail

ForTail:                                          ; preds = %ForLoop
  %LoadInst3 = load i32, i32* %i, align 4
  %1 = add i32 %LoadInst3, 1
  store i32 %1, i32* %i, align 4
  br label %ForCond

ForEnd:                                           ; preds = %ForCond
  %tmpvar4 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32 0
  store i8 97, i8* %tmpvar4, align 1
  %tmpvar5 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32 1
  store i8 98, i8* %tmpvar5, align 1
  %tmpvar6 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32 2
  store i8 99, i8* %tmpvar6, align 1
  %tmpvar7 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32 3
  store i8 100, i8* %tmpvar7, align 1
```

```
  %tmpvar8 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32 4
  store i8 101, i8* %tmpvar8, align 1
  %tmpvar9 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32 0,
i32 0
  store double 1.100000e+00, double* %tmpvar9, align 8
  %tmpvar10 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32 0,
i32 1
  store double 2.200000e+00, double* %tmpvar10, align 8
  %tmpvar11 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32 0,
i32 2
  store double 3.300000e+00, double* %tmpvar11, align 8
  %tmpvar12 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32 0,
i32 3
  store double 4.400000e+00, double* %tmpvar12, align 8
  %tmpvar13 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32 0,
i32 4
  store double 5.500000e+00, double* %tmpvar13, align 8
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16 x i8], [16 x
i8]* @_Const_String_, i32 0, i32 0))
  store i32 0, i32* %i, align 4
  br label %ForCond14

ForCond14:                                        ; preds = %ForTail21, %ForEnd
  %LoadInst15 = load i32, i32* %i, align 4
  %icmptmp16 = icmp slt i32 %LoadInst15, 5
  br i1 %icmptmp16, label %ForLoop17, label %ForEnd23

ForLoop17:                                        ; preds = %ForCond14
  %LoadInst18 = load i32, i32* %i, align 4
  %tmparray = getelementptr inbounds [5 x i32], [5 x i32]* %intArr, i32 0, i32
%LoadInst18
  %tmpvar19 = load i32, i32* %tmparray, align 4
  %printf20 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
i8]* @_Const_String_.1, i32 0, i32 0), i32 %tmpvar19)
  br label %ForTail21

ForTail21:                                        ; preds = %ForLoop17
  %LoadInst22 = load i32, i32* %i, align 4
  %2 = add i32 %LoadInst22, 1
  store i32 %2, i32* %i, align 4
  br label %ForCond14

ForEnd23:                                         ; preds = %ForCond14
  %printf24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x
i8]* @_Const_String_.2, i32 0, i32 0))
  %printf25 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([18 x i8], [18
x i8]* @_Const_String_.3, i32 0, i32 0))
  store i32 0, i32* %i, align 4
  br label %ForCond26

ForCond26:                                        ; preds = %ForTail34, %ForEnd23
  %LoadInst27 = load i32, i32* %i, align 4
```

```
  %icmptmp28 = icmp slt i32 %LoadInst27, 5
  br i1 %icmptmp28, label %ForLoop29, label %ForEnd36

ForLoop29:                                        ; preds = %ForCond26
  %LoadInst30 = load i32, i32* %i, align 4
  %tmparray31 = getelementptr inbounds [5 x i8], [5 x i8]* %charArr, i32 0, i32
%LoadInst30
  %tmpvar32 = load i8, i8* %tmparray31, align 1
  %printf33 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
i8]* @_Const_String_.4, i32 0, i32 0), i8 %tmpvar32)
  br label %ForTail34

ForTail34:                                        ; preds = %ForLoop29
  %LoadInst35 = load i32, i32* %i, align 4
  %3 = add i32 %LoadInst35, 1
  store i32 %3, i32* %i, align 4
  br label %ForCond26

ForEnd36:                                         ; preds = %ForCond26
  %printf37 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x
i8]* @_Const_String_.5, i32 0, i32 0))
  %printf38 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([15 x i8], [15
x i8]* @_Const_String_.6, i32 0, i32 0))
  store i32 0, i32* %i, align 4
  br label %ForCond39

ForCond39:                                        ; preds = %ForTail47, %ForEnd36
  %LoadInst40 = load i32, i32* %i, align 4
  %icmptmp41 = icmp slt i32 %LoadInst40, 5
  br i1 %icmptmp41, label %ForLoop42, label %ForEnd49

ForLoop42:                                        ; preds = %ForCond39
  %LoadInst43 = load i32, i32* %i, align 4
  %tmparray44 = getelementptr inbounds [5 x double], [5 x double]* %doubleArr, i32
0, i32 %LoadInst43
  %tmpvar45 = load double, double* %tmparray44, align 8
  %printf46 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x
i8]* @_Const_String_.7, i32 0, i32 0), double %tmpvar45)
  br label %ForTail47

ForTail47:                                        ; preds = %ForLoop42
  %LoadInst48 = load i32, i32* %i, align 4
  %4 = add i32 %LoadInst48, 1
  store i32 %4, i32* %i, align 4
  br label %ForCond39

ForEnd49:                                         ; preds = %ForCond39
  %printf50 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x
i8]* @_Const_String_.8, i32 0, i32 0))
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -96
    .cfi_def_cfa_offset 96
    sw  ra, 92(sp)                      # 4-byte Folded Spill
    sw  s0, 88(sp)                      # 4-byte Folded Spill
    sw  s1, 84(sp)                      # 4-byte Folded Spill
    sw  s2, 80(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    .cfi_offset s0, -8
    .cfi_offset s1, -12
    .cfi_offset s2, -16
    sw  zero, 8(sp)
    li  a0, 4
    addi    a1, sp, 12
    lw  a2, 8(sp)
    blt a0, a2, .LBB0_2
.LBB0_1:                                # %ForLoop
                                        # =>This Inner Loop Header: Depth=1
    lw  a2, 8(sp)
    slli    a3, a2, 2
    add a3, a1, a3
    addi    a2, a2, 1
    sw  a2, 0(a3)
    sw  a2, 8(sp)
    lw  a2, 8(sp)
    bge a0, a2, .LBB0_1
.LBB0_2:                                # %ForEnd
    li  a0, 97
    sb  a0, 35(sp)
    li  a0, 98
    sb  a0, 36(sp)
    li  a0, 99
    sb  a0, 37(sp)
    li  a0, 100
    sb  a0, 38(sp)
    li  a0, 101
    sb  a0, 39(sp)
    lui a0, 261914
    addi    a0, a0, -1639
```

```
    sw  a0, 44(sp)
    lui a0, 629146
    addi    a0, a0, -1638
    sw  a0, 40(sp)
    lui a1, 262170
    addi    a1, a1, -1639
    sw  a1, 52(sp)
    sw  a0, 48(sp)
    lui a1, 262310
    addi    a1, a1, 1638
    sw  a1, 60(sp)
    lui a1, 419430
    addi    a1, a1, 1638
    sw  a1, 56(sp)
    lui a1, 262426
    addi    a1, a1, -1639
    sw  a1, 68(sp)
    sw  a0, 64(sp)
    lui a0, 262496
    sw  a0, 76(sp)
    sw  zero, 72(sp)
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    call    printf@plt
    sw  zero, 8(sp)
    li  s1, 4
    addi    s2, sp, 12
    lui a0, %hi(.L_Const_String_.1)
    addi    s0, a0, %lo(.L_Const_String_.1)
    lw  a0, 8(sp)
    blt s1, a0, .LBB0_4
.LBB0_3:                                # %ForLoop17
                                        # =>This Inner Loop Header: Depth=1
    lw  a0, 8(sp)
    slli    a0, a0, 2
    add a0, s2, a0
    lw  a1, 0(a0)
    mv  a0, s0
    call    printf@plt
    lw  a0, 8(sp)
    addi    a0, a0, 1
    sw  a0, 8(sp)
    lw  a0, 8(sp)
    bge s1, a0, .LBB0_3
.LBB0_4:                                # %ForEnd23
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
    call    printf@plt
    lui a0, %hi(.L_Const_String_.3)
    addi    a0, a0, %lo(.L_Const_String_.3)
    call    printf@plt
    sw  zero, 8(sp)
```

```
    li  s1, 4
    addi    s2, sp, 35
    lui a0, %hi(.L_Const_String_.4)
    addi    s0, a0, %lo(.L_Const_String_.4)
    lw  a0, 8(sp)
    blt s1, a0, .LBB0_6
.LBB0_5:                                # %ForLoop29
                                        # =>This Inner Loop Header: Depth=1
    lw  a0, 8(sp)
    add a0, s2, a0
    lb  a1, 0(a0)
    mv  a0, s0
    call    printf@plt
    lw  a0, 8(sp)
    addi    a0, a0, 1
    sw  a0, 8(sp)
    lw  a0, 8(sp)
    bge s1, a0, .LBB0_5
.LBB0_6:                                # %ForEnd36
    lui a0, %hi(.L_Const_String_.5)
    addi    a0, a0, %lo(.L_Const_String_.5)
    call    printf@plt
    lui a0, %hi(.L_Const_String_.6)
    addi    a0, a0, %lo(.L_Const_String_.6)
    call    printf@plt
    sw  zero, 8(sp)
    li  s1, 4
    addi    s2, sp, 40
    lui a0, %hi(.L_Const_String_.7)
    addi    s0, a0, %lo(.L_Const_String_.7)
    lw  a0, 8(sp)
    blt s1, a0, .LBB0_8
.LBB0_7:                                # %ForLoop42
                                        # =>This Inner Loop Header: Depth=1
    lw  a0, 8(sp)
    slli    a0, a0, 3
    add a0, s2, a0
    lw  a2, 0(a0)
    ori a0, a0, 4
    lw  a3, 0(a0)
    mv  a0, s0
    call    printf@plt
    lw  a0, 8(sp)
    addi    a0, a0, 1
    sw  a0, 8(sp)
    lw  a0, 8(sp)
    bge s1, a0, .LBB0_7
.LBB0_8:                                # %ForEnd49
    lui a0, %hi(.L_Const_String_.8)
    addi    a0, a0, %lo(.L_Const_String_.8)
    call    printf@plt
    li  a0, 0
```

```
    lw  ra, 92(sp)                      # 4-byte Folded Reload
    lw  s0, 88(sp)                      # 4-byte Folded Reload
    lw  s1, 84(sp)                      # 4-byte Folded Reload
    lw  s2, 80(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 96
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
.L_Const_String_:
    .asciz  "Integer array:\n"
    .size   .L_Const_String_, 16

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
.L_Const_String_.1:
    .asciz  "%d "
    .size   .L_Const_String_.1, 4

    .type   .L_Const_String_.2,@object      # @_Const_String_.2
.L_Const_String_.2:
    .asciz  "\n"
    .size   .L_Const_String_.2, 2

    .type   .L_Const_String_.3,@object      # @_Const_String_.3
    .p2align    4
.L_Const_String_.3:
    .asciz  "Character array:\n"
    .size   .L_Const_String_.3, 18

    .type   .L_Const_String_.4,@object      # @_Const_String_.4
.L_Const_String_.4:
    .asciz  "%c "
    .size   .L_Const_String_.4, 4

    .type   .L_Const_String_.5,@object      # @_Const_String_.5
.L_Const_String_.5:
    .asciz  "\n"
    .size   .L_Const_String_.5, 2

    .type   .L_Const_String_.6,@object      # @_Const_String_.6
.L_Const_String_.6:
    .asciz  "Double array:\n"
    .size   .L_Const_String_.6, 15

