



List - Doubly Linked List, Stack and Queue

Data Structures and Algorithms

Luu Quang Huan, MsC

*Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM*

Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Overview

① Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

② Stacks

- Implementation of Stacks
 - Linked-list implementation
 - Array implementation
- Applications of Stack

③ Queues

- Implementation of Queue
 - Linked-list implementation
 - Array implementation
- Applications of Queue

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
 - Linked-list implementation
 - Array implementation
- Applications of Stack

Queues

- Implementation of Queue
 - Linked-list implementation
 - Array implementation
- Applications of Queue



Other linked lists

Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Doubly Linked List

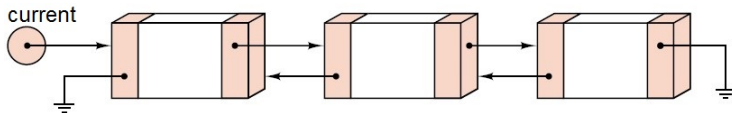


Figure: Doubly Linked List allows going forward and backward.

```
node
  data <dataType>
  next <pointer>
  previous <pointer>
end node
```

```
list
  current <pointer>
end list
```



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Doubly Linked List

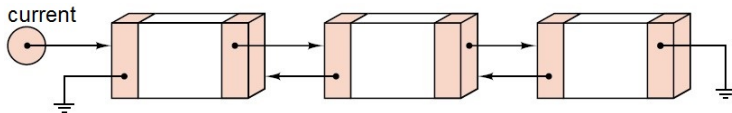


Figure: Doubly Linked List allows going forward and backward.

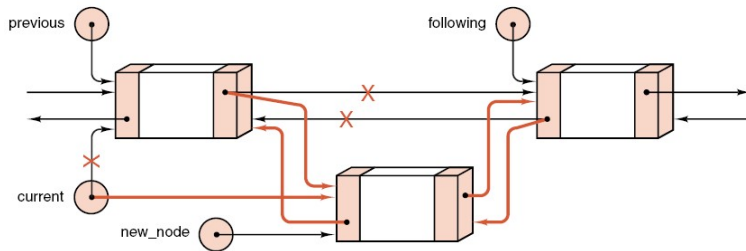


Figure: Insert an element in Doubly Linked List.



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

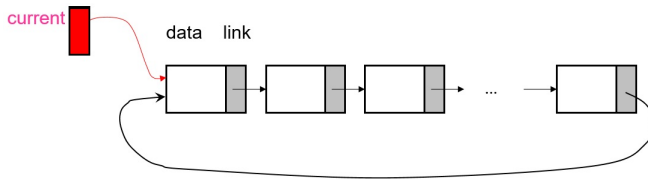
Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Circularly Linked List



```
node
  data <dataType>
  link <pointer>
end node
```

```
list
  current <pointer>
end list
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

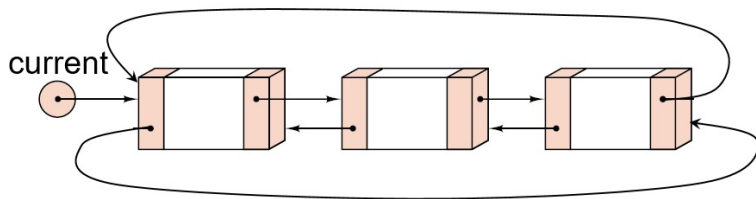
Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Double circularly Linked List



```
node
  data <dataType>
  next <pointer>
  previous <pointer>
end node
```

```
list
  current <pointer>
end list
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Multilinked List

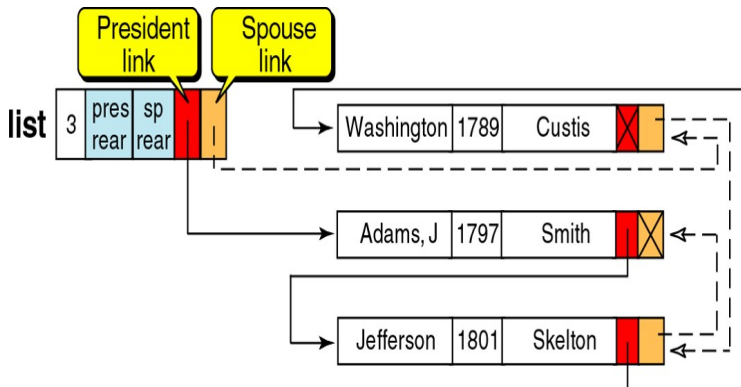


Figure: Multilinked List allows traversing in different order.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Skip List

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

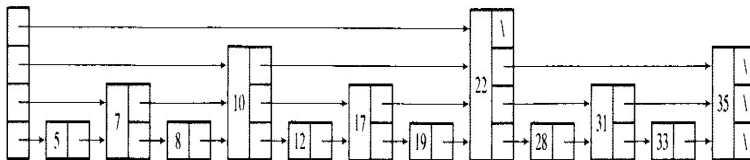


Figure: Skip List improves sequential searching.

Choice of variants of Linked List

To choose among linked Implementations of List, consider:

- Which of the operations will actually be performed on the list and which of these are the most important?
- Is there locality of reference? That is, if one entry is accessed, is it likely that it will next be accessed again?
- Are the entries processed in sequential order? If so, then it may be worthwhile to maintain the last-used position as part of list.
- Is it necessary to move both directions through the list? If so, then doubly linked lists may prove advantageous.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

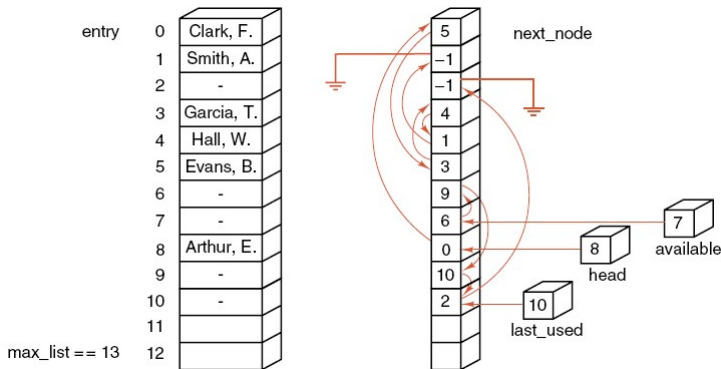
Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Linked List In Array



