编译原理研讨课实验PR002实验报告

一、任务说明

本次实验的任务为，为elementwise新建语法/语义规则，使其支持赋值，加法，乘法三种操作，并由此构建新的AST表示。

二、成员组成

李昊宸 2017K8009929044

李颖彦 2017K8009929025

陆润宇 2017K8009929027

三、实验设计

对赋值，加法，乘法三种操作进行逐一改动，使其支持含elementWise的AST的生成：

1. 提供对elementWise标注的函数内部对数组的“+”，“\*”，“=”操作
2. 提供对“int”数据类型的支持（而非integer）
3. 支持一维静态大小输入
4. 设计思路

重点在于在CreateBuiltinBinOP函数中对于以下三个函数的修改：

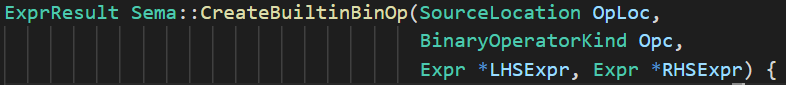
CheckAssignmentOperands，CheckMultiplyDivideOperands，CheckAdditionOperands

我们需要增加与elementWise相关的if分支来处理elementWise的相关情况，并且对一些特殊情况（例如是否存在合法的类型转换）进行处理。

1. 实验实现

首先找到入口函数

/home/clang7/llvm/tools/clang/lib/Sema/SemaExpr.cpp



这个函数负责创造AST的有关bulitin的算符的结点，而实验要求的=，\*，+均属于builtin算符。

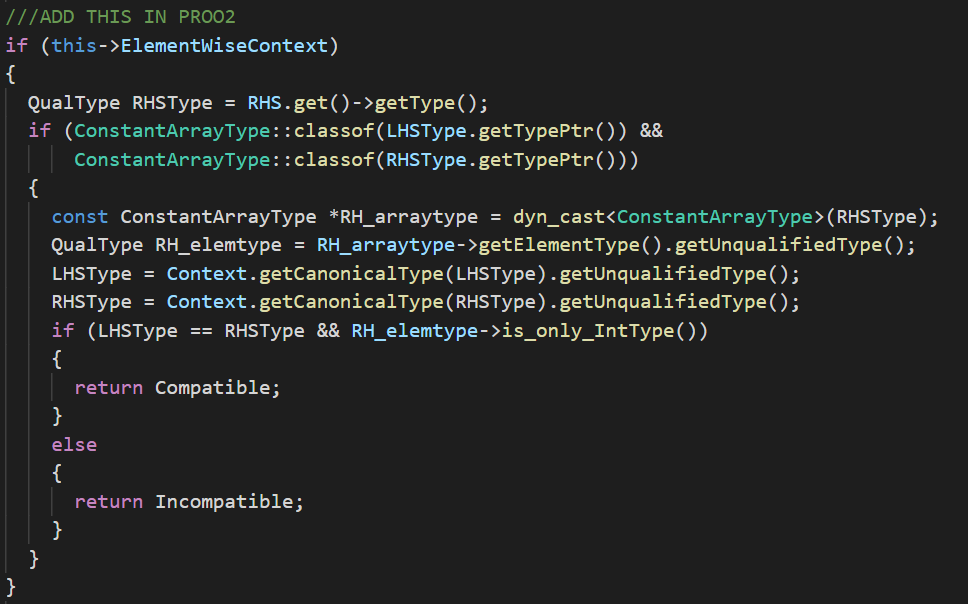
对opc进行分析：

case1: 赋值



此时我们要改动check函数，使其支持类似A=B的数组赋值功能。

修改CheckAssignmentOperands函数中的CheckSingleAssignmentConstraints函数，在函数中加入下列代码：



代码当ElementWiseContext为true时开始执行，ElementWiseContext即为在PRO01我们实现的对函数定义的elementWise标注，如果为真则说明该定义支持element Wise操作。

