```cpp
#include "PCH.h"
std::mt19937 mt{ std::random_device{}() };

GA::GA(int chromosomeDim, int populationDim, double crossoverProb, int randomSelectionChance, int maxGenerations, int numPrelimRuns, int maxPrelimGenerations, double mutationProb, int crossoverType, bool computeStatistics)
{
    this->randomSelectionChance = randomSelectionChance;
    this->crossoverType = crossoverType;
    this->chromosomeDim = chromosomeDim;
    this->populationDim = populationDim;
    this->computeStatistics = computeStatistics;

    this->chromosomes = std::vector<Chromosome*>(populationDim);
    this->chromNextGen = std::vector<Chromosome*>(populationDim);
    this->prelimChrom = std::vector<Chromosome*>(populationDim);
    this->genAvgDeviation = std::vector<double>(maxGenerations);
    this->genAvgFitness = std::vector<double>(maxGenerations);

    this->crossoverProb = crossoverProb;
    this->maxGenerations = maxGenerations;
    this->numPrelimRuns = numPrelimRuns;
    this->maxPrelimGenerations = maxPrelimGenerations;
    this->mutationProb = mutationProb;
}

double GA::getAvgDeviation(int iGeneration)
{
    return (this->genAvgDeviation[iGeneration]);
}

double GA::getAvgFitness(int iGeneration)
{
    return (this->genAvgFitness[iGeneration]);
}

double GA::getMutationProb()
{
    return mutationProb;
}

int GA::getMaxGenerations()
{
    return maxGenerations;
}

int GA::getNumPrelimRuns()
{
    return numPrelimRuns;
}

int GA::getMaxPrelimGenerations()
{
    return maxPrelimGenerations;
}
```

```

int GA::getRandomSelectionChance()
{
    return randomSelectionChance;
}

double GA::getCrossoverProb()
{
    return crossoverProb;
}

int GA::getChromosomeDim()
{
    return chromosomeDim;
}

int GA::getPopulationDim()
{
    return populationDim;
}

int GA::getCrossoverType()
{
    return crossoverType;
}

bool GA::getComputeStatistics()
{
    return computeStatistics;
}

Chromosome * GA::getFittestChromosome()
{
    return (this->chromosomes[bestFitnessChromIndex]);
}

double GA::getFittestChromosomesFitness()
{
    return (this->chromosomes[bestFitnessChromIndex]->fitness);
}

int GA::getRandom(int upperBound)
{
    std::uniform_int_distribution<int> dist(0, upperBound);
    return dist(mt);
}

double GA::getRandom(double upperBound)
{
    std::uniform_real_distribution<double> dist(0.0, upperBound);
    return dist(mt);
}



int GA::evolve()
```

```cpp
{
    int iGen;
    int iPrelimChrom, iPrelimChromToUsePerRun;
    std::cout << "CPGene start" << std::endl;

    if (numPrelimRuns > 0)
    {
        iPrelimChrom = 0;
        iPrelimChromToUsePerRun = populationDim / numPrelimRuns;
        for (int iPrelimRuns = 1; iPrelimRuns <= numPrelimRuns; iPrelimRuns++)
        {
            iGen = 0;
            initPopulation();
            while (iGen < maxPrelimGenerations)
            {
                std::cout << iPrelimRuns << " z " << numPrelimRuns << " Uruchomien wstepnych. Pokolenie: " << (iGen + 1) << " z " << maxPrelimGenerations << std::endl;

                computeFitnessRankings();
                doGeneticMating();
                copyNextGenToThisGen();

                if (computeStatistics == true)
                {
                    this->genAvgDeviation[iGen] = getAvgDeviationAmongChroms();
                    this->genAvgFitness[iGen] = getAvgFitness();
                }
                iGen++;
            }
            computeFitnessRankings();
            int iNumPrelimSaved = 0;
            for (int i = 0; i < populationDim && iNumPrelimSaved < iPrelimChromToUsePerRun; i++)
            {
                if (this->chromosomes[i]->fitnessRank >= populationDim - iPrelimChromToUsePerRun)
                {
                    this->prelimChrom[iPrelimChrom + iNumPrelimSaved]->copyChromGenes(this->chromosomes[i]);
                    iNumPrelimSaved++;
                }
            }
            iPrelimChrom += iNumPrelimSaved;
        }
        for (int i = 0; i < iPrelimChrom; i++)
        {
            this->chromosomes[i]->copyChromGenes(this->prelimChrom[i]);
        }
        std::cout << "Populacja wstepna (po generacji wstepnej):" << std::endl;
    }
    else
    {
        std::cout << "Populacja wstepna (bez generacji wstepnej):" <<
```

```
            std::endl;
    }

    addChromosomesToLog(0, 10);

    iGen = 0;
    while (iGen < maxGenerations)
    {
        computeFitnessRankings();
        doGeneticMating();
