实验报告

实验三：中间代码生成 1

1.1 实验目的和要求

本实验旨在通过实践操作深入理解LLVM IR（中间表示）的生成和分析过程，具体目标如下：

1. 掌握LLVM IR的基本语法和结构。

2. 学习如何从高级语言（如C++）代码生成LLVM IR代码。

3. 分析LLVM IR代码，理解其与源代码的对应关系。

4. 掌握`getelementptr`指令的两种不同用法及其适用场景。

1.2 实验环境

1. 硬件环境：鲲鹏开发板。

2. 软件环境：openEuler操作系统。

3. 开发工具：LLVM编译器框架。

4. 测试环境：提供一组C++测试代码。

1.3 实验内容

本实验内容包括以下几个部分：

1. 从C++代码生成LLVM IR代码。

2. 分析生成的LLVM IR代码，理解其结构和语义。

3. 掌握`getelementptr`指令的两种用法及其区别。

1.4 实验过程

1.4.1 生成LLVM IR代码

通过LLVM编译器框架，将C++代码转换为LLVM IR代码。实验中提供了多个C++代码示例，包括赋值语句、函数调用、条件语句和循环语句等。

1.4.2 分析LLVM IR代码

分析生成的LLVM IR代码，理解其与C++代码的对应关系。特别关注`alloca`、`load`、`store`、`br`、`cond\_br`、`icmp`、`fcmp`、`add`、`sub`、`mul`、`div`、`ret`等指令的用法。

1.4.3 掌握`getelementptr`指令

通过实验，掌握`getelementptr`指令的两种用法及其区别。第一种方法是针对数组或结构体等复合数据类型的寻址，第二种方法是针对单个元素的寻址。

1.5 实验结果

实验中生成了多个LLVM IR代码示例，包括：

- `assign\_hand.ll`与`assign\_generator.cpp`：展示了赋值语句的LLVM IR代码。

- `fun\_hand.ll`与`fun\_generator.cpp`：展示了函数调用的LLVM IR代码。

- `if\_hand.ll`与`if\_generator.cpp`：展示了条件语句的LLVM IR代码。

- `while\_hand.ll`与`stu\_while\_generate.cpp`：展示了循环语句的LLVM IR代码。

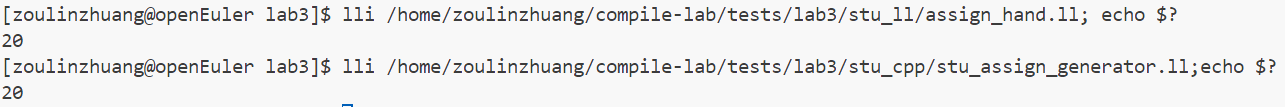
assign\_hand.ll与assign\_generator.cpp

assign\_hand.ll

|  |
| --- |
| ; ModuleID = '/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c'  source\_filename = "/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c"  target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32:64-S128"  target triple = "aarch64-unknown-linux-gnu"  ; Function Attrs: noinline nounwind optnone uwtable  define dso\_local i32 @main() #0 {  %1 = alloca [10 x i32], align 4  %2 = getelementptr inbounds [10 x i32], ptr %1, i64 0, i64 0  store i32 10, ptr %2, align 4  %3= load i32, ptr %2, align 4  %4 = getelementptr inbounds [10 x i32], ptr %1, i64 0, i64 1  %5 = mul nsw i32 %3, 2  store i32 %5, ptr %4, align 4  %6 = load i32, ptr %4, align 4  ret i32 %5  }  attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="non-leaf" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="generic" "target-features"="+fp-armv8,+neon,+outline-atomics,+v8a,-fmv" }  !llvm.module.flags = !{!0, !1, !2, !3, !4}  !llvm.ident = !{!5}  !0 = !{i32 1, !"wchar\_size", i32 4}  !1 = !{i32 8, !"PIC Level", i32 2}  !2 = !{i32 7, !"PIE Level", i32 2}  !3 = !{i32 7, !"uwtable", i32 2}  !4 = !{i32 7, !"frame-pointer", i32 1}  !5 = !{!"BiSheng Enterprise 4.0.0.B014 clang version 17.0.6 (0ac87bc45325)"} |

