lab2 实验报告

学号 姓名

问题1: 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`

  这里的%1是一个指向 指向有10个i32元素数组的指针 的指针，第一个i32 0对这一层引用解引用(即偏移0)，第二个i32%0才开始对这个指向有10个i32元素数组的指针进行操作。

- `%2 = getelementptr i32, i32* %1 i32 %0`

  这里的%1是一个指向i32的指针，可以直接进行地址计算

  因此，两个用法区别在于是否需要对传入的指针进行解引用，需要，则有两个值，不需要，则只需要一个值。

问题2: cpp 与 .ll 的对应

请说明你的 cpp 代码片段和 .ll 的每个 BasicBlock 的对应关系。(在代码中标明了)

1. assign

```
int main()
{
    auto module = new Module("cminus code");
    auto builder = new IRBuilder(nullptr, module);
    Type *Int32Type = Type::get_int32_type(module);

    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                    "main", module);
    auto bb = BasicBlock::create(module, "entry", mainFun);
    builder->set_insert_point(bb);
    //create array
    auto *arrayType = ArrayType::get(Int32Type, 10);
    //auto initializer = ConstantZero::get(Int32Type, module);
    auto array_addr = builder->create_alloca(arrayType);

    auto array_gep0 = builder->create_gep(array_addr,
{CONST_INT(0),CONST_INT(0)});

    builder->create_store(CONST_INT(10),array_gep0);

    //auto a_0allo=builder->create_alloca(Int32Type);

    //auto array_gep0 = builder->create_gep(array_addr,{CONST_INT(0)});

    auto a_0load=builder->create_load(array_gep0);

    auto timesa_0 = builder->create_imul(a_0load,CONST_INT(2));

    auto array_gep1 = builder->create_gep(array_addr,
{CONST_INT(0),CONST_INT(1)});

    builder->create_store(timesa_0,array_gep1);

    builder->create_ret(timesa_0);
```

```
    std::cout<<module->print();
    delete module;
    return 0;
}
```

1个基本块，该bb即对应整个函数体

```
define dso_local i32 @main(){
    %1 = alloca [10 x i32],align 16
    %2 = getelementptr [10 x i32], [10 x i32]* %1, i32 0, i32 0
    store i32 10 ,i32* %2,align 4
    %3 = load i32, i32* %2,align 4
    %4 = mul i32 %3, 2
    %5 = getelementptr [10 x i32], [10 x i32]* %1, i64 0, i64 1
    store i32 %4, i32* %5, align 4
    %6 = load i32, i32* %5,align 4
    ret i32 %6
}
```

2. fun

```
int main()
{
    auto module = new Module("cminus code");
    auto builder = new IRBuilder(nullptr, module);
    Type *Int32Type = Type::get_int32_type(module);

    //create fun callee
    std::vector<Type *>Ints(1,Int32Type);//params type
    auto calleeFunTy = FunctionType::get(Int32Type,Ints);
    auto calleeFun = Function::create(calleeFunTy,"callee",module);
    //now is fun
    //bb1
    auto bb = BasicBlock::create(module, "entry", calleeFun);
    builder->set_insert_point(bb);

    auto ret = builder->create_alloca(Int32Type); // 在内存中分配返回值的位置
    auto aAlloca = builder->create_alloca(Int32Type);  // 在内存中分配参数a
的位置

    std::vector<Value *> args;  // 获取callee函数的形参,通过Function中的
iterator
    for (auto arg = calleeFun->arg_begin(); arg != calleeFun->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
    }
    //just 1 param
    builder->create_store(args[0],aAlloca);
    //load a
    auto aLoad = builder->create_load(aAlloca);
    auto mul = builder->create_imul(aLoad,CONST_INT(2));
    builder->create_ret(mul);
    //finish bb1
    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                    "main", module);
    //bb2
    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 call = builder->create_call(calleeFun,{CONST_INT(110)});
    builder->create_ret(call);

    std::cout << module->print();
    delete module;
    return 0;
}
```