There are two linked lists in array:

- One (**head**) manages used entries.
- Another (**available**) manages empty entries (have been used or not yet)

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

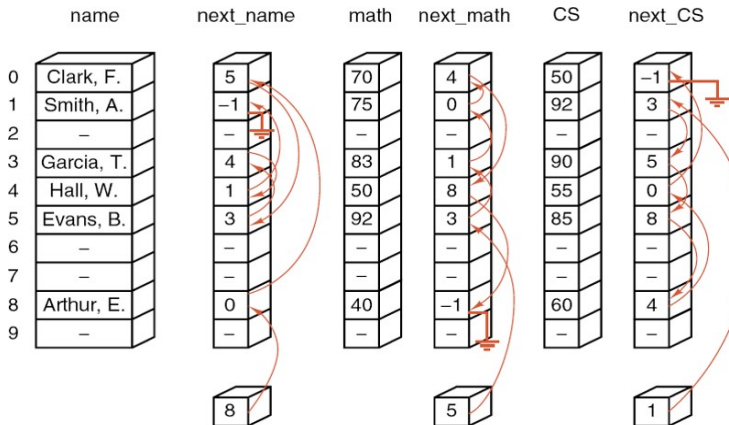
Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Multilinked List In Array



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Sparse Matrice

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

students

courses

	1	2											8,000
1					A				C				
2	A		B								B		
							A						
			C							A			
	B				B		C						
300			A										



student array of linked list

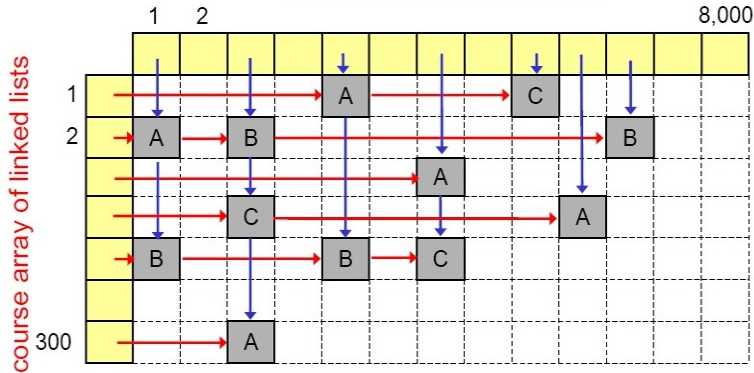


Figure: Two one-dimensional arrays of Linked List are used

Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

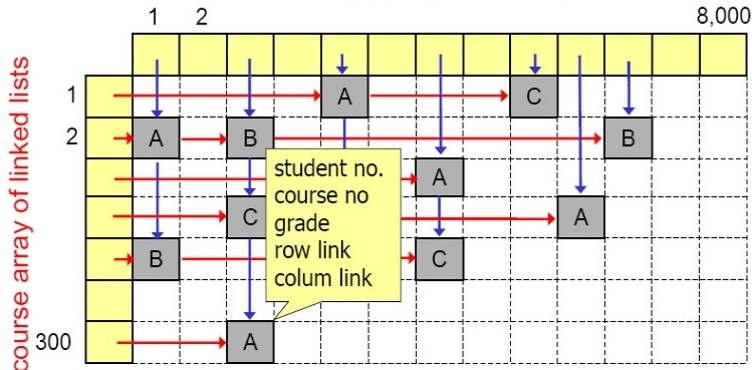
Sparse Matrice

List (P.2)

Luu Quang Huan,
MsC



student array of linked list



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

- Why **two** arrays of linked lists?
- How about **two linked lists** of linked lists?
- How about **3-D** sparse matrices?



Basic operations of Stacks

Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Linear List Concepts

General list:

- No restrictions on which operation can be used on the list.
- No restrictions on where data can be inserted/deleted.

Restricted list:

- Only some operations can be used on the list.
- Data can be inserted/deleted **only at the ends** of the list.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

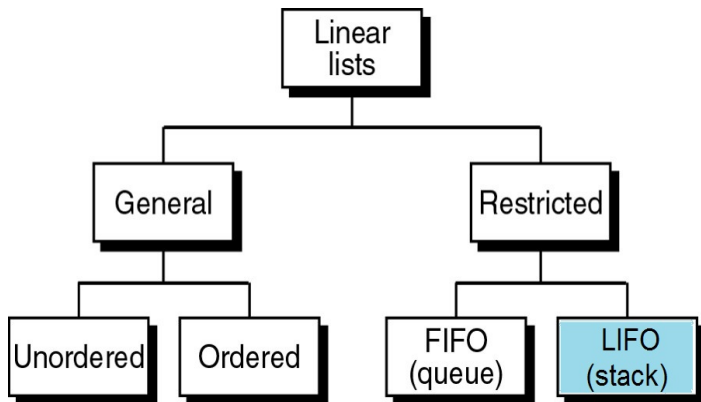
Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Linear list concepts



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Stack

Definition

A **stack** of elements of type T is a finite, ordered sequence of elements of T , in which all insertions and deletions are restricted to one end, called the **top**.

Stack is a Last In - First Out (**LIFO**) data structure.
LIFO: The last item put on the stack is the first item that can be taken off.



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Basic operations:

- Construct a stack, leaving it empty.
- Push an element: put a new element on to the top of the stack.
- Pop an element: remove the top element from the top of the stack.
- Top an element: retrieve the top element.

Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Extended operations:

- Determine whether the stack is empty or not.
- Determine whether the stack is full or not.
- Find the size of the stack.
- Clear the stack to make it empty.

Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Basic operations of Stacks: Push

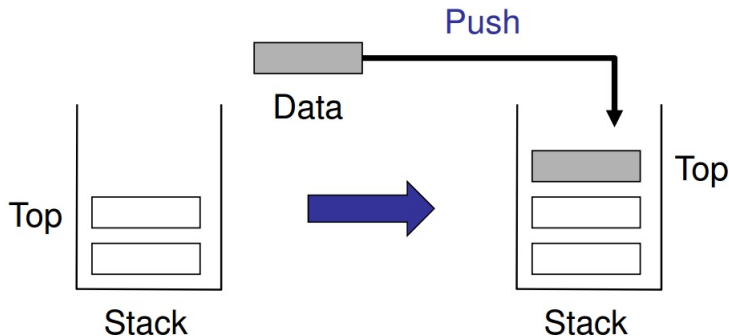


Figure: Successful Push operation

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Stacks: Push

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

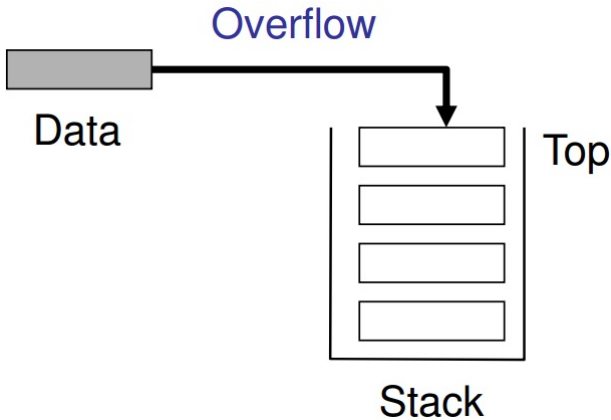


Figure: Unsuccessful Push operation. Stack remains unchanged.

Basic operations of Stacks: Pop

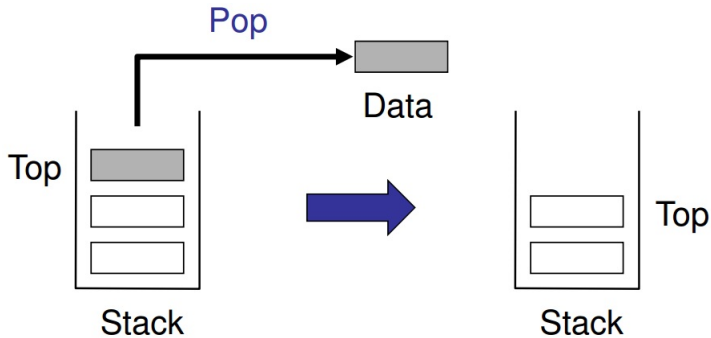


Figure: Successful Pop operation

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Stacks: Pop

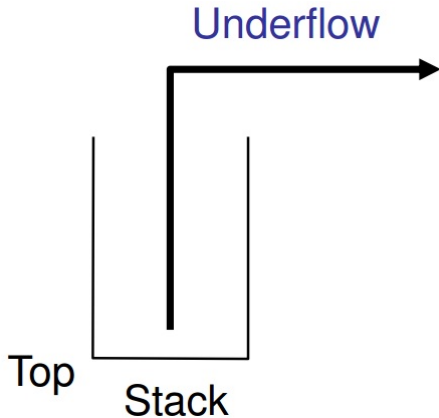


Figure: Unsuccessful Pop operation. Stack remains unchanged.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Stacks: Top

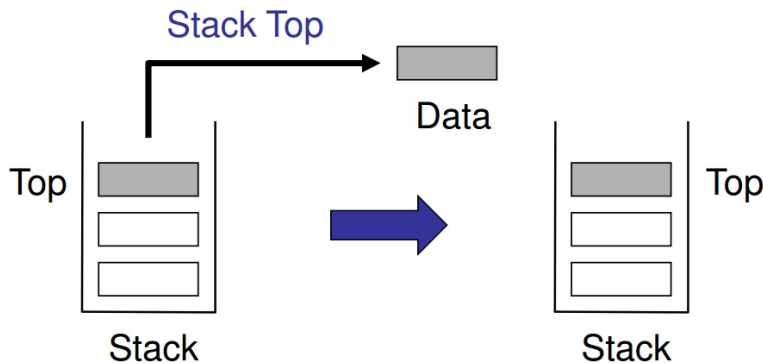


Figure: Successful Top operation. Stack remains unchanged.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Stacks: Top

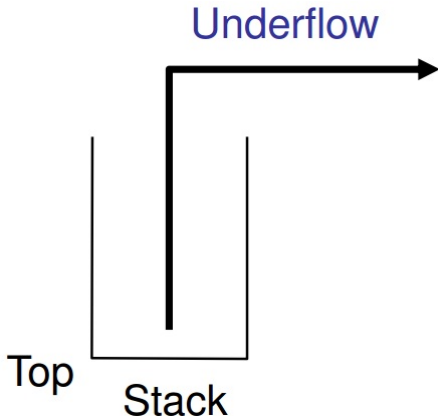


Figure: Unsuccessful Top operation. Stack remains unchanged.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Implementation of Stacks

Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

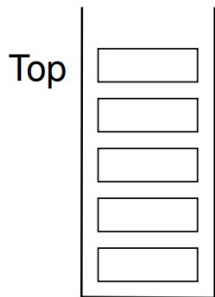
Implementation of Queue

Linked-list implementation

Array implementation

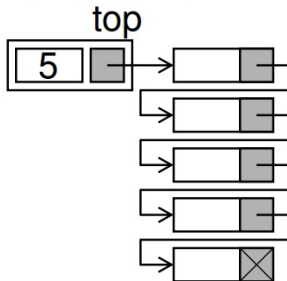
Applications of Queue

Linked-list implementation



Conceptual

Stack structure



Physical

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

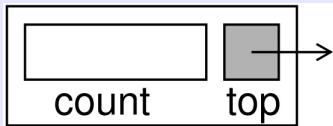
- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

