```
#include <cstdlib>
#include <iostream>
#include <math.h>
#include "conio.h"
#include <ga/ga.h>

using namespace std;

int popsize  = 100;
int ngen     = 1000;
float pmut   = 0.01;
float pcross = 0.6;

float objective(GAGenome &);

int main(int argc, char *argv[])
{
  GABin2DecPhenotype map;
  map.add(30, 0, 1*M_PI);
  GABin2DecGenome genome(map, objective);

  GASimpleGA ga(genome);
  ga.populationSize(popsize);
  ga.nGenerations(ngen);
  ga.pMutation(pmut);
  ga.pCrossover(pcross);
  ga.scoreFilename("zbieznosc.dat");
  ga.scoreFrequency(10);
  ga.flushFrequency(50);
  ga.evolve((unsigned)time(0));

  genome = ga.statistics().bestIndividual();
  cout << "Najlepsze rozwiazanie to F(x=";
  cout << genome.phenotype(0) << ")=";
  cout << objective(genome) << endl;

  getch();
  return EXIT_SUCCESS;
}
```

GAlib:
A C++ Library of Genetic Algorithm Components

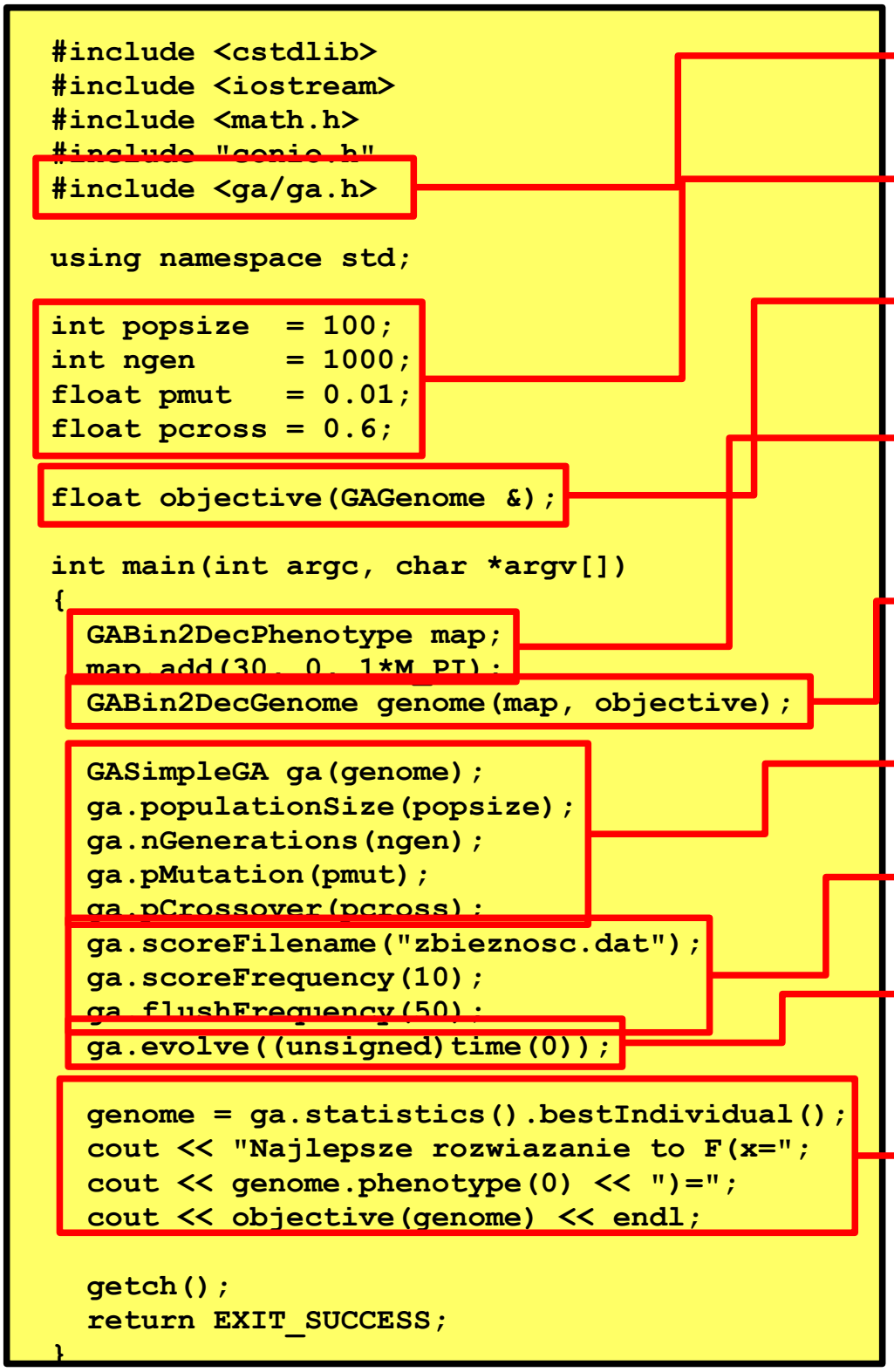
version 2.4

Matthew Wall

Mechanical Engineering Department
Massachusetts Institute of Technology

http://lancet.mit.edu/ga/
galib-request@mit.edu

Copyright © 1996 Matthew Wall



Włączamy plik nagłówkowy biblioteki

Inicjalizujemy parametry algorytmu

Informujemy kompilator, że gdzieś dalej wystąpi funkcja dopasowania.

Definiujemy fenotyp, jako liczbę dziesiętną z zakresu 0 – M_PI, reprezentowaną na 30 bitach.

Tworzymy genom oparty na zdefiniowanym fenotypie i przypisujemy mu funkcję dostosowania nazwaną „objective”.

Inicjalizujemy klasyczny algorytm genetyczny (taki jak SGA) zdefiniowanym genomem oraz ustalamy parametry algorytmu.

Ustalamy parametry wyjściowe.

Uruchamiamy ewolucję, startując generator liczb losowych przypadkową wartością.

