Alternatives & Recommendations:

Originally, the code was set up with an abstract base class Individual and two inherited classes IndivCubic and IndivLang. The original abstract base class would have been essentially useless, because the method procreate could not be overridden. The procreate method had a reference to Individual in its parameter list and the inherited base classes IndivLang and IndivCubic had IndivLang and IndivCubic references in their parameter lists, respectively. These signatures were not recognized as being the same as an Individual& and therefore, the method could not be properly overridden. Since procreate could not be overridden, the inherited classes remained abstract and therefore objects of those classes could not be generated.

The goal was to implement one Evolution Class that could optimize both functions. For this, polymorphism was required.

The second version of the program involves an abstract base class, Model, containing only two methods: A method which returns the number of features of the function and a method for computing fitness, both of which are pure virtual. There exists a separate class called Individual which contains the methods for procreation as well as a base class Model pointer. Through polymorphism, the base class pointer's actions depend on the object that it is pointing to, not the type of the pointer. This makes it so that the pointer will call the method of the object it is pointing to. If the Model pointer is pointing to CubicFit and the fitness method is called, the fitness method of CubicFit, will be called. If the Model pointer is pointing to LangFit and the fitness method is called, the fitness method of LangFit, will be called. With the fitness function as a pure virtual function of Model, the only difference is the type of model.

Using polymorphism, the whole evolution class becomes much easier to write generically. Without polymorphism, there would have had to exist two separate evolution classes in order to apply the genetic algorithm to both functions.

Additionally, the two inherited classes are practically identical. Using std::list, it was possible to have either a list of CubicFit or LangFit. Each has a vector of x values, CubicFit has 4 elements in its feature vector and LangFit has 2 elements in its feature vector.

The problem solving approach described in the guidelines suggested creating a separate parent pool from the elements chosen via tournament selection and eventually merging the parent pool and the offspring pool. Instead of pushing parents into a separate parent container and then merging the offspring pool and parent pool together, the offspring pool is merged with the population pool. There is more diversity if each time more offspring were generated, a new set of parents were generated, than if you were to preselect a pool of 100 parents and discard the rest of the population. All that is known of the element with the best fitness from a randomly selected group of three individuals is that the element has the best fitness of those three, not that he's better than everyone else in the population. Therefore, it was a better decision not to have a separate parent container.

```
EvolutionLang()
    fill the population pool to popSize with individuals of type IndivLang
    sort the population pool from lowest to best fitness (lowest fitness being the best)
    if verbose
        display intermediate results:
         -number of iterations
         -the fitness of the best individual
         -the features of the best individual
         -the fitness of the worst individual
         -the features of the worst individual
    while the stop criterion has not been met:
        while offspring pool size < 10 * popSize
                choose two parents from population via tournament selection
                use two parents to make two children using procreate()
                add the two children to the offspring pool
        merge the offspring pool into the population pool
        sort the new population pool from fittest to least fit
        delete all but the fittest popSize individuals
        increment the iteration counter
        if verbose
                display intermediate results

EvolutionCubic()
    fill the population pool to popSize with individuals of type IndivCubic
    sort the population pool from lowest to best fitness (lowest fitness being the best)
    if verbose
        display intermediate results:
         -number of iterations
         -the fitness of the best individual
         -the features of the best individual
         -the fitness of the worst individual
         -the features of the worst individual
    while the stop criterion has not been met:
        while offspring pool size < 10 * popSize
                choose two parents from population via tournament selection
                use two parents to make two children using procreate()
                add the two children to the offspring pool
        merge the offspring pool into the population pool
        sort the new population pool from fittest to least fit
```

```
        delete all but the fittest popSize individuals
        increment the iteration counter
        if verbose
                display intermediate results
```

	Selected Design	Rejected Design
uses the same classes to optimize any function	yes	no
succint	yes	no
uses polymorphism	yes	no