Advanced Programming

Abstract Data Types

Ramaseshan Ramachandran

1. Arrays and Lists
2. Abstract data Type
   Stack

Queue
Set
Map

ARRAYS AND LISTS

- *Arrays*
  - Elements are stored in sequential memory locations
  - Random access to elements by their index
  - size must be known at compile time
  - Fixed size - not possible to change the size once created

Language	Example
Python	`int_array = [1,2,3]` `int_array = numpy.array([1, 2, 3]) #using numpy module`
C/C++	`int x[] = {1, 2, 3, 4, 5};` `int *intArray = (int *)malloc(size * sizeof(int));`
Rust	`let numbers: [i32; 3] = [1, 2, 3];` `let int_vector = vec![1, 2, 3, 4, 5];`

ABSTRACT DATA TYPE (ADT)

Definition

ADT is a conceptual model for representing and manipulating data, without exposing the internal workings of the data structure.

- It specifies what operations are possible and the semantics/operations
- Specifies the expected behavior of these operations
- Does not specify how there semantics should be implemented

Key characteristics

- Encapsulation - data and operations
- Well defined operations
- Invariance - rules that must always hold true for the data structure and the operations on the data structure

Data Representation

- Each ADT specifies data type - could be concrete or custom data structure

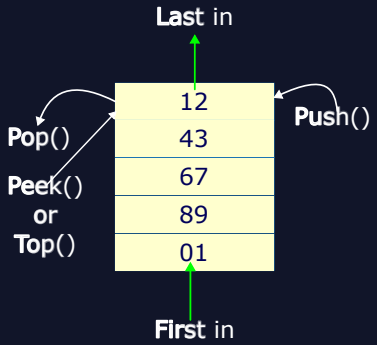
Example - Virtual Payment Address (VPA) used in UPI -

STACK

Stack is a Last-In, First-Out (LIFO) Abstract Data Type

Operations allowed on a stack

1. `Push()` an entry to the top of the stack
2. `Pop()` removes the top of the stack (youngest!) and returns it
   `undo operations in an editor`
3. `Top()/Peek()` Check/view the last entry at the top the stack
4.



LINKED LISTS

A linked list is a linear ADT consisting of a sequence of elements, where each element points to the next element in the sequence

Key components

- `Node`: Each element in the linked list is a node, containing data and a reference (link) to the next node
- `Head`: The first node in the list
- `Tail`: The last node in the list
- `Null`: Indicates the end of the list

- Dynamic structure - can grow or shrink as needed
- Efficient insertion and deletion anywhere
- Random access is slower than arrays
- Extra memory overhead for pointers

Recursion

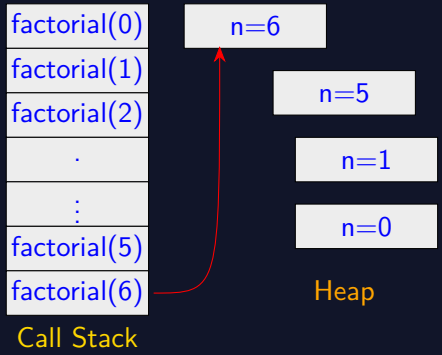
A problem solving technique in which problems are solved by using a smaller version of the original problem until a `base case (must have property)` is reached, at which point the function returns a result without making any further recursive calls

- Follows a **`divide and conquer`** approach
- Each subproblem is solved independently
- Solutions are combined to solve the original problem
- Each recursive call adds a new stack frame
- On reaching the base case, the stack unwinds
- The results are combined (`conquer part`)as the stack frames are popped off.

STACK FRAMES

```
def factorial(n):
    return 1 if (n==1 or n==0)\
        else n * factorial(n-1)
```

```
def factorial(n):
    if n < 0:
        return 0
    elif n == 0 or n == 1:
        return 1
    else:
        fact = 1
        while(n > 1):
            fact *= n
            n -= 1
    return fact
```



PROGRAMMING EXERCISE - RECURSION

- For any $n \geq 1$, find x^n, using recursion
- Find whether a given string is a palindrome or not. What is the base case?

  Test cases - racecar, madam, Rotator
- Permutations of an array. What is the base case?

