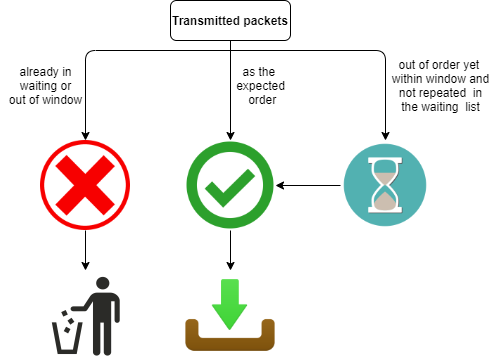
|  |  |  |  |
| --- | --- | --- | --- |
| رقم المقعد | الفصل | رقم الجلوس | اسم الطالب |
| **40** | **3** | **33170** | محمد جمال رشدي عبد الفتاح |

**Ordered delivery of packets**

# Overview

As the title suggests this project aims to deliver packets in the correct succession even if they are transmitted out of order due to unpredictable network conditions, this is done using a circular buffer (circular priority queue). The **main categories of the structures** used are **arrays** and **queues**.

## Structures

* **TX (array):** Transmitted packets IDs
* **RX (array):** Received packets IDs
* **W (queue):** A circular priority queue for the waiting packed IDs
* **D (array):** dropped packets IDS

## Operations

Figure 1 an illustration of the project’s main idea

* **Receive:** if the packet ID transmitted is the expected packet ID
* **Enqueue(waiting):** if a packet ID is not expected yet it is within the windows limits
* **Dequeue(waiting):** if the expected packet ID is in the waiting queue
* **Drop:** if the transmitted packet ID is out of window or already in the waiting list

## Control parameters

* **SequenceNumBits:** Indicates the **MaxSeqNumber** which is,the max sequence number determines the number of elements in the sequence for e.g. if SeqNumBits =2 , MaxSeqNumber =4 and the elements in the sequence are 0,1,2,3
* **Window Size:** Indicates the maximum window size which can be used (It’s also the maxQueue size)
* **Initial Sequence:** Informs the receiver what packets to expect at the beginning of operation