    .type   .L_Const_String_.7,@object      # @_Const_String_.7
.L_Const_String_.7:
    .asciz  "%lf "
    .size   .L_Const_String_.7, 5
```

```
    .type   .L_Const_String_.8,@object      # @_Const_String_.8
.L_Const_String_.8:
    .asciz  "\n"
    .size   .L_Const_String_.8, 2

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

- 运行结果

```
zhengyihao@zhengyihao-virtual-machine:~/2023_foc_complier/test$ ./test_array_
types
Integer array:
1 2 3 4 5
Character array:
a b c d e
Double array:
1.100000 2.200000 3.300000 4.400000 5.500000
```

运算测试

- 测试代码

```c
int main() {
    int a = 10;
    int b = 5;

    // Arithmetic Operations
    printf("Addition: %d + %d = %d\n", a, b, a + b);
    printf("Subtraction: %d - %d = %d\n", a, b, a - b);
    printf("Multiplication: %d * %d = %d\n", a, b, a * b);
    printf("Division: %d / %d = %d\n", a, b, a / b);


    // // Logical Operations
    printf("Logical AND: (a == b) && (a != b) = %d\n", (a == b) && (a != b));
    printf("Logical OR: (a == b) || (a != b) = %d\n", (a == b) || (a != b));

    // // Comparison Operations
    printf("Equals: a == b -> %d\n", a == b);
    printf("Not Equals: a != b -> %d\n", a != b);
    printf("Less Than: a < b -> %d\n", a < b);
    printf("Greater Than: a > b -> %d\n", a > b);
    printf("Less Than or Equals: a <= b -> %d\n", a <= b);
    printf("Greater Than or Equals: a >= b -> %d\n", a >= b);

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [24 x i8] c"Addition: %d + %d = %d\0A\00"
@_Const_String_.1 = private constant [27 x i8] c"Subtraction: %d - %d = %d\0A\00"
@_Const_String_.2 = private constant [30 x i8] c"Multiplication: %d * %d = %d\0A\00"
@_Const_String_.3 = private constant [24 x i8] c"Division: %d / %d = %d\0A\00"
@_Const_String_.4 = private constant [40 x i8] c"Logical AND: (a == b) && (a != b) =
%d\0A\00"
@_Const_String_.5 = private constant [39 x i8] c"Logical OR: (a == b) || (a != b) =
%d\0A\00"
@_Const_String_.6 = private constant [22 x i8] c"Equals: a == b -> %d\0A\00"
@_Const_String_.7 = private constant [26 x i8] c"Not Equals: a != b -> %d\0A\00"
@_Const_String_.8 = private constant [24 x i8] c"Less Than: a < b -> %d\0A\00"
@_Const_String_.9 = private constant [27 x i8] c"Greater Than: a > b -> %d\0A\00"
@_Const_String_.10 = private constant [35 x i8] c"Less Than or Equals: a <= b ->
%d\0A\00"
@_Const_String_.11 = private constant [38 x i8] c"Greater Than or Equals: a >= b ->
%d\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %b = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 10, i32* %a, align 4
  store i32 5, i32* %b, align 4
  %LoadInst = load i32, i32* %a, align 4
  %LoadInst1 = load i32, i32* %b, align 4
  %LoadInst2 = load i32, i32* %a, align 4
  %LoadInst3 = load i32, i32* %b, align 4
  %0 = add i32 %LoadInst2, %LoadInst3
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([24 x i8], [24 x
i8]* @_Const_String_, i32 0, i32 0), i32 %LoadInst, i32 %LoadInst1, i32 %0)
  %LoadInst4 = load i32, i32* %a, align 4
  %LoadInst5 = load i32, i32* %b, align 4
  %LoadInst6 = load i32, i32* %a, align 4
  %LoadInst7 = load i32, i32* %b, align 4
  %1 = sub i32 %LoadInst6, %LoadInst7
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([27 x i8], [27
x i8]* @_Const_String_.1, i32 0, i32 0), i32 %LoadInst4, i32 %LoadInst5, i32 %1)
  %LoadInst9 = load i32, i32* %a, align 4
  %LoadInst10 = load i32, i32* %b, align 4
  %LoadInst11 = load i32, i32* %a, align 4
  %LoadInst12 = load i32, i32* %b, align 4
  %2 = mul i32 %LoadInst11, %LoadInst12
  %printf13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([30 x i8], [30
x i8]* @_Const_String_.2, i32 0, i32 0), i32 %LoadInst9, i32 %LoadInst10, i32 %2)
  %LoadInst14 = load i32, i32* %a, align 4
```

```
  %LoadInst15 = load i32, i32* %b, align 4
  %LoadInst16 = load i32, i32* %a, align 4
  %LoadInst17 = load i32, i32* %b, align 4
  %3 = sdiv i32 %LoadInst16, %LoadInst17
  %printf18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([24 x i8], [24
x i8]* @_Const_String_.3, i32 0, i32 0), i32 %LoadInst14, i32 %LoadInst15, i32 %3)
  %LoadInst19 = load i32, i32* %a, align 4
  %LoadInst20 = load i32, i32* %b, align 4
  %icmptmp = icmp eq i32 %LoadInst19, %LoadInst20
  %LoadInst21 = load i32, i32* %a, align 4
  %LoadInst22 = load i32, i32* %b, align 4
  %icmptmp23 = icmp ne i32 %LoadInst21, %LoadInst22
  %tmpAnd = and i1 %icmptmp, %icmptmp23
  %printf24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([40 x i8], [40
x i8]* @_Const_String_.4, i32 0, i32 0), i1 %tmpAnd)
  %LoadInst25 = load i32, i32* %a, align 4
  %LoadInst26 = load i32, i32* %b, align 4
  %icmptmp27 = icmp eq i32 %LoadInst25, %LoadInst26
  %LoadInst28 = load i32, i32* %a, align 4
  %LoadInst29 = load i32, i32* %b, align 4
  %icmptmp30 = icmp ne i32 %LoadInst28, %LoadInst29
  %tmpOR = or i1 %icmptmp27, %icmptmp30
  %printf31 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([39 x i8], [39
x i8]* @_Const_String_.5, i32 0, i32 0), i1 %tmpOR)
  %LoadInst32 = load i32, i32* %a, align 4
  %LoadInst33 = load i32, i32* %b, align 4
  %icmptmp34 = icmp eq i32 %LoadInst32, %LoadInst33
  %printf35 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([22 x i8], [22
x i8]* @_Const_String_.6, i32 0, i32 0), i1 %icmptmp34)
  %LoadInst36 = load i32, i32* %a, align 4
  %LoadInst37 = load i32, i32* %b, align 4
  %icmptmp38 = icmp ne i32 %LoadInst36, %LoadInst37
  %printf39 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8], [26
x i8]* @_Const_String_.7, i32 0, i32 0), i1 %icmptmp38)
  %LoadInst40 = load i32, i32* %a, align 4
  %LoadInst41 = load i32, i32* %b, align 4
  %icmptmp42 = icmp slt i32 %LoadInst40, %LoadInst41
  %printf43 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([24 x i8], [24
x i8]* @_Const_String_.8, i32 0, i32 0), i1 %icmptmp42)
  %LoadInst44 = load i32, i32* %a, align 4
  %LoadInst45 = load i32, i32* %b, align 4
  %icmptmp46 = icmp sgt i32 %LoadInst44, %LoadInst45
  %printf47 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([27 x i8], [27
x i8]* @_Const_String_.9, i32 0, i32 0), i1 %icmptmp46)
  %LoadInst48 = load i32, i32* %a, align 4
  %LoadInst49 = load i32, i32* %b, align 4
  %icmptmp50 = icmp sle i32 %LoadInst48, %LoadInst49
  %printf51 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([35 x i8], [35
x i8]* @_Const_String_.10, i32 0, i32 0), i1 %icmptmp50)
  %LoadInst52 = load i32, i32* %a, align 4
  %LoadInst53 = load i32, i32* %b, align 4
  %icmptmp54 = icmp sge i32 %LoadInst52, %LoadInst53
```

```
  %printf55 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([38 x i8], [38
x i8]* @_Const_String_.11, i32 0, i32 0), i1 %icmptmp54)
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -32
    .cfi_def_cfa_offset 32
    sw  ra, 28(sp)                      # 4-byte Folded Spill
    sw  s0, 24(sp)                      # 4-byte Folded Spill
    sw  s1, 20(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    .cfi_offset s0, -8
    .cfi_offset s1, -12
    li  a0, 10
    sw  a0, 12(sp)
    li  a0, 5
    sw  a0, 16(sp)
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    li  a1, 10
    li  a2, 5
    li  a3, 15
    call    printf@plt
    lw  a1, 12(sp)
    lw  a2, 16(sp)
    sub a3, a1, a2
    lui a0, %hi(.L_Const_String_.1)
    addi    a0, a0, %lo(.L_Const_String_.1)
    call    printf@plt
    lw  s0, 12(sp)
    lw  s1, 16(sp)
    mv  a0, s0
    mv  a1, s1
    call    __mulsi3@plt
    mv  a3, a0
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
```

```
    mv  a1, s0
    mv  a2, s1
    call    printf@plt
    lw  s0, 12(sp)
    lw  s1, 16(sp)
    mv  a0, s0
    mv  a1, s1
    call    __divsi3@plt
    mv  a3, a0
    lui a0, %hi(.L_Const_String_.3)
    addi    a0, a0, %lo(.L_Const_String_.3)
    mv  a1, s0
    mv  a2, s1
    call    printf@plt
    lui a0, %hi(.L_Const_String_.4)
    addi    a0, a0, %lo(.L_Const_String_.4)
    li  a1, 0
    call    printf@plt
    lui a0, %hi(.L_Const_String_.5)
    addi    a0, a0, %lo(.L_Const_String_.5)
    li  a1, 1
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    xor a0, a0, a1
    seqz    a1, a0
    lui a0, %hi(.L_Const_String_.6)
    addi    a0, a0, %lo(.L_Const_String_.6)
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    xor a0, a0, a1
    snez    a1, a0
    lui a0, %hi(.L_Const_String_.7)
    addi    a0, a0, %lo(.L_Const_String_.7)
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    slt a1, a0, a1
    lui a0, %hi(.L_Const_String_.8)
    addi    a0, a0, %lo(.L_Const_String_.8)
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    slt a1, a1, a0
    lui a0, %hi(.L_Const_String_.9)
    addi    a0, a0, %lo(.L_Const_String_.9)
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    slt a0, a1, a0
    xori    a1, a0, 1
```

```
    lui a0, %hi(.L_Const_String_.10)
    addi    a0, a0, %lo(.L_Const_String_.10)
    call    printf@plt
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    slt a0, a0, a1
    xori    a1, a0, 1
    lui a0, %hi(.L_Const_String_.11)
    addi    a0, a0, %lo(.L_Const_String_.11)
    call    printf@plt
    li  a0, 0
    lw  ra, 28(sp)                      # 4-byte Folded Reload
    lw  s0, 24(sp)                      # 4-byte Folded Reload
    lw  s1, 20(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 32
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
    .p2align    4
.L_Const_String_:
    .asciz  "Addition: %d + %d = %d\n"
    .size   .L_Const_String_, 24

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
    .p2align    4
.L_Const_String_.1:
    .asciz  "Subtraction: %d - %d = %d\n"
    .size   .L_Const_String_.1, 27

    .type   .L_Const_String_.2,@object      # @_Const_String_.2
    .p2align    4
.L_Const_String_.2:
    .asciz  "Multiplication: %d * %d = %d\n"
    .size   .L_Const_String_.2, 30

    .type   .L_Const_String_.3,@object      # @_Const_String_.3
    .p2align    4
.L_Const_String_.3:
    .asciz  "Division: %d / %d = %d\n"
    .size   .L_Const_String_.3, 24

    .type   .L_Const_String_.4,@object      # @_Const_String_.4
    .p2align    4
.L_Const_String_.4:
    .asciz  "Logical AND: (a == b) && (a != b) = %d\n"
    .size   .L_Const_String_.4, 40

    .type   .L_Const_String_.5,@object      # @_Const_String_.5
```

```
    .p2align    4
.L_Const_String_.5:
    .asciz  "Logical OR: (a == b) || (a != b) = %d\n"
    .size   .L_Const_String_.5, 39

    .type   .L_Const_String_.6,@object      # @_Const_String_.6
    .p2align    4
.L_Const_String_.6:
    .asciz  "Equals: a == b -> %d\n"
    .size   .L_Const_String_.6, 22

    .type   .L_Const_String_.7,@object      # @_Const_String_.7
    .p2align    4
.L_Const_String_.7:
    .asciz  "Not Equals: a != b -> %d\n"
    .size   .L_Const_String_.7, 26

    .type   .L_Const_String_.8,@object      # @_Const_String_.8
    .p2align    4
.L_Const_String_.8:
    .asciz  "Less Than: a < b -> %d\n"
    .size   .L_Const_String_.8, 24

    .type   .L_Const_String_.9,@object      # @_Const_String_.9
    .p2align    4
.L_Const_String_.9:
    .asciz  "Greater Than: a > b -> %d\n"
    .size   .L_Const_String_.9, 27

    .type   .L_Const_String_.10,@object     # @_Const_String_.10
    .p2align    4
.L_Const_String_.10:
    .asciz  "Less Than or Equals: a <= b -> %d\n"
    .size   .L_Const_String_.10, 35

    .type   .L_Const_String_.11,@object     # @_Const_String_.11
    .p2align    4
.L_Const_String_.11:
    .asciz  "Greater Than or Equals: a >= b -> %d\n"
    .size   .L_Const_String_.11, 38

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

- 运行结果

