Lab3补充提示

提醒

希望同学们仔细阅读实验文档`cminusf.md`和`LightIR.md`，在lab3编写的过程中也结合自己在lab2所做的工作。

如果在编写过程中不知道具体用哪些函数，可以去查看给出代码中的函数所在的头文件，里面包含部分其他需要用的函数。

部分函数的注释

```
//存储当前value
Value *tmp_val = nullptr;
// 当前函数
Function *cur_fun = nullptr;

// 表示是否在之前已经进入scope，用于CompoundStmt
// 进入CompoundStmt不仅包括通过Fundeclaration进入，也包括selection-stmt等。
// pre_enter_scope用于控制是否在CompoundStmt中添加scope.enter,scope.exit
bool pre_enter_scope = false;

// types
Type *VOID_T;
Type *INT1_T;
Type *INT32_T;
Type *INT32PTR_T;
Type *FLOAT_T;
Type *FLOATPTR_T;

/*
 * use CMinusfBuilder::Scope to construct scopes
 * scope.enter: enter a new scope
 * scope.exit: exit current scope
 * scope.push: add a new binding to current scope
 * scope.find: find and return the value bound to the name
 */
```

```cpp

void CminusfBuilder::visit(ASTProgram &node) {
    VOID_T = Type::get_void_type(module.get());
    INT1_T = Type::get_int1_type(module.get());
    INT32_T = Type::get_int32_type(module.get());
    INT32PTR_T = Type::get_int32_ptr_type(module.get());
    FLOAT_T = Type::get_float_type(module.get());
    FLOATPTR_T = Type::get_float_ptr_type(module.get());

    for (auto decl: node.declarations) { // program -> declaration-list
        decl->accept(*this);//进入下一层函数
    }
}


void CminusfBuilder::visit(ASTFunDeclaration &node) {
    FunctionType *fun_type;
    Type *ret_type;
    std::vector<Type *> param_types;
    if (node.type == TYPE_INT)//函数返回值类型
        ret_type = INT32_T;
    else if (node.type == TYPE_FLOAT)
        ret_type = FLOAT_T;
    else
        ret_type = VOID_T;

    for (auto& param: node.params) { //补全param_types
        //TODO:
        //根据param的类型分类
        //需要考虑int型数组，int型，float型数组，float型
        //对于不同的类型，向param_types中添加不同的Type
        //param_types.push_back


    }

    fun_type = FunctionType::get(ret_type, param_types);
    auto fun =
        Function::create(
                fun_type,
                node.id,
                module.get());//定义函数变量
    scope.push(node.id, fun);
    cur_fun = fun;
    auto funBB = BasicBlock::create(module.get(), "entry", fun);//创建基本块
    builder->set_insert_point(funBB);
    scope.enter();
    pre_enter_scope = true;
    std::vector<Value*> args;
    for (auto arg = fun->arg_begin();arg != fun->arg_end();arg++) {
        args.push_back(*arg);
    }
    for (int i = 0;i < node.params.size();++i) {
        //TODO:
        //需要考虑int型数组，int型，float型数组，float型
        //builder->create_alloca创建alloca语句
        //builder->create_store创建store语句
        //scope.push


    }
    node.compound_stmt->accept(*this);//fun-declaration -> type-specifier ID ( params ) compound-stmt
    if (builder->get_insert_block()->get_terminator() == nullptr){//创建ret语句
        if (cur_fun->get_return_type()->is_void_type())
            builder->create_void_ret();
        else if (cur_fun->get_return_type()->is_float_type())
            builder->create_ret(CONST_FP(0.));
        else
            builder->create_ret(CONST_INT(0));
    }
    scope.exit();
}

void CminusfBuilder::visit(ASTParam &node) { }

void CminusfBuilder::visit(ASTCompoundStmt &node) {
    //TODO: 此函数为完整实现
    bool need_exit_scope = !pre_enter_scope;//添加need_exit_scope变量
    if (pre_enter_scope) {
        pre_enter_scope = false;
    } else {
        scope.enter();
    }

    for (auto& decl: node.local_declarations) {//compound-stmt -> { local-declarations statement-list }
        decl->accept(*this);
    }

    for (auto& stmt: node.statement_list) {
        stmt->accept(*this);
        if (builder->get_insert_block()->get_terminator() != nullptr)
            break;
    }

    if (need_exit_scope) {
        scope.exit();
    }
}

void CminusfBuilder::visit(ASTReturnStmt &node) {//return-stmt -> return ; | return expression ;
    if (node.expression == nullptr) {
        builder->create_void_ret();
    } else {
        //TODO:
        //需要考虑类型转换
        //函数返回值和表达式值类型不同时，转换成函数返回值的类型
        //用cur_fun获取当前函数返回值类型
        //类型转换：builder->create_fptosi
        //ret语句

    }
}
```