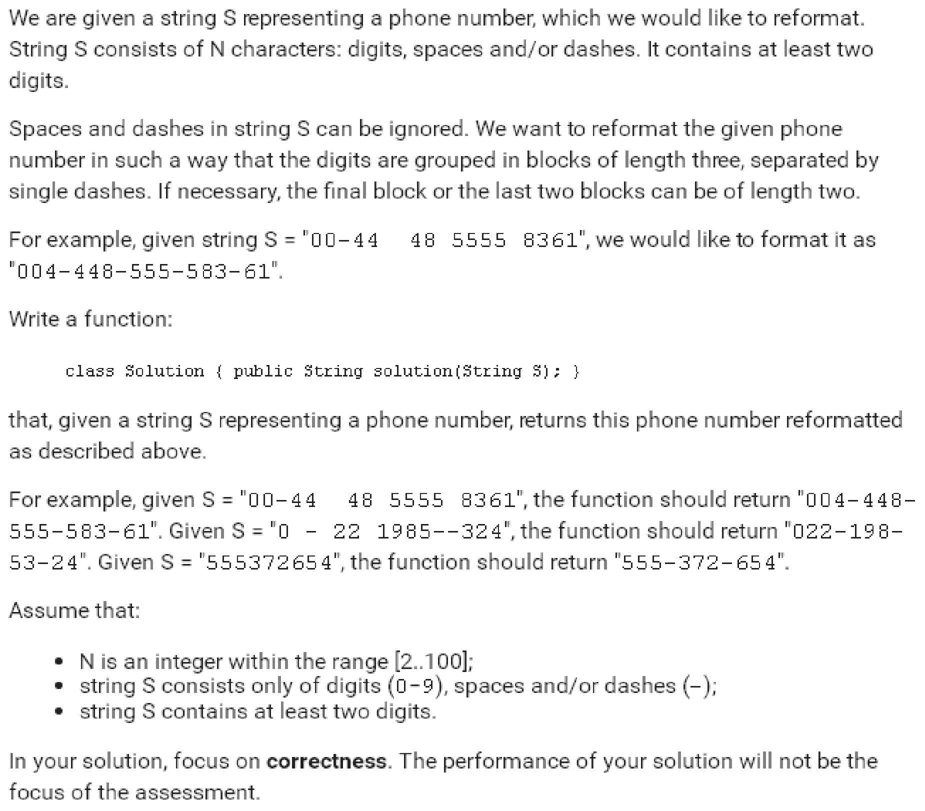
Volant Trading – Online test in 2.5hr *2016 Nov 06*

Question 1 (my score 100)



We are given a string S representing a phone number, which we would like to reformat. String S consists of N characters: digits, spaces and/or dashes. It contains at least two digits.

Spaces and dashes in string S can be ignored. We want to reformat the given phone number in such a way that the digits are grouped in blocks of length three, separated by single dashes. If necessary, the final block or the last two blocks can be of length two.

For example, given string S = "`00-44  48 5555 8361`", we would like to format it as "`004-448-555-583-61`".

Write a function:

```
class Solution { public String solution(String S); }
```

that, given a string S representing a phone number, returns this phone number reformatted as described above.

For example, given S = "`00-44  48 5555 8361`", the function should return "`004-448-555-583-61`". Given S = "`0 - 22 1985--324`", the function should return "`022-198-53-24`". Given S = "`555372654`", the function should return "`555-372-654`".

Assume that:

- N is an integer within the range [2..100];
- string S consists only of digits (`0-9`), spaces and/or dashes (`-`);
- string S contains at least two digits.

In your solution, focus on **correctness**. The performance of your solution will not be the focus of the assessment.

