INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Queue - Data Structures

Dr. Animesh Chaturvedi

Assistant Professor: IIIT Dharwad

Young Researcher: Heidelberg Laureate Forum

Postdoc: King's College London & The Alan Turing Institute

PhD: IIT Indore MTech: IIITDM Jabalpur

Queue

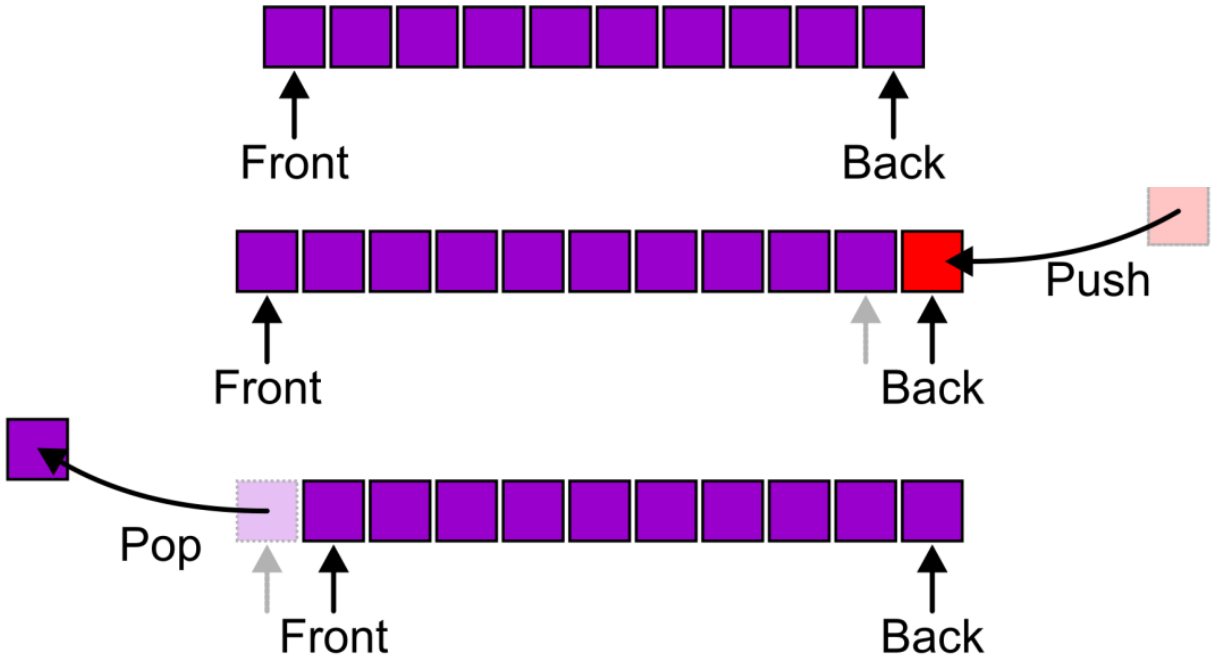
- Queue is an abstract data structure, somewhat similar to Stacks.
- Unlike stacks, a queue is open at both its ends.
- One end is used to insert data (enqueue) and the other is used to remove data (dequeue).
- A queue can be implemented using Arrays, Linked-lists, Pointers and Structures.

Abstract Queue

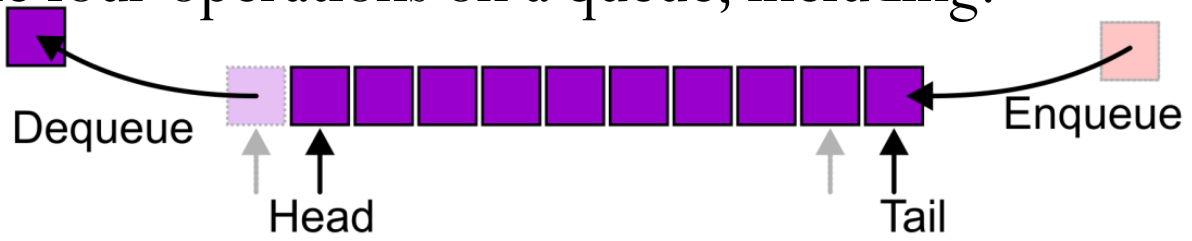
- Queue is an abstract data type that emphasizes operation for sequential linear ordering
  - Insertions and removals are performed individually
  - There are no restrictions on objects inserted into (pushed onto) the queue—that object is designated the back of the queue
  - The object designated as the front of the queue is the object which was in the queue the longest
  - The remove operation (popping from the queue) removes the current front of the queue

Graphical view of Queue Operations

- Also called a first-in–first-out (FIFO) data structure

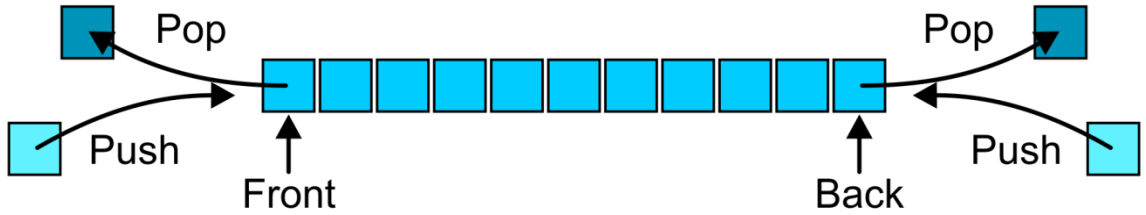


- Alternative terms may be used for the four operations on a queue, including:



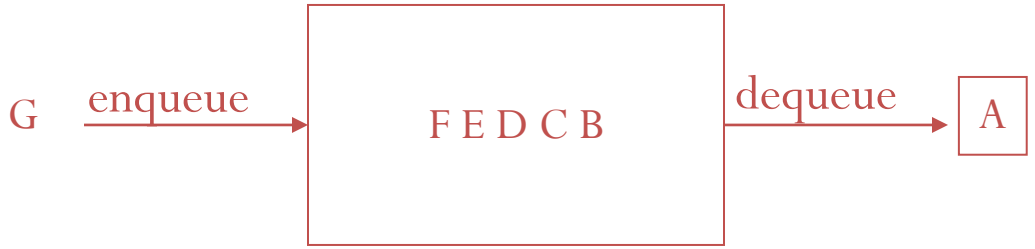
Abstract Deque

- An Abstract Deque (Deque ADT) is an abstract data structure which emphasizes specific operations:
  - Uses a explicit linear ordering
  - Insertions and removals are performed individually
  - Allows insertions at both the front and back of the deque



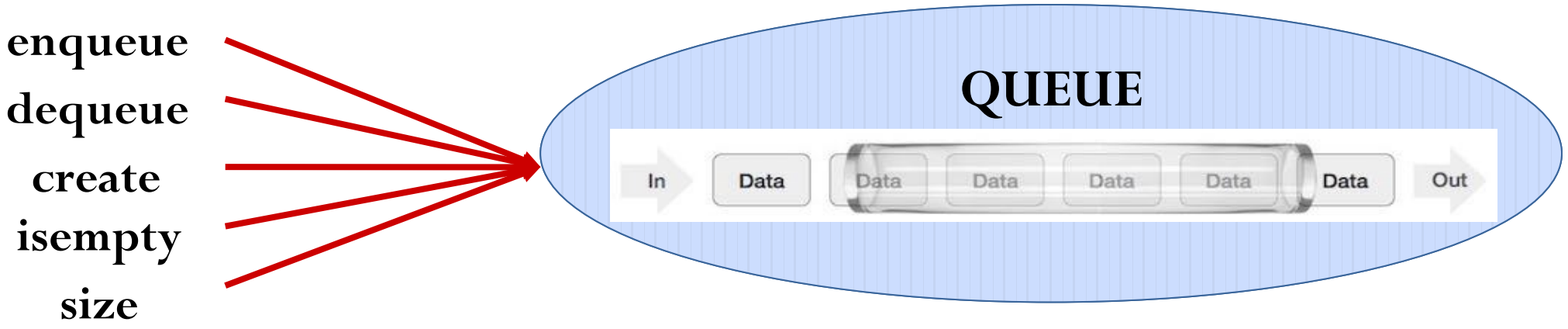
Queue ADT

- Queue operations
  - create
  - destroy
  - enqueue
  - dequeue
  - is_empty



- Queue property: if x is enQed before y is enQed, then x will be deQed before y is deQed

  FIFO: First In First Out



Abstract Deque

- The operations will be called

front	back
push_front	push_back
pop_front	pop_back

- There are four errors associated with this abstract data type:
  - It is an undefined operation to access or pop from an empty deque
- The implementations are clear:
  - We must use either a doubly linked list or a circular array

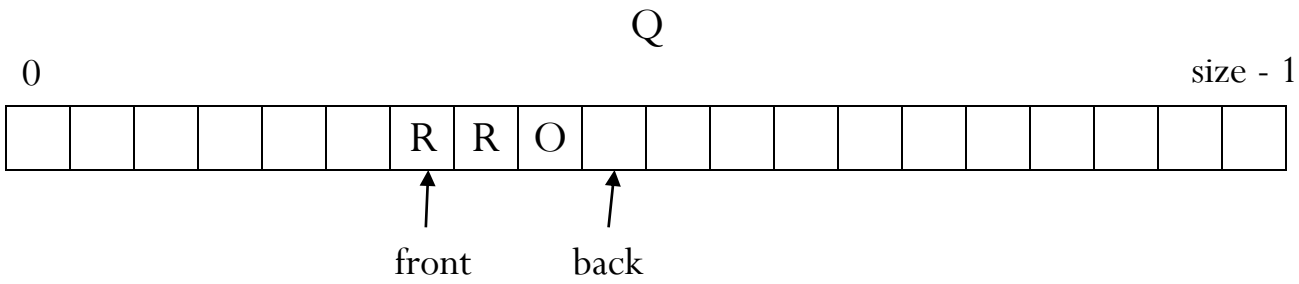
Array Queue Data Structure

enqueue R

enqueue R

enqueue O

dequeue




```
void enqueue(Object x) {
  Queue[back] = x
  back = (back + 1)
}
Object dequeue() {    \\remove R from front
  x = Queue[front]
  front = (front + 1)
  return x
}
```

