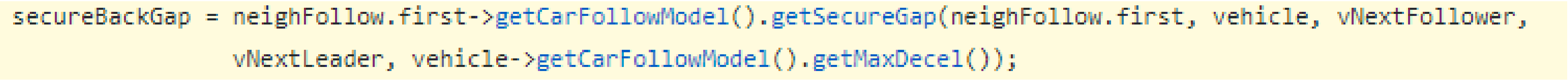
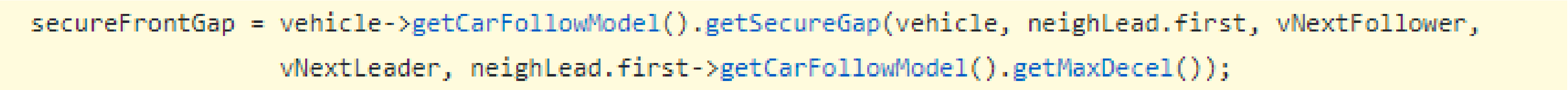
Calculating Required Gap – Used Functions

Below functions were used to calculate the required gap

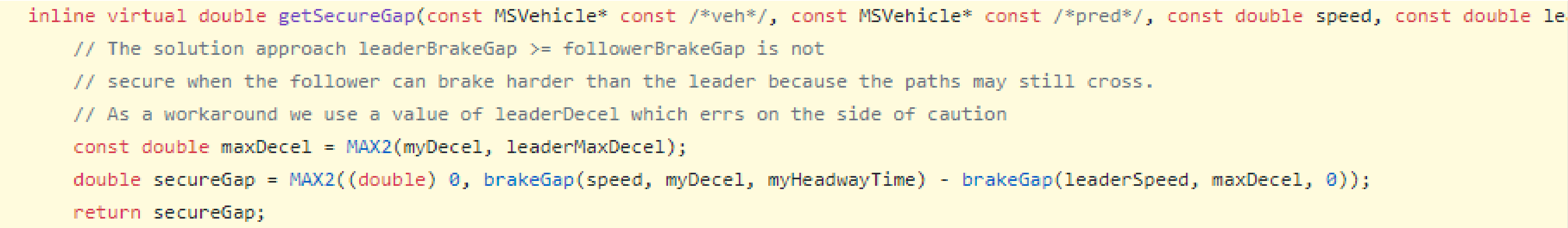
➢ Required Gap
= secureBackGap + followerMinGap + subjectLength + subjectMinGap + secureFrontGap



```
secureBackGap = neighFollow.first->getCarFollowModel().getSecureGap(neighFollow.first, vehicle, vNextFollower,
                vNextLeader, vehicle->getCarFollowModel().getMaxDecel());
```

