Kozitsky Igor. Laboratory work №1

First scenario: probability of mutation is 1 percent

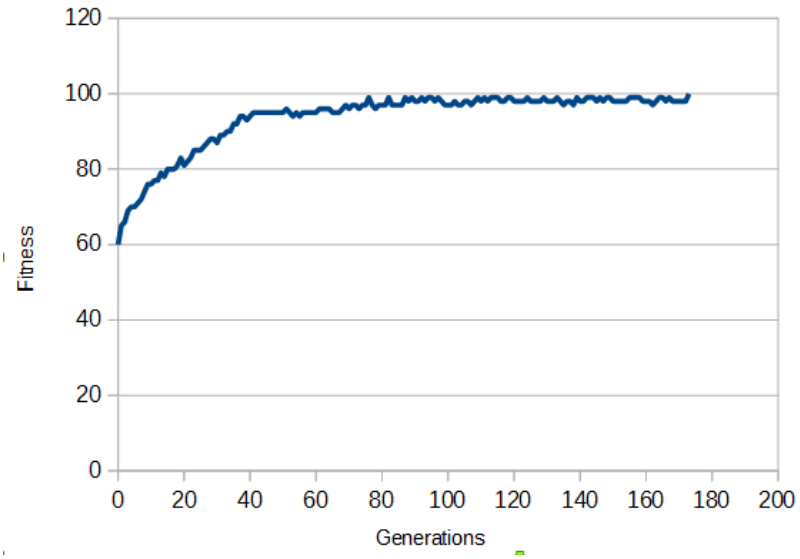
Code on C++:

```
#include "stdafx.h"
#include <iostream>
#include <vector>

using namespace std;

unsigned power(vector<bool> chromo);
vector<vector<bool>> selection(vector<vector<bool>> gen, unsigned number_of_chr);
vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen);
void mutation(vector<vector<bool>> &generation, unsigned procent);
void vector_output(vector<vector<bool>> vec);

bool check = false;

int main()
{

	unsigned number_of_gen = 100;
	unsigned number_of_chr = 100;
	unsigned limit = 1000;
	unsigned procent = 1;

	vector<vector<vector<bool>>> library;
	vector <vector<bool>> generation;

	for (int i = 0; i < number_of_chr; i++) {
		vector<bool> temp;
		for (int j = 0; j < number_of_gen; j++) {
			temp.push_back(rand() % 2);
		}
		generation.push_back(temp);
	}
	library.push_back(generation);
	cout << "Generation 0:" << endl;
	vector_output(generation);
	unsigned number = 1;

	while (true) {
		vector<vector<bool>> parents = selection(generation, number_of_chr);
		generation = evolution(parents, number_of_chr, number_of_gen);
		mutation(generation, procent);
		library.push_back(generation);
		cout << "Generation " << number << ":" << endl;
		number++;
		vector_output(generation);
		if (check == true || number == limit) break;
	}
    return 0;
}

unsigned power(vector<bool> chromo) {
	unsigned res = 0;
	for (int i = 0; i < chromo.size(); i++) {
		if (chromo[i] == true) res++;
	}
	return res;
```

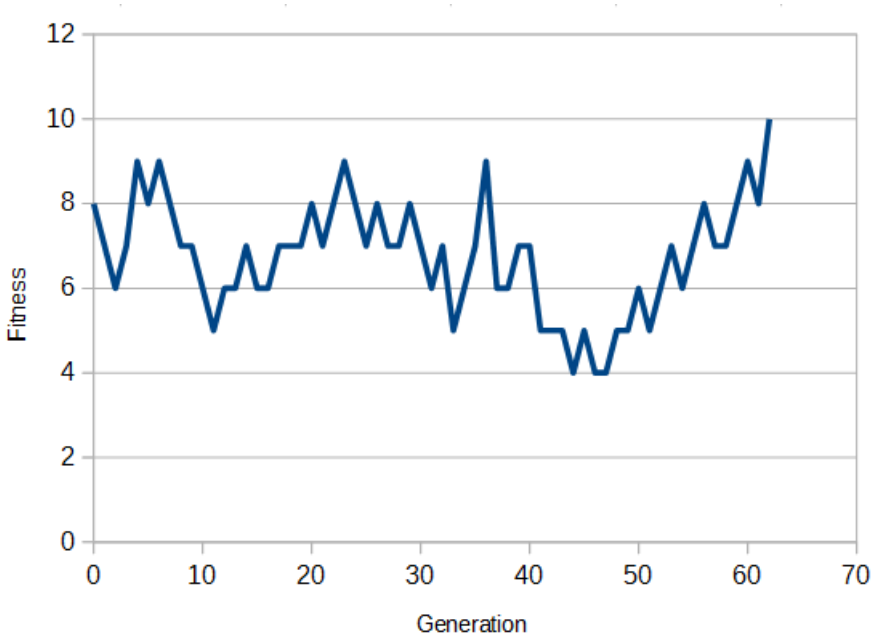
```
}

vector<vector<bool>> selection(vector<vector<bool>> gen, unsigned number_of_chr) {
	vector<vector<bool>> temp;
	vector<unsigned> numbers;
	vector<unsigned> values;
	for (int i = 0; i < gen.size(); i++) {
		values.push_back(power(gen[i]));
		numbers.push_back(i);
		if (values[i] == gen[i].size()) check = true;
	}
	for (int i = 0; i < gen.size(); i++) {
		for (int j = gen.size() - 1; j >= i; j--) {
			if (values[i] <= values[j] && i != j) {
				int T = values[i];
				values[i] = values[j];
				values[j] = T;
				numbers[i] = j;
				numbers[j] = i;
			}
		}
	}
	unsigned NumChromes = number_of_chr / 2;
	for (int i = 0; i < NumChromes; i++) {
		temp.push_back(gen[numbers[i]]);
	}
	return temp;
}

vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen) {
	vector<vector<bool>> result;
	for (int i = 0; i < par.size(); i++) {
		unsigned mother = rand() % par.size();
		unsigned father;
		while (true) {
			father = rand() % par.size();
			if (mother != father) break;
		}
		unsigned point = rand() % number_of_gen;
		vector<bool> child;
		for (int j = 0; j < number_of_gen; j++) {
			if (j <= point) child.push_back(par[mother][j]);
			else child.push_back(par[father][j]);
		}
		result.push_back(child);
	}
	return result;
}

void mutation(vector<vector<bool>> &generation, unsigned procent) {
	for (int i = 0; i < generation.size(); i++) {
		if (rand() % 100 + 1 >= procent) {
			int j = rand() % generation[i].size();
			if (generation[i][j] == 0) generation[i][j] = 1;
			else generation[i][j] = 0;
		}
	}
}

void vector_output(vector<vector<bool>> vec) {
	for (int i = 0; i < vec.size(); i++) {
		for (int j = 0; j < vec[i].size(); j++) {
			cout << vec[i][j];
		}
```

```
            cout << "    ";
            if (i % 2 == 0 && i != 0) cout << endl;
        }
        cout << endl;
}
```

