HACETTEPE UNIVERSITY

BBM 203

ASSIGNMENT III

Name & Surname : Anıl Helvacı

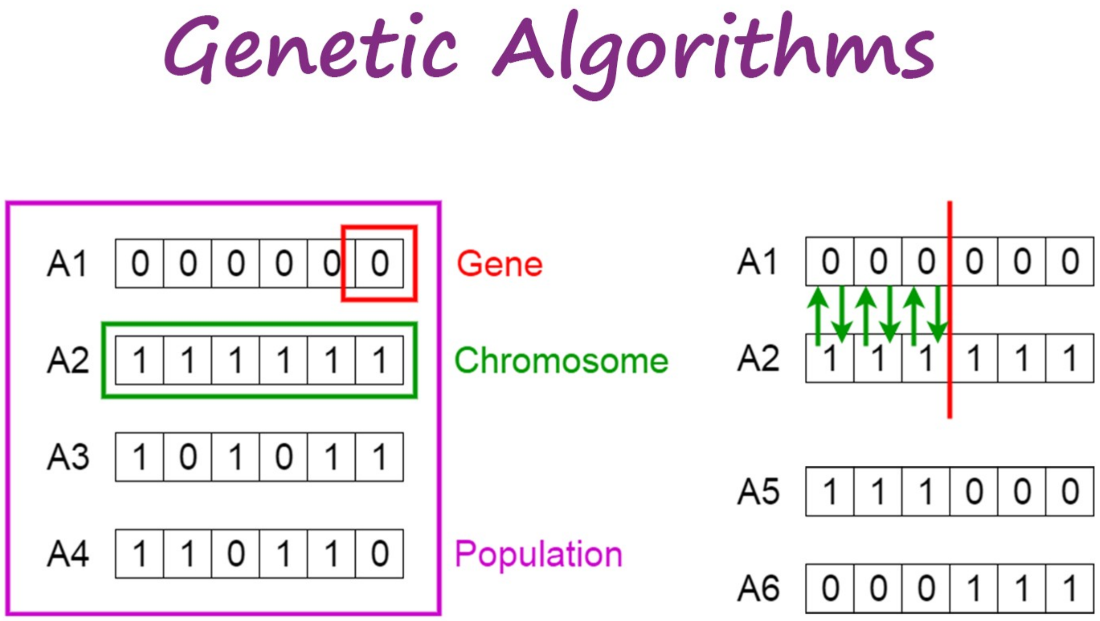
Student ID : 21527084

TA : Selim Yılmaz

Task : Implementation of Genetic Algorithm

Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin’s theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.



The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

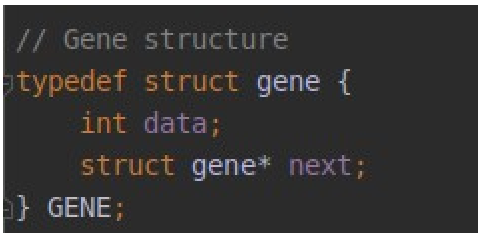
Five phases are considered in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Implementation

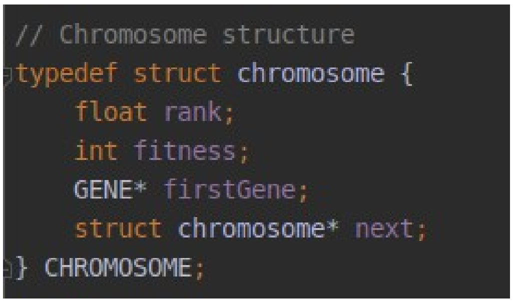
I started my implementation by defining the following structures

- Gene



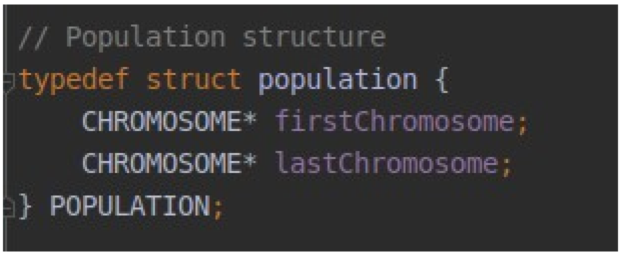
```
// Gene structure
typedef struct gene {
    int data;
    struct gene* next;
} GENE;
```

- Chromosome



```
// Chromosome structure
typedef struct chromosome {
    float rank;
    int fitness;
    GENE* firstGene;
    struct chromosome* next;
} CHROMOSOME;
```