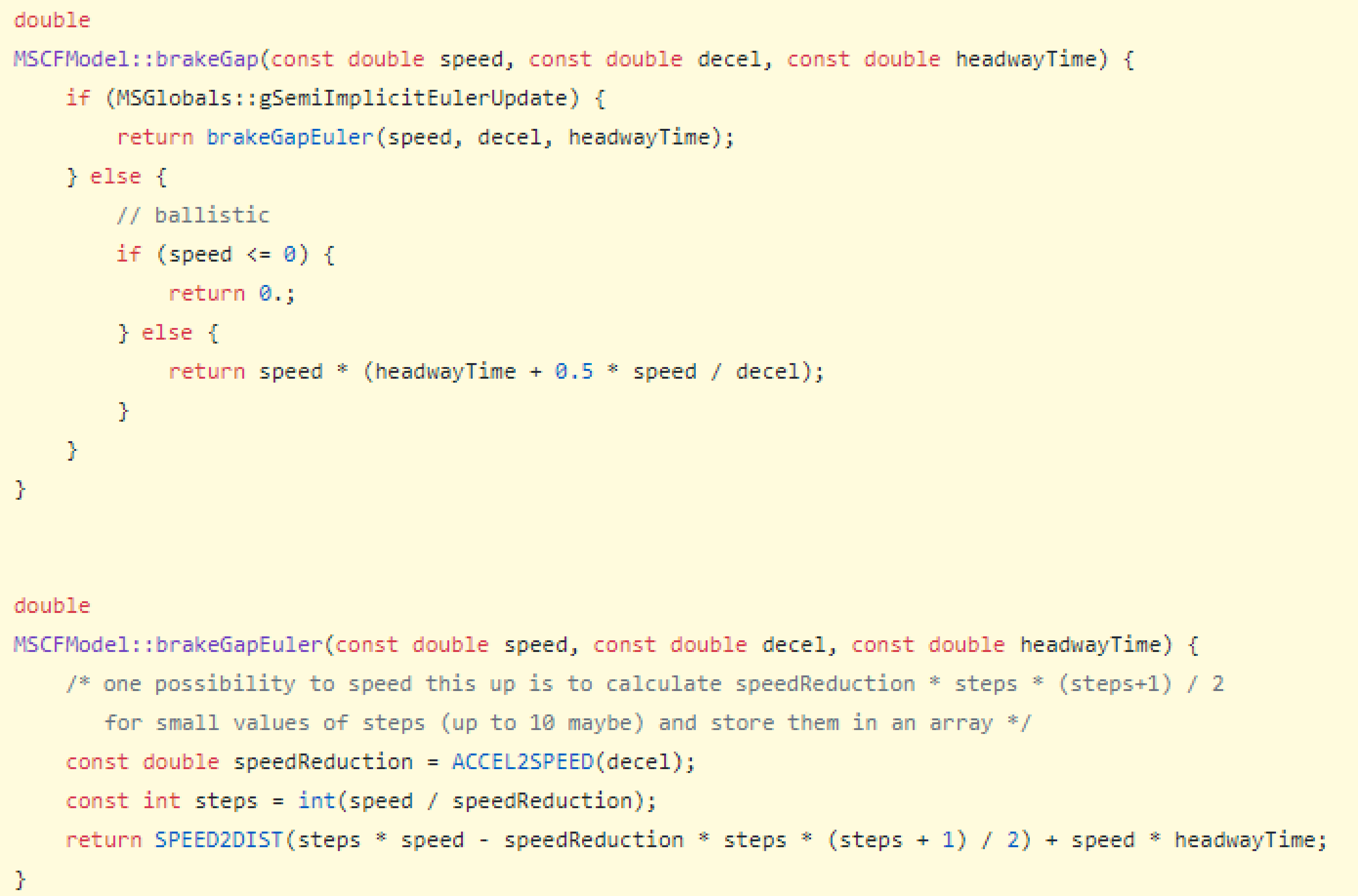


```
secureFrontGap = vehicle->getCarFollowModel().getSecureGap(vehicle, neighLead.first, vNextFollower,
                 vNextLeader, neighLead.first->getCarFollowModel().getMaxDecel());
```



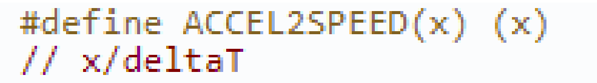
```
inline virtual double getSecureGap(const MSVehicle* const /*veh*/, const MSVehicle* const /*pred*/, const double speed, const double le
    // The solution approach leaderBrakeGap >= followerBrakeGap is not
    // secure when the follower can brake harder than the leader because the paths may still cross.
    // As a workaround we use a value of leaderDecel which errs on the side of caution
    const double maxDecel = MAX2(myDecel, leaderMaxDecel);
    double secureGap = MAX2((double) 0, brakeGap(speed, myDecel, myHeadwayTime) - brakeGap(leaderSpeed, maxDecel, 0));
    return secureGap;
```

Calculating Required Gap – Used Functions

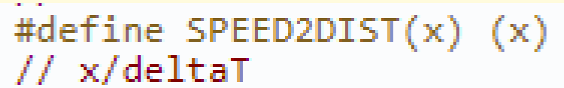


```
double
MSCFModel::brakeGap(const double speed, const double decel, const double headwayTime) {
    if (MSGlobals::gSemiImplicitEulerUpdate) {
        return brakeGapEuler(speed, decel, headwayTime);
    } else {
        // ballistic
        if (speed <= 0) {
            return 0.;
        } else {
            return speed * (headwayTime + 0.5 * speed / decel);
        }
    }
}


double
MSCFModel::brakeGapEuler(const double speed, const double decel, const double headwayTime) {
    /* one possibility to speed this up is to calculate speedReduction * steps * (steps+1) / 2
       for small values of steps (up to 10 maybe) and store them in an array */
    const double speedReduction = ACCEL2SPEED(decel);
    const int steps = int(speed / speedReduction);
    return SPEED2DIST(steps * speed - speedReduction * steps * (steps + 1) / 2) + speed * headwayTime;
}
```

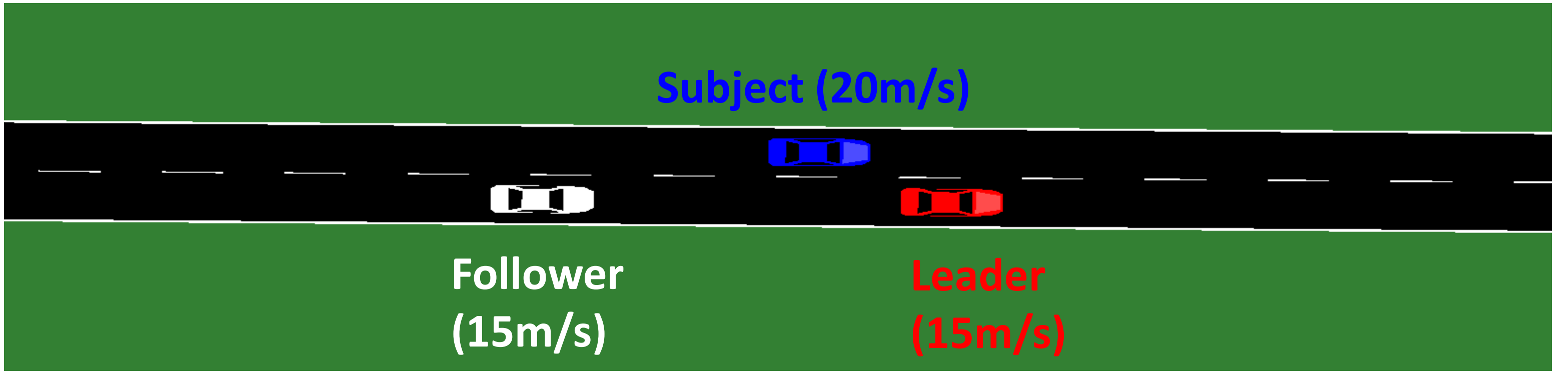


```
#define ACCEL2SPEED(x) (x)
// x/deltaT
```



```
#define SPEED2DIST(x) (x)
// x/deltaT
```

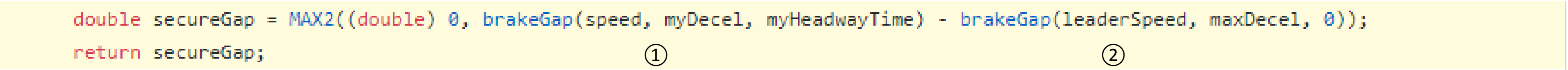
Calculating Required Gap – Scenario Settings



	Follower	Subject	Leader
MinGap	2.5	0	2.5
Tau	1	0.1	2
MaxSpeed	15	20	15
Current Speed	15	20	15
lcAssertive	Default	1	Default
Decel	4.5	4.5	4.5
CarFollowingModel	Default (Krauss)		
Lane-Changing Model	Default (LC2013)		
Time-Step Length	Default		

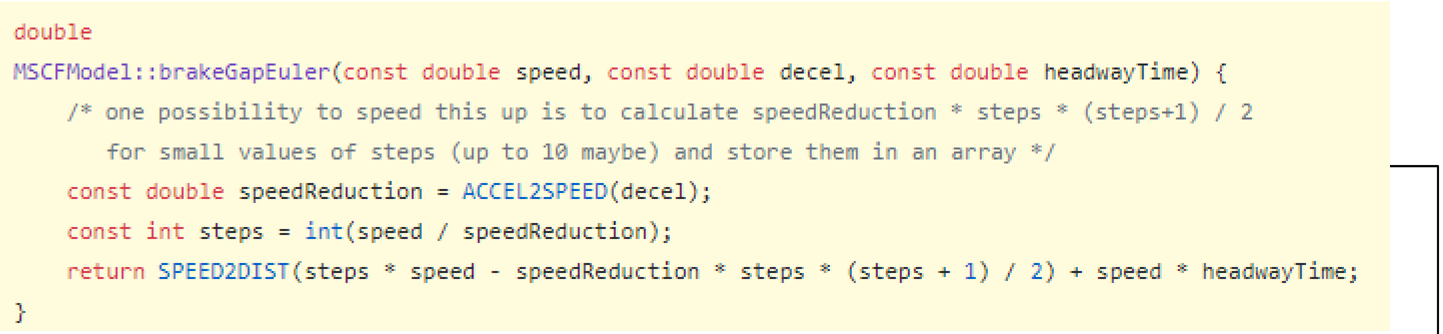
Calculating Required Gap – SecureBackGap

Starting with SecureBackGap



```
    double secureGap = MAX2((double) 0, brakeGap(speed, myDecel, myHeadwayTime) - brakeGap(leaderSpeed, maxDecel, 0));
    return secureGap;
```

① ②



```
double
MSCFModel::brakeGapEuler(const double speed, const double decel, const double headwayTime) {
    /* one possibility to speed this up is to calculate speedReduction * steps * (steps+1) / 2
       for small values of steps (up to 10 maybe) and store them in an array */
    const double speedReduction = ACCEL2SPEED(decel);
    const int steps = int(speed / speedReduction);
    return SPEED2DIST(steps * speed - speedReduction * steps * (steps + 1) / 2) + speed * headwayTime;
}
```

① speedReduction = ACCEL2SPEED(4.5) = $\frac{x}{deltaT} = \frac{4.5}{1}$ = 4.5

steps = $\frac{speed}{speedReduction}$ = $\frac{15}{4.5} = 3.333$

SPEED2DIST(3.333 * 15 – 4.5*3.333*(3.333+1) / 2) = $\frac{x}{deltaT} = \frac{17.5}{1}$ = 17.5

17.5 + speed * headwayTime = 17.5 + 15* 1 (Follower's Tau) = 32.5m

② speedReduction = ACCEL2SPEED(4.5) = $\frac{x}{deltaT} = \frac{4.5}{1}$ = 4.5

steps = $\frac{speed}{speedReduction}$ = $\frac{15}{4.5} = 3.333$

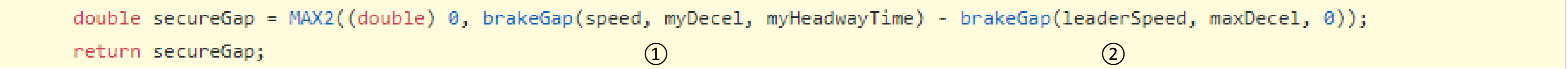
SPEED2DIST(3.333 * 15 – 4.5*3.333*(3.333+1) / 2) = $\frac{x}{deltaT} = \frac{17.5}{1}$ = 17.5