Po zakończeniu ewolucji wyświetlamy najlepsze rozwiązanie.

Hierarchia klas

```
GAGeneticAlgorithm
        GASteadyStateGA
        GASimpleGA
        GAIncrementalGA
        GADemeGA

GAStatistics

GAParameterList

GAPopulation

GAScalingScheme
        GANoScaling
        GALinearScaling
        GASigmaTruncationScaling
        GAPowerLawScaling
        GASharing

GASelectionScheme
        GARankSelector
        GARouletteWheelSelector
        GATournamentSelector
        GAUniformSelector
        GASRSSelector
        GADSSelector
```

```
GAGenome
        GA1DBinaryStringGenome
                GABin2DecGenome
GA2DBinaryStringGenome
GA3DBinaryStringGenome
GA1DArrayGenome<>
        GA1DArrayAlleleGenome<>
        GAStringGenome
        GARealGenome
GA2DArrayGenome<>
        GA2DArrayAlleleGenome<>
GA3DArrayGenome<>
        GA3DArrayAlleleGenome<>
GATreeGenome<>
GAListGenome<>
```

```
GAArray<>
GAAlleleSetArray<>
GAAlleleSet<>
GABinaryString
GABin2DecPhenotype
GATree<>
GATreeIter<>
GAList<>
GAListIter<>
```

Typy globalne

```
typedef float GAProbability, GAProb
typedef enum _GABoolean {gaFalse, gaTrue} GABoolean, GABool
typedef enum _GAStatus {gaSuccess, gaFailure} GAStatus
typedef unsigned char GABit
```

Stałe i zmienne predefiniowane globalnie

```
char* gaErrMsg;
int gaDefScoreFrequency1 = 1;
int gaDefScoreFrequency2 = 100;
float gaDefLinearScalingMultiplier = 1.2;
float gaDefSigmaTruncationMultiplier = 2.0;
float gaDefPowerScalingFactor = 1.0005;
float gaDefSharingCutoff = 1.0;
```

Obsługa błędów

```
extern char gaErrMsg[];
void GAReportErrors(GABoolean flag);
void GASetErrorStream(ostream&);
```

