第二遍扫描带标注的 AST 以建立 TAC 语句：

- ArrayList<VTable> //VTable 列表

  ArrayList<Functy> //Function 列表

- **扫描到变量结点**

```
@Override
public void visitVarDef(Tree.VarDef varDef) {
    //为局部变量绑定临时变量
    if (varDef.symbol.isLocalVar()) {
        Temp t = Temp.createTempI4();
        t.sym = varDef.symbol;
        varDef.symbol.setTemp(t);
    }
}
```

- **扫描到函数结点**

```java
@Override
public void visitMethodDef(Tree.MethodDef funcDefn) {

    //获取当前非静态函数this参数的临时变量
    if (!funcDefn.statik) {
        currentThis = ((Variable) funcDefn.symbol.getAssociatedScope
                .lookup("this")).getTemp();
    }

    //开始函数体的TAC代码生成
    tr.beginFunc(funcDefn.symbol);

    //遍历函数体的语句块并执行visit方法
    funcDefn.body.accept(this);

    //结束函数体的TAC代码生成
    tr.endFunc();
    currentThis = null;
}
```

- **开始函数的生成**

```java
public void beginFunc(Function func) {
    currentFuncty = func.getFuncty();

    //生成memo'XXX': 专为TAC使用的指导命令
    currentFuncty.paramMemo = memoOf(func);

    //指明函数的入口标号
    genMark(func.getFuncty().label);
}
```

```
Class Computer{
  Int cpu;
  void Crash(int numTimes);
}
```

Memo ‘_T0:4_T1:8’ 的含义：

临时量_T0：与函数的形式参数 this 对应，_T0 的偏移量固定是 4；

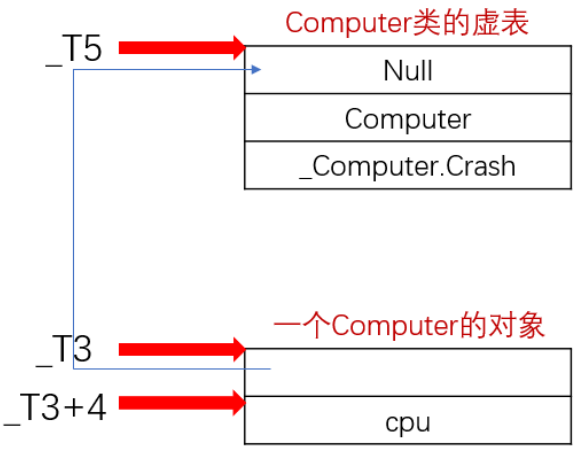
临时量_T1：与函数的形式参数 numTimes 对应，_T1 的偏移量固定是 8；

- **结束函数的生成**

```
public void endFunc() {
    //加入已生成函数的列表中
    funcs.add(currentFuncty);
    currentFuncty = null;
}
```

- **_Computer_new 函数：初始化一个 Computer 对象**

```
class Computer {
   int cpu;
   void Crash(int numTimes) {
      int i;
      for (i = 0; i < numTimes; i = i + 1)
         Print("sad\n");
   }
}
```

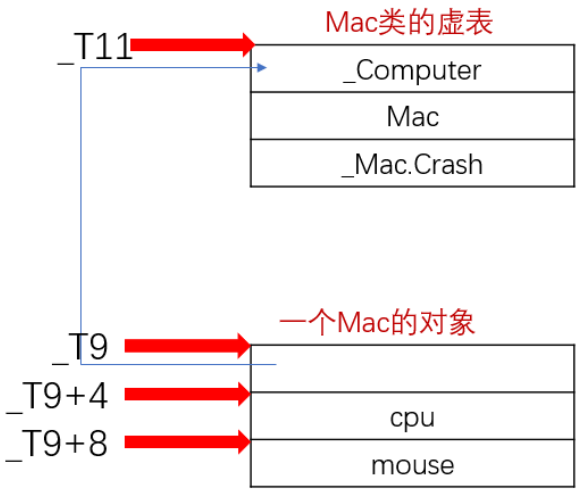



```
FUNCTION(_Computer_New) {
memo ''
_Computer_New:
   _T2 = 8
   parm _T2
   _T3 =  call _Alloc
   _T4 = 0
   *(_T3 + 4) = _T4
   _T5 = VTBL <_Computer>
   *(_T3 + 0) = _T5
   return _T3
}
```

- _Computer_Crash 函数

```
FUNCTION(_Computer.Crash) {
memo '_T0:4 _T1:8'
_Computer.Crash:
    _T16 = 0
    _T15 = _T16
    branch _L14
_L15:
    _T17 = 1
    _T18 = (_T15 + _T17)
    _T15 = _T18
_L14:
    _T19 = (_T15 < _T1)
    if (_T19 == 0) branch _L16
    _T20 = "sad\n"
    parm _T20
    call _PrintString
    branch _L15
_L16:
}
```

- _Mac_New：子类的初始化函数




```
FUNCTION(_Mac_New) {
memo ''
_Mac_New:
    _T8 = 12
    parm _T8
    _T9 =  call _Alloc
    _T10 = 0
    *(_T9 + 4) = _T10
    *(_T9 + 8) = _T10
    _T11 = VTBL <_Mac>
    *(_T9 + 0) = _T11
    return _T9
}
```

- **静态方法直接得到地址直接调用：创建 Main 类实例并且调用 main 函数的过程将自动成为一个 Funcky 对象，并将其作为整个程序的入口。**

```
class Main {
    static void main() {
        class Mac powerbook;
        powerbook = new Mac();
        powerbook.Crash(2);
    }
}
```

```
FUNCTION(main) {
memo ''
main:
    _T23 =  call _Mac_New
    _T22 = _T23
    _T24 = 2
    parm _T22
    parm _T24
    _T25 = *(_T22 + 0)
    _T26 = *(_T25 + 8)
    call _T26
}
```

_T22: 创建的 Mac 对象

_T24: 存储形参：2

_T25: 指向 Mac 类的 VTable

_T26: 指向 Mac 类的 Crash 函数