앵무새 (Parrots)

경화는 열광적인 새 애호가이다. 경화는 새를 이용한 IP 프로토콜인 IP over Avian Carriers (IPoAC) 에 대한 내용을 읽고 앵무새들을 다년간 조련하였다. 그 결과 앵무새들은 **0** 이상 **R** 이하의 정수 하나를 기억할 수 있게 되었다. 경화는 앵무새들을 모두 **K** 마리 조련하였고 이 앵무새들은 똑같이 생겨서 구별이 불가능하다.

이제 그녀는 이 **K** 마리의 앵무새들을 이용하여 메시지 **M**을 먼 곳에 전달하려 한다. 메시지 **M**은 **0** 이상 **255** 이하의 정수 **N** 개로 이루어진 리스트이며 이 리스트에서 같은 정수가 여러 번 나타날 수도 있다.

처음에는 각각의 앵무새에게 정수를 하나씩 기억하게 하고 순서대로 보냈다. 모든 앵무새가 도착지에 잘 도착했지만 **실제 출발 순서대로 도착하지 않아** 메시지를 제대로 복원하는데 실패하고 말았다.

이제 당신의 임무는 앵무새를 이용하여 메시지 전송이 가능하도록 경화를 도와주는 것이다. 그녀는 당신이 아래의 두 가지 작업을 할 수 있는 프로그램을 짜주기를 원한다.

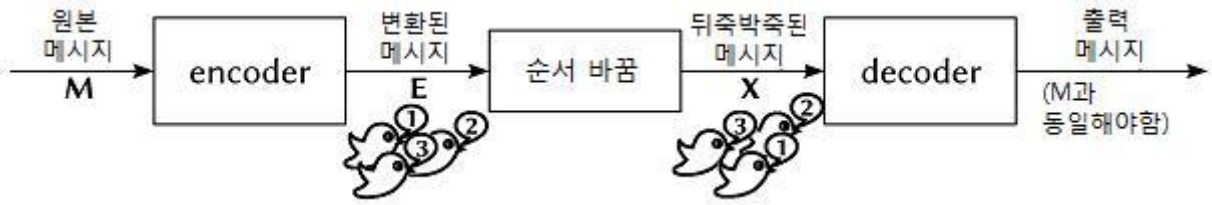
- 먼저 당신의 프로그램은 메시지 **M**을 읽어서 최대 **K** 개의 **0** 이상 **R** 이하의 정수로 이루어진 리스트로 변환해야 한다. 이 정수 리스트를 앵무새들에게 기억시켜서 전송할 것이다.

- 둘째, 당신의 프로그램은 도착한 앵무새들로부터 얻은 **0** 이상 **R** 이하의 정수들로부터 원래의 메시지 **M**을 복원해야 한다. 앵무새의 도착 순서가 출발 순서와 다를 수 있음을 유의하라.

모든 앵무새는 항상 도착지에 도달하며 각각의 앵무새는 숫자를 정확하게 기억하고 있다고 가정해도 좋다. 또한 앵무새의 숫자가 많아야 **K** 마리임을 유의하라.

태스크

두개의 함수 **encode** 와 **decode** 를 작성하라. **encode** 는 앵무새를 보내는 쪽에서 사용하는 함수이며 **decode** 는 앵무새를 받는 쪽에서 사용하는 함수이다. 전체적인 과정은 아래 그림과 같다.



당신이 작성할 함수의 구체적인 내용은 다음과 같다:

- 함수 **encode(N,M)**은 아래의 입력 파라미터를 받는다:
  - **N** – 메시지의 길이
  - **M** – **N** 개의 정수로 이루어진 일차원 배열

    (**0** ≤ **M[i]** ≤ **255**, **0** ≤ **i** < **N**)

  이 함수는 메시지 **M** 을 받아서 **0** 이상 **R** 이하의 정수 리스트로 변환한다. 변환된 정수 리스트를 전송할 때는 각 정수 **a** 마다 **send(a)** 함수를 이용하여 전송해야 한다.

- 함수 **decode(N,L,X)**는 아래의 입력 파라미터를 받는다:
  - **N** – 원래 메시지의 길이
  - **L** – 전송된 메시지의 길이 (앵무새가 전송한 정수의 개수)
  - **X** – 전송된 **L** 개의 정수들을 저장한 배열. 각 정수 **X[i]** (**0** ≤ **i** < **L**)는 **encode** 함수가 만든 정수 값이다. 그러나 **encode** 함수가 전송한 순서와 다를 수 있다.

