lab2 实验报告

PB20111694 毛陈诚·

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

  第一个：将指针%1（指针类型为指向包含10个整型元素数组），做寻址偏移0个单位，然后将%1所指向数组（看做一个结构体）中第%0个元素的地址赋给%2

  第二个：将指针类型为整型的指针%1，做寻址偏移%0个单位，然后将其所指向元素的地址赋给%2

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。

1. assign.c

   总共一个代码块label_entry

▶ assign.cpp

```cpp
#include "BasicBlock.h"
#include "Constant.h"
#include "Function.h"
#include "IRBuilder.h"
#include "Module.h"
#include "Type.h"

#include <iostream>
#include <memory>
#define DEBUG
#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("Cminus code");
  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); // 创建并返回函数，参数依次是待创建函数类型 ty，函数名字 name (不可为空)，函数所属的模块 parent

  auto bb = BasicBlock::create(module, "entry", mainFun);

  builder->set_insert_point(bb);

  auto *arraytype = ArrayType::get(Int32Type, 10);
  //auto *arraytype = Type::get_array_type(Int32Type,10);
  auto a_alloca = builder->create_alloca(arraytype);//创建int a[10]


  auto x0GEP = builder->create_gep(a_alloca, {CONST_INT(0), CONST_INT(0)});

  builder->create_store(CONST_INT(10), x0GEP);//  a[0] = 10

  auto x0load =builder->create_load(x0GEP);//取出x0的值
  auto mul = builder->create_imul(CONST_INT(2), x0load);  //   a[0] * 2
  auto x1GEP = builder->create_gep(a_alloca, {CONST_INT(0), CONST_INT(1)});//获得x1地址
  builder->create_store(mul, x1GEP);
  auto x01load =builder->create_load(x1GEP);

  builder->create_ret(x01load);
  std::cout << module->print();
  delete module;
  return 0;
}
```

▶ assign.ll

```
label_entry:
  %op0 = alloca [10 x i32]
  %op1 = getelementptr [10 x i32], [10 x i32]* %op0, i32 0, i32 0
  store i32 10, i32* %op1
  %op2 = load i32, i32* %op1
  %op3 = mul i32 2, %op2
  %op4 = getelementptr [10 x i32], [10 x i32]* %op0, i32 0, i32 1
  store i32 %op3, i32* %op4
  %op5 = load i32, i32* %op4
  ret i32 %op5
```

2. fun.c

一共有两个代码块，分别对应caleefun函数与mian函数

▶ fun.cpp中calleefun函数

```
  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);

  // callee函数
```

```
  // 函数参数类型的vector
  std::vector<Type *> Ints(1, Int32Type);
  //通过返回值类型与参数类型列表得到函数类型
  auto calleeFunTy = FunctionType::get(Int32Type, Ints);
  // 由函数类型得到函数
  auto calleeFun = Function::create(calleeFunTy,
                                    "callee", module);

  auto bb = BasicBlock::create(module, "entry", calleeFun);
  builder->set_insert_point(bb);

  auto retAlloca = 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u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
  }

  builder->create_store(args[0], aAlloca);  // 将参数a store下来

  auto aLoad = builder->create_load(aAlloca);
  auto mul = builder->create_imul(CONST_INT(2), aLoad);  //   a * 2
  builder->create_store(mul, aAlloca);
  auto a1Load = builder->create_load(aAlloca);
  builder->create_ret(a1Load);
```

▶ fun.ll中calleefun函数

```
 define i32 @callee(i32 %arg0) {
callee:
  %op1 = alloca i32
  %op2 = alloca i32
  store i32 %arg0, i32* %op2
  %op3 = load i32, i32* %op2
  %op4 = mul i32 2, %op3
  store i32 %op4, i32* %op2
  %op5 = load i32, i32* %op2
  ret i32 %op5
}
```