Pseudocode

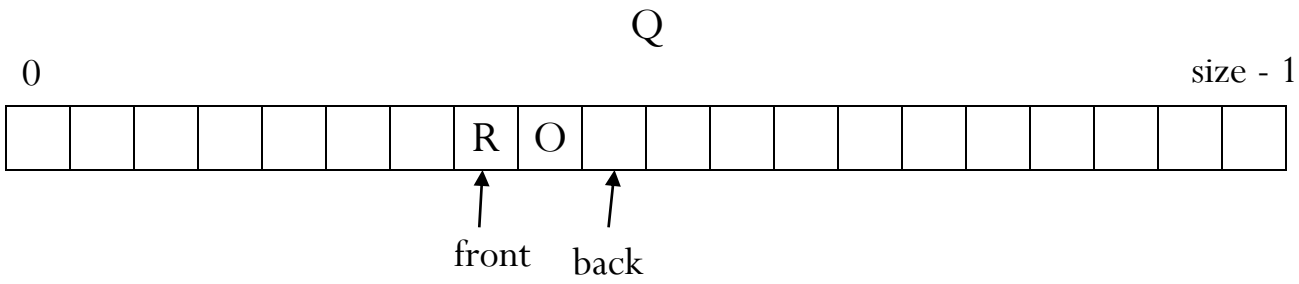
Array Queue Data Structure

enqueue R

enqueue R

enqueue O

dequeue

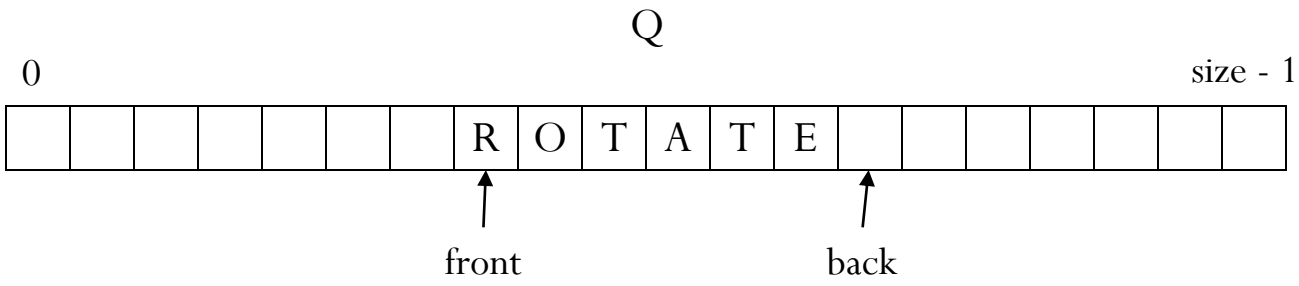



```
void enqueue(Object x) {
  Queue[back] = x
  back = (back + 1)
}
Object dequeue() { \\removed R from front
  x = Queue[front]
  front = (front + 1)
  return x
}
```

Pseudocode

Array Queue Data Structure

enqueue R
enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
enqueue E

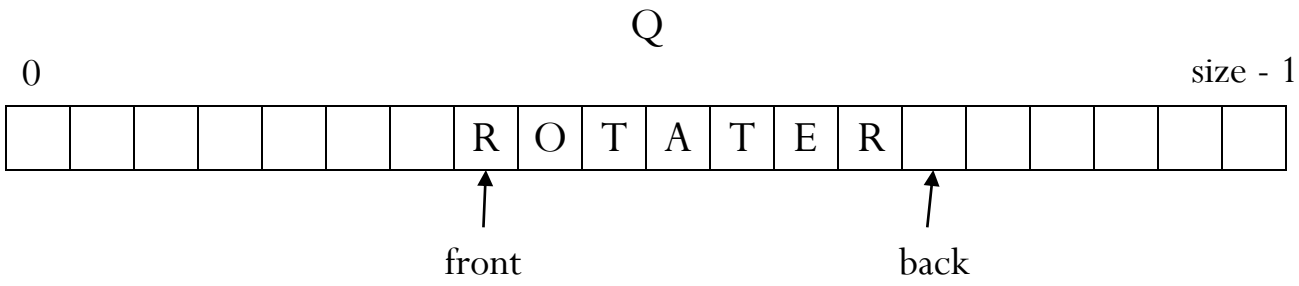



```
void enqueue(Object x) {
  Queue[back] = x
  back = (back + 1)
}
Object dequeue() {
  x = Queue[front]
  front = (front + 1)
  return x
}
```

Pseudocode

Array Queue Data Structure

enqueue R
enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
enqueue E
enqueue R




```
void enqueue(Object x) {      //passed R
  Queue[back] = x             // x = R
  back = (back + 1)
}
Object dequeue() {
  x = Queue[front]
  front = (front + 1)
  return x
}
```

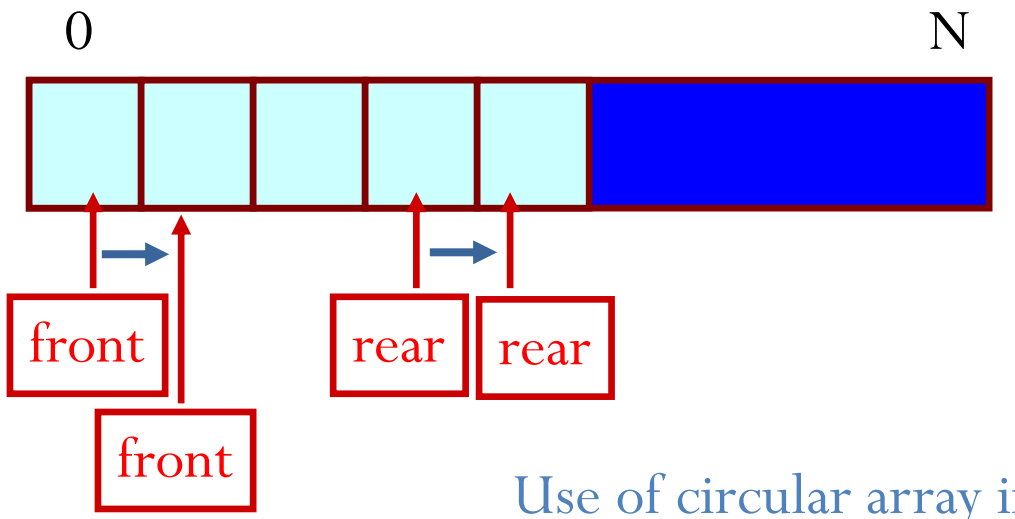
Pseudocode

Problem With Array Implementation

- The size of the queue depends on the number and order of enqueue and dequeue.
- It may be situation where memory is available but enqueue is not possible.

