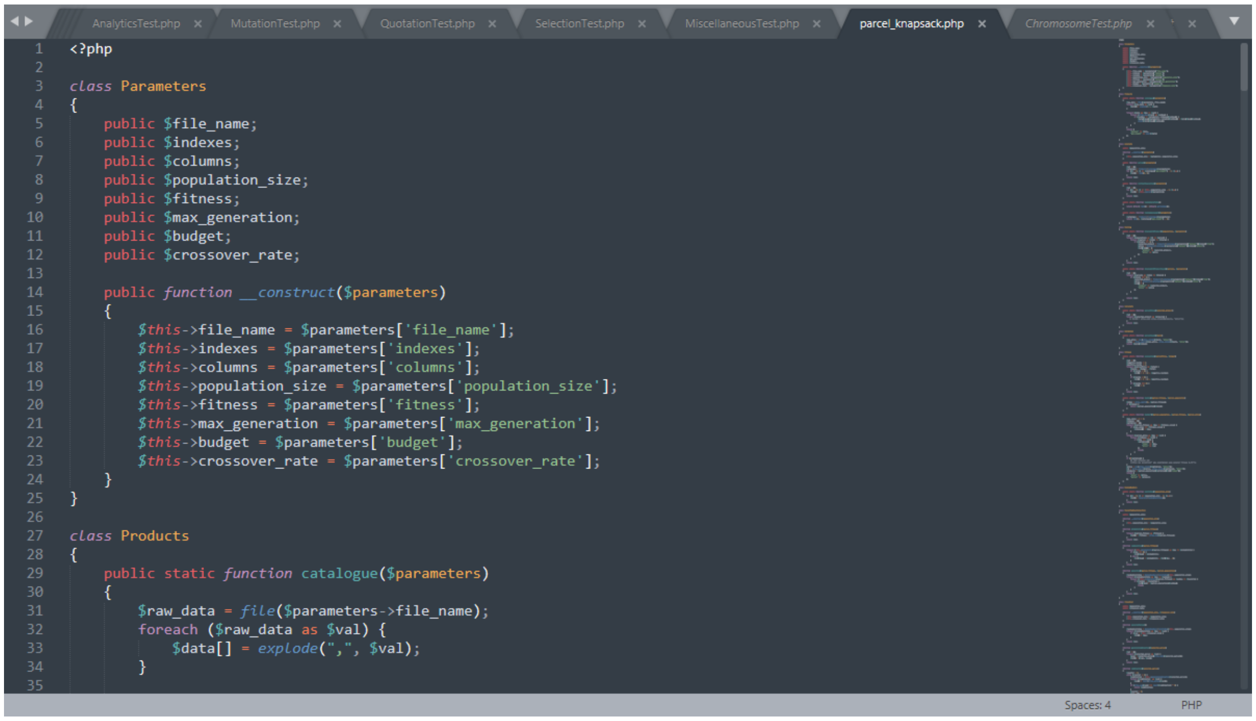
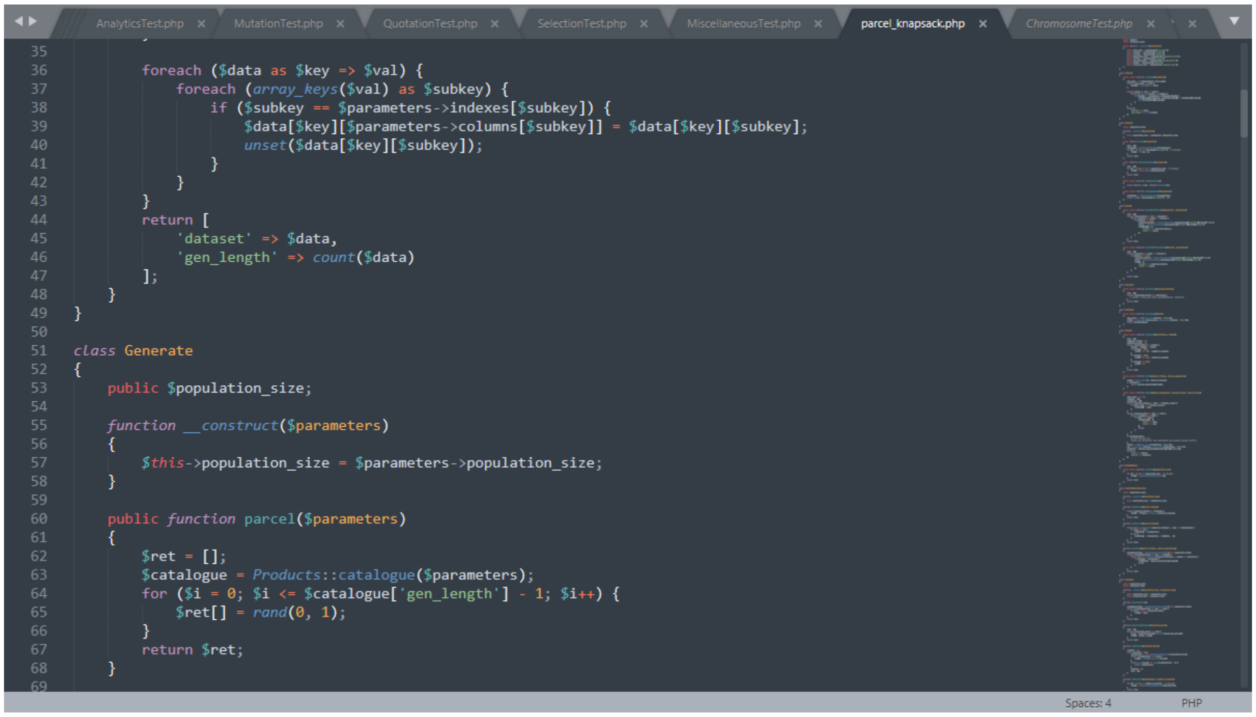
NAMA : IRMA ERYANTI PUTRI

