# 编译原理研讨课实验PR002实验报告

## 任务说明

本部分的内容就在于扩展C99的语言标准以支持对静态数组的element-wise的加乘操作，具体来说，形如：

```
#pragma elementWise
int func_name(){
    int A[1000];
    int B[1000];
    int C[1000];
    C = A + B;
    C = A * B;
    C = A;
    return 0;
}
```

的代码，等价于如下代码：

```
#pragma elementWise
int func_name(){
    int A[1000];
    int B[1000];
    int C[1000];
    for(int i = 0; i < 1000; i++)
        C[i] = A[i] + B[i];
    for(int i = 0; i < 1000; i++)
        C[i] = A[i] * B[i];
    for(int i = 0; i < 1000; i++)
        C[i] = A[i];
    return 0;
}
```

**要求如下：**

1. 支持'+','*','='三种操作
2. 支持C语言标准的int类型
3. 操作数应为静态大小的一维数组
4. 扩展AST的表示已支持element-wise的操作
5. 操作匹配：类型匹配（静态数组，类型相同），大小匹配（大小相等）
6. 生成合法的AST
7. 不破坏原有C语言代码的语义

## 成员组成

| 姓名 | 学号 |
| --- | --- |
| 万炎广 | 2017K8009907017 |
| 张磊 | 2017K8009922027 |
| 郭豪 | 2017K8009929011 |

# 实验设计

## 设计思路

需要增加的是对`+`,`*`,`=`三个操作符号的支持。这三个符号都有对应的处理流。在原有的情况下，数组进行操作会进行报错。因此主要修改三个操作符号对应的代码，使得`elementWise`情况下的函数内，允许数组进行该操作。

## 实验实现

首先定位到`+`操作：

文件：`SemaExpr.cpp`，函数`CheckAdditionOperands()`：增加代码如下：

```
if (Sema::NeedElementWiseFlag == 1 && //如果有ElementWise标记
      ConstantArrayType::classof(LHS.get()->getType().getTypePtr()) && //且是数组
      ConstantArrayType::classof(RHS.get()->getType().getTypePtr())) {
    const Expr *LHS_Expr = LHS.get();
    const Type *LHS_Type = LHS_Expr->getType().getTypePtr();
    const Expr *RHS_Expr = RHS.get();
    const Type *RHS_Type = RHS_Expr->getType().getTypePtr();
    const ConstantArrayType *LHS_t = dyn_cast<ConstantArrayType>(LHS_Type);
    const ConstantArrayType *RHS_t = dyn_cast<ConstantArrayType>(RHS_Type);
    const llvm::APInt LHS_Data_Num = LHS_t->getSize();
    const llvm::APInt RHS_Data_Num = RHS_t->getSize();
    QualType LHS_Data_t = LHS_t->getElementType().getUnqualifiedType();
    QualType RHS_Data_t = RHS_t->getElementType().getUnqualifiedType();
    if(LHS_Data_Num == RHS_Data_Num && //判断大小
       LHS_Data_t == RHS_Data_t && //判断类型
       LHS_t->getElementType()->isIntegerType()) { //判断是不是int类型
      if(!(LHS_Expr->isRValue())) { //左值转右值
        Qualifiers temp;
        ImplicitCastExpr * LHS_ltor = ImplicitCastExpr::Create(
          Context,
          Context.getUnqualifiedArrayType(LHS_Expr-
>getType().getUnqualifiedType(), temp),
          CK_LValueToRValue,
          const_cast<Expr*>(LHS_Expr),
          0,
          VK_RValue);
        LHS = LHS_ltor;
      }
      if(!(RHS_Expr->isRValue())) { //左值转右值
        Qualifiers temp;
        ImplicitCastExpr * RHS_ltor = ImplicitCastExpr::Create(
          Context,
          Context.getUnqualifiedArrayType(RHS_Expr-
>getType().getUnqualifiedType(), temp),
```

```
          CK_LValueToRValue,
          const_cast<Expr*>(RHS_Expr),
          0,
          VK_RValue);
        RHS = RHS_ltor;
      }
      return LHS.get()->getType();
    }
  }
//如果不满足以上条件，在后续处理中会报错并退出
```

