lab2 实验报告

PB20000137 李远航

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
```

第一种指针类型是 [10 x i32]*, 后面的0是指指向 [10 x i32] [0], 然后偏移量为 %0, 我认为可以理解成二维数组

第二种指针类型是 132*, 直接是一个 %0 的偏移

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 #include "BasicBlock.h"
2 #include "Constant.h"
3 #include "Function.h"
4 #include "IRBuilder.h"
5 #include "Module.h"
   #include "Type.h"
7
    #include <iostream>
9
    #include <memory>
10
   #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
12
    #else
14
   #define DEBUG_OUTPUT
    #endif
15
16
17
    #define CONST_INT(num) ConstantInt::get(num, module)
18
    #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()
21
23
        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);
```

```
27
          auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                            "main", module);
28
29
          auto bb = BasicBlock::create(module, "entry", mainFun);
30
          // BasicBlock的名字在生成中无所谓,但是可以方便阅读
         builder->set_insert_point(bb);
31
32
33
         auto retAlloca = builder->create_alloca(Int32Type);
         builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
34
         // int a[10];
35
36
         auto *arrayType = ArrayType::get(Int32Type, 10);
37
         auto aAlloca = builder->create_alloca(arrayType);
38
39
         // a[0] = 10;
         auto a0GEP = builder->create_gep(aAlloca, {CONST_INT(0), CONST_INT(0)});
40
         builder->create_store(CONST_INT(10), a0GEP);
41
42
43
         // a[1] = a[0] * 2
44
         auto a1GEP = builder->create_gep(aAlloca, {CONST_INT(0), CONST_INT(1)});
45
         a0GEP = builder->create_gep(aAlloca, {CONST_INT(0), CONST_INT(0)});
46
         auto a0Load = builder->create_load(a0GEP);
         auto mul = builder->create_imul(a0Load, CONST_INT(2)); // a[0] * 2
47
48
         builder->create_store(mul, a1GEP);
49
         // return a[1];
51
         builder->create_store(mul, retAlloca);
52
         auto retLoad = builder->create_load(retAlloca);
53
         builder->create_ret(retLoad);
54
55
         std::cout << module->print();
56
         delete module;
57
          return 0;
58
     define i32 @main() {
 2
     label_entry:
 3
       \%op0 = alloca i32
 4
       store i32 0, i32* %op0
 5
       %op1 = alloca [10 x i32]
       \text{%op2} = \text{getelementptr} [10 \times i32], [10 \times i32] * \text{\%op1}, i32 0, i32 0
 6
       store i32 10, i32* %op2
 7
       \text{%op3} = \text{getelementptr} [10 \times \text{i}32], [10 \times \text{i}32] * \text{%op1}, i32 0, i32 1
 8
 9
       %op4 = getelementptr [10 x i32], [10 x i32] * %op1, i32 0, i32 0
10
       %op5 = load i32, i32* %op4
11
       %op6 = mul i32 %op5, 2
       store i32 %op6, i32* %op3
12
       store i32 %op6, i32* %op0
13
       %op7 = load i32, i32* %op0
14
15
       ret i32 %op7
16
```