NIM : 1800018272

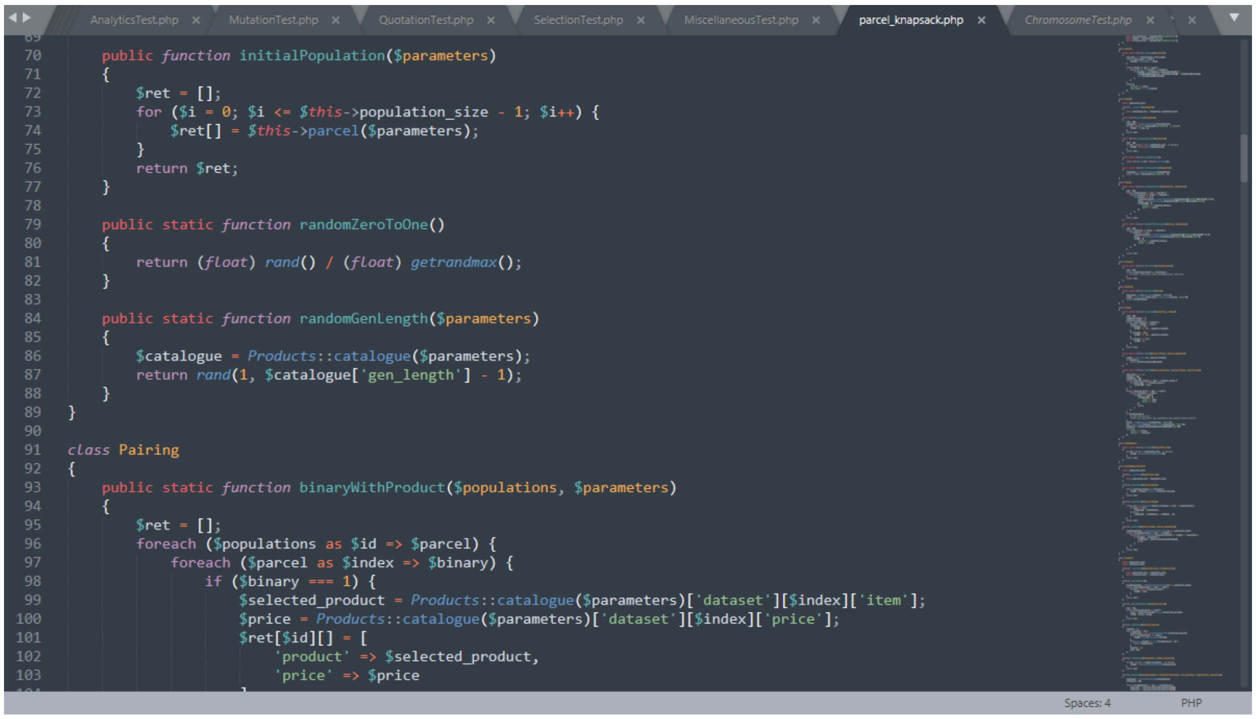
<u>Parcel_knapsack.php</u>



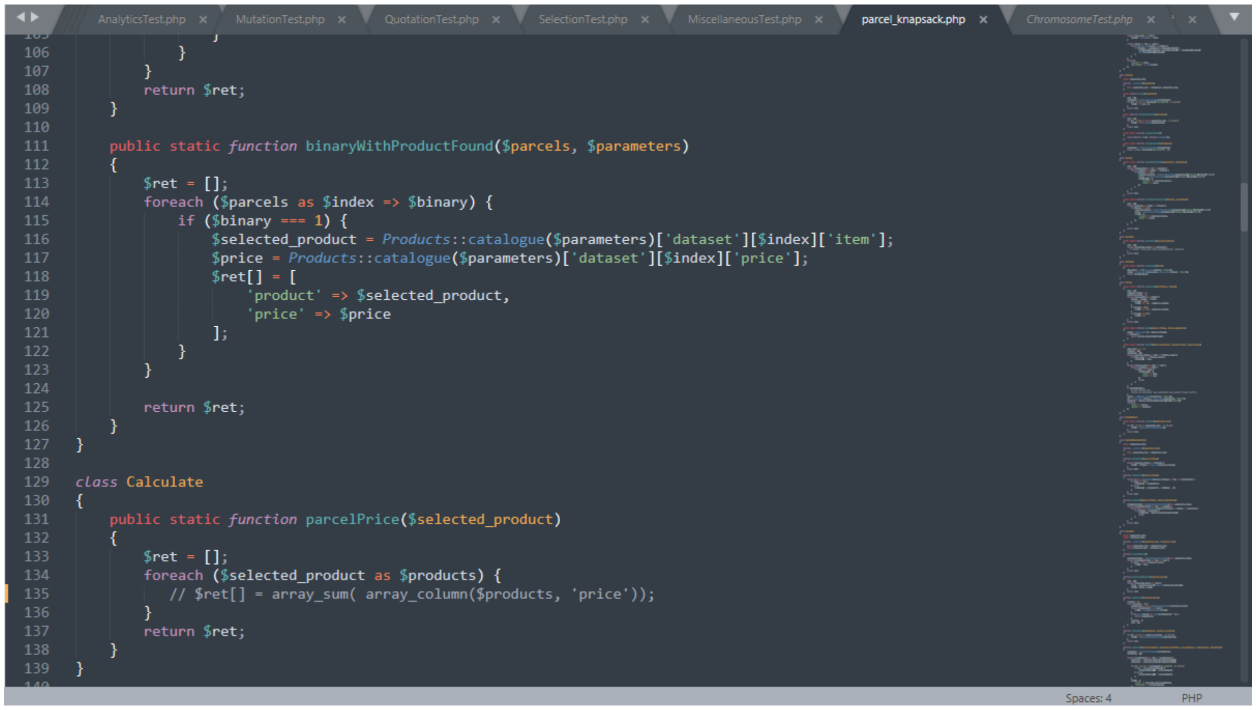
```php
<?php

class Parameters
{
    public $file_name;
    public $indexes;
    public $columns;
    public $population_size;
    public $fitness;
    public $max_generation;
    public $budget;
    public $crossover_rate;

    public function __construct($parameters)
    {
        $this->file_name = $parameters['file_name'];
        $this->indexes = $parameters['indexes'];
        $this->columns = $parameters['columns'];
        $this->population_size = $parameters['population_size'];
        $this->fitness = $parameters['fitness'];
        $this->max_generation = $parameters['max_generation'];
        $this->budget = $parameters['budget'];
        $this->crossover_rate = $parameters['crossover_rate'];
    }
}

class Products
{
    public static function catalogue($parameters)
    {
        $raw_data = file($parameters->file_name);
        foreach ($raw_data as $val) {
            $data[] = explode(",", $val);
        }

```



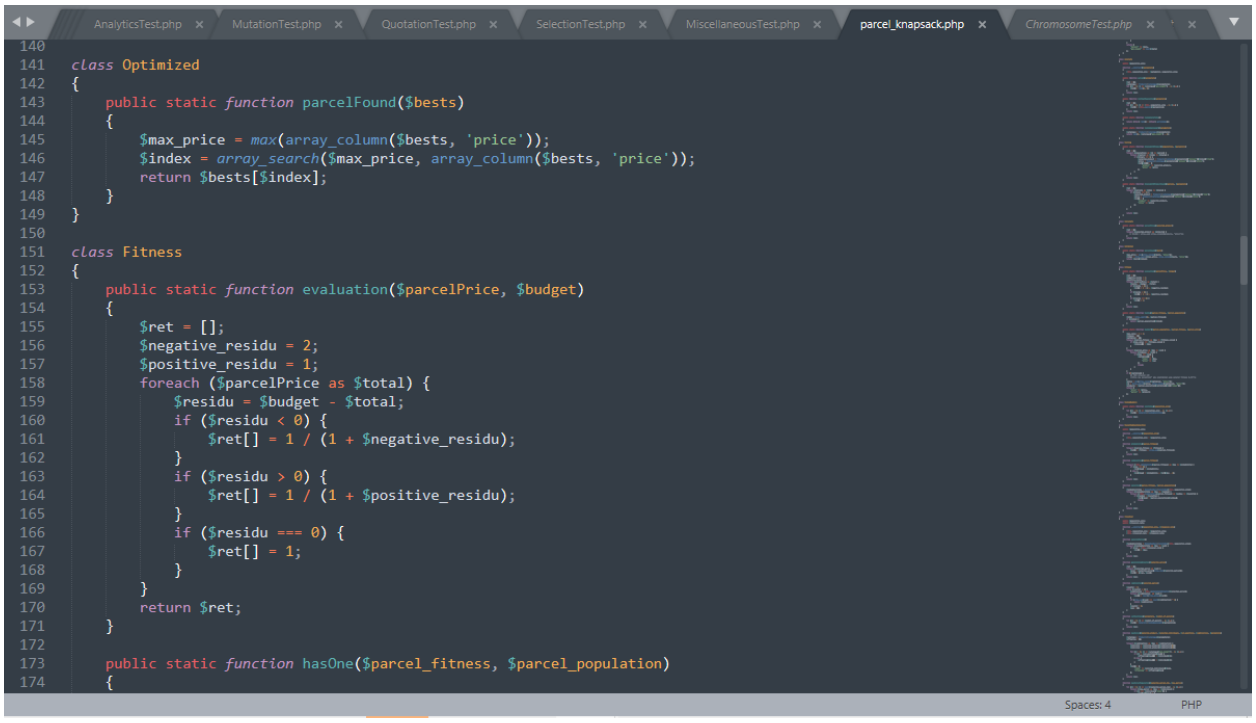
```php

        foreach ($data as $key => $val) {
            foreach (array_keys($val) as $subkey) {
                if ($subkey == $parameters->indexes[$subkey]) {
                    $data[$key][$parameters->columns[$subkey]] = $data[$key][$subkey];
                    unset($data[$key][$subkey]);
                }
            }
        }
        return [
            'dataset' => $data,
            'gen_length' => count($data)
        ];
    }
}

class Generate
{
    public $population_size;

    function __construct($parameters)
    {
        $this->population_size = $parameters->population_size;
    }

    public function parcel($parameters)
    {
        $ret = [];
        $catalogue = Products::catalogue($parameters);
        for ($i = 0; $i <= $catalogue['gen_length'] - 1; $i++) {
            $ret[] = rand(0, 1);
        }
        return $ret;
    }

