编绎原理 · stage-5 · 实验报告

计01　容逸朗　2020010869

实验内容

实验目标

本阶段需要支持 MiniDecaf 语言的数组（包含下标操作、初始化和传参）。

实验思路

数组初始化

在符号表阶段访问 `ast::ArrayType` 时，可以递归构造一个层层嵌套的 `type::ArrayType`，这是为了方便后续数组比较和求数组大小所致的。

（需要注意 `DimList *iexpr` 的值是倒序存储的，即 `arr[1][2][3][4]` 会被记为 {4, 3, 2, 1}）

```
void SemPass1::visit(ast::ArrayType *atype) {
    ast::DimList *iexpr = atype->index;
    assert(iexpr != NULL);
    Type *t = BaseType::Int;

    for (auto it = iexpr->begin(); it != iexpr->end(); ++it) {
        if (*it <= 0)
            issue(atype->getLocation(), new ZeroLengthedArrayError());
        t = new ArrayType(t, *it);
    }

    atype->ATTR(type) = t;
}
```

在语法分析阶段，若数组有初值，那么我们可以把大括号内的初始值顺序放入一个列表中，然后对于不同类型的数组要做不同的处理：

- 对于全局变量：

  我们可以在符号表构建阶段访问 `VarDecl` 节点时调用 `Variable::setArrayInit`（此函数是自定义的，其功能是保存一个 `List<int>`）保存全局数组的初始值。

  在后端的 `RiscvDesc::emitPieces` 遍历全局符号表，若符号是数组类型则可以用如下方式输出：

```
/* 下面的例子是全局变量 arr[4] = {3, 2, 1} 的初始化*/
.data
          .globl arr
          .size arr, 16
arr:
          .word 3
          .word 2
          .word 1
          .zero 4
```

注：若全局数组无初始化则用 `.zero {arr_size}` 填充。

- 对于局部变量：

  在中端翻译 `VarDecl` 节点时，首先调用 `TransHelper::genAlloc` 的方法分配一段栈上的连续空间以存放数组，若初始化不为空还需要调用 `TransHelper::genStore` 函数把数组初始值结合偏移量存入正确的栈空间。

下标操作

一般而言，数组的下标操作是和左值节点 `LvalueExpr` 相关的，在该节点内只需要找到数组的首地址并计算出偏移量即可，具体来说，我们可以调用 `genLoadSymbol` 的方式取得全局数组的首位位置或使用 `Symbol::getTemp()` 的方法取得局部数组的初始值，然后使用数组索引节点 `IndexExpr` 的值作偏移量即可找到对应位置。

其中，对于数组索引节点 `IndexExpr` 的翻译如下：

- 由于在符号表创建阶段时数组类型是层层嵌套的，因此对于数组索引运算可以直接调用 `ArrayType::getSize()` 的方法取得子类的大小，再把值乘上对应维度的索引值即可，例如：
  - 在 `arr[10][5][5]` 中索引 `arr[4][3][2]` 时：取位置 `arr + 4*100 + 3*20 + 2*4 = arr + 468`
- 当然也可以使用 `ArrayType::getLength()` 取得下一维长度，然后用递归的方式计算索引位置，例如：
  - 在 `arr[10][5][5]` 中索引 `arr[4][3][2]` 时：取位置 `arr + ((4*5+3)*5+2)*4 = arr + 468`

数组比较 / 类型检查

由于增加了非 `INT` 的类型，因此需要增加对数组类型的比较方式。具体来说，我对 `type::ArrayType` 的 `equal` 函数进行了修改，使得检查函数类型时可以递归比较（除第一维以外的）的每一层大小。显然，若每一维的子类大小都相等，那么（除第一维以外的）每一维的长度也是相等的。

```
bool ArrayType::equal(Type *t) {
    mind_assert(NULL != t);

    if (!t->isArrayType())
        return false;
    else {
        ArrayType *at = (ArrayType *)t;
        mind_assert(at->getElementType()->getSize() == getElementType()->getSize());
        return (element_type->equal(((ArrayType *)t)->element_type));
    }
}
```