```
1.100000 2.200000 3.300000 4.400000 5.500000
zhengyihao@zhengyihao-virtual-machine:~/2023_foc_complier/test$ ./test_operat
ions
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2
Logical AND: (a == b) && (a != b) = 0
Logical OR: (a == b) || (a != b) = 1
Equals: a == b -> 0
Not Equals: a != b -> 1
Less Than: a < b -> 0
Greater Than: a > b -> 1
Less Than or Equals: a <= b -> 0
Greater Than or Equals: a >= b -> 1
```

控制流测试

分支测试

- 测试代码

```c
int main() {
    int a = 10;
    int b = 5;

    // If-Else Branching
    if (a > b) {
        printf("a is greater than b\n");
    } else {
        printf("b is greater than a\n");
    }

    // Switch Case
    switch (a) {
        case 10:{
            printf("a is 10\n");
            break;
        }
        case 5:{
            printf("a is 5\n");
            break;
        }
    }

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [21 x i8] c"a is greater than b\0A\00"
@_Const_String_.1 = private constant [21 x i8] c"b is greater than a\0A\00"
@_Const_String_.2 = private constant [9 x i8] c"a is 10\0A\00"
@_Const_String_.3 = private constant [8 x i8] c"a is 5\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %b = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 10, i32* %a, align 4
  store i32 5, i32* %b, align 4
  %LoadInst = load i32, i32* %a, align 4
  %LoadInst1 = load i32, i32* %b, align 4
  %icmptmp = icmp sgt i32 %LoadInst, %LoadInst1
  br i1 %icmptmp, label %Then, label %Else

Then:                                             ; preds = %entry
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21 x
i8]* @_Const_String_, i32 0, i32 0))
  br label %Merge4

Else:                                             ; preds = %entry
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21
x i8]* @_Const_String_.1, i32 0, i32 0))
  br label %Merge4

Merge4:                                           ; preds = %Else, %Then
  %LoadInst3 = load i32, i32* %a, align 4
  %icmptmp5 = icmp eq i32 %LoadInst3, 10
  br i1 %icmptmp5, label %Case0, label %Comparison1

Case0:                                            ; preds = %Merge4
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x
i8]* @_Const_String_.2, i32 0, i32 0))
  br label %SwitchEnd

Comparison1:                                      ; preds = %Merge4
  %icmptmp7 = icmp eq i32 %LoadInst3, 5
  br i1 %icmptmp7, label %Case1, label %SwitchEnd

Case1:                                            ; preds = %Comparison1
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([8 x i8], [8 x
i8]* @_Const_String_.3, i32 0, i32 0))
  br label %SwitchEnd
```

```
SwitchEnd:                                        ; preds = %Case1, %Comparison1,
%Case0
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -16
    .cfi_def_cfa_offset 16
    sw  ra, 12(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    li  a0, 10
    sw  a0, 4(sp)
    li  a0, 5
    sw  a0, 8(sp)
    bnez    zero, .LBB0_2
# %bb.1:                                # %Then
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    j   .LBB0_3
.LBB0_2:                                # %Else
    lui a0, %hi(.L_Const_String_.1)
    addi    a0, a0, %lo(.L_Const_String_.1)
.LBB0_3:                                # %Merge4
    call    printf@plt
    lw  a0, 4(sp)
    li  a1, 10
    bne a0, a1, .LBB0_5
# %bb.4:                                # %Case0
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
    j   .LBB0_7
.LBB0_5:                                # %Comparison1
    li  a1, 5
    bne a0, a1, .LBB0_8
# %bb.6:                                # %Case1
    lui a0, %hi(.L_Const_String_.3)
    addi    a0, a0, %lo(.L_Const_String_.3)
.LBB0_7:                                # %SwitchEnd
```

```
    call    printf@plt
.LBB0_8:                                # %SwitchEnd
    li  a0, 0
    lw  ra, 12(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 16
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
    .p2align    4
.L_Const_String_:
    .asciz  "a is greater than b\n"
    .size   .L_Const_String_, 21

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
    .p2align    4
.L_Const_String_.1:
    .asciz  "b is greater than a\n"
    .size   .L_Const_String_.1, 21

    .type   .L_Const_String_.2,@object      # @_Const_String_.2
.L_Const_String_.2:
    .asciz  "a is 10\n"
    .size   .L_Const_String_.2, 9

    .type   .L_Const_String_.3,@object      # @_Const_String_.3
.L_Const_String_.3:
    .asciz  "a is 5\n"
    .size   .L_Const_String_.3, 8

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

- 运行结果

```
zhengyihao@zhengyihao-virtual-machine:~/2023 foc complier/test$ ./test_branch
a is greater than b
a is 10
```

循环测试

- 测试代码

```
int main() {
    // For Loop
    int i;
    for (i = 0; i < 5; i = i+1) {
        printf("For loop iteration: %d\n", i);
```

```
    }

    // While Loop
    int count = 0;
    while (count < 5) {
        printf("While loop iteration: %d\n", count);
        count = count + 1;
    }

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [24 x i8] c"For loop iteration: %d\0A\00"
@_Const_String_.1 = private constant [26 x i8] c"While loop iteration: %d\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %count = alloca i32, align 4
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  br label %ForCond

ForCond:                                          ; preds = %ForTail, %entry
  %LoadInst = load i32, i32* %i, align 4
  %icmptmp = icmp slt i32 %LoadInst, 5
  br i1 %icmptmp, label %ForLoop, label %ForEnd

ForLoop:                                          ; preds = %ForCond
  %LoadInst1 = load i32, i32* %i, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([24 x i8], [24 x
i8]* @_Const_String_, i32 0, i32 0), i32 %LoadInst1)
  br label %ForTail

ForTail:                                          ; preds = %ForLoop
  %LoadInst2 = load i32, i32* %i, align 4
  %0 = add i32 %LoadInst2, 1
  store i32 %0, i32* %i, align 4
  br label %ForCond

ForEnd:                                           ; preds = %ForCond
```

```
  store i32 0, i32* %count, align 4
  br label %WhileCond

WhileCond:                                        ; preds = %WhileLoop, %ForEnd
  %LoadInst3 = load i32, i32* %count, align 4
  %icmptmp4 = icmp slt i32 %LoadInst3, 5
  br i1 %icmptmp4, label %WhileLoop, label %WhileEnd

WhileLoop:                                        ; preds = %WhileCond
  %LoadInst5 = load i32, i32* %count, align 4
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8], [26
x i8]* @_Const_String_.1, i32 0, i32 0), i32 %LoadInst5)
  %LoadInst7 = load i32, i32* %count, align 4
  %1 = add i32 %LoadInst7, 1
  store i32 %1, i32* %count, align 4
  br label %WhileCond

WhileEnd:                                         ; preds = %WhileCond
  ret i32 0
}
```

- riscv32汇编指令

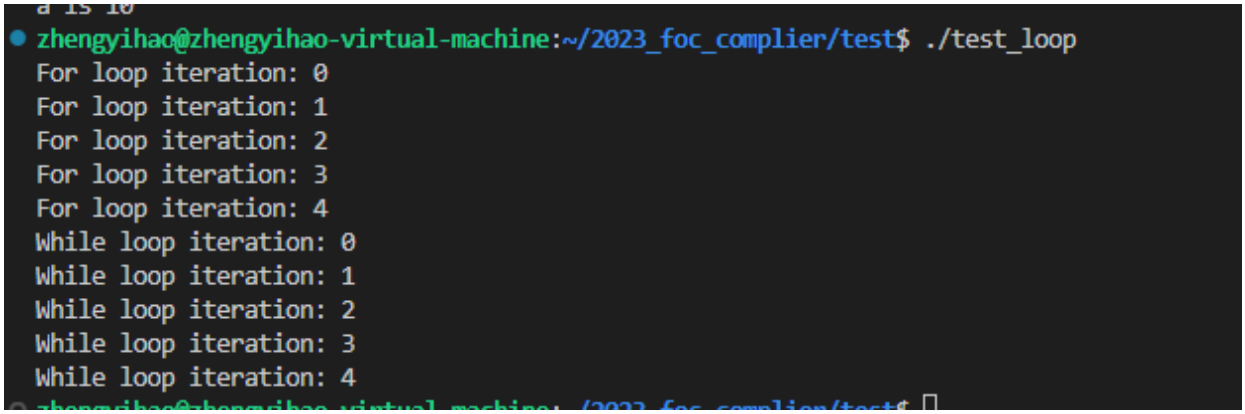
```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -32
    .cfi_def_cfa_offset 32
    sw  ra, 28(sp)                      # 4-byte Folded Spill
    sw  s0, 24(sp)                      # 4-byte Folded Spill
    sw  s1, 20(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    .cfi_offset s0, -8
    .cfi_offset s1, -12
    sw  zero, 12(sp)
    li  s1, 4
    lui a0, %hi(.L_Const_String_)
    addi    s0, a0, %lo(.L_Const_String_)
    lw  a0, 12(sp)
    blt s1, a0, .LBB0_2
.LBB0_1:                                # %ForLoop
                                        # =>This Inner Loop Header: Depth=1
```

```
    lw  a1, 12(sp)
    mv  a0, s0
    call    printf@plt
    lw  a0, 12(sp)
    addi    a0, a0, 1
    sw  a0, 12(sp)
    lw  a0, 12(sp)
    bge s1, a0, .LBB0_1
.LBB0_2:                                # %ForEnd
    sw  zero, 16(sp)
    li  s1, 4
    lui a0, %hi(.L_Const_String_.1)
    addi    s0, a0, %lo(.L_Const_String_.1)
    lw  a0, 16(sp)
    blt s1, a0, .LBB0_4
.LBB0_3:                                # %WhileLoop
                                        # =>This Inner Loop Header: Depth=1
    lw  a1, 16(sp)
    mv  a0, s0
    call    printf@plt
    lw  a0, 16(sp)
    addi    a0, a0, 1
    sw  a0, 16(sp)
    lw  a0, 16(sp)
    bge s1, a0, .LBB0_3
.LBB0_4:                                # %WhileEnd
    li  a0, 0
    lw  ra, 28(sp)                      # 4-byte Folded Reload
    lw  s0, 24(sp)                      # 4-byte Folded Reload
    lw  s1, 20(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 32
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
    .p2align    4
.L_Const_String_:
    .asciz  "For loop iteration: %d\n"
    .size   .L_Const_String_, 24

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
    .p2align    4
.L_Const_String_.1:
    .asciz  "While loop iteration: %d\n"
    .size   .L_Const_String_.1, 26

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

- 运行结果



函数测试

简单函数测试

- 测试代码