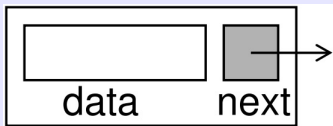
Linked-list implementation

Stack structure



```
stack
  count <integer>
  top <node pointer>
end stack
```

Stack node structure



```
node
  data <dataType>
  next <node pointer>
end node
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Linked-list implementation in C++

```
template <class ItemType>
struct Node {
    ItemType data;
    Node<ItemType> *next;
};
```

```
template <class List_ItemType>
class Stack {
public:
    Stack();
    ~Stack();
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Linked-list implementation in C++

```
void Push(List_ItemType dataIn);  
int Pop(List_ItemType &dataOut);  
int GetStackTop(List_ItemType &dataOut);  
void Clear();  
int IsEmpty();  
int GetSize();  
Stack<List_ItemType>* Clone();  
void Print2Console();
```

```
private:
```

```
Node<List_ItemType>* top;  
int count;
```

```
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation

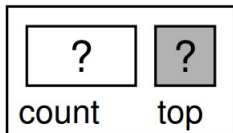
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

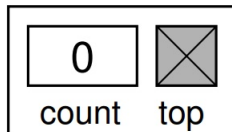
Create an empty Linked Stack

Before



(no stack)

After



(empty stack)

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation**

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Create an empty Linked Stack

```
1 Algorithm createStack(ref stack  
   |metadata|)  
2 Initializes the metadata of a stack  
3 Pre: stack is a metadata structure of a  
   stack  
4 Post: metadata initialized  
  
5 stack.count = 0  
6 stack.top = null  
7 return  
8 End createStack
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation**

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Create an empty Linked Stack

```
template <class List_ItemType>
Stack<List_ItemType >::Stack(){
    this->top = NULL;
    this->count = 0;
}

template <class List_ItemType>
Stack<List_ItemType >::~~Stack(){
    this->Clear();
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation

- Array implementation

- Applications of Stack

Queues

- Implementation of Queue

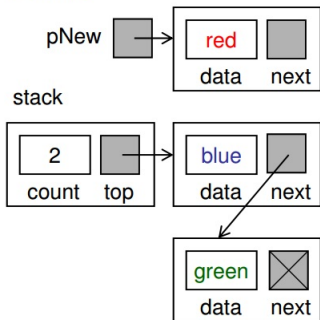
- Linked-list implementation

- Array implementation

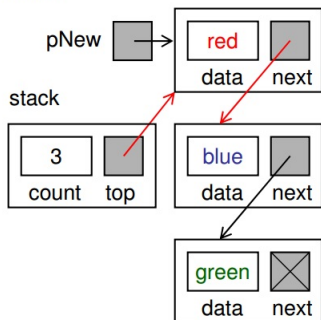
- Applications of Queue

Push data into a Linked Stack

Before



After



- 1 Allocate memory for the new node and set up data.
- 2 Update pointers:
 - Point the new node to the top node (before adding the new node).
 - Point top to the new node.
- 3 Update count

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Push data into a Linked Stack

- 1 **Algorithm** pushStack(ref stack
metadata , *val*datadataType)
- 2 Inserts (pushes) one item into the stack
- 3 **Pre:** stack is a metadata structure to a
valid stack
- 4 data contains value to be pushed into the
stack
- 5 **Post:** data have been pushed in stack
- 6 **Return** true if successful; false if
memory overflow

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Push data into a Linked Stack

```
1 if stack full then  
2   | success = false  
3 else  
4   | allocate (pNew)  
5   | pNew -> data = data  
6   | pNew -> next = stack.top  
7   | stack.top = pNew  
8   | stack.count = stack.count + 1  
9   | success = true  
10 end  
11 return success  
12 End pushStack
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Push data into a Linked Stack

```
template <class List_ItemType>
void Stack<List_ItemType>::Push
    (List_ItemType value){
    Node<List_ItemType>* pNew =
        new Node<List_ItemType>();
    pNew->data = value;
    pNew->next = this->top;
    this->top = pNew;
    this->count++;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Push data into a Linked Stack

- Push is successful when allocation memory for the new node is successful.
- There is **no difference** between push data into **a stack having elements** and push data into **an empty stack** (top having NULL value is assigned to pNew->next: that's corresponding to a list having only one element).

```
pNew->next = top  
top = pNew  
count = count + 1
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

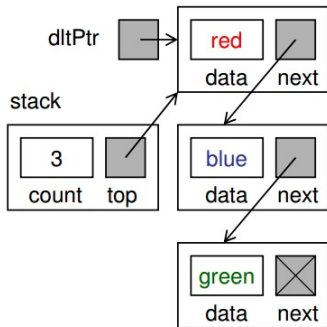
Array implementation
Applications of Stack

Queues

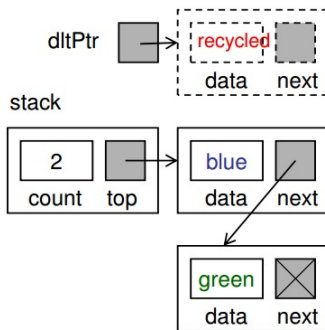
Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Pop Linked Stack

Before



After



- 1 dlPtr holds the element on the top of the stack.
- 2 top points to the next element.
- 3 Recycle dlPtr. Decrease count by 1.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

- 1 **Algorithm** popStack(ref stack
 jmetadata;, ref dataOut jdataType;)
- 2 Pops the item on the top of the stack and
 returns it to caller
- 3 **Pre:** stack is a metadata structure to a
 valid stack
- 4 dataOut is to receive the popped data
- 5 **Post:** data have been returned to caller
- 6 **Return** true if successful; false if stack
 is empty

Pop Linked Stack

```
1 if stack empty then  
2   |   success = false  
3 else  
4   |   dltPtr = stack.top  
5   |   dataOut = stack.top -> data  
6   |   stack.top = stack.top -> next  
7   |   stack.count = stack.count - 1  
8   |   recycle(dltPtr)  
9   |   success = true  
10 end  
11 return success  
12 End popStack
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Pop Linked Stack

```
template <class List_ItemType>
int Stack<List_ItemType >::Pop
    (List_ItemType &dataOut){
    if (this->GetSize() == 0)
        return 0;
    Node<List_ItemType>* dltPtr = this->top;
    dataOut = dltPtr->data;
    this->top = dltPtr->next;
    this->count--;
    delete dltPtr;
    return 1;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Pop Linked Stack

- Pop is successful when the stack is not empty.
- There is **no difference** between pop an element from a **stack having elements** and pop the **only-one element** in the stack (`dltPtr->next` having NULL value is assigned to `top`: that's corresponding to an empty stack).

```
top = dltPtr->next  
recycle dltPtr  
count = count - 1
```



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

- 1 **Algorithm** stackTop(ref stack
 jmetadata;, ref dataOut jdataType;)
- 2 Retrieves the data from the top of the
 stack without changing the stack
- 3 **Pre:** stack is a metadata structure to a
 valid stack
- 4 dataOut is to receive top stack data
- 5 **Post:** data have been returned to caller
- 6 **Return** true if successful; false if stack
 is empty

Stack Top

```
1 if stack empty then  
2   |   success = false  
3 else  
4   |   dataOut = stack.top -j data  
5   |   success = true  
6 end  
7 return success  
8 End stackTop
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue


```
template <class List_ItemType>
int Stack<List_ItemType>::GetStackTop
    (List_ItemType &dataOut){

    if (this->GetSize() == 0)
        return 0;

    dataOut = this->top->data;

    return 1;

}
```



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Destroy Stack

- 1 **Algorithm** `destroyStack(ref stack
 ;metadata;)`
- 2 Releases all nodes back to memory
- 3 **Pre:** `stack` is a metadata structure to a valid stack
- 4 **Post:** stack empty and all nodes recycled



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Destroy Stack

```
1 if stack not empty then  
2   while stack.top not null do  
3     temp = stack.top  
4     stack.top = stack.top -> next  
5     recycle(temp)  
6   end  
7 end  
8 stack.count = 0  
9 return  
0 End destroyStack
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation**

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Destroy Stack

```
template <class List_ItemType>
void Stack<List_ItemType>::Clear() {
    Node<List_ItemType>* temp;
    while (this->top != NULL){
        temp = this->top;
        this->top = this->top->next;
        delete temp;
    }
    this->count = 0;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

isEmpty Linked Stack

```
1 Algorithm isEmpty(ref stack ;metadata;)
2 Determines if the stack is empty
3 Pre: stack is a metadata structure to a
   valid stack
4 Post: return stack status
5 Return true if the stack is empty, false
   otherwise
6 if count = 0 then
7   | Return true
8 else
9   | Return false
10 end
11 End isEmpty
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

isEmpty Linked Stack

```
template <class ListItemType>
int Stack<ListItemType>::IsEmpty() {
    return (count == 0);
}

template <class ListItemType>
int Stack<ListItemType>::GetSize() {
    return count;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

isFull Linked Stack

```
template <class List_ItemType>
int Stack<List_ItemType >::IsFull () {
    Node<List_ItemType>* pNew =
        new Node<List_ItemType>();

    if (pNew != NULL) {
        delete pNew;
        return 0;
    } else {
        return 1;
    }
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Print Stack

```
template <class List_ItemType>
void Stack<List_ItemType>::Print2Console()
{
    Node<List_ItemType>* p;
    p = this->top;
    while (p != NULL){
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Using Stack

```
int main(int argc, char* argv[]){  
    Stack<int> *myStack = new Stack<int>();  
    int val;  
    myStack->Push(7);  
    myStack->Push(9);  
    myStack->Push(10);  
    myStack->Push(8);  
    myStack->Print2Console();  
    myStack->Pop(val);  
    myStack->Print2Console();  
    delete myStack;  
    return 0;  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks

- Linked-list implementation

- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based stack implementation

Implementation of array-based stack is very simple. It uses `top` variable to point to the topmost stack's element in the array.

- ① Initially `top = -1`;
- ② `push` operation increases `top` by one and writes pushed element to `storage[top]`;
- ③ `pop` operation checks that `top` is not equal to -1 and decreases `top` variable by 1;
- ④ `getTop` operation checks that `top` is not equal to -1 and returns `storage[top]`;
- ⑤ `isEmpty` returns boolean if `top == -1`.

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Array-based stack implementation

```
#include <string>
using namespace std;

class ArrayStack {
private:
    int top;
    int capacity;
    int *storage;

public:
    ArrayStack(int capacity) {
        storage = new int[capacity];
        this->capacity = capacity;
        top = -1;
    }
    // ...
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation**
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based stack implementation

```
~ArrayStack() {  
    delete[] storage;  
}  
  
void push(int value) {  
    if (top == capacity - 1)  
        throw string("Stack_is_overflow");  
    top++;  
    storage[top] = value;  
}  
  
void pop(int &dataOut) {  
    if (top == -1)  
        throw string("Stack_is_empty");  
    dataOut = storage[top];  
    top--;  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation**
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based stack implementation

```
int getTop() {  
    if (top == -1)  
        throw string("Stack is empty");  
    return storage[top];  
}  
  
bool isEmpty() {  
    return (top == -1);  
}  
  
bool isFull() {  
    return (top == capacity - 1);  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation**
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based stack implementation