▶ fun.cpp中main函数

```
  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);
```

```
  // callee函数
  // 函数参数类型的vector
  std::vector<Type *> Ints(1, Int32Type);
  //通过返回值类型与参数类型列表得到函数类型
  auto calleeFunTy = FunctionType::get(Int32Type, Ints);
  // 由函数类型得到函数
  auto calleeFun = Function::create(calleeFunTy,
                                    "callee", module);

  auto bb = BasicBlock::create(module, "entry", calleeFun);
  builder->set_insert_point(bb);

  auto retAlloca = 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u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
  }

  builder->create_store(args[0], aAlloca);  // 将参数a store下来

  auto aLoad = builder->create_load(aAlloca);
  auto mul = builder->create_imul(CONST_INT(2), aLoad);  //   a * 2
  builder->create_store(mul, aAlloca);
  auto a1Load = builder->create_load(aAlloca);
  builder->create_ret(a1Load);
```

▶ fun.ll中main函数

```
define i32 @main() {
label_entry:
  %op0 = call i32 @callee(i32 110)
  ret i32 %op0
}

```

3. if.c

一共有三个代码块label_entry,label_trueBB,label_falseBB

▶ if.cpp中label_entry函数

```
 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 *floatType = Type::get_float_type(module);
```

```
7
8    // main函数
9    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
10                                    "main", module); // 创建并返回函数，参数依次是待创建函数类型 ty，函数名字 name (不可为空)，函数所属的模块 parent
11
12   auto bb = BasicBlock::create(module, "entry", mainFun);
13
14   // BasicBlock的名字在生成中无所谓,但是可以方便阅读
15   builder->set_insert_point(bb);
16
17
18   auto a_alloca = builder->create_alloca(floatType);//创建float a
19
20
21   auto x0GEP = builder->create_gep(a_alloca, {CONST_INT(0)});
22   builder->create_store(CONST_FP(5.555), x0GEP);//  a = 5.555
23
24   auto x0Load =builder->create_load(x0GEP);//取出a的值
25   auto fcmp = builder->create_fcmp_gt(x0Load, CONST_FP(1.0));
```

▶ if.ll中label_entry块

```
1  label_entry:
2    %op0 = alloca float
3    %op1 = getelementptr float, float* %op0, i32 0
4    store float 0x40163851e0000000, float* %op1
5    %op2 = load float, float* %op1
6    %op3 = fcmp ugt float %op2,0x3ff0000000000000
7    br i1 %op3, label %label_trueBB, label %label_falseBB
```

▶ if.cpp中true_BB块

```
1    auto trueBB = BasicBlock::create(module, "trueBB", mainFun);    // true分支
2    auto br = builder->create_cond_br(fcmp, trueBB, falseBB);  // 条件BR
3    builder->set_insert_point(trueBB);  // if true; 分支的开始需要SetInsertPoint设置
4    builder->create_ret({CONST_INT(233)});
```

▶ if.ll中true_BB块

```
1  label_trueBB:                                     ; preds = %label_entry
2    ret i32 233
```

▶ if.cpp中false_BB块

```
  auto falseBB = BasicBlock::create(module, "falseBB", mainFun);  // false分支
  auto br = builder->create_cond_br(fcmp, trueBB, falseBB);  // 条件BR
  builder->set_insert_point(falseBB);  // if false
  builder->create_ret({CONST_INT(0)});
```

▶ if.ll中false_BB块

```
label_falseBB:                                    ; preds =%label_entry
  ret i32 0
```

4. while.c

一共有四个代码块label_entry，label_haha，label_trueBB，label_falseBB

▶ while.cpp中块entry

```
auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                "main", module); // 创建并返回函数，参数依次是待创建函数类型 ty，函数名字 name (不可为空)，函数所属的模块 parent

  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 a_alloca = builder->create_alloca(Int32Type);//创建int a
  auto i_alloca = builder->create_alloca(Int32Type);//创建int i


  auto a0GEP = builder->create_gep(a_alloca, {CONST_INT(0)});
  builder->create_store(CONST_INT(10), a0GEP);//  a = 10

  auto i0GEP = builder->create_gep(i_alloca, {CONST_INT(0)});
  builder->create_store(CONST_INT(0), i0GEP);//  i = 0
```