```
// Simple function to add two integers
int add(int a, int b) {
    return a + b;
}

int main() {
    int a = 10;
    int b = 5;

    // Function Call
    int sum = add(a, b);
    printf("Sum of a and b: %d\n", sum);

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [20 x i8] c"Sum of a and b: %d\0A\00"

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

declare i32 @scanf(...)
```

```
define i32 @add(i32 %0, i32 %1) {
entry:
  %b = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 %0, i32* %a, align 4
  store i32 %1, i32* %b, align 4
  %LoadInst = load i32, i32* %a, align 4
  %LoadInst1 = load i32, i32* %b, align 4
  %2 = add i32 %LoadInst, %LoadInst1
  ret i32 %2
}

define i32 @main() {
entry:
  %sum = alloca i32, align 4
  %b = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 10, i32* %a, align 4
  store i32 5, i32* %b, align 4
  %LoadInst = load i32, i32* %a, align 4
  %LoadInst1 = load i32, i32* %b, align 4
  %0 = call i32 @add(i32 %LoadInst, i32 %LoadInst1)
  store i32 %0, i32* %sum, align 4
  %LoadInst2 = load i32, i32* %sum, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([20 x i8], [20 x
i8]* @_Const_String_, i32 0, i32 0), i32 %LoadInst2)
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  add                             # -- Begin function add
    .p2align    2
    .type   add,@function
add:                                    # @add
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -16
    .cfi_def_cfa_offset 16
    sw  a0, 8(sp)
    add a0, a0, a1
    sw  a1, 12(sp)
    addi    sp, sp, 16
    ret
.Lfunc_end0:
```

```
    .size   add, .Lfunc_end0-add
    .cfi_endproc
                                        # -- End function
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -16
    .cfi_def_cfa_offset 16
    sw  ra, 12(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    li  a0, 10
    sw  a0, 0(sp)
    li  a0, 5
    sw  a0, 4(sp)
    li  a0, 10
    li  a1, 5
    call    add@plt
    mv  a1, a0
    sw  a0, 8(sp)
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    call    printf@plt
    li  a0, 0
    lw  ra, 12(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 16
    ret
.Lfunc_end1:
    .size   main, .Lfunc_end1-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
    .p2align    4
.L_Const_String_:
    .asciz  "Sum of a and b: %d\n"
    .size   .L_Const_String_, 20

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

- 运行结果

```
zhengyihao@zhengyihao-virtual-machine:~/2023_foc_complier/test$ ./test_simple_function
Sum of a and b: 15
zhengyihao@zhengyihao-virtual-machine:~/2023_foc_complier/test$
```

递归函数测试

- 测试代码

```
int main() {
    int a = 10;
    int b = 5;

    // If-Else Branching
    if (a > b) {
        printf("a is greater than b\n");
    } else {
        printf("b is greater than a\n");
    }

    // Switch Case
    switch (a) {
        case 10:{
            printf("a is 10\n");
            break;
        }
        case 5:{
            printf("a is 5\n");
            break;
        }
    }

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [21 x i8] c"a is greater than b\0A\00"
@_Const_String_.1 = private constant [21 x i8] c"b is greater than a\0A\00"
@_Const_String_.2 = private constant [9 x i8] c"a is 10\0A\00"
@_Const_String_.3 = private constant [8 x i8] c"a is 5\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %b = alloca i32, align 4
  %a = alloca i32, align 4
```

```
  store i32 10, i32* %a, align 4
  store i32 5, i32* %b, align 4
  %LoadInst = load i32, i32* %a, align 4
  %LoadInst1 = load i32, i32* %b, align 4
  %icmptmp = icmp sgt i32 %LoadInst, %LoadInst1
  br i1 %icmptmp, label %Then, label %Else

Then:                                             ; preds = %entry
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21 x
i8]* @_Const_String_, i32 0, i32 0))
  br label %Merge4

Else:                                             ; preds = %entry
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21
x i8]* @_Const_String_.1, i32 0, i32 0))
  br label %Merge4

Merge4:                                           ; preds = %Else, %Then
  %LoadInst3 = load i32, i32* %a, align 4
  %icmptmp5 = icmp eq i32 %LoadInst3, 10
  br i1 %icmptmp5, label %Case0, label %Comparison1

Case0:                                            ; preds = %Merge4
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x
i8]* @_Const_String_.2, i32 0, i32 0))
  br label %SwitchEnd

Comparison1:                                      ; preds = %Merge4
  %icmptmp7 = icmp eq i32 %LoadInst3, 5
  br i1 %icmptmp7, label %Case1, label %SwitchEnd

Case1:                                            ; preds = %Comparison1
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([8 x i8], [8 x
i8]* @_Const_String_.3, i32 0, i32 0))
  br label %SwitchEnd

SwitchEnd:                                        ; preds = %Case1, %Comparison1,
%Case0
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
```

```
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -16
    .cfi_def_cfa_offset 16
    sw  ra, 12(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    li  a0, 10
    sw  a0, 4(sp)
    li  a0, 5
    sw  a0, 8(sp)
    bnez    zero, .LBB0_2
# %bb.1:                                # %Then
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    j   .LBB0_3
.LBB0_2:                                # %Else
    lui a0, %hi(.L_Const_String_.1)
    addi    a0, a0, %lo(.L_Const_String_.1)
.LBB0_3:                                # %Merge4
    call    printf@plt
    lw  a0, 4(sp)
    li  a1, 10
    bne a0, a1, .LBB0_5
# %bb.4:                                # %Case0
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
    j   .LBB0_7
.LBB0_5:                                # %Comparison1
    li  a1, 5
    bne a0, a1, .LBB0_8
# %bb.6:                                # %Case1
    lui a0, %hi(.L_Const_String_.3)
    addi    a0, a0, %lo(.L_Const_String_.3)
.LBB0_7:                                # %SwitchEnd
    call    printf@plt
.LBB0_8:                                # %SwitchEnd
    li  a0, 0
    lw  ra, 12(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 16
    ret
.Lfunc_end0:
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
    .p2align    4
.L_Const_String_:
    .asciz  "a is greater than b\n"
    .size   .L_Const_String_, 21
```

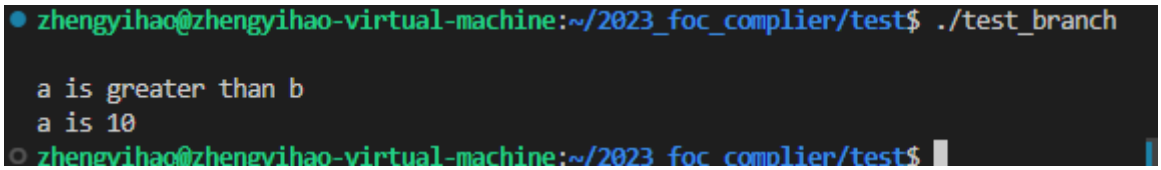
```
    .type   .L_Const_String_.1,@object      # @_Const_String_.1
    .p2align    4
.L_Const_String_.1:
    .asciz  "b is greater than a\n"
    .size   .L_Const_String_.1, 21

    .type   .L_Const_String_.2,@object      # @_Const_String_.2
.L_Const_String_.2:
    .asciz  "a is 10\n"
    .size   .L_Const_String_.2, 9

    .type   .L_Const_String_.3,@object      # @_Const_String_.3
.L_Const_String_.3:
    .asciz  "a is 5\n"
    .size   .L_Const_String_.3, 8

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

- 运行结果



综合测试

矩阵乘法

- 测试代码

```
int A[200000];
int B[200000];
int C[200000];

char input[100000];

int main() {
    int A_M;
    int A_N;
    int B_M;
    int B_N;
    //scanf("%s", input);
    //scanf("%s", input);
    scanf("%d %d\n", &A_M, &A_N);
    int i = 0;
    int j = 0;
```

```
    while(i < A_M) {
        j = 0;
        while(j < A_N) {
            scanf("%d", &A[i * A_N + j]);
            j = j + 1;
        }
        i = i + 1;
    }

    //scanf("%s", input);
    //scanf("%s", input);
    scanf("%d %d", &B_M, &B_N);
    i = 0;
    j = 0;
    while(i < B_M) {
        j = 0;
        while(j < B_N) {
            scanf("%d", &B[i * B_N + j]);
            j = j + 1;
        }
        i = i + 1;
    }

    if(A_N != B_M) {
        printf("Incompatible Dimensions\n");
        return 0;
    }

    i = 0;
    j = 0;
    while(i < A_M) {
        j = 0;
        while(j < B_N) {
            C[i * B_N + j] = 0;
            int k = 0;
            while(k < A_N) {
                C[i * B_N + j] = C[i * B_N + j] + A[i * A_N + k] * B[k * B_N + j];
                k = k + 1;
            }
            j = j + 1;
        }
        i = i + 1;
    }

    i = 0;
    while(i < A_M) {
        j = 0;
        while(j < B_N) {
            printf("%10d", C[i * B_N + j]);
            j = j + 1;
        }
        printf("\n");
```

```
        i = i + 1;
    }
    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@A = global [200000 x i32] zeroinitializer
@B = global [200000 x i32] zeroinitializer
@C = global [200000 x i32] zeroinitializer
@input = global [100000 x i8] zeroinitializer
@_Const_String_ = private constant [7 x i8] c"%d %d\0A\00"
@_Const_String_.1 = private constant [3 x i8] c"%d\00"
@_Const_String_.2 = private constant [6 x i8] c"%d %d\00"
@_Const_String_.3 = private constant [3 x i8] c"%d\00"
@_Const_String_.4 = private constant [25 x i8] c"Incompatible Dimensions\0A\00"
@_Const_String_.5 = private constant [5 x i8] c"%10d\00"
@_Const_String_.6 = private constant [2 x i8] c"\0A\00"

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

declare i32 @scanf(...)

