lab2 实验报告

PB20111635　蒋磊

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`

区别在于:

第一个返回的指针类型是[10 x i32]，假设第三个参数是n，第四个参数是m，返回的指针类型是[10 x i32]，但是要前移10n+m个单位

第二个返回的指针类型是i32，偏移%0

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。

assign

```
1  define dso_local i32 @main() #0{
2  entry:
3    %0 = alloca i32
4    store i32 0, i32* %0
5    %1 = alloca [10 x i32]
6    %2 = getelementptr [10 x i32], [10 x i32]* %1, i32 0, i32 0
```

```
  store i32 10, i32* %2
  %3 = getelementptr [10 x i32], [10 x i32]* %1, i32 0, i32 1
  %4 = load i32, i32* %2
  %5 = mul i32 %4, 2
  store i32 %5, i32* %3
  store i32 %5, i32* %0
  br label %6
6:
  %7 = load i32, i32* %0
  ret i32 %7
}
```

```
#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");  // module name是什么无关紧要
    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);  // 一个bb的开始,将当前插入指令点的位置设在bb

    auto retAlloca = builder->create_alloca(Int32Type); // 在内存中分配返回值的位置
    builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0

```

```
21      auto *arrayType = ArrayType::get(Int32Type, 10);
22      auto aAlloca = builder->create_alloca(arrayType);   // 在内存中分配
    a[10]的位置
23
24      auto a0GEP = builder->create_gep(aAlloca, {CONST_INT(0),
    CONST_INT(0)});  // GEP: 这里为什么是{0, 0}呢? (实验报告相关)
25      builder->create_store(CONST_INT(10), a0GEP);
26
27      auto a1GEP = builder->create_gep(aAlloca, {CONST_INT(0),
    CONST_INT(1)});
28
29      auto a0Load = builder->create_load(a0GEP);
30      auto mul = builder->create_imul(a0Load, CONST_INT(2)); // a[0] * 2
31      builder->create_store(mul, a1GEP);
32
33      auto retBB = BasicBlock::create(module, "", mainFun);   // return分
    支
34      builder->create_store(mul, retAlloca);
35      builder->create_br(retBB);  //br retBB
36
37      builder->set_insert_point(retBB);   // ret分支
38      auto retLoad = builder->create_load(retAlloca);
39      builder->create_ret(retLoad);
40
41      std::cout << module->print();
42      delete module;
43      return 0;
44  }
```

对应关系：一共有两个BasicBlock：

1. `auto bb = BasicBlock::create(module, "entry", mainFun);` 对应标记 `entry`
2. `autoretBB = BasicBlock::create(module, "", mainFun);` 对应标记 `6`

fun

```
define i32 @callee(i32 %0) {
entry:
  %1 = alloca i32
  store i32 0, i32* %1
  %2 = alloca i32
  store i32 %0, i32* %2
  %3 = load i32, i32* %2
  %4 = mul i32 %3, 2
  store i32 %4, i32* %1
  br label %5
5:
  %6 = load i32, i32* %1
  ret i32 %6
}
define i32 @main() {
entry:
  %0 = alloca i32
  store i32 0, i32* %0
  %1 = call i32 @callee(i32 110)
  ret i32 %1
}
```

```
#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");  // module name是什么无关紧要
    auto builder = new IRBuilder(nullptr, module);
    Type *Int32Type = Type::get_int32_type(module);

    // callee function
```

```
11      // 函数参数类型的vector
12      std::vector<Type *> Ints(1, Int32Type);
13
14      // 由函数类型得到函数
15      auto calleeFun = Function::create(FunctionType::get(Int32Type,
    Ints), "callee", module);
16
17      // BB的名字在生成中无所谓,但是可以方便阅读
18      auto bb = BasicBlock::create(module, "entry", calleeFun);
19
20      builder->set_insert_point(bb); // 一个BB的开始,将当前插入指令点的位置设在
    bb
21
22      auto retAlloca = builder->create_alloca(Int32Type); // 在内存中分配返
    回值的位置
23      builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
24
25      auto aAlloca = builder->create_alloca(Int32Type); // 在内存中分配参数a
    的位置
26
27      std::vector<Value *> args;  // 获取callee函数的形参,通过Function中的
    iterator
28      for(auto arg = calleeFun->arg_begin(); arg != calleeFun->arg_end();
    arg++){
29          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
30      }
31      builder->create_store(args[0], aAlloca); // 将参数a store下来
32      auto aLoad = builder->create_load(aAlloca); // 将参数a Load上去
33      auto mul = builder->create_imul(aLoad, CONST_INT(2));   // a * 2
34
35      auto retBB = BasicBlock::create(module, "", calleeFun); // return 分
    支
36      builder->create_store(mul, retAlloca);
37      builder->create_br(retBB);  // br retBB
38
```

```
39      builder->set_insert_point(retBB);   // ret分支
40      auto retLoad = builder->create_load(retAlloca);
41      builder->create_ret(retLoad);
42
43      // main函数
44      auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
    "main", module);
45
46      // BB的名字在生成中无所谓,但是可以方便阅读
47      bb = BasicBlock::create(module, "entry", mainFun);
48      builder->set_insert_point(bb); // 一个BB的开始,将当前插入指令点的位置设在
    bb
49
50      retAlloca = builder->create_alloca(Int32Type);  // 在内存中分配返回值的
    位置
51      builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
52
53      auto call = builder->create_call(calleeFun, {CONST_INT(110)});
54      builder->create_ret(call);
55
56      std::cout << module->print();
57      delete module;
58      return 0;
59  }
```