首先调用ConstantArrayType::classof函数，判定两个数组是否均为常数组类型。如果都是，首先调用getElmentType返回QualType类型的ElementType，再调用此类型的public操作getUnqualifiedType得到每个元素的类型。

之后，调用AST语法树中的public操作getCanonicalType：首先调用QualType类型的LHSType（RHSType）下的public操作getCanonicalType（这里有名字空间的调用）返回一个切割好的QualType类型的变量，随后调用CanQualType名字空间下的CreateUnsafe创建对象，并最终返回。

最后，判断lefthandside与righthandside的类型是否相同，并且righthandside的元素必须为int类型（通过调用自己实现的is\_only\_IntType函数），若满足就返回可编译。

总体上流程是这样，但实际上还需要做一些补充。

对于A =（B + C）这样的表达式，是可以进行编译的，因为对于赋值号而言，右侧的（B + C）在AST中存在一个可访问的右值，所以可以拿来给A赋值；但是对于（B + C）= A这样的表达式，却是不可以赋值的，因为左侧不存在一个可访问的左值。为解决这样的左右差异，我们需要查看CheckAssignmentOperands函数中是如何处理左侧的左值性，如下：

*// Verify that LHS is a modifiable lvalue, and emit error if not.*

  if (**CheckForModifiableLvalue**(LHSExpr, Loc, \*this))

    return **QualType**();

可以发现，基本的处理逻辑就是调用CheckForModifiableLvalue检查left\_hand\_side是否为可编辑的左值：

static bool CheckForModifiableLvalue(Expr \*E, SourceLocation Loc, Sema

&S) {

……

  unsigned Diag = 0;

  bool NeedType = false;

  switch (IsLV) {

……

case **Expr**::MLV\_ArrayTemporary:

    Diag = **diag**::err\_typecheck\_array\_not\_modifiable\_lvalue;

    NeedType = true;

    break;

……

此处的MLV\_ArrayTemporary就是描述的形如（A+B）产生的表达式类型，其中A和B都是数组。如果检查时进入了该case，说明在赋值号左侧出现了形如（A+B）的表达式，这样的赋值是不合法的，我们做出以下修改：

case **Expr**::MLV\_ArrayTemporary:

    if (S.ElementWiseContext) return false;

    Diag = **diag**::err\_typecheck\_array\_not\_modifiable\_lvalue;

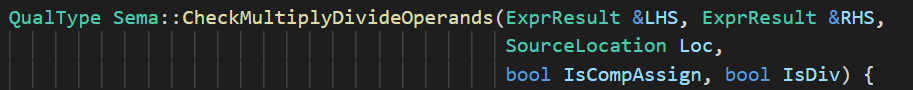
    NeedType = true;

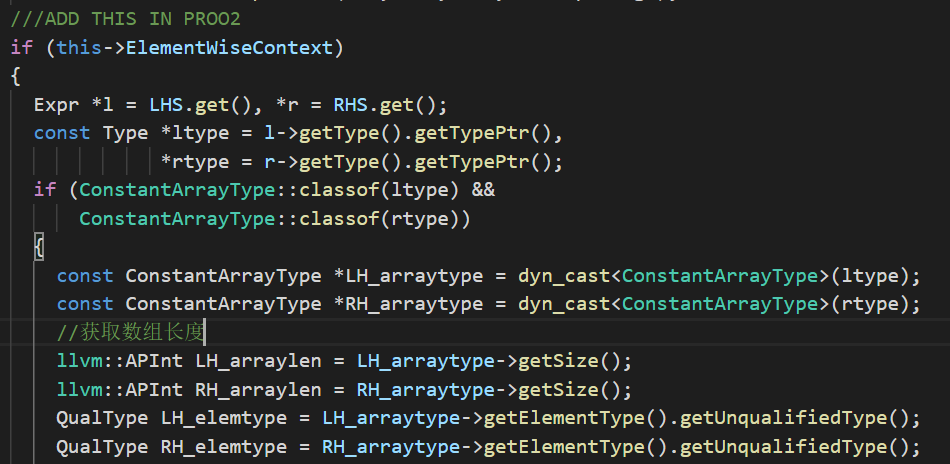
    break;