```



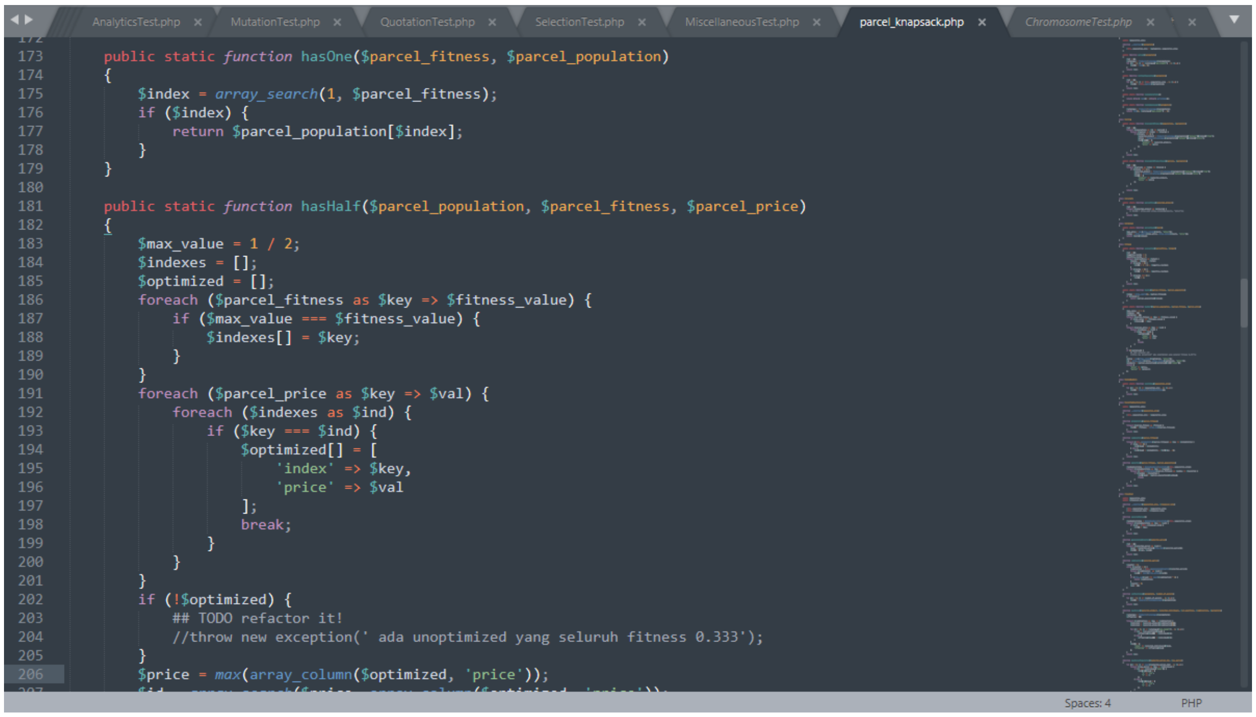
```php
    public function initialPopulation($parameters)
    {
        $ret = [];
        for ($i = 0; $i <= $this->population_size - 1; $i++) {
            $ret[] = $this->parcel($parameters);
        }
        return $ret;
    }

    public static function randomZeroToOne()
    {
        return (float) rand() / (float) getrandmax();
    }

    public static function randomGenLength($parameters)
    {
        $catalogue = Products::catalogue($parameters);
        return rand(1, $catalogue['gen_length'] - 1);
    }
}

class Pairing
{
    public static function binaryWithProduct($populations, $parameters)
    {
        $ret = [];
        foreach ($populations as $id => $parcel) {
            foreach ($parcel as $index => $binary) {
                if ($binary === 1) {
                    $selected_product = Products::catalogue($parameters)['dataset'][$index]['item'];
                    $price = Products::catalogue($parameters)['dataset'][$index]['price'];
                    $ret[$id][] = [
                        'product' => $selected_product,
                        'price' => $price
```



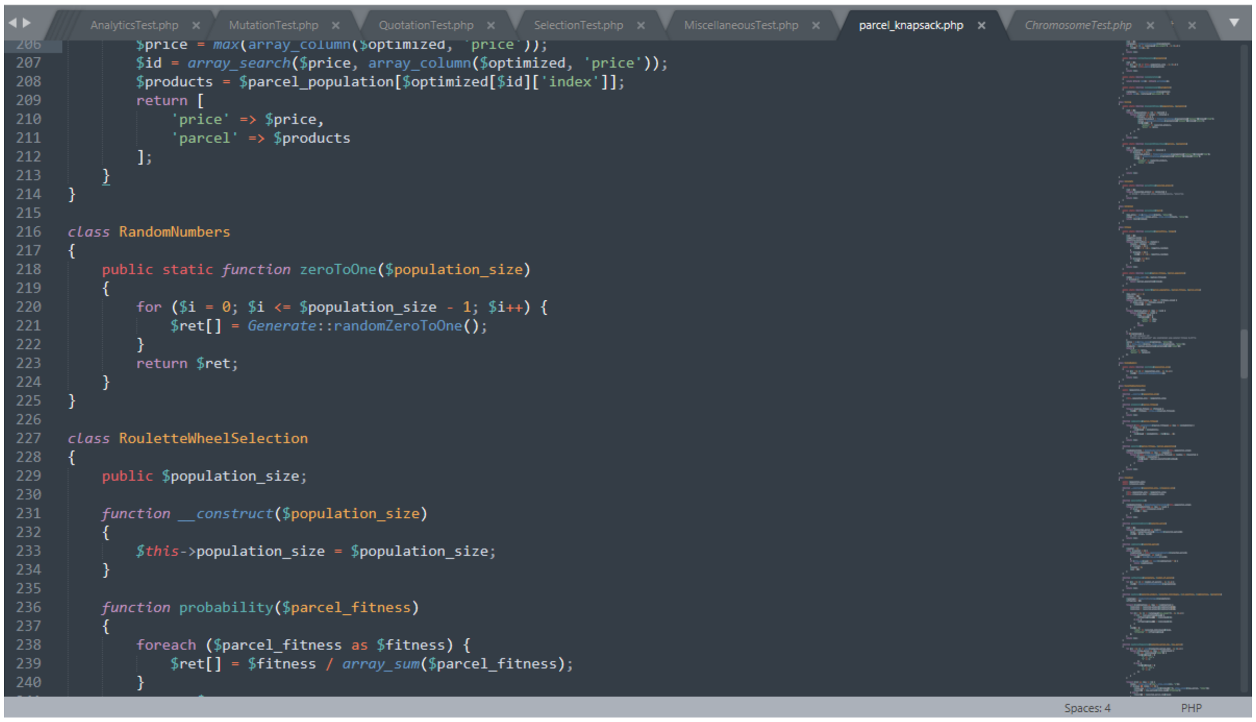
```php
            }
        }
        return $ret;
    }

    public static function binaryWithProductFound($parcels, $parameters)
    {
        $ret = [];
        foreach ($parcels as $index => $binary) {
            if ($binary === 1) {
                $selected_product = Products::catalogue($parameters)['dataset'][$index]['item'];
                $price = Products::catalogue($parameters)['dataset'][$index]['price'];
                $ret[] = [
                    'product' => $selected_product,
                    'price' => $price
                ];
            }
        }

        return $ret;
    }
}

class Calculate
{
    public static function parcelPrice($selected_product)
    {
        $ret = [];
        foreach ($selected_product as $products) {
            // $ret[] = array_sum( array_column($products, 'price'));
        }
        return $ret;
    }
}
```



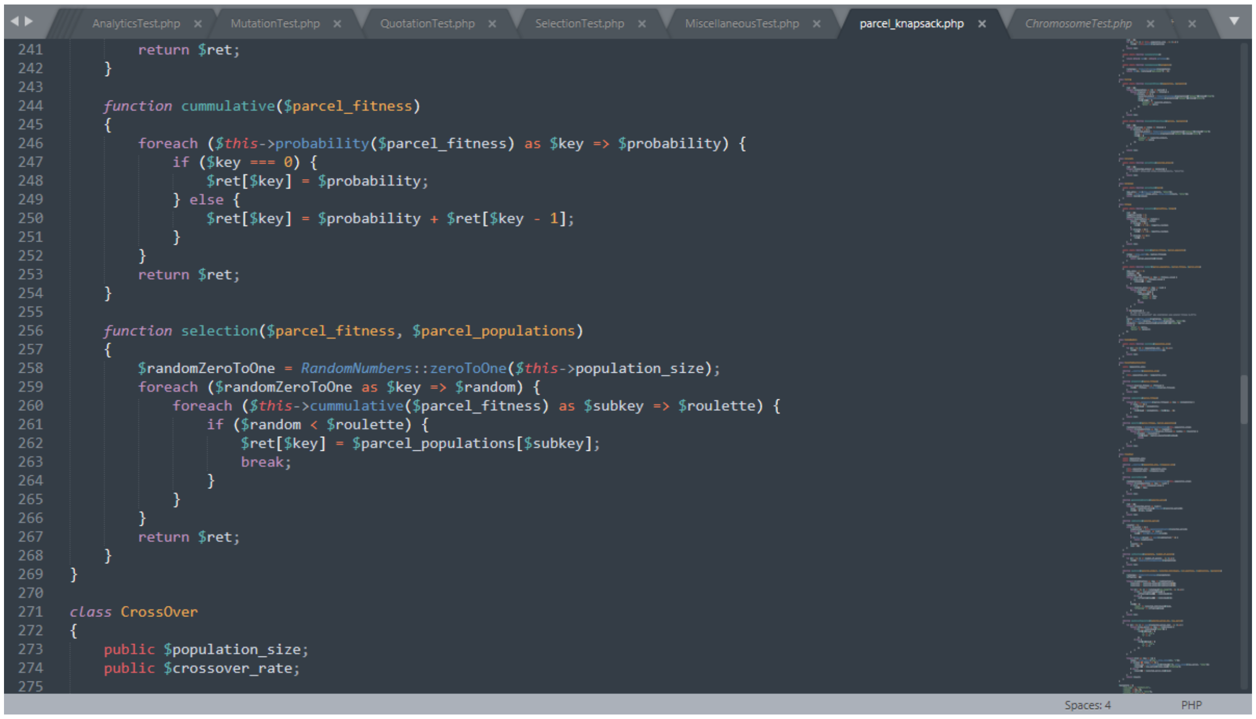
```php

class Optimized
{
    public static function parcelFound($bests)
    {
        $max_price = max(array_column($bests, 'price'));
        $index = array_search($max_price, array_column($bests, 'price'));
        return $bests[$index];
    }
}

class Fitness
{
    public static function evaluation($parcelPrice, $budget)
    {
        $ret = [];
        $negative_residu = 2;
        $positive_residu = 1;
        foreach ($parcelPrice as $total) {
            $residu = $budget - $total;
            if ($residu < 0) {
                $ret[] = 1 / (1 + $negative_residu);
            }
            if ($residu > 0) {
                $ret[] = 1 / (1 + $positive_residu);
            }
            if ($residu === 0) {
                $ret[] = 1;
            }
        }
        return $ret;
    }

    public static function hasOne($parcel_fitness, $parcel_population)
    {
```



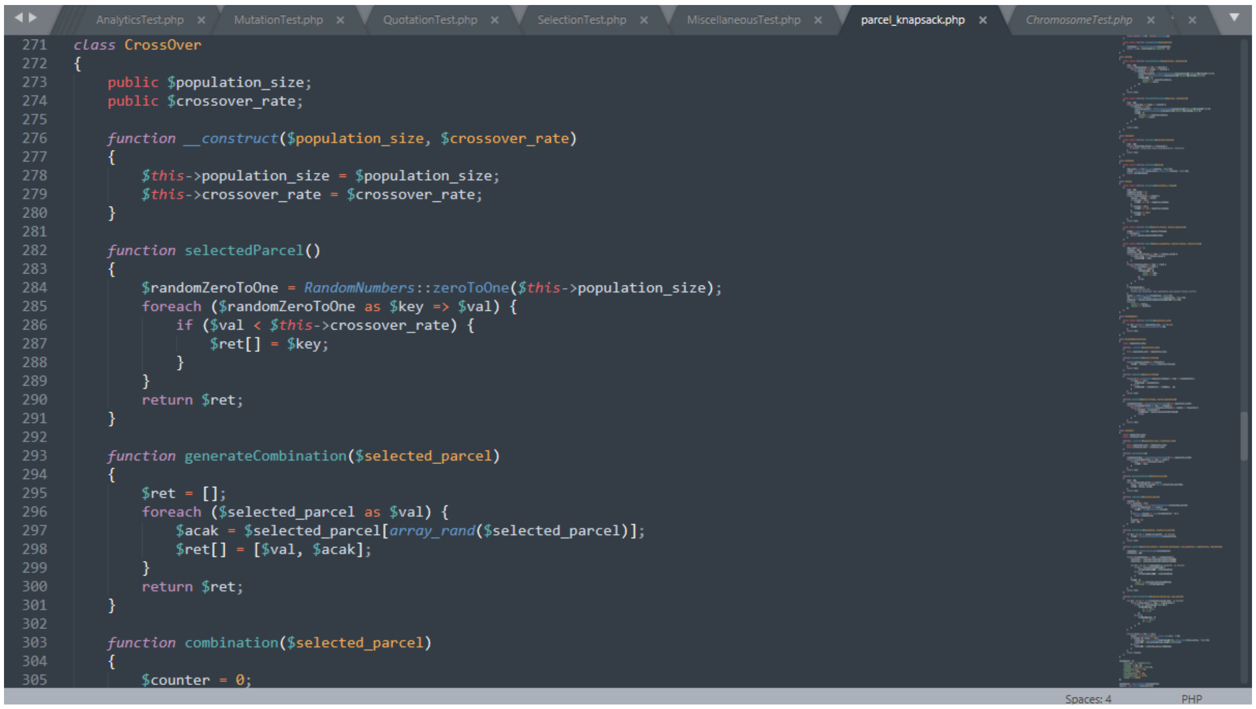
```php
    public static function hasOne($parcel_fitness, $parcel_population)
    {
        $index = array_search(1, $parcel_fitness);
        if ($index) {
            return $parcel_population[$index];
        }
    }

    public static function hasHalf($parcel_population, $parcel_fitness, $parcel_price)
    {
        $max_value = 1 / 2;
        $indexes = [];
        $optimized = [];
        foreach ($parcel_fitness as $key => $fitness_value) {
            if ($max_value === $fitness_value) {
                $indexes[] = $key;
            }
        }
        foreach ($parcel_price as $key => $val) {
            foreach ($indexes as $ind) {
                if ($key === $ind) {
                    $optimized[] = [
                        'index' => $key,
                        'price' => $val
                    ];
                    break;
                }
            }
        }
        if (!$optimized) {
            ## TODO refactor it!
            //throw new exception(' ada unoptimized yang seluruh fitness 0.333');
        }
        $price = max(array_column($optimized, 'price'));
```