RECURSION DEPTH

- Every time a stack frame is created
- Frame holds local variables, function arguments, and the return address
- Refers to the number of times a function can call itself recursively
- Python specify a limit for the depth - This limit prevents infinite recursion and avoids program crashes.

```
# Python has a built in featire that prevents infinite
  recursion
import sys
print(sys.getrecursionlimit())
#1000
```

STACK DEPTH LIMITS IN SOME PROGRAMMING LANGUAGES

Language	Limit
Python	`Default = 1000`. It is possible to increase using `sys.setrecursionlimit()`
C/C++, Rust, Go	No limit - leaves it to the programmer. If not handled cleanly, it could lead to a stack overflow and a program crash
Java	No limit, but can be configured with specific JVM arguments such as initial and maximum Java Virtual Machine (JVM) heap size using the `-Xms and -Xmx and -Xss` options

POSTFIX NOTATION

Definition

Postfix notation, also known as Reverse Polish Notation (RPN), is a mathematical notation in which every operator follows all of its operands

- Evaluate expressions by scanning from left to right
- When an operand is encountered, push it onto the stack
- When an operator is encountered, pop the operands, evaluate the expression and push the result onto the stack

Infix	Postfix
$2+3$	$23+$
5-3	5 3 -
2*6	2 6 *
$8/2$	8 2 /
(4 * 2) + (12 / 3)	4 2 * 12 3 / +
(4*15)-(18/2) + 9	15 4 * 18 2 / - 9 +

QUEUE

Queue is a First-In, First-Out (FIFO) Abstract Data Type

A queue implementation must have the following operations

1. `enqueue(item)` - adding an item at the end of the queue
2. `dequeue()/remove()` - removing the first item from the queue and return it
3. `peek()/front()` - check the first item in the list
4. `size()` - returns the number of items in the list
5. `is_empty()` - returns a `bool`, if the queue is empty

```
...
queue = []
# Enqueue elements (append to the end)
queue.append("Doe") #  Enqueue elements
queue.append("Dee") # (append to the end)
# Dequeue elements (remove from the beginning)
print(queue.pop(0))  # Output: Doe
print(queue.pop(0))  # Output: Dee
```

QUEUE - PYTHON IMPLEMENTATION

```
from queue import Queue
my_q = Queue()
#Enqueue
my_q.put(12)
my_q.put(45)
#Dequeue
q_item = my_q.get() #returns 12
q_item = my_q.get() #returns 45
print(my_q.qsize())
```

Set

A set is an unordered collection of distinct elements. The elements can be anything: numbers, letters, objects, etc. The order of elements does not matter, and each element is unique within the set.

Operations on a set[1]

- `add(element)`: Adds a new element to the set.
- `remove(element)`: Removes an element from the set, if present.
- `contains(element)`: Returns True/False, depending on the presence/absence of the element
- `is_empty()`: Checks if the set is empty, returning True if it contains no elements, False otherwise.
- `size()`: Returns the current number of elements in the set.

NOTE: `set()` object is not subscriptable

[1] Python provides several built-in set operations, such as union, intersection, difference, membership testing (subset or superset)

SET - EXAMPLES

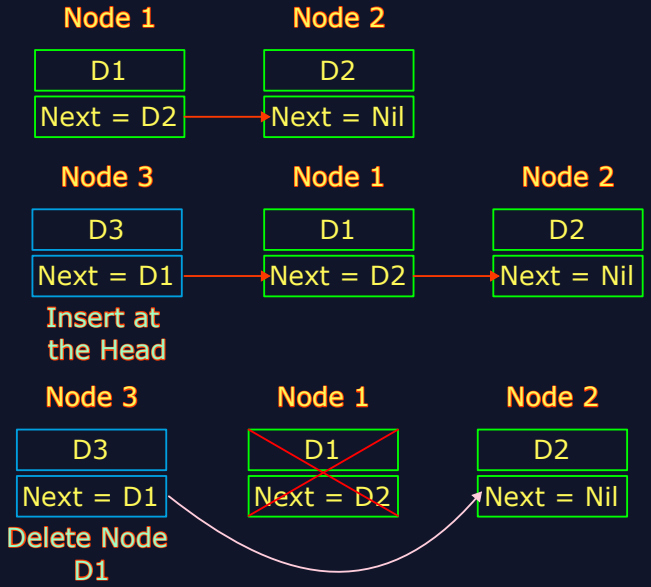
- Finding unique words in a huge collection of text
- Average complexity of checking for the membership element in Python is $\mathrm{O}(1)$. Python uses hashing strategies (hashmap[2])to implement set. Hence, the worst complexity is $\mathrm{O}(n)$