```
int getSize() {  
    return top + 1;  
}  
  
void print2Console() {  
    if (top > -1) {  
        for (int i = top; i >= 0; i--) {  
            cout << storage[i] << " ";  
        }  
        cout << endl;  
    }  
}  
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation**
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Using array-based stack

```
int main(int argc, char* argv[]){
    ArrayStack *myStack = new ArrayStack(10);
    int val;
    myStack->push(7);
    myStack->push(9);
    myStack->push(10);
    myStack->push(8);
    myStack->print2Console();
    myStack->pop(val);
    myStack->print2Console();
    delete myStack;
    return 0;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation**
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation

Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Applications of Stack

Applications of Stack

- Reversing data items
 - Reverse a list
 - Convert Decimal to Binary
- Parsing
 - Brackets Parse
- Postponement of processing data items
 - Infix to Postfix Transformation
 - Evaluate a Postfix Expression
- Backtracking
 - Goal Seeking Problem
 - Knight's Tour
 - Exiting a Maze
 - Eight Queens Problem

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation

Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Basic operations of Queues

Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Queue

Definition

A **queue** of elements of type T is a finite sequence of elements of T , in which data can only be inserted at one end called the **rear**, and deleted from the other end called the **front**.

Queue is a First In - First Out (**FIFO**) data structure.
FIFO: The first item stored in the queue is the first item that can be taken out.



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

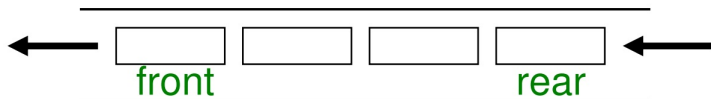
Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Queues

Basic operations:

- Construct a queue, leaving it empty.
- Enqueue: put a new element in to the rear of the queue.
- Dequeue: remove the first element from the front of the queue.
- Queue Front: retrieve the front element.
- Queue Rear: retrieve the rear element.



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Basic operations of Queues: Enqueue

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

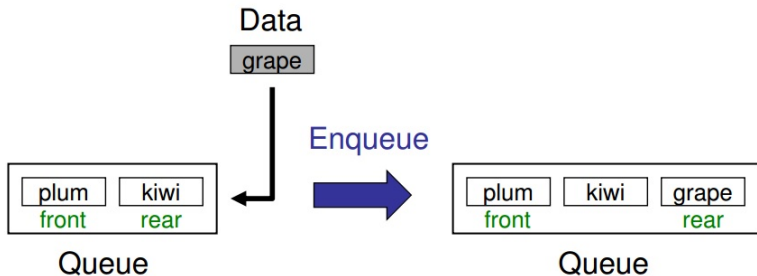
- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

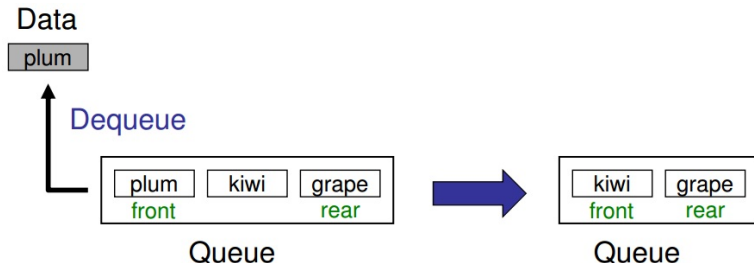
- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Basic operations of Queues: Dequeue



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

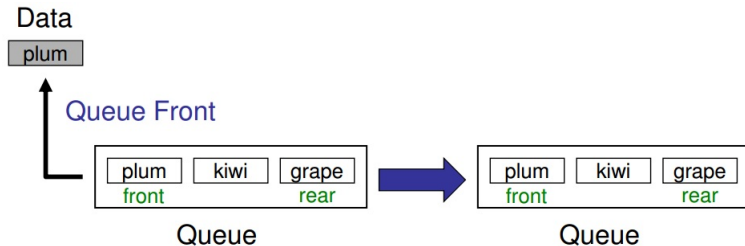
Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Queues: Queue Front



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

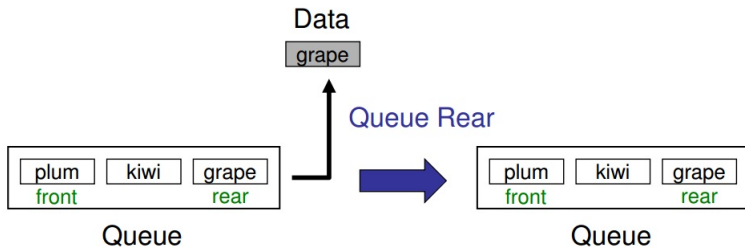
Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Basic operations of Queues: Queue Rear



List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Implementation of Queue

Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

Implementation of Queue

- Linked-list implementation
- Array implementation
- Applications of Queue

Linked-list implementation

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

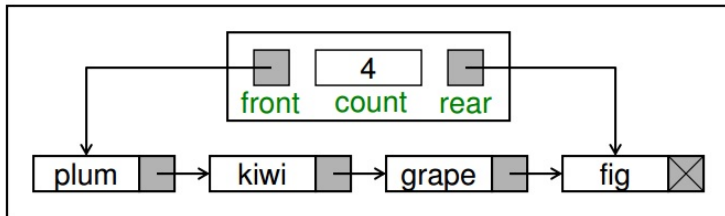
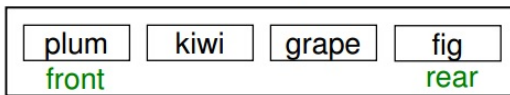
Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Conceptual



Physical

Linked-list implementation

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

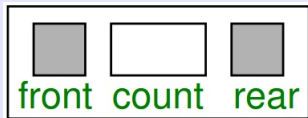
Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Queue structure



```
queue
  count <integer>
  front <node pointer>
  rear <node pointer>
endqueue
```

Queue node structure



```
node
  data <dataType>
  next <node pointer>
end node
```

Linked-list implementation in C++

```
template <class ItemType>
struct Node {
    ItemType data;
    Node<ItemType> *next;
};

template <class List_ItemType>
class Queue {
public:
    Queue();
    ~Queue();
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue

Linked-list implementation in C++

```
void Enqueue(List_ItemType dataIn);
int Dequeue(List_ItemType &dataOut);
int GetQueueFront(List_ItemType &dataOut);
int GetQueueRear(List_ItemType &dataOut);
void Clear();
int IsEmpty();
int GetSize();
void Print2Console();

private:
    Node<List_ItemType> *front, *rear;
    int count;
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Create Queue

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

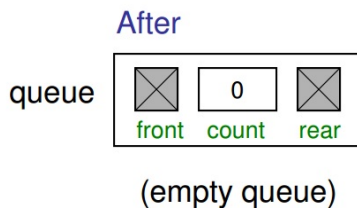
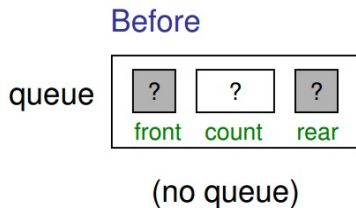
Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue



Create Queue

```
1 Algorithm createQueue(ref queue  
   imetadata;)  
2 Initializes the metadata of a queue  
3 Pre: queue is a metadata structure of a  
   queue  
4 Post: metadata initialized  
  
5 queue.count= 0  
6 queue.front = null  
7 queue.rear = null  
8 return  
9 End createQueue
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue

Create Queue

```
template <class List_ItemType>
Queue<List_ItemType >::Queue() {
    this->count = 0;
    this->front = NULL;
    this->rear = NULL;
}
```

```
template <class List_ItemType>
Queue<List_ItemType >::~~Queue() {
    this->Clear();
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Enqueue: Insert into an empty queue

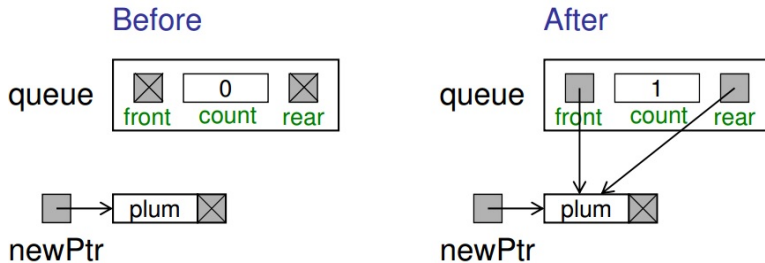


Figure: Insert into an empty queue



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue

Enqueue: Insert into a queue with data

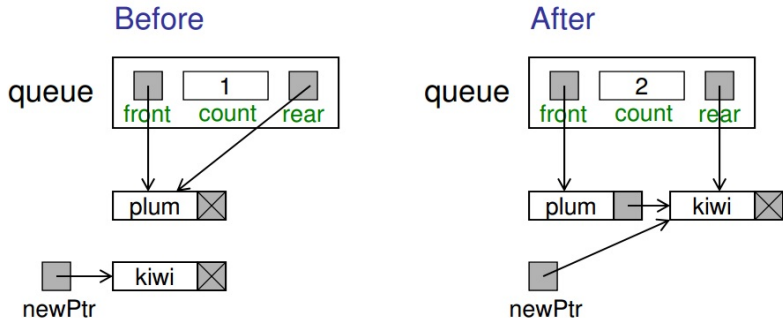


Figure: Insert into a queue with data





Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

- 1 **Algorithm** enqueue(ref queue
 jmetadata¿, val data jdataType¿)
- 2 Inserts one item at the rear of the queue
- 3 **Pre:** queue is a metadata structure of a
 valid queue
- 4 data contains data to be inserted into
 queue
- 5 **Post:** data have been inserted in queue
- 6 **Return** true if successful, false if
 memory overflow

Enqueue

```
1  if queue full then
2      | return false
3  end
4  allocate (newPtr)
5  newPtr->data = data
6  newPtr->next = null
7  if queue.count = 0 then
8      | // Insert into an empty queue
9      | queue.front = newPtr
10 else
11     | // Insert into a queue with data
12     | queue.rear->next = newPtr
13 end
14 queue.rear = newPtr
15 queue.count = queue.count + 1
16 return true
17 End enqueue
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Enqueue

```
template <class List_ItemType>
void Queue<List_ItemType>::Enqueue
    (List_ItemType value){
    Node<List_ItemType>* newPtr = new
        Node<List_ItemType>();
    newPtr->data = value;
    newPtr->next = NULL;
    if (this->count == 0)
        this->front = newPtr;
    else
        this->rear->next = newPtr;
    this->rear = newPtr;
    this->count++;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Deque: Delete data in a queue with only one item

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

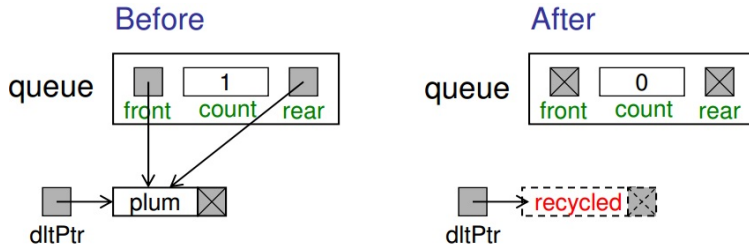


Figure: Delete data in a queue with only one item

Deque: Delete data in a queue with more than one item

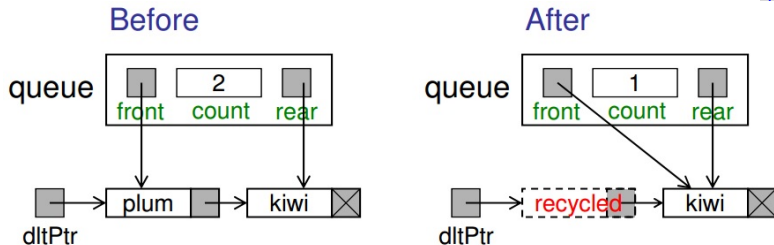


Figure: Delete data in a queue with more than one item

Deque

- 1 **Algorithm** dequeue(ref queue
 jmetadataj, ref dataOut jdataTypej)
- 2 Deletes one item at the front of the queue
 and returns its data to caller
- 3 **Pre:** queue is a metadata structure of a
 valid queue
- 4 dataOut is to receive dequeued data
- 5 **Post:** front data have been returned to
 caller
- 6 **Return** true if successful, false if
 memory overflow



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Deque