ENQUEUE DEQUEUE

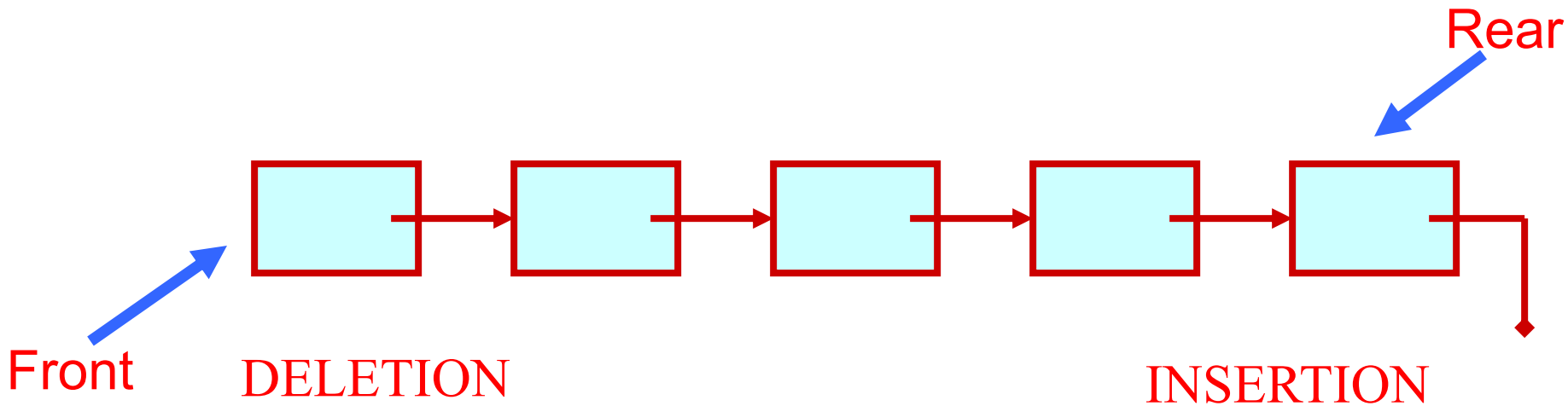
Effective queuing storage area of array gets reduced.



Use of circular array indexing

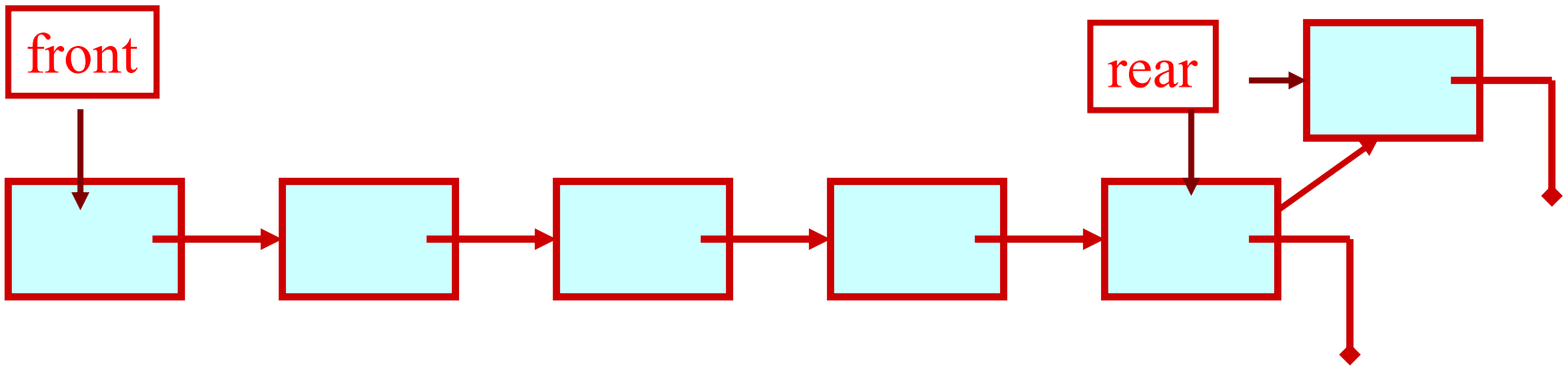
Queue: Linked List Structure

- Create a linked list to which items would be added to one end and deleted from the other end.
- Two pointers will be maintained:
  - One pointing to the beginning of the list (point from where elements will be deleted).
  - Another pointing to the end of the list (point where new elements will be inserted).



