SILESIAN UNIVERSITY OF TECHNOLOGY

Faculty of Automatic control, Electronics and Informatics

Fundamentals of computer programming

«Darwin»

Author: **Youssef AL BALI**

Instructor: **dr inż. Sławomir Lasota**

Year: 2021/2022

1 Project topic

Write a program for simulating of evolution of individuals in a population. The population may have any number of individuals. Each individual has a chromosome that is a sequence of integer numbers. The lengths of chromosomes may differ. In each generation k pairs of individuals are taken at random. Individuals of these pairs are then crossed with a crossing-over operator. That means that in each individual its chromosome is broken at random into left and right parts. The left part of the first individual is joined with the right part of the second one and the left part of the second individual is joined with the right part of the first one. One individual may be present in many crossing-over pairs. After crossing-over a fitness $f \in [0,1]$ of each individual is elaborated with a fitness function. Individuals with $f < w$ (w is an extinction threshold) are removed from the population. Individuals with $f > r$ (r is a proliferation threshold) are cloned. Individuals with $w \leq f \leq r$ are not modified.

This is a command line program with switches:

`-i` input file with a population
`-o` output file with a population
`-w` extinction threshold $w \in |0,1|$
`-r` proliferation threshold $r \in [0,1]$
`-p` number of generations p
`-k` number k of pair to cross-over

Running the program with no parameters prints a short help how to use the program.

Input file format: Each line holds a chromosome of one individual as a sequence of integer numbers separated with white spaces. E.g., file with four individuals

```
2 9 84 9 5 6 25 12
2 98 56 2 54
5 2
8 5 22 5 48 6 1 9 8 7 554 25 235 32
```

Output file format is exactly the same.

2 Analysis of the task

This task focuses on creating a new population by crossing-over the individuals and fitness-check them and based on the return value of the fitness function, the individual is either cloned or removed.

2.1 Data structures

Containers:

Vectors: To hold the individuals (vector <individual>) and also to store chromosomes of each individual (vector<int>).

Examples:

```
std::vector<Individual> population; /**< Vector of individuals */

vector<int> chromosomes; /**< Vector of chromosomes for each Individual  */
```

Classes:

Population: It has a vector of type individual to keep track of the current and the future generated population, its methods take care of reading and writing to the output file, this class has as well some functions to cross-over and fitness check all the population.

Individual: Contains a vector of chromosomes of type int, the two important methods of this class are fitness-check (returns a double value) and the overloaded output operator which prints which can print the chromosomes of each individual in a separate line (We will use this operator to write to the output file).

2.2 Algorithms and the program overview

We can divide this program into different small tasks as the following:

1. Read from the input file.
2. Crossing over.
3. Fitness check.
4. Write to the output file.

To read from the input file:

Right after creating an instance of class population, we are calling one of its functions which read the individuals from the input file and assigns them to the vector POPULATION.

```
_population.read_population_from_file( filename: input_filename);
```

Input data example:

```
11 18 15 26 10 28 29 30 3 6 24 16 16 29 14 12 28 13 18 24
24 17 16 6 13 14 26 15 18 21 22 23 26 15 16
13 18 24
24 25 25 11 26 19 16 23 24 4 28 26 27 20 21 11
18 14 27 19 12 13 26 25 12 12 29 20 21 24
24 19 13 14 15 2 3 4 5 6 7 8 9 10
17 24 24 7 8 9 13
```

To crossover the individuals:

We are calling our cross-over function, which will crossover a number of pairs (K) from the population and assign the results to the POPULATION vector.

```
_population.crossingover( k: NUMBER_OF_PAIRS);
```

In the beginning this method generates random integers concerning the individuals that will be crossed as well the position when the chromosomes will get broke by calling an existing method:

```
static int randomInteger(int begin, int end);
```

which generates a random integer on a specific range.
After that we insert both of the left and the right part into a temporary vector of integers
For example, the insertion of the first part:

```
for (int j = 0; j < chromosomes_divider_1; j++) {
    tmp.emplace_back(population[index1].chromosomes[j]);
}
```

Each iteration the mentioned temporary vector is emplaced back into a vector of type individual which will be moved in the end of the function into POPULATION.