Prototypy funkcji

```
GABoolean (*GAGeneticAlgorithm::Terminator)(GAGeneticAlgorithm&)

GAGenome& (*GAIncrementalGA::ReplacementFunction)(GAGenome&, GAPopulation&)

void (*GAPopulation::Initializer)(GAPopulation &)

void (*GAPopulation::Evaluator)(GAPopulation &)

void (*GAGenome::Initializer)(GAGenome &)

float (*GAGenome::Evaluator)(GAGenome &)

int (*GAGenome::Mutator)(GAGenome &, float)

float (*GAGenome::Comparator)(const GAGenome &, const GAGenome&)

int (*GAGenome::SexualCrossover)(const GAGenome&, const GAGenome&, GAGenome*,
GAGenome*)

int (*GAGenome::AsexualCrossover)(const GAGenome&, GAGenome*)

int (*GABinaryEncoder)(float& value, GABit* bits, unsigned int nbits, float min,
float max)

int (*GABinaryDecoder)(float& value, const GABit* bits, unsigned int nbits,float
min, float max)
```

Generatory liczb losowych

```
void GARandomSeed(unsigned s = 0)

int GARandomInt()
int GARandomInt(int low, int high)

double GARandomDouble()
double GARandomDouble(double low, double high)

float GARandomFloat()
float GARandomFloat(float low, float high)

int GARandomBit()

GABoolean GAFlipCoin(float p)

int GAGaussianInt(int stddev)
float GAGaussianFloat(float stddev)
double GAGaussianDouble(double stddev)
double GAUnitGaussian()
```

class GAGeneticAlgorithm : public GAID

Typy i definicje

```
GABoolean (*GAGeneticAlgorithm::Terminator)(GAGeneticAlgorithm&)
enum { MINIMIZE = -1, MAXIMIZE = 1 };
```

Spis funkcji

```
static GAParameterList& registerDefaultParameters(GAParameterList&)
void * userData()
void * userData(void *)
void initialize(unsigned int seed=0)
void evolve(unsigned int seed=0)
void step()
GABoolean done()
GAGeneticAlgorithm::Terminator terminator()
GAGeneticAlgorithm::Terminator terminator(GAGeneticAlgorithm::Terminator)
const GAStatistics & statistics() const
float convergence() const
int generation() const
void flushScores()
int minimaxi(int) / const
int minimize()
int maximize()
int nGenerations(unsigned int) / const
int nConvergence(unsigned int) / const
float pConvergence(float) / const
float pMutation(float) / const
float pCrossover(float) / const
GAGenome::SexualCrossover crossover(GAGenome::SexualCrossover func) / const
GAGenome::AsexualCrossover crossover(GAGenome::AsexualCrossover func) / const
const GAPopulation & population(const GAPopulation&) / const
int populationSize(unsigned int n) / const
int nBestGenomes(unsigned int n) / const
GAScalingScheme & scaling(const GAScalingScheme&) / const
GASelectionScheme & selector(const GASelectionScheme& s) / const
void objectiveFunction(GAGenome::Evaluator)
void objectiveData(const GAEvalData&)
```

Uwaga! Zapis:
`int nGenerations(unsigned int) / const`
oznacza, że funkcja może być również wywoływana w postaci:
`int nGenerations() const`

class GAGeneticAlgorithm : public GAID

Spis funkcji (cd)

```
int scoreFrequency(unsigned int frequency) / const
int flushFrequency(unsigned int frequency) / const
char* scoreFilename(const char *filename) / const
int selectScores(GAStatistics::ScoreID which) / const
GABoolean recordDiversity(GABoolean flag) / const
const GAParameterList & parameters()
const GAParameterList & parameters(const GAParameterList &)
const GAParameterList & parameters(int& argc, char** argv, GABoolean flag = gaFalse)
const GAParameterList & parameters(const char* filename, GABoolean flag = gaFalse)
const GAParameterList & parameters(istream&, GABoolean flag = gaFalse);
int set(const char* s, int v)
int set(const char* s, unsigned int v)
int set(const char* s, char v)
int set(const char* s, const char* v)
int set(const char* s, const void* v)
int set(const char* s, double v);
int write(const char* filename)
int write(ostream&)
int read(const char* filename)
int read(ostream&)
```

class GASimpleGA : public GAGeneticAlgorithm

Konstruktory

```
GASimpleGA(const GAGenome&)
GASimpleGA(const GAPopulation&)
GASimpleGA(const GASimpleGA&)
```

