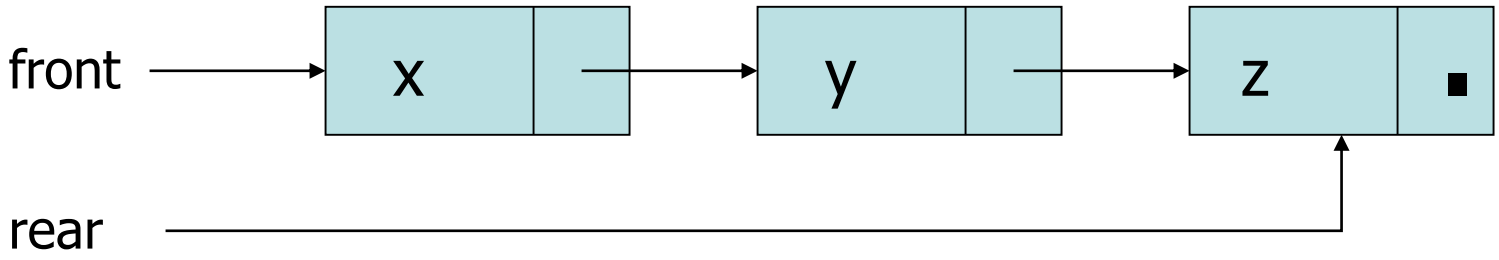
Data Structures

Fall 2023

Pointer-based Implementation of Queue

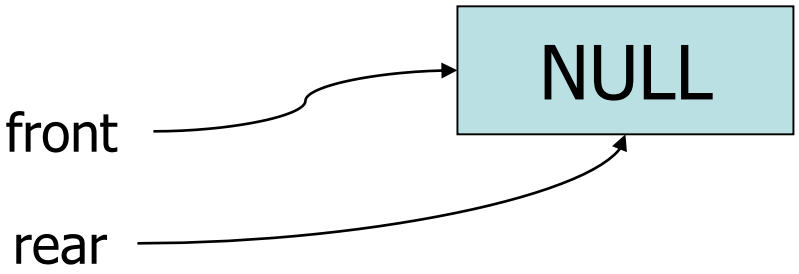
Pointer-Based Implementation of Queues

- Queue Class maintains two pointers
  - `front:` A pointer to the first element of the queue
  - `rear:` A pointer to the last element of the queue

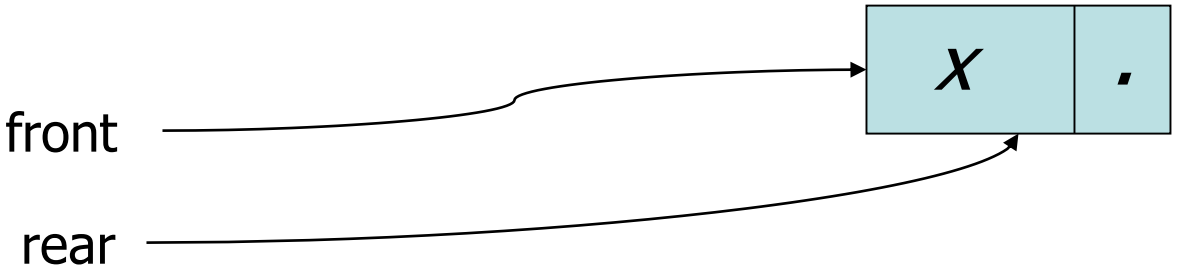


Queue Operations (1)