▶ while.ll中块entry

```
define i32 @main() {
label_entry:
  %op0 = alloca i32
  %op1 = alloca i32
  %op2 = getelementptr i32, i32* %op0, i32 0
  store i32 10, i32* %op2
  %op3 = getelementptr i32, i32* %op1, i32 0
  store i32 0, i32* %op3
  br label %label_haha
```

▶ while.cpp中块haha

```
  auto retBB = BasicBlock::create(
      module, "haha", mainFun);  // ,提前create,以便true分支可以br
  builder->create_br(retBB);  // br haha
  builder->set_insert_point(retBB);

  auto i0Load =builder->create_load(i0GEP);//取出i的值
  auto icmp = builder->create_icmp_lt(i0Load, CONST_INT(10));  // i和10的比较,注意ICMPEQ
```

▶ while.ll中块haha

```
label_haha:                                       ; preds = %label_entry, %label_trueBB
  %op4 = load i32, i32* %op3
  %op5 = icmp slt i32 %op4, 10
  br i1 %op5, label %label_trueBB, label %label_falseBB
```

▶ while.cpp中true_BB块

```
  auto trueBB = BasicBlock::create(module, "trueBB", mainFun);    // true分支
  auto br = builder->create_cond_br(icmp, trueBB, falseBB);  // 条件BR
  builder->set_insert_point(trueBB);  // if true; 分支的开始需要SetInsertPoint设置
  auto iLoad = builder->create_load(i_alloca);
  auto add = builder->create_iadd(CONST_INT(1), iLoad);  //   i++
  builder->create_store(add, i_alloca);
  auto i1Load = builder->create_load(i_alloca);
  auto a1Load = builder->create_load(a_alloca);
  auto add1 = builder->create_iadd(i1Load, a1Load);  //   a = a + i
  builder->create_store(add1, a_alloca);
  builder->create_br(retBB);  // br retBB
```

▶ while.ll中块true_BB

```
label_trueBB:                                     ; preds = %label_haha
  %op6 = load i32, i32* %op1
  %op7 = add i32 1, %op6
  store i32 %op7, i32* %op1
  %op8 = load i32, i32* %op1
  %op9 = load i32, i32* %op0
  %op10 = add i32 %op8, %op9
  store i32 %op10, i32* %op0
  br label %label_haha
```

▶ while.cpp中false_BB块

```
  auto falseBB = BasicBlock::create(module, "falseBB", mainFun);  // false分支
  auto br = builder->create_cond_br(icmp, trueBB, falseBB);  // 条件BR
  builder->set_insert_point(falseBB);  // if false
  auto a0Load = builder->create_load(a0GEP);//取出a的值
  builder->create_ret(a0Load);
```