[2]We will cover this topic later

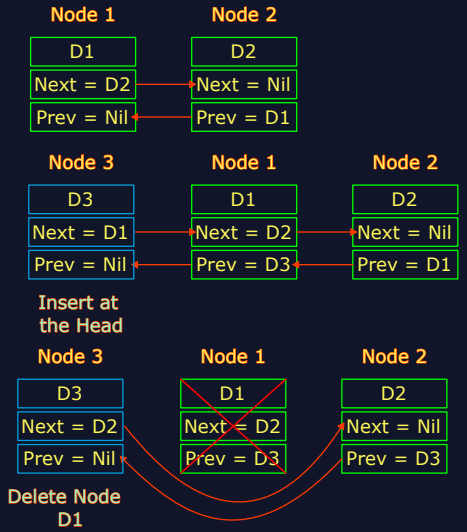
MAP

Self Study. This is called as dictionary in Python 😃

LINKED LIST


Node 1
D1
Next = D2
Node 2
D2
Next = Nil
Node 3
D3
Next = D1
Node 1
D1
Next = D2
Node 2
D2
Next = Nil
Insert at the Head
Node 3
D3
Next = D1
Node 1
D1
Next = D2
Node 2
D2
Next = Nil
Delete Node D1

DOUBLY LINKED LIST



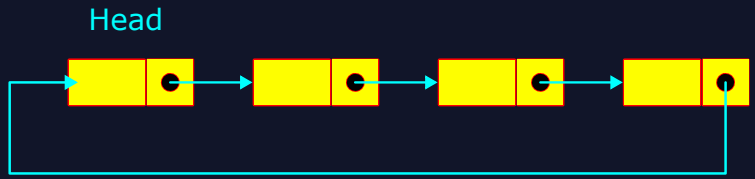
D1.Prev ← D1 ▷ //Insert Op//
D3.Next ← D1
D3.Prev ← Nil
D2.Prev ← D3 ▷ //Delete Op - D1//
D3.Next ← D2
Del Node D1

DEQUE OPERATIONS

Function Name	Function	Complexity
`enqueue_front(item)`	places given item at the front of the queue	$O(1)$
`enqueue_back(item)`	places the item at the back	$O(1)$
`dequeue_front`	removes front value and returns	$O(1)$
`dequeue_back`	removes back value and returns it	$O(1)$
`peek_front`	returns the front value without removing it	$O(1)$
`peek_back`	returns the back value without removing it	$O(1)$

- **Use cases**
  - **Web Browsers**: Implementation of $\triangleleft$ and $\triangleright$ buttons
  - **Text Editors**: Implementation of ↺ and ↻
  - **Music Players**: Play lists with insertions, deletions, and reordering ($=$) of songs

CIRCULAR LISTS



- The head will point to NULL at the beginning or when there no nodes
- Possible to access all of the list starting at any given point
- What are the basic operations on circular lists?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.head = None
        # Empty list: both
   head and tail are null
        self.tail = None
```

CUSTOM ABSTRACT DATA TYPES?

Definition

A concept that hides the details of the operations on its variables, but allows external world to access its complex functionality

C allows structures while C++ enhanced it with classes

```
/* C structure*/
struct name {
    char first_name[50];
    char last_name[50];
    int age;
};
```

```
    class name {
        private:
            char first_name[50];
            char last_name[50];
            int age;
        public:
            Person(const std::string& first,
                   const std::string& last, int a)
        : first_name(first), last_name(last), age(a) {}
    };
```

WHAT ARE CLASSES?

A class has two parts

- An Interface (abstraction)
- An implementation

Languages like