- Population

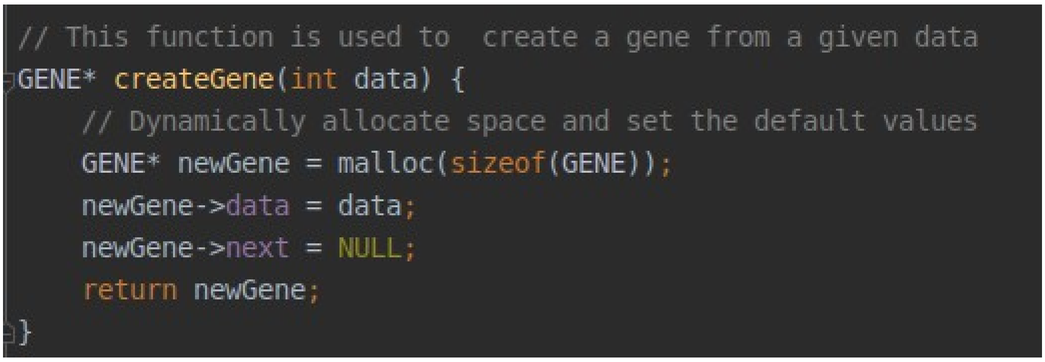


```
// Population structure
typedef struct population {
    CHROMOSOME* firstChromosome;
    CHROMOSOME* lastChromosome;
} POPULATION;
```

Initialization

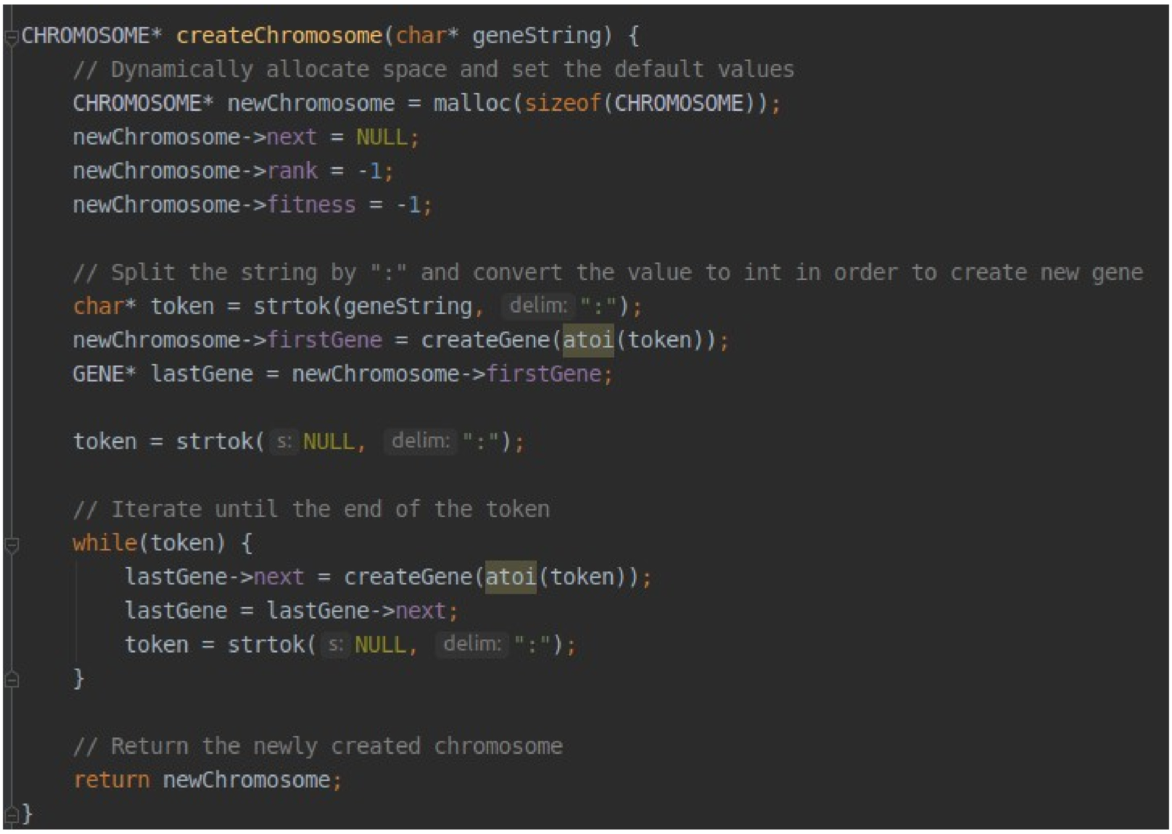
In order to complete the initialization part I had to allocate space for the genes, chromosomes and the population with the default values. Then build the linked-lists to represent the relations between genes, chromosomes and generations. Every gene has a pointer to the next gene in the chromosome. Every chromosome has a pointer to the next chromosome in the population. And the population only has a pointer to the first and the last chromosome that are present. I stored the last pointer for ease in adding new chromosome instead of iterating all chromosomes until the one that points Null in its next pointer. The following screenshots belong to the functions that performs the initialization tasks mentioned above;

- Create Gene



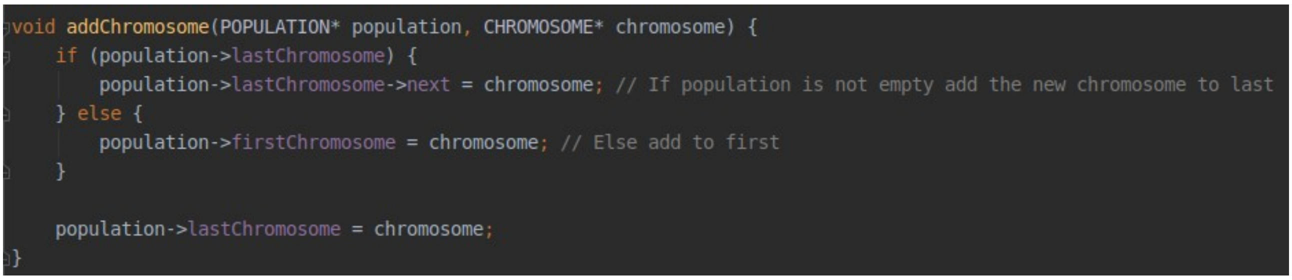
```
// This function is used to  create a gene from a given data
GENE* createGene(int data) {
    // Dynamically allocate space and set the default values
    GENE* newGene = malloc(sizeof(GENE));
    newGene->data = data;
    newGene->next = NULL;
    return newGene;
}
```

- Create Chromosome



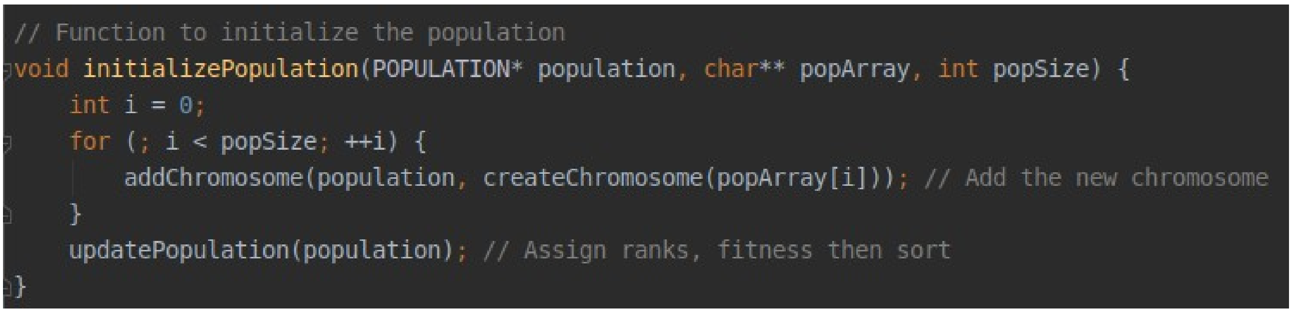
```
CHROMOSOME* createChromosome(char* geneString) {
    // Dynamically allocate space and set the default values
    CHROMOSOME* newChromosome = malloc(sizeof(CHROMOSOME));
    newChromosome->next = NULL;
    newChromosome->rank = -1;
    newChromosome->fitness = -1;

    // Split the string by ":" and convert the value to int in order to create new gene
    char* token = strtok(geneString, delim: ":");
    newChromosome->firstGene = createGene(atoi(token));
    GENE* lastGene = newChromosome->firstGene;

    token = strtok( s: NULL, delim: ":");

    // Iterate until the end of the token
    while(token) {
        lastGene->next = createGene(atoi(token));
        lastGene = lastGene->next;
        token = strtok( s: NULL, delim: ":");
    }

    // Return the newly created chromosome
    return newChromosome;
}
```

- Add Chromosome



```
void addChromosome(POPULATION* population, CHROMOSOME* chromosome) {
    if (population->lastChromosome) {
        population->lastChromosome->next = chromosome; // If population is not empty add the new chromosome to last
    } else {
        population->firstChromosome = chromosome; // Else add to first
    }

    population->lastChromosome = chromosome;
}
```

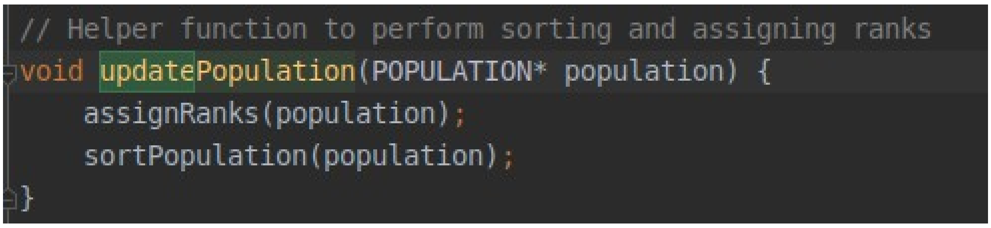
- Initialize Population



```
// Function to initialize the population
void initializePopulation(POPULATION* population, char** popArray, int popSize) {
    int i = 0;
    for (; i < popSize; ++i) {
        addChromosome(population, createChromosome(popArray[i])); // Add the new chromosome
    }
    updatePopulation(population); // Assign ranks, fitness then sort
}
```

initializePopulation() function iterates the given “population” file and adds the created chromosome via createChromosome and addChromosome methods. Calculates the fitness and rank values then sorts them by the fitness value via via the updatePopulation function.