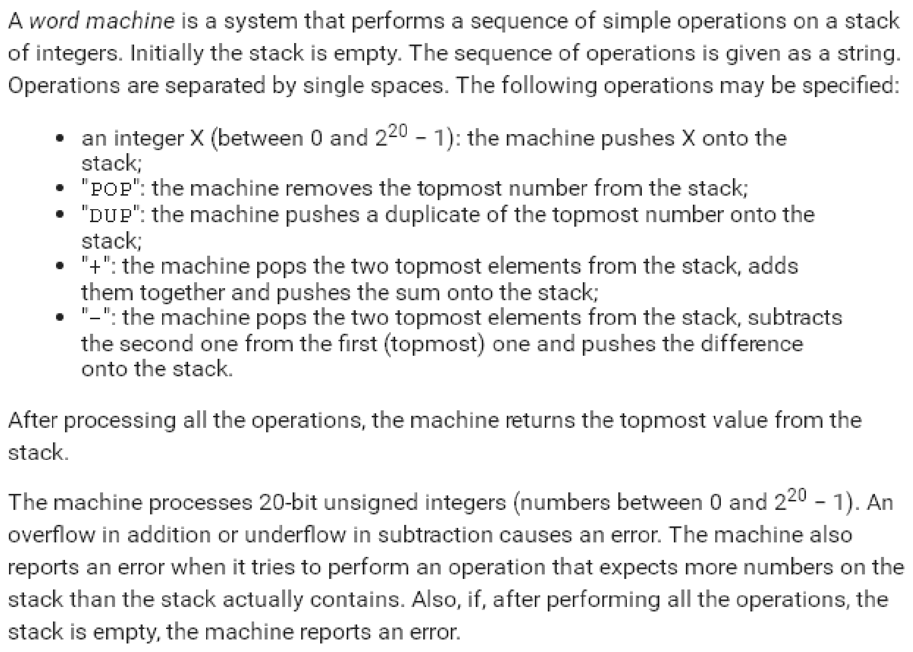
Answer

```
std::string number_convert(const std::string& s)
{
	std::string s0;
	std::string s1;

	std::for_each(s.begin(), s.end(), [&s0](char c){ if (isdigit(c)) s0.push_back(c); });

	if (s0.size() <= 3)
	{
		s1 = s0;
	}
	else if (s0.size()%3 == 0 || s0.size()%3 == 2)
	{
		for(int n=0; n!=s0.size(); ++n)
		{
			s1.push_back(s0[n]);
			if ((n+1)%3==0 && n+1!=s0.size()) s1.push_back('-');
		}
	}
	else // if (s0.size()%3 == 1)
	{
		for(int n=0; n!=s0.size(); ++n)
		{
			s1.push_back(s0[n]);
			if ((n+1)%3==0 && n+1<=s0.size()-4 || n+1==s0.size()-2) s1.push_back('-');
		}
	}
	return s1;
}
```

Question 2 (my score 91)



A *word machine* is a system that performs a sequence of simple operations on a stack of integers. Initially the stack is empty. The sequence of operations is given as a string. Operations are separated by single spaces. The following operations may be specified:

- an integer X (between 0 and 2^{20} − 1): the machine pushes X onto the stack;
- "POP": the machine removes the topmost number from the stack;
- "DUP": the machine pushes a duplicate of the topmost number onto the stack;
- "+": the machine pops the two topmost elements from the stack, adds them together and pushes the sum onto the stack;
- "−": the machine pops the two topmost elements from the stack, subtracts the second one from the first (topmost) one and pushes the difference onto the stack.

After processing all the operations, the machine returns the topmost value from the stack.

The machine processes 20-bit unsigned integers (numbers between 0 and 2^{20} − 1). An overflow in addition or underflow in subtraction causes an error. The machine also reports an error when it tries to perform an operation that expects more numbers on the stack than the stack actually contains. Also, if, after performing all the operations, the stack is empty, the machine reports an error.

