// C++ program to create target string, starting from

// random string using Genetic Algorithm

#include <bits/stdc++.h>

using namespace std;

// Number of individuals in each generation

#define POPULATION\_SIZE 100

// Valid Genes

const string GENES = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP"\

"QRSTUVWXYZ 1234567890, .-;:\_!\"#%&/()=?@${[]}";

// Target string to be generated

const string TARGET = "I love GeeksforGeeks";

// Function to generate random numbers in given range

int random\_num(int start, int end)

{

int range = (end-start)+1;

int random\_int = start+(rand()%range);

return random\_int;

}

// Create random genes for mutation

char mutated\_genes()

{

int len = GENES.size();

int r = random\_num(0, len-1);

return GENES[r];

}

// create chromosome or string of genes

string create\_gnome()

{

int len = TARGET.size();

string gnome = "";

for(int i = 0;i<len;i++)

gnome += mutated\_genes();

return gnome;

}

// Class representing individual in population

class Individual

{

public:

string chromosome;

int fitness;

Individual(string chromosome);

Individual mate(Individual parent2);

int cal\_fitness();

};

Individual::Individual(string chromosome)

{

this->chromosome = chromosome;

fitness = cal\_fitness();

};

// Perform mating and produce new offspring

Individual Individual::mate(Individual par2)

{

// chromosome for offspring

string child\_chromosome = "";

int len = chromosome.size();

for(int i = 0;i<len;i++)

{

// random probability

float p = random\_num(0, 100)/100;

// if prob is less than 0.45, insert gene

// from parent 1

if(p < 0.45)

child\_chromosome += chromosome[i];

// if prob is between 0.45 and 0.90, insert

// gene from parent 2

else if(p < 0.90)

child\_chromosome += par2.chromosome[i];

// otherwise insert random gene(mutate),

// for maintaining diversity

else

child\_chromosome += mutated\_genes();

}

// create new Individual(offspring) using

// generated chromosome for offspring

return Individual(child\_chromosome);

};

// Calculate fittness score, it is the number of

// characters in string which differ from target

// string.

int Individual::cal\_fitness()

{

int len = TARGET.size();

int fitness = 0;

for(int i = 0;i<len;i++)

{

if(chromosome[i] != TARGET[i])

fitness++;

}

return fitness;

};

// Overloading < operator

bool operator<(const Individual &ind1, const Individual &ind2)

{

return ind1.fitness < ind2.fitness;

}

// Driver code

int main()

{

srand((unsigned)(time(0)));

// current generation

int generation = 0;

vector<Individual> population;

bool found = false;

// create initial population

for(int i = 0;i<POPULATION\_SIZE;i++)

{

string gnome = create\_gnome();

population.push\_back(Individual(gnome));

}

while(! found)

{

// sort the population in increasing order of fitness score

sort(population.begin(), population.end());

// if the individual having lowest fitness score ie.

// 0 then we know that we have reached to the target

// and break the loop

if(population[0].fitness <= 0)

{

found = true;

break;

}

// Otherwise generate new offsprings for new generation

vector<Individual> new\_generation;

// Perform Elitism, that mean 10% of fittest population

// goes to the next generation

int s = (10\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

new\_generation.push\_back(population[i]);

// From 50% of fittest population, Individuals

// will mate to produce offspring

s = (90\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

{

int len = population.size();

int r = random\_num(0, 50);

Individual parent1 = population[r];

r = random\_num(0, 50);

Individual parent2 = population[r];

Individual offspring = parent1.mate(parent2);

new\_generation.push\_back(offspring);

}

population = new\_generation;

cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

generation++;

}

cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

}

Output

Generation: 1 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

Generation: 2 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

Generation: 3 String: .#lRWf9k\_Ifslw #O$k\_ Fitness: 17

Generation: 4 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

Generation: 5 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

Generation: 6 String: A#ldW) #lIkslw cVek) Fitness: 14

Generation: 7 String: A#ldW) #lIkslw cVek) Fitness: 14

Generation: 8 String: (, o x \_x%Rs=, 6Peek3 Fitness: 13

.

.

.

Generation: 29 String: I lope Geeks#o, Geeks Fitness: 3

Generation: 30 String: I loMe GeeksfoBGeeks Fitness: 2

Generation: 31 String: I love Geeksfo0Geeks Fitness: 1

Generation: 32 String: I love Geeksfo0Geeks Fitness: 1

Generation: 33 String: I love Geeksfo0Geeks Fitness: 1

Generation: 34 String: I love GeeksforGeeks Fitness: 0