  이 함수는 원본 메시지를 복원한다. 복원된 메시지를 출력할 때는 각 정수 **b** 에 대해서 **output(b)** 함수를 호출하여 출력한다. 출력하는 순서는 원본 메시지의 순서와 동일해야 한다.

  R 과 **K** 는 입력 파라미터로 주어지지 않음에 유의하라. (자세한 내용은 서브태스크 설명을 참고하라.)

이 문제를 정확하게 풀기 위해서는 당신이 작성한 함수들이 아래의 조건을 만족해야 한다.

- 함수 **encode** 가 함수 **send** 를 이용하여 전송하는 정수들은 서브태스크에 정의된 범위 이내라야 한다.
- 함수 **encode** 가 함수 **send** 를 호출하는 횟수는 **K** 번을 넘을 수 없다. 횟수 **K** 는 서브태스크마다 정의되어 있다. 횟수 **K** 가 메시지의 길이에 따라 변하는 값임을 유의하라.
- 함수 **decode** 는 원본 메시지 **M** 을 정확하게 복원해야 하며 함수 **output(b)** 를 정확하게 **N** 번만 호출해야 한다. 각 **b** 는 **M[0]**, **M[1]**, ..., **M[N-1]**과 동일해야 한다.

마지막 서브태스크에서 당신의 점수는 변환된 메시지와 원본 메시지의 길이의 비율에 따라 달라진다.

예제

N = 3 이고 **M** 이 다음과 같은 경우를 생각해 보자.

M= 10
30
20

이 경우 함수 **encode(N,M)**이 변환한 정수값이 **(7, 3, 2, 70, 15, 20, 3)**이라고 가정해 보자. 이 정수 값들을 전송하려면 다음과 같이 함수 **send** 를 호출해야 한다:

send(7)
send(3)
send(2)
send(70)
send(15)
send(20)
send(3)

앵무새가 다음과 같은 순서대로 도착했다고 생각해 보자: **(3, 20, 70, 15, 2, 3, 7)**. 그러면 함수 **decode** 의 입력 파라미터는 **N=3**, **L=7** 이 되며 **X** 는 다음과 같다.

3
20
70
X= 15
2
3
7

함수 **decode** 는 원본 메시지 **(10, 30, 20)**을 복원하여 함수 **ouput** 을 다음과 같은 순서로 호출해야 한다:

output(10)
output(30)
output(20)

서브태스크

서브태스크 1 (17 점)

- **N = 8** 이고 배열 **M** 에 저장된 정수는 **0** 또는 **1** 이다.
- 각각의 변환된 정수의 범위는 **0** 이상 **R=65535** 이하이다.
- 함수 **send** 는 최대 **K=10×N** 번까지 호출할 수 있다.

서브태스크 2 (17 점)

- **1 ≤ N ≤ 16**.
- 각각의 변환된 정수의 범위는 **0** 이상 **R=65535** 이하이다.
- 함수 **send** 는 최대 **K=10×N** 번까지 호출할 수 있다.

서브태스크 3 (18 점)

- **1 ≤ N ≤ 16**.
- 각각의 변환된 정수의 범위는 **0** 이상 **R=255** 이하이다.
- 함수 **send** 는 최대 **K=10×N** 번까지 호출할 수 있다.

서브태스크 4 (29 점)

- **1 ≤ N ≤ 32**.
- 각각의 변환된 정수의 범위는 **0** 이상 **R=255** 이하이다.
- 함수 **send** 는 최대 **K=10×N** 번까지 호출할 수 있다.

서브태스크 5 (최대 19 점까지)

- **16 ≤ N ≤ 64**.
- 각각의 변환된 정수의 범위는 **0** 이상 **R=255** 이하이다.
- 함수 **send** 는 최대 **K=15×N** 번까지 호출할 수 있다.
- **중요**: 이 서브태스크의 점수는 원본 메시지와 변환된 메시지의 길이의 비율에 의해 정해진다.

  주어진 테스트 케이스 **t** 에 대해 변환된 메시지의 길이 **Lt** 을 원본 메시지의 길이 **Nt** 로 나눈 값을 **Pt (=Lt/Nt)**로 정의하자. **P** 를 모든 **Pt** 값의 최대값이라 하자. 그러면 당신의 점수는 다음의 규칙으로 계산된다:

  - **P ≤ 5** 이면 **19** 점 만점을 얻는다.
  - **5 < P ≤ 6** 이면 **18** 점을 얻는다.
  - **6 < P ≤ 7** 이면 **17** 점을 얻는다.
  - **7 < P ≤ 15** 이면 당신의 점수는 **1 + 2 × (15 - P)** 에서 정수부분이다.
  - **P > 15** 이거나 하나라도 답이 틀리면 **0** 이다.