define i32 @main() {
entry:
  %k = alloca i32, align 4
  %j = alloca i32, align 4
  %i = alloca i32, align 4
  %B_N = alloca i32, align 4
  %B_M = alloca i32, align 4
  %A_N = alloca i32, align 4
  %A_M = alloca i32, align 4
  %scanf = call i32 (...) @scanf(i8* getelementptr inbounds ([7 x i8], [7 x i8]*
@_Const_String_, i32 0, i32 0), i32* %A_M, i32* %A_N)
  store i32 0, i32* %i, align 4
  store i32 0, i32* %j, align 4
  br label %WhileCond

WhileCond:                                        ; preds = %WhileEnd, %entry
  %LoadInst = load i32, i32* %i, align 4
  %LoadInst1 = load i32, i32* %A_M, align 4
  %icmptmp = icmp slt i32 %LoadInst, %LoadInst1
  br i1 %icmptmp, label %WhileLoop, label %WhileEnd13

WhileLoop:                                        ; preds = %WhileCond
  store i32 0, i32* %j, align 4
  br label %WhileCond2
```

```
WhileCond2:                                       ; preds = %WhileLoop6, %WhileLoop
  %LoadInst3 = load i32, i32* %j, align 4
  %LoadInst4 = load i32, i32* %A_N, align 4
  %icmptmp5 = icmp slt i32 %LoadInst3, %LoadInst4
  br i1 %icmptmp5, label %WhileLoop6, label %WhileEnd

WhileLoop6:                                       ; preds = %WhileCond2
  %LoadInst7 = load i32, i32* %i, align 4
  %LoadInst8 = load i32, i32* %A_N, align 4
  %0 = mul i32 %LoadInst7, %LoadInst8
  %LoadInst9 = load i32, i32* %j, align 4
  %1 = add i32 %0, %LoadInst9
  %elePtr = getelementptr inbounds [200000 x i32], [200000 x i32]* @A, i32 0, i32 %1
  %scanf10 = call i32 (...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@_Const_String_.1, i32 0, i32 0), i32* %elePtr)
  %LoadInst11 = load i32, i32* %j, align 4
  %2 = add i32 %LoadInst11, 1
  store i32 %2, i32* %j, align 4
  br label %WhileCond2

WhileEnd:                                         ; preds = %WhileCond2
  %LoadInst12 = load i32, i32* %i, align 4
  %3 = add i32 %LoadInst12, 1
  store i32 %3, i32* %i, align 4
  br label %WhileCond

WhileEnd13:                                       ; preds = %WhileCond
  %scanf14 = call i32 (...) @scanf(i8* getelementptr inbounds ([6 x i8], [6 x i8]*
@_Const_String_.2, i32 0, i32 0), i32* %B_M, i32* %B_N)
  store i32 0, i32* %i, align 4
  store i32 0, i32* %j, align 4
  br label %WhileCond15

WhileCond15:                                      ; preds = %WhileEnd31, %WhileEnd13
  %LoadInst16 = load i32, i32* %i, align 4
  %LoadInst17 = load i32, i32* %B_M, align 4
  %icmptmp18 = icmp slt i32 %LoadInst16, %LoadInst17
  br i1 %icmptmp18, label %WhileLoop19, label %WhileEnd33

WhileLoop19:                                      ; preds = %WhileCond15
  store i32 0, i32* %j, align 4
  br label %WhileCond20

WhileCond20:                                      ; preds = %WhileLoop24,
%WhileLoop19
  %LoadInst21 = load i32, i32* %j, align 4
  %LoadInst22 = load i32, i32* %B_N, align 4
  %icmptmp23 = icmp slt i32 %LoadInst21, %LoadInst22
  br i1 %icmptmp23, label %WhileLoop24, label %WhileEnd31

WhileLoop24:                                      ; preds = %WhileCond20
```

```
  %LoadInst25 = load i32, i32* %i, align 4
  %LoadInst26 = load i32, i32* %B_N, align 4
  %4 = mul i32 %LoadInst25, %LoadInst26
  %LoadInst27 = load i32, i32* %j, align 4
  %5 = add i32 %4, %LoadInst27
  %elePtr28 = getelementptr inbounds [200000 x i32], [200000 x i32]* @B, i32 0, i32
%5
  %scanf29 = call i32 (...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@_Const_String_.3, i32 0, i32 0), i32* %elePtr28)
  %LoadInst30 = load i32, i32* %j, align 4
  %6 = add i32 %LoadInst30, 1
  store i32 %6, i32* %j, align 4
  br label %WhileCond20

WhileEnd31:                                       ; preds = %WhileCond20
  %LoadInst32 = load i32, i32* %i, align 4
  %7 = add i32 %LoadInst32, 1
  store i32 %7, i32* %i, align 4
  br label %WhileCond15

WhileEnd33:                                       ; preds = %WhileCond15
  %LoadInst34 = load i32, i32* %A_N, align 4
  %LoadInst35 = load i32, i32* %B_M, align 4
  %icmptmp36 = icmp ne i32 %LoadInst34, %LoadInst35
  br i1 %icmptmp36, label %Then, label %Else

Then:                                             ; preds = %WhileEnd33
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([25 x i8], [25 x
i8]* @_Const_String_.4, i32 0, i32 0))
  ret i32 0

Else:                                             ; preds = %WhileEnd33
  br label %Merge

Merge:                                            ; preds = %Else
  store i32 0, i32* %i, align 4
  store i32 0, i32* %j, align 4
  br label %WhileCond37

WhileCond37:                                      ; preds = %WhileEnd76, %Merge
  %LoadInst38 = load i32, i32* %i, align 4
  %LoadInst39 = load i32, i32* %A_M, align 4
  %icmptmp40 = icmp slt i32 %LoadInst38, %LoadInst39
  br i1 %icmptmp40, label %WhileLoop41, label %WhileEnd78

WhileLoop41:                                      ; preds = %WhileCond37
  store i32 0, i32* %j, align 4
  br label %WhileCond42

WhileCond42:                                      ; preds = %WhileEnd74,
%WhileLoop41
  %LoadInst43 = load i32, i32* %j, align 4
```

```
  %LoadInst44 = load i32, i32* %B_N, align 4
  %icmptmp45 = icmp slt i32 %LoadInst43, %LoadInst44
  br i1 %icmptmp45, label %WhileLoop46, label %WhileEnd76

WhileLoop46:                                      ; preds = %WhileCond42
  %LoadInst47 = load i32, i32* %i, align 4
  %LoadInst48 = load i32, i32* %B_N, align 4
  %8 = mul i32 %LoadInst47, %LoadInst48
  %LoadInst49 = load i32, i32* %j, align 4
  %9 = add i32 %8, %LoadInst49
  %tmpvar = getelementptr inbounds [200000 x i32], [200000 x i32]* @C, i32 0, i32 %9
  store i32 0, i32* %tmpvar, align 4
  store i32 0, i32* %k, align 4
  br label %WhileCond50

WhileCond50:                                      ; preds = %WhileLoop54,
%WhileLoop46
  %LoadInst51 = load i32, i32* %k, align 4
  %LoadInst52 = load i32, i32* %A_N, align 4
  %icmptmp53 = icmp slt i32 %LoadInst51, %LoadInst52
  br i1 %icmptmp53, label %WhileLoop54, label %WhileEnd74

WhileLoop54:                                      ; preds = %WhileCond50
  %LoadInst55 = load i32, i32* %i, align 4
  %LoadInst56 = load i32, i32* %B_N, align 4
  %10 = mul i32 %LoadInst55, %LoadInst56
  %LoadInst57 = load i32, i32* %j, align 4
  %11 = add i32 %10, %LoadInst57
  %tmpvar58 = getelementptr inbounds [200000 x i32], [200000 x i32]* @C, i32 0, i32
%11
  %LoadInst59 = load i32, i32* %i, align 4
  %LoadInst60 = load i32, i32* %B_N, align 4
  %12 = mul i32 %LoadInst59, %LoadInst60
  %LoadInst61 = load i32, i32* %j, align 4
  %13 = add i32 %12, %LoadInst61
  %tmparray = getelementptr inbounds [200000 x i32], [200000 x i32]* @C, i32 0, i32
%13
  %tmpvar62 = load i32, i32* %tmparray, align 4
  %LoadInst63 = load i32, i32* %i, align 4
  %LoadInst64 = load i32, i32* %A_N, align 4
  %14 = mul i32 %LoadInst63, %LoadInst64
  %LoadInst65 = load i32, i32* %k, align 4
  %15 = add i32 %14, %LoadInst65
  %tmparray66 = getelementptr inbounds [200000 x i32], [200000 x i32]* @A, i32 0,
i32 %15
  %tmpvar67 = load i32, i32* %tmparray66, align 4
  %LoadInst68 = load i32, i32* %k, align 4
  %LoadInst69 = load i32, i32* %B_N, align 4
  %16 = mul i32 %LoadInst68, %LoadInst69
  %LoadInst70 = load i32, i32* %j, align 4
  %17 = add i32 %16, %LoadInst70
```

```
  %tmparray71 = getelementptr inbounds [200000 x i32], [200000 x i32]* @B, i32 0,
i32 %17
  %tmpvar72 = load i32, i32* %tmparray71, align 4
  %18 = mul i32 %tmpvar67, %tmpvar72
  %19 = add i32 %tmpvar62, %18
  store i32 %19, i32* %tmpvar58, align 4
  %LoadInst73 = load i32, i32* %k, align 4
  %20 = add i32 %LoadInst73, 1
  store i32 %20, i32* %k, align 4
  br label %WhileCond50

WhileEnd74:                                       ; preds = %WhileCond50
  %LoadInst75 = load i32, i32* %j, align 4
  %21 = add i32 %LoadInst75, 1
  store i32 %21, i32* %j, align 4
  br label %WhileCond42

WhileEnd76:                                       ; preds = %WhileCond42
  %LoadInst77 = load i32, i32* %i, align 4
  %22 = add i32 %LoadInst77, 1
  store i32 %22, i32* %i, align 4
  br label %WhileCond37

WhileEnd78:                                       ; preds = %WhileCond37
  store i32 0, i32* %i, align 4
  br label %WhileCond79

WhileCond79:                                      ; preds = %WhileEnd96, %WhileEnd78
  %LoadInst80 = load i32, i32* %i, align 4
  %LoadInst81 = load i32, i32* %A_M, align 4
  %icmptmp82 = icmp slt i32 %LoadInst80, %LoadInst81
  br i1 %icmptmp82, label %WhileLoop83, label %WhileEnd99

WhileLoop83:                                      ; preds = %WhileCond79
  store i32 0, i32* %j, align 4
  br label %WhileCond84

WhileCond84:                                      ; preds = %WhileLoop88,
%WhileLoop83
  %LoadInst85 = load i32, i32* %j, align 4
  %LoadInst86 = load i32, i32* %B_N, align 4
  %icmptmp87 = icmp slt i32 %LoadInst85, %LoadInst86
  br i1 %icmptmp87, label %WhileLoop88, label %WhileEnd96

WhileLoop88:                                      ; preds = %WhileCond84
  %LoadInst89 = load i32, i32* %i, align 4
  %LoadInst90 = load i32, i32* %B_N, align 4
  %23 = mul i32 %LoadInst89, %LoadInst90
  %LoadInst91 = load i32, i32* %j, align 4
  %24 = add i32 %23, %LoadInst91
  %tmparray92 = getelementptr inbounds [200000 x i32], [200000 x i32]* @C, i32 0,
i32 %24
```

```
  %tmpvar93 = load i32, i32* %tmparray92, align 4
  %printf94 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x
i8]* @_Const_String_.5, i32 0, i32 0), i32 %tmpvar93)
  %LoadInst95 = load i32, i32* %j, align 4
  %25 = add i32 %LoadInst95, 1
  store i32 %25, i32* %j, align 4
  br label %WhileCond84

WhileEnd96:                                       ; preds = %WhileCond84
  %printf97 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x
i8]* @_Const_String_.6, i32 0, i32 0))
  %LoadInst98 = load i32, i32* %i, align 4
  %26 = add i32 %LoadInst98, 1
  store i32 %26, i32* %i, align 4
  br label %WhileCond79

WhileEnd99:                                       ; preds = %WhileCond79
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -80
    .cfi_def_cfa_offset 80
    sw  ra, 76(sp)                      # 4-byte Folded Spill
    sw  s0, 72(sp)                      # 4-byte Folded Spill
    sw  s1, 68(sp)                      # 4-byte Folded Spill
    sw  s2, 64(sp)                      # 4-byte Folded Spill
    sw  s3, 60(sp)                      # 4-byte Folded Spill
    sw  s4, 56(sp)                      # 4-byte Folded Spill
    sw  s5, 52(sp)                      # 4-byte Folded Spill
    sw  s6, 48(sp)                      # 4-byte Folded Spill
    sw  s7, 44(sp)                      # 4-byte Folded Spill
    sw  s8, 40(sp)                      # 4-byte Folded Spill
    .cfi_offset ra, -4
    .cfi_offset s0, -8
    .cfi_offset s1, -12
    .cfi_offset s2, -16
    .cfi_offset s3, -20
```

```
    .cfi_offset s4, -24
    .cfi_offset s5, -28
    .cfi_offset s6, -32
    .cfi_offset s7, -36
    .cfi_offset s8, -40
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    addi    a1, sp, 12
    addi    a2, sp, 16
    call    scanf@plt
    sw  zero, 28(sp)
    sw  zero, 32(sp)
    lui a0, %hi(A)
    addi    s1, a0, %lo(A)
    lui a0, %hi(.L_Const_String_.1)
    addi    s0, a0, %lo(.L_Const_String_.1)
    j   .LBB0_2
.LBB0_1:                                # %WhileEnd
                                        #   in Loop: Header=BB0_2 Depth=1
    lw  a0, 28(sp)
    addi    a0, a0, 1
    sw  a0, 28(sp)
.LBB0_2:                                # %WhileCond
                                        # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_4 Depth 2
    lw  a0, 28(sp)
    lw  a1, 12(sp)
    bge a0, a1, .LBB0_6
# %bb.3:                                # %WhileLoop
                                        #   in Loop: Header=BB0_2 Depth=1
    sw  zero, 32(sp)
.LBB0_4:                                # %WhileCond2
                                        #   Parent Loop BB0_2 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
    lw  a0, 32(sp)
    lw  a1, 16(sp)
    bge a0, a1, .LBB0_1
# %bb.5:                                # %WhileLoop6
                                        #   in Loop: Header=BB0_4 Depth=2
    lw  a0, 28(sp)
    lw  a1, 16(sp)
    call    __mulsi3@plt
    lw  a1, 32(sp)
    add a0, a0, a1
    slli    a0, a0, 2
    add a1, s1, a0
    mv  a0, s0
    call    scanf@plt
    lw  a0, 32(sp)
    addi    a0, a0, 1
    sw  a0, 32(sp)
    j   .LBB0_4
```

```
.LBB0_6:                                # %WhileEnd13
    lui a0, %hi(.L_Const_String_.2)
    addi    a0, a0, %lo(.L_Const_String_.2)
    addi    a1, sp, 20
    addi    a2, sp, 24
    call    scanf@plt
    sw  zero, 28(sp)
    sw  zero, 32(sp)
    lui a0, %hi(B)
    addi    s1, a0, %lo(B)
    lui a0, %hi(.L_Const_String_.3)
    addi    s0, a0, %lo(.L_Const_String_.3)
    j   .LBB0_8
.LBB0_7:                                # %WhileEnd31
                                        #   in Loop: Header=BB0_8 Depth=1
    lw  a0, 28(sp)
    addi    a0, a0, 1
    sw  a0, 28(sp)
.LBB0_8:                                # %WhileCond15
                                        # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_10 Depth 2
    lw  a0, 28(sp)
    lw  a1, 20(sp)
    bge a0, a1, .LBB0_12
# %bb.9:                                # %WhileLoop19
                                        #   in Loop: Header=BB0_8 Depth=1
    sw  zero, 32(sp)
.LBB0_10:                               # %WhileCond20
                                        #   Parent Loop BB0_8 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
    lw  a0, 32(sp)
    lw  a1, 24(sp)
    bge a0, a1, .LBB0_7
# %bb.11:                               # %WhileLoop24
                                        #   in Loop: Header=BB0_10 Depth=2
    lw  a0, 28(sp)
    lw  a1, 24(sp)
    call    __mulsi3@plt
    lw  a1, 32(sp)
    add a0, a0, a1
    slli    a0, a0, 2
    add a1, s1, a0
    mv  a0, s0
    call    scanf@plt
    lw  a0, 32(sp)
    addi    a0, a0, 1
    sw  a0, 32(sp)
    j   .LBB0_10
.LBB0_12:                               # %WhileEnd33
    lw  a0, 16(sp)
    lw  a1, 20(sp)
    beq a0, a1, .LBB0_15
```

```
# %bb.13:                               # %Then
    lui a0, %hi(.L_Const_String_.4)
    addi    a0, a0, %lo(.L_Const_String_.4)
    call    printf@plt
.LBB0_14:                               # %WhileEnd99
    li  a0, 0
    lw  ra, 76(sp)                      # 4-byte Folded Reload
    lw  s0, 72(sp)                      # 4-byte Folded Reload
    lw  s1, 68(sp)                      # 4-byte Folded Reload
    lw  s2, 64(sp)                      # 4-byte Folded Reload
    lw  s3, 60(sp)                      # 4-byte Folded Reload
    lw  s4, 56(sp)                      # 4-byte Folded Reload
    lw  s5, 52(sp)                      # 4-byte Folded Reload
    lw  s6, 48(sp)                      # 4-byte Folded Reload
    lw  s7, 44(sp)                      # 4-byte Folded Reload
    lw  s8, 40(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 80
    ret
.LBB0_15:                               # %Else
    sw  zero, 28(sp)
    sw  zero, 32(sp)
    lui a0, %hi(C)
    addi    s3, a0, %lo(C)
    lui a0, %hi(A)
    addi    s4, a0, %lo(A)
    lui a0, %hi(B)
    addi    s5, a0, %lo(B)
    j   .LBB0_17
.LBB0_16:                               # %WhileEnd76
                                        #   in Loop: Header=BB0_17 Depth=1
    lw  a0, 28(sp)
    addi    a0, a0, 1
    sw  a0, 28(sp)
.LBB0_17:                               # %WhileCond37
                                        # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_20 Depth 2
                                        #       Child Loop BB0_22 Depth 3
    lw  a0, 28(sp)
    lw  a1, 12(sp)
    bge a0, a1, .LBB0_24
# %bb.18:                               # %WhileLoop41
                                        #   in Loop: Header=BB0_17 Depth=1
    sw  zero, 32(sp)
    j   .LBB0_20
.LBB0_19:                               # %WhileEnd74
                                        #   in Loop: Header=BB0_20 Depth=2
    lw  a0, 32(sp)
    addi    a0, a0, 1
    sw  a0, 32(sp)
.LBB0_20:                               # %WhileCond42
                                        #   Parent Loop BB0_17 Depth=1
                                        # =>  This Loop Header: Depth=2
```

```
                                        #       Child Loop BB0_22 Depth 3
    lw  a0, 32(sp)
    lw  a1, 24(sp)
    bge a0, a1, .LBB0_16
# %bb.21:                               # %WhileLoop46
                                        #   in Loop: Header=BB0_20 Depth=2
    lw  a0, 28(sp)
    lw  a1, 24(sp)
    call    __mulsi3@plt
    lw  a1, 32(sp)
    add a0, a0, a1
    slli    a0, a0, 2
    add a0, s3, a0
    sw  zero, 0(a0)
    sw  zero, 36(sp)
.LBB0_22:                               # %WhileCond50
                                        #   Parent Loop BB0_17 Depth=1
                                        #     Parent Loop BB0_20 Depth=2
                                        # =>    This Inner Loop Header: Depth=3
    lw  a0, 36(sp)
    lw  a1, 16(sp)
    bge a0, a1, .LBB0_19
# %bb.23:                               # %WhileLoop54
                                        #   in Loop: Header=BB0_22 Depth=3
    lw  s0, 28(sp)
    lw  s1, 24(sp)
    mv  a0, s0
    mv  a1, s1
    call    __mulsi3@plt
    lw  s6, 32(sp)
    add a0, a0, s6
    slli    a0, a0, 2
    add s7, s3, a0
    lw  s8, 0(s7)
    lw  a1, 16(sp)
    mv  a0, s0
    call    __mulsi3@plt
    lw  s0, 36(sp)
    add a0, a0, s0
    slli    a0, a0, 2
    add a0, s4, a0
    lw  s2, 0(a0)
    mv  a0, s0
    mv  a1, s1
    call    __mulsi3@plt
    add a0, a0, s6
    slli    a0, a0, 2
    add a0, s5, a0
    lw  a1, 0(a0)
    mv  a0, s2
    call    __mulsi3@plt
    add a0, s8, a0
```

```
    sw  a0, 0(s7)
    addi    a0, s0, 1
    sw  a0, 36(sp)
    j   .LBB0_22
.LBB0_24:                               # %WhileEnd78
    sw  zero, 28(sp)
    lui a0, %hi(C)
    addi    s2, a0, %lo(C)
    lui a0, %hi(.L_Const_String_.5)
    addi    s0, a0, %lo(.L_Const_String_.5)
    lui a0, %hi(.L_Const_String_.6)
    addi    s1, a0, %lo(.L_Const_String_.6)
    j   .LBB0_26
.LBB0_25:                               # %WhileEnd96
                                        #   in Loop: Header=BB0_26 Depth=1
    mv  a0, s1
    call    printf@plt
    lw  a0, 28(sp)
    addi    a0, a0, 1
    sw  a0, 28(sp)
.LBB0_26:                               # %WhileCond79
                                        # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_28 Depth 2
    lw  a0, 28(sp)
    lw  a1, 12(sp)
    bge a0, a1, .LBB0_14
# %bb.27:                               # %WhileLoop83
                                        #   in Loop: Header=BB0_26 Depth=1
    sw  zero, 32(sp)
.LBB0_28:                               # %WhileCond84
                                        #   Parent Loop BB0_26 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
    lw  a0, 32(sp)
    lw  a1, 24(sp)
    bge a0, a1, .LBB0_25
# %bb.29:                               # %WhileLoop88
                                        #   in Loop: Header=BB0_28 Depth=2
    lw  a0, 28(sp)
    lw  a1, 24(sp)
    call    __mulsi3@plt
    lw  a1, 32(sp)
    add a0, a0, a1
    slli    a0, a0, 2
    add a0, s2, a0
    lw  a1, 0(a0)
    mv  a0, s0
    call    printf@plt
    lw  a0, 32(sp)
    addi    a0, a0, 1
    sw  a0, 32(sp)
    j   .LBB0_28
.Lfunc_end0:
```

```
    .size   main, .Lfunc_end0-main
    .cfi_endproc
                                        # -- End function
    .type   A,@object                   # @A
    .bss
    .globl  A
    .p2align    4
A:
    .zero   800000
    .size   A, 800000