对应关系：一共有三个BasicBlock：

1. `auto bb = BasicBlock::create(module, "entry", calleeFun);` 对应 `callee` 函数中的标记 `entry`
2. `auto retBB = BasicBlock::create(module, "", calleeFun);` 对应 `callee` 函数中的标记 `5`
3. `bb = BasicBlock::create(module, "entry", mainFun);` 对应 `main` 中的标记 `entry`

if

```
define i32 @main() {
entry:
  %0 = alloca i32
  store i32 0, i32* %0
  %1 = alloca float
  store float 0x40163851e0000000, float* %1
  %2 = load float, float* %1
  %3 = fcmp ugt float %2,0x3ff0000000000000
  br i1 %3, label %trueBB, label %falseBB
trueBB:
  store i32 233, i32* %0
  br label %4
falseBB:
  store i32 0, i32* %0
  br label %4
4:
  %5 = load i32, i32* %0
  ret i32 %5
}
```

```
#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");  // 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);

    // main函数
```

```
12      auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
    "main", module);
13
14      // BB的名字在生成中无所谓,但是可以方便阅读
15      auto bb = BasicBlock::create(module, "entry", mainFun);
16      builder->set_insert_point(bb); // 一个BB的开始,将当前插入指令点的位置设在
    bb
17
18      auto retAlloca = builder->create_alloca(Int32Type);  // 在内存中分配返
    回值的位置
19      builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
20
21      auto aAlloca = builder->create_alloca(FloatType);   // 在内存中分配参
    数a的位置
22      builder->create_store(CONST_FP(5.555), aAlloca);
23
24      auto aLoad = builder->create_load(aAlloca); // 将参数a load上来
25      auto fcmp = builder->create_fcmp_gt(aLoad, CONST_FP(1));   // 将a与1
    比较
26
27      auto trueBB = BasicBlock::create(module, "trueBB", mainFun);    //
    true分支
28      auto falseBB = BasicBlock::create(module, "falseBB", mainFun);  //
    false分支
29      auto retBB = BasicBlock::create(module, "", mainFun);   // return分
    支
30
31      auto br = builder->create_cond_br(fcmp, trueBB, falseBB);   // 条件
    BR
32      DEBUG_OUTPUT
33
34      builder->set_insert_point(trueBB);  // if true; 分支的开始需要
    SetInsertPoint设置
35      builder->create_store(CONST_INT(233), retAlloca);
36      builder->create_br(retBB);  // br retBB
37
```

```
    builder->set_insert_point(falseBB); //if false;
    builder->create_store(CONST_INT(0), retAlloca);
    builder->create_br(retBB);  //br retBB

    builder->set_insert_point(retBB);
    auto retLoad = builder->create_load(retAlloca);
    builder->create_ret(retLoad);

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

对应关系：一共有四个BasicBlock：

1. `auto bb = BasicBlock::create(module, "entry", mainFun);` 对应标记 `entry`
2. `auto trueBB = BasicBlock::create(module, "trueBB", mainFun);` 对应标记 `trueBB`
3. `auto falseBB = BasicBlock::create(module, "falseBB", mainFun);` 对应标记 `falseBB`
4. `auto retBB = BasicBlock::create(module, "", mainFun);` 对应标记 `4`

while

```
define i32 @main() {
entry:
  %0 = alloca i32
  store i32 0, i32* %0
  %1 = alloca i32
  store i32 10, i32* %1
  %2 = alloca i32
  store i32 0, i32* %2
  %3 = load i32, i32* %1
  %4 = load i32, i32* %2
  br label %loop
loop:
```