```
1  if queue.empty then
2      return false
3  end
4  dataOut = queue.front -> data
5  dltPtr = queue.front
6  if queue.count == 1 then
7      // Delete data in a queue with only one item
8      queue.rear = NULL
9  end
10 queue.front = queue.front->next
11 queue.count = queue.count - 1
12 recycle(dltPtr)
13 return true
14 End dequeue
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Dequeue

```
template <class List_ItemType>
int Queue<List_ItemType >::Dequeue(
    List_ItemType &dataOut){
    if (count == 0)
        return 0;
    dataOut = front->data;
    Node<List_ItemType>* dltPtr= this->front;
    if (count == 1)
        this->rear = NULL;
    this->front = this->front->next;
    this->count--;
    delete dltPtr;
    return 1;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue

Queue Front

```
template <class List_ItemType>
int Queue<List_ItemType >::GetQueueFront
    (List_ItemType &dataOut){
    if (count == 0)
        return 0;
    dataOut = this->front->data;
    return 1;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue



```
template <class List_ItemType>
int Queue<List_ItemType >::GetQueueRear
    (List_ItemType &dataOut){
    if (count == 0)
        return 0;
    dataOut = this->rear->data;
    return 1;
}
```

Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Destroy Queue

- 1 **Algorithm** destroyQueue(ref queue
 jmetadata;)
- 2 Deletes all data from a queue
- 3 **Pre:** queue is a metadata structure of a
 valid queue
- 4 **Post:** queue empty and all nodes recycled
- 5 **Return** nothing

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation**
- Array implementation
- Applications of Queue

Destroy Queue

```
1 if queue not empty then  
2   while queue.front not null do  
3     temp = queue.front  
4     queue.front = queue.front->next  
5     recycle(temp)  
6   end  
7 end  
8 queue.front = NULL  
9 queue.rear = NULL  
0 queue.count = 0  
1 return  
2 End destroyQueue
```



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Destroy Queue

```
template <class List_ItemType>
void Queue<List_ItemType>::Clear() {
    Node<List_ItemType>* temp;
    while (this->front != NULL){
        temp = this->front;
        this->front = this->front->next;
        delete temp;
    }
    this->front = NULL;
    this->rear = NULL;
    this->count = 0;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Queue Empty

```
template <class List_ItemType>
int Queue<List_ItemType>::IsEmpty() {
    return (this->count == 0);
}

template <class List_ItemType>
int Queue<List_ItemType>::GetSize() {
    return this->count;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Print Queue

```
template <class List_ItemType>
void Queue<List_ItemType >::Print2Console () {
    Node<List_ItemType>* p;
    p = this->front;
    cout << "Front: ";
    while (p != NULL) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Using Queue

```
int main(int argc, char* argv[]){
    Queue<int> *myQueue = new Queue<int>();
    int val;
    myQueue->Enqueue(7);
    myQueue->Enqueue(9);
    myQueue->Enqueue(10);
    myQueue->Enqueue(8);
    myQueue->Print2Console();
    myQueue->Dequeue(val);
    myQueue->Print2Console();
    delete myQueue;
    return 1;
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based queue implementation

```
#include <string>
using namespace std;
class ArrayQueue {
private:
    int capacity;
    int front;
    int rear;
    int *storage;

public:
    ArrayQueue(int capacity) {
        storage = new int[capacity];
        this->capacity = capacity;
        front = -1;
        rear = -1;
    }
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation
Applications of Queue

Array-based queue implementation

```
~ArrayQueue() {  
    delete [] storage;  
}  
  
void enqueue(int value) {  
    if (isFull()) throw string("Queue is full");  
    if (front == -1) front = 0;  
    rear++;  
    storage[rear % capacity] = value;  
}  
  
void dequeue(int &valueOut) {  
    if (isEmpty())  
        throw string("Queue is empty");  
    valueOut = storage[front % capacity];  
    front++;  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based queue implementation

```
int getFront() {  
    if (isEmpty())  
        throw string("Queue_is_empty");  
    return storage[front % capacity];  
}  
  
int getRear() {  
    if (isEmpty())  
        throw string("Queue_is_empty");  
    return storage[rear % capacity];  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Array-based queue implementation

```
bool isEmpty() {  
    return (front > rear || front == -1);  
}  
  
bool isFull() {  
    return (rear - front + 1 ==  
capacity);  
}  
  
int getSize() {  
    return rear - front + 1;  
}  
};
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

Using Array-based queue

```
int main(int argc , char* argv[]){  
    ArrayQueue *myQueue = new ArrayQueue(10);  
    int val;  
    myQueue->enqueue(7);  
    myQueue->enqueue(9);  
    myQueue->enqueue(10);  
    myQueue->enqueue(8);  
    myQueue->dequeue(val);  
    delete myQueue;  
    return 1;  
}
```

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue



Other linked lists

Doubly Linked List

Circularly Linked List

Multilinked List

Stacks

Implementation of Stacks

Linked-list implementation

Array implementation

Applications of Stack

Queues

Implementation of Queue

Linked-list implementation

Array implementation

Applications of Queue

Applications of Queue



Other linked lists

Doubly Linked List
Circularly Linked List
Multilinked List

Stacks

Implementation of Stacks
Linked-list implementation
Array implementation
Applications of Stack

Queues

Implementation of Queue
Linked-list implementation
Array implementation

Applications of Queue

- Polynomial Arithmetic
- Categorizing Data
- Evaluate a Prefix Expression
- Radix Sort
- Queue Simulation

[Read more on](#)

Data Structures and Algorithm Analysis

Edition 3.2 (C++ Version)

Part II, Chapter 4: Lists, Stacks, and Queues

List (P.2)

Luu Quang Huan,
MsC



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue

THANK YOU.



Other linked lists

- Doubly Linked List
- Circularly Linked List
- Multilinked List

Stacks

- Implementation of Stacks
- Linked-list implementation
- Array implementation
- Applications of Stack

Queues

- Implementation of Queue
- Linked-list implementation
- Array implementation
- Applications of Queue