进入该case，如果发现ElementWiseContext为1，也就是编译制导含有ElementWise，就要对CheckAssignmentOperands返回false，编译报错。

case2: 乘除法

修改函数：





当ELementWiseContext为true时，即函数有#elementWise的定义时：

与之前的类似，先确定左右表达式是否均为常数组类型，之后通过类中的getSize方法获得数组的长度，通过getType方法获得数组元素的类型（此部分过程与case1完全相同）。

之后：



根据elementWise的定义，我们需要两个数组等长，即LH\_arraylen==RH\_arraylen，元素相同且为整数，即LH\_elemtype == RH\_elemtype && LH\_elemtype->is\_only\_IntType()。

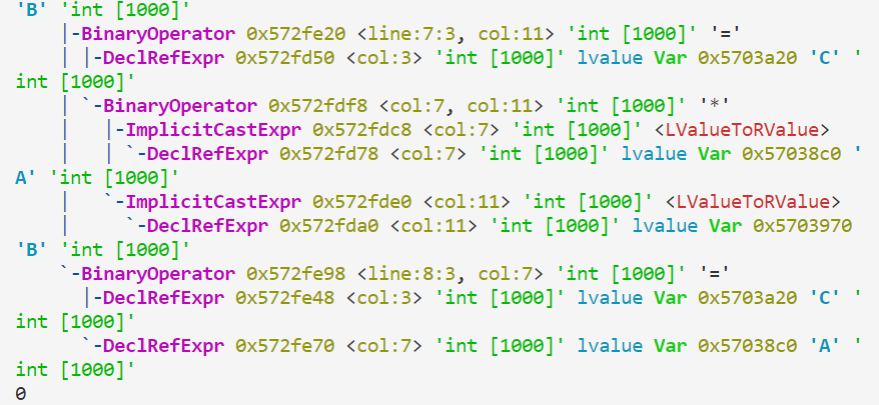
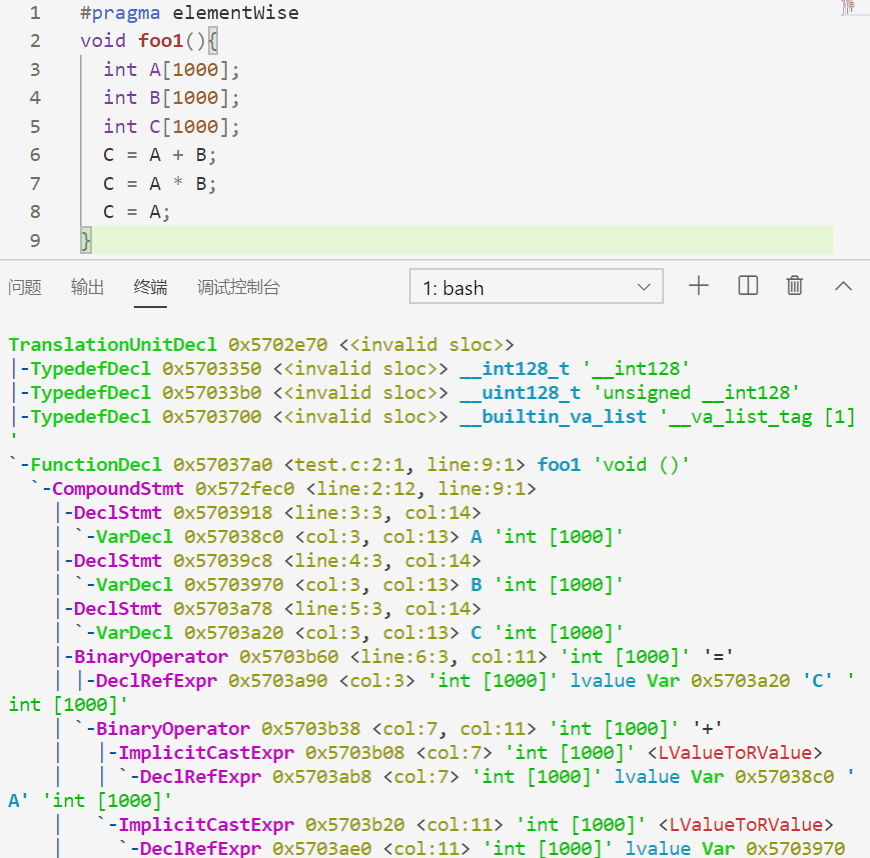
之后，如果表达式不是右值，那么新建一个类型转化，将表达式转化为右值的情况。ImplicitCastExpr::Create实现加入一层ImplicitCastExpr后的AST节点的创建，我们随后将原有结点替换为新生成的结点即可。