assign\_generator.cpp

|  |
| --- |
| #include "BasicBlock.h"  #include "Constant.h"  #include "Function.h"  #include "IRBuilder.h"  #include "Module.h"  #include "Type.h"  #include <iostream>  #include <memory>  #ifdef DEBUG *// 用于调试信息,大家可以在编译过程中通过" -DDEBUG"来开启这一选项*  #define DEBUG\_OUTPUT std::cout << \_\_LINE\_\_ << std::endl; *// 输出行号的简单示例*  #else  #define DEBUG\_OUTPUT  #endif  #define CONST\_INT(num) \  ConstantInt::get(num, module)  #define CONST\_FP(num) \  ConstantFP::get(num, module) *// 得到常数值的表示,方便后面多次用到*  **int** **main**() {  **auto** **module** = **new** Module("assign code"); *// 生成assign的代码*  **auto** builder = **new** IRBuilder(nullptr, **module**);  Type \*Int32Type = Type::get\_int32\_type(**module**);  *// BB的名字在生成中无所谓,但是可以方便阅读*    *// main函数*  **auto** mainFun = Function::create(FunctionType::get(Int32Type, {}),  "main", **module**);  **auto** bb = BasicBlock::create(**module**, "entry", mainFun);    *// BasicBlock的名字在生成中无所谓,但是可以方便阅读*  builder->set\_insert\_point(bb);  **auto** retAlloca = builder->create\_alloca(Int32Type);  **auto** \*arrayType = ArrayType::get(Int32Type,10);  **auto** a\_array = builder->create\_alloca(arrayType);  **auto** aa = builder->create\_gep(a\_array,{CONST\_INT(0),CONST\_INT(0)});  builder->create\_store(CONST\_INT(10),aa);  **auto** a0load = builder->create\_load(aa);  **auto** a1data = builder->create\_imul(a0load,CONST\_INT(2));  **auto** a1p = builder->create\_gep(a\_array,{CONST\_INT(0),CONST\_INT(1)});  builder->create\_store(a1data,a1p);  **auto** a1load =builder->create\_load(a1p);  **auto** retp = builder->create\_alloca(Int32Type);  builder->create\_store(a1load, retp);  **auto** retdata=builder->create\_load(retp);  builder->create\_ret(retdata);  std::cout<<**module**->print();  **delete** **module**;  **return** 0;  } |



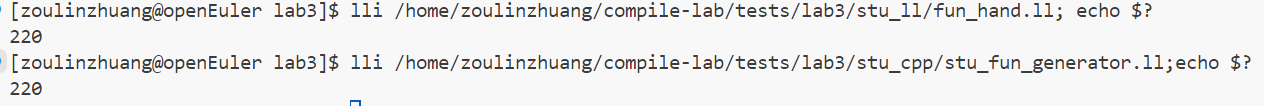
fun\_hand.ll与stu\_fun\_generator.ll

fun\_hand.ll

|  |
| --- |
| ; ModuleID = '/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c'  source\_filename = "/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c"  target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32:64-S128"  target triple = "aarch64-unknown-linux-gnu"  ; Function Attrs: noinline nounwind optnone uwtable  define dso\_local i32 @callee(i32 noundef %0) #0 {  %2 =alloca i32, align 4  store i32 %0,ptr %2,align 4  %3 = load i32,ptr %2,align 4  %4 = mul nsw i32 %3, 2    ret i32 %4  }  ; Function Attrs: noinline nounwind optnone uwtable  define dso\_local i32 @main() #0 {  %1 = alloca i32, align 4  store i32 110, ptr %1, align 4  %2 =load i32, ptr %1, align 4  %3 = call i32 @callee(i32 noundef %2)  ret i32 %3  }  attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="non-leaf" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="generic" "target-features"="+fp-armv8,+neon,+outline-atomics,+v8a,-fmv" }  !llvm.module.flags = !{!0, !1, !2, !3, !4}  !llvm.ident = !{!5}  !0 = !{i32 1, !"wchar\_size", i32 4}  !1 = !{i32 8, !"PIC Level", i32 2}  !2 = !{i32 7, !"PIE Level", i32 2}  !3 = !{i32 7, !"uwtable", i32 2}  !4 = !{i32 7, !"frame-pointer", i32 1}  !5 = !{!"BiSheng Enterprise 4.0.0.B014 clang version 17.0.6 (0ac87bc45325)"} |

