CHAPTER 11: ABSTRACT DATA TYPES

11.1 STACKS

11.1.1 Characteristics

- **LIFO:** Last In, First Out
- Add and remove from only one end (top)

11.1.2 Operations

Push (Add):

<PSEUDOCODE>

```
PROCEDURE Push(item)
 IF topPointer < stackFull THEN
 topPointer ← topPointer + 1
 myStack[topPointer] ← item
 ELSE
 OUTPUT "Stack is full"
 ENDIF
ENDPROCEDURE
```

Pop (Remove):

<PSEUDOCODE>

```
PROCEDURE Pop()
 IF topPointer = -1 THEN
 OUTPUT "Stack is empty"
 ELSE
 item ← myStack[topPointer]
 topPointer ← topPointer - 1
 ENDIF
```

11.1.3 Python Implementation

<PYTHON>

```
class Stack:
 def __init__(self, StackFull):
 self.StackFull = StackFull
 self.TopPointer = -1
 self.stack = []
 for i in range(StackFull):
 self.stack.append(0)

 def push(self, item):
 if self.TopPointer < self.StackFull - 1:
 self.TopPointer += 1
 self.stack[self.TopPointer] = item
 else:
 print("Stack is full")

 def pop(self):
 if self.TopPointer == -1:
 print("Stack is empty")
 return None
 else:
 item = self.stack[self.TopPointer]
 self.TopPointer -= 1
 return item

 def printStack(self):
 print(self.stack)
myStack = Stack(10)
myStack.push(5)
myStack.push(10)
print(myStack.pop()) # Returns 10
```

11.1.4 Uses

- Interrupt handling
- Evaluating RPN expressions
- Procedure call stack
- Undo mechanisms

11.2 QUEUES

11.2.1 Characteristics

- **FIFO:** First In, First Out
- Add to rear, remove from front
- Can be circular

11.2.2 Operations

Enqueue (Add):

<PSEUDOCODE>

```
PROCEDURE Enqueue(item)
 IF queueLength < queueFull THEN
 IF rearPointer < upperBound THEN
 rearPointer ← rearPointer + 1
 ELSE
 rearPointer ← 1
 ENDIF
 queue[rearPointer] ← item
 queueLength ← queueLength + 1
 ELSE
 OUTPUT "Queue is full"
 ENDIF
ENDPROCEDURE
```

Dequeue (Remove):

<PSEUDOCODE>

```
PROCEDURE Dequeue()
 IF queueLength = 0 THEN
 OUTPUT "Queue is empty"
 ELSE
 item ← queue[frontPointer]
 IF frontPointer = upperBound THEN
 frontPointer ← 1
 ELSE
 frontPointer ← frontPointer + 1
 ENDIF
```

11.2.3 Python Implementation

<PYTHON>

```
class Queue:
 def __init__(self, QueueFull):
 self.QueueFull = QueueFull
 self.FrontPointer = 0
 self.RearPointer = -1
 self.QueueLength = 0
 self.queue = []
 for i in range(QueueFull):
 self.queue.append(0)

 def enQueue(self, item):
 if self.QueueLength < self.QueueFull:
 if self.RearPointer < self.QueueFull - 1:
 self.RearPointer += 1
 else:
 self.RearPointer = 0
 self.queue[self.RearPointer] = item
 self.QueueLength += 1
 else:
 print("Queue is full")

 def deQueue(self):
 if self.QueueLength == 0:
 print("Queue is empty")
 return None
```

11.2.4 Uses

- Print queues
- Task scheduling
- Breadth-first search

11.3 LINKED LISTS

11.3.1 Characteristics

- Dynamic data structure
- Nodes contain data and pointer to next node
- Can grow/shrink easily

11.3.2 Node Structure

<PYTHON>

```
class Node:
 def __init__(self, item):
 self.Data = item
 self.Pointer = -1
```

11.3.3 Linked List Operations

Insert:

<PYTHON>

```
def add(self, itemAdd):
 # Find position and insert
 # Update pointers
```

Delete:

<PYTHON>

```
def delete(self, itemDelete):
 # Find node
 # Update pointers to bypass
```

Search:

<PYTHON>

```
def search(self, itemFind):
 # Traverse until found or end
```

11.3.4 Uses

- Dynamic memory allocation
- Implementation of other ADTs
- Symbol tables

11.4 BINARY TREES

11.4.1 Characteristics

- Hierarchical structure
- Each node has at most two children
- Left child < parent, Right child > parent (in ordered tree)

11.4.2 Node Structure

<PYTHON>

```
class Node:
 def __init__(self, item):
 self.Data = item
 self.LeftPointer = -1
 self.RightPointer = -1
 self.UpPointer = -1
```

11.4.3 Operations

Insert:

- Start at root
- Compare with current node
- Go left if smaller, right if larger
- Repeat until empty position found

Search:

- Start at root
- Compare with current node
- If match, found
- If target < current, go left; else go right
- Repeat until found or null pointer

Traverse (In-order):

<PYTHON>

```
def inOrder(self, Pointer):
 if Pointer == -1:
 return
```

Revision #1
Created 2026-03-16 12:18:43 UTC by Samuel Lee
Updated 2026-03-16 12:19:01 UTC by Samuel Lee