- Update Population

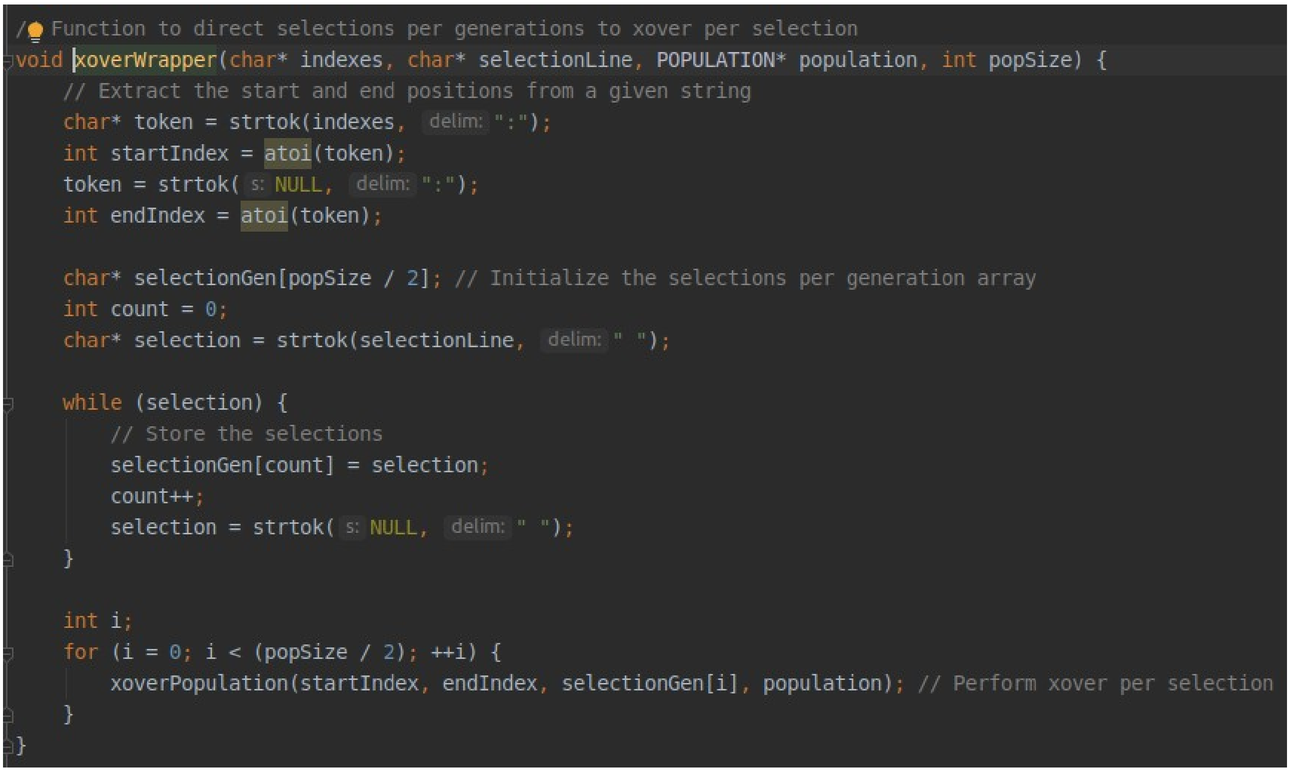


```
// Helper function to perform sorting and assigning ranks
void updatePopulation(POPULATION* population) {
    assignRanks(population);
    sortPopulation(population);
}
```