Spis funkcji

```
static GAParameterList& registerDefaultParameters(GAParameterList&)
GASimpleGA & operator++()
GABoolean elitist() const
GABoolean elitist(GABoolean flag)
```

class GASteadyStateGA : public GAGeneticAlgorithm

Konstruktory

```
GASteadyStateGA(const GAGenome&)
GASteadyStateGA(const GAPopulation&)
GASteadyStateGA(const GASteadyStateGA&))
```

Spis funkcji

```
static GAParameterList& registerDefaultParameters(GAParameterList&)
GASteadyStateGA & operator++()
float pReplacement() const
float pReplacement(float percentage)
int nReplacement() const
int nReplacement(unsigned int)
```

class GADemeGA : public GAGeneticAlgorithm

class GAIncrementalGA : public GAGeneticAlgorithm

Typy i definicje

```
GAGenome& (*GAIncrementalGA::ReplacementFunction)(GAGenome &, GAPopulation &)
enum ReplacementScheme {
        RANDOM = GAPopulation::RANDOM,
        BEST = GAPopulation::BEST,
        WORST = GAPopulation::WORST,
        CUSTOM = -30,
        CROWDING = -30,
        PARENT = -10
};
```

Konstruktory

```
GAIncrementalGA(const GAGenome&)
GAIncrementalGA(const GAPopulation&)
GAIncrementalGA(const GAIncrementalGA&)
```

Spis funkcji

```
static GAParameterList& registerDefaultParameters(GAParameterList&)
GASteadyStateGA & operator++()
ReplacementScheme replacement()
ReplacementScheme replacement(ReplacementScheme, ReplacementFunction f = NULL)
int nOffspring() const
int nOffspring(unsigned int n)
```

Terminatory *(funkcje ustalające warunki zakończenia ewolucji)*

```
GABoolean GAGeneticAlgorithm::TerminateUponGeneration(GAGeneticAlgorithm &)
GABoolean GAGeneticAlgorithm::TerminateUponConvergence(GAGeneticAlgorithm &)
GABoolean GAGeneticAlgorithm::TerminateUponPopConvergence(GAGeneticAlgorithm &)
```

class GAGenome : public GAID

Typy i definicje

```
enum GAGenome::Dimension { LENGTH, WIDTH, HEIGHT, DEPTH }
enum GAGenome::CloneMethod { CONTENTS, ATTRIBUTES }
enum { FIXED_SIZE = -1, ANY_SIZE = -10 }
float (*GAGenome::Evaluator)(GAGenome &)
void (*GAGenome::Initializer)(GAGenome &)
int (*GAGenome::Mutator)(GAGenome &, float)
float (*GAGenome::Comparator)(const GAGenome &, const GAGenome&)
int (*GAGenome::SexualCrossover)(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);
int (*GAGenome::AsexualCrossover)(const GAGenome&, GAGenome*);
```

Spis funkcji

```
virtual void copy(const GAGenome & c)
virtual GAGenome * clone(CloneMethod flag = CONTENTS)
float score(float s) / const
int nevals()
float evaluate(GABoolean flag = gaFalse) const
GAGenome::Evaluator evaluator(GAGenome::Evaluator func) / const
void initialize()
GAGenomeInitializer initializer(GAGenome::Initializer func) / const
int mutate(float pmutation)
GAGenome::Mutator mutator(GAGenome::Mutator func) / const
float compare(const GAGenome& g) const
GAGenome::Comparator comparator(GAGenome::Comparator c) / const
GAGenome::SexualCrossover crossover(GAGenome::SexualCrossover f)
GAGenome::SexualCrossover sexual()
GAGenome::AsexualCrossover crossover(GAGenome::AsexualCrossover f)
GAGenome::AsexualCrossover asexual()
GAGeneticAlgorithm * geneticAlgorithm(GAGeneticAlgorithm &) / const
void * userData(void * data) / const
GAEvalData * evalData(void * data) / const
virtual int read(istream &)
virtual int write(ostream &) const
virtual int equal(const GAGenome &) const
virtual int notequal(const GAGenome &) const
int operator==(const GAGenome&, const GAGenome&)
int operator!=(const GAGenome&, const GAGenome&)
ostream & operator<<(ostream&, const GAGenome&)
```