17.5 + speed * headwayTime = 17.5 + 15* 0 = 17.5m

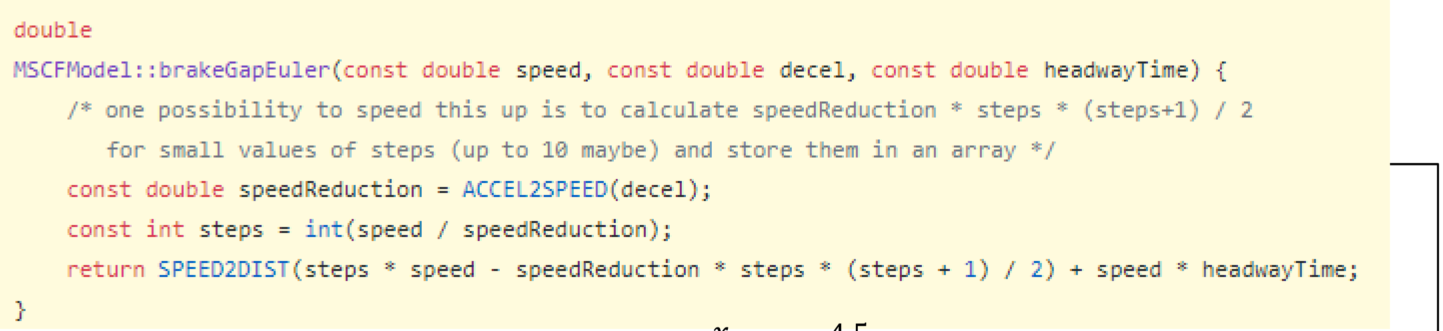
secureGap = SecureBackGap = MAX2(0, 32.5 – 17.5) = 15m

Calculating Required Gap – SecureFrontGap

SecureFrontGap



```
double secureGap = MAX2((double) 0, brakeGap(speed, myDecel, myHeadwayTime) - brakeGap(leaderSpeed, maxDecel, 0));
return secureGap;
```



```
double
MSCFModel::brakeGapEuler(const double speed, const double decel, const double headwayTime) {
    /* one possibility to speed this up is to calculate speedReduction * steps * (steps+1) / 2
       for small values of steps (up to 10 maybe) and store them in an array */
    const double speedReduction = ACCEL2SPEED(decel);
    const int steps = int(speed / speedReduction);
    return SPEED2DIST(steps * speed - speedReduction * steps * (steps + 1) / 2) + speed * headwayTime;
}
```

① speedReduction = ACCEL2SPEED(4.5) = $\frac{x}{deltaT} = \frac{4.5}{1}$ = 4.5

steps = $\frac{speed}{speedReduction}$ = $\frac{20}{4.5} = 4.444$

SPEED2DIST(4.444 * 20 – 4.5*4.444*(4.444+1) / 2) = $\frac{x}{deltaT} = \frac{34.45}{1}$ = 34.45

34.45 + speed * headwayTime = 34.45 + 20* 0.1 (Subject's Tau) = 36.45m

② speedReduction = ACCEL2SPEED(4.5) = $\frac{x}{deltaT} = \frac{4.5}{1}$ = 4.5

steps = $\frac{speed}{speedReduction}$ = $\frac{15}{4.5} = 3.333$

SPEED2DIST(3.333 * 15 – 4.5*3.333*(3.333+1) / 2) = $\frac{x}{deltaT} = \frac{17.5}{1}$ = 17.5

17.5 + speed * headwayTime = 17.5 + 15* 0 = 17.5m

secureGap = SecureBackGap = MAX2(0, 36.45 – 17.5) = 18.95m

Calculating Required Gap – Required Gap

- Required Gap
  = secureBackGap + followerMinGap + subjectLength + subjectMinGap + secureFrontGap
  = 15m + 0m + 5m + 0m + 18.95m
  = 38.95m