Exercise 1

Define black box tests for the following function, using equivalence classes and boundary conditions.

```
int triangleType(int side1, int side2, int side3 )
```

This function receives three integers that represent the length of three sides of a triangle.

It returns 1 if the three sides define an equilateral triangle, 2 if isosceles, 3 if scalene, -1 if the three sides cannot be composed in a triangle.

Ex:

```
triangleType( 1, 1, 1) -> 1
triangleType( 1, 1, 3) -> 2
triangleType( 3, 4, 5) -> 3
triangleType( 1, 1, 2) -> -1
```

Exercise 2

Define black box tests for the following function, using equivalence classes and boundary conditions. You can prune the solution space if the combinations of the possible equivalence classes are too many.

The Piedmont region issues benefits for a free sanitary ticket to certain classes of people. To benefit of the privilege, a person must satisfy a set of conditions. A person is eligible if he/she is younger than 6 years old, or older than 65 years old, and his family income is less than €36151.98. An unemployed person of any age is eligible for the benefit if he/she has a family income lower than €8263.57.

The function isEligibleForTicket receives as parameter age (the age of the person), family_income (the total income of the person's family), isEmployed (an integer parameter which is 0 if the person is unemployed, and 1 if the person is employed). It gives as output 1 if the person is eligible for the free sanitary ticket, 0 otherwise.

```
int isEligibleForTicket(int age, double family_income, int
isEmployed);
```

Examples:

isEligibleForTicket(5, 21000, 0) -> 1

isEligibleForTicket(75, 40000, 0) -> 0

isEligibleForTicket(35, 7000, 1) -> 0

isEligibleForTicket(35, 7000, 0) -> 1

Exercise 3

Define black box tests for the following function, using equivalence classes and boundary conditions.

In a bike race, the bikers must complete the entire track within a maximum time, otherwise their race is not valid. The maximum time is computed, for each race, based on the winner's time, on the average speed on the track, and on the category of the track.

For tracks of category 'A' (easy tracks) the maximum time is computed as the winner's time increased by 5% if the average speed is lower than 30 km/h (30 included), 10% if the average speed is between 30 and 35 km/h (35 included), and 15% if the average speed is higher than 35 km/h.

For tracks of category 'B' (normal tracks) the maximum time is computed as the winner's time increased by 20% if the average speed is lower than 30 km/h (30 included), 25% if the average speed is between 30 and 35 km/h (35 included), and 30% if the average speed is higher than 35 km/h.

For tracks of category 'C' (hard tracks) the maximum time does not depend on average speed, and is always computed as the winner's time increased by 50%.

The function computeMaxTime receives as parameters winner_time (the time of the winner, in minutes), avg_speed (the average speed of the track, in km/h) and track_type (a char, whose valid values are 'A', 'B', or 'C'). It gives as output the maximum time, 0 if there are errors in the input.

```
double computeMaxTime(double winner_time, double avg_speed, char
track_type)
```

Examples:

```
computeMaxTime(50, 27, 'A') -> 50 + 50*0.05 = 52.5

computeMaxTime(60, 33, 'B') -> 60 + 60*0.25 = 75

computeMaxTime(80, 40, 'C') -> 80 + 80*0.5 = 120
```

Exercise 4

For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage, path coverage. For the test cases, write only the input value.

```
1       int compute_tax(int wage) {
2
3       int ranges[] = {5000,15000,30000};
4       int amount_due = 0;
5       int level=0;
6
7       for(int i=0;iranges[i])  {
8               if (wage>ranges[i])
9                       level++;
10      }
11      if(level==1)
12              amount_due = 500;
13      else if(level == 2)
14              amount_due = 1500;
15      else if(level == 3)
16              amount_due = 3000;
17      return amount_due;
18      }
```

Exercise 5

For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage, path coverage. For the test cases, write only the input value.

```
1     int grade(int answers[5][4], int solutions[5][4]) {
2
3     int grade = 0;
4     int i, j;
5
6     for (i = 0; i < 5; i++) {
7           for (j = 0; j < 4; j++) {
8           if (answers[i][j] == solutions[i][j] && solutions[i][j] == 1)
9                       grade+=3;
10                else if (answers[i][j] != solutions[i][j] &&
solutions[i][j] == 0)
11                      grade-=1;
12                }
13          }
14    if (grade < 0)
15          return 0;
16    else
17          return grade;
18    }
```

Exercise 6

For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage, path coverage. For the test cases, write only the input value.

```
1    boolean racer_disqualified(int times[3], int winner_times[3], int
n_penalties, int* penalties) {
2
3          bool disqualified = false;
4          int i;
5          int max_time;
6          int tot_penalties;
7
8          for (i = 0; i < n_penalties; i++) {
9               tot_penalties += penalties[i];
10              if (penalties[i] > 100)
11                   disqualified = true;
12         }
13         if (tot_penalties > 100 || n_penalties > 5)
14
15              disqualified = true;
16
17         for (i=0; i max_time[i]) {
18
19              max_time = winner_times[i] * 1.5;
20
21              if (time[i] > max_time[i])
22                   disqualified = true;
23         }
24
25   }
```