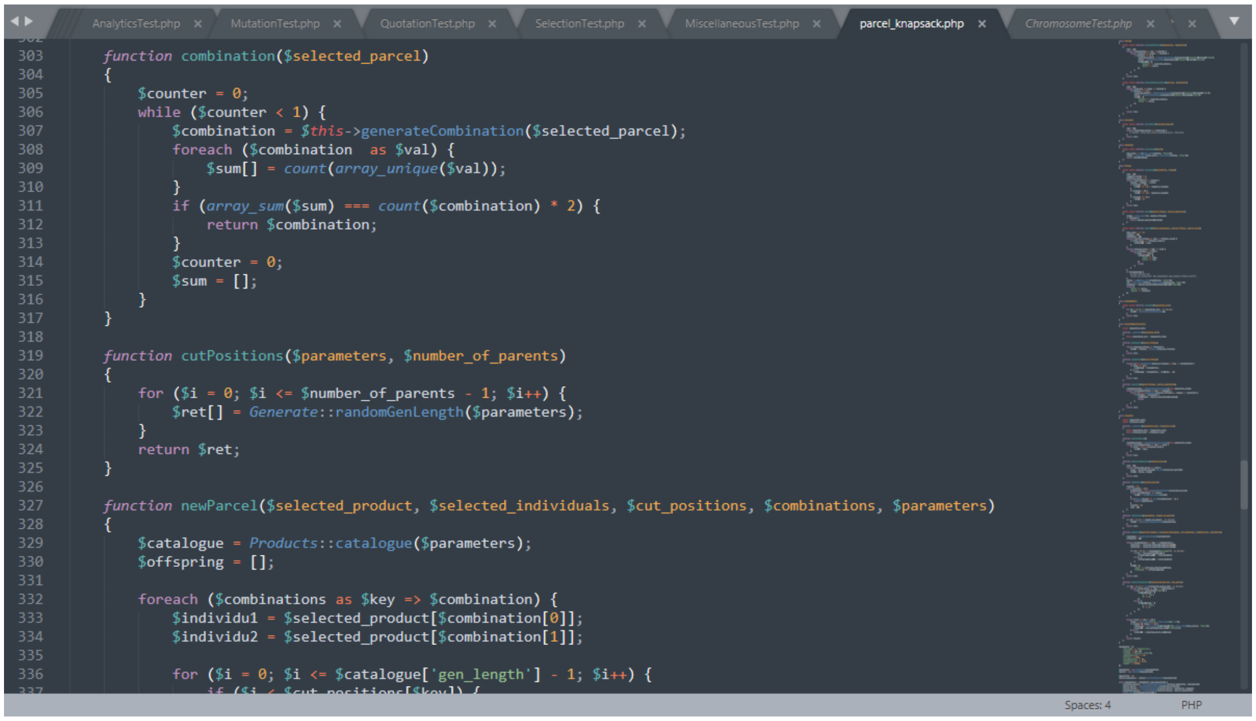
```php
        $price = max(array_column($optimized, 'price'));
        $id = array_search($price, array_column($optimized, 'price'));
        $products = $parcel_population[$optimized[$id]['index']];
        return [
            'price' => $price,
            'parcel' => $products
        ];
    }
}

class RandomNumbers
{
    public static function zeroToOne($population_size)
    {
        for ($i = 0; $i <= $population_size - 1; $i++) {
            $ret[] = Generate::randomZeroToOne();
        }
        return $ret;
    }
}

class RouletteWheelSelection
{
    public $population_size;

    function __construct($population_size)
    {
        $this->population_size = $population_size;
    }

    function probability($parcel_fitness)
    {
        foreach ($parcel_fitness as $fitness) {
            $ret[] = $fitness / array_sum($parcel_fitness);
        }
```