一共2个basic block，分别是callee函数的函数体和main的函数体

```
define dso_local i32 @callee(i32 %0)#0 {
    %2 = alloca i32, align 4
    store i32 %0, i32* %2,align 4
    %3 = load i32, i32* %2,align 4
    %4 = mul i32 %3 ,2
    ret i32 %4
}
define dso_local i32 @main()#0{
    %1 = alloca i32,align 4
    store i32 110, i32* %1,align 4
    %2 = load i32, i32* %1,align 4
    %3 = call i32 @callee(i32 %2)
    ret i32 %3
}
```

3. if

```
int main()
{
    auto module = new Module("Cminus code");  // module name是什么无关紧要
    auto builder = new IRBuilder(nullptr, module);
    Type *Int32Type = Type::get_int32_type(module);
    Type *FloatTy = Type::get_float_type(module);

    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                    "main", module);
    //bb1
    auto bb = BasicBlock::create(module, "entry", mainFun);
    builder->set_insert_point(bb);
    auto iAlloc = builder->create_alloca(FloatTy);
    auto retAlloc = builder->create_alloca(Int32Type);
    builder->create_store(CONST_FP(5.555),iAlloc);
    auto iLoad = builder->create_load(iAlloc);
    auto fcmp = builder->create_fcmp_gt(iLoad,CONST_FP(1.00));
    //bb1 end
    //3 bb
    auto trueBB = BasicBlock::create(module,"truebb",mainFun);
    auto falseBB = BasicBlock::create(module,"falsebb",mainFun);
    auto retBB = BasicBlock::create(module, "", mainFun);

    auto br = builder->create_cond_br(fcmp,trueBB,falseBB);
    //here true bb
    builder->set_insert_point(trueBB);  // if true; s分支的开始需要
SetInsertPoint设置
    builder->create_store(CONST_INT(233),retAlloc);
    builder->create_br(retBB);
    //true bb end
```

```
    //here false bb
    builder->set_insert_point(falseBB);
    builder->create_store(CONST_INT(0),retAlloc);
    builder->create_br(retBB);
    //false bb end
    //here ret bb
    builder->set_insert_point(retBB);
    auto retNum = builder->create_load(retAlloc);
    builder->create_ret(retNum);
    //ret bb end
    std::cout << module->print();
    delete module;
    return 0;

}
```

```
define dso_local i32 @main()#0{
    %1 = alloca float,align 4
    store float 0x40163851E0000000,float* %1,align 4
    %2 = load float, float* %1,align 4
    %3 = fcmp ogt float %2, 1.000000e+00
    %4 = alloca i32,align 4
    br i1 %3, label %5, label %7
5:
    store i32 233, i32* %4,align 4
    br label %7
6:
    store i32 0, i32* %4,align 4
    br label %7
7:
    %8 = load i32, i32* %4,align 4
    ret i32 %8
}
```

一共4个bb，主函数bb,正确分支bb(5)，否定分支bb(6)，返回值bb(7)

4. while

```
int main()
{
    auto module = new Module("Cminus code");  // module name是什么无关紧要
    auto builder = new IRBuilder(nullptr, module);
    Type *Int32Type = Type::get_int32_type(module);

    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                    "main", module);
    //main bb
    auto bb = BasicBlock::create(module, "entry", mainFun);
    builder->set_insert_point(bb);

    auto aAlloc = builder->create_alloca(Int32Type);
    auto iAlloc = builder->create_alloca(Int32Type);
    builder->create_store(CONST_INT(10),aAlloc);
    builder->create_store(CONST_INT(0),iAlloc);
    auto detBB = BasicBlock::create(module, "detBB", mainFun);//detect
    auto loopBB = BasicBlock::create(module, "loopBB", mainFun);//loop
    auto retBB = BasicBlock::create(module, "retBB", mainFun);//ret
    //det bb start
    auto br = builder->create_br(detBB);
```

```
    builder->set_insert_point(detBB);
    auto iLoad = builder->create_load(iAlloc);
    auto icmp = builder->create_icmp_lt(iLoad,CONST_INT(10));
    builder->create_cond_br(icmp,loopBB,retBB);
    //det bb end
    //loop bb start
    builder->set_insert_point(loopBB);
    auto iLoad_1 = builder->create_load(iAlloc);
    auto aLoad = builder->create_load(aAlloc);
    auto iadd = builder->create_iadd(iLoad_1,CONST_INT(1));
    auto aAddi = builder->create_iadd(aLoad,iadd);
    builder->create_store(iadd,iAlloc);
    builder->create_store(aAddi,aAlloc);
    builder->create_br(detBB);
    //loop bb end
    //ret bb start
    builder->set_insert_point(retBB);
    auto retNum = builder->create_load(aAlloc);
    builder->create_ret(retNum);
    //ret bb end
    std::cout << module->print();
    delete module;
    return 0;

}
```

```
define dso_local i32 @main() #0 {
    %1 = alloca i32,align 4 ;a addr
    %2 = alloca i32,align 4 ;i addr
    store i32 10, i32* %1,align 4
    store i32 0, i32* %2,align 4
    %3 = load i32, i32* %1,align 4 ;a
    br label %4
4:
    %5 = load i32, i32* %2,align 4 ;i
    %6 = icmp slt i32 %5, 10
    br i1 %6, label %7, label %12
7:
    %8 = load i32, i32* %1,align 4 ;a
    %9 = load i32, i32* %2,align 4 ;i
    %10 = add i32 %9, 1 ;i+1
    %11 = add i32 %8, %10 ;a+i
    store i32 %11, i32* %1,align 4
    store i32 %10, i32* %2,align 4
    br label %4
12:
    %13 = load i32, i32* %1,align 4 ;i
    ret i32 %13
}
```