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

• fun

```
#include "BasicBlock.h"
#include "Constant.h"
```

```
3 #include "Function.h"
    #include "IRBuilder.h"
5
    #include "Module.h"
    #include "Type.h"
7
8
    #include <iostream>
9
    #include <memory>
10
    #ifdef DEBUG
11
                                                          // 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
12
13
    #else
    #define DEBUG_OUTPUT
14
15
    #endif
16
17
    #define CONST_INT(num) ConstantInt::get(num, module)
18
19
    #define CONST_FP(num) ConstantFP::get(num, module) // 得到常数值的表示,方便后面多
     次用到
20
21
    int main()
22
        auto module = new Module("Cminus code"); // module name是什么无关紧要
23
24
        auto builder = new IRBuilder(nullptr, module);
        Type *Int32Type = Type::get_int32_type(module);
26
27
        std::vector<Type *> Ints(1, Int32Type);
28
29
        //通过返回值类型与参数类型列表得到函数类型
30
        auto calleFunTy = FunctionType::get(Int32Type, Ints);
31
32
        // 由函数类型得到函数
33
        auto callee = Function::create(calleFunTy,
34
                                      "callee", module);
35
36
        // BB的名字在生成中无所谓,但是可以方便阅读
37
        auto bb = BasicBlock::create(module, "entry", callee);
38
39
        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
40
41
        auto retAlloca = builder->create_alloca(Int32Type); // 在内存中分配返回值的
    位置
42
        auto aAlloca = builder->create_alloca(Int32Type); // 在内存中分配参数a的位
     置.
43
44
        std::vector<Value *> args; // 获取gcd函数的形参,通过Function中的iterator
45
        for (auto arg = callee->arg_begin(); arg != callee->arg_end(); arg++)
46
        {
47
            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
48
49
        builder->create_store(args[0], aAlloca); // 将参数a store下来
51
        auto aLoad = builder->create_load(aAlloca);
52
        auto mul = builder->create_imul(aLoad, CONST_INT(2));
        builder->create_store(mul, retAlloca);
53
        auto retLoad = builder->create_load(retAlloca);
```

```
55
            builder->create_ret(retLoad);
   56
   57
            auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                           "main", module);
   58
   59
            // BasicBlock的名字在生成中无所谓,但是可以方便阅读
            bb = BasicBlock::create(module, "entry", mainFun);
   60
            builder->set_insert_point(bb);
   61
   62
   63
            retAlloca = builder->create_alloca(Int32Type);
   64
            builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
   65
            auto call = builder->create_call(callee, {CONST_INT(110)});
   66
            builder->create_ret(call);
   67
   68
   69
            std::cout << module->print();
   70
            delete module;
   71
            return 0;
   72
        define i32 @callee(i32 %arg0) {
    2
        label_entry:
    3
         %op1 = alloca i32
    4
         %op2 = alloca i32
         store i32 %arg0, i32* %op2
    5
    6
         %op3 = load i32, i32* %op2
    7
         %op4 = mul i32 %op3, 2
         store i32 %op4, i32* %op1
    8
         %op5 = load i32, i32* %op1
    9
   10
         ret i32 %op5
   11
       }
        define i32 @main() {
   12
   13
       label_entry:
   14
         %op0 = alloca i32
         store i32 0, i32* %op0
   15
         %op1 = call i32 @callee(i32 110)
         ret i32 %op1
   17
   18
    • auto bb = BasicBlock::create(module, "entry", callee); 对应 calle 中的
       label_entry
    • bb = BasicBlock::create(module, "entry", mainFun); 对应 main中的
       label_entry
• if
       #include "BasicBlock.h"
    2 #include "Constant.h"
        #include "Function.h"
    4 #include "IRBuilder.h"
        #include "Module.h"
        #include "Type.h"
    6
```

```
7
8
     #include <iostream>
9
     #include <memory>
10
```

```
11 #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
12
13
     #else
     #define DEBUG_OUTPUT
14
     #endif
15
16
17
     #define CONST_INT(num) ConstantInt::get(num, module)
18
19
     #define CONST_FP(num) ConstantFP::get(num, module) // 得到常数值的表示,方便后面多
     次用到
20
21
     int main()
22
23
         auto module = new Module("Cminus code"); // module name是什么无关紧要
24
         auto builder = new IRBuilder(nullptr, module);
25
         Type *Int32Type = Type::get_int32_type(module);
26
27
         auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
28
                                        "main", module);
29
         auto bb = BasicBlock::create(module, "entry", mainFun);
30
         // BasicBlock的名字在生成中无所谓,但是可以方便阅读
31
         builder->set_insert_point(bb);
32
         auto retAlloca = builder->create_alloca(Int32Type);
34
         builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
35
         Type *FloatType = Type::get_float_type(module);
36
37
         auto aAlloca = builder->create_alloca(FloatType); // a
38
         builder->create_store(CONST_FP(5.555), aAlloca);
39
40
         auto aLoad = builder->create_load(aAlloca);
                                                                 // load a
41
         auto fcmp = builder->create_fcmp_gt(aLoad, CONST_FP(1)); // if
42
         auto trueBB = BasicBlock::create(module, "trueBB", mainFun); // true分
43
     支
         auto falseBB = BasicBlock::create(module, "falseBB", mainFun); // false分
44
     支
         auto retBB = BasicBlock::create(module, "", mainFun);
45
                                                                      // return
     分支
46
         auto br = builder->create_cond_br(fcmp, trueBB, falseBB);
47
48
49
         builder->set_insert_point(trueBB); // if true; 分支的开始需要SetInsertPoint
     设置
50
         builder->create_store(CONST_INT(233), retAlloca);
51
         builder->create_br(retBB); // br retBB
52
53
         builder->set_insert_point(falseBB); // if false
54
         builder->create_store(CONST_INT(0), retAlloca);
55
         builder->create_br(retBB);
56
57
         builder->set_insert_point(retBB); // ret分支
58
         auto retLoad = builder->create_load(retAlloca);
59
         builder->create_ret(retLoad);
```

```
std::cout << module->print();
   62
            delete module;
   63
           return 0;
   64
        define i32 @main() {
    2 label_entry:
         %op0 = alloca i32
    3
    4
        store i32 0, i32* %op0
         %op1 = alloca float
         store float 0x40163851e0000000, float* %op1
         %op2 = load float, float* %op1
         %op3 = fcmp ugt float %op2,0x3ff0000000000000
         br i1 %op3, label %label_trueBB, label %label_falseBB
    9
   10 label_trueBB:
                                                                ; preds =
       %label_entry
        store i32 233, i32* %op0
   11
        br label %label4
   12
   13 label_falseBB:
                                                                  ; preds =
       %label_entry
         store i32 0, i32* %op0
   14
        br label %label4
   15
   16 label4:
                                                           ; preds =
       %label_trueBB, %label_falseBB
         %op5 = load i32, i32* %op0
   17
   18
        ret i32 %op5
   19 }
    • auto bb = BasicBlock::create(module, "entry", mainFun); 对应 label_entry
    • auto trueBB = BasicBlock::create(module, "trueBB", mainFun); 对应
      label_trueBB
    • auto falseBB = BasicBlock::create(module, "falseBB", mainFun); 对应
      label_falseBB
    • auto retBB = BasicBlock::create(module, "", mainFun); 对应 label4
while
      #include "BasicBlock.h"
    2 #include "Constant.h"
       #include "Function.h"
    4 #include "IRBuilder.h"
       #include "Module.h"
       #include "Type.h"
    8
       #include <iostream>
    9
       #include <memory>
   10
       #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
   12
        #else
   13
   14
        #define DEBUG_OUTPUT
        #endif
   15
   16
```