- **중요:** 서브태스크 **1-4** 경우에는 큰 번호의 서브태스크를 푸는 방법은 작은 번호의 모든 서브태스크를 풀 수 있다. 예를 들어 서브태스크 **3** 을 푸는 방법은 서브태스크 **1** 과 **2** 를 모두 풀 수 있다. 그러나, 서브태스크 **5** 를 푸는 방법은 작은 번호의 서브태스크를 풀지 못하는 경우도 있다. 그 이유는 서브태스크 **5** 의 **K** 값이 작은 번호의 서브태스크보다 크기 때문이다.

구현시 유의사항

제약조건

- 채점환경: 실제 채점 환경에서, 여러분이 제출한 코드는 두 개의 프로그램 **e** 와 **d** 에 포함되어 따로 컴파일되고 독립적으로 실행된다. 여러분의 인코더와 디코더 모듈은 두 실행파일에 모두 링크되지만 **e** 는 **encode** 만 호출하고, **d** 는 **decode** 만 호출한다.
- **CPU** 제한 시간: 프로그램 **e** 는 함수 **encode** 를 50 번 호출하도록 할 것이고, 이 프로그램은 2 초안에 수행이 끝나야 한다. 프로그램 **d** 는 함수 **decode** 를 50 번

호출하도록 할 것이고, 이 프로그램은 2 초안에 수행이 끝나야 한다.

- 메모리 제한: **256 MB**

  유의사항: 스택 메모리 크기에 대한 제한은 명시되지 않는다. 스택 메모리는 전체메모리 사용량에 계산된다.

인터페이스 (API)

- 프로그램 작업폴더: parrots/
- 참가자가 작성할 파일:
  - encoder.c 또는 encoder.cpp 또는 encoder.pas
  - decoder.c 또는 decoder.cpp 또는 decoder.pas

  유의사항(C/C++ 프로그램의 경우): 견본 채점프로그램과 실제 채점 프로그램에서는 encoder.c[pp]와 decoder.c[pp]가 채점프로그램과 함께 링크된다. 그러므로 각 파일 내에 있는 모든 글로벌(전역) 변수는 다른 파일에 있는 변수와 충돌되지 않도록 **static** 으로 선언해야 한다.

- 참가자 인터페이스:
  - encoder.h 또는 encoder.pas
  - decoder.h 또는 decoder.pas
- 채점프로그램 인터페이스:
  - encoderlib.h 또는 encoderlib.pas
  - decoderlib.h 또는 decoderlib.pas
- 견본 채점프로그램 (sample grader): grader.c 또는 grader.cpp 또는 grader.pas

  견본 채점프로그램은 두 번의 라운드로 나누어 실행한다. 각 라운드에서는 먼저 주어진 데이터를 가지고 encode 를 호출한다. 그리고 난 후, 여러분의 함수 encode 가 만든 출력결과를 가지고 decode 를 호출한다. 첫 번째 라운드에서 채점프로그램은 인코드된 메시지에 있는 정수들의 순서를 바꾸지 않는다. 두 번째 라운드에서는 견본 채점 프로그램은 홀수와 짝수 위치의 정수들을 바꾼다. 실제 채점프로그램은 인코드된 메시지에 다양한 종류의 순열을 적용한다. 여러분은 함수 shuffle (C 또는 C++의 경우) 또는 Shuffle (파스칼의 경우)을 수정하여 견본 채점프로그램의 데이터 뒤섞는 방법을 바꿀 수 있다.

또한, 견본 채점프로그램은 인코드된 데이터의 범위와 길이에 대하여 검사를 한다. 특별하게 지정하지 않으면 인코드된 데이터가 **0** 이상 **65535** 이하인지, 그리고 길이가 **10xN** 이하인지를 검사한다. 여러분이 상수 **channel_range** (예를 들어, **65535** 를 **255** 로)와 **max_expansion** (예를 들어 10 을 15 나 7 로)를 수정하여 이것을 바꿀 수 있다.

- 견본 채점프로그램 입력: grader.in.1, grader.in.2, ...
  유의사항: 견본 채점프로그램은 다음과 같은 양식으로 입력을 읽는다:
  - 1 번째 줄: N
  - 2 번째 줄: N 개 정수들 리스트: M[0], M[1], ..., M[N-1]
- 견본 채점프로그램 입력에 대하여 예상되는 출력: grader.expect.1, grader.expect.2, ...
  이 태스크에 대해서 각각의 파일은 정확히 "**Correct.**"를 포함해야 한다.