        copyNextGenToThisGen();
        //std::cout << "Gen: " << iGen << std::endl;

        if (computeStatistics == true)
        {
            this->genAvgDeviation[iGen] = getAvgDeviationAmongChroms();
            this->genAvgFitness[iGen] = getAvgFitness();
        }

        iGen++;
    }
    std::cout << "Gen: " << (iGen) << " Avg Fitness = " << this->genAvgFitness
      [iGen-1] << " Avg DEV = " << this->genAvgDeviation[iGen-1] << std::endl;
    addChromosomesToLog(iGen, 10);
    computeFitnessRankings();
    std::cout << "Najlepszy chromosom: ";
    std::cout << this->chromosomes[this->bestFitnessChromIndex]->getGenesAsStr
      () << " Fitness = " << std::fixed << std::setprecision(8) << this-
      >chromosomes[this->bestFitnessChromIndex]->fitness << std::endl;

    std::cout << "CPGene stop" << std::endl;
    return (iGen);
}



double GA::getAvgFitness()
{
    double rSumFitness = 0.0;

    for (int i = 0; i < populationDim; i++)
    {
        rSumFitness += this->chromosomes[i]->fitness;
    }
    return (rSumFitness / populationDim);
}


void GA::selectTwoParents(std::vector<int>& indexParents)
{
    int indexParent1 = indexParents[0];
    int indexParent2 = indexParents[1];
    bool bFound = false;
    int index;

    while (bFound == false)
```

```cpp
    {
        index = getRandom(populationDim-1);

        if (randomSelectionChance > getRandom(100))
        {
            indexParent1 = index;
            bFound = true;
        }
        else
        {

            if (this->chromosomes[index]->fitnessRank + 1 > getRandom(populationDim-1))
            {
                indexParent1 = index;
                bFound = true;
            }
        }
    }
    bFound = false;
    while (bFound == false)
    {
        index = getRandom(populationDim-1);

        if (randomSelectionChance > getRandom(100))
        {
            if (index != indexParent1)
            {
                indexParent2 = index;
                bFound = true;
            }
        }
        else
        {

            if ((index != indexParent1) && (this->chromosomes[index]->fitnessRank + 1 > getRandom(populationDim-1)))
            {
                indexParent2 = index;
                bFound = true;
            }
        }
    }
    indexParents[0] = indexParent1;
    indexParents[1] = indexParent2;
}

int GA::getFitnessRank(double fitness)
{
    int fitnessRank = -1;
    for (int i = 0; i < populationDim; i++)
    {
        if (fitness >= this->chromosomes[i]->fitness)
        {
            fitnessRank++;
        }
    }

    return (fitnessRank);
    return (fitnessRank);
}

void GA::computeFitnessRankings()
{
    for (int i = 0; i < populationDim; i++)
    {
        this->chromosomes[i]->fitness = getFitness(i);
    }

    for (int i = 0; i < populationDim; i++)
    {
        this->chromosomes[i]->fitnessRank = getFitnessRank(this->chromosomes
          [i]->fitness);
    }

    double rBestFitnessVal;
    double rWorstFitnessVal;
    for (int i = 0; i < populationDim; i++)
    {
        if (this->chromosomes[i]->fitnessRank == populationDim - 1)
        {
            rBestFitnessVal = this->chromosomes[i]->fitness;
            this->bestFitnessChromIndex = i;
        }
        if (this->chromosomes[i]->fitnessRank == 0)
        {
            rWorstFitnessVal = this->chromosomes[i]->fitness;
            this->worstFitnessChromIndex = i;
        }
    }
}

void GA::doGeneticMating()
{
    int iCnt, iRandom;
    int indexParent1 = -1, indexParent2 = -1;
    Chromosome *Chrom1, *Chrom2;

    iCnt = 0;

    //elityzm
    this->chromNextGen[iCnt]->copyChromGenes(this->chromosomes[this-
      >bestFitnessChromIndex]);
    iCnt++;
    this->chromNextGen[iCnt]->copyChromGenes(this->chromosomes[this-
      >bestFitnessChromIndex]);
    iCnt++;

    if (dynamic_cast<GAString*>(this) != nullptr)
    {
        Chrom1 = new ChromChars(chromosomeDim);
        Chrom2 = new ChromChars(chromosomeDim);
    }
    else if (dynamic_cast<GAFloat*>(this) != nullptr)
    {
        Chrom1 = new ChromFloat(chromosomeDim);
        Chrom2 = new ChromFloat(chromosomeDim);
    }
    else
    {
        Chrom1 = nullptr;
        Chrom2 = nullptr;
    }

    do
    {
        std::vector<int> indexes = { indexParent1, indexParent2 };
        selectTwoParents(indexes);
        indexParent1 = indexes[0];
        indexParent2 = indexes[1];

