lab2 实验报告

PB23000114 刘阳

问题1: getelementptr

请结合 `IR.md` 中提到的两种 getelementptr 用法的区别,并举例加解释:

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

第一种写法是由一个指向长度为10的数组的个元素的第0个元素的指针,并还该数组由32组成,每个元素只有一个段,因此在计算第一个参数是应该是整个10 x i32。
第二个写法是直接1维数组的指针。

问题2: cpp 与 .ll 的对应

请说明您的 cpp 代码片段和 .ll 的每个 BasicBlock 的对应关系。

assign

```
#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一个命令来开启或一直使
#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) // 浮点数常量的表示,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);

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

    auto retAlloca = builder->create_alloca(Int32Type);
    builder->create_store(CONST_INT(0), retAlloca);

    auto arrayType = ArrayType::get(Int32Type,10);
    auto a0ptr = builder->create_alloca(arrayType);

    auto a0GEP = builder->create_gep(a, {CONST_INT(0), CONST_INT(0)});
    builder->create_store(CONST_INT(10),a0GEP);
    auto a1GEP = builder->create_gep(a, {CONST_INT(0), CONST_INT(1)});
    auto a2GEP = builder->create_gep(a, {CONST_INT(0), CONST_INT(2)});
    builder->create_store(mul,a2GEP);

//    builder->create_store(ans,retAlloca);

//    auto retLoad=builder->create_load(retAlloca);
    builder->create_ret(mul);


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

```
define i32 @main() {
label_entry:
  %op0 = alloca i32
  store i32 0, i32* %op0
  %op1 = alloca [10 x i32]
  %op2 = getelementptr [10 x i32], [10 x i32]* %op1, i32 0, i32 0
  store i32 10, i32* %op2
  %op3 = alloca i32*, i32 %op2
  %op4 = mul i32 %op2, 4
  %op5 = getelementptr [10 x i32], [10 x i32]* %op1, i32 0, i32 1
  store i32 %op4, i32* %op5
  ret i32 %op4
}
```

一个代码块,auto bb = BasicBlock::create(module, "entry", mainFun);对应label_entry

if

```
#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一个命令来开启或一直使
#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) // 浮点数常量的表示,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);

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

    auto retAlloca = builder->create_alloca(Int32Type);
    builder->create_store(CONST_INT(0), retAlloca);

    auto a0alloca=builder->create_alloca(Int32Type);
    auto b0alloca=builder->create_alloca(Int32Type);
    auto c0alloca=builder->create_alloca(Int32Type);
    auto i1alloca=builder->create_alloca(floatType);
    auto cmp0alloca=builder->create_alloca(Int32Type);
    auto cmp1alloca=builder->create_alloca(Int32Type);
    auto cmp2alloca=builder->create_alloca(Int32Type);
    auto cmp3alloca=builder->create_alloca(Int32Type);

    auto trueBB=BasicBlock::create(module,"true",mainFun);
    auto falseBB=BasicBlock::create(module,"false",mainFun);

    auto a0value=builder->create_load(a0);
    auto res=builder->create_fcmp_gt(a0value, CONST_FP(1));

    auto br=builder->create_cond_br(res,retTrue,retFalse);

    builder->set_insert_point(retTrue);
    builder->create_store(CONST_INT(2),retAlloca);
    auto ans1=builder->create_load(retAlloca);
    builder->create_ret(ans1);
    builder->set_insert_point(retFalse);
    builder->create_store(CONST_INT(0),retAlloca);
    auto ans2=builder->create_load(retAlloca);
    builder->create_ret(ans2);

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

```
define i32 @main() {
label_entry:
  %op0 = alloca i32
  store i32 0, i32* %op0
  %op1 = alloca float
  store float 0x4081111400000000, float* %op1
  %op2 = load float, float* %op1
  %op3 = fcmp ugt float %op2,0x7fd0000000000000
  br i1 %op3, label %label_true, label %label_false
label_true:                                       ; preds = %label_entry
  store i32 2, i32* %op0
  %op4 = load i32, i32* %op0
  ret i32 %op4
label_false:                                      ; preds = %label_entry
  store i32 0, i32* %op0
  %op5 = load i32, i32* %op0
  ret i32 %op5
}
```

三个代码块,auto bb = BasicBlock::create(module, "entry", mainFun);对应label_entry,auto retTrue =BasicBlock::create(module, "true",mainFun);对应label_true ,auto retFalse=BasicBlock::create(module, "false",mainFun);对应label_false.