fun\_generator.cpp

|  |
| --- |
| #include "BasicBlock.h"  #include "Constant.h"  #include "Function.h"  #include "IRBuilder.h"  #include "Module.h"  #include "Type.h"  #include <iostream>  #include <memory>  #ifdef DEBUG *// 用于调试信息,大家可以在编译过程中通过" -DDEBUG"来开启这一选项*  #define DEBUG\_OUTPUT std::cout << \_\_LINE\_\_ << std::endl; *// 输出行号的简单示例*  #else  #define DEBUG\_OUTPUT  #endif  #define CONST\_INT(num) \  ConstantInt::get(num, module)  #define CONST\_FP(num) \  ConstantFP::get(num, module) *// 得到常数值的表示,方便后面多次用到*  **int** **main**() {  **auto** **module** = **new** Module("fun code"); *// 生成assign的代码*  **auto** builder = **new** IRBuilder(nullptr, **module**);  Type \*Int32Type = Type::get\_int32\_type(**module**);      std::vector<Type \*> **Ints**(1, Int32Type);  **auto** callee\_funtype=FunctionType::get(Int32Type,Ints);  **auto** callee\_fun=Function::create(FunctionType::get(Int32Type,{Int32Type}),"callee",**module**);  **auto** bb =BasicBlock::create(**module**,"entry",callee\_fun);  builder->set\_insert\_point(bb);  **auto** a\_ca =builder->create\_alloca(Int32Type);  std::vector<Value \*> args; *// 获取gcd函数的形参,通过Function中的iterator*  **for** (**auto** arg = callee\_fun->arg\_begin(); arg != callee\_fun->arg\_end(); arg++) {  args.push\_back(\*arg); *// \* 号运算符是从迭代器中取出迭代器当前指向的元素*  }  builder->create\_store(args[0],a\_ca);  **auto** aload=builder->create\_load(a\_ca);  **auto** ans=builder->create\_imul(CONST\_INT(2),aload);  **auto** retAlloca = builder->create\_alloca(Int32Type); *// 在内存中分配返回值的位置*  builder->create\_store(ans,retAlloca);  **auto** retload=builder->create\_load(retAlloca);  builder->create\_ret(retload);  **auto** main\_fun=Function::create(FunctionType::get(Int32Type,{ }),"main",**module**);  **auto** bb1=BasicBlock::create(**module**,"main",main\_fun);  builder->set\_insert\_point(bb1);  **auto** tmp=builder->create\_alloca(Int32Type);  builder->create\_store(CONST\_INT(110),tmp);  **auto** tmpload=builder->create\_load(tmp);  **auto** res=builder->create\_call(callee\_fun,{tmpload});  builder->create\_ret(res);  std::cout<<**module**->print();  **delete** **module**;  **return** 0;  *// BB的名字在生成中无所谓,但是可以方便阅读*  } |



