Problem 1: Get val soon! (15 marks)

In this problem, we will model a very simple machine. This machine operates on integers. The operation of this machine is described in the next paragraph.

This machine has a `storage` and operates on an `input`. The input consists of space separated `integers` and `operators`. The operator types are `+`, `-`, `*`.

Whenever the machine gets an `integer` as input, it puts the integer into the storage. Whenever the machine gets an operator as input, it removes the last `two` integers from the storage, performs the relevant operation on these integers and finally stores the result back into the storage.

While performing the operation, the number which was stored `later`, in the storage, goes on the `RHS` of the operator, and the other number goes to the `LHS`.

Your task is to implement an efficient method to compute the result of a given input sequence. The input sequence could contain any number of space separated entities (integers or operators).

You have to implement the following function:

```
void compute_result(string infilename, string outfilename) {

        // Complete the  function. DO NOT WRITE A MAIN

        // DO NOT CHANGE THE FUNCTION NAME

        // Write your code below

}
```

Here, `infilename` and `outfilename` are the input and output filenames respectively.

EXAMPLE

Consider the input sequence as follows, read from the `inputefilename`:

```
20 32 + 11 6 * -
```

Below is the description of how your machine will work.

```
input = 20 32 + 11 6 * -

storage = [ ] , empty storage initially
```

item_scanned = 20, 1st character of the input sequence

```
storage = [ 20 ], 20 goes inside the storage
```

item_scanned = 32, 2nd character of the input sequence

```
storage = [ 20 32 ], 32 goes inside the storage
```

item_scanned = " + " (without quotes), 3rd character of the input sequence

```
storage = [ 52 ], 20 + 32 = 52 goes inside the storage
```

item_scanned = 11, 4th character of the input sequence

```
storage = [ 52 11 ], 11 goes inside the storage
```

item_scanned = 6, 5th character of the input sequence

```
storage = [ 52 11 6 ], 6 goes inside the storage
```

item_scanned = " * " (without quotes), 6th character of the input sequence

```
storage = [ 52 66 ], 11 * 6 = 66 goes inside the storage
```

item_scanned = " - " (without quotes), 7th and last character of the input sequence

```
storage = [ -14 ], 52 - 66 = -14 goes inside the storage
```

Hence the final output stored in the file `outfilename`, is:

```
-14
```

CONSTRAINTS

- The size of the input sequence varies from `15` to `255`
- Time limit: `2s`
- You can safely assume that the given input sequence will be a valid one

Problem 2: Goods for nothing! (15 Marks)

You have to deliver goods to a group of houses which lie along a circle. At every house, you collect some amount of money in exchange for the goods. The next house is some distance apart and you have to spend `1` unit of money to travel `1` unit of distance towards the next house.

You and your friend begin the delivery process at one of the houses, with no money in your pocket. Your friend decides to wait at the starting point while you have to visit all the houses, in the order they appear along the circle and meet your friend back at the starting position.

If after delivering the good at a house, the net amount of money in your pocket goes less than the amount required to travel to the next house, you won't be able to go any further and hence won't be able to complete the delivery process.

You will be given a list containing the payment to be collected at every house and the distance to the next house, in order, along the circle. Your task is to output the `first` such index of a house in the list, from which you should start, so that you can successfully complete the delivery process and reach back at the starting point.

You have to implement the following function:

```
void compute_first_index(string infilename, string outfilename) {

        // Complete the  function. DO NOT WRITE A MAIN

        // DO NOT CHANGE THE FUNCTION NAME

        // Write your code below

}
```

EXAMPLE

Consider the following list of houses, as read from file `infilename`:

```
4 6

6 5

7 3

4 5
```

Here, The every line corresponds to a house. Thus, the `4` lines correspond to house numbers `0` to `3`.

In each line, the first number corresponds to the `payment` to be collected from that house and the second number corresponds to the `distance` for the next house from this house.

Let us see what is the first such index, starting from `0` from where we can successfully complete the tour.

- Starting from index `0`
    - Right at the first house, you collect `4` units of money and require `6` units for money to reach the next house. Hence the delivery process can't continue
- Starting from index `1`
    - At the first house (index `1`), you collect `6` units and have to spend `5` units to reach the next house.
    - At the next house (index `1`), you have `1` unit of money remaining. Here, you collect `7` units of money and have to spend `3` units to reach the next house.
    - At the next house (index `2`), you have `5` unit of money remaining. Here, you collect `4` units of money and have to spend `5` units to reach the next house.
    - At the next house (index `0`), you have `4` unit of money remaining. Here, you collect `4` units of money and have to spend `6` units to reach the next house.
    - Finally you reach house `1` again.
- Hence, the output for the given input will be `1`

The final output written in file `outfilename`, will be:

```
1
```

CONSTRAINTS

- The size of the list varies from `1` to `40000`.
- The time limit will be `2s`.
- You can assume that there will `always` exist an index starting at which, you can complete the circle.

HINTS

- There exist a solution where you can start from all the indices and check if the tour can be completed. What is the complexity of such a solution?
- We now want to find a solution which runs in time linear in the number of households, i.e O(n). How can we do that?
- Think about what structure among the ones you have read, can you use to traverse the households?
- If we start from an index and then at some point realise that we can't successfully complete the delivery process, we would have already computed the net amount required to reach the current point from the starting point. How can we avoid recomputing this value when we try out the next index? Can we reuse this value in some way to compute the net amount when we start from the next index?

PROBLEM

You have to a implement a very simple plagiarism checker. We say 2 students have copied from each other if -

- •The number of unique words their answers have in common is greater than or equal to a given threshold X.
- •Out of the words that are common, at least Y percentage have the same number of occurrences in both the answers.

Given **s1** and **s2** corresponding to the students' answers and 2 integers X and Y corresponding to the above thresholds, you have to implement the function **bool plagiarism_checker(string s1, string s2, int X, int Y)**, which will return true, if the answers are have been copied, false otherwise.

NOTE:-

- The strings will not contain any punctuation.
- Words are case sensitive- "Apple" and "apple" are different words.
- You should **ONLY** write this function. Do not write a **MAIN FUNCTION.** If you do so, your code will not compile and will be awarded a zero.
- Also, **do not change the name of the function** and fill in your code, only write in the function body provided.
- Also, you do not need to take any input from the user. So, do not use any cin or cout statements.

CONSTRAINTS

- Total Execution Time = 6s

EXAMPLE:

Consider X = 8, Y=80, and the 2 strings:

```
s1 = "A set can be used to count the number of unique words in some text"

s2 = "Sets which are a part of the STL library of collections are used to count the number of
words in texts"
```

There are 9 common words, and hence the 1st condition is satisfied.

```
a, used, to, count, the, number, of, words, in
```

Out of these, 7 words: **a, used, to, count, number, words, in**, have the same number of occurences in both s1 and s2. Since 7/9 is less than 80%, the second condition is not satisfied and hence the answer is false.