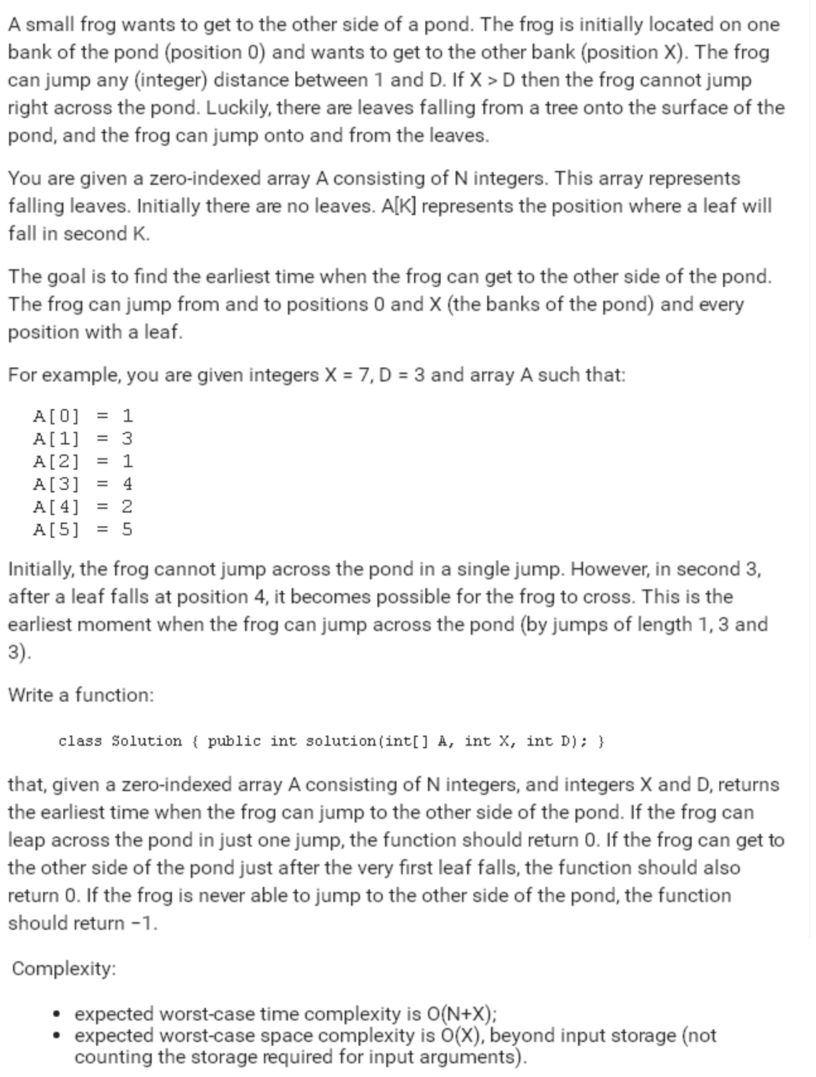
Answer

```
      signed long machine(const std::string& s)
      {
            std::stack<unsigned long> my_stack;
            unsigned long max_limit = (2<<19);
            size_t pos0 = 0;
            size_t pos1 = 0;

            while(pos0 < s.size())
            {
                  std::string str;
                  pos1 = s.find(“ “, pos0);
                  if (pos1!=std::string::npos)
                        { str = s.substr(pos0,pos1-pos0);          pos0 = pos1+1;          }
                  else  { str = s.substr(pos0);                    pos0 = s.size();        }

                  if (isdigit(str[0]))                             my_stack.push(std::stoul(str));
                  else if (str == “POP” && str.size()>=1)          my_stack.pop();
                  else if (str == “DUP” && str.size()>=1)          my_stack.push(my_stack.pop());
                  else if (str == “+” && str.size()>=2)
                  {
                        unsigned long i0 = my_stack.top();         my_stack.pop();
                        unsigned long i1 = my_stack.top();         my_stack.pop();
                        unsigned long i2 = i0+i1;
                        if (i2 < max_limit) my_stack.push(i2);     else return -1;
                  }
                  else if (str == “-” && str.size()>=2)
                  {
                        unsigned long i0 = my_stack.top();         my_stack.pop();
                        unsigned long i1 = my_stack.top();         my_stack.pop();
                        if (i0 >= i1) my_stack.push(i0-i1);        else return -1;
                  }
                  else return -1;
            }
            if (my_stack.size()>0) return my_stack.top();
            else return -1;
      }
```

Question 3 (my score 100)



A small frog wants to get to the other side of a pond. The frog is initially located on one bank of the pond (position 0) and wants to get to the other bank (position X). The frog can jump any (integer) distance between 1 and D. If X > D then the frog cannot jump right across the pond. Luckily, there are leaves falling from a tree onto the surface of the pond, and the frog can jump onto and from the leaves.

You are given a zero-indexed array A consisting of N integers. This array represents falling leaves. Initially there are no leaves. A[K] represents the position where a leaf will fall in second K.

The goal is to find the earliest time when the frog can get to the other side of the pond. The frog can jump from and to positions 0 and X (the banks of the pond) and every position with a leaf.

For example, you are given integers X = 7, D = 3 and array A such that:

```
  A[0] = 1
  A[1] = 3
  A[2] = 1
  A[3] = 4
  A[4] = 2
  A[5] = 5
```

Initially, the frog cannot jump across the pond in a single jump. However, in second 3, after a leaf falls at position 4, it becomes possible for the frog to cross. This is the earliest moment when the frog can jump across the pond (by jumps of length 1, 3 and 3).

Write a function:

```
    class Solution { public int solution(int[] A, int X, int D); }
```

that, given a zero-indexed array A consisting of N integers, and integers X and D, returns the earliest time when the frog can jump to the other side of the pond. If the frog can leap across the pond in just one jump, the function should return 0. If the frog can get to the other side of the pond just after the very first leaf falls, the function should also return 0. If the frog is never able to jump to the other side of the pond, the function should return −1.

Complexity:

- expected worst-case time complexity is O(N+X);
- expected worst-case space complexity is O(X), beyond input storage (not counting the storage required for input arguments).

```
int frog_jump(const std::vector<int>& leaves, int max_jump, int target)
{
	int current_pos = 0;

	std::vector<bool> hist(target, false);
	for(int n=0; n!=leaves.size(); ++n) hist[leaves[n]] = true;
	for(int n=0; n!=leaves.size(); ++n)
	{
		// step 1a : jump to next leaf
		if (pos + max_jump >= leaves[n])
		{
			pos = leaves[n];

			// step 1b : re-check fallen leaves
			bool flag = true;
			while(flag)
			{
				flag = false;
				for(int jump=1; jump<=max_jump; ++jump)
				{
					if (hist[pos+jump]) { pos = pos+jump; flag = true; }
				}
			}
		}

		// step 2 : jump to target
		if (pos + max_jump >=target) return n;
	}
	return -1; // never finished
}
```