注：由于不会出现 `int a[10], b[10]; a = b;` 的情况，因此不比较第一维的长度不会对程序正确性有影响。

数组传参

由于作为函数参数的数组类型第一维可以为空，在语法分析时需要加入对应情况的语法：

```
OptIndex    : LBRACK RBRACK
                { $$ = new ast::DimList();
                  $$->append(1); }
            | LBRACK RBRACK Index
                { $$ = $3;
                  $$->append(1);}
            | LBRACK ICONST RBRACK
                { $$ = new ast::DimList();
                  $$->append(1); }
            | LBRACK ICONST RBRACK Index
                { $$ = $4;
                  $$->append(1); }
            ;
```

- 注1: Index 接受如下语法： `('[' ICONST ']')+`
- 注2: OptIndex 的写法无视了函数第一维的大小（全部设为 1 是为了方便数组检查，同时为了避免影响数组维度计算所致的）

数组传参时需要传入数组首位的地址，这样做可以令函数内部调用数组时采用局部数组访问的方式，从而统一了数组寻参的方式。由于把数组作为一个变量传参，因此标识符（如 `arr`）会被语法分析树判为一个 `VarRef` 节点（对应左值类型为 `Lvalue::SIMPLE_VAR`）而不是一个 `ArrayRef` 节点。

故在翻译阶段遍历到 `LvalueExpr` 节点时需要特判全局数组变量的引用：只需调用 `genLoadSymbol` 取得数组首地址而**不需要**调用 `genLoad` 取出值。对于局部数组则只需要调用对应符号的 `Temp` 值即可。

思考题

1. C 语言规范规定，允许局部变量是可变长度的数组（Variable Length Array，VLA），在我们的实验中为了简化，选择不支持它。请你简要回答，如果我们决定支持一维的可变长度的数组(即允许类似 `int n = 5; int a[n];` 这种，但仍然不允许类似 `int n = ...; int m = ...; int a[n][m];` 这种)，而且要求数组仍然保存在栈上（即不允许用堆上的动态内存申请，如 `malloc` 等来实现它），应该在现有的实现基础上做出那些改动？

- 由于我们仅支持一维的可变长度的数组，因此可以加入新的语法节点：

  ```
  VLA         : Type IDENTIFIER LBRACK Expr RBRACK
  ```

  由于数组仅有一维，因此对应的数组节点可以记为 `ArrayType(BaseType::Int, 1)`

  （由于第一维的大小不参与索引计算和数组比较，因此第一维赋为任意正整数都不会影响程序正确性）
- 在翻译此节点时，首先把运算式 `Expr` 的 `Temp` 值和 `4` 相乘得到数组大小，然后 `ALLOC` 此大小的空间即可：

```
T0 = Expr→ATTR(val)
T1 = 4
T2 = T0 * T1
T3 = ALLOC T2
```

注：由于数组第一维的大小仅对内存分配有影响，和索引无关。因此实现一维 VLA 时不需增加符号记录数组开辟时第一维的大小。

- 为了支持非常数的栈空间分配，我们需要增加 ALLOC 的另一种翻译方式：

```
ALLOC t0:
            sub sp, sp, t0
```

- 由于函数作用域结束时栈桢会恢复为调用者的栈桢，因此不用考虑数组释放内存的问题。

2. 作为函数参数的数组类型第一维可以为空。事实上，在 C/C++ 中即使标明了第一维的大小，类型检查依然会当作第一维是空的情况处理。如何理解这一设计？

- 首先回顾数组索引的操作方式：
  - 在 `arr[10][5][5]` 中索引 `arr[4][3][2]` 时：取位置 `arr + ((4*5+3)*5+2)*4 = arr + 468`
- 从上面的式子可知，第一维的长度并不参与数组索引运算，因此第一维无论赋值与否也不会影响索引值的正确性。
- 同时由于这是一个函数参数，并不需要为数组分配内存空间，故不会因为第一维为空而影响内存分配。
- 既然第一维的长度在函数中是无作用的，处理数组时可以统一地把第一维当作是空的情况处理即可。

参考

实现代码的过程中参考了以下资料：

1. 实验思路指导与问答墙.