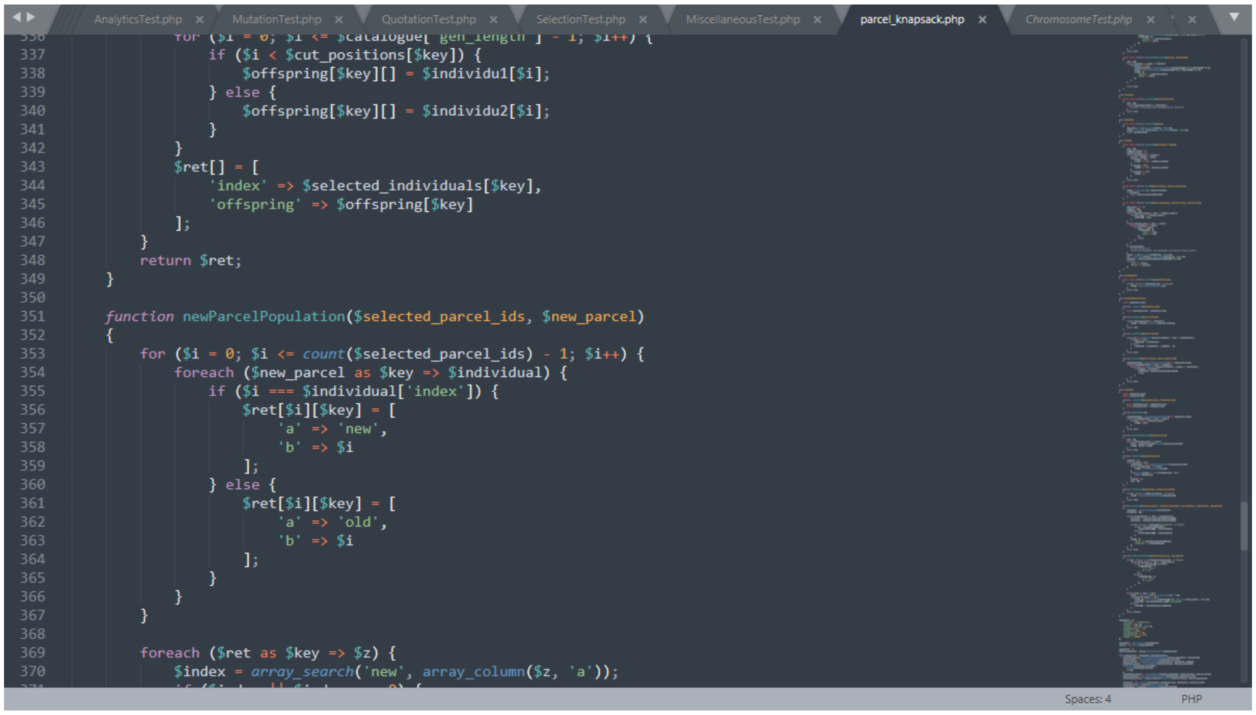
```php
        return $ret;
    }

    function cummulative($parcel_fitness)
    {
        foreach ($this->probability($parcel_fitness) as $key => $probability) {
            if ($key === 0) {
                $ret[$key] = $probability;
            } else {
                $ret[$key] = $probability + $ret[$key - 1];
            }
        }
        return $ret;
    }

    function selection($parcel_fitness, $parcel_populations)
    {
        $randomZeroToOne = RandomNumbers::zeroToOne($this->population_size);
        foreach ($randomZeroToOne as $key => $random) {
            foreach ($this->cummulative($parcel_fitness) as $subkey => $roulette) {
                if ($random < $roulette) {
                    $ret[$key] = $parcel_populations[$subkey];
                    break;
                }
            }
        }
        return $ret;
    }
}

class CrossOver
{
    public $population_size;
    public $crossover_rate;

```



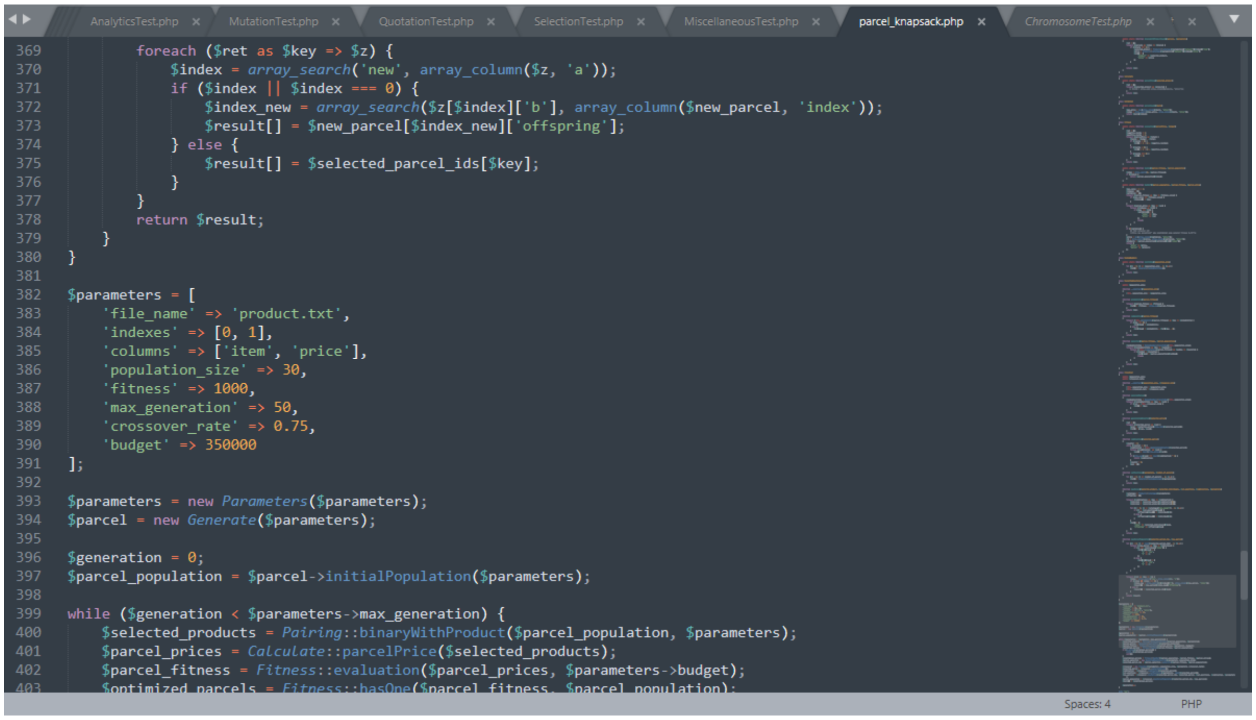
```php
class CrossOver
{
    public $population_size;
    public $crossover_rate;

    function __construct($population_size, $crossover_rate)
    {
        $this->population_size = $population_size;
        $this->crossover_rate = $crossover_rate;
    }

    function selectedParcel()
    {
        $randomZeroToOne = RandomNumbers::zeroToOne($this->population_size);
        foreach ($randomZeroToOne as $key => $val) {
            if ($val < $this->crossover_rate) {
                $ret[] = $key;
            }
        }
        return $ret;
    }

    function generateCombination($selected_parcel)
    {
        $ret = [];
        foreach ($selected_parcel as $val) {
            $acak = $selected_parcel[array_rand($selected_parcel)];
            $ret[] = [$val, $acak];
        }
        return $ret;
    }

    function combination($selected_parcel)
    {
        $counter = 0;
```



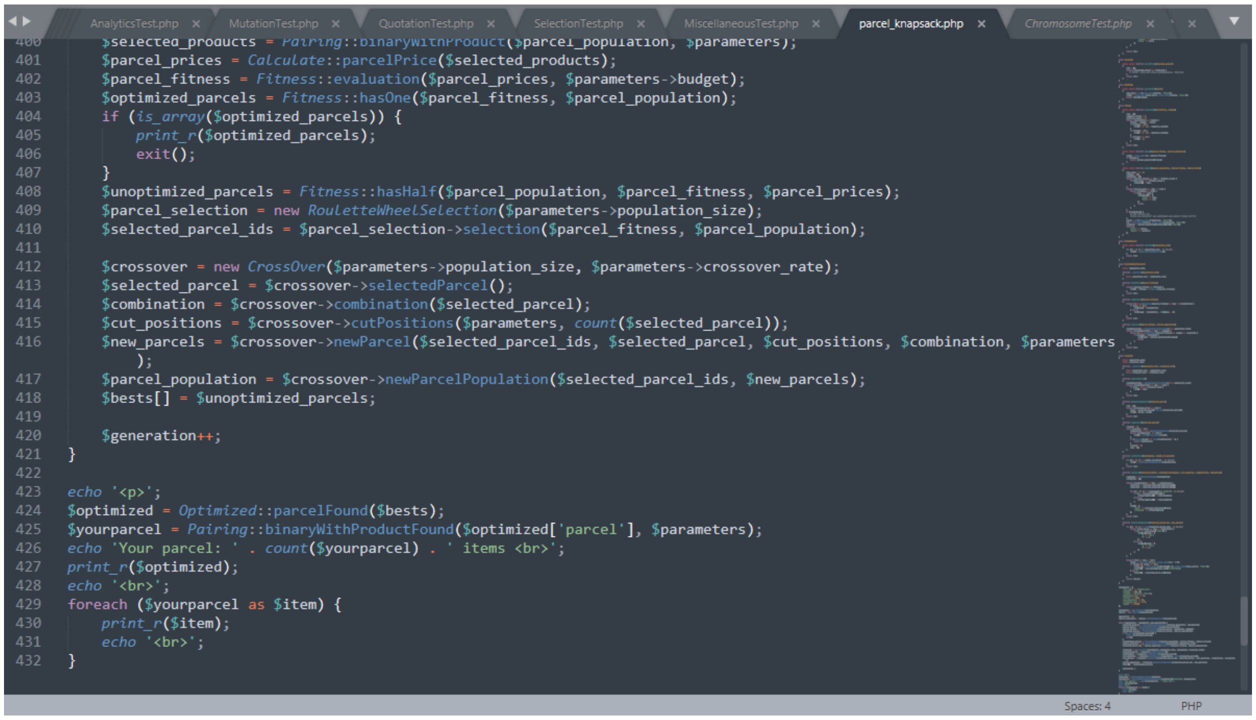
```php
    function combination($selected_parcel)
    {
        $counter = 0;
        while ($counter < 1) {
            $combination = $this->generateCombination($selected_parcel);
            foreach ($combination  as $val) {
                $sum[] = count(array_unique($val));
            }
            if (array_sum($sum) === count($combination) * 2) {
                return $combination;
            }
            $counter = 0;
            $sum = [];
        }
    }

    function cutPositions($parameters, $number_of_parents)
    {
        for ($i = 0; $i <= $number_of_parents - 1; $i++) {
            $ret[] = Generate::randomGenLength($parameters);
        }
        return $ret;
    }

    function newParcel($selected_product, $selected_individuals, $cut_positions, $combinations, $parameters)
    {
        $catalogue = Products::catalogue($parameters);
        $offspring = [];

        foreach ($combinations as $key => $combination) {
            $individu1 = $selected_product[$combination[0]];
            $individu2 = $selected_product[$combination[1]];

            for ($i = 0; $i <= $catalogue['gen_length'] - 1; $i++) {
                if ($i < $cut_positions[$key]) {
```



```php
            for ($i = 0; $i <= $catalogue['gen_length'] - 1; $i++) {
                if ($i < $cut_positions[$key]) {
                    $offspring[$key][] = $individu1[$i];
                } else {
                    $offspring[$key][] = $individu2[$i];
                }
            }
            $ret[] = [
                'index' => $selected_individuals[$key],
                'offspring' => $offspring[$key]
            ];
        }
        return $ret;
    }

    function newParcelPopulation($selected_parcel_ids, $new_parcel)
    {
        for ($i = 0; $i <= count($selected_parcel_ids) - 1; $i++) {
            foreach ($new_parcel as $key => $individual) {
                if ($i === $individual['index']) {
                    $ret[$i][$key] = [
                        'a' => 'new',
                        'b' => $i
                    ];
                } else {
                    $ret[$i][$key] = [
                        'a' => 'old',
                        'b' => $i
                    ];
                }
            }
        }

        foreach ($ret as $key => $z) {
            $index = array_search('new', array_column($z, 'a'));
            if ($index || $index === 0) {
```

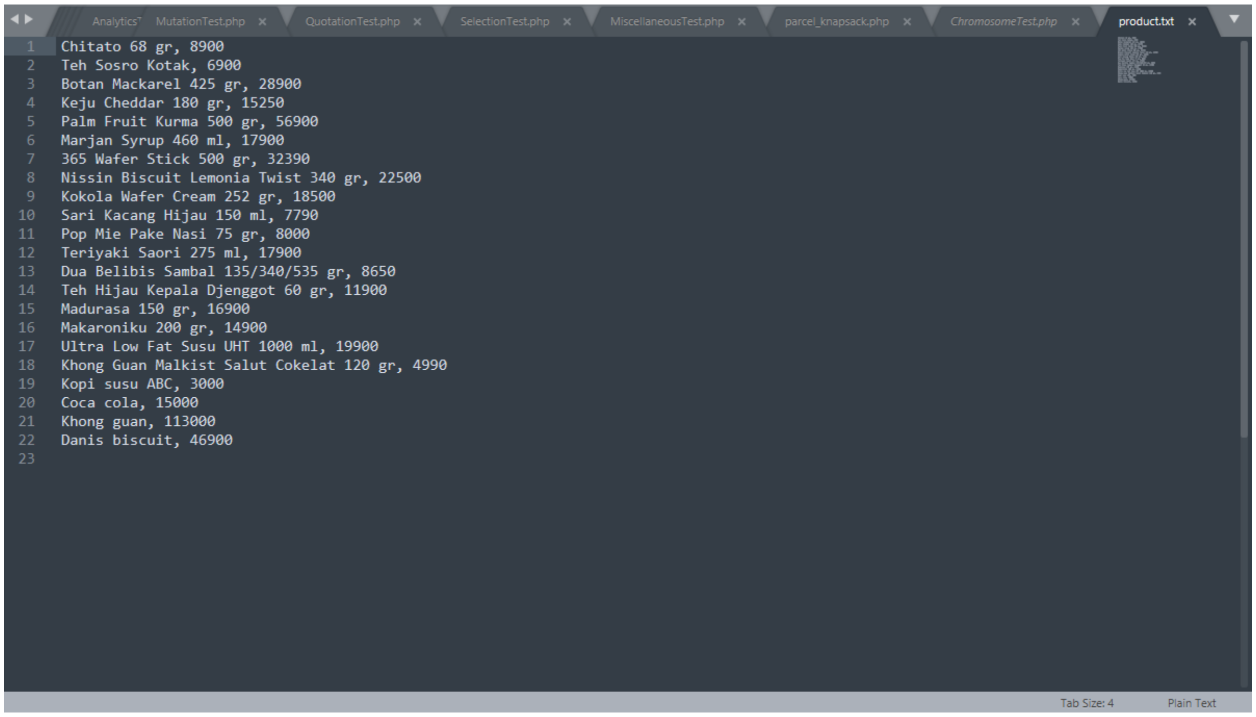


```php
        foreach ($ret as $key => $z) {
            $index = array_search('new', array_column($z, 'a'));
            if ($index || $index === 0) {
                $index_new = array_search($z[$index]['b'], array_column($new_parcel, 'index'));
                $result[] = $new_parcel[$index_new]['offspring'];
            } else {
                $result[] = $selected_parcel_ids[$key];
            }
        }
        return $result;
    }
}

$parameters = [
    'file_name' => 'product.txt',
    'indexes' => [0, 1],
    'columns' => ['item', 'price'],
    'population_size' => 30,
    'fitness' => 1000,
    'max_generation' => 50,
    'crossover_rate' => 0.75,
    'budget' => 350000
];

$parameters = new Parameters($parameters);
$parcel = new Generate($parameters);

$generation = 0;
$parcel_population = $parcel->initialPopulation($parameters);

while ($generation < $parameters->max_generation) {
    $selected_products = Pairing::binaryWithProduct($parcel_population, $parameters);
    $parcel_prices = Calculate::parcelPrice($selected_products);
    $parcel_fitness = Fitness::evaluation($parcel_prices, $parameters->budget);
    $optimized_parcels = Fitness::hasOne($parcel_fitness, $parcel_population);
```



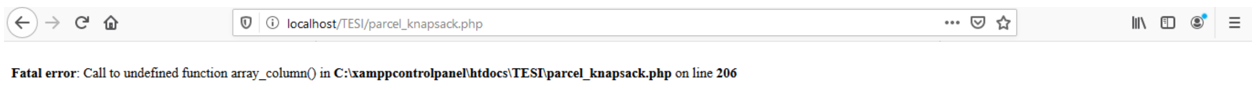
```php
        $selected_products = Pairing::binaryWithProduct($parcel_population, $parameters);
        $parcel_prices = Calculate::parcelPrice($selected_products);
        $parcel_fitness = Fitness::evaluation($parcel_prices, $parameters->budget);
        $optimized_parcels = Fitness::hasOne($parcel_fitness, $parcel_population);
        if (is_array($optimized_parcels)) {
            print_r($optimized_parcels);
            exit();
        }
        $unoptimized_parcels = Fitness::hasHalf($parcel_population, $parcel_fitness, $parcel_prices);
        $parcel_selection = new RouletteWheelSelection($parameters->population_size);
        $selected_parcel_ids = $parcel_selection->selection($parcel_fitness, $parcel_population);

        $crossover = new CrossOver($parameters->population_size, $parameters->crossover_rate);
        $selected_parcel = $crossover->selectedParcel();
        $combination = $crossover->combination($selected_parcel);
        $cut_positions = $crossover->cutPositions($parameters, count($selected_parcel));
        $new_parcels = $crossover->newParcel($selected_parcel_ids, $selected_parcel, $cut_positions, $combination, $parameters
            );
        $parcel_population = $crossover->newParcelPopulation($selected_parcel_ids, $new_parcels);
        $bests[] = $unoptimized_parcels;

        $generation++;
    }

    echo '<p>';
    $optimized = Optimized::parcelFound($bests);
    $yourparcel = Pairing::binaryWithProductFound($optimized['parcel'], $parameters);
    echo 'Your parcel: ' . count($yourparcel) . ' items <br>';
    print_r($optimized);
    echo '<br>';
    foreach ($yourparcel as $item) {
        print_r($item);
        echo '<br>';
    }
```

<u>Produtcs.txt</u>

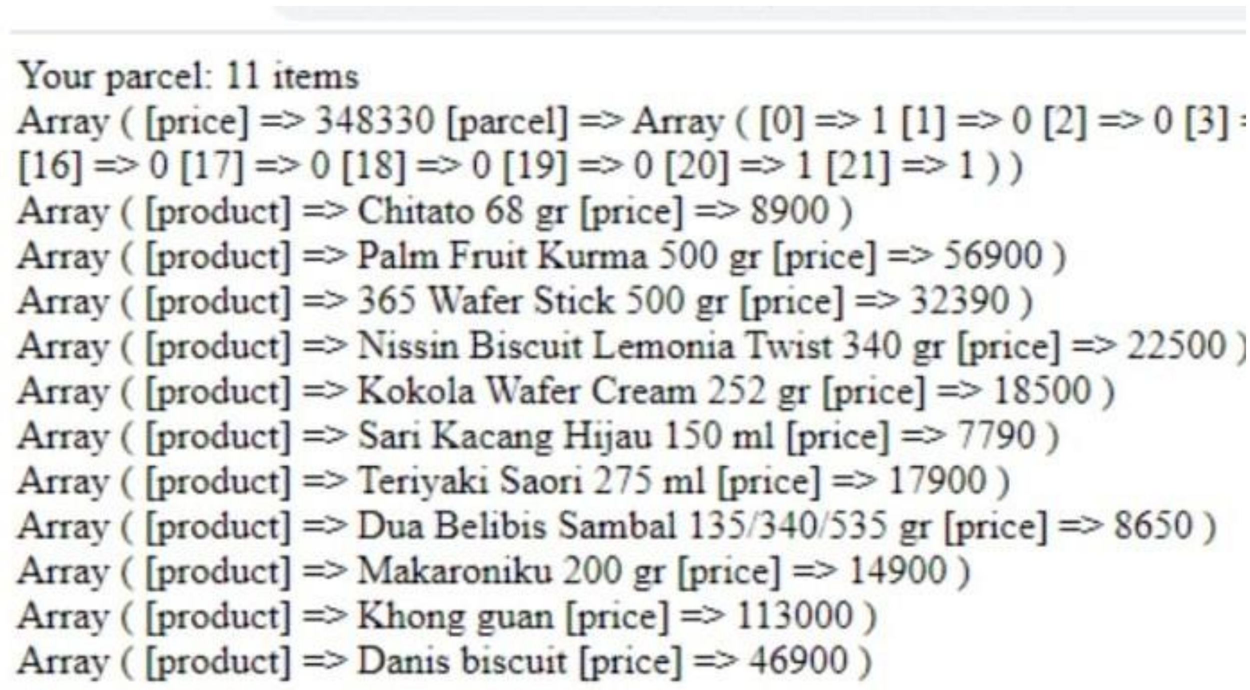


```
Chitato 68 gr, 8900
Teh Sosro Kotak, 6900
Botan Mackarel 425 gr, 28900
Keju Cheddar 180 gr, 15250
Palm Fruit Kurma 500 gr, 56900
Marjan Syrup 460 ml, 17900
365 Wafer Stick 500 gr, 32390
Nissin Biscuit Lemonia Twist 340 gr, 22500
Kokola Wafer Cream 252 gr, 18500
Sari Kacang Hijau 150 ml, 7790
Pop Mie Pake Nasi 75 gr, 8000
Teriyaki Saori 275 ml, 17900
Dua Belibis Sambal 135/340/535 gr, 8650
Teh Hijau Kepala Djenggot 60 gr, 11900
Madurasa 150 gr, 16900
Makaroniku 200 gr, 14900
Ultra Low Fat Susu UHT 1000 ml, 19900
Khong Guan Malkist Salut Cokelat 120 gr, 4990
Kopi susu ABC, 3000
Coca cola, 15000
Khong guan, 113000
Danis biscuit, 46900
```

Output:



```
Fatal error: Call to undefined function array_column() in C:\xamppcontrolpanel\htdocs\TESI\parcel_knapsack.php on line 206
```



```
Your parcel: 11 items
Array ( [price] => 348330 [parcel] => Array ( [0] => 1 [1] => 0 [2] => 0 [3]
[16] => 0 [17] => 0 [18] => 0 [19] => 0 [20] => 1 [21] => 1 ) )
Array ( [product] => Chitato 68 gr [price] => 8900 )
Array ( [product] => Palm Fruit Kurma 500 gr [price] => 56900 )
Array ( [product] => 365 Wafer Stick 500 gr [price] => 32390 )
Array ( [product] => Nissin Biscuit Lemonia Twist 340 gr [price] => 22500 )
Array ( [product] => Kokola Wafer Cream 252 gr [price] => 18500 )
Array ( [product] => Sari Kacang Hijau 150 ml [price] => 7790 )
Array ( [product] => Teriyaki Saori 275 ml [price] => 17900 )
Array ( [product] => Dua Belibis Sambal 135/340/535 gr [price] => 8650 )
Array ( [product] => Makaroniku 200 gr [price] => 14900 )
Array ( [product] => Khong guan [price] => 113000 )
Array ( [product] => Danis biscuit [price] => 46900 )
```