class GA1DArrayGenome<T> : public GAArray<T>, public GAGenome

Konstruktory

```
GA1DArrayGenome(unsigned int length, GAGenome::Evaluator objective = NULL,
void * userData = NULL)
GA1DArrayGenome(const GA1DArrayGenome<T> &)
```

Spis funkcji

```
const T & gene(unsigned int x=0) const
T & gene(unsigned int x=0)
T & gene(unsigned int x, const T& value) const
T & operator[](unsigned int x) / const
int length() const
int length(int l)
int resize(int x)
int resizeBehaviour() const
int resizeBehaviour(unsigned int minx, unsigned int maxx)
void copy(const GA1DArrayGenome<T>& original, unsigned int dest, unsigned int src,
unsigned int length)
void swap(unsigned int x1, unsigned int x2)
```

Dozwolone operatory genetyczne

```
GA1DArrayGenome<>::SwapMutator
GA1DArrayGenome<>::ElementComparator
GA1DArrayGenome<>::UniformCrossover
GA1DArrayGenome<>::EvenOddCrossover
GA1DArrayGenome<>::OnePointCrossover
GA1DArrayGenome<>::TwoPointCrossover
GA1DArrayGenome<>::PartialMatchCrossover
GA1DArrayGenome<>::OrderCrossover
GA1DArrayGenome<>::CycleCrossover
```

Ustawienia domyślne

```
inicjalizacja:     GAGenome::NoInitializer
porównywanie:              GA1DArrayGenome<>::ElementComparator
mutacja:                   GA1DArrayGenome<>::SwapMutator
krzyżowanie:               GA1DArrayGenome<>::OnePointCrossover
```

class GA1DArrayAlleleGenome<T> : public GAArrayGenome<T>

Konstruktory

```
GA1DArrayAlleleGenome(unsigned int length, const GAAlleleSet<T>& alleleset,
GAGenome::Evaluator objective = NULL, void * userData = NULL)
GA1DArrayAlleleGenome(const GAAlleleSetArray<T>& allelesets, GAGenome::Evaluator
objective
= NULL, void * userData = NULL)
GA1DArrayAlleleGenome(const GA1DArrayAlleleGenome<T>&)
```

Spis funkcji

```
const GAAlleleSet<T>& alleleset(unsigned int i = 0) const
```

Dozwolone operatory genetyczne

```
GA1DArrayAlleleGenome<>::UniformInitializer
GA1DArrayAlleleGenome<>::OrderedInitializer
GA1DArrayAlleleGenome<>::FlipMutator
```

Ustawienia domyślne

```
inicjalizacja:     GA1DArrayAlleleGenome<>::UniformInitializer
porównywanie:              GA1DArrayGenome<>::ElementComparator
mutacja:                   GA1DArrayAlleleGenome<>::FlipMutator
krzyżowanie:               GA1DArrayGenome<>::OnePointCrossover
```

```
class GA2DArrayGenome<T> : public GAArray<T>, public GAGenome

class GA1DArrayAlleleGenome<T> : public GAArrayGenome<T>

class GA3DArrayGenome<T> : public GAArray<T>, public GAGenome

class GA1DArrayAlleleGenome<T> : public GAArrayGenome<T>

class GA1DBinaryStringGenome : public GABinaryString, public GAGenome

class GA2DBinaryStringGenome : public GABinaryString, public GAGenome

class GA3DBinaryStringGenome : public GABinaryString, public GAGenome

class GAListGenome<T> : public GAList<T>, public GAGenome

class GATreeGenome<T> : public GATree<T>, public GAGenome
```

class GABin2DecGenome : public GA1DBinaryStringGenome

Konstruktory

```
GABin2DecGenome(const GABin2DecPhenotype &, GAGenome::Evaluator objective = NULL,
void *userData = NULL)
GABin2DecGenome(const GABin2DecGenome&)
```

