stage-5 实验报告

秦若愚 2019011115

实验内容

step-11

实验目标

- 支持数组的声明
- 数组的下标操作

实验内容

在词法语法分析阶段，为了支持数组的声明，我在 frontend/parser/ply_parser.py 中添加了解析数组声明的函数，并将解析得到的数组大小存于 `Declaration` 类的 array_size 属性中。为了支持数组的下标操作，我新增了 `ArrayCall` 这一 AST 节点，用于表示对数组的下标访问，并在 frontend/parser/ply_parser.py 中添加了相应的解析函数。

在语义分析的符号表构建阶段，我在 frontend/typecheck/namer.py 的 visitDeclaration 函数中添加了对数组类型的声明的处理，此外还新增了 visitArrayCall 函数用于处理数组的下标访问。

由于引入了数组，所以从 step-11 开始需要进行类型检查。我在 frontend/typecheck/typer.py 中新增了类型检查的操作，主要检查表达式中的变量类型（包括 int 类型和数组类型）是否合法。和 namer.py 中类似，typer.py 中也对 AST 进行了一遍扫描，但在这里只需要检查 AST 节点本身，不需要使用 Visitor（设置为 None）。

在中间代码生成阶段，我在 frontend/tacgen/tacgen.py 的 visitIdentifier、visitDeclaration、visitAssignment 以及 step-10 中新增的全局变量处理过程中均添加了对数组类型变量的处理，此外还新增了 visitArrayCall 函数用于处理对数组的下标访问。与 int 型变量的处理主要有三点不同：（1）数组类型额外新增了 array_size 属性，因此对应的 symbol 也要增加这一属性。（2）全局数组声明时需要将其信息加入 TACProg 中（同 step-10），局部数组声明时需要使用新增的 TAC 指令 `Alloc` 来分配内存空间，我在 utils/tac/funcvisitor.py 中定义了 visitAlloc 来实现。（3）数组在访问时需要调用 visitArrayCall 来获取下标对应的栈上或者全局数据段中的地址，然后通过 visitLoadWord（见 step-10）来读取；而数组在赋值时同样需要 visitArrayCall 来获取地址，然后通过 visitStoreWord（见 step-10）来赋值。

在目标代码生成阶段，需要将 TAC 指令 `Alloc` 翻译为 riscv 代码，具体实现是当前函数的栈帧空间扩大等同于数组大小的空间，同时将栈上数组的首地址分配给一个寄存器。此外，未初始化的全局数组需要存放于 bss 段中，.space 后面的数字需要是数组的空间大小（单位：字节）。

step-12

实验目标

- 支持数组的初始化
- 支持数组传参

实验内容

在词法语法分析阶段，为了支持数组的初始化，我在 frontend/parser/ply_parser.py 中解析数组声明的函数组中添加了对数组初始化的解析，并将初始化值存于 `Declaration` 类的 array_init 属性中。为了支持数组传参，我在解析函数参数的函数组中添加了对数组类型参数的解析，支持参数数组第一维为空和不为空两种形式。

在语义分析的符号表构建阶段，在 frontend/typecheck/namer.py 中的 visitDeclaration 函数中，我将数组类型变量的初始化值存于 symbol 的 initValue 中。此外，为了后续阶段能调用 fill_n 函数，我在 visitProgram 中添加了 fill_n 函数对应的 FuncSymbol。

在语义分析的类型检查阶段，在 frontend/typecheck/typer.py 中，我添加了对数组初始化值和调用函数时参数的类型检查。

在中间代码生成阶段，在 frontend/tacgen/tacgen.py 中，我将局部数组的初始化转化为了以下 TAC 指令：

```
_T0 = ALLOC [array_size]
PARAM _T0
_T1 = 0
PARAM _T1
_T2 = [int(array_size/4)]
PARAM _T2
_T3 = CALL FUNCTION<fill_n>
```

即通过调用 fill_n 函数来精简初始化语句。为了能调用 fill_n 函数，还需要在 ProgramWriter 添加fill_n 对应的 FuncLabel。

在目标代码生成阶段，只需将初始化的全局数组存放于 .data 段，其中初始化的值用 .word [value] 声明，剩下默认初始化为 0 的部分用 .zero [default_num*4] 声明。

思考题

step-11

1. 答：

对于这种情况，需要在函数体中为 VLA 分配栈上空间时动态改变 SP 寄存器的值，每次声明 VLA 时就从 SP 中减去对应空间大小的值。同时，需要额外使用一个寄存器维护 VLA 使用空间总大小，以便正确读取栈上保存的其它数据（固定长度的数组以及保存的寄存器的值）。

step-12

1. 答：

因为第一维大小没有实际的作用。首先第一维大小不同不影响数组元素的类型，其次函数中并不会为参数中的数组分配内存空间，最后 C/C++ 编译期间并不检查数组越界。