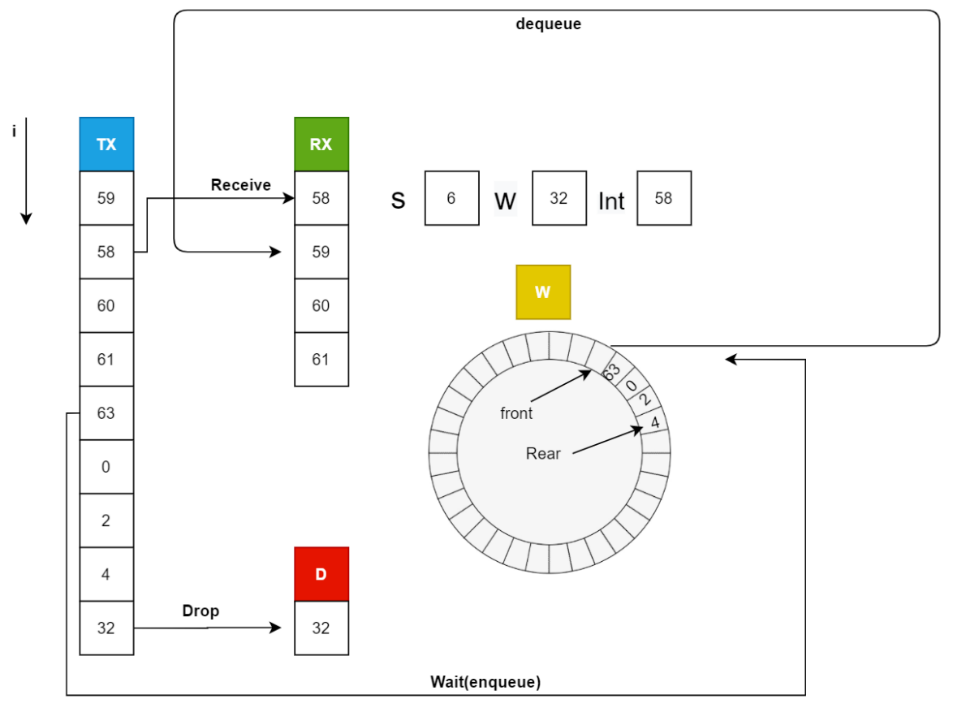


Figure 2 an illustration of the main structures and operations used

# Main Flows

## The main program

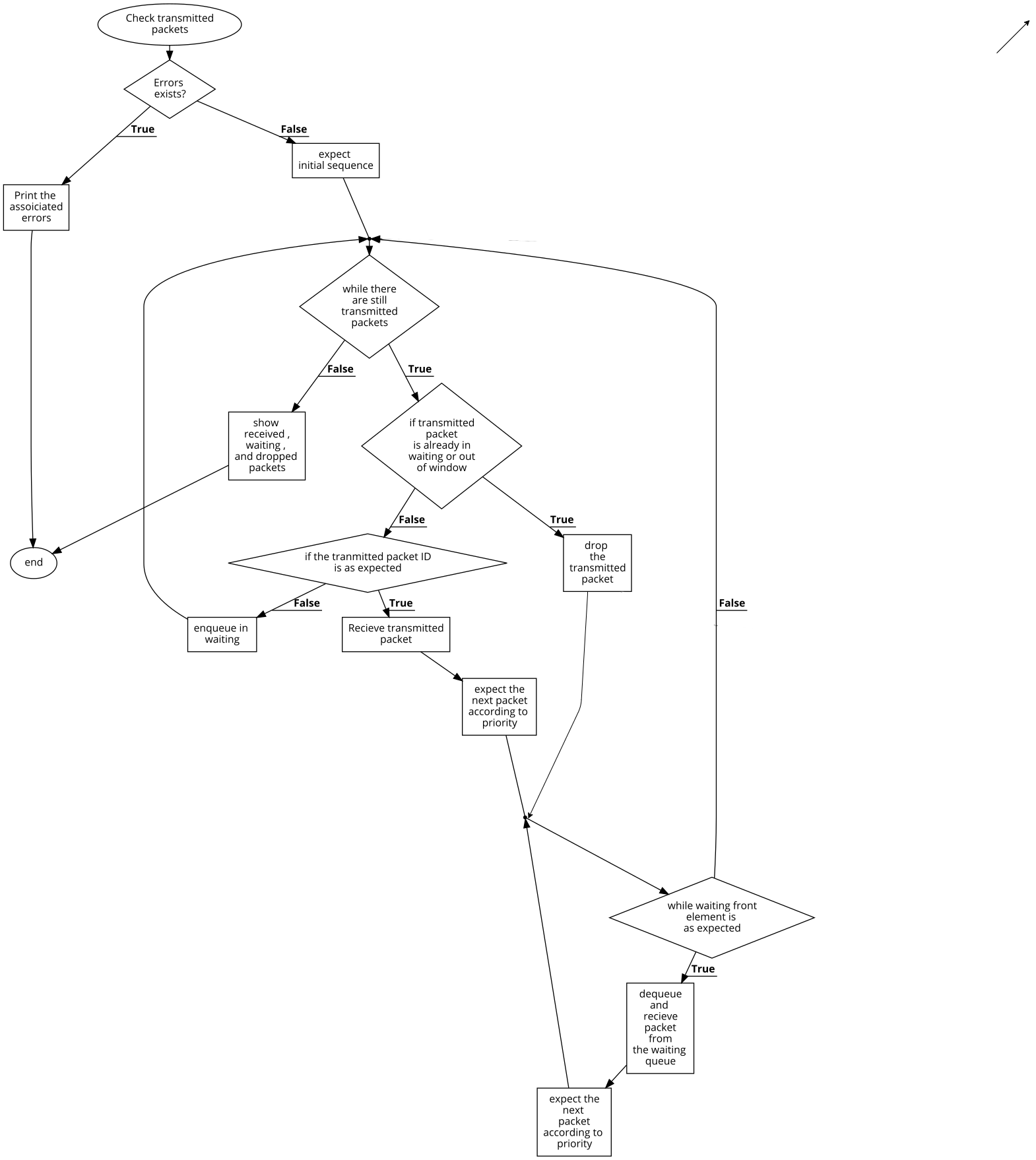
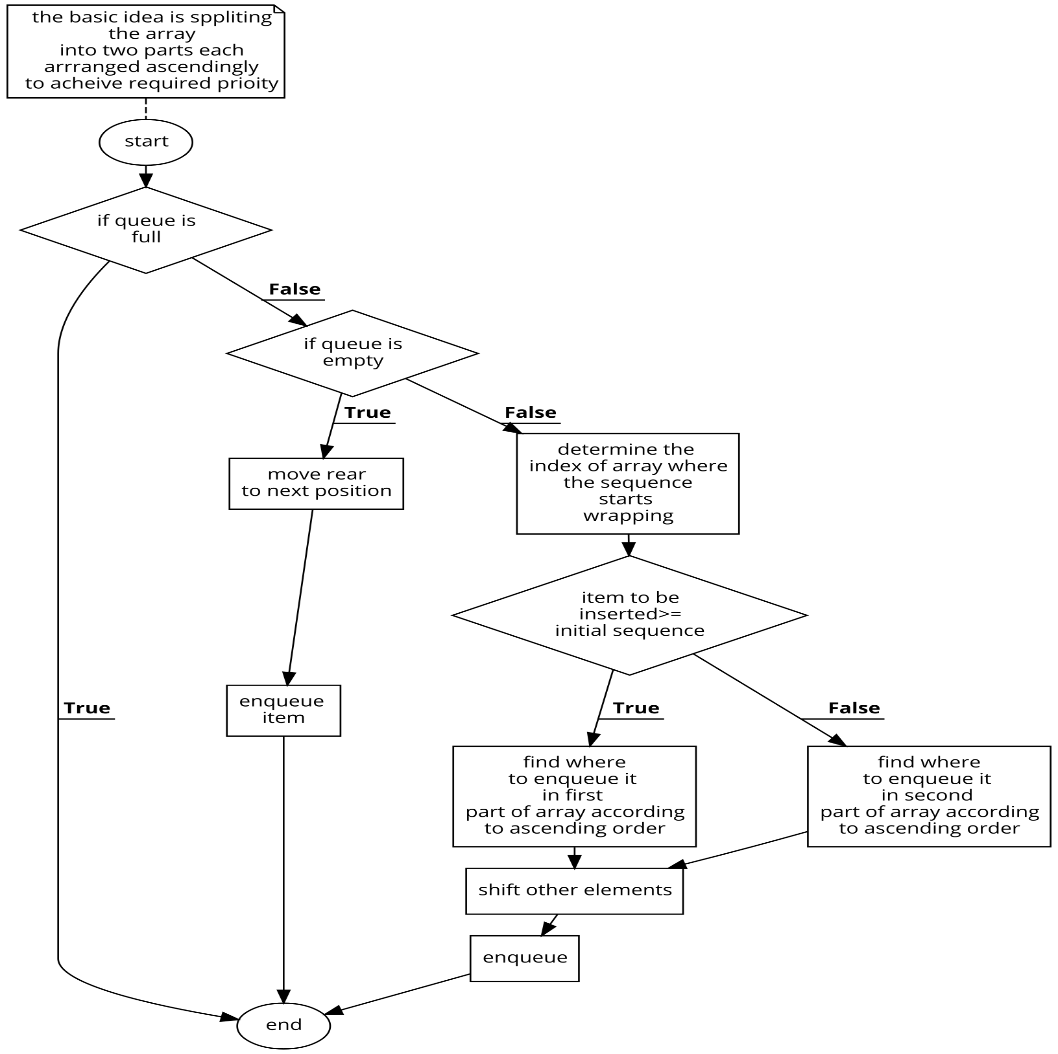


Figure 3 The driving program flowchart



## The enqueue function

if seqNumBits equals 6 and initSeq

equals 61, and the transmitted

packets are 63, 0, 62, 1, 3, 61

respectively the queue must

 store these packets in that order

to achieve this the queue array was

separated into two parts each arranged

ascendingly, the first part includes numbers

that are greater than or equal the initseq

while the second part contains the smaller

ones.

Figure 4 The enqueue function flowchart

# Added functions to the priority queue class

**Prevposition:** works like i-- but in a circular queue

int CPQueue::prevPosition(int pointer)

{

return ((pointer+maxQueue-1) % maxQueue);

}

**Iswaiting:** To check if the transmitted packet is already in the waiting list but not at front

bool CPQueue::isWaiting(QueueElementType& item)

{

if ((fronti == reari)){

// if queue is empty

return false ;

}

else{

int i =nextPosition(fronti) ;

bool found=0 ;

while (i!=nextPosition(reari)){

if(queuearray[i]==item){

found=1 ;break ;

}

i=nextPosition(i);

}

if(found){return true ;}

else{return false ;}

}}