Spis funkcji

```
const GABin2DecPhenotype& phenotypes(const GABin2DecPhenotype &)
const GABin2DecPhenotype& phenotypes() const
int nPhenotypes() const
float phenotype(unsigned int n) const
float phenotype(unsigned int n, float value)
void encoder(GABinaryEncoder)
void decoder(GABinaryDecoder)
```

Ustawienia domyślne

```
inicjalizacja:     GA1DBinaryStringGenome::UniformInitializer
porównywanie:              GA1DBinaryStringGenome::BitComparator
mutacja:                   GA1DBinaryStringGenome::FlipMutator
krzyżowanie:               GA1DBinaryStringGenome::OnePointCrossover
kodowanie:                 GABinaryEncode
dekodowanie:               GABinaryDecode
```

GARealGenome

```
typedef GAAlleleSet<float> GARealAlleleSet
typedef GAAlleleSetCore<float> GARealAlleleSetCore
typedef GAAlleleSetArray<float> GARealAlleleSetArray
typedef GA1DArrayAlleleGenome<float> GARealGenome
```

Konstruktory

```
GARealGenome(unsigned int length, const GARealAlleleSet &, GAGenome::Evaluator
objective =
NULL, void * userData = NULL)
GARealGenome(const GARealAlleleSetArray &, GAGenome::Evaluator objective = NULL,
void *userData = NULL)
GARealGenome(const GARealGenome&)
```

Dozwolone operatory genetyczne

```
GARealGenome::UniformInitializer
GARealGenome::OrderedInitializer
GARealGenome::FlipMutator
GARealGenome::SwapMutator
GARealGaussianMutator
GARealGenome::UniformCrossover
GARealGenome::EvenOddCrossover
GARealGenome::OnePointCrossover
GARealGenome::TwoPointCrossover
GARealGenome::PartialMatchCrossover
GARealGenome::OrderCrossover
GARealGenome::CycleCrossover
```

Ustawienia domyślne

```
inicjalizacja:     GARealGenome::UniformInitializer
porównywanie:              GARealGenome::ElementComparator
mutacja:                   GARealGaussianMutator
krzyżowanie:               GARealGenome::UniformCrossover
```

GAStringGenome

```
typedef GAAlleleSet<char> GAStringAlleleSet
typedef GAAlleleSetCore<char> GAStringAlleleSetCore
typedef GAAlleleSetArray<char> GAStringAlleleSetArray
typedef GA1DArrayAlleleGenome<char> GAStringGenome
```

Konstruktory

```
GAStringGenome(unsigned int length, const GAStringAlleleSet &, GAGenome::Evaluator
objective = NULL, void * userData = NULL)
GAStringGenome(const GAStringAlleleSetArray &, GAGenome::Evaluator objective = NULL,
void * userData = NULL)
GAStringGenome(const GAStringGenome&))
```

Dozwolone operatory genetyczne

```
GAStringGenome::UniformInitializer
GAStringGenome::OrderedInitializer
GAStringGenome::FlipMutator
GAStringGenome::SwapMutator
GAStringGenome::UniformCrossover
GAStringGenome::EvenOddCrossover
GAStringGenome::OnePointCrossover
GAStringGenome::TwoPointCrossover
GAStringGenome::PartialMatchCrossover
GAStringGenome::OrderCrossover
GAStringGenome::CycleCrossover
```

Ustawienia domyślne

```
inicjalizacja:     GAStringGenome::UniformInitializer
porównywanie:              GAStringGenome::ElementComparator
mutacja:                   GAStringGenome::FlipMutator
krzyżowanie:               GAStringGenome::UniformCrossover
```

GABin2DecPhenotype

Konstruktory

```
GABin2DecPhenotype()
GABin2DecPhenotype(const GABin2DecPhenotype&)
```

