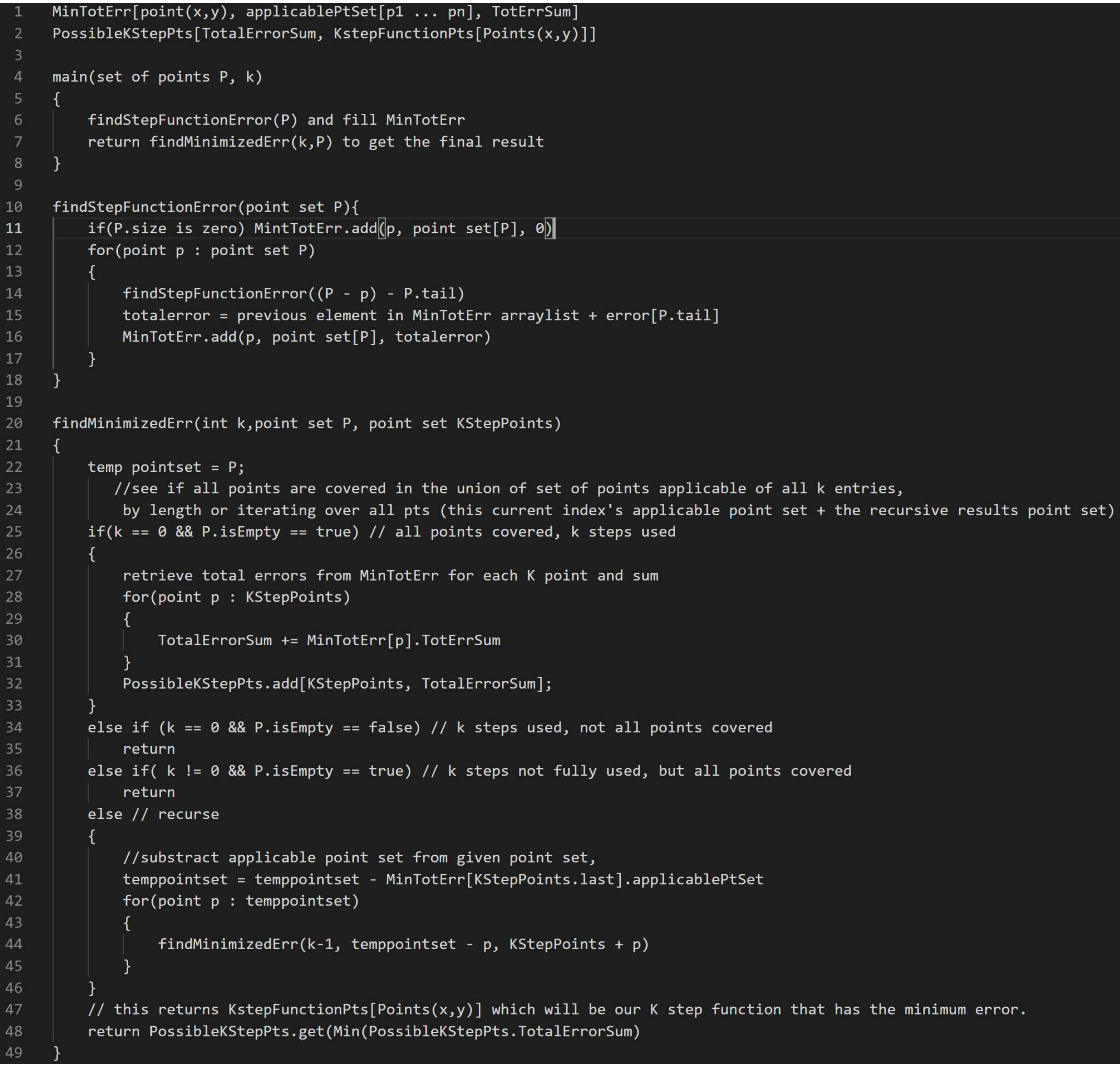

```
MinTotErr[point(x,y), applicablePtSet[p1 ... pn], TotErrSum]
PossibleKStepPts[TotalErrorSum, KstepFunctionPts[Points(x,y)]]

main(set of points P, k)
{
    findStepFunctionError(P) and fill MinTotErr
    return findMinimizedErr(k,P) to get the final result
}

findStepFunctionError(point set P){
    if(P.size is zero) MintTotErr.add(p, point set[P], 0)
    for(point p : point set P)
    {
        findStepFunctionError((P - p) - P.tail)
        totalerror = previous element in MinTotErr arraylist + error[P.tail]
        MinTotErr.add(p, point set[P], totalerror)
    }
}

findMinimizedErr(int k,point set P, point set KStepPoints)
{
    temp pointset = P;
       //see if all points are covered in the union of set of points applicable of all k entries,
        by length or iterating over all pts (this current index's applicable point set + the recursive results point set)
    if(k == 0 && P.isEmpty == true) // all points covered, k steps used
    {
        retrieve total errors from MinTotErr for each K point and sum
        for(point p : KStepPoints)
        {
            TotalErrorSum += MinTotErr[p].TotErrSum
        }
        PossibleKStepPts.add[KStepPoints, TotalErrorSum];
    }
    else if (k == 0 && P.isEmpty == false) // k steps used, not all points covered
        return
    else if( k != 0 && P.isEmpty == true) // k steps not fully used, but all points covered
        return
    else // recurse
    {
        //substract applicable point set from given point set,
        temppointset = temppointset - MinTotErr[KStepPoints.last].applicablePtSet
        for(point p : temppointset)
        {
            findMinimizedErr(k-1, temppointset - p, KStepPoints + p)
        }
    }
    // this returns KstepFunctionPts[Points(x,y)] which will be our K step function that has the minimum error.
    return PossibleKStepPts.get(Min(PossibleKStepPts.TotalErrorSum)
}
```