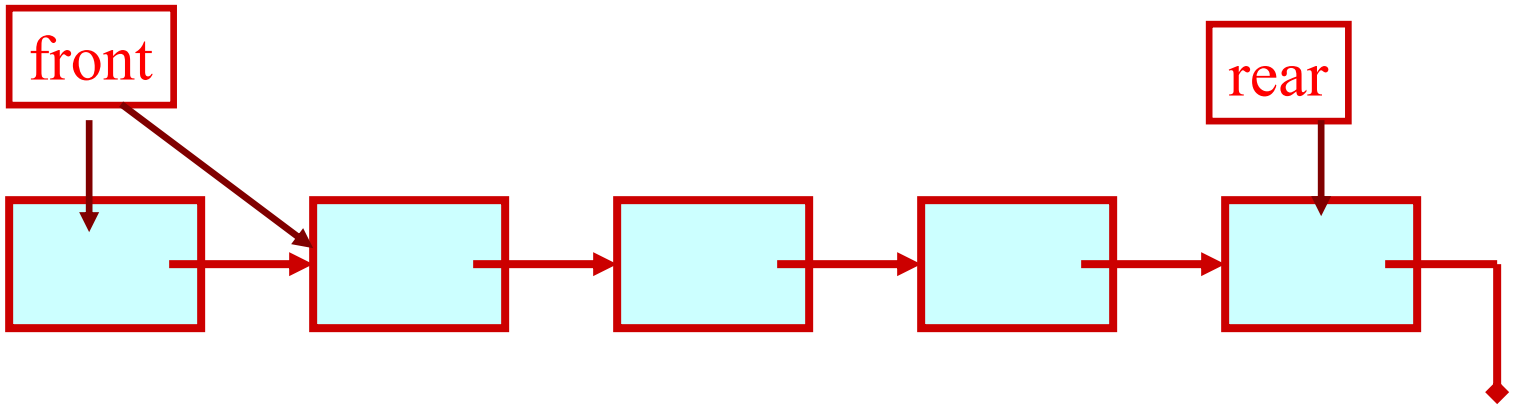
Queue: Linked List Structure

ENQUEUE



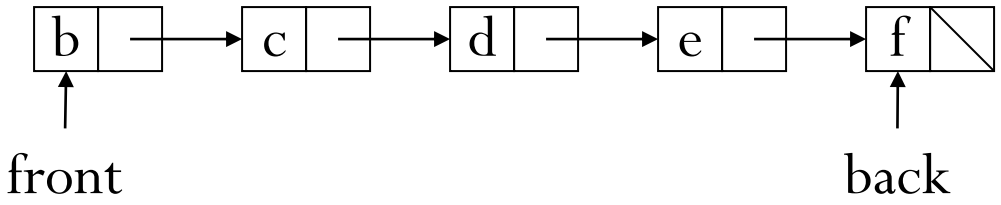
Queue: Linked List Structure

DEQUEUE



Linked List Queue Data Structure

- There are two exceptions associated when is the Queue empty.
- On an empty queue, it is an undefined operation to call
  - either dequeue
  - or enqueue (on front).




```
void enqueue(Object x) {
   if (is_empty())
        front = back = new Node(x)
   else
        back->next = new Node(x)
        back = back->next
}
// enqueue on back
```

```
Object dequeue() {
   assert(!is_empty)
   return_data = front->data
   temp = front
   front = front->next
   delete temp
   return return_data
}
// dequeue on front
```

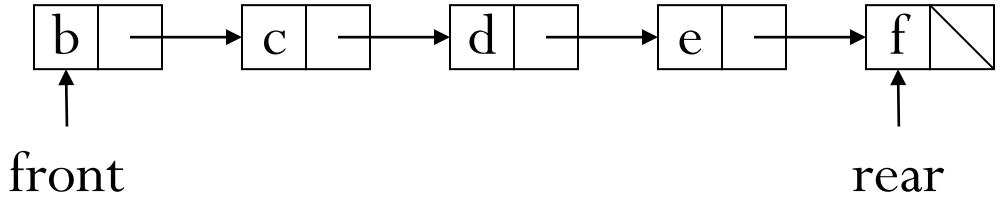
QUEUE: First-In-First-Out (FIFO)

```
queue *create();
                    /* Create a new queue */

void enqueue (queue *q, int element);
                    /* Insert an element in the queue */
int dequeue (queue *q);
                    /* Remove an element from the queue */
int isempty (queue *q);
                    /* Check if queue is empty */
int size (queue *q);
                    /* Return the no. of elements in queue */
```

Assume: queue contains integer elements.

Queue using Linked List




```
struct qnode
{
  int val;
   struct qnode *next;
};

struct queue
{
   struct qnode *qfront, *qrear;
};
typedef struct queue QUEUE;
```

```
void enqueue (QUEUE *q,int element)
{
   struct qnode *q1;
   q1=(struct qnode *)malloc(sizeof(struct qnode));
   q1->val= element;
   q1->next=q->qfront;
   q->qfront=q1;
}
```

// enqueue (on front)

```
int size (queue *q)
{  queue *q1;
   int count=0;
   q1=q;
   while(q1!=NULL)
   { q1=q1->next;
     count++;
   }
return count;
}

int peek (queue *q)
{  queue *q1;
   q1=q;
   while(q1->next!=NULL)
      q1=q1->next;
return (q1->val);
}

int dequeue (queue *q)
{  int val;
   queue *q1,*prev;
   q1=q;
   while(q1->next!=NULL)
   {  prev=q1;
      q1=q1->next;
   }
   val=q1->val;
   prev->next=NULL;
   free(q1);
return (val);
}
```

// dequeue (on rear)

Applications of the Queue

- Direct applications:-
  - Waiting lists
  - Access to shared resources (e.g., printer)
  - Multiprogramming

- Indirect applications:-
  - Auxiliary data structure for algorithms
  - Component of other data structures

