Problem 1

Write a function `string replace(string str, string s1, string s2)` that returns a copy of `str` with all occurrences of `s1` replaced with `s2`. For example, if `str` is equal to `"London"`, `s1` is equal to `"on"`, and `s2` is equal to `"xyz"` then `replace(str, s1, s2)` would return `"Lxyzdxyz"`. Test this function in a main of your choice.

Problem 2

Without declaring any variable and without modifying the contents of the array, complete the function `max` so that it returns the maximum value in an array of integers between the indices specified by `start` and `end`.

```
int max(int a[], int start, int end)
{
     …
}
```

Problem 3

Write a function `string chooseRandomly(string *names, int N)` that returns a name at random from the dynamic array pointed to by `names` and whose size is equal to `N`. Also, test this function in a main that repeatedly selects student names at random (you can read the names from a file as in exercise 1 of assignment 10).

Problem 4

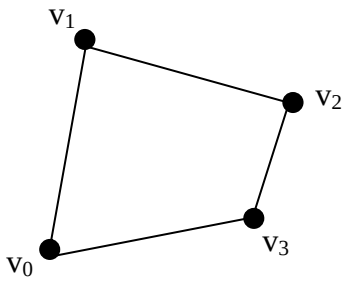
Write a **recursive** function `int ternarySearch(int x, int A[], int start, int end)` that searches for x in the **sorted** array A. In contrast with binary search, this function splits the array at two locations (`third` and `twoThirds`) which results in three nearly equal parts: (`start`, `third`), (`third+1`, `twoThirds`), and (`twoThirds+1`, `end`). If `x` is not found at `A[third]` and `A[twoThirds]` then the function recursively searches in one of the three parts. Define the base case as the one where `end-start` ≤ 5. If the base case is reached, you should search for the element sequentially.

Problem 5

Write and test a new sorting algorithm called `combinedSort`. This algorithm splits the array to be sorted into two equal parts. It sorts the first part using `bubbleSort` and the second part using `selectionSort`. Then it combines the two parts using the merging mechanism employed in `mergeSort`.

Problem 6

In geometry, polygons are represented using a set of vertices that are joined with line segments to form a closed chain. For example, in the figure below, the vertices v_0, v_1, v_2, and v_3 form a quadrilateral, which is a polygon with 4 vertices.



In this exercise, you are required to develop C++ code to represent polygons as follows:

- Define a class `Vertex` which is similar to class `Point` used in assignment 8 except that the constructor doesn't initialize the coordinates to random values. You also need to add a function `double distance(Vertex other)` that computes the Euclidean distance between the calling object and `other`.
- Define a class `Polygon` with the following members:
    - Two private members: `int nbVertices` and `Vertex *vertices`. The second variable will be used to point to a dynamic array that stores the vertices of a given polygon. Choosing a pointer is suitable in this case as the number of vertices might vary from one instance of `Polygon` to another.
    - A constructor `Polygon(int n=3);` that sets `nbVertices` to `n` and dynamically creates an array of n vertices pointed to by `vertices`. Initially, these vertices would be initialized according to the default constructor of `Vertex`.

    - A function `void set(int i, int a, int b);` that sets the coordinates of the vertex at index `i` to `a` and `b`.
    - A function `double perimeter();` that computes the perimeter of the polygon. This can be done by computing the sum of the distances between every two consecutive vertices.
- Using the above, write a main that reads the necessary information about one polygon from the user and computes its perimeter.

Problem 7

Write the following functions and test them in a main of your choice:

- `void appendElement(int x, int* &A, int sizeA)`. This function appends `x` to the end of the dynamic array pointed to by `A`. To implement this function, you have to create a new array whose size is equal to `sizeA+1`, copy the elements of the original array to the new one, add `x` to the end, delete the old array, and let `A` point to the new one.
- `void insertElement(int x, int index, int* &A, int sizeA)`. This function is similar to the above except that `x` is inserted between the elements at locations `index` and `index+1`.
- `void deleteElement(int index, int* &A, int sizeA)`. This function deletes the element at `index`. That is, you should create a new dynamic array whose size is equal to `sizeA-1`, copy to it all the elements of the original array except `A[index]`, delete the old array, and let `A` point to the new one.