Main predicates:-

1) straight_chain_alkene(N,A) : All this method does it that it calls the helper method sca(N,1,A).

2)branched_alkane(N,BA) :    gets a straight chain alkene with length N and adds it in the predicate branchedhelper(BC,BA,SN).

3)isomers(N,X): makes a straight chain alkene and puts them in a list then uses that list to call isomers_helper predicate

Helper predicates:-

1)sca(N,C,A) : this is a helper that returns a straight chain alkene, N is it's length, C is a counter that determines the position of the current carbon, and A is the output, this predicate has 4 cases, output for 1st carbon, output for any middle carbon, output for last carbon and a case where it expects an input of 1 to return carb(h,h,h,h).

2)branch_name(Z,X): given in the project notes, takes a number and returns its corresponding carbon

3)seqA(F,L,O) : returns a sequence from F to L in O in ascending order

4)check_validity(List, I):  checks the second condition "The number of carbon atoms in any branch should not lead to any new chains that are longer than the current longest chain. However, it is OK if it leads to a chain of a size less than or equal to the current longest chain", it accesses every element in the list with nth0/3 predicate and checks the maximum carbon using max_branch and compares it the the length before it and the length after it.

5)split2(L,I,I2,N,L2):  this predicate takes the list L and starting index I and ending index I2, and returns a sublist from list L from index I to I2.

6)total_branch_number2([H I T],Z) : returns the total number of carbons in the middle branches in the whole list using total_branch_number predicate

7)total_branch_number(carb(A,B,C,D),T) : returns in T the total number of carbons in the middle (at B and C)

8)branchedhelper(BC,BA,SN):  That's our main method, it does the following :
combines every possible combination of middle chains, ie : tries to combine the second and the third element and then eliminates them and put the new merged one instead, then tries to eliminate the third and the fourth, etc...
the number of branches that can be eliminated every time is bounded by the maximum number of carbons that can be formed , ie : if we are forming c3h7 we will remove 3 branches.
At the end we get a new combined list, we filter the outputs using  total_branch_number2 to make sure the number of carbons is the exact number needed and  check_validity to check the second condition, if this combination is right we call the function again putting as input this new combination to get the rest of the solutions.

9)merge2(carb(A,B,C,D),carb(E,F,G,H),R,CL,T): Takes two carbon branches and the number of carbons you want to add (CL) and returns the merged branch at R, and also returns T which is the number of the highest carbon in the merged branch

10)max_branch([carb(A,B,C,D)],T) : takes a carb and returns the length of the number of highest branch out of this carb, this predicate uses branch_number.

11)branch_number(X,Z): takes a branch and counts the number of carbons in it.

12)mirror(L1, L2): takes a list and returns it’s mirror.

13)isomers_helper(N,X,S,Set): uses the counter S to access all the list and look for mirrored versions of it,
if found, it removes it from the list and continues to recurse through the whole list.

NOTE: when trying to get branched alkane of 15 to get it’s answers, we received a full global stack error, so we added this line of code (set_prolog_stack(global, limit(3*10**9)).) and after it we were able to check branched alkanes and isomers of 15 and had the same number of result as the correct answer