Selection & Crossover

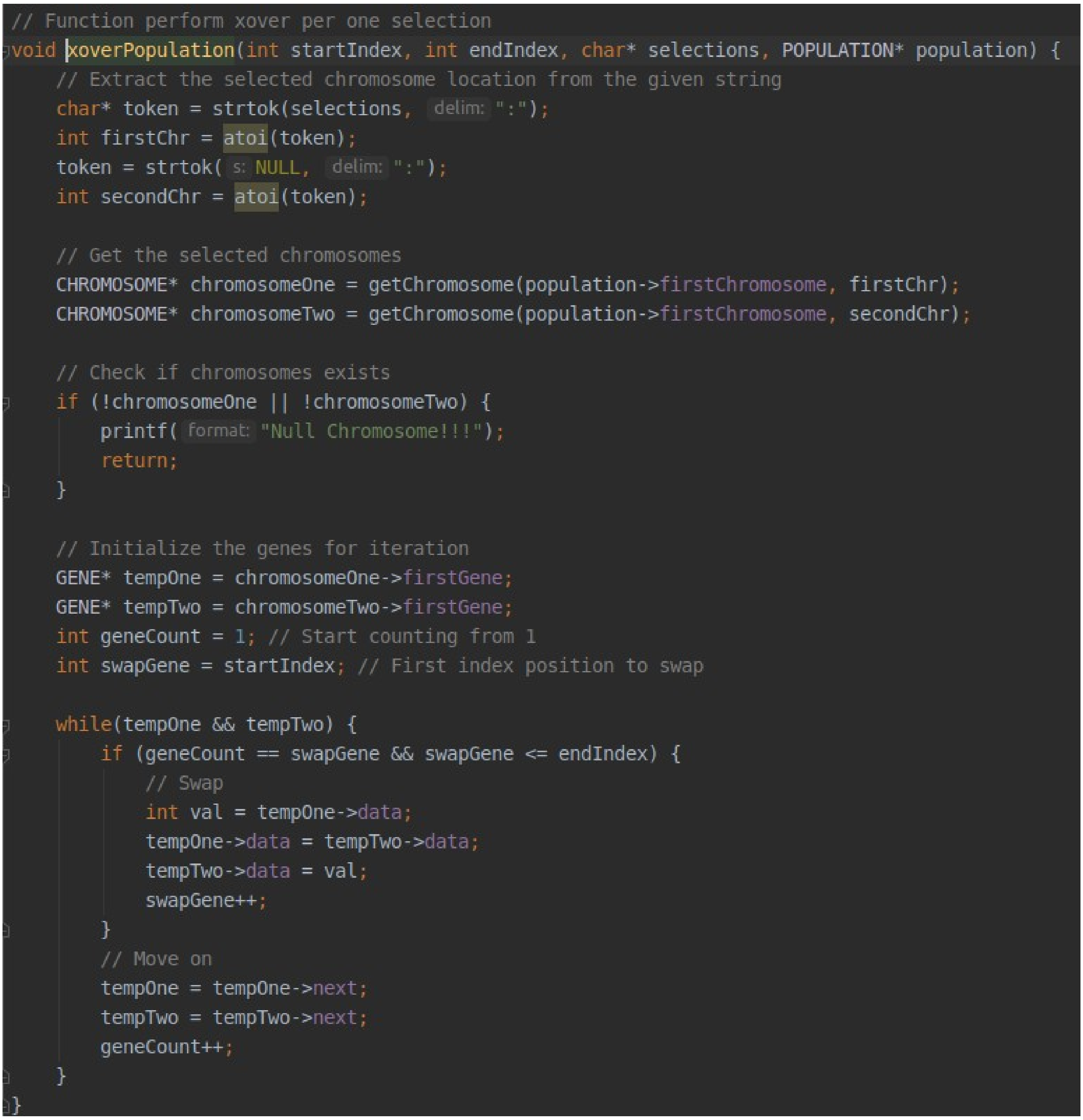
In my implementation selection is used to determine which chromosomes will perform crossover operation. As a result selection is part of the crossover process which is the reason there are no separate functions to perform selection.

- **xoverWrapper** function is the entry point to the **selection & crossover** process during iteration of the producing new generations. It gets the string data from files then forwars them to the function which performs the actual crossover operation. Below is the screenshot of this function;



```
// Function to direct selections per generations to xover per selection
void xoverWrapper(char* indexes, char* selectionLine, POPULATION* population, int popSize) {
    // Extract the start and end positions from a given string
    char* token = strtok(indexes,  delim: ":");
    int startIndex = atoi(token);
    token = strtok( s: NULL,  delim: ":");
    int endIndex = atoi(token);

    char* selectionGen[popSize / 2]; // Initialize the selections per generation array
    int count = 0;
    char* selection = strtok(selectionLine,  delim: " ");

    while (selection) {
        // Store the selections
        selectionGen[count] = selection;
        count++;
        selection = strtok( s: NULL,  delim: " ");
    }

    int i;
    for (i = 0; i < (popSize / 2); ++i) {
        xoverPopulation(startIndex, endIndex, selectionGen[i], population); // Perform xover per selection
    }
}
```