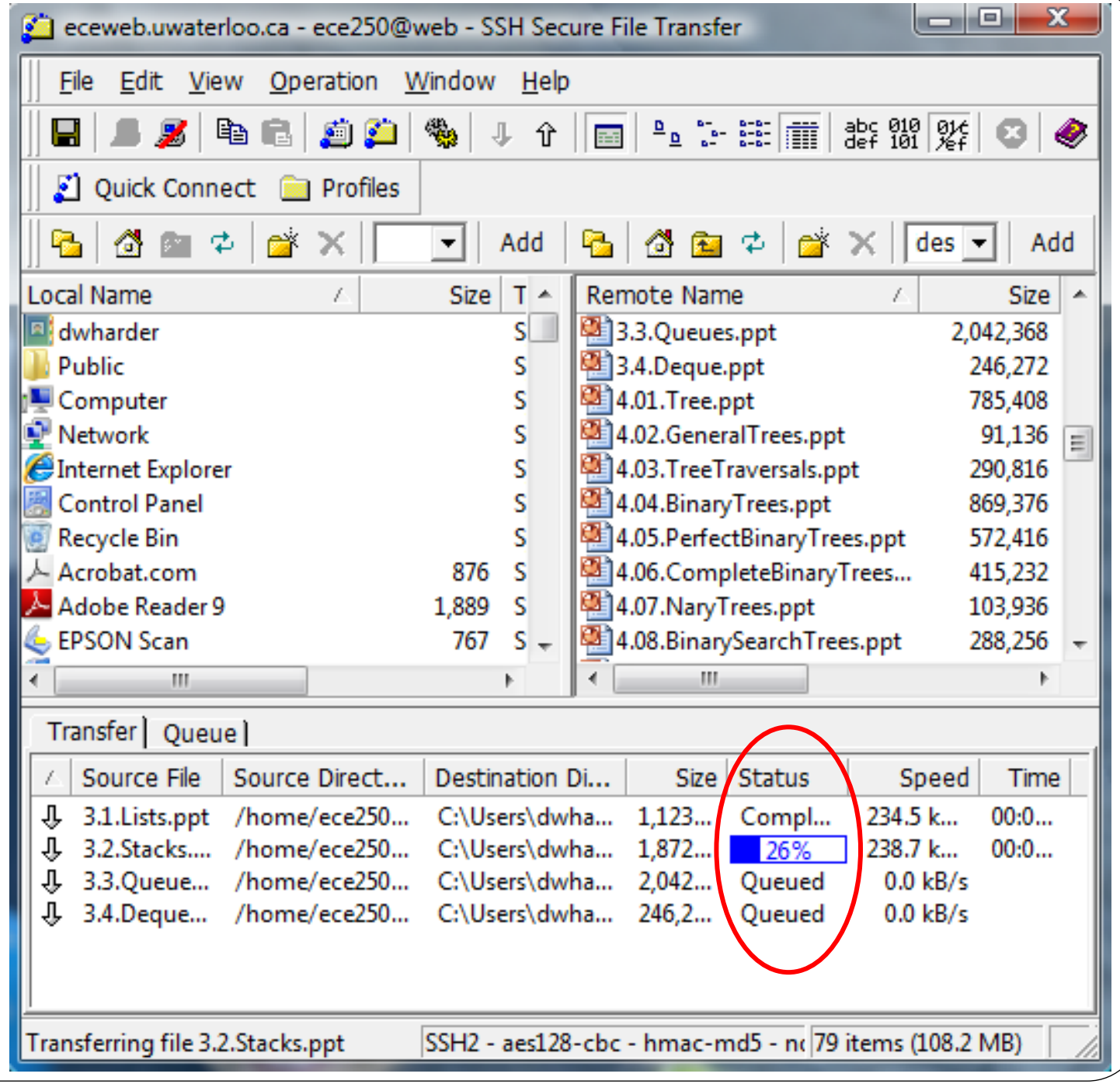
- Hold jobs for a printer
- Store packets on network routers
- Hold memory “freelists”
- Make waitlists fair
- Breadth first search

Applications of the Queue

- The most common application is in client-server models
  - Multiple clients may be requesting services from one or more servers
  - Some clients may have to wait while the servers are busy
  - Those clients are placed in a queue and serviced in the order of arrival
- Grocery stores, banks, and airport security use queues
- The SSH Secure Shell and SFTP are clients
- Most shared computer services are servers:
  - Web, file, ftp, database, mail, printers, WOW, etc.

Applications of the Queue

- For example, in downloading these presentations from the ECE 250 web server, those requests not currently being downloaded are marked as "Queued"
- Useful as a general-purpose tool
- Consider solving a maze by adding or removing a constructed path at the front
- Once the solution is found, iterate from the back for the solution

