The double tournament consists of a selection technique where candidates must win two tournaments to be selected. These competitions will be based on their fitness and their parsimony (size), the order of the criteria being defined by the user.

The primary reason to apply this technique over the single tournament, that only considers the fitness, is to attempt to counter the bloating problem. As the number of iterations increases the new population increases its complexity, trees with more branches, this is easily understood by looking into the genetic operator's logic, but this increase in size it's not necessarily accompanied with a better fitness function, as so adding a tournament phase to select simpler functions will result in simpler genetic operations, this is simpler trees.

The function *double_tournament* (GeneticOperators, Line 27-94) takes six parameters: 'rng', an instance of the random number generator used for randomisation purposes; 'population', a list of individuals sorted from best fitness to worst; 'sf' and 'sp', corresponding to the fitness tournament size and the parsimony tournament size, respectively; 'n', the number of individuals selected for the respective first tournament; and 'switch', a boolean determining the order of the tournaments, with False being the default.

If 'switch'==False and 'sp' is lower or equal to 'sf', the function performs the fitness tournament first. In this case, a loop iterates over the range of 'sf' and selects the individual with minimum index, corresponding to a higher fitness, from 'n' random individuals. The index of the selected individual is then appended to the 'winners_one' list, that stores the individuals for the following tournament. In the beginning of the size tournament, 'sp' random individuals are chosen from the list 'winners_one'. Similarly, to the previous tournament, a loop iterates through the candidates. However, this time the size of each individual is received using the function *getSize()* (Individual, Line 92-98) and saved separately. Next, the individual with the lowest size is determined. If there are multiple individuals with the same smallest size, the function selects one of them randomly. Finally, the winning individual from the population is returned.

In the case when the parsimony tournament is performed first (elif 'switch' == True and 'sp' higher or equal to 'sf'), the function selects 'sp' winners from the first tournament by performing 'sp' tournaments of 'n' individuals each and selecting the individual with the smallest size from each tournament, utilizing the function *getSize()*. The indexes of these individuals are then stored in the "winners_1" list. For the fitness tournament 'sf' random individuals from the winners are chosen. Finally, the fittest individual (smallest index) from the 'sf 'chosen candidates is selected as the winner and returned.

Lastly, if 'switch'==True and 'sp' is lower than 'sf' or if 'switch' == False and 'sp' higher than 'sf', the function will kill the process utilising the *'exit'* function from the sys library and will print an error message, which tells the user what to change.

After the completion of the function, the double tournament needed to be implemented into the *StdGP* Framework. First, it got implemented into the crossover function *STXO* (GeneticOperators, 136-160) with the purpose of selecting two individuals for crossover. Moreover, it was implemented into the mutation function *STMUT* (GeneticOperators, Line 163-183) for the selection of one individual which gets mutated. Lastly, the *getOffspring* (GeneticOperators, 106-125) function was adapted accordingly since it uses the mutation, or the crossover function described.

Finally, the double tournament was implemented into the StdGP class provided. First the parameters of the function were passed to the init arguments (StdGP, Line 42-46) and then implemented into the init-function (Line 85) with some exemplary values which can be overwritten if needed. Next, the defined arguments get assigned to the respective attributes of the class (Line 102-106) which then get included in the verbose function (StdGP, Line 209-212), to inform the user of the parameters used. The last implementation step was to adapt the parameters of the *getOffspring* function also in the StdGP class (StdGP, Line 338).