```
public class MemoryAllocation extends DefaultVisitor {

  public void process(AST ast) {
    ast.accept(this, null);
  }
  ;

  @Override
  public Object visit(Program program, Object param) {
    super.visit(program, param);

    int currentAddress = 0;

    for (var definition : program.getDefinitions()) {
      definition.accept(this, param);
      if (definition instanceof VarDefinition) {
        VarDefinition def = (VarDefinition) definition;
        def.setAddress(currentAddress);
        currentAddress += def.getType().getSize();
      }
    }
    return null;
  }

  @Override
  public Object visit(StructDefinition structDefinition, Object param)
{
    super.visit(structDefinition, param);

    int currentAddress = 0;

    for (FieldDefinition field :
structDefinition.getFieldDefinitions()) {
      field.setAddress(currentAddress);
      currentAddress += field.getType().getSize();
```

```
    }

    structDefinition.setAddress(currentAddress);

    return null;
  }

  @Override
  public Object visit(FunctionDefinition functionDefinition, Object
param) {
    super.visit(functionDefinition, param);

    int definitionsAddress = 0;

    for (VarDefinition def : functionDefinition.getLocals()) {
      definitionsAddress -= def.getType().getSize();
      def.setAddress(definitionsAddress);
    }

    // Se reserva memoria para la direcci n de retorno

    int paramsAddress = 4;

    List<VarDefinition> params = functionDefinition.getParameters();
    for (int i = params.size() - 1; i >= 0; i--) {
      VarDefinition param_ = params.get(i);
      param_.setAddress(paramsAddress);
      paramsAddress += param_.getType().getSize();
    }

    return null;
  }
}
```