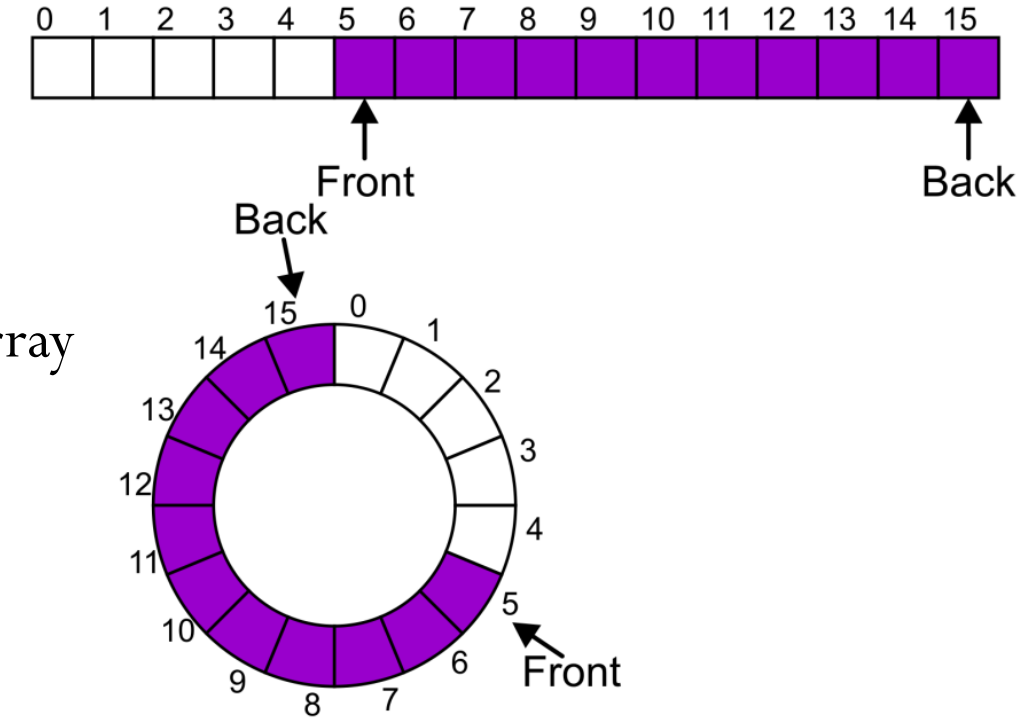


Implementations

- We will look at two implementations of queues:
  - Singly linked lists (whatever we have seen till now)
  - Circular arrays (next)

Member Functions

- Suppose that:
  - The array capacity is 16
  - We have performed 16 pushes
  - We have performed 5 pops
    - The queue size is now 11
  - We perform one further push
- The array is not full and yet
  - we cannot place any more objects in to the array

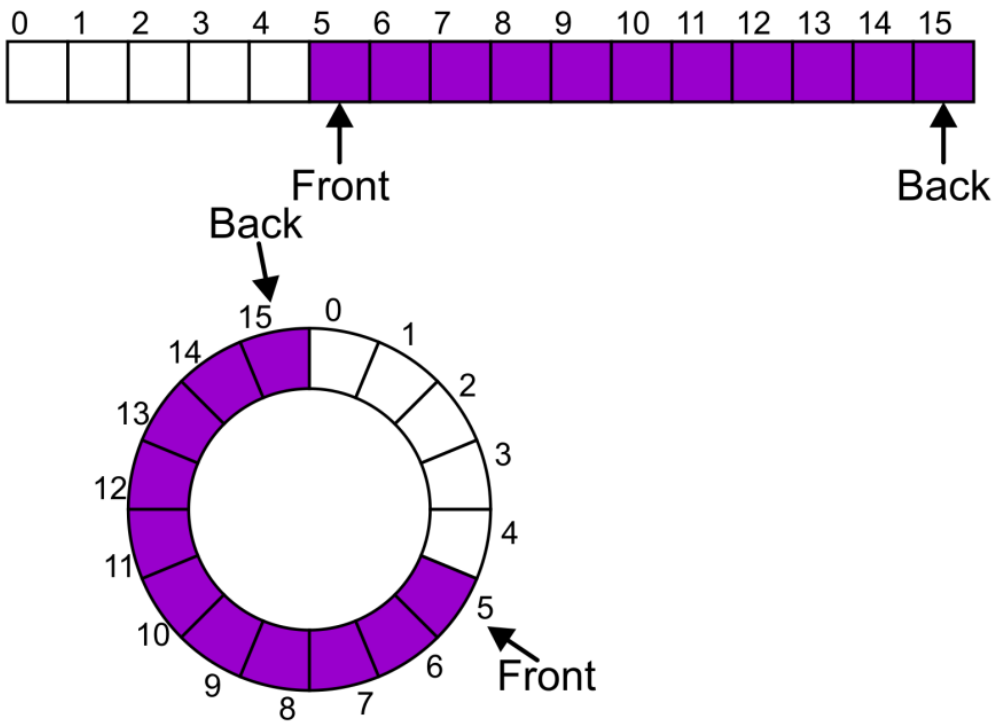


Member Functions

- Instead of viewing the array on the range 0, …, 15, consider the indices being cyclic:

…,14, 15, 0, 1, …, 14, 15, 0, 1, …, 14,15, 0, 1, …

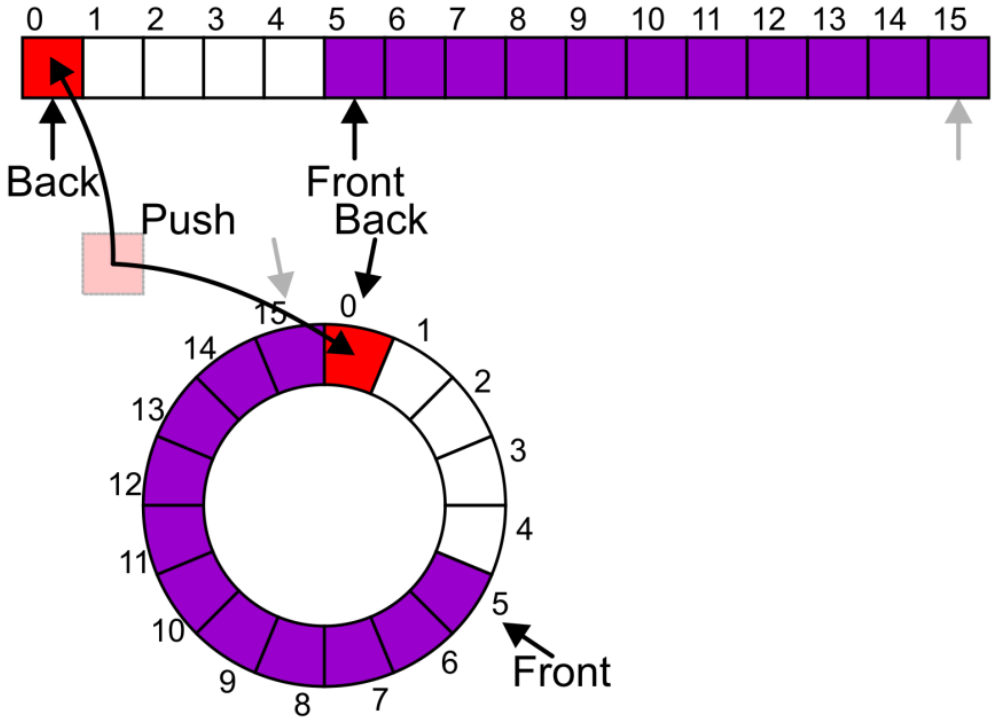
- This is referred to as a *circular array*