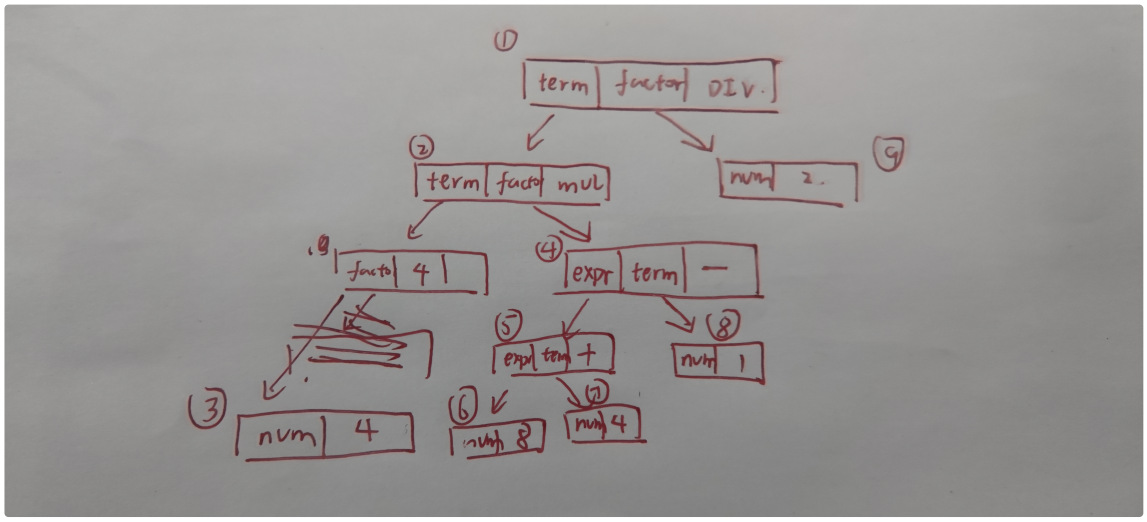
一共4个bb，主函数bb，判断bb(4)，循环bb(7)，跳出并返回bb(12)

问题3: Visitor Pattern

分析 `calc` 程序在输入为 `4 * (8 + 4 - 1) / 2` 时的行为：

1. 请画出该表达式对应的抽象语法树（使用 `calc_ast.hpp` 中的 `CalcAST*` 类型和在该类型中存储的值来表示），并给节点使用数字编号。

   2,3之间factor可省略

2. 请指出示例代码在用访问者模式遍历该语法树时的遍历顺序。

   做一个先序遍历

   1->2->3...->9(结点类型已经在图中表示，且 [2]和[3]之间的那个节点应该是没有的)

序列请按如下格式指明（序号为问题 2.1 中的编号）：
3->2->5->1

实验难点

ast build中的一部分代码没看懂(对c++特性不太了解，不知道怎么看一个比较复杂的代码。。。)
ll文件编写的时候不知道怎么调试

实验反馈

吐槽?建议?