- `MAKENULL(Q)`

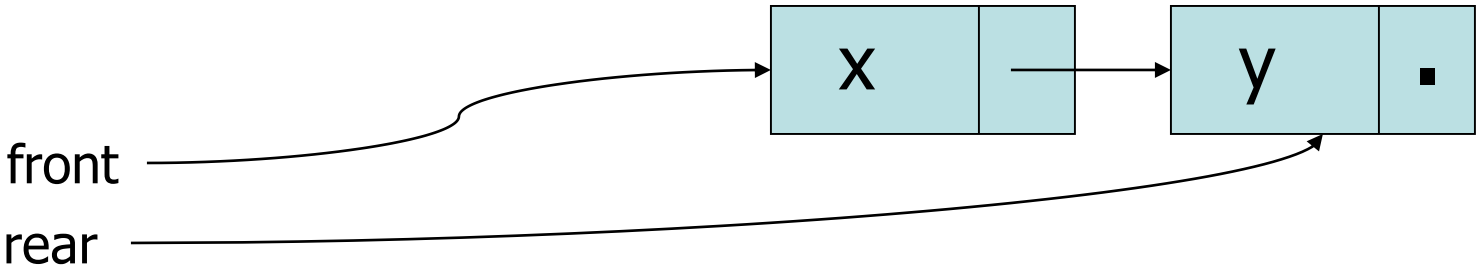


- `ENQUEUE (x, Q)`

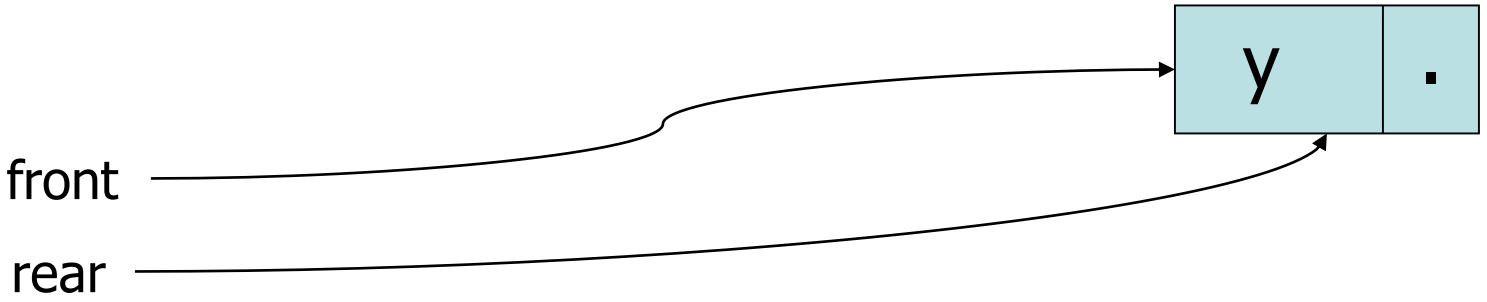


Queue Operations

- `ENQUEUE(y, Q)`



- `DEQUEUE (Q)`



Pointer Implementation – Code (1)

```
class DynIntQueue
{
   private:
      struct QueueNode  // Structure to define linked list node
      {
         int value;
         QueueNode *next;
      };
      QueueNode *front; // pointer to the first node
      QueueNode *rear;  // pointer to the last node
      int numItems;     // Number of nodes in the linked list
   public:
      DynIntQueue(void);
      ~DynIntQueue(void);
      void enqueue(int);
      int dequeue(void);
      bool isEmpty(void);
      void makeNull(void);
};
```

Pointer Implementation – Code (2)

- Constructor

```
DynIntQueue::DynIntQueue(void)
{
   front = NULL;
   rear = NULL;
   numItems = 0;
}
```

- `isEmpty()` returns true if the queue is full and false otherwise

```
bool DynIntQueue::isEmpty(void)
{
   if (numItems)
      return false;
   else
      return true;
}
```

Array Implementation – Code (3)

- Function `enqueue` inserts the value in `num` at the end of Queue

```
void DynIntQueue::enqueue(int num)
{
   QueueNode *newNode;
   newNode = new QueueNode;
   newNode->value = num;
   newNode->next = NULL;
   if (isEmpty()) {
      front = newNode;
      rear = newNode;
   }
   else {
      rear->next = newNode;
      rear = newNode;
   }
   numItems++;
}
```

Array Implementation – Code (4)

- Function dequeue removes and returns the value at the front of the Queue

```
int DynIntQueue::dequeue(void)
{
   QueueNode *temp;
   int num;
   if (isEmpty())
      cout << "The queue is empty.\n";
   else {
      num = front->value;
      temp = front->next;
      delete front;
      front = temp;
      numItems--;
      if(!numItems) rear = NULL;
   }
   return num;
}
```

Pointer Implementation – Code (5)

- Destructor

```
DynIntQueue::~DynIntQueue(void)
{
   makeNull();
}
```

- `makeNull()` resets `front` & `rear` indices to `NULL` and sets `numItems` to `0`

```
void DynIntQueue::makeNull(void)
{
   while(!isEmpty()){
      dequeue();
   }
}
```

Using Queues

```
void main(void)
{
   DynIntQueue iQueue;

   cout << "Enqueuing 5 items...\n";
   // Enqueue 5 items
   for (int x = 0; x < 5; x++)
      iQueue.enqueue(x);

   // Deqeue and retrieve all items in the queue
   cout << "The values in the queue were:\n";
   while (!iQueue.isEmpty())
   {
      int value;
      value= iQueue.dequeue();
      cout << value << endl;
   }
}
```