case3: 加法

实现过程与case2乘除法完全相同。

四、 结果测试：

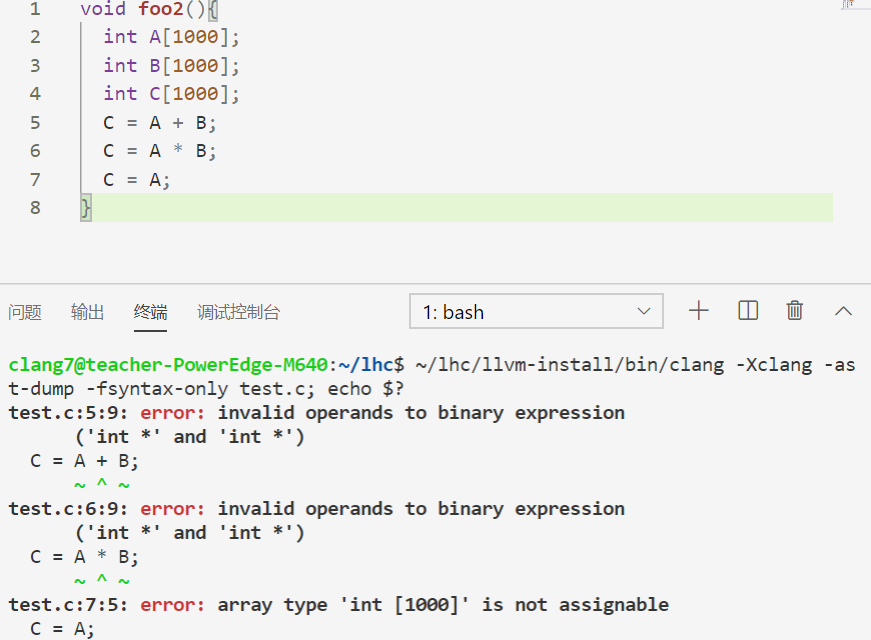
Test 1：



【测试结果】

打印0，未报编译错误，合法。

Test 2：

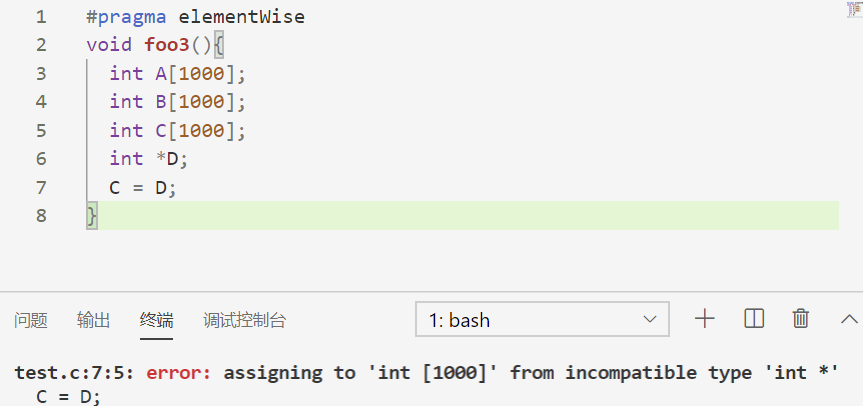


【测试结果】

编译报错：不合法的操作符，以及不允许的赋值对象。

这是因为缺少了编译制导elementwise。

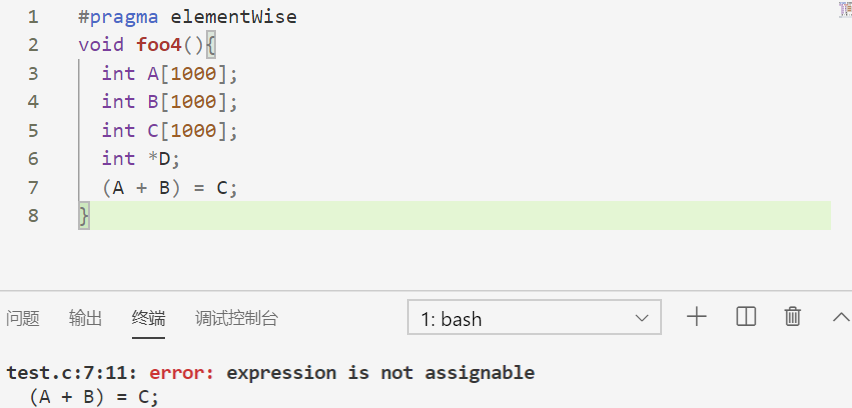
Test 3：



【测试结果】

编译报错：不合法的赋值，赋值左右类型不能兼容。

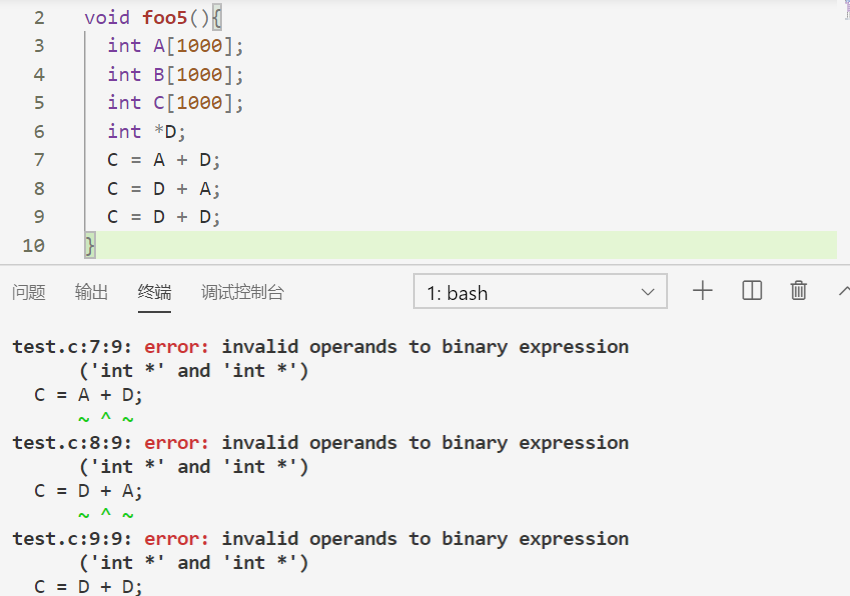
Test 4：



【测试结果】

编译报错：表达式不可被赋值。

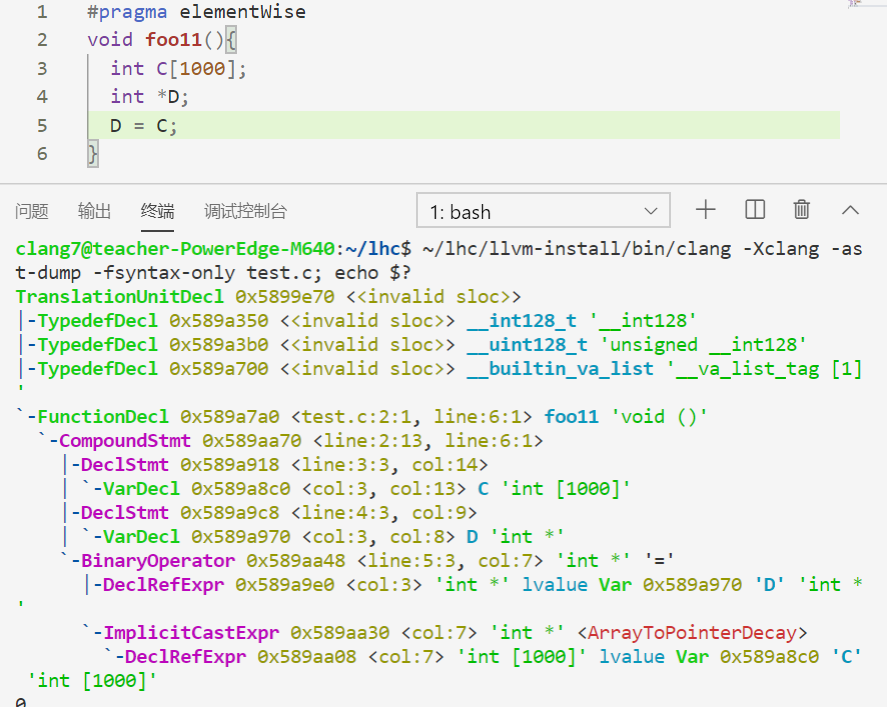
Test 5：



【测试结果】

编译报错：不合法的操作对象，二元操作对象为指针类。

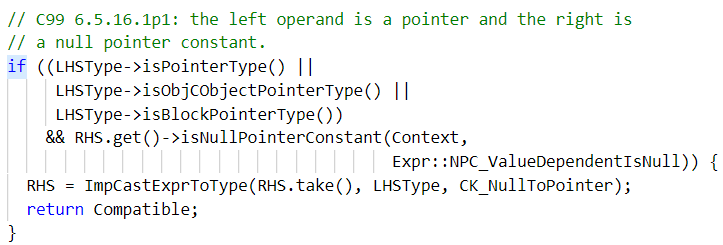
Test 6：



【测试结果】

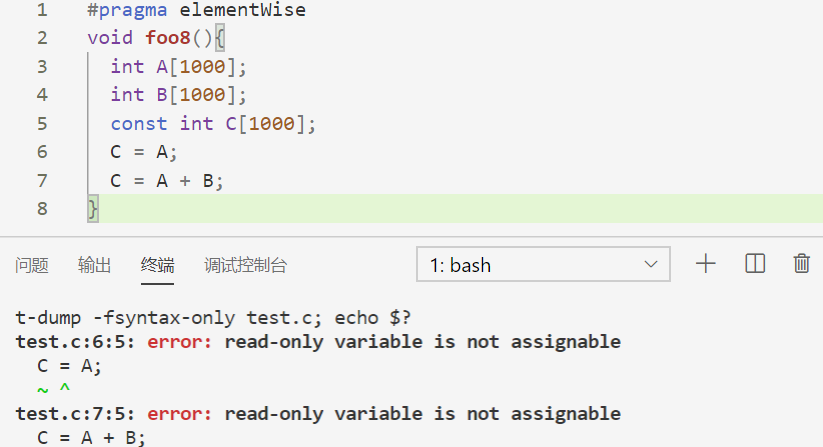
输出0，编译无报错。

这里需要注意该测试与Test 3的区别，由于CheckSingleAssignmentConstraints中存在



的特判，我们允许给指针赋常数值（如数组起始地址），并且会将一层到指针的类型转换附加在原RHS外。

Test 7：



【测试结果】