同样，在乘法操作中增加以上内容。

接下来，处理赋值符号：

文件：`SemaExpr.cpp`，函数`CheckAssignmentOperands()`：

```
// Verify that LHS is a modifiable lvalue, and emit error if not.
  if (CheckForModifiableLvalue(LHSExpr, Loc, *this))
    return QualType();
//要放在这个函数之后，这样不可修改的量就不会进入判断导致错误。
//类似加法乘法一样的操作，但是由于没有LHS这个量，因此使用LHSExpr这个输入代替LHS->get()
  if (Sema::NeedElementWiseFlag == 1 &&
      ConstantArrayType::classof(LHSExpr->getType().getTypePtr()) &&
      ConstantArrayType::classof(RHS.get()->getType().getTypePtr())) {
    const Expr *LHS_Expr = LHSExpr;
    const Type *LHS_Type = LHS_Expr->getType().getTypePtr();
    const Expr *RHS_Expr = RHS.get();
    const Type *RHS_Type = RHS_Expr->getType().getTypePtr();
    const ConstantArrayType *LHS_t = dyn_cast<ConstantArrayType>(LHS_Type);
    const ConstantArrayType *RHS_t = dyn_cast<ConstantArrayType>(RHS_Type);
    const llvm::APInt LHS_Data_Num = LHS_t->getSize();
    const llvm::APInt RHS_Data_Num = RHS_t->getSize();
    QualType LHS_Data_t = LHS_t->getElementType().getUnqualifiedType();
    QualType RHS_Data_t = RHS_t->getElementType().getUnqualifiedType();
    if(LHS_Data_Num == RHS_Data_Num &&
       LHS_Data_t == RHS_Data_t &&
       LHS_t->getElementType()->isIntegerType()) {
      if(!(LHS_Expr->isRValue())) {
        if(!(RHS_Expr->isRValue())) {
        Qualifiers temp;
        ImplicitCastExpr * RHS_ltor = ImplicitCastExpr::Create(
          Context,
          Context.getUnqualifiedArrayType(RHS_Expr-
>getType().getUnqualifiedType(), temp),
          CK_LValueToRValue,
          const_cast<Expr*>(RHS_Expr),
          0,
          VK_RValue);
        RHS = RHS_ltor;
        }
        return LHSExpr->getType();
      }
    }
  }
```

由于数组类型本来是属于不可复制的项，因此还需要修改检查的值使其可以被修改：

文件`ExprClassification.cpp`，函数`IsModifiable()`：（也就是上面赋值语句的第一个检查项目）

```
if (CT->isArrayType())
    return Cl::CM_Modifiable;//更改之后使得数组项可以被修改
```

# 总结

## 实验结果总结

对于给出的样例我们进行测试：

### case 1

```
#pragma elementWise
void foo1(){
  int A[1000];
  int B[1000];
  int C[1000];
  C = A + B;
  C = A * B;
  C = A;
}
```

结果为0，表示正确。

### case 2

```
void foo2(){
  int A[1000];
  int B[1000];
  int C[1000];
  C = A + B;
  C = A * B;
  C = A;
}
```

```
PR002TEST/pr002_case2.c:5:9: error: invalid operands to binary expression ('int *' and 'int *')
  C = A + B;
      ~ ^ ~
PR002TEST/pr002_case2.c:6:9: error: invalid operands to binary expression ('int *' and 'int *')
  C = A * B;
      ~ ^ ~
PR002TEST/pr002_case2.c:7:5: error: assigning to 'int [1000]' from incompatible type 'int [1000]'
  C = A;
    ^ ~
```

由于没有elementWise标记，不进入新增的if判断，在原先语句中自然报错。结果为1。

### case 3

```
#pragma elementWise
void foo3(){
  int A[1000];
  int B[1000];
  int C[1000];
  int *D;
  C = D;
}
```

```
PR002TEST/pr002_case3.c:7:5: error: assigning to 'int [1000]' from incompatible type 'int *'
  C = D;
    ^ ~
```