#define CONST_INT(num) ConstantInt::get(num, module)

17

18

```
#define CONST_FP(num) ConstantFP::get(num, module) // 得到常数值的表示,方便后面多
     次用到
20
21
     int main()
22
         auto module = new Module("Cminus code"); // module name是什么无关紧要
23
24
         auto builder = new IRBuilder(nullptr, module);
25
         Type *Int32Type = Type::get_int32_type(module);
26
27
         auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
28
                                         "main", module);
         auto bb = BasicBlock::create(module, "entry", mainFun);
29
         // BasicBlock的名字在生成中无所谓,但是可以方便阅读
30
31
         builder->set_insert_point(bb);
32
33
         auto retAlloca = builder->create_alloca(Int32Type);
         builder->create_store(CONST_INT(0), retAlloca); // 默认 ret 0
34
35
36
         auto aAlloca = builder->create_alloca(Int32Type);
37
         builder->create_store(CONST_INT(10), aAlloca);
         auto iAlloca = builder->create_alloca(Int32Type);
38
39
         builder->create_store(CONST_INT(0), iAlloca);
40
         auto aLoad = builder->create_load(aAlloca);
         auto iLoad = builder->create_load(iAlloca);
41
42
43
         auto trueBB = BasicBlock::create(module, "trueBB", mainFun); // true分
         auto falseBB = BasicBlock::create(module, "falseBB", mainFun); // false分
44
     支
45
         auto icmp = builder->create_icmp_lt(iLoad, CONST_INT(10)); // while
46
47
         auto br = builder->create_cond_br(icmp, trueBB, falseBB);
48
49
         builder->set_insert_point(trueBB);
50
         // i = i + 1;
51
         iLoad = builder->create_load(iAlloca);
52
         auto addi = builder->create_iadd(iLoad, CONST_INT(1));
53
         builder->create_store(addi, iAlloca);
54
         // a = a + i;
55
         aLoad = builder->create_load(aAlloca);
56
         iLoad = builder->create_load(iAlloca);
         auto adda = builder->create_iadd(iLoad, aLoad);
57
58
         builder->create_store(adda, aAlloca);
59
         //
60
         icmp = builder->create_icmp_lt(iLoad, CONST_INT(10));
         br = builder->create_cond_br(icmp, trueBB, falseBB);
61
62
63
         builder->set_insert_point(falseBB);
64
         aLoad = builder->create_load(aAlloca);
65
         builder->create_store(aLoad, retAlloca);
         auto retLoad = builder->create_load(retAlloca);
66
67
         builder->create_ret(retLoad);
68
69
         std::cout << module->print();
70
         delete module;
71
         return 0;
```

```
72
     define i32 @main() {
    label_entry:
 3
     %op0 = alloca i32
      store i32 0, i32* %op0
     %op1 = alloca i32
 5
      store i32 10, i32* %op1
 6
 7
     %op2 = alloca i32
      store i32 0, i32* %op2
      %op3 = load i32, i32* %op1
 9
      %op4 = load i32, i32* %op2
10
      %op5 = icmp slt i32 %op4, 10
11
     br i1 %op5, label %label_trueBB, label %label_falseBB
12
   label_trueBB:
13
                                                                ; preds =
    %label_entry, %label_trueBB
     %op6 = load i32, i32* %op2
14
     %op7 = add i32 %op6, 1
15
16
     store i32 %op7, i32* %op2
     %op8 = load i32, i32* %op1
17
     %op9 = load i32, i32* %op2
18
     %op10 = add i32 %op9, %op8
19
20
     store i32 %op10, i32* %op1
      %op11 = icmp slt i32 %op9, 10
21
22
     br i1 %op11, label %label_trueBB, label %label_falseBB
23 label_falseBB:
                                                                 ; preds =
    %label_entry, %label_trueBB
24
     %op12 = load i32, i32* %op1
25
     store i32 %op12, i32* %op0
     %op13 = load i32, i32* %op0
26
     ret i32 %op13
27
28
   }
```