        Chrom1->copyChromGenes(this->chromosomes[indexParent1]);
        Chrom2->copyChromGenes(this->chromosomes[indexParent2]);

        if (getRandom(1.0) < crossoverProb)
        {
            if (this->crossoverType == Crossover::ctOnePoint)
            {
                doOnePtCrossover(Chrom1, Chrom2);
            }
            else if (this->crossoverType == Crossover::ctTwoPoint)
            {
                doTwoPtCrossover(Chrom1, Chrom2);
            }
            else if (this->crossoverType == Crossover::ctUniform)
            {
                doUniformCrossover(Chrom1, Chrom2);
            }
            else if (this->crossoverType == Crossover::ctRoulette)
            {
                iRandom = getRandom(3);
                if (iRandom < 1)
                {
                    doOnePtCrossover(Chrom1, Chrom2);
                }
                else if (iRandom < 2)
                {
                    doTwoPtCrossover(Chrom1, Chrom2);
                }
                else
                {
                    doUniformCrossover(Chrom1, Chrom2);
                }
            }

            this->chromNextGen[iCnt]->copyChromGenes(Chrom1);
            iCnt++;
            this->chromNextGen[iCnt]->copyChromGenes(Chrom2);
            iCnt++;
        }
        else
        {
            this->chromNextGen[iCnt]->copyChromGenes(Chrom1);
            iCnt++;

            this->chromNextGen[iCnt]->copyChromGenes(Chrom2);
            iCnt++;
        }
    } while (iCnt < populationDim);

}

void GA::copyNextGenToThisGen()
{
    for (int i = 0; i < populationDim; i++)
    {
        this->chromosomes[i]->copyChromGenes(this->chromNextGen[i]);

        if (i != this->bestFitnessChromIndex)
        {
            if ((i == this->worstFitnessChromIndex) || (getRandom(1.0) < mutationProb))
            {
                doRandomMutation(i);
            }
        }
    }
}

void GA::addChromosomesToLog(int iGeneration, int iNumChromosomesToDisplay)
{
    std::string sGen, sChrom;

    if (iNumChromosomesToDisplay > this->populationDim)
    {
        iNumChromosomesToDisplay = this->chromosomeDim;
    }

    for (int i = 0; i < iNumChromosomesToDisplay; i++)
    {
        this->chromosomes[i]->fitness = getFitness(i);
        sGen = "" + std::to_string(iGeneration);
        if (sGen.length() < 2)
        {
            sGen = sGen + " ";
        }
        sChrom = "" + std::to_string(i);
        if (sChrom.length() < 2)
        {
            sChrom = sChrom + " ";
        }
        std::cout << "Gen " << sGen << ": Chrom" << sChrom << " = " << this->chromosomes[i]->getGenesAsStr() << ", fitness = " << std::fixed << std::setprecision(8) <<  this->chromosomes[i]->fitness << std::endl;
    }
}


long long GA::binaryStrToInt(const std::string & sBinary)
{
    long long digit, iResult = 0;

    int iLen = sBinary.length();
    for (int i = iLen - 1; i >= 0; i--)
    {
        if (sBinary[i] == '1')
        {
            digit = 1;
        }
        else
        {
            digit = 0;
        }
        iResult += (digit << (iLen - i - 1));
    }
    return (iResult);
}


double GA::getAvgDeviationAmongChroms()
{
    int devCnt = 0;
    for (int iGene = 0; iGene < this->chromosomeDim; iGene++)
    {
        if (dynamic_cast<GAString*>(this) != nullptr)
        {
            wchar_t bestFitGene = (static_cast<ChromChars*>(this->chromosomes
              [this->bestFitnessChromIndex]))->getGene(iGene);
            for (int i = 0; i < this->populationDim; i++)
            {
                wchar_t thisGene = (static_cast<ChromChars*>(this->chromosomes
                  [i]))->getGene(iGene);
                if (thisGene != bestFitGene)
                {
                    devCnt++;
                }
            }
        }
        else if (dynamic_cast<GAFloat*>(this) != nullptr)
        {
            double bestFitGene = (static_cast<ChromFloat*>(this->chromosomes
              [this->bestFitnessChromIndex]))->getGene(iGene);
            for (int i = 0; i < populationDim; i++)
            {
                double thisGene = (static_cast<ChromFloat*>(this->chromosomes
                  [i]))->getGene(iGene);
                if (thisGene != bestFitGene)
                {
                    devCnt++;
                }
            }
        }
        else
        {
            //Chromstrings
        }
    }

    return (static_cast<double>(devCnt));
}

GA::~GA()
{
}
```