To fitness check the population:

The fitness check function of population will take action it takes two arguments of type double (extinction threshold, proliferation threshold).

```
_population.fitnesscheckpopulation( extinction_threshold: EXTINCTION_THRESHOLD,  proliferation_threshold: PROLIFERATION_THRESHOLD);
```

The function will iterate through all the individuals, in each iteration there is a call of fitness-check which is a method of our individual class that returns a double, based on this return value, we will either clone the individual using “emplace_back”, or remove it using “erase”:

```
population.emplace_back(population[i]);
```

```
population.erase( position: population.begin() + i);
```

Otherwise, if the fitness value of the individual is between the extinction threshold and proliferation threshold, then nothing is performed.

To Write the final Population to the output file

We are finally calling the method of our Population class that takes care of writing the individuals to the file by iterating through the POPULATION vector and writing the chromosomes of each individual in it to the output file with the help of our overloaded output operator (an operator of our individual class).

```
_population.writeToFile( OutputFile: output_filename);
```

```
/**
* output overload operator
* @param os output stream
* @param individual object of class Individual
*/
friend ostream &operator<<(ostream &os, const Individual &ind);
```

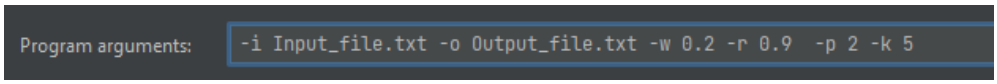
Output Data Example:

```
24 19 13 14 15 2 3 4 7 8 9 10 24
2 15 2 24
31 22 23 12 14 3 4 5 6 7 8 9 10
12 18 2 3 4 5 6 10
13 27 14 14 15 27 28 5 6 7 8 9 10 22 23 19 25 27 26 27 28 301 4 25 20 21 21 301 2 3 4 5 6 7 8 9 10
24 14 24
```

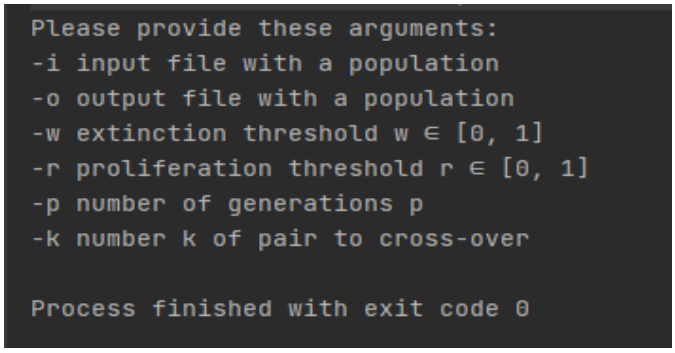
3 External specification

The Program is launched in command line. It requires to type the name of the input file after the -i switch, the output file after the -o switch, proliferation threshold after -r switch, extinction threshold after the -w switch, the number of pairs to cross-over after the -k switch and finally the number of generations after the switch -p.

Here is an input example:



The program called without parameters will print a short manual:



Program called with a wrong input file will print the following message:

```
Failed to open input file: input.txt
```

Program called with an empty input file will print the following message:

```
The input file text.txt is empty
```

4 Internal specification - 4.1 Description of types and functions

Description od types and functions are moved to the appendix

5 Testing

The program has been tested on .txt files with different and various input data (Large, small, empty…)

Handling errors:

The program will notify you if you:

- Entered too many or too little arguments;
- Entered a non-existing input file;
- Want to read data from an empty file

6 Conclusion:

During the implementation of this program there was some issues due to a memory leak while copying to a vector of type individual*. That's why I decided to make the vector POPULATION of type <individual> since the usage of pointers was not quite needed in this case. I was also able to change the push back method to emplace back and it's more efficient as we are not creating new individuals.