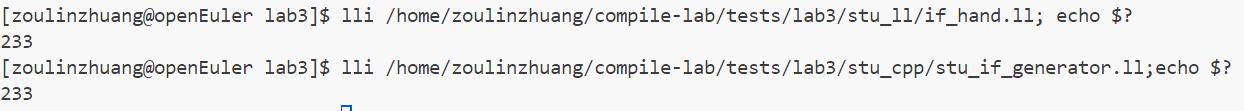
if\_hand.ll与if\_generator.cpp

if\_hand.ll

|  |
| --- |
| ; ModuleID = '/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c'  source\_filename = "/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c"  target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32:64-S128"  target triple = "aarch64-unknown-linux-gnu"  ; Function Attrs: noinline nounwind optnone uwtable  define dso\_local i32 @main() #0 {  %1 = alloca float, align 4  store float 0x40163851E0000000, ptr %1, align 4  %2 = alloca i32 ,align 4  store i32 233 ,ptr %2,align 4  %3= alloca i32 ,align 4  store i32 0 ,ptr %3,align 4  %4= load float,ptr %1,align 4  %5 = fcmp contract ogt float %4, 1.000000e+00  br i1 %5,label %6,label %8  6:  %7=load i32,ptr %2,align 4  ret i32 %7    8:  %9=load i32,ptr %3,align 4  ret i32 %9  }  attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="non-leaf" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="generic" "target-features"="+fp-armv8,+neon,+outline-atomics,+v8a,-fmv" }  !llvm.module.flags = !{!0, !1, !2, !3, !4}  !llvm.ident = !{!5}  !0 = !{i32 1, !"wchar\_size", i32 4}  !1 = !{i32 8, !"PIC Level", i32 2}  !2 = !{i32 7, !"PIE Level", i32 2}  !3 = !{i32 7, !"uwtable", i32 2}  !4 = !{i32 7, !"frame-pointer", i32 1}  !5 = !{!"BiSheng Enterprise 4.0.0.B014 clang version 17.0.6 (0ac87bc45325)"} |

if\_generator.cpp

|  |
| --- |
| #include "BasicBlock.h"  #include "Constant.h"  #include "Function.h"  #include "IRBuilder.h"  #include "Module.h"  #include "Type.h"  #include <iostream>  #include <memory>  #ifdef DEBUG *// 用于调试信息,大家可以在编译过程中通过" -DDEBUG"来开启这一选项*  #define DEBUG\_OUTPUT std::cout << \_\_LINE\_\_ << std::endl; *// 输出行号的简单示例*  #else  #define DEBUG\_OUTPUT  #endif  #define CONST\_INT(num) \  ConstantInt::get(num, module)  #define CONST\_FP(num) \  ConstantFP::get(num, module) *// 得到常数值的表示,方便后面多次用到*  **int** **main**() {  **auto** **module** = **new** Module("if code"); *// 生成assign的代码*  **auto** builder = **new** IRBuilder(nullptr,**module**);  Type\* Int32Type = Type::get\_int32\_type(**module**);  Type\* floatType = Type::get\_float\_type(**module**);  *// BB的名字在生成中无所谓,但是可以方便阅读*    *// main函数*  **auto** mainFun = Function::create(FunctionType::get(Int32Type, {}),  "main", **module**);  **auto** bb = BasicBlock::create(**module**, "entry", mainFun);    *// BasicBlock的名字在生成中无所谓,但是可以方便阅读*  builder->set\_insert\_point(bb);  **auto** retAlloca = builder->create\_alloca(Int32Type);  **auto** ret\_ca=builder->create\_alloca(Int32Type);  **auto** a\_ca=builder->create\_alloca(floatType);  builder->create\_store(CONST\_FP(5.555),a\_ca);  **auto** aload=builder->create\_load(a\_ca);  **auto** cmp=builder->create\_fcmp\_gt(aload,CONST\_FP(1.000));  **auto** true\_bb=BasicBlock::create(**module**,"trueBB",mainFun);  **auto** false\_bb=BasicBlock::create(**module**,"false\_BB",mainFun);  **auto** ret\_bb=BasicBlock::create(**module**,"retBB",mainFun);  **auto** br=builder->create\_cond\_br(cmp,true\_bb,false\_bb);  builder->set\_insert\_point(true\_bb);  builder->create\_store(CONST\_INT(233),ret\_ca);  builder->create\_br(ret\_bb);  builder->set\_insert\_point(false\_bb);  builder->create\_store(CONST\_INT(0),ret\_ca);  builder->create\_br(ret\_bb);  builder->set\_insert\_point(ret\_bb);  **auto** retload=builder->create\_load(ret\_ca);  builder->create\_ret(retload);  std::cout<<**module**->print();  **delete** **module**;  **return** 0;  } |