由于D不是数组，而是指针，因此报错。输出为1。

## case 4

```
#pragma elementWise
void foo4(){
  int A[1000];
  int B[1000];
  int C[1000];
  int *D;
  (A + B) = C;
}
```

```
PR002TEST/pr002_case4.c:8:11: error: expression is not assignable
  (A + B) = C;
  ~~~~~~~ ^
```

由于A+B不能放在公式左边，产生自然报错。输出结果为1。

## case 5

```
#pragma elementWise
void foo5(){
  int A[1000];
  int B[1000];
  int C[1000];
  int *D;
  C = A + D;
  C = D + A;
  C = D + D;
}
```

```
PR002TEST/pr002_case5.c:8:9: error: invalid operands to binary expression ('int *' and 'int *')
  C = A + D;
      ~ ^ ~
PR002TEST/pr002_case5.c:9:9: error: invalid operands to binary expression ('int *' and 'int *')
  C = D + A;
      ~ ^ ~
PR002TEST/pr002_case5.c:10:9: error: invalid operands to binary expression ('int *' and 'int *')
  C = D + D;
      ~ ^ ~
```

同样是指针量无法加入运算。

## case 6

```
#pragma elementWise
void foo6(){
  int A[1000];
  int B[1000];
  int C[1000];
  int *D;
  (A + B) = C;
}
```

和case4一样。（怀疑给的测试样例有误。）

## case 7

```
#pragma elementWise
void foo7(){
  int A[1000];
  int B[1000];
  int C[1000];
  int *D;
  int E[10][100];
  E = A;
  E = A + B;
  E = A * B;
}
```

```
PR002TEST/pr002_case7.c:9:5: error: assigning to 'int [10][100]' from incompatible type 'int [1000]'
  E = A;
    ^ ~
PR002TEST/pr002_case7.c:10:5: error: assigning to 'int [10][100]' from incompatible type 'int [1000]'
  E = A + B;
    ^ ~~~~~
PR002TEST/pr002_case7.c:11:5: error: assigning to 'int [10][100]' from incompatible type 'int [1000]'
  E = A * B;
    ^ ~~~~~
```

二维数组无法加入运算，因为数组类型不同。

## case 8

```
#pragma elementWise
void foo8(){
  int A[1000];
  int B[1000];
  const int C[1000];
  C = A;
  C = A + B;
}
```

```
PR002TEST/pr002_case8.c:6:5: error: read-only variable is not assignable
  C = A;
  ~ ^
PR002TEST/pr002_case8.c:7:5: error: read-only variable is not assignable
  C = A + B;
  ~ ^
```

在assign判断的第一条语句就会导致const类型返回报错。（这也是判断语句应当放在该语句之后的原因。）

## case 9

```
#pragma elementWise
void foo9(){
  int A[1000];
  const int B[1000];
  int C[1000];
  C = B;
  C = A + B;
}
```

这个操作是合法的，const可以作为运算数，返回0。

## case 10

```
#pragma elementWise
void foo10(){
  int A[1000];
  int B[1000];
  int C[1000];
  int D[1000];
  D = A + B + C;
  D = A * B + C;
  D = (D = A + B);
  D = (A + B) * C;
  D = (A + B) * (C + D);
}
```

综合测试，没有问题。返回0。

# 分成员总结

## 万炎广

本次实验修改内容比较少，主要是熟悉掌握编译器中给中类型的方法使用，包括类型判断，返回类型，左右值转换等知识。虽然代码量较少，但是每一步都是精华。稍错漏一步都会导致错误，导致调试时间较长。总体来说本次实验较难，和组成员相互讨论调研学习后才能完成代码设计。

## 张磊

本次实验难度较大，由于实验的时候思路不是很清楚，导致出了很多bug，非常感谢万炎广同学的讲解，经过我们组员的讨论之后，思路清晰了很多，顺利完成了此次实验。本次实验也让我更进一步的了解了编译器的实现原理，强化了理论课的知识。最后，感谢我的队友和助教老师。

## 郭豪

本次实验需要改动的不多，主要实现了对elementwise的扩展，完成相应的AST修改，类型判断，转换等操作。通过本次实验，了解了相关操作的处理流程，加深了对左值右值的理解。