Algorithm

- We start with a value context sensitive flow sensitive analysis over the whole program.
- We use the points-to information at each program point to determine a set of live variables(Var Set) and the set of live objects(Ref Set).
- We start Liveness analysis from end of main with empty Var Set and Ref Set and propagaate backwards.

Flow Sensitive Analysis

- Let the abstract stack be represented as S
- Let the abstract heap be represented as H

Rules for Var Set and Ref Set

- Var Set

Statement	Use	Def
x = new A()	-	x
x = y	y	x
x = y.f	y	x
x.f = y	x, y	-

- Ref Set

Statement	Use
x = new A()	-
x = y	S(y)
x = y.f	($\cup$ [H(o,f)] for all o $\in$ S(y)) $\cup$ S(y)
x.f = y	S(y)

- Def Set is only relevant for x = new A() statements.
- Rule for both sets Live_before = (Live_after - Def) $\cup$ Use

Function Call

- For encountering a function call, we first create a mapping from the formal parameters to the actual parameters.

```
main()
{
    .
    .
    foo(a,b,c);         //Call 1
        // Var Set = {a,b,c} Ref Set = {O1, O2, O3}
    .
    .
    foo(a,d,c);         //Call 2
        // Var Set = {a,d} Ref Set = {O1, O2}
    .
    .
}

foo(int x, int y, int z)
{
    .
    .
}
```

- We create a mapping and start the analysis from the bottom of foo() call with given Var and Ref Set.

```
foo(int x, int y, int z)    // Call 1 Analysis
{
    .
    .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
}

foo(int x, int y, int z)    // Call 2 Analysis
{
    .
    .
        // Var Set = {x,y} Ref Set = {O1, O2}
}
```

- On returning, we use the same mapping from the formal parameters to the actual parameters and start the analysis from above the foo() call.

```
main()
{
    .
    .
    foo(a,b,c);         //Call 1
        // Var Set = {a,b,c} Ref Set = {O1, O2, O3}
    .
    .
    foo(a,d,c);         //Call 2
        // Var Set = {a,d} Ref Set = {O1, O2}
    .
    .
}

foo(int x, int y, int z)
{
    .
    .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
    .
    .
}

foo(int x, int y, int z)
{
    .
    .
        // Var Set = {x,y} Ref Set = {O1, O2}
    .
    .
}
```

- Thus we have value context and live context sensitive analysis for each function call.

If-else Cases

- Let us take the general case of encountering multiple paths L1, L2, L3 .., Ln.
- We perform liveness individually on each path and take the union of the Var Set and Ref Set at the top.
- Let the Ref Set be {O1, O2, O3} before encountering 3 possible control flow paths.

```
    // Ref Set = {O1, O2, O3, O4, O5, O6, O7} Union of Ref Sets of all
paths
    if(cond1)       // L1
    {
        // Ref Set = {O1, O2, O3, O4}
        .
        .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
    }
    else if(cond2)  // L2
    {
        // Ref Set = {O1, O2, O3, O5}
        .
        .
        // Var Set = {x,y} Ref Set = {O1, O2}
    }
    else            // L3
    {
        // Ref Set = {O1, O2, O3, O6, O7}
        .
        .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
    }
    // Ref Set = {O1, O2, O3}
```

- We can see that O4, O5, O6, O7 are dead objects and can be garbage collected.
- Deaths detected by live analysis on individual paths are as follows:
  - Along L1, O4 death is detected
  - Along L2, O5 death is detected
  - Along L3, O6, O7 death is detected
- We must add the death of the other objects for each path at the start of control flow path.

```
    // Ref Set = {O1, O2, O3, O4, O5, O6, O7} Union of Ref Sets of all
paths
    if(cond1)       // L1
    {
        // Ref Set = {O1, O2, O3, O4}
        // Kill O5, O6, O7
        .
        .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
    }
    else if(cond2)  // L2
    {
        // Ref Set = {O1, O2, O3, O5}

        // Kill O4, O6, O7
        .
        .
        // Var Set = {x,y} Ref Set = {O1, O2}
    }
    else            // L3
    {
        // Ref Set = {O1, O2, O3, O6, O7}

        // Kill O4, O5
        .
        .
        // Var Set = {x,y,z} Ref Set = {O1, O2, O3}
    }
    // Ref Set = {O1, O2, O3}
```

While Loop Case

- When we encounter while loop we have to run multiple iterations of liveness analysis.
- First iteration of liveness analysis determines what object are last used during the loop and are considered dead after the loop.
- After the first iteration, we must use the Var and Ref Set at the top of the loop and run again starting from the bottom of the loop.
- We repeat the process until there in no change in Var and Ref set at all program points in loop and the object deaths we determine at this final iteration are the object that can be killed during the loop.

```
// x.f points to O4

while(cond)
{
    .
    y = new A(); // O5
    .
    .
    Use(x.f)
    .
    .
    Use(y)
    .
    .
}
// Ref Set = {O1, O2, O3}

Iteration 1:

while(cond)
{
    .
    // Ref Set = {O1, O2, O3, O4}
    y = new A();
    .
    .
    // Ref Set = {O1, O2, O3, O4, O5}
    Use(x.f)
    // O4 kill
    .
    .
    // Ref Set = {O1, O2, O3, O5}
    Use(y)
    // O5 kill
    .
    .
}
// Ref Set = {O1, O2, O3}

Final Live Set:
while(cond)
{
    .
    // Ref Set = {O1, O2, O3, O4}
    y = new A();
    .
    .
    // Ref Set = {O1, O2, O3, O4, O5}
    Use(x.f)
    .
    .
    // Ref Set = {O1, O2, O3, O4, O5}
    Use(y)
    // O5 kill
    .
    .
    // Ref Set = {O1, O2, O3, O4}
}
```

- From the analysis above we can place the kill points as follows:

```

    // x.f points to O4

    while(cond)
    {
        .
        y = new A(); // O5
        .
        .
        Use(x.f)
        .
        .
        Use(y)
        // O5 kill
        .
        .
    }
    // O4 kill

```