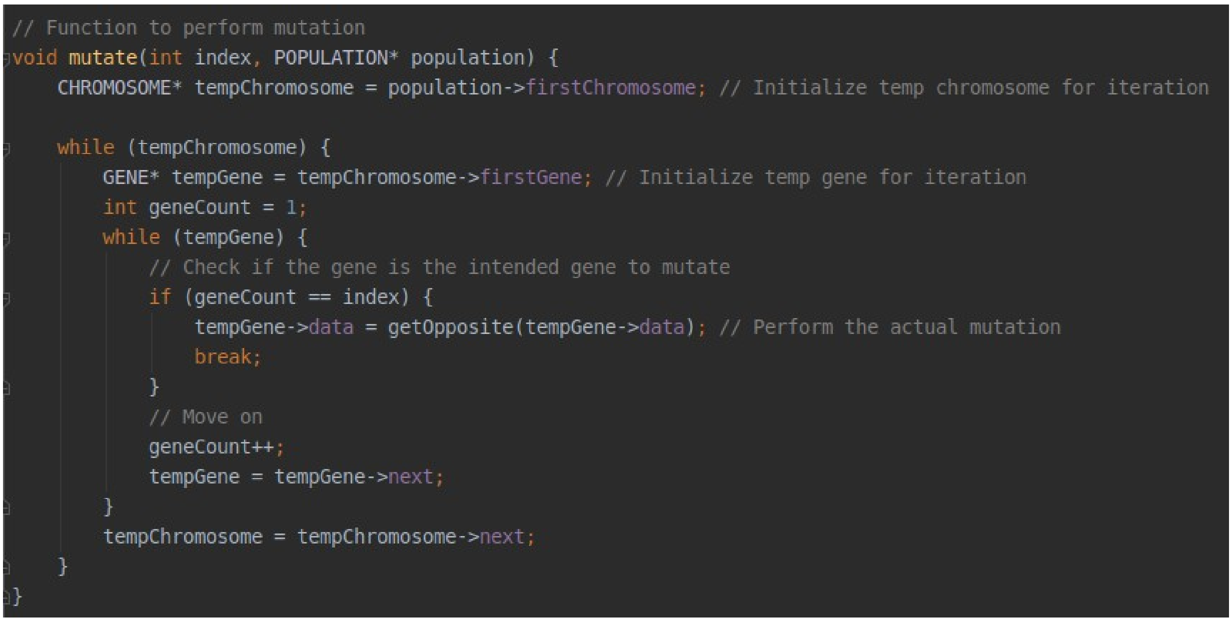
- **xoverPopulation** function fetches the selected chromosomes, check if they are null, then swaps the data between corresponding genes of the chromosomes. Below is a screenshot of the mentioned function;



```c
// Function perform xover per one selection
void xoverPopulation(int startIndex, int endIndex, char* selections, POPULATION* population) {
    // Extract the selected chromosome location from the given string
    char* token = strtok(selections, delim: ":");
    int firstChr = atoi(token);
    token = strtok( s: NULL, delim: ":");
    int secondChr = atoi(token);

    // Get the selected chromosomes
    CHROMOSOME* chromosomeOne = getChromosome(population->firstChromosome, firstChr);
    CHROMOSOME* chromosomeTwo = getChromosome(population->firstChromosome, secondChr);

    // Check if chromosomes exists
    if (!chromosomeOne || !chromosomeTwo) {
        printf( format: "Null Chromosome!!!");
        return;
    }

    // Initialize the genes for iteration
    GENE* tempOne = chromosomeOne->firstGene;
    GENE* tempTwo = chromosomeTwo->firstGene;
    int geneCount = 1; // Start counting from 1
    int swapGene = startIndex; // First index position to swap

    while(tempOne && tempTwo) {
        if (geneCount == swapGene && swapGene <= endIndex) {
            // Swap
            int val = tempOne->data;
            tempOne->data = tempTwo->data;
            tempTwo->data = val;
            swapGene++;
        }
        // Move on
        tempOne = tempOne->next;
        tempTwo = tempTwo->next;
        geneCount++;
    }
}
```

Mutation

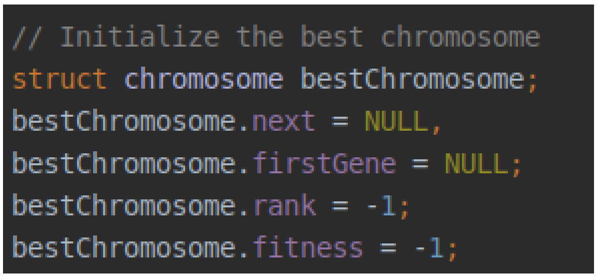
- This operation is done by the **mutate** function which iterates every chromosome until it reaches the gene to be mutated. A simple screenshot is below;



```
// Function to perform mutation
void mutate(int index, POPULATION* population) {
    CHROMOSOME* tempChromosome = population->firstChromosome; // Initialize temp chromosome for iteration

    while (tempChromosome) {
        GENE* tempGene = tempChromosome->firstGene; // Initialize temp gene for iteration
        int geneCount = 1;
        while (tempGene) {
            // Check if the gene is the intended gene to mutate
            if (geneCount == index) {
                tempGene->data = getOpposite(tempGene->data); // Perform the actual mutation
                break;
            }
            // Move on
            geneCount++;
            tempGene = tempGene->next;
        }
        tempChromosome = tempChromosome->next;
    }
}
```

