> 刘雅迪
>
> 计26
>
> 学号：2021010521

step10: 全局变量

思考题

1. 写出 `la v0, a` 这一 RiscV 伪指令可能会被转换成哪些 RiscV 指令的组合（说出两种可能即可）。

   答：

   ①使用 `auipc` 和 `addi`：当a是一个相对地址时

   ```
   auipc v0, %pcrel_hi(a)    # 将a的高20位地址加载到v0中
   addi  v0, v0, %pcrel_lo(a) # 加载a的低12位地址并加到v0上
   ```

   ②使用 `lui` 和 `addi`：当a是一个绝对地址时

   ```
   lui  v0, %hi(a)       # 将符号a的高20位加载到v0中
   addi v0, v0, %lo(a)   # 加载符号a的低12位并加到v0上
   ```

实验内容

一、词法语法分析

修改`frontend/parser/ply_parser.py`和`frontend/ast/tree.py`中与Program有关的部分，加入declaration：

```python
def p_program(p):
    """
    program : program element
    """
    if p[2] is not NULL:
        p[1].children.append(p[2])
    p[0] = p[1]

def p_element(p):
    """
    element : function
            | declaration Semi
    """
    p[0] = p[1]

def p_program_empty(p):
    """
    program : empty
```

```
    """
    p[0] = Program()
```

```
class Program(ListNode[Union["Function", "Declaration"]]):
    """
    AST root. It should have only one children before step9.
    """

    def __init__(self, *children: Union[Function, Declaration]) -> None:
        super().__init__("program", list(children))
```

二、语义分析

修改`frontend/typecheck/namer.py`中的`visitDeclaration`函数，增加对于`symbol`是全局变量的判断与处理：

```
    def visitDeclaration(self, decl: Declaration, ctx: ScopeStack) -> None:
        """
        1. Use ctx.lookup to find if a variable with the same name has been declared.
        2. If not, build a new VarSymbol, and put it into the current scope using
ctx.declare.
        3. Set the 'symbol' attribute of decl.
        4. If there is an initial value, visit it.
        """
        if ctx.isConflict(decl.ident.value): raise DecafUndefinedVarError(f"Variable
{decl.ident.value} already declared")
        # if ctx.lookup(decl.ident.value) is not None: raise
DecafUndefinedVarError(f"Variable {decl.ident.value} already declared")
        else:
            new_symbol = VarSymbol(decl.ident.value, decl.var_t)
            if ctx.isGlobalScope():
                new_symbol.isGlobal = True
                if decl.init_expr:
                    new_symbol.initValue = decl.init_expr.value
                elif new_symbol.type == INT:
                    decl.init_expr = IntLiteral(0)
            ctx.declare(new_symbol)
            decl.setattr("symbol", new_symbol)
            if decl.init_expr is not None:
                decl.init_expr.accept(self, ctx)
```

此外，`visitProgram`函数有个小bug：

将`for func in program.functions().values():`改为`for func in program.children:`，否则只能访问到函数，无法访问到全局变量。

三、中间代码生成

在`utils/tac/tacinstr.py`模仿`LoadImm4`类实现`LoadAddress`类、`LoadData`类和`StoreData`类，分别用来加载全局变量的地址以及加载和储存全局变量。

```
class LoadAddress(TACInstr):
    def __init__(self, symbol, dst: Temp) -> None:
        super().__init__(InstrKind.SEQ, [dst], [], None)
        self.symbol = symbol

    def __str__(self) -> str:
        return "%s = LOAD_SYMBOL %s" % (self.dsts[0], self.symbol.name)

    def accept(self, v: TACVisitor) -> None:
        v.visitLoadAddress(self)


class LoadData(TACInstr):
    def __init__(self, dst: Temp, base: Temp, offset: int):
        super().__init__(InstrKind.SEQ, [dst], [base], None)
        self.offset = offset

    def __str__(self) -> str:
        return "%s = LOAD %s %d" % (self.dsts[0], self.srcs[0], self.offset)

    def accept(self, v: TACVisitor) -> None:
        v.visitLoadData(self)

class StoreData(TACInstr):
    def __init__(self, src: Temp, base: Temp, offset: int):
        super().__init__(InstrKind.SEQ, [], [src, base], None)
        self.offset = offset

    def __str__(self) -> str:
        return "STORE %s %s, %d" % (self.srcs[0], self.srcs[1], self.offset)

    def accept(self, v: TACVisitor) -> None:
        return v.visitStoreData(self)
```

