编译原理研讨课实验PR002任务书

C语言添加element-wise加/乘操作

实验内容：

向量数据的element-wise操作在深度学习中被普遍应用，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. 操作数类型匹配：类型相同且大小相同
5. 编译器可以直接编译符合规范的源代码文件生成二进制文件并正确执行

在PR001当中，需要做到：

1. 扩展AST的表示已支持element-wise的操作

2. 操作匹配：类型匹配（静态数组，类型相同），大小匹配（大小相等）
3. 生成合法的AST
4. 不破坏原有C语言代码的语义

示例：

Sample code 1:

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

Sample code 2:

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

Status:编译报错，因为foo2未标注支持elementWise操作。

Sample code 3:

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

Status: **不合法**

Sample code 4:

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

Status: **不合法**

Sample code 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;
}
```

Status: **不合法**

Sample code 6:

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

Status: **不合法**

Sample code 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;
}
```

`Status:` **不合法**

`Sample code 8:`

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

`Status:` **不合法**

`Sample code 9:`

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

`Status:` **合法**

`Sample code 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);
}
```

`Status:` **合法**

`Notes:` 请大家注意这个测试用例，仔细设计自己的AST表示

验收标准：

1. 实验报告提交到课程网站
    1. 课程网站有模板供参考。
2. 实验源代码放在本组的帐号下
    1. 所有文件都放在同一个目录 `/home/account_name/PR002` 中
    2. 修改后的源代码放入 `./src` 中
    3. 编写一个可执行脚本，放在 `./scripts` 下，为 `compile_and_check.sh` 。该脚本接受一个参数，参数内容为源文件名称，该脚本的功能是调用修改后的Clang编译输入参数指定的源文件，并将生成的AST输出到标准输出上。