stage-5 实验报告

姓名：郑友捷　学号：2021010771　班级：计14

step 11

本步骤要求支持数组操作

词法语法分析

为了支持数组声明定义，添加了`Indexlist`节点，代表数组的索引，并把该节点作为声明的附属节点。当声明的变量含有`Indexlist`时，说明出现了索引，应当为数组类型。

为了支持数组的访问，添加了`IndexExpr`节点，代表对数组元素的访问，以数组的变量名作为首地址，索引中的数字作为偏移量完成数组元素索引。此后每一个操作数不仅可以是一个变量，还可以是一个数组索引表达式。

由此变量的类型就不仅有int一种，还出现了Array[int]的类型，因此需要对程序进行类型检查。

语义分析

为了支持数组类型，需要新添加一种符号类型为`arraysymbol`，存储数组的索引、首地址等要素。之后在遍历时，若当前节点为一个数组，也需要检查是否与任一数组名或者变量名同名。若不存在同名，则加入到符号表中。

对于数组元素访问需要进行如下检查：

1. 数组名是否存在
2. 数组维度是否与声明时相同
3. 数组的索引下标是否为负数

但由于加入数组之后，变量类型不仅包括int，还包括Array[int]，可能会出现一个变量名与一个数组名进行运算的错误情况，因此需要进行类型检查。

类型检查包括：

1. 进行数组索引运算时，名称需要为一个数组的名字而非一个int变量的名字，如：

   ```
   int a = 1;
   a[1] = 1;
   ```

   则不合法。

2. 进行变量获取时，获取的变量应该是一个int变量，而不是一个数组的名字，如：

   ```
   int a[10];
   a = 2;
   ```

   则不合法。

对于赋值，由于step 11暂未支持集合初始化赋值，因此赋值号右边仅为一个int变量，不会出现类型问题。

中间代码生成

对于数组的中间代码主要涉及数组的声明定义与数组元素的访问、修改。

1. 对于数组的声明定义，由于目前暂未支持初始化，故只要申请内存即可。故新增了一条中间指令：

   ```
   ALLOC size
   ```

   代表分配 size 字节的内存，并返回内存首地址。

   对于全局数组，不需要生成中间变量，如同全局变量一样直接交给后端处理。

   对于局部数组，先为数组分配一个与名字对应的寄存器之后，申请与数组大小相匹配的内存之后让寄存器指向这块内存首地址即可。

2. 对于数组元素的访问：

   若是全局数组，通过`LOAD_SYMBOL`指令获取数组的首地址。若是局部数组，通过之前分配的虚拟寄存器获取其首地址。

   通过数组的索引计算出被访问元素相对于首地址的偏移量，然后使用`LOAD`指令从指定的地址获得对应的数组元素即可。

3. 对于数组元素的修改：

   修改了数组元素之后，由于寄存器是临时分配的，因此需要把修改实时响应到内存中。

   先通过步骤2获得被修改数组元素的地址，之后使用`STORE`指令（类似于sw指令）将新修改的元素载入到对应地址中。

目标代码生成

对于数组需要考虑数组内存的申请以及元素的访问修改。

1. 对于内存申请，若当前指令为数组的初始化，即为`ALLOC`指令，对应为`temp = Alloc size`，则只需要让栈帧向低地址移动对应大小开辟新的空间用于数组存储，并将`temp`对应分配的寄存器的值改为`sp`的值，即新的空间的首地址。
2. 对于元素访问，只需要获取数组对应的寄存器上存储的地址，然后计算其偏移量便可以完成元素访问。
3. 对于元素修改，在获取到对应地址之后使用`sw`指令便可以完成修改。

step 12

本步骤涉及数组初始化和传参

词法语法分析

对于数组初始化，新建了一个节点`InitList`代表初始化的集合，仅有类型为数组的节点才可以接受集合的初始化。因此这个节点仅会出现在对数组的声明与全局声明部分。

语义分析

相比于之前step 11，新增了数组初始化，因此需要添加更多语义检查与类型检查。

对于语义检查部分：

1. 集合仅有可能在数组初始化的时候出现，其他时候的出现均为不合法
2. 数组的索引仅允许为非负整数，不允许携带变量。
3. 数组参数定义时，其维度需要严格等于定义时的维度，第一维允许内容为空
4. 允许传入参数时参数为数组名，代表传入数组首地址，其他情况下不允许直接访问数组名。

对于类型检查部分：

1. 函数传递参数时，不同类型的变量传递顺序要保持一致。若第一二个参数类型为int与Array[int]，则传递的参数也需要严格按照这个类型顺序。
2. 传入数组的维度大小需要与参数定义时的大小保持一致（第一维允许不一样，但其他维度必须保持大小一致）

中间代码生成

对于数组初始化，若采用集合初始化，则在集合初始化未覆盖的元素统一置为0。因此先调用fill_n函数使数组元素清0，之后对集合中每一个元素进行赋值操作即可。

对于数组传参，应当传入的是数组的首地址，因此需要从全局或者局部获取首地址。若是全局数组，采用`LOAD_SYMBOL`TAC码即可。

目标代码生成

若中间代码生成无问题，则对于数组传参部分，目标代码不需要做额外的修改。

对于全局数组初始化部分，需要注意：

1. 若采用了集合初始化的方法，应当让这个数组处于`.data`段，对于有初始化值的元素设置为`.word init_value`，其余元素统一置为`.word 0`代表初始化值为0。

   例：全局数组`int a[4] = {1, 2};`对应的目标代码部分为：

   ```
   .global a
   a:
           .word   1                               # 0x1
           .word   2                               # 0x2
           .word   0                               # 0x0
           .word   0                               # 0x0
   ```

2. 若未初始化数组，则将数组置于`.bss`段即可，与一般的全局变量一致，并开好对应大小的空间。

对于局部数组初始化，若中间代码生成无误，则不需要做额外修改。

思考题

1. 变长数组操作如下：
    1. 对于变长数组，在现有实验框架下，`bruteregalloc.py`遍历语句生成初步的riscv指令时，只对该数组名分配寄存器或者栈帧空间，占据的空间仅为一个字4个字节，记录该数组的首地址，但目前该首地址仍为空。
    2. 当所有固定大小的局部变量已经分配完毕，此时栈帧会进行第一次开辟，获取，即将输出已经生成的riscv指令时，判断当前的指令是否是为了生成变长数组。若是，则寻找决定其长度的变量的寄存器，从而获得数组的实际长度。此时让存储该数组首地址的寄存器或者栈帧空间的值改为sp，代表该变长数组的首地址为sp。接着让sp指针继续向低地址移动，扩大栈帧空间，作为该数组开辟出来的空间。这样就完成了变长数组的地址分配，并更新sp的偏移量。
    3. 在函数结束之后，按照更新后的sp偏移量向高地址移动sp，回收包括动态数组等栈帧空间。
2. 传参时一般会把数组的第一维忽略，因为若仅传入一维数组，只需要传入该数组的首地址，因此编译器会直接把第一维忽略，视为传入了一个指针。而对于多维数组，第一维的大小没有意义，编译器仅关心数组的首地址在哪里以及地址的变化规则，不关心地址的大小限制，所以第一维会被忽略。