**Objective:**

- Implementing and analyzing an application of Queue. [*Process Scheduling [Round Robin Algorithm] and some competition problems.*
- Exploring few programming competition problems related to Stack/Queue.
- Exploring Random Number Manipulation.

**Task-1:**

A Double Ended Queue (**Deque (also called as DECK)**) is a queue in which you can insert and remove elements both at the *front* and at the *rear end* of the queue. You task is to create a class **Deque**, with the following interface:

- **void insertAtTail ( T ).**
- **void insertAtHead( T ).**
- **T deleteAtTail( ).**
- **T deleteAtHead( ).**
- **isEmpty():**
- **isFull():**
- **int getCapacity()**
- **int getNoOfElements()**

Write routines to support the deque that take O(1) time per operation.

**Task-2:**

In this task you are required to simulate ***Schedule***r. Scheduler is something which is responsible for assigning the CPU time to the processes.

- The ***Scheduler*** should take in ***Processes*** and add them to a Queue.
- Once all the ***Processes*** are read, your program would execute each.
- But before executing any process, the program should print the name and other information about the process.
- Now it should start executing the processes in the Queue. For each process, execution just means that process stays at the head of the Queue until time equal to its ***execution time*** has passed. The process is deleted from the list after that. At this moment your program should print that such and such process has finished execution. E.g. ***Messenger.exe, 6 completed execution***
- You must remember that this is a simulator. This means that you'll have to define your own timer, one which increments in unit steps.
- After the passage of every 10 time units, you program must stop the processing of current process and start the next process, the current process will be taken from the front of the queue to the end of the queue and the time remaining to complete process should be decremented.

Hints:

Design a class "Process" which will have following data members,

- int processId
- string processName
- int executionTime
- And a method to display its state.

Create a circular Queue and add the process in it.

Process the "Process" at front of Queue and then remove it from front and add at the end and continue the processing. Remove the "Process" completely from queue if it has "0" "execution time" remaining.

Repeat this process until the Queue is not empty.

*You have to create the following files and will have to code all the classes listed in these files.

**//DRIVER.CPP**

```
#include<iostream.h>
#include "schedular.h"

void main()
{
    Process arr[4] = { Process(1,"notepad",20), Process(13,"mp3player",5),
Process(4,"bcc",30), Process(11,"explorer",2) };
    Schedular s(arr, 4, 5);
    s.assignProcessor();
}
```

**//PROCESS.h**

```
class Process
{
public:
    int processId;
    string processName;
    int processExecTime;
    Process(int id=1, string pName="notepad", int burstTime=10 );
};
```

**//SCHEDULAR.h**

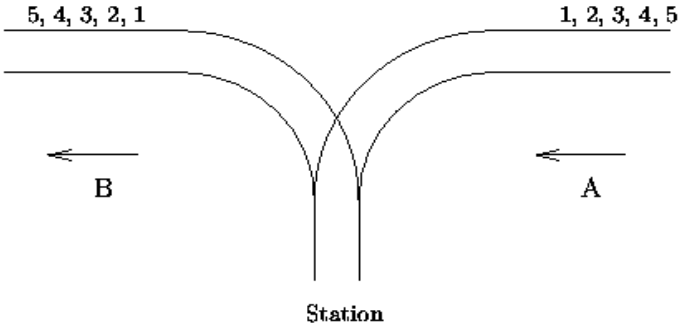
```
#include<iostream.h>
#include "CSTRING.h"
#include "queue.h"

class Schedular
{
    Process* processArray;
    int processArrayLength;
    int timeQuantum;

public:

    Schedular( Process* p, int length, int quantum );

    void assignProcessor();

};
```

**//QUEUE.h**

```
template<class T>
class Queue
{
    int front;
    int rear;
    int capacity;
    T* data;
public:
    Queue( int size );
    void enQueue ( T p );
    T deQueue ();
    int isEmpty ();
    int isFull ();
}
```

**Task-3:** **Rails** Time limit: 3.000 seconds

There is a famous railway station in PopPush City. Country there is incredibly hilly. The station was built in last century. Unfortunately, funds were extremely limited that time. It was possible to establish only a surface track. Moreover, it turned out that the station could be only a dead-end one (see picture) and due to lack of available space it could have only one track.




The local tradition is that every train arriving from the direction A continues in the direction B with coaches reorganized in some way. Assume that the train arriving from the direction A has $N \le 1000$ coaches numbered in increasing order $1,2,3,...,N$. The chief for train reorganizations must know whether it is possible to marshal coaches continuing in the direction B so that their order will be $a_1, a_2, ..., a_n$. Help him and write a program that decides whether it is possible to get the required order of coaches. You can assume that single coaches can be disconnected from the train before they enter the station and that they can move themselves until they are on the track in the direction B. You can also suppose that at any time there can be located as many coaches as necessary in the station. But once a coach has entered the station it cannot return to the track in the direction A and also once it has left the station in the direction B it cannot return back to the station.

