```
// Establish obstacle avoidance constraints
void operator()(ADvector &fg, const ADvector &vars) {
   //…
        for (int t = 0; t < N; t++) {
            const AD<double>& x11_rear_axle = vars[x_start + t];
            const AD<double>& y11_rear_axle = vars[y_start + t];
            const AD<double>& psi11 = vars[psi_start + t];

             double middle_to_rear=vehcle_param_.length/2-vehcle_param_.back_edge_to_center;
               AD<double> x11=(x11_rear_axle+middle_to_rear*CppAD::cos(psi11));
             AD<double> y11=(y11_rear_axle+middle_to_rear*CppAD::sin(psi11));

            std::vector<std::vector<Vec2d>> obstacles_vertices_vec = this->parking_scenario_.obstacles;



            std::vector<Vec2d> obstacle_points;
            for (auto aaa: obstacles_vertices_vec) {
                for (auto bbb: aaa) {
                    obstacle_points.push_back(bbb);
                }
            }

            for (auto obstacle_point: obstacle_points) {

                AD<double> ref_x1 = (obstacle_point.x - x11);
                AD<double> ref_y1 = (obstacle_point.y - y11);
                AD<double> x1 =
                        ref_x1 * CppAD::cos(psi11) + ref_y1 * CppAD::sin(psi11);
                AD<double> y1 =
                        -ref_x1 * CppAD::sin(psi11) + ref_y1 * CppAD::cos(psi11);

                fg[1 + collide_constraint_start + collide_constraint_index] =
                         CppAD::abs(x1) - vehcle_param_.length / 2 + CppAD::abs(y1) - vehcle_param_.width / 2 +
                         CppAD::abs(CppAD::abs(x1) - vehcle_param_.length / 2) +
                         CppAD::abs(CppAD::abs(y1) - vehcle_param_.width / 2);
                collide_constraint_index++;

            }
            ADvector corners_x(4);
            ADvector corners_y(4);

            double half_length=vehcle_param_.length/2;
            double half_width=vehcle_param_.width/2;
            AD<double> sin_theta=CppAD::sin(psi11);
```

```
            AD<double> cos_theta=CppAD::cos(psi11);

            for (Rectangle_obs obj_rectangle:obs_rectangles_fg_){

            corners_x[0] = x11 + half_length * cos_theta - half_width * sin_theta;
            corners_y[0] = y11 + half_length * sin_theta + half_width * cos_theta;

            corners_x[1] = x11 - half_length * cos_theta - half_width * sin_theta;
            corners_y[1] = y11 - half_length * sin_theta + half_width * cos_theta;

            corners_x[2] = x11 - half_length * cos_theta + half_width * sin_theta;
            corners_y[2] = y11 - half_length * sin_theta - half_width * cos_theta;

            corners_x[3] = x11 + half_length * cos_theta + half_width * sin_theta;
            corners_y[3] = y11 + half_length * sin_theta - half_width * cos_theta;

            for (int i=0;i<4;i++){
                AD<double> ref_x2 = (corners_x[i] - obj_rectangle.center.x);
                AD<double> ref_y2 = (corners_y[i] - obj_rectangle.center.y);
                AD<double> x2 =
                        ref_x2 * CppAD::cos(obj_rectangle.angle) + ref_y2 * CppAD::sin(obj_rectangle.angle);
                AD<double> y2 =
                        -ref_x2 * CppAD::sin(obj_rectangle.angle) + ref_y2 * CppAD::cos(obj_rectangle.angle);

                fg[1 + collide_constraint_start + collide_constraint_index] =
                CppAD::abs(x2) - obj_rectangle.length / 2 + CppAD::abs(y2) - obj_rectangle.width / 2 +
                CppAD::abs(CppAD::abs(x2) - obj_rectangle.length / 2) +
                CppAD::abs(CppAD::abs(y2) - obj_rectangle.width / 2);

                collide_constraint_index++;
            }
            }
        }
    }
};
```