```

switch (CrossoverOperator)
{
    case CrossoverType.PMX:
        // i1, i2 cut locations
        // m[] partial map, m[i] => the mapping target of i

        // define two crossover map for chromosome
        index_1 = randomizer.Next(numberOfGenes);
        while (index_2 == index_1)
        {
            index_2 = randomizer.Next(numberOfGenes);
        }
        // swap two numbers if index_1 is larger
        if (index_2 < index_1)
        {
            temp_num = index_1;
            index_1 = index_2;
            index_2 = temp_num;
        }
        // initiate mapping
        mapping = new int[numberOfGenes];
        for (int i = 0; i < mapping.Length; i++) mapping[i] = -1;
        // build up the mapping
        for (int i = index_1; i < index_2; i++)
        {

            if (chromosomes[fatherIdx][i] == chromosomes[motherIdx][i]) continue;
            if (mapping[chromosomes[fatherIdx][i]] == -1 &&
            mapping[chromosomes[motherIdx][i]] == -1)
            {
                mapping[chromosomes[fatherIdx][i]] = chromosomes[motherIdx][i];
                mapping[chromosomes[motherIdx][i]] = chromosomes[fatherIdx][i];
            }
            else if (mapping[chromosomes[fatherIdx][i]] == -1)
            {
                mapping[chromosomes[fatherIdx][i]] = mapping[chromosomes[motherIdx][i]];
                try
                {
                    mapping[mapping[chromosomes[motherIdx][i]]] =
                    chromosomes[fatherIdx][i];
                }
                catch (System.IndexOutOfRangeException Exception)
                {

                }

                mapping[chromosomes[motherIdx][i]] = -2;
            }
            else if (mapping[chromosomes[motherIdx][i]] == -1)
            {
                try
                {
                    mapping[chromosomes[motherIdx][i]] =
                    mapping[chromosomes[fatherIdx][i]];
                }
                catch (System.IndexOutOfRangeException Exception) { }
                try
                {
                    mapping[mapping[chromosomes[fatherIdx][i]]] =
                    chromosomes[motherIdx][i];
                }
                catch (System.IndexOutOfRangeException Exception) { }
                try
                {
                    mapping[chromosomes[fatherIdx][i]] = -2;
                }
                catch (System.IndexOutOfRangeException Exception) { }
            }
```

```
                else
                {
                    try
                    {
                        mapping[mapping[chromosomes[motherIdx][i]]] =
                        mapping[chromosomes[fatherIdx][i]];
                    }
                    catch (System.IndexOutOfRangeException Exception) { }

                    try
                    {
                        mapping[mapping[chromosomes[fatherIdx][i]]] =
                        mapping[chromosomes[motherIdx][i]];
                    }
                    catch (System.IndexOutOfRangeException Exception) { }


                    mapping[chromosomes[fatherIdx][i]] = -3;
                    mapping[chromosomes[motherIdx][i]] = -3;
                }
            }

            // crossover and make two children
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (index_1 <= i && i < index_2)
                {
                    chromosomes[child1Idx][i] = chromosomes[motherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[fatherIdx][i];
                }
                else
                {
                    if (mapping[chromosomes[fatherIdx][i]] < 0) chromosomes[child1Idx][i] =
                    chromosomes[fatherIdx][i];
                    else chromosomes[child1Idx][i] = mapping[chromosomes[fatherIdx][i]];

                    if (mapping[chromosomes[motherIdx][i]] < 0) chromosomes[child2Idx][i] =
                    chromosomes[motherIdx][i];
                    else chromosomes[child2Idx][i] = mapping[chromosomes[motherIdx][i]];
                }
            }
            // crossover finished.

            // check if all gene are distinct, if not let other chromosome replace it
            for (int i = 0; i < populationSize * 3; i++)
            {
                temp = new int[numberOfGenes];
                for (int j = 0; j < numberOfGenes; j++)
                {
                    temp[j] = j;
                }

                if (chromosomes[i].Distinct().ToArray().Count() != numberOfGenes)
                {
                    chromosomes[i] = temp.OrderBy(x => randomizer.Next()).ToArray();
                }
                else
                {
                    break;
                }
            }

            break;
        case CrossoverType.OX:
            // order crossover
            // define two crossover map for chromosome
            index_1 = randomizer.Next(numberOfGenes);
            while (index_2 == index_1)
            {
                index_2 = randomizer.Next(numberOfGenes);
            }
            // swap two numbers if index_1 is larger
            if (index_2 < index_1)
            {
                temp_num = index_1;
                index_1 = index_2;
                index_2 = temp_num;
            }
            // start creating crossover children
            temp_num = 0;
            temp_num_2 = 0;
            for (int i = index_1; i < index_2; i++)
            {
                chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
            }
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (!(chromosomes[child1Idx].Contains(chromosomes[motherIdx][i])))
                {
                    if (temp_num == index_1) temp_num = index_2;
                    chromosomes[child1Idx][temp_num] = chromosomes[motherIdx][i];
                    temp_num += 1;
                }
                if (!(chromosomes[child2Idx].Contains(chromosomes[fatherIdx][i])))
                {
                    if (temp_num_2 == index_1) temp_num_2 = index_2;
                    chromosomes[child2Idx][temp_num] = chromosomes[fatherIdx][i];
                    temp_num_2 += 1;
                }
            }
            break;
        case CrossoverType.POX:
            // position-based crossover
            temp_num = (int)Math.Round(numberOfGenes * crossoverRate);
            temp_1 = new int[temp_num];
            // create random index array
            for (int i = 0; i < temp_num; i++)
            {
                temp_1[i] = randomizer.Next(numberOfGenes);
            }
            // crossover children
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (temp_1.Contains(i))
                {
                    chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
                }
                else
                {
                    chromosomes[child1Idx][i] = chromosomes[motherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[fatherIdx][i];
                }

            }
            break;
        case CrossoverType.OSS:
            // order-based crossover
            temp_num = (int)Math.Round(numberOfGenes * crossoverRate);
            temp_1 = new int[temp_num];
            // create random value array
            for (int i = 0; i < temp_num; i++)
            {
                temp_1[i] = randomizer.Next(numberOfGenes);
            }
            // crossover children
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (!(temp_1.Contains(chromosomes[fatherIdx][i])))
                {
                    chromosomes[child1Idx][i] = chromosomes[motherIdx][i];
                }
                else
                {
                    chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                }

                if (!(temp_1.Contains(chromosomes[motherIdx][i])))
                {
                    chromosomes[child2Idx][i] = chromosomes[fatherIdx][i];
                }
                else
                {
                    chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
                }
            }

            break;
        case CrossoverType.OCCC:
            break;
    }



    switch (CrossoverOperator)
    {
        case CrossoverType.OnePointCut:
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (i < temp_num)
                {
                    chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
                }
                else
                {
                    chromosomes[child2Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child1Idx][i] = chromosomes[motherIdx][i];
                }

            }
            break;
        case CrossoverType.TwoPointCut:
            for (int i = 0; i < numberOfGenes; i++)
            {
                if (i < temp_num)
                {
                    chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
                }
                else if (temp_num < i && i < temp_num_2)
                {
                    chromosomes[child2Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child1Idx][i] = chromosomes[motherIdx][i];
                }
                else
                {
                    chromosomes[child1Idx][i] = chromosomes[fatherIdx][i];
                    chromosomes[child2Idx][i] = chromosomes[motherIdx][i];
                }

            }
            break;
    }

```