Spis funkcji

```
void add(unsigned int nbits, float min, float max)
void remove(unsigned int which)
int size() const
int nPhenotypes() const
float min(unsigned int which) const
float max(unsigned int which) const
int length(unsigned int which) const
int offset(unsigned int which) const
void link(GABin2DecPhenotype&)
void unlink()
```

GAAlleleSet<T>

Konstruktory

```
GAAlleleSet()
GAAlleleSet(unsigned int n, const T a[])
GAAlleleSet(const T& lower, const T& upper, GAAllele::BoundType
lowerbound=GAAllele::INCLUSIVE, GAAllele::BoundType upperbound=GAAllele::INCLUSIVE)
GAAlleleSet(const T& lower, const T& upper, const T& increment, GAAllele::BoundType
lowerbound=GAAllele::INCLUSIVE, GAAllele::BoundType upperbound=GAAllele::INCLUSIVE)
GAAlleleSet(const GAAlleleSet<T>& set)
```

Spis funkcji

```
GAAlleleSet<T> * clone() const
T add(const T& allele)
T remove(T& allele)
T allele() const
T allele(unsigned int i)
int size() const
T lower() const
T upper() const
T inc() const
GAAllele::BoundType lowerBoundType() const
GAAllele::BoundType upperBoundType() const
GAAllele::Type type() const
void link(GAAlleleSet<T>&) void unlink()
```

GAAlleleSetArray<T>

Konstruktory

```
GAAlleleSetArray()
GAAlleleSetArray(const GAAlleleSet<T>&)
GAAlleleSetArray(const GAAlleleSetArray<T>&)
```

Spis funkcji

```
int size() const
const GAAlleleSet<T>& set(unsigned int i) const
int add(const GAAlleleSet<T>& s)
int add(unsigned int n, const T a[])
int add(const T& lower, const T& upper, GAAllele::BoundType lb=GAAllele::INCLUSIVE,
GAAllele::BoundType ub=GAAllele::INCLUSIVE)
int add(const T& lower, const T& upper, const T& increment, GAAllele::BoundType
lb=GAAllele::INCLUSIVE, GAAllele::BoundType ub=GAAllele::INCLUSIVE)
int remove(unsigned int)
```

GAStatistics

Typy i definicje

```
enum { NoScores, Mean, Maximum, Minimum, Deviation, Diversity, AllScores }
```

Spis funkcji

```
void copy(const GAStatistics &);
float online() const
float offline() const
float initial(ScoreID w=Maximum) const
float current(ScoreID w=Maximum) const
float worstEver() const
float bestEver() const
int generation() const
float convergence() const
int selections() const
int crossovers() const
int mutations() const
int replacements() const
int nConvergence(unsigned int) / const
int nBestGenomes(const GAGenome&, unsigned int) / const
int scoreFrequency(unsigned int x) / const
int flushFrequency(unsigned int x) / const
char* scoreFilename(const char *filename) / const
int selectScores(int whichScores) / const
GABoolean recordDiversity(GABoolean flag) / const
void flushScores()
void update(const GAPopulation& pop)
void reset(const GAPopulation& pop)
const GAPopulation& bestPopulation() const
const GAGenome& bestIndividual(unsigned int n=0) const
int scores(const char* filename, ScoreID which=NoScores)
int scores(ostream& os, ScoreID which=NoScores)
int write(const char* filename) const
int write(ostream& os) const;
ostream& operator<<(ostream&, const GAStatistics&)
```

GAPopulation

Typy i definicje

```
void (*GAPopulation::Initializer)(GAPopulation &)
void (*GAPopulation::Evaluator)(GAPopulation &)
enum SortBasis { RAW, SCALED };
enum SortOrder { LOW_IS_BEST, HIGH_IS_BEST };
enum Replacement { BEST = -1, WORST = -2, RANDOM = -3 };
```

Konstruktory

```
GAPopulation()
GAPopulation(const GAGenome&, unsigned int popsize = gaDefPopSize)
GAPopulation(const GAPopulation&)
```