Member Functions

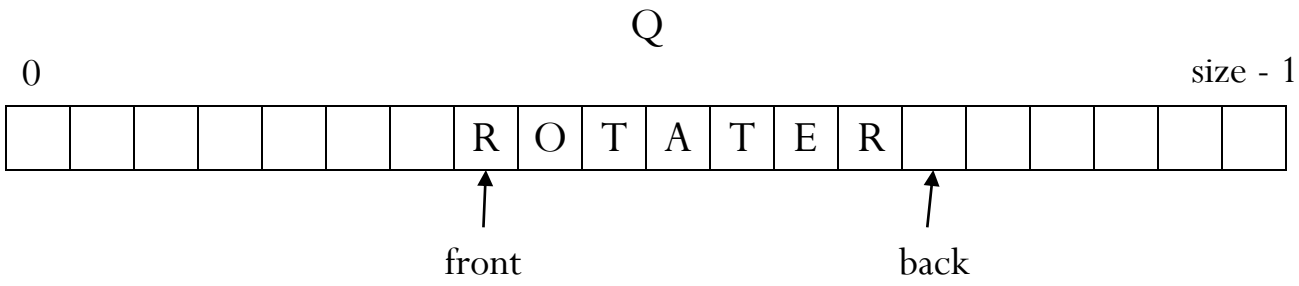
Push may be performed in the next available location of the circular array:

```
++iback;
if ( iback == capacity() ) {
   iback = 0;
}
```



Circular Array Queue Data Structure

enqueue R
enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
enqueue E
enqueue R




```
void enqueue(Object x) {
  Queue[back] = x
  back = (back + 1) % size
}
Object dequeue() {
  x = Queue[front]
  front = (front + 1) % size
  return x
}
```

Pseudocode

Exceptions

- As with a stack, there are a number of options which can be used if the array is filled

- If the array is filled, we have five options:
  - Increase the size of the array
  - Throw an exception
  - Ignore the element being pushed
  - Put the pushing process to "sleep" until something else pops the front of the queue

- Include a member function **bool full()**

T &deque::operator[](int)
T &deque::at(int)

```
#include <iostream>
#include <deque>
using namespace std;

int main() {
    deque<int> ideque;
    ideque.push_front( 5 );   ideque.push_back( 4 );
    ideque.push_front( 3 );   ideque.push_back( 6 );  //   5 3 4 6

    for ( int i = 0; i <= ideque.size(); ++i ) {
        cout << ideque[i] << " " << ideque.at( i ) << "  ";
    }

    cout << endl;
    return 0;
}
```

```
{eceunix:1} ./a.out # output
5 5  3 3  4 4  6 6  0
terminate called after throwing an
instance of 'std::out_of_range'
  what():  deque::_M_range_check
Abort
```

Summary

- The queue is one of the most common abstract data structures
- Understanding how a queue works is trivial
- The implementation is only slightly more difficult than that of a stack
- Applications

References

- Donald E. Knuth, The Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd Ed., Addison Wesley, 1997, §2.2.1, p.238.
- Weiss, Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley, §3.3.1, p.75.
- Cormen, Leiserson, and Rivest, Introduction to Algorithms, McGraw Hill, 1990, §11.1, p.200.
- Koffman and Wolfgang, "Objects, Abstraction, Data Structures and Design using C++", John Wiley & Sons, Inc., Ch. 6.
- Wikipedia, http://en.wikipedia.org/wiki/Double-ended_queue
- CSE326: Data Structure, Department of Computer Science and Engineering, University of Washington https://courses.cs.washington.edu/courses/cse326
- Mike Scott, CS 307 Fundamentals of Computer Science, https://www.cs.utexas.edu/~scottm/cs307/
- Debasis Samanta, Computer Science & Engineering, Indian Institute of Technology Kharagpur, Spring-2017, Programming and Data Structures. https://cse.iitkgp.ac.in/~dsamanta/courses/pds/index.html

ขอบคุณ
Thai

Grazie
Italian

תודה רבה
Hebrew

धन्यवादः
Sanskrit

ಧನ್ಯವಾದಗಳು
Kannada

Ευχαριστώ
Greek

Thank You
English

Gracias
Spanish

Спасибо
Russian

Obrigado
Portuguese

شكراً
Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci
French

多謝
Traditional
Chinese

धन्यवाद
Hindi

Danke
German

多谢
Simplified
Chinese

நன்றி
Tamil
Tamil

ありがとうございました
Japanese

감사합니다
Korean