```
  %5 = load i32, i32* %2
  %6 = icmp slt i32 %5, 10
  br i1 %6, label %trueBB, label %falseBB
trueBB:
  %7 = load i32, i32* %2
  %8 = add i32 %7, 1
  store i32 %8, i32* %2
  %9 = load i32, i32* %1
  %10 = load i32, i32* %2
  %11 = add i32 %10, %9
  store i32 %11, i32* %1
  br label %loop
falseBB:
  %12 = load i32, i32* %1
  store i32 %12, i32* %0
  br label %13
13:
  %14 = load i32, i32* %0
  ret i32 %14
}
```

```
#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");  // 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);

    // main函数
    auto mainFun = Function::create(FunctionType::get(Int32Type, {}), "main", module);

```

```
14      // BB的名字在生成中无所谓,但是可以方便阅读
15      auto bb = BasicBlock::create(module, "entry", mainFun);
16      builder->set_insert_point(bb); // 一个BB的开始,将当前插入指令点的位置设在
    bb
17
18      auto retAlloca = builder->create_alloca(Int32Type);  // 在内存中分配返
    回值的位置
19      builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
20
21      auto aAlloca = builder->create_alloca(Int32Type);   // 在内存中分配a的
    位置
22      builder->create_store(CONST_INT(10), aAlloca);  // a = 10
23
24      auto iAlloca = builder->create_alloca(Int32Type);   // 在内存中分配i的
    位置
25      builder->create_store(CONST_INT(0), iAlloca);  // i = 0
26
27      auto aLoad = builder->create_load(aAlloca); // 将a load上来
28      auto iLoad = builder->create_load(iAlloca); // 将i load上来
29
30      auto loop = BasicBlock::create(module, "loop", mainFun);
31      builder->create_br(loop);   // br loop
32      builder->set_insert_point(loop); // the entry for loop
33
34      iLoad = builder->create_load(iAlloca);  // 将参数i load上来
35      auto icmp = builder->create_icmp_lt(iLoad, CONST_INT(10));  // 将i与
    10比较
36
37      auto trueBB = BasicBlock::create(module, "tureBB", mainFun); //
    inside while
38      auto falseBB = BasicBlock::create(module, "falseBB", mainFun); //
    after while
39      auto retBB = BasicBlock::create(module, "", mainFun); // return 分支
40
41      auto br = builder->create_cond_br(icmp, trueBB, falseBB);   // 条件
    BR
```