Best Chromosome

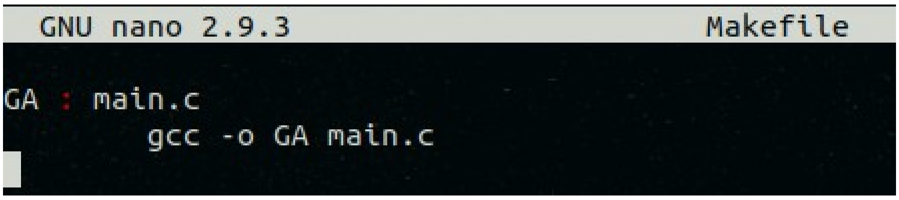
A separate chromosome is stored in main as the best chromosome and updated in every generation if the best fitness value of the new generation is better than the one found so far.



```
// Initialize the best chromosome
struct chromosome bestChromosome;
bestChromosome.next = NULL;
bestChromosome.firstGene = NULL;
bestChromosome.rank = -1;
bestChromosome.fitness = -1;
```

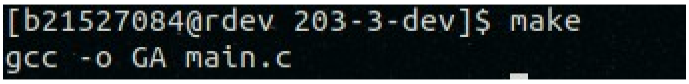
Running

Content of The Makefile



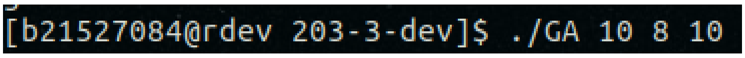
```
  GNU nano 2.9.3                         Makefile

GA : main.c
        gcc -o GA main.c
```

Compile



```
[b21527084@rdev 203-3-dev]$ make
gcc -o GA main.c
```

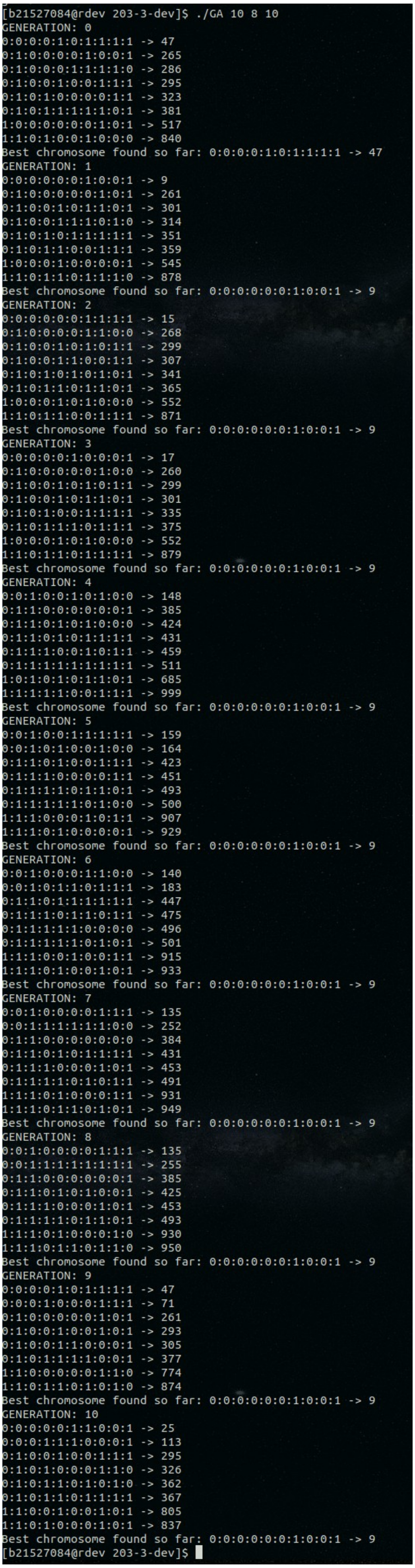
Run



```
[b21527084@rdev 203-3-dev]$ ./GA 10 8 10
```