    .type   B,@object                   # @B
    .globl  B
    .p2align    4
B:
    .zero   800000
    .size   B, 800000

    .type   C,@object                   # @C
    .globl  C
    .p2align    4
C:
    .zero   800000
    .size   C, 800000

    .type   input,@object               # @input
    .globl  input
    .p2align    4
input:
    .zero   100000
    .size   input, 100000

    .type   .L_Const_String_,@object    # @_Const_String_
    .section    .rodata,"a",@progbits
.L_Const_String_:
    .asciz  "%d %d\n"
    .size   .L_Const_String_, 7

    .type   .L_Const_String_.1,@object  # @_Const_String_.1
.L_Const_String_.1:
    .asciz  "%d"
    .size   .L_Const_String_.1, 3

    .type   .L_Const_String_.2,@object  # @_Const_String_.2
.L_Const_String_.2:
    .asciz  "%d %d"
    .size   .L_Const_String_.2, 6

    .type   .L_Const_String_.3,@object  # @_Const_String_.3
.L_Const_String_.3:
    .asciz  "%d"
    .size   .L_Const_String_.3, 3
```

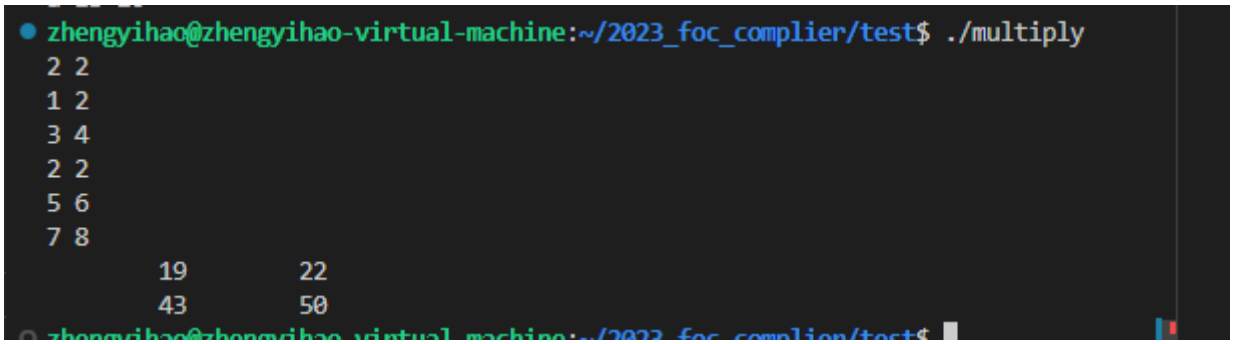
```
    .type   .L_Const_String_.4,@object      # @_Const_String_.4
    .p2align    4
.L_Const_String_.4:
    .asciz  "Incompatible Dimensions\n"
    .size   .L_Const_String_.4, 25

    .type   .L_Const_String_.5,@object      # @_Const_String_.5
.L_Const_String_.5:
    .asciz  "%10d"
    .size   .L_Const_String_.5, 5

    .type   .L_Const_String_.6,@object      # @_Const_String_.6
.L_Const_String_.6:
    .asciz  "\n"
    .size   .L_Const_String_.6, 2

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

- 运行结果



快排

- 测试代码

```
void quicksort(int A[10], int left, int right) {
    int i;
    int j;
    int x;
    int y;
    i = left;
    j = right;
    x = A[(left + right) / 2];
    while(i <= j) {
        while (A[i] < x) {
            i = i + 1;
        }
        while (x < A[j]) {
            j = j - 1;
```

```
        }
        if (i <= j) {
            y = A[i];
            A[i] = A[j];
            A[j] = y;
            i = i + 1;
            j = j - 1;
        }
    }
    if (left < j) {
        quicksort(A, left, j);
    }
    if (i < right) {
        quicksort(A, i, right);
    }
    return;
}

int main()
{
    int B[1000000];
    int N;
    scanf("%d", &N);
    int i = 0;
    while(i < N) {
        scanf("%d", &B[i]);
        i = i + 1;
    }
    int left = 0;
    int right = N - 1;
    quicksort(B, left, right);
    i = 0;
    while(i < N) {
        printf("%d\n", B[i]);
        i = i + 1;
    }

    return 0;
}
```

- LLVM IR

```
; ModuleID = 'main'
source_filename = "main"

@_Const_String_ = private constant [3 x i8] c"%d\00"
@_Const_String_.1 = private constant [3 x i8] c"%d\00"
@_Const_String_.2 = private constant [4 x i8] c"%d\0A\00"

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

```
declare i32 @scanf(...)

define void @quicksort(i32* %0, i32 %1, i32 %2) {
entry:
  %y = alloca i32, align 4
  %x = alloca i32, align 4
  %j = alloca i32, align 4
  %i = alloca i32, align 4
  %right = alloca i32, align 4
  %left = alloca i32, align 4
  %A = alloca i32*, align 8
  store i32* %0, i32** %A, align 8
  store i32 %1, i32* %left, align 4
  store i32 %2, i32* %right, align 4
  %LoadInst = load i32, i32* %left, align 4
  store i32 %LoadInst, i32* %i, align 4
  %LoadInst1 = load i32, i32* %right, align 4
  store i32 %LoadInst1, i32* %j, align 4
  %LoadInst2 = load i32, i32* %left, align 4
  %LoadInst3 = load i32, i32* %right, align 4
  %3 = add i32 %LoadInst2, %LoadInst3
  %4 = sdiv i32 %3, 2
  %5 = load i32*, i32** %A, align 8
  %tmparray = getelementptr inbounds i32, i32* %5, i32 %4
  %tmpvar = load i32, i32* %tmparray, align 4
  store i32 %tmpvar, i32* %x, align 4
  br label %WhileCond

WhileCond:                                        ; preds = %Merge, %entry
  %LoadInst4 = load i32, i32* %i, align 4
  %LoadInst5 = load i32, i32* %j, align 4
  %icmptmp = icmp sle i32 %LoadInst4, %LoadInst5
  br i1 %icmptmp, label %WhileLoop, label %WhileEnd39

WhileLoop:                                        ; preds = %WhileCond
  br label %WhileCond6

WhileCond6:                                       ; preds = %WhileLoop12, %WhileLoop
  %LoadInst7 = load i32, i32* %i, align 4
  %6 = load i32*, i32** %A, align 8
  %tmparray8 = getelementptr inbounds i32, i32* %6, i32 %LoadInst7
  %tmpvar9 = load i32, i32* %tmparray8, align 4
  %LoadInst10 = load i32, i32* %x, align 4
  %icmptmp11 = icmp slt i32 %tmpvar9, %LoadInst10
  br i1 %icmptmp11, label %WhileLoop12, label %WhileEnd

WhileLoop12:                                      ; preds = %WhileCond6
  %LoadInst13 = load i32, i32* %i, align 4
  %7 = add i32 %LoadInst13, 1
  store i32 %7, i32* %i, align 4
  br label %WhileCond6
```

```
WhileEnd:                                         ; preds = %WhileCond6
  br label %WhileCond14

WhileCond14:                                      ; preds = %WhileLoop20, %WhileEnd
  %LoadInst15 = load i32, i32* %x, align 4
  %LoadInst16 = load i32, i32* %j, align 4
  %8 = load i32*, i32** %A, align 8
  %tmparray17 = getelementptr inbounds i32, i32* %8, i32 %LoadInst16
  %tmpvar18 = load i32, i32* %tmparray17, align 4
  %icmptmp19 = icmp slt i32 %LoadInst15, %tmpvar18
  br i1 %icmptmp19, label %WhileLoop20, label %WhileEnd22

WhileLoop20:                                      ; preds = %WhileCond14
  %LoadInst21 = load i32, i32* %j, align 4
  %9 = sub i32 %LoadInst21, 1
  store i32 %9, i32* %j, align 4
  br label %WhileCond14

WhileEnd22:                                       ; preds = %WhileCond14
  %LoadInst23 = load i32, i32* %i, align 4
  %LoadInst24 = load i32, i32* %j, align 4
  %icmptmp25 = icmp sle i32 %LoadInst23, %LoadInst24
  br i1 %icmptmp25, label %Then, label %Else

Then:                                             ; preds = %WhileEnd22
  %LoadInst26 = load i32, i32* %i, align 4
  %10 = load i32*, i32** %A, align 8
  %tmparray27 = getelementptr inbounds i32, i32* %10, i32 %LoadInst26
  %tmpvar28 = load i32, i32* %tmparray27, align 4
  store i32 %tmpvar28, i32* %y, align 4
  %LoadInst29 = load i32, i32* %i, align 4
  %11 = load i32*, i32** %A, align 8
  %tmpvar30 = getelementptr inbounds i32, i32* %11, i32 %LoadInst29
  %LoadInst31 = load i32, i32* %j, align 4
  %12 = load i32*, i32** %A, align 8
  %tmparray32 = getelementptr inbounds i32, i32* %12, i32 %LoadInst31
  %tmpvar33 = load i32, i32* %tmparray32, align 4
  store i32 %tmpvar33, i32* %tmpvar30, align 4
  %LoadInst34 = load i32, i32* %j, align 4
  %13 = load i32*, i32** %A, align 8
  %tmpvar35 = getelementptr inbounds i32, i32* %13, i32 %LoadInst34
  %LoadInst36 = load i32, i32* %y, align 4
  store i32 %LoadInst36, i32* %tmpvar35, align 4
  %LoadInst37 = load i32, i32* %i, align 4
  %14 = add i32 %LoadInst37, 1
  store i32 %14, i32* %i, align 4
  %LoadInst38 = load i32, i32* %j, align 4
  %15 = sub i32 %LoadInst38, 1
  store i32 %15, i32* %j, align 4
  br label %Merge
```

```
Else:                                             ; preds = %WhileEnd22
  br label %Merge

Merge:                                            ; preds = %Else, %Then
  br label %WhileCond

WhileEnd39:                                       ; preds = %WhileCond
  %LoadInst40 = load i32, i32* %left, align 4
  %LoadInst41 = load i32, i32* %j, align 4
  %icmptmp42 = icmp slt i32 %LoadInst40, %LoadInst41
  br i1 %icmptmp42, label %Then43, label %Else47

Then43:                                           ; preds = %WhileEnd39
  %LoadInst44 = load i32*, i32** %A, align 8
  %LoadInst45 = load i32, i32* %left, align 4
  %LoadInst46 = load i32, i32* %j, align 4
  call void @quicksort(i32* %LoadInst44, i32 %LoadInst45, i32 %LoadInst46)
  br label %Merge48

Else47:                                           ; preds = %WhileEnd39
  br label %Merge48

Merge48:                                          ; preds = %Else47, %Then43
  %LoadInst49 = load i32, i32* %i, align 4
  %LoadInst50 = load i32, i32* %right, align 4
  %icmptmp51 = icmp slt i32 %LoadInst49, %LoadInst50
  br i1 %icmptmp51, label %Then52, label %Else56

Then52:                                           ; preds = %Merge48
  %LoadInst53 = load i32*, i32** %A, align 8
  %LoadInst54 = load i32, i32* %i, align 4
  %LoadInst55 = load i32, i32* %right, align 4
  call void @quicksort(i32* %LoadInst53, i32 %LoadInst54, i32 %LoadInst55)
  br label %Merge57

Else56:                                           ; preds = %Merge48
  br label %Merge57