```
42      DEBUG_OUTPUT
43
44      builder->set_insert_point(trueBB);  // if true; 分支的开始需要
    SetInsertPoint设置
45
46      iLoad = builder->create_load(iAlloca);  // 将参数i load上来
47      auto addi = builder->create_iadd(iLoad, CONST_INT(1));  // add
    result for i
48      builder->create_store(addi, iAlloca);
49
50      aLoad = builder->create_load(aAlloca);  // 将参数a load上来
51      iLoad = builder->create_load(iAlloca);  // 将参数i load上来
52      auto adda = builder->create_iadd(iLoad, aLoad); // a + i
53      builder->create_store(adda, aAlloca);
54      builder->create_br(loop);   // br loop
55      builder->set_insert_point(loop);    // the entry for loop
56
57      builder->set_insert_point(falseBB); // after while
58      aLoad = builder->create_load(aAlloca);  // 将参数a load上来
59      builder->create_store(aLoad, retAlloca);
60      builder->create_br(retBB);
61
62      builder->set_insert_point(retBB);   // ret 分支
63      auto retLoad = builder->create_load(retAlloca);
64      builder->create_ret(retLoad);
65
66      std::cout << module->print();
67      delete module;
68      return 0;
69  }
```

对应关系：一共有五个BasicBlock：

1. `auto bb = BasicBlock::create(module, "entry", mainFun);` 对应 entry
2. `auto loop = BasicBlock::create(module, "loop", mainFun);` 对应 loop
3. `auto trueBB = BasicBlock::create(module, "trueBB", mainFun);` 对应标记

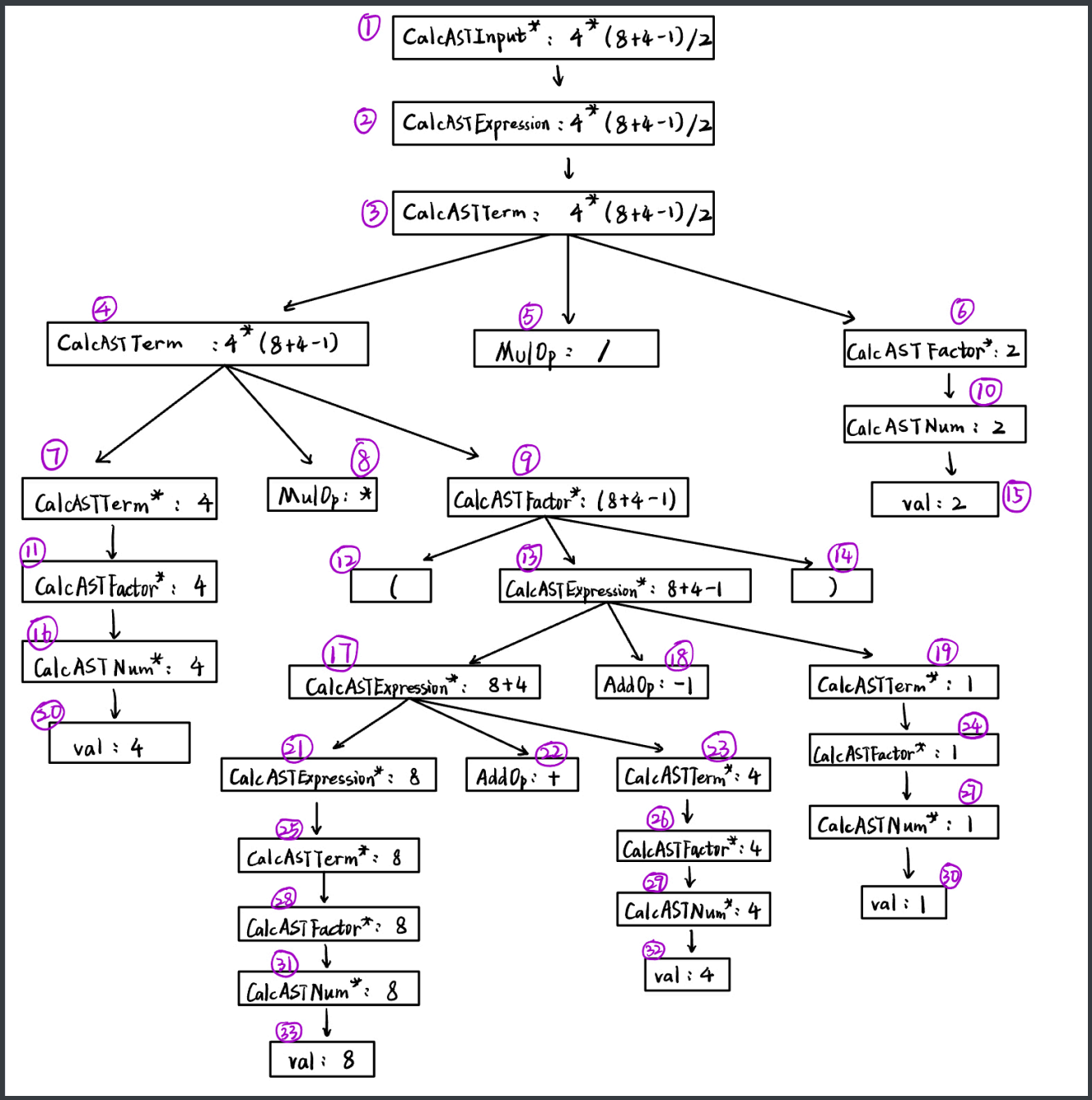
`trueBB`

4. `auto falseBB = BasicBlock::create(module, "falseBB", mainFun);` 对应标记 `falseBB`
5. `auto retBB = BasicBlock::create(module, "", mainFun);` 对应标记 `13`

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。
>
> 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

1. 如图



2. 从4 个 `visit` 方法中，可以看出是先序遍历左子树，所以遍历顺序为：

   1->2->3->4->7->11->16->20->8->9->12->13->17->21->25->28->31->33->22->23->26->29->32->18->19->24->27->30->14->5->6->10->15

实验难点

1. 编写.ll文件时非常容易写错，对于3操作数的语言特性感到不适应。
2. 对于C++掌握不熟，看助教写的部分代码感到生疏，花了很多时间进行理解。
3. 对于cpp文件的编写，即使是照葫芦画瓢，仍然会出现各种bug，部分接口函数不熟悉名字，常常需要查阅示例代码。
4. 实验报告中的问题3 比较复杂，语法树的作图有些繁琐。

实验反馈

助教老师补充的注释帮了大忙，希望下次实验可以保持本次实验文档的详细程度。