编译报错：常量不可被赋值。

Test 8：



【测试结果】

输出0，编译无报错。

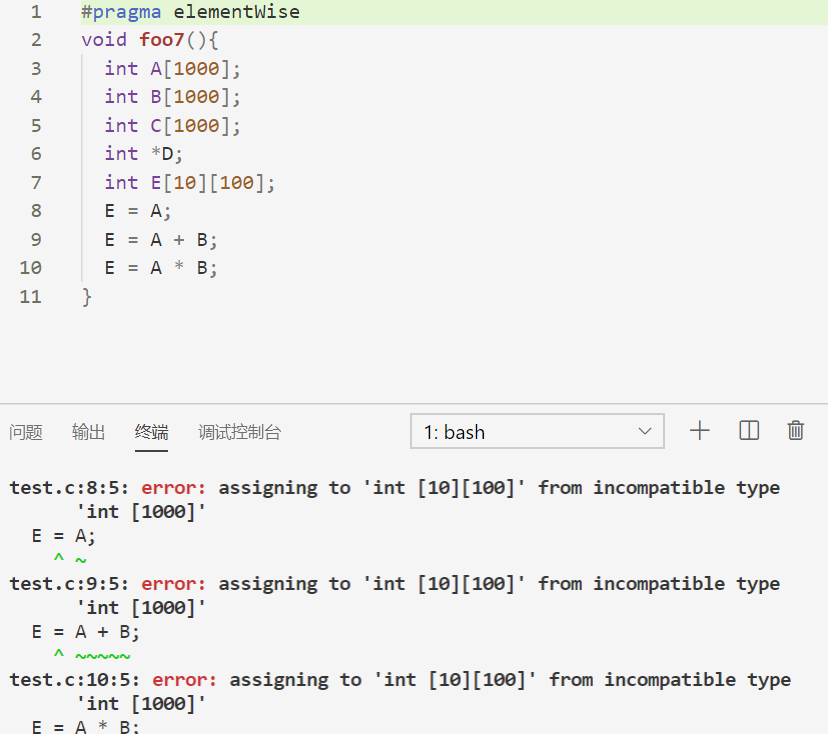
Test 9：



【测试结果】

输出0，编译无报错。

Test 10：



【测试结果】

编译报错：类型不匹配。

五、总结

本次实验主要是熟悉class Sema的结构，尤其是type相关方面。通过判断数组类型，数组长度，数组元素类型以及进行左值与右值的转换，使得AST中对于数组也支持形如A = B + C的子树。

在结合测试样例阅读理解了相关的代码框架之后，我们可以检查已有代码是否对各种情况进行了正确的处理，从而最终通过所有测试。

1. 实验结果总结

本次实验成功实现了任务要求，覆盖了各种要求情况。

1. 分成员总结

李昊宸：

本次实验负责实验代码的书写，结果测试和实验报告的补充。在PRO 01的基础上，本次实验需要修改的地方少了很多，需要修改哪里以及如何修改老师也给了很多提示。仿照llvm源码的格式，增加新的功能确实很有趣（除了一但出bug几乎无法确定是哪里的问题外，因为不能查看全部的源码）。

李颖彦：

本次实验负责实验报告的书写。本次实验相较于第一次实验更加清晰，可能是老师给了更多提示的缘故。三种运算的实现大同小异，重点注意加法和乘法时左值右值的类型转换。

陆润宇：

本次实验负责实验报告的修改和补充。本次实验基础的部分（三种运算的类型检查和AST维护）可以结合讲义中的提示，参考代码框架中其他的代码段落进行补充。Assignment情况下的类型转换会相对复杂一点，也是测试样例中比较特殊的地方，需要更加仔细地理解框架才能给出恰当的处理。