

Question 1:

Implement the function:

```
string* createWordsArray(string sentence, int& outWordsArrSize)
```

This function gets a string `sentence` containing a sentence.

When called, it should create and return a new array (of strings), that contains all the words in `sentence`. The function should also update the output parameter, `outWordsArrSize`, with the logical size of the new array that was created.

Note: Assume that the words in the sentence are separated by a single space.

For example, if `sentence="You can do it"`, after calling `createWordsArray(sentence, outWordsArrSize)`, the function should create and return an array that contains `["You" , "can" , "do" , "it"]`, and update the value in `outWordsArrSize` to be 4.

Implementation requirements:

1. You may want to use some of the string methods, such as `find`, `substr`, etc.
2. Your function should run in linear time. That is, if `sentence` contains n characters, your function should run in $\theta(n)$.
3. Write a `main()` program that tests this function..

Question 2:

Implement the function:

```
int* findMissing(int arr[], int n, int& resArrSize)
```

This function gets an array of integers `arr` and its logical size `n`. All elements in `arr` are in the range $\{0, 1, 2, \dots, n\}$.

Note that since the array contains `n` numbers taken from a range of size $n+1$, there must be at least one number that is missing (could be more than one number missing, if there are duplicate values in `arr`).

When called, it should create and return a new array, that contains all the numbers in range $\{0, 1, 2, \dots, n\}$ that are not in `arr`. The function should also update the output parameter, `resArrSize`, with the logical size of the new array that was created.

For example, if `arr=[3, 1, 3, 0, 6, 4]`, after calling `findMissing(arr, 6, resArrSize)`, the function should create and return an array that contains `[2, 5]`, and update the value in `resArrSize` to be 2.

Implementation requirements:

1. Your function should run in **linear time**. That is, it should run in $\theta(n)$.
2. Write a `main()` program that tests this function..

Question 3:

In this question, you will write **two versions** of a program that reads from the user a sequence of positive integers ending with -1, and another positive integer `num` that the user wishes to search for.

The program should then print all the line numbers in sequence entered by the user, that contain `num`, or a message saying that `num` does not show at all in the sequence.

Your program should interact with the user **exactly** as it shows in the following example:

Please enter a sequence of positive integers, each in a separate line.

End you input by typing -1.

13

5

8

2

9

5

8

8

-1

Please enter a number you want to search.

5

5 shows in lines 2, 6.

- a) The first version of the program, is not allowed to use the `vector` data structure.
- b) The second version of the program, should use the `vector` data structure.

Implementation requirements (for both programs):

1. Think how to break down your implementation to functions.
2. Your programs should run in **linear time**. That is, if there are n numbers in the input sequence, your program should run in $\theta(n)$.
3. Write the two programs in two functions named `main1()` and `main2()`. Also have the `main()` test these two functions.