1 Things to do

1. Flatten the code to three address code.
2. Rename variables (make every variable unique the entire program.(for inline function))
3. Determine stack offset of each variable.
4. Translation.
   a. prologue
   b. block translation
   c. global variable translation

2 General Structure of Low Level IR (Optional)

2.1 Three Address Code Format

- `x := y op z`
- `x := op z`
- `x := y`
- `goto L`
- `if x rel_op y goto L`
- `param p, n //load p as the nth param`
- `(x := ) call func //assignment is optional`
- `ret x //return a variable`

2.2 Block Structure

```
case class Block() {
    def label:String //block label
    val stmt: Vector[Statement]
}
```

The whole program is a Vector of Blocks.

3 A Concreate Example

3.1 Original Code

```
int foo(int a) {
    return a + a / 10;
}
```

```
void bar(int x) {
    int a, b, z;
    a = 10;
    b = 5;
    z = a * 2 + b;
    a = foo(z);
}
void main() {
    bar(4);
}
```

3.2 Flattened Code And Rename

```
int foo(int foo_a) {
    t1 = foo_a / 10;
    t2 = foo_a + t1;
    return t2;
}
void bar(int bar_x) {
    int bar_a, bar_b, bar_z;
    bar_a = 10; // temp store any value other than direct assignment.
    bar_b = 5; // or we can do store everything, and let optimizer do clean
up.
    t3 = bar_a * 2;
    bar_z = t3 + bar_b;
    t4 = bar_z;
    bar_a = foo(bar_z);
}
void main() {
    bar(4);
}
```

3.3 In 3 Address Code

```
foo:
    t1 := foo_a / 10;
    t2 := foo_a + t1;
    ret t2;
bar:
    bar_a := 10;
    bar_b := 5;
    t3 := bar_a * 2;
    bar_z := t3 + bar_b;
    t4 := bar_z;
    t5 := bar_x;
    param bar_z, 1;
    call foo
    bar_a := foo;
    bar_x := t5;
main:
    param 4, 1
    call bar
```