在`utils/tac/tacvisitor.py`里实现对应的`visitLoadAddress`函数、`visitLoadData`函数和`visitStoreData`函数：

```
    def visitLoadAddress(self, instr: LoadAddress) -> None:
        self.visitOther(instr)

    def visitLoadData(self, instr: LoadData) -> None:
        self.visitOther(instr)

    def visitStoreData(self, instr: StoreData) -> None:
        self.visitOther(instr)
```

在`frontend/tacgen/tacgen.py`的`TACFuncEmitter`类中实现对应的三个函数：

```
    def visitLoadAddress(self, symbol: VarSymbol) -> Temp:
        temp = self.freshTemp()
        self.func.add(LoadAddress(symbol, temp))
        return temp

    def visitLoadData(self, symbol: VarSymbol, offset: int = 0) -> Temp:
        address = self.visitLoadAddress(symbol)
        self.func.add(LoadData(address, address, offset))
        return address

    def visitStoreData(self, symbol: VarSymbol, value: Temp, offset: int = 0) -> None:
        address = self.visitLoadAddress(symbol)
        self.func.add(StoreData(value, address, offset))
```

此外，修改`TACGen`类的`visitIdentifier`函数和`visitAssignment`函数，增加对全局变量的判断与处理：

```
    def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter) -> None:
        """
        1. Set the 'val' attribute of ident as the temp variable of the 'symbol' attribute
 of ident.
        """
        symbol = ident.getattr("symbol")
        if symbol.isGlobal:
            ident.setattr("val", mv.visitLoadData(symbol))
        else:
            ident.setattr("val", symbol.temp)

    def visitAssignment(self, expr: Assignment, mv: TACFuncEmitter) -> None:
        """
        1. Visit the right hand side of expr, and get the temp variable of left hand side.
        2. Use mv.visitAssignment to emit an assignment instruction.
        3. Set the 'val' attribute of expr as the value of assignment instruction.
        """
        expr.rhs.accept(self, mv)
        rhs_symbol = expr.rhs.getattr("val")
        if expr.lhs.getattr("symbol").isGlobal:
            mv.visitStoreData(expr.lhs.getattr('symbol'), rhs_symbol)
        else:
            lhs_symbol = expr.lhs.getattr("symbol").temp
            rhs_temp = mv.visitAssignment(lhs_symbol, rhs_symbol)
        expr.setattr("val", rhs_symbol)
```

四、目的代码生成

在`utils/riscv.py`中模仿`LoadImm`类，增加`LoadAddress`类、`LoadData`类和`StoreData`类：

```
    class LoadAddress(BackendInstr):
        def __init__(self, symbol: str, dst: Temp) -> None:
            super().__init__(InstrKind.SEQ, [dst], [], None)
            self.symbol = symbol
        def __str__(self) -> str:
            return "la " + Riscv.FMT2.format(str(self.dsts[0]), self.symbol)

    class LoadData(BackendInstr):
        def __init__(self, dst: Temp, base: Temp, offset: int) -> None:
            super().__init__(InstrKind.SEQ, [dst], [base], None)
            self.offset = offset

        def __str__(self) -> str:
            assert -2048 <= self.offset <= 2047  # Riscv imm [11:0]
            return "lw " + Riscv.FMT_OFFSET.format(str(self.dsts[0]), str(self.offset),
str(self.srcs[0]))

    class StoreData(BackendInstr):
        def __init__(self, src: Temp, base: Temp, offset: int) -> None:
            super().__init__(InstrKind.SEQ, [], [src, base], None)
            self.offset = offset

        def __str__(self) -> str:
            assert -2048 <= self.offset <= 2047  # Riscv imm [11:0]
            return "sw " + Riscv.FMT_OFFSET.format(str(self.srcs[0]), str(self.offset),
str(self.srcs[1]))
```