Merge57:                                          ; preds = %Else56, %Then52
  ret void
}

define i32 @main() {
entry:
  %right = alloca i32, align 4
  %left = alloca i32, align 4
  %i = alloca i32, align 4
  %N = alloca i32, align 4
  %B = alloca [1000000 x i32], align 4
  %scanf = call i32 (...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@_Const_String_, i32 0, i32 0), i32* %N)
  store i32 0, i32* %i, align 4
```

```
  br label %WhileCond

WhileCond:                                        ; preds = %WhileLoop, %entry
  %LoadInst = load i32, i32* %i, align 4
  %LoadInst1 = load i32, i32* %N, align 4
  %icmptmp = icmp slt i32 %LoadInst, %LoadInst1
  br i1 %icmptmp, label %WhileLoop, label %WhileEnd

WhileLoop:                                        ; preds = %WhileCond
  %LoadInst2 = load i32, i32* %i, align 4
  %elePtr = getelementptr inbounds [1000000 x i32], [1000000 x i32]* %B, i32 0, i32
%LoadInst2
  %scanf3 = call i32 (...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@_Const_String_.1, i32 0, i32 0), i32* %elePtr)
  %LoadInst4 = load i32, i32* %i, align 4
  %0 = add i32 %LoadInst4, 1
  store i32 %0, i32* %i, align 4
  br label %WhileCond

WhileEnd:                                         ; preds = %WhileCond
  store i32 0, i32* %left, align 4
  %LoadInst5 = load i32, i32* %N, align 4
  %1 = sub i32 %LoadInst5, 1
  store i32 %1, i32* %right, align 4
  %arrayPtr = getelementptr [1000000 x i32], [1000000 x i32]* %B, i32 0, i32 0
  %LoadInst6 = load i32, i32* %left, align 4
  %LoadInst7 = load i32, i32* %right, align 4
  call void @quicksort(i32* %arrayPtr, i32 %LoadInst6, i32 %LoadInst7)
  store i32 0, i32* %i, align 4
  br label %WhileCond8

WhileCond8:                                       ; preds = %WhileLoop12, %WhileEnd
  %LoadInst9 = load i32, i32* %i, align 4
  %LoadInst10 = load i32, i32* %N, align 4
  %icmptmp11 = icmp slt i32 %LoadInst9, %LoadInst10
  br i1 %icmptmp11, label %WhileLoop12, label %WhileEnd15

WhileLoop12:                                      ; preds = %WhileCond8
  %LoadInst13 = load i32, i32* %i, align 4
  %tmparray = getelementptr inbounds [1000000 x i32], [1000000 x i32]* %B, i32 0,
i32 %LoadInst13
  %tmpvar = load i32, i32* %tmparray, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
i8]* @_Const_String_.2, i32 0, i32 0), i32 %tmpvar)
  %LoadInst14 = load i32, i32* %i, align 4
  %2 = add i32 %LoadInst14, 1
  store i32 %2, i32* %i, align 4
  br label %WhileCond8

WhileEnd15:                                       ; preds = %WhileCond8
  ret i32 0
}
```

- riscv32汇编指令

```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p0"
    .file   "main"
    .globl  quicksort                       # -- Begin function quicksort
    .p2align    2
    .type   quicksort,@function
quicksort:                                  # @quicksort
    .cfi_startproc
# %bb.0:                                    # %entry
    addi    sp, sp, -32
    .cfi_def_cfa_offset 32
    sw  ra, 28(sp)                          # 4-byte Folded Spill
    .cfi_offset ra, -4
    sw  a0, 0(sp)
    sw  a1, 4(sp)
    sw  a2, 8(sp)
    sw  a1, 12(sp)
    sw  a2, 16(sp)
    add a1, a1, a2
    srli    a2, a1, 31
    add a1, a1, a2
    slli    a1, a1, 1
    andi    a1, a1, -4
    add a0, a0, a1
    lw  a0, 0(a0)
    sw  a0, 20(sp)
.LBB0_1:                                    # %whileCond
                                            # =>This Loop Header: Depth=1
                                            #     Child Loop BB0_2 Depth 2
                                            #     Child Loop BB0_5 Depth 2
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    blt a1, a0, .LBB0_8
.LBB0_2:                                    # %whileCond6
                                            #   Parent Loop BB0_1 Depth=1
                                            # =>  This Inner Loop Header: Depth=2
    lw  a0, 12(sp)
    lw  a1, 0(sp)
    slli    a0, a0, 2
    add a0, a1, a0
    lw  a0, 0(a0)
    lw  a1, 20(sp)
    bge a0, a1, .LBB0_5
# %bb.3:                                    # %whileLoop12
                                            #   in Loop: Header=BB0_2 Depth=2
```

```
    lw  a0, 12(sp)
    addi    a0, a0, 1
    sw  a0, 12(sp)
    j   .LBB0_2
.LBB0_4:                                # %WhileLoop20
                                        #   in Loop: Header=BB0_5 Depth=2
    lw  a0, 16(sp)
    addi    a0, a0, -1
    sw  a0, 16(sp)
.LBB0_5:                                # %WhileCond14
                                        #   Parent Loop BB0_1 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
    lw  a0, 16(sp)
    lw  a1, 0(sp)
    lw  a2, 20(sp)
    slli    a0, a0, 2
    add a0, a1, a0
    lw  a0, 0(a0)
    blt a2, a0, .LBB0_4
# %bb.6:                                # %WhileEnd22
                                        #   in Loop: Header=BB0_1 Depth=1
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    blt a1, a0, .LBB0_1
# %bb.7:                                # %Then
                                        #   in Loop: Header=BB0_1 Depth=1
    lw  a0, 12(sp)
    lw  a1, 0(sp)
    slli    a0, a0, 2
    add a0, a1, a0
    lw  a2, 0(a0)
    lw  a3, 16(sp)
    sw  a2, 24(sp)
    slli    a2, a3, 2
    add a1, a1, a2
    lw  a1, 0(a1)
    sw  a1, 0(a0)
    lw  a0, 16(sp)
    lw  a1, 0(sp)
    lw  a2, 24(sp)
    slli    a0, a0, 2
    add a0, a1, a0
    sw  a2, 0(a0)
    lw  a0, 12(sp)
    lw  a1, 16(sp)
    addi    a0, a0, 1
    sw  a0, 12(sp)
    addi    a0, a1, -1
    sw  a0, 16(sp)
    j   .LBB0_1
.LBB0_8:                                # %WhileEnd39
    lw  a0, 4(sp)
```

```
    lw  a1, 16(sp)
    bge a0, a1, .LBB0_10
# %bb.9:                                # %Then43
    lw  a0, 0(sp)
    lw  a1, 4(sp)
    lw  a2, 16(sp)
    call    quicksort@plt
.LBB0_10:                               # %Merge48
    lw  a0, 12(sp)
    lw  a1, 8(sp)
    bge a0, a1, .LBB0_12
# %bb.11:                               # %Then52
    lw  a0, 0(sp)
    lw  a1, 12(sp)
    lw  a2, 8(sp)
    call    quicksort@plt
.LBB0_12:                               # %Merge57
    lw  ra, 28(sp)                      # 4-byte Folded Reload
    addi    sp, sp, 32
    ret
.Lfunc_end0:
    .size   quicksort, .Lfunc_end0-quicksort
    .cfi_endproc
                                        # -- End function
    .globl  main                            # -- Begin function main
    .p2align    2
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    addi    sp, sp, -2032
    .cfi_def_cfa_offset 2032
    sw  ra, 2028(sp)                    # 4-byte Folded Spill
    sw  s0, 2024(sp)                    # 4-byte Folded Spill
    sw  s1, 2020(sp)                    # 4-byte Folded Spill
    .cfi_offset ra, -4
    .cfi_offset s0, -8
    .cfi_offset s1, -12
    lui a0, 976
    addi    a0, a0, 304
    sub sp, sp, a0
    .cfi_def_cfa_offset 4000032
    lui a0, %hi(.L_Const_String_)
    addi    a0, a0, %lo(.L_Const_String_)
    lui a1, 977
    addi    a1, a1, -1788
    add a1, sp, a1
    call    scanf@plt
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    sw  zero, 0(a0)
```

```
    addi    s1, sp, 4
    lui a0, %hi(.L_Const_String_.1)
    addi    s0, a0, %lo(.L_Const_String_.1)
.LBB1_1:                                # %WhileCond
                                        # =>This Inner Loop Header: Depth=1
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    lui a1, 977
    addi    a1, a1, -1788
    add a1, sp, a1
    lw  a1, 0(a1)
    bge a0, a1, .LBB1_3
# %bb.2:                                # %WhileLoop
                                        #   in Loop: Header=BB1_1 Depth=1
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    slli    a0, a0, 2
    add a1, s1, a0
    mv  a0, s0
    call    scanf@plt
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    addi    a0, a0, 1
    lui a1, 977
    addi    a1, a1, -1784
    add a1, sp, a1
    sw  a0, 0(a1)
    j   .LBB1_1
.LBB1_3:                                # %WhileEnd
    lui a0, 977
    addi    a0, a0, -1788
    add a0, sp, a0
    lw  a0, 0(a0)
    lui a1, 977
    addi    a1, a1, -1780
    add a1, sp, a1
    sw  zero, 0(a1)
    addi    a2, a0, -1
    lui a0, 977
    addi    a0, a0, -1776
    add a0, sp, a0
    sw  a2, 0(a0)
    addi    a0, sp, 4
    addi    s1, sp, 4
    li  a1, 0
    call    quicksort@plt
```

```
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    sw  zero, 0(a0)
    lui a0, %hi(.L_Const_String_.2)
    addi    s0, a0, %lo(.L_Const_String_.2)
.LBB1_4:                                # %WhileCond8
                                        # =>This Inner Loop Header: Depth=1
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    lui a1, 977
    addi    a1, a1, -1788
    add a1, sp, a1
    lw  a1, 0(a1)
    bge a0, a1, .LBB1_6
# %bb.5:                                # %WhileLoop12
                                        #   in Loop: Header=BB1_4 Depth=1
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    slli    a0, a0, 2
    add a0, s1, a0
    lw  a1, 0(a0)
    mv  a0, s0
    call    printf@plt
    lui a0, 977
    addi    a0, a0, -1784
    add a0, sp, a0
    lw  a0, 0(a0)
    addi    a0, a0, 1
    lui a1, 977
    addi    a1, a1, -1784
    add a1, sp, a1
    sw  a0, 0(a1)
    j   .LBB1_4
.LBB1_6:                                # %WhileEnd15
    li  a0, 0
    lui a1, 976
    addi    a1, a1, 304
    add sp, sp, a1
    lw  ra, 2028(sp)                    # 4-byte Folded Reload
    lw  s0, 2024(sp)                    # 4-byte Folded Reload
    lw  s1, 2020(sp)                    # 4-byte Folded Reload
    addi    sp, sp, 2032
    ret
.Lfunc_end1:
    .size   main, .Lfunc_end1-main
    .cfi_endproc
                                        # -- End function
```

```
    .type   .L_Const_String_,@object        # @_Const_String_
    .section    .rodata,"a",@progbits
.L_Const_String_:
    .asciz  "%d"
    .size   .L_Const_String_, 3

    .type   .L_Const_String_.1,@object      # @_Const_String_.1
.L_Const_String_.1:
    .asciz  "%d"
    .size   .L_Const_String_.1, 3

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

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

- 运行结果