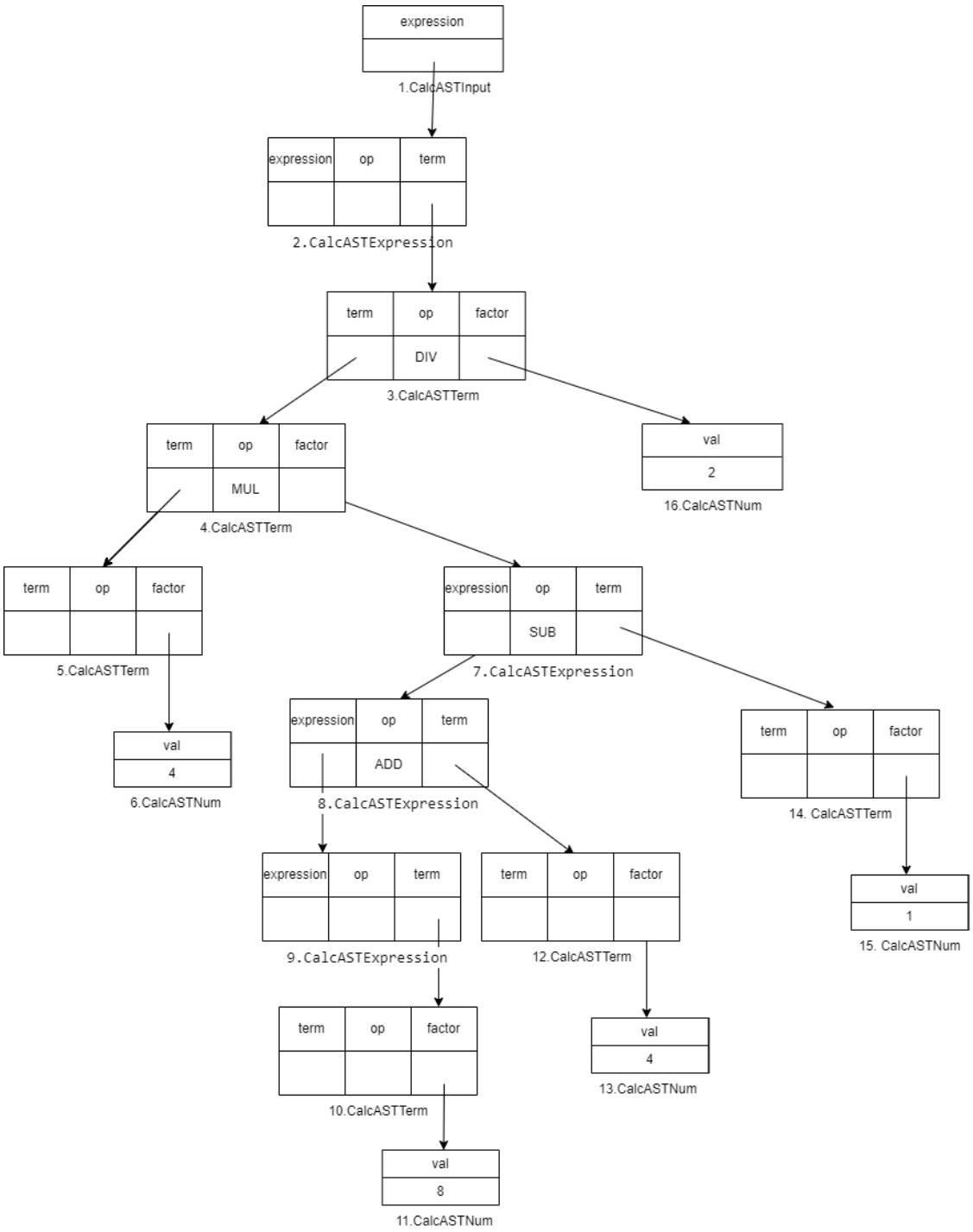
▶ while.ll中块false_BB

```
label_falseBB:                                    ; preds = %label_haha
  %op11 = load i32, i32* %op2
  ret i32 %op11
}
```

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. 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。

   类似于后序遍历，生成后序表达式

   1.expression→2.term→3.term→4.term→5.factor→6.val→4.factor→7.expression→8.expression→9.term→10.factor→11.val→8.term→12.factor→13.val→8.ADDop→7.term→14.factor→15.val→7.opSUB→4.opMUL→3.factor→16.val→3.opDIV

序列请按如下格式指明（序号为问题 3.1 中的编号）：
3->2->5->1

实验难点

描述在实验中遇到的问题、分析和解决方案。

1. c++一些高级语法看不太懂，比如智能指针，一些强制类型转换函数static_cast等

   查了一些资料，有些语法貌似看不懂也不影响写代码（因为有示例代码照葫芦画瓢就行了

2. 访问者模式比较难懂

   似懂非懂，不知道语法树写得对不对。

实验反馈

吐槽?建议?

访问者模式令人费解。。。

getelementptr的英文文档看麻了。。。