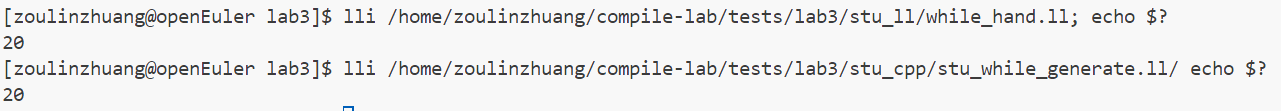


while\_hand.ll

|  |
| --- |
| ; ModuleID = '/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c'  source\_filename = "/home/zoulinzhuang/compiler-lab-24fall/tests/lab3/ta\_gcd/assign.c"  target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32:64-S128"  target triple = "aarch64-unknown-linux-gnu"  ; Function Attrs: noinline nounwind optnone uwtable  define dso\_local i32 @main() #0 {  %1 = alloca i32, align 4  %2 = alloca i32, align 4  store i32 10, ptr %1, align 4  store i32 0, ptr %2, align 4  br label %3  3:  %4 = load i32, ptr %2, align 4  %5 = icmp slt i32 %4, 10  br i1 %5, label %6, label %10  6:  %7 = load i32, ptr %1, align 4  %8 = add nsw i32 %7, 1  store i32 %8, ptr %1, align 4  %9 = add nsw i32 %4, 1  store i32 %9, ptr %2, align 4  br label %3  10:  %11 = load i32, ptr %1, align 4  ret i32 %11  }  attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="non-leaf" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="generic" "target-features"="+fp-armv8,+neon,+outline-atomics,+v8a,-fmv" }  !llvm.module.flags = !{!0, !1, !2, !3, !4}  !llvm.ident = !{!5}  !0 = !{i32 1, !"wchar\_size", i32 4}  !1 = !{i32 8, !"PIC Level", i32 2}  !2 = !{i32 7, !"PIE Level", i32 2}  !3 = !{i32 7, !"uwtable", i32 2}  !4 = !{i32 7, !"frame-pointer", i32 1}  !5 = !{!"BiSheng Enterprise 4.0.0.B014 clang version 17.0.6 (0ac87bc45325)"} |