在`backend/riscv/riscvasmemitter.py`的`RiscvAsmEmitter`类的初始化中增加全局变量`globalVars: dict[str, Declaration]`，并打印全局变量：

```
self.printer.println(".data")
for symbol, decl in globalVars.items():
    self.printer.printGlobalVar(symbol, decl.getattr("symbol").initValue)
self.printer.println("")
```

在`RiscvInstrSelector`类中模仿`visitLoadImm4`函数增加三个visit函数：

```
        def visitLoadAddress(self, instr: LoadAddress) -> None:
            self.seq.append(Riscv.LoadAddress(instr.symbol.name, instr.dsts[0]))

        def visitLoadData(self, instr: LoadData) -> None:
            self.seq.append(Riscv.LoadData(instr.dsts[0], instr.srcs[0], instr.offset))

        def visitStoreData(self, instr: StoreData) -> None:
            self.seq.append(Riscv.StoreData(instr.srcs[0], instr.srcs[1], instr.offset))
```

在`utils/tac/tacprog.py`中增加全局变量：

```
class TACProg:
    def __init__(self, funcs: list[TACFunc], vars: Dict[str, Declaration]) -> None:
        self.funcs = funcs
        self.vars = vars
```

并在`backend/asm.py`新建的`RiscvAsmEmitter`对象中增加全局变量`emitter = RiscvAsmEmitter(Riscv.AllocatableRegs, Riscv.CallerSaved, prog.vars)`

step11: 数组

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?

   答：声明VLA时，将当前的SP和VLA大小保存在栈上，移动SP。访问VLA时，计算偏移量的地址。当VLA离开作用域后，恢复SP的值。

> 提示：不能再像现在这样，在进入函数时统一给局部变量分配内存，在离开函数时统一释放内存。
>
> 你可以认为可变长度的数组的长度不大于0是未定义行为，不需要处理。

实验内容

一、语法词法分析

在`frontend/parser/ply_parser.py`中按照实验文档提供的规范修改相关部分：

```python
def p_declaration_array(p):
    """
    declaration : type Identifier dim_list
    """
    p[0] = Declaration(p[1], p[2], None, p[3])

def p_declaration_array_empty(p):
    """
    dim_list : empty
    """
    p[0] = []

def p_declaration_array_dim(p):
    """
    dim_list : dim_list LBracket Integer RBracket
    """
    p[1].append(p[3])
    p[0] = p[1]

def p_postfix_index_expr(p):
    """
    postfix : postfix LBracket expression RBracket
    """
    p[0] = IndexExpr(p[1], p[3])
```

同时，在`frontend/ast/tree.py`中`Declaration`类增加`init_dim`参数：`init_dim: Optional[list[IntLiteral]] = None`

在`Function`类增加`arrays`参数：`self.arrays = {}`

增加数组的索引运算节点：

```python
class IndexExpr(Expression):
    def __init__(self, base: Expression, index: Expression) -> None:
        super().__init__("index_expr")
        self.base = base
        self.index = index

    def __getitem__(self, key: int) -> Node:
        return (self.base, self.index)[key]

    def __len__(self) -> int:
        return 2

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitIndexExpr(self, ctx)
```

二、语义分析

`frontend/typecheck/namer.py`：

主要在`visitDeclaration`函数中增加对`decl`为数组的处理，

```python
if decl.init_dim:
    for dim in decl.init_dim:
        if dim.value <= 0:
            raise DecafBadArraySizeError()
    decl_type = ArrayType.multidim(decl.var_t.type, *[dim.value for dim in decl.init_dim])
    new_symbol = VarSymbol(decl.ident.value, decl_type)
    self.arrays[decl.ident.value] = new_symbol
else:
    new_symbol = VarSymbol(decl.ident.value, decl.var_t.type)
```

以及增加对数组索引的处理函数`visitIndexExpr`函数：

```python
    def visitIndexExpr(self, expr: IndexExpr, ctx: ScopeStack) -> None:
        expr.base.accept(self, ctx)
        expr.index.accept(self, ctx)
        if isinstance(expr.base, Identifier): #有个递归
            expr.setattr('type', expr.base.getattr('symbol').type.indexed)
        else:
            expr.setattr('type', expr.base.getattr('type').indexed)
```

