Exercise 3

一、编码以及初始种群的产生

编码采用 16 位二进制进行编码，初始种群的染色体信息（基因）采用二位数组的形式进行储存，每一行表示一个个体的染色体，每一个染色体又由 16 个基因位组成。

根据题目中要求的定义域为[-1,15],且用 16 位二进制进行编码。设定二进制下的 0000000000000000 表示十进制下的-1，1111111111111111 表示十进制下的 15。根据这个设定，则在[-1,15]区间内的所有数字均可表示为 value = -1 + 16 * value / (2^16-1) （value 为已经由 16 位二进制数转换出来的十进制数）。

代码中函数 void decode(chromosome &population_current)就是实现这个功能的。

设置初始种群共有 100 个染色体，16 位二进制数的 0-1 值都是随机产生的。

函数 void population_initialize(chromosome (&population_current)[Population_size]) 就是实现这个功能的。

在主函数中还要初始化定义一些种群和个体，包括：当前种群 population_current，产生的下一代的种群 population_next_generation，一个所有参数均为 0 的个体，用于种群中某个个体的重置 zeros_chromosome，一个用来记录适应度最小值的个体 best_individual。

二、适应度计算

根据题目中的要求，适应度函数为 x*sin(x)，故只需将二进制数转换为十进制数后，代入函数求值即可。

函数 double objective_function(double x)就是实现这个功能的。

三、选择运算

初始种群产生后，使用轮盘赌选择法从种群中选出若干个体进行交叉、变异，也就是个体被选中的概率与其适应度函数值成正比，在本题中函数值越小，被选中的概率越大。

首先需要一个函数来更新种群内个体的属性值。当种群中个体的二进制串确定后，就可以计算每个个体 fitness、value、rate_fit 、cumu_fit。具体来说，就是计算出每个染色体的适应度、适应度占总体的百分比、积累概率，对应结构体 Chromosome 中定义的参数为 fitness、rate_fit、cumu_fit。由于本题中要求函数最小值，显然最小值为一个负数，为了让 fitness 值为负的染色体遗传下来的概率增大，我设定在计算 rate_fit 的时候，所有 fitness 值为正的染色体，他们的 rate_fit 均为 0；并且计算 fitness 的和的时候，只计算 fitness 为负值时的和，即 sum 的值为所有负的 fitness 的和。

在计算 cumu_fit 的值的时候，综合考虑这一项的 rate_fit 值和前一项的 cumu_fit 的值。其具体计算方法为： population_current[j].cumu_fit = population_current[j].rate_fit + population_current[j - 1].cumu_fit

函数 void fresh_property(chromosome(&population_current)[Population_size])就是实现这个功能的。

其次需要一个函数来基于轮盘赌选择方法，对种群中的染色体进行选择。首先生成一个随机数种子，产生一个随机概率，和每一个染色体的积累概率 cumu_fit 进行比较，决定子代中是否保留下来这个染色体。在这个过程中是可以尽可能保证 fitness 为负的染色体较大概率被挑选，特别是 fitness 值越小的染色体被挑选的概率越大。进而生成新的子代。

并且在新的子代生成后，还要找出这一代中的最优解，也即函数值最小时的 value 和最小的 fitness，并且记录下来。

函数 void seletc_prw(chromosome(&population_current)[Population_size], chromosome(&population_next_generation)[Population_size], chromosome &best_individual) 就是实现这个功能的。

四、交叉运算

接下来就要对新种群中选出的两个个体进行交叉操作，我选择的交叉方法是最简单的单点交叉。即交叉点随机产生。因为交叉操作要在一定的概率下进行，所以我设置了交叉概率 rate_crossover = 0.7。

具体过程为：先对群体进行随机配对，其次随机设置交叉点位置，最后在相互交换染色体之间的部分基因。

通过交叉操作，衍生出子代，以补充被淘汰掉的个体，新个体的适应度 fitness 较原来会有所提高。

函数 void crossover(chromosome(&population_next_generation)[Population_size])就是用来实现这个功能的。

五、变异运算

变异就是对染色体的基因进行变异，使其改变原来的结构（适应度也就改变），从而达到突变进化的目的。变异操作也要遵从一定的概率来进行，我设置变异概率为 0.3，即 rate_mutation = 0.3，以小概率进行基因突变。

我采用的变异方法是直接采取基因位反转变异法，即 0 变为 1，1 变为 0。要进行变异的基因位的选取也是由随机种子随机生成的。进而通过变异操作，衍生出子代。

函数 void mutation(chromosome(&population_next_generation)[Population_size])就是用来实现这个功能的。

六、终止规则

在这个程序中，程序停止运行的条件有两个：一个是迭代次数达到 5000 次，自动停止迭代，输出当前的最优解。为此，我设置了一个参数 iteration_num = 5000，用它来控制迭代次数的上限。另外一个条件是迭代出来的平均最优解约等于实际最优解，即 $x^* = \arg\min_x x*\sin x \approx 11.0857$，$x^*$只要落在(11.08,11.10)中间即可终止程序。

七、一些参数的设置

设置种群数量 Population_size = 100，之所以设置它为 100，是因为如果种群数量过小，会带来迭代次数过多，容易形成局部最优解的问题。如果种群数量很大，会出现迭代次数过少，甚至不需要迭代就可以得出结果的情况。所以经过了试验之后，我选择了 100 作为种群数量。

设置交叉率 rate_crossover = 0.7 和变异率 rate_mutation = 0.3，也是经过多次试验之后得到的相对较好的参数，可以大概率避免陷入局部最优解的情况，也可以在恰当的迭代轮数得到结果。

总而言之这三个参数的设置会严重影响解的品质,而目前就我的能力而言，这些参数的选择大部分是依靠试验。