### *Input*

The input file consists of blocks of lines. Each block except the last describes one train and possibly more requirements for its reorganization. In the first line of the block there is the integer $N$ described above. In each of the next lines of the block there is a permutation of $1,2,3,...,N$. The last line of the block contains just 0. The last block consists of just one line containing 0.

### Output

The output file contains the lines corresponding to the lines with permutations in the input file. A line of the output file contains Yes if it is possible to marshal the coaches in the order required on the corresponding line of the input file. Otherwise it contains No. In addition, there is one empty line after the lines corresponding to one block of the input file. There is no line in the output file corresponding to the last "null"' block of the input file.

### *Sample Input*

```
5
1 2 3 4 5
5 4 1 2 3
0
6
6 5 4 3 2 1
0
0
```

### Sample Output

```
Yes
No

Yes
```

**Task-4:** **Messing Up Cards** Time Limit: 3 Seconds

A **card game** is any game using playing cards as the primary device with which the game is played, be they traditional or game-specific. Countless card games exist, including families of related games (such as poker). A small number of card games played with traditional decks have formally standardized rules, but most are folk games whose rules vary by region, culture, and person.

Many games that are not generally placed in the family of card games do in fact use cards for some aspect of their gameplay. Similarly, some games that are placed in the card game genre involve a board. The distinction is that the gameplay of a card game primarily depends on the use of the cards by players (the board is simply a guide for scorekeeping or for card placement), while board games (the principal non-card game genre to use cards) generally focus on the players' positions on the board, and use the cards for some secondary purpose.

You are to accomplish here a basic task used in a popular Card game. Consider a deck of cards with numbers 1 to n. deck is sorted with card 1 at the top and last card i.e. n at the bottom. The basic operation which you will perform on the given deck is as follows:

- ➔ Discard the card at the top and move next card (which is now on top) to the beck bottom.

You are to find the sequence in which cards are removed/discarded from the deck and the last card remaining on the deck.

Each line in the input file contains a number n<=50 and the last line contains 0 means termination of input. For each value of n you will write two line in output file:

- First line shows the sequence in which cards are discarded and
- In the next line you will write the number of last card on the deck.

**Sample input**

```
7
19
10
6
0
```

**Output for sample input**

```
Removed Sequence: 1, 3, 5, 7, 4, 2
Last card: 6
Removed Sequence: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8, 12, 16, 2, 10, 18, 14
Last card: 6
Removed Sequence: 1, 3, 5, 7, 9, 2, 6, 10, 8
Last card: 4
Removed Sequence: 1, 3, 5, 2, 6
Last card: 4
```

**Task-5:**
A *randomized queue* is similar to a stack or queue, except that the item removed is chosen uniformly at random from items in the data structure. Create a generic data type RandomizedQueue as follows:

```
template<typename T>
class RandomizedQueue
{
private:
      T * queue;
      int numberOfItems;
      int capacity;
public:
      RandomizedQueue(); // construct an empty randomized queue.
      bool isEmpty(); // is the queue empty?
      bool isFull(); // is the queue full?
      int getNumberOfElements(); // return the number of items in queue.
      int getCapacity(); // return the capacity.
      void enqueue(T item); // add the item.
      void resize(int newSize); //resize as we did it in stack
      T dequeue(); // delete and return a random item.
      T sample(); // return (but do not delete) a random item.
};
```

In order to implement this task you may like to look into following code snippets

***Code Snippet – 1***

```
int main()
{
    RandomizedQueue<int> rq;
    rq.enqueue(1);
    rq.enqueue(2);
    rq.enqueue(3);
    rq.enqueue(4);
    cout<<rq.getCapacity()<<endl;
    cout<<rq.dequeue();
    cout<<rq.dequeue();
    cout<<rq.dequeue();
    cout<<endl<<rq.getCapacity();


    const int nrolls = 10000; // number of experiments
    const int nstars = 95;    // maximum number of stars to distribute

    std::default_random_engine generator;
    std::uniform_int_distribution<int> distribution(0,9);

    int p[10]={};

    for (int i=0; i<nrolls; ++i)
    {
        int number = distribution(generator);
        ++p[number];
    }

    std::cout << "uniform_int_distribution (0,9):" << std::endl;
    for (int i=0; i<10; ++i)
        cout << p[i]<<endl;

    cout<<endl<<"random number "<<generator();
    return 1;
}
```

***Code Snippet – 2***

```
//http://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
int main()
{
      std::random_device rd;
      std::mt19937 gen(rd());
      std::uniform_int_distribution<> dis(1, 6);

      int f[6]={0};
      int x;
      for (int n=1; n<=100; ++n)
      {
          x = dis(gen);
          cout<<"\n"<<x;
          f[dis(gen)-1]++;
          }
          for (int n=0; n<6; ++n)
          cout<<"\n"<<f[n];

          std::cout << '\n';
      }
}
```