Spis funkcji

```
GAPopulation * clone() const
void copy(const GAPopulation&)
int size(unsigned int popsize)/const
float sum() const
float ave() const
float var() const
float dev() const
float max() const
float min() const
float div() const
float div(unsigned int i, unsigned int j) const
float fitsum() const
float fitave() const
float fitmax() const
float fitmin() const
float fitvar() const
float fitdev() const
float psum(unsigned int i) const
```

GAPopulation (cd)

Spis funkcji (cd)

```
int nevals() const
void touch()
void statistics(GABoolean flag = gaFalse) const
void diversity(GABoolean flag = gaFalse) const
void prepselect(GABoolean flag = gaFalse) const
GAGenome& select()
GASelectionScheme& selector(const GASelectionScheme&)/const
void scale(GABoolean flag = gaFalse) const
GAScalingScheme& scaling(const GAScalingScheme&)/const
void sort(GABoolean flag = gaFalse, SortBasis basis = RAW) const
SortOrder order(SortOrder flag)/const
void evaluate(GABoolean flag = gaFalse) const
GAPopulation::Evaluator evaluator(GAPopulation::Evaluator func)
GAPopulation::Evaluator evaluator(GAPopulation::Evaluator func)
void initialize()
GAPopulation::Initializer initializer(GAPopulation::Initializer func)
GAGeneticAlgorithm * geneticAlgorithm(GA&)/const
void * userData(void * u)/const
GAEvalData * evalData(const GAEvalData&)/const
GAGenome& individual(unsigned int x, SortBasis basis = RAW) const
GAGenome& best(unsigned int i = 0, SortBasis basis = RAW) const
GAGenome& worst(unsigned int i = 0, SortBasis basis = RAW) const
GAGenome * add(GAGenome *)
GAGenome * add(const GAGenome&)
GAGenome * remove(unsigned int i, SortBasis basis = RAW)
GAGenome * remove(GAGenome *)
GAGenome * replace(GAGenome *, int which = gaPopReplaceRandom, SortBasis basis = RAW)
GAGenome * replace(GAGenome *, GAGenome *)
void destroy(int w = WORST, SortBasis basis = RAW)
virtual void read(istream &)
virtual void write(ostream &) const
ostream& operator<<(ostream &, const GAPopulation &)
istream& operator>>(istream &, GAPopulation &)
```

GAScalingScheme

Konstruktory

```
GAScalingScheme()
GAScalingScheme(const GAScalingScheme& s)
```

Spis funkcji

```
virtual GAScalingScheme * clone() const
virtual void copy(const GAScalingScheme &)
virtual void evaluate(const GAPopulation & p)
```

Wbudowane sposoby skalowania

```
GANoScaling()


GALinearScaling(float c = gaDefLinearScalingMultiplier)


GASigmaTruncationScaling(float c = gaDefSigmaTruncationMultiplier)


GAPowerLawScaling(int k = gaDefPowerScalingFactor)


GASharing(GAGenome::Comparator func=0,float cutoff = gaDefSharingCutoff, float alpha
= 1)
```

GASelectionScheme

Typy i definicje

```
enum { RAW, SCALED };
```

Konstruktory

```
GASelectionScheme(int which = FITNESS)
GASelectionScheme(const GASelectionScheme&)
```

Spis funkcji

```
virtual GASelectionScheme* clone() const;
virtual void copy(const GASelectionScheme& orig)
virtual void assign(GAPopulation& pop)
virtual void update()
virtual GAGenome& select() const;
```

Wbudowane sposoby selekcji

```
                                          // metoda
GARankSelector(int w=GASelectionScheme::SCALED)                                  //
rankingowa
GARouletteWheelSelector(int w=GASelectionScheme::SCALED)                  // ruletki
GATournamentSelector(int w=GASelectionScheme::SCALED)                     //
turniejowa
GADSSelector(int w=GASelectionScheme::SCALED)                                    //
deterministyczna
GASRSSelector(int w=GASelectionScheme::SCALED)                            // wg reszt
bez powtorzen
GAUniformSelector(int w=GASelectionScheme::SCALED)                        // czysto
losowa (brak

                                          // presji selekcyjnej)
```