Result with 100 genes and 100 chromosomes:



Result with 10 genes and 10 chromosomes:



Second scenario: probability of mutation is 50 percent

Code:

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <fstream>
```

```
using namespace std;

unsigned power(vector<bool> chromo);
vector<vector<bool>> selection(vector<vector<bool>> gen, unsigned number_of_chr);
vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen);
void mutation(vector<vector<bool>> &generation, unsigned procent);
void vector_output(vector<vector<bool>> vec);

bool check = false;

int main()
{

	unsigned number_of_gen = 100;
	unsigned number_of_chr = 100;
	unsigned limit = 1000;
	unsigned procent = 50;

	vector<vector<vector<bool>>> library;
	vector <vector<bool>> generation;

	for (int i = 0; i < number_of_chr; i++) {
		vector<bool> temp;
		for (int j = 0; j < number_of_gen; j++) {
			temp.push_back(rand() % 2);
		}
		generation.push_back(temp);
	}
	library.push_back(generation);
	cout << "Generation 0:" << endl;
	vector_output(generation);
	unsigned number = 1;
	vector<int> graph;

	while (true) {
		vector<vector<bool>> parents = selection(generation, number_of_chr);
		graph.push_back(power(parents[0]));
		generation = evolution(parents, number_of_chr, number_of_gen);
		mutation(generation, procent);
		library.push_back(generation);
		cout << "Generation " << number << ":" << endl;
		number++;
		vector_output(generation);
		if (check == true || number == limit) break;
	}

	ofstream file;
	file.open("D:\\Test1.txt");
	for (int i = 0; i < graph.size(); i++) {
		file << i << "	" << graph[i] << '\n';
	}
	file.close();
    return 0;
}

unsigned power(vector<bool> chromo) {
	unsigned res = 0;
	for (int i = 0; i < chromo.size(); i++) {
		if (chromo[i] == true) res++;
	}
	return res;
}

vector<vector<bool>> selection(vector<vector<bool>> gen, unsigned number_of_chr) {
```

```
	vector<vector<bool>> temp;
	vector<unsigned> numbers;
	vector<unsigned> values;
	for (int i = 0; i < gen.size(); i++) {
		values.push_back(power(gen[i]));
		numbers.push_back(i);
		if (values[i] == gen[i].size()) check = true;
	}
	for (int i = 0; i < gen.size(); i++) {
		for (int j = gen.size() - 1; j >= i; j--) {
			if (values[i] <= values[j] && i != j) {
				int T = values[i];
				values[i] = values[j];
				values[j] = T;
				numbers[i] = j;
				numbers[j] = i;
			}
		}
	}
	unsigned NumChromes = number_of_chr / 2;
	for (int i = 0; i < NumChromes; i++) {
		temp.push_back(gen[numbers[i]]);
	}
	return temp;
}

vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen) {
	vector<vector<bool>> result;
	for (int i = 0; i < par.size(); i++) {
		unsigned mother = rand() % par.size();
		unsigned father;
		while (true) {
			father = rand() % par.size();
			if (mother != father) break;
		}
		unsigned point = rand() % number_of_gen;
		vector<bool> child;
		for (int j = 0; j < number_of_gen; j++) {
			if (j <= point) child.push_back(par[mother][j]);
			else child.push_back(par[father][j]);
		}
		result.push_back(child);
	}
	return result;
}

void mutation(vector<vector<bool>> &generation, unsigned procent) {
	for (int i = 0; i < generation.size(); i++) {
		if (rand() % 100 + 1 >= procent) {
			int j = rand() % generation[i].size();
			if (generation[i][j] == 0) generation[i][j] = 1;
			else generation[i][j] = 0;
		}
	}
}

void vector_output(vector<vector<bool>> vec) {
	for (int i = 0; i < vec.size(); i++) {
		for (int j = 0; j < vec[i].size(); j++) {
			cout << vec[i][j];
		}
		cout << "   ";
		if (i % 2 == 0 && i != 0) cout << endl;
	}
```

```
        cout << endl;
}
```

Result: