Llista circular. Llegenda de Josephus U66160_ca

Diu la llegenda que al segle I, l'historiador Josephus es va trobar, juntament amb altres 40 soldats jueus, assetjat per l'exèrcit romà. La situació era tan desesperada que van preferir treure's la vida abans de ser esclaus de Roma. Van decidir posar-se en cercle i anar eliminant una de cada tres persones fins que només en quedés una (que suposadament s'hauria de suïcidar). En Josephus, que volia viure, va calcular ràpidament on posar-se i va ser l'únic supervivent.
Sabent el nombre n de persones i el nombre k del comptador el nostre objectiu és calcular quina serà la persona supervivent. Per resoldre aquest problema utilitzarem una llista circular, simplement encadenada i sense element fantasma, inicialitzada amb els elements $1, 2, 3, \ldots, n$.
Donada la classe *Llista* que permet guardar seqüències d'enters amb una llista simplement encadenada, sense fantasma i circular, cal implementar els mètodes:

```
Llista (nat n);
// Pre: n > 0
// Post: El p.i. és una llista de n elements que contenen els naturals de 1 a n ordenadament.

void josephus(nat k);
// Pre: k > 0, p.i. conté un o més elements
// Post: Al p.i. s'han eliminat un de cada k elements fins que només en queda un.
```

Cal enviar a jutge.org la següent especificació de la classe *Llista* i la implementació dels mètodes dins del mateix fitxer. Indica dins d'un comentari a la capçalera de cada mètode el seu cost en funció del nombre d'elements n de la llista (i de també de k en el mètode *josephus*).

```
#include <cstddef>
using namespace std;
typedef unsigned int nat;

class Llista {
  // Llista simplement encadenada, sense fantasma i circular.

  public:
    Llista ();
    // Pre: True
    // Post: El p.i. és una llista buida.

    Llista (nat n);
    // Pre: n > 0
    // Post: El p.i. és una llista de n elements que contenen els naturals de 1 a n ordenadament.

    ~Llista ();
    // Post: Destrueix els elements del p.i.

    nat longitud() const;
    // Pre: True
```

```
    // Post: Retorna el nombre d'elements del p.i.

    void mostra() const;
    // Pre: True
    // Post: Mostra el p.i. pel canal estàndard de sortida.

    void josephus(nat k);
    // Pre: k > 0, p.i. conté un o més elements
    // Post: Al p.i. s'han eliminat un de cada k elements fins que només en queda un.

  private:
    struct node {
      int info;   // Informació del node
      node *seg;  // Punter al següent element
    };
    node *_prim; // Punter al primer element
    nat _long;   // Nombre d'elements

    // Aquí va l'especificació dels mètodes privats addicionals
};

// Aquí va la implementació dels mètodes Llista(nat n) i josephus i privats addicionals
```

Pots veure més exemples en els jocs de prova públics.

Entrada

L'entrada conté dues línies amb un enter cadascuna. El primer és el nombre d'elements que tindrà la llista inicialment. El segon és el valor k que s'usarà per anar eliminant cada k elements de la llista inicial.

Sortida

Es mostra la llista dues vegades, una després de crear-la amb el nombre d'elements inicials i una altra cop després d'haver eliminat cada k elements fins a deixar-ne només un. Per cada llista s'escriu el nombre d'elements de la llista seguit d'un espai i dels elements de la llista entre claudàtors i separats per espais.

Observació

Només cal enviar la classe requerida i la implementació dels mètodes *Llista*(*natn*) i *josephus* amb el seu cost en funció del nombre d'elements n de la llista (i de també de k en el mètode *josephus*). Pots ampliar la classe amb mètodes privats. Segueix estrictament la definició de la classe de l'enunciat.

Exemple d'entrada 1	Exemple de sortida 1
1 1	1 [1] 1 [1]

Exemple d'entrada 2

```
1
3
```

Exemple de sortida 2

```
1 [1]
1 [1]
```

Exemple d'entrada 3

```
5
2
```

Exemple de sortida 3

```
5 [1 2 3 4 5]
1 [3]
```

Exemple d'entrada 4

```
10
3
```

Exemple de sortida 4

```
10 [1 2 3 4 5 6 7 8 9 10]
1 [4]
```

Exemple d'entrada 5

```
2
2
```

Exemple de sortida 5

```
2 [1 2]
1 [1]
```

Exemple d'entrada 6

```
2
1
```

Exemple de sortida 6

```
2 [1 2]
1 [2]
```

Informació del problema

Autoria: Jordi Esteve

Generació: 2026-01-25T13:02:31.943Z