while

```
#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一个命令来开启或一直使
#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) // 浮点数常量的表示,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);

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

    auto retAlloca = builder->create_alloca(Int32Type);
    builder->create_store(CONST_INT(0), retAlloca);

    auto a0alloca=builder->create_alloca(Int32Type);
    auto b0alloca=builder->create_alloca(Int32Type);
    auto c0alloca=builder->create_alloca(Int32Type);
    auto i1alloca=builder->create_alloca(Int32Type);
    auto cmp0alloca=builder->create_alloca(Int32Type);

    auto whileBB=BasicBlock::create(module,"while",mainFun);
    auto retBB=BasicBlock::create(module,"return",mainFun);
    auto i1ans=builder->create_load(i1);
    auto cmp=builder->create_icmp_lt(i1ans, CONST_INT(10));
    auto br=builder->create_cond_br(cmp,whileBB,retBB);

    builder->set_insert_point(whileBB);
    auto i0value=builder->create_load(i1);
    auto addValue=builder->create_iadd(i0value, CONST_INT(1));
    auto i1store=builder->create_store(addValue,i1);
    auto Avalueload=builder->create_load(a);
    auto addb=builder->create_iadd(addValue,AvalueL);
    auto bstore=builder->create_store(addb,b);
    auto judge=builder->create_icmp_lt(addValue, CONST_INT(10));
    auto whiletrueBr=builder->create_cond_br(judge,whileBB,retBB);

    builder->set_insert_point(retBB);
    auto ans=builder->create_load(b);
    builder->create_ret(ans);

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

```
define i32 @main() {
label_entry:
  %op0 = alloca i32
  store i32 0, i32* %op0
  %op1 = alloca i32
  %op2 = alloca i32
  store i32 10, i32* %op1
  store i32 0, i32* %op2
  %op3 = load i32, i32* %op1
  %op4 = load i32, i32* %op2
  %op5 = icmp slt i32 %op3, 10
  br i1 %op5, label %label_while, label %label_return
label_while:                                      ; preds = %label_entry, %label_while
  %op6 = load i32, i32* %op2
  %op7 = add i32 %op6, 1
  store i32 %op7, i32* %op2
  %op8 = load i32, i32* %op2
  %op9 = add i32 %op8, %op2
  store i32 %op9, i32* %op2
  %op10 = icmp slt i32 %op9, 10
  br i1 %op10, label %label_while, label %label_return
label_return:                                     ; preds = %label_entry, %label_while
  %op11 = load i32, i32* %op2
  ret i32 %op11
}
```

三个代码块,auto bb = BasicBlock::create(module, "entry", mainFun);对应label_entry,auto whileBB=BasicBlock::create(module,"while",mainFun);对应label_while,auto retBB=BasicBlock::create(module,"return",mainFun);对应label_return.

fun

```
#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一个命令来开启或一直使
#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) // 浮点数常量的表示,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);

    auto callerFunction=create(FunctionType::get(Int32Type,{Int32Type}),"caller",module);
    auto bb = BasicBlock::create(module, "entry", callerF);
    builder->set_insert_point(bb);                        // 一个特殊的开始,用法跟前面入参不动的位置语句相
    auto retAlloca = builder->create_alloca(Int32Type);   // 在内存中分配返回值的位置
    auto a0Loca = builder->create_alloca(Int32Type);
    auto a0store=builder->create_store(a0,a0Loca);
    for (auto arg :callerFunc->get_args()) {
        args.push_back(arg);
    }

    auto a0value=builder->create_load(arg[0], a0Loca);
    auto res=builder->create_iadd(a0value,{0}, CONST_INT(2));
    builder->create_store(retValue,retAlloca);
    auto ans=builder->create_load(retAlloca);
    builder->create_ret(ans);

    auto mainFun = Function::create(FunctionType::get(Int32Type, {}),
                                    "main", module);
    auto BB=BasicBlock::create(module,"start",mainFun);
    builder->set_insert_point(BB);
//  auto callerFunc=builder->create_call(callerFunc);
//  builder->create_store(CONST_INT(10),value);
    auto callbuilder=builder->create_call(callerF,{CONST_INT(10)});
    builder->create_ret(call);

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

```
define i32 @callee(i32 %arg0) {
label_entry:
  %op1 = alloca i32
  %op2 = alloca i32
  store i32 %arg0, i32* %op2
  %op3 = mul i32 %arg0, 2
  %op4 = add i32 %op3, 2
  %op5 = load i32, i32* %op2
  ret i32 %op5
}
define i32 @main() {
label_entry:
  %op6 = call i32 @callee(i32 110)
  ret i32 %op6
}
```

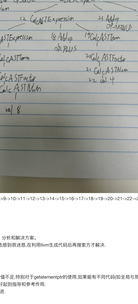
两个代码块,auto bb = BasicBlock::create(module, "entry", callee);对应label_entry,auto BB=BasicBlock::create(module,"start",mainFun);对应start.

问题3: Visitor Pattern

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

1. 请画出语言正式对应的抽象语法树(使用 `calc_ast.hpp` 中的 `CalcAstPtr` 类型和相应语言型中存储的值来表示),并标出产生这棵树的字编号。
2. 请画出示例代码应用访问者模式遍历该语法树时的遍历顺序。

序列表现形下格式指明 (序号为问题 2.1 中的编号):
3->2->5->1



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

实验难点

描述在实验中遇到的问题、分析和解决方案。
tiKzupx中的鼠标操作的写法感到很迷惑,在和Prof=ts生成代码后再搜索方式解决。

实验反馈

gcd的示例代码感觉参考价值不足,特别对于getelementptr的使用,如果能有不同代码指令应用场景的组合,基本类型与结构体数组之间的比可能能更好起到指导和参考作用。
代码模写为长量过于枯燥感。