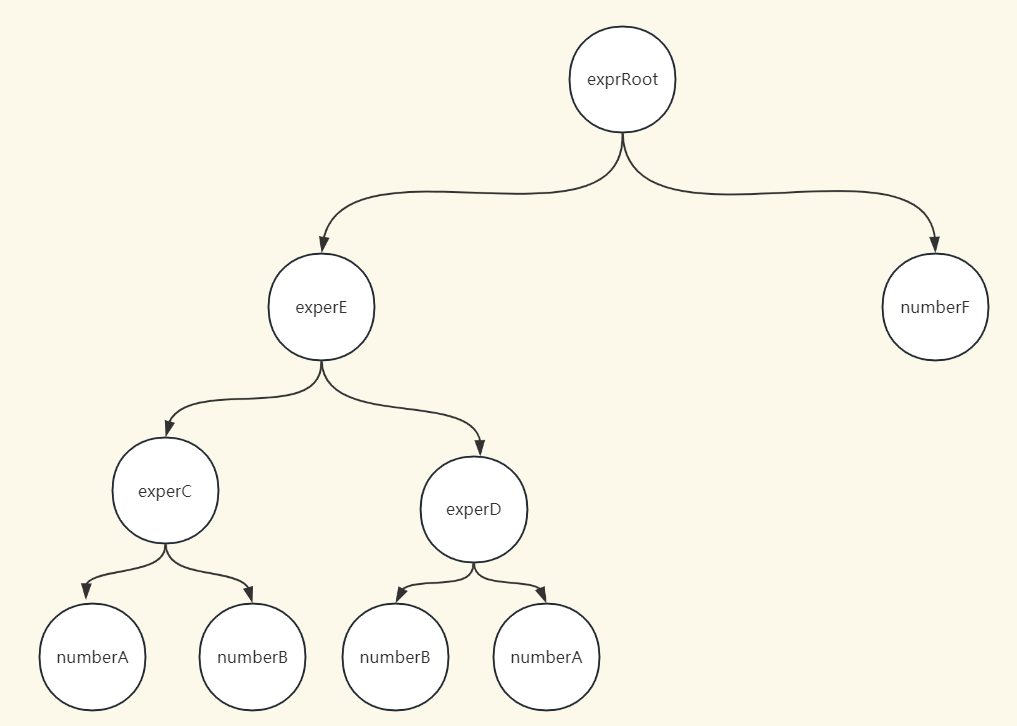
stu\_while\_generate.ll

|  |
| --- |
| #include "BasicBlock.h"  #include "Constant.h"  #include "Function.h"  #include "IRBuilder.h"  #include "Module.h"  #include "Type.h"  #include <iostream>  #include <memory>  #ifdef DEBUG *// 用于调试信息,大家可以在编译过程中通过" -DDEBUG"来开启这一选项*  #define DEBUG\_OUTPUT std::cout << \_\_LINE\_\_ << std::endl; *// 输出行号的简单示例*  #else  #define DEBUG\_OUTPUT  #endif  #define CONST\_INT(num) \  ConstantInt::get(num, module)  #define CONST\_FP(num) \  ConstantFP::get(num, module) *// 得到常数值的表示,方便后面多次用到*  **int** **main**() {  **auto** **module** = **new** Module("while code"); *// 生成assign的代码*  **auto** builder = **new** IRBuilder(nullptr,**module**);  Type\* Int32Type = Type::get\_int32\_type(**module**);    *// main函数*  **auto** mainFun = Function::create(FunctionType::get(Int32Type, {}),  "main", **module**);  **auto** bb = BasicBlock::create(**module**, "entry", mainFun);    *// BasicBlock的名字在生成中无所谓,但是可以方便阅读*  builder->set\_insert\_point(bb);  **auto** retAlloca = builder->create\_alloca(Int32Type);  **auto** ret\_ca=builder->create\_alloca(Int32Type);  **auto** a\_ca=builder->create\_alloca(Int32Type);  builder->create\_store(CONST\_INT(10),a\_ca);  **auto** i\_ca=builder->create\_alloca(Int32Type);  builder->create\_store(CONST\_INT(0),i\_ca);    *//*    **auto** true\_bb=BasicBlock::create(**module**,"true\_bb",mainFun);  **auto** false\_bb=BasicBlock::create(**module**,"false\_bb",mainFun);  **auto** run\_bb=BasicBlock::create(**module**,"run\_bb",mainFun);  builder->create\_br(run\_bb);  builder->set\_insert\_point(run\_bb);  **auto** iload=builder->create\_load(i\_ca);  **auto** cmp=builder->create\_icmp\_lt(iload,CONST\_INT(10));  **auto** br=builder->create\_cond\_br(cmp,true\_bb,false\_bb);    builder->set\_insert\_point(true\_bb);  **auto** aload=builder->create\_load(a\_ca);  **auto** ans=builder->create\_iadd(aload,CONST\_INT(1));  builder->create\_store(ans,a\_ca);  **auto** res=builder->create\_iadd(iload,CONST\_INT(1));  builder->create\_store(res,i\_ca);  builder->create\_br(run\_bb);  builder->set\_insert\_point(false\_bb);  **auto** retload=builder->create\_load(a\_ca);  builder->create\_ret(retload);  std::cout<<**module**->print();  **delete** **module**;  **return** 0;  } |