还要注意一些可能导致本应报错的failcases正常运行的边界情况，主要注意对类型的排查，如`visitAssignment`函数：

```
    def visitAssignment(self, expr: Assignment, ctx: ScopeStack) -> None:
        expr.lhs.accept(self, ctx)
        expr.rhs.accept(self, ctx)
        # breakpoint()
        if expr.lhs.getattr('type') != expr.rhs.getattr('type'):
            raise DecafTypeMismatchError()
        expr.setattr('type', expr.lhs.getattr('type'))
```

三、中间代码生成

`frontend/tacgen/tacgen.py`：

在`FACFuncEmitter`类中增加按照地址读与储存数组元素的函数：

```
    def visitLoadArrayAddress(self, address: Temp):
        dst = self.freshTemp()
        self.func.add(LoadData(dst, address, 0))
        return dst

    def visitStoreArray(self, value: Temp, address: Temp):
        self.func.add(StoreData(value, address, 0))
```

此外，修改`TACGen`类中的`visitIdentifier`函数和`visitAssignment`函数：增加对数组的处理

```
    def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter) -> None:
        """
        1. Set the 'val' attribute of ident as the temp variable of the 'symbol' attribute of ident.
        """
        symbol = ident.getattr("symbol")
        if isinstance(symbol.type, ArrayType):
            ident.setattr("addr", mv.visitLoadAddress(symbol))
        elif symbol.isGlobal:
            ident.setattr("val", mv.visitLoadData(symbol))
        else:
            ident.setattr("val", symbol.temp)

    def visitAssignment(self, expr: Assignment, mv: TACFuncEmitter) -> None:
        """
        1. Visit the right hand side of expr, and get the temp variable of left hand side.
        2. Use mv.visitAssignment to emit an assignment instruction.
        3. Set the 'val' attribute of expr as the value of assignment instruction.
        """
        expr.rhs.accept(self, mv)
        rhs_symbol = expr.rhs.getattr("val")
        if isinstance(expr.lhs, IndexExpr):
            expr.lhs.setattr('slice', True)
            expr.lhs.accept(self, mv)
            mv.visitStoreArray(expr.rhs.getattr('val'), expr.lhs.getattr('addr'))
        elif expr.lhs.getattr("symbol").isGlobal:
            mv.visitStoreData(expr.lhs.getattr('symbol'), rhs_symbol)
        else:
            lhs_symbol = expr.lhs.getattr("symbol").temp
            rhs_temp = mv.visitAssignment(lhs_symbol, rhs_symbol)
        expr.setattr("val", rhs_symbol)
```

增加`visitIndexExpr`函数：采用递归的方式计算数组元素的地址

```
    def visitIndexExpr(self, expr: IndexExpr, mv: TACFuncEmitter) -> None:
        expr.base.setattr('slice', True)
        expr.base.accept(self, mv)
        expr.index.accept(self, mv)
        # 递归计算偏移量
        addr = mv.visitLoad(expr.getattr('type').size)
        mv.visitBinarySelf(tacop.TacBinaryOp.MUL, addr, expr.index.getattr('val'))
        mv.visitBinarySelf(tacop.TacBinaryOp.ADD, addr, expr.base.getattr('addr'))
        expr.setattr('addr', addr)
        if not expr.getattr('slice'):
            expr.setattr('val', mv.visitLoadArrayAddress(addr))
```

四、目标代码生成

在`utils/riscv.py`中增加`ImmAdd`类，用于加载保存在栈上的局部数组：

```
    class ImmAdd(BackendInstr):
        def __init__(self, dst: Temp, src: Temp, value: int) -> None:
            super().__init__(InstrKind.SEQ, [dst], [src], None)
            self.value = value

        def __str__(self) -> str:
            assert -2048 <= self.value <= 2047  # Riscv imm [11:0]
            return "addi " + Riscv.FMT3.format(
                str(self.dsts[0]), str(self.srcs[0]), str(self.value)
            )
```