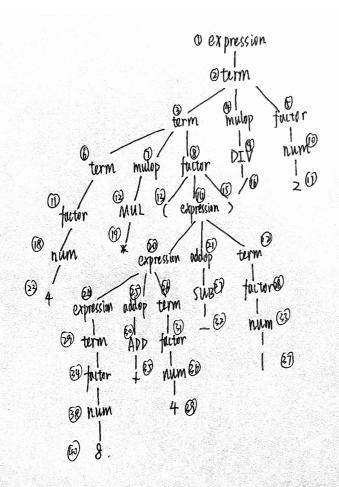
```
    auto bb = BasicBlock::create(module, "entry", mainFun); 对应 label_entry
    auto trueBB = BasicBlock::create(module, "trueBB", mainFun); 对应 label_trueBB
    auto falseBB = BasicBlock::create(module, "falseBB", mainFun); 对应 label_falseBB
```

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 3.1 中的编号): 3->2->5->1



 $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow [1 \rightarrow R18 \rightarrow 23 \rightarrow 8 \rightarrow 14 \rightarrow 20 \rightarrow 24 \rightarrow 29 \rightarrow 34 \rightarrow 38 \rightarrow 40 \rightarrow 26 \rightarrow 31 \rightarrow 36 \rightarrow 39 \rightarrow 30 \rightarrow 35 \rightarrow 22 \rightarrow 8 \rightarrow 33 \rightarrow 37 \rightarrow 21 \rightarrow 27 \rightarrow 7 \rightarrow 12 \rightarrow 19 \rightarrow 5 \rightarrow 10 \rightarrow 17 \rightarrow 4 \rightarrow 9 \rightarrow 16$

实验难点

描述在实验中遇到的问题、分析和解决方案。

- 实验文档中好像没有解释 nsw 的地方, 找了挺久
- generator 能够正常输出比较容易,但是判断输出的 .11 是否符合要求或者出错在哪里,比较困难,搞不清楚,然后换一种写法,就能正常完成实验
- ret 分支可以直接顺序执行的时候,不需要单独开辟一个 block
- generator 中有的参数有的是分配的内存,有的参数是值,需要仔细区分
- 插件的直接跳转很好用, 能快速看清函数结构

实验反馈

吐槽?建议?

- calc 好多各种各样的 visit
- 感觉题目问题描述的有点抽象,就按照自己理解写了