1.6思考题

1.6.1 Visitor Pattern

请指出visitor.cpp中，treeVisitor.visit(exprRoot)执行时，以下几个Node的遍历序列:numberA、numberB、exprC、exprD、exprE、numberF、exprRoot。 序列请按如下格式指明： exprRoot->numberF->exprE->numberA->exprD 

|  |
| --- |
| **class** **TreeVisitorCalculator** : **public** TreeVisitor { *// Implements triggering of all*  *// kind of elements (nodes)*  **public**:  **int** **visit**(AddSubNode& node) **override** {  **auto** right = node.rightNode.accept(\***this**);  **auto** left = node.leftNode.accept(\***this**);  **if** (node.op == "add") {  **return** left + right;  }  **else** {  **return** left - right;  }  }  **int** **visit**(NumberNode& node) **override** {  **return** node.number;  }  **int** **visit**(MulDivNode& node) **override** {  **auto** left = node.leftNode.accept(\***this**);  **auto** right = node.rightNode.accept(\***this**);  **if** (node.op == "mul") {  **return** left \* right;  }  **else** {  **return** left / right;  }  }  }; |

对于AddSubNode，在visit函数中是先访问右子节点在访问左子节点

对于NumberNode，在visit函数中是直接访问对应数字的值

对于MulDivNode，在visit函数中是先访问左子节点再访问右子节点的。

**观察构造表达式节点的代码：** **观察构造表达式节点的代码：**

|  |
| --- |
| int main() {  // construct the expression nodes and the tree  // the expression: 4 \* 2 - 2 / 4 + 5  auto numberA = NumberNode(4);  auto numberB = NumberNode(2);  auto exprC = MulDivNode(numberA, numberB, "mul");  auto exprD = MulDivNode(numberB, numberA, "div");  auto exprE = AddSubNode(exprC, exprD, "sub");  auto numberF = NumberNode(5);  auto exprRoot = AddSubNode(exprE, numberF, "add");  TreeVisitorCalculator treeVisitor;  // traverse the tree and calculate  int result = treeVisitor.visit(exprRoot);  std::cout << "4 \* 2 - 2 / 4 + 5 evaluates: " << result << std::endl;  return 0;  } |

**访问顺序**

根据上述分析的得知，访问顺序为：

|  |
| --- |
| exprRoot→numberF→experE→experD→numberB→numberA→experC→numberA→numberB |

1.6.2 getelementptr

请给出IR.md中提到的两种getelementptr用法的区别,并稍加解释:

|  |
| --- |
| %2 = getelementptr [10 x i32], [10 x i32]\* %1, i32 0, i32 %0  %2 = getelementptr i32, i32\* %1 i32 %0 |

1.第一种方法，%1是我们的基址，两个索引0和%0。这是一个数组，但是getelementptr是需要通过指针访问它。第一个索引用于分割指针，第二个用于索引数组本身，对应于C语言取出a[0]的地址。

2.第二种方法，%1是我们的基址，一个索引%0，也是对应于C语言取出a[0]的地址。

由上分析其实可以看出，这两种用法最后的得到的结果是相同的，只不过两种方法适用于不同的情况。第一种方法是[10 x i32]\*类型，素以指针指向的地址内容认可以视为指针，所以需要多一重偏移得到数组的指针。这种方法使用与数组的寻址，也适用于结构体、向量等数据结构的寻址。

1.7 实验难点与反馈

1. 在生成`if\_hand.ll`文件中的浮点数赋值语句时，遇到了编译错误。原因是5.555转换成二进制是无限循环的，而编译器只接受精确的小数。通过查阅资料，使用了十六进制表示法解决了问题。具体修改为：`store float 0x40163851E0000000,float\*%1;`。

2. `getelementptr`指令的理解较为困难，通过查阅资料和实验，最终理解了其两种用法的区别和适用场景。

1.8 实验总结

通过本次实验，深入理解了LLVM IR的生成过程和结构，掌握了LLVM IR中核心指令的用法。实验不仅加深了对编译器工作原理的理解，而且提高了对计算机系统和汇编代码的认识。通过亲手构造IR代码，对计算机系统的底层运作有了更为直观的感受，为后续的编译器开发和优化打下了坚实的基础。

姓名：邹林壮 学号：202208040412