在`backend/subroutineinfo.py`中增加`arrays`参数，并计算数组的偏移量以及栈帧大小：

```
    def __init__(self, funcLabel: FuncLabel, numArgs: int, arrays: Dict[str, VarSymbol]) ->
None:
        self.funcLabel = funcLabel
        self.numArgs = numArgs
        self.arrays = arrays

        self.offsets: Dict[str, int] = {}
        self.size = 0

        for name, symbol in self.arrays.items():
            self.offsets[name] = self.size
            self.size += symbol.type.size
```

在`utils/asmcodeprinter.py`中增加对全局数组的打印函数：

```
    def printGlobalArray(self, symbol: str, value: int):
        self.buffer += ".globl " + symbol + "\n" + symbol + ":\n" + self.INDENTS + ".space
" + str(value) + "\n"
```

在`backend/riscv/riscvasmemitter.py`的`RiscvAsmEmitter`类的初始化中增加对全局数组的打印：

```
        self.printer.println(".data")
        for symbol, decl in globalVars.items():
            if not decl.init_dim:
                self.printer.printGlobalVar(symbol, decl.getattr("symbol").initValue)
        self.printer.println("")

        self.printer.println(".bss")
        for symbol, decl in globalVars.items():
            if decl.init_dim and not decl.init_expr:
                self.printer.printGlobalArray(symbol, decl.getattr("symbol").type.size)
        self.printer.println("")
```

`RiscvInstrSelector`类的`visitLoadAddress`函数中增加对数组的处理：

```
        def visitLoadAddress(self, instr: LoadAddress) -> None:
            if instr.symbol.isGlobal:
                self.seq.append(Riscv.LoadAddress(instr.symbol.name, instr.dsts[0]))
            else:
                self.seq.append(Riscv.ImmAdd(instr.dsts[0], Riscv.SP,
self.info.offsets[instr.symbol.name]))
```

step12: 为数组添加更多支持

思考题

1. 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？

   答：因为C/C++数组传参时编译器会把参数解析成指向数组的指针，所以无论数组第一维大小是否标明，都只传递了一个指向内层数组的指针，此时第一维大小没有什么用处。

实验内容

一、词法语法分析

按照step12的语法规范修改`frontend/parser/ply_parser.py`，在语法树中增加`InitList`类，作为数组的初始化列表，便于后续的初始化与传参：

```
class InitList(Node):
    def __init__(self, init_list: List[IntLiteral]):
        super().__init__("init_list")
        self.init_list = init_list
        self.value = [item.value for item in init_list]

    def __getitem__(self, item):
        return self.init_list[item]

    def __len__(self):
        return len(self.init_list)

    def accept(self, v: Visitor[T, U], ctx: T) -> None:
        pass
```

此外，在`Parameter`类中增加`init_dim`参数，在`Function`类中增加`p_arrays`参数。

二、语义分析

在初始化中增加`p_arrays`参数。修改`visitFunction`函数：增加对函数参数数组的声明

```
        ctx.addScope(func_scope)
        self.arrays = {}
        self.p_arrays = []
        func.params.accept(self, ctx)
        # func.body.accept(self, ctx, func.params)
        for index, param in enumerate(func.params.children):
            if isinstance(param.ident.getattr('type'), ArrayType):
                self.p_arrays.append(param.getattr('symbol'))
        for child in func.body.children:
            child.accept(self, ctx)
        func.arrays = self.arrays
        self.arrays = {}
        func.p_arrays = self.p_arrays
        self.p_arrays = []
        ctx.popScope()
```

此外，修改`visitParameter`函数：增加对参数数组的处理，尤其注意数组第一维为空的情况

```
    def visitParameter(self, param: Parameter, ctx: ScopeStack) -> None:
        if ctx.lookup(param.ident.value):
            raise DecafDeclConflictError(param.ident.value)
        if param.init_dim:
            for index, dim in enumerate(param.init_dim):
                if index == 0 and dim is NULL: # 函数参数数组的第一维为空的情况
                    continue
                if dim.value <= 0:
                    raise DecafBadArraySizeError()
            decl_type = ArrayType.multidim(param.var_t.type, *[dim.value if dim else None
 for dim in param.init_dim])
            newSymbol = VarSymbol(param.ident.value, decl_type)
        else:
            newSymbol = VarSymbol(param.ident.value, param.var_t.type)
        ctx.declare(newSymbol)
        param.setattr("symbol", newSymbol)
        param.ident.setattr('type', newSymbol.type)
```