Result

Please see the last page.



```
[b21527084@rdev 203-3-dev]$ ./GA 10 8 10
GENERATION: 0
0:0:0:0:1:0:1:1:1:1 -> 47
0:1:0:0:0:0:1:0:0:1 -> 265
0:1:0:0:0:1:1:1:1:0 -> 286
0:1:0:0:1:0:0:1:1:1 -> 295
0:1:0:1:0:0:0:0:1:1 -> 323
0:1:0:1:1:1:1:1:0:1 -> 381
1:0:0:0:0:0:0:1:0:1 -> 517
1:1:0:1:0:0:1:0:0:0 -> 840
Best chromosome found so far: 0:0:0:0:1:0:1:1:1:1 -> 47
GENERATION: 1
0:0:0:0:0:0:1:0:0:1 -> 9
0:1:0:0:0:0:0:1:0:1 -> 261
0:1:0:0:1:0:1:1:0:1 -> 301
0:1:0:0:1:1:1:0:1:0 -> 314
0:1:0:1:0:1:1:1:1:1 -> 351
0:1:0:1:1:0:0:1:1:1 -> 359
1:0:0:0:1:0:0:0:0:1 -> 545
1:1:0:1:1:0:1:1:1:0 -> 878
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 2
0:0:0:0:0:0:1:1:1:1 -> 15
0:1:0:0:0:0:1:1:0:0 -> 268
0:1:0:0:1:0:1:0:1:1 -> 299
0:1:0:0:1:1:0:0:1:1 -> 307
0:1:0:1:0:1:0:1:0:1 -> 341
0:1:0:1:1:0:1:1:0:1 -> 365
1:0:0:0:1:0:1:0:0:0 -> 552
1:1:0:1:1:0:0:1:1:1 -> 871
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 3
0:0:0:0:0:1:0:0:0:1 -> 17
0:1:0:0:0:0:0:1:0:0 -> 260
0:1:0:0:1:0:1:0:1:1 -> 299
0:1:0:0:1:0:1:1:0:1 -> 301
0:1:0:1:0:0:1:1:1:1 -> 335
0:1:0:1:1:1:0:1:1:1 -> 375
1:0:0:0:1:0:1:0:0:0 -> 552
1:1:0:1:1:0:1:1:1:1 -> 879
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 4
0:0:1:0:0:1:0:1:0:0 -> 148
0:1:1:0:0:0:0:0:0:1 -> 385
0:1:1:0:1:0:1:0:0:0 -> 424
0:1:1:0:1:0:1:1:1:1 -> 431
0:1:1:1:0:0:1:0:1:1 -> 459
0:1:1:1:1:1:1:1:1:1 -> 511
1:0:1:0:1:0:1:1:0:1 -> 685
1:1:1:1:1:0:0:1:1:1 -> 999
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 5
0:0:1:0:0:1:1:1:1:1 -> 159
0:0:1:0:1:0:0:1:0:0 -> 164
0:1:1:0:1:0:0:1:1:1 -> 423
0:1:1:1:0:0:0:0:1:1 -> 451
0:1:1:1:1:0:1:1:0:1 -> 493
0:1:1:1:1:1:0:1:0:0 -> 500
1:1:1:0:0:0:1:0:1:1 -> 907
1:1:1:0:1:0:0:0:0:1 -> 929
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 6
0:0:1:0:0:0:1:1:0:0 -> 140
0:0:1:0:1:1:0:1:1:1 -> 183
0:1:1:0:1:1:1:1:1:1 -> 447
0:1:1:1:0:1:1:0:1:1 -> 475
0:1:1:1:1:1:0:0:0:0 -> 496
0:1:1:1:1:1:0:1:0:1 -> 501
1:1:1:0:0:1:0:0:1:1 -> 915
1:1:1:0:1:0:0:1:0:1 -> 933
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 7
0:0:1:0:0:0:0:1:1:1 -> 135
0:0:1:1:1:1:1:1:0:0 -> 252
0:1:1:0:0:0:0:0:0:0 -> 384
0:1:1:0:1:0:1:1:1:1 -> 431
0:1:1:1:0:0:0:1:0:1 -> 453
0:1:1:1:1:0:1:0:1:1 -> 491
1:1:1:0:1:0:0:0:1:1 -> 931
1:1:1:0:1:1:0:1:0:1 -> 949
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 8
0:0:1:0:0:0:0:1:1:1 -> 135
0:0:1:1:1:1:1:1:1:1 -> 255
0:1:1:0:0:0:0:0:0:1 -> 385
0:1:1:0:1:0:1:0:0:1 -> 425
0:1:1:1:0:0:0:1:0:1 -> 453
0:1:1:1:1:0:1:1:0:1 -> 493
1:1:1:0:1:0:0:0:1:0 -> 930
1:1:1:0:1:1:0:1:1:0 -> 950
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 9
0:0:0:0:1:0:1:1:1:1 -> 47
0:0:0:1:0:0:0:1:1:1 -> 71
0:1:0:0:0:0:0:1:0:1 -> 261
0:1:0:0:1:0:0:1:0:1 -> 293
0:1:0:0:1:1:0:0:0:1 -> 305
0:1:0:1:1:1:1:0:0:1 -> 377
1:1:0:0:0:0:0:1:1:0 -> 774
1:1:0:1:1:0:1:0:1:0 -> 874
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 10
0:0:0:0:0:1:1:0:0:1 -> 25
0:0:0:1:1:1:0:0:0:1 -> 113
0:1:0:0:1:0:0:1:1:1 -> 295
0:1:0:1:0:0:0:1:1:0 -> 326
0:1:0:1:1:0:1:0:1:0 -> 362
0:1:0:1:1:0:1:1:1:1 -> 367
1:1:0:0:1:0:0:1:0:1 -> 805
1:1:0:1:0:0:0:1:0:1 -> 837
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
[b21527084@rdev 203-3-dev]$ 
```