Output:

```
Enqueuing 5 items...
The values in the queue were:
0
1
2
3
4
```

Data Structures

Fall 2023

---

14. Priority Queue

Definition

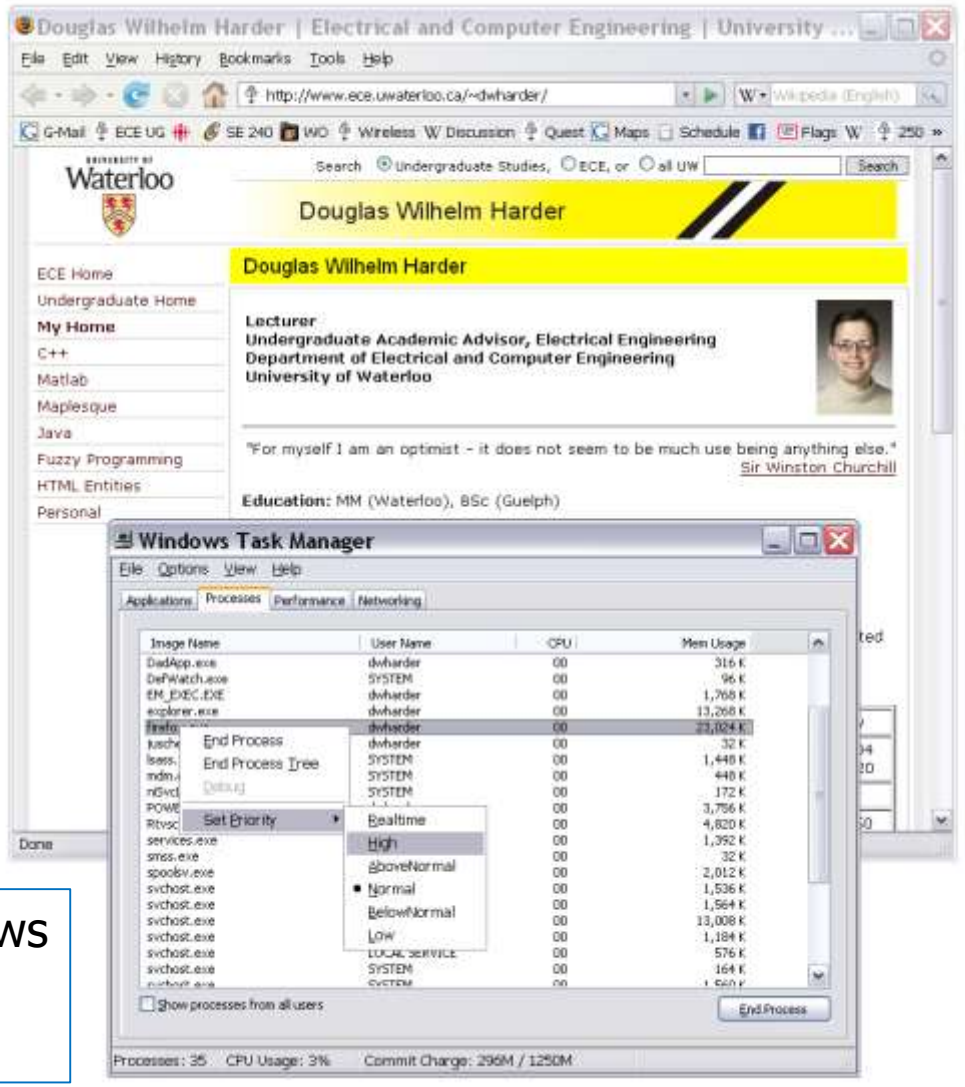
- With queues the order may be summarized by first in, first out
  - Some tasks may be more important or timely than others
  - Higher priority

- Priority queues
  - Enqueue objects using a partial ordering based on priority
  - Dequeue that object which has highest priority

- Performance goal is to make the run time of each operation as close to O(1) as possible

Applications Of Priority Queue

- Hold jobs for a printer in order of length – shortest job first
- Store packets on network routers in order of urgency
- Ordering CPU jobs
- Emergency room admission processing



The priority of processes in Windows may be set in the Windows Task Manager

Priority Queue

- A priority queue is a data structure in which prioritized insertion and deletion operations on elements can be performed according to their priority values.

- There are two types of priority queues:
  - Ascending Priority queue, and a
  - Descending Priority queue

Types of Priority Queue

- **<u>Ascending Priority queue</u>**: a collection of items into which items can be inserted ***randomly*** but only the ***smallest*** item can be removed

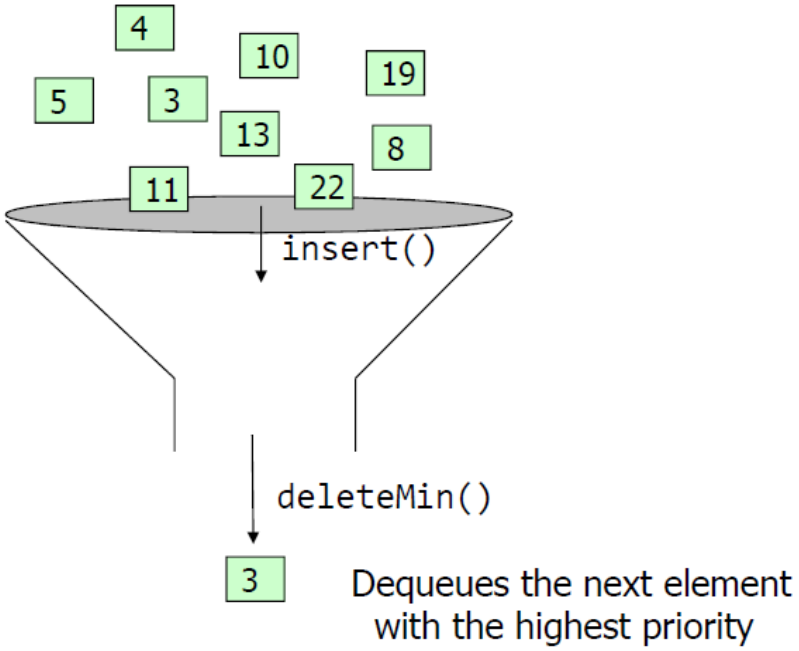
- If "**A-Priority-Q**" is an ascending priority queue then
  - Enqueue() will insert item 'x' into **A-Priority-Q**,
  - minDequeue() will remove the minimum item from **A-Priority-Q** and return its value

Types of Priority Queue

- **Descending Priority queue**: a collection of items into which items can be inserted ***randomly*** but only the ***largest*** item can be removed

- If "**D-Priority-Q**" is a descending priority queue then
  - Enqueue() will insert item x into **D-Priority-Q**,
  - maxDequeue( ) will remove the maximum item from **D-Priority-Q** and return its value

Priority Queue



Priority Queue – ADT

- `insert` (i.e., enqueue)
  - Add to queue
  - Specification of a priority level (0=highest, 1, 2, 3, … , lowest)

- `deleteMin` (i.e., dequeue)
  - Returns the current "highest priority" element in the queue
  - Deletes that element from the queue

- Performance goal is to make the run time of each operation as close to `O(1)` as possible

Priority Queue

- The elements of a priority queue can be numbers, characters or any complex structures such as phone book entries

- But we must have some criteria to determine the priority of its constituent elements. For example,
  - Ascending priority: item with smallest value has maximum "priority"
  - Descending priority: item with highest value has maximum "priority"

- For elements with equal priority, the FIFO technique is applied.

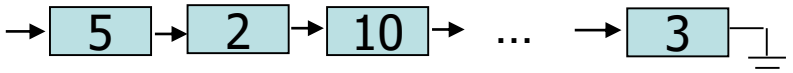
Priority Queue Issues

- In what manner should the items be inserted in a priority queue
  - Ordered (so that retrieval is simple, but insertion will become complex)
  - Arbitrary (insertion is simple but retrieval will require elaborate search mechanism)

- Unordered linked list
  - Insert – O(1) step
  - delete – O(n) steps
- Ordered linked list
  - insert – O(n) steps
  - delete – O(1) step

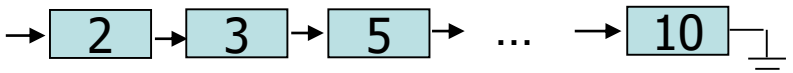
Simple Implementations

- Unordered linked list



  - `Insert` – `O(1)` step
    - Random insertion anywhere in list, e.g. at start
  - `deleteMin` – `O(n)` steps
    - Search list for element with highest priority and return

- Ordered linked list



  - `insert` – `O(n)` steps
    - Insert every element at its correct position in list based on priority
  - `deleteMin` – `O(1)` step
    - Return the first element since list is always sorted