三、中间代码生成

在`utils/tac/tacfunc.py`的初始化中增加`p_arrays`参数：`p_arrays: Dict[int, VarSymbol]`

`frontend/tacgen/tacgen.py`中：

在`TACFuncEmitter`类的初始化中增加`p_arrays`参数。

在`TACGen`类中：

修改`visitIdentifier`函数：增加对参数数组的处理

```
    def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter) -> None:
        """
        1. Set the 'val' attribute of ident as the temp variable of the 'symbol' attribute
 of ident.
        """
        symbol = ident.getattr("symbol")
        if isinstance(symbol.type, ArrayType):
            if symbol.isGlobal or symbol not in mv.func.p_arrays: #对全局数组或局部数组
                ident.setattr("addr", mv.visitLoadAddress(symbol))
            else: #对参数数组
                ident.setattr("addr", symbol.temp)
            ident.setattr('val', ident.getattr('addr'))
        elif symbol.isGlobal:
            ident.setattr("val", mv.visitLoadData(symbol))
        else:
            ident.setattr("val", symbol.temp)
```

修改`visitDeclaration`函数：同样地，增加对参数数组的处理。参考实验文档，在main.py中增加`memsetFunc`，加在代码的最前面，作为内置函数，实现数组的清零。

```
    def visitDeclaration(self, decl: Declaration, mv: TACFuncEmitter) -> None:
        """
        1. Get the 'symbol' attribute of decl.
        2. Use mv.freshTemp to get a new temp variable for this symbol.
        3. If the declaration has an initial value, use mv.visitAssignment to set it.
        """
        symbol = decl.getattr("symbol")
        new_temp = mv.freshTemp()
        symbol.temp = new_temp
        if decl.init_expr is not NULL:
            if isinstance(decl.init_expr, InitList):
                addr = mv.visitLoadAddress(symbol)
                size = symbol.type.full_indexed.size
                interval = mv.visitLoad(size)
                mv.visitParam(addr)
                mv.visitParam(mv.visitLoad(symbol.type.size // size))
                mv.visitCall(FuncLabel("fill_array"))
                for value in decl.init_expr.value:
                    mv.visitStoreArray(mv.visitLoad(value), addr)
                    mv.visitBinarySelf(tacop.TacBinaryOp.ADD, addr, interval)
            else:
                init_temp = decl.init_expr.accept(self, mv)
                decl.setattr("val", mv.visitAssignment(new_temp,
decl.init_expr.getattr("val")))
```

```
memsetFunc = "int fill_array(int array[], int cnt){\n    for(int i = 0; i < cnt; i = i + 1)
array[i] = 0;\n    return 0;\n}"

def step_parse(args: argparse.Namespace):
    code = str(memsetFunc) + "\n" + readCode(args.input)
    ...
```

四、目标代码生成

只需增加打印即可。

在`utils/asmcodeprinter.py`中增加对带初始化列表的全局数组的打印函数：

```
    def printGlobalInitArray(self, symbol: str, vals: list[int]):
        self.buffer += ".globl " + symbol + "\n" + symbol + ":\n"
        for v in vals:
            self.buffer += self.INDENTS + ".word " + str(v) + "\n"
```

在`backend/riscv/riscvasmemitter.py`的`RiscvAsmEmitter`类的初始化中增加对带初始化列表的全局数组的打印：

```
        self.printer.println(".data")
        for symbol, decl in globalVars.items():
            if not decl.init_dim:
                self.printer.printGlobalVar(symbol, decl.getattr("symbol").initValue)
            if decl.init_dim and decl.init_expr:
                self.printer.printGlobalInitArray(symbol, decl.getattr("symbol").initValue)
        self.printer.println("")
```

Honor Code

在对边界情况的报错中（即failcases）参考了https://github.com/chengsx21/MiniDecaf/tree/main中比对类型的处理方式。

在全局变量的目标代码生成中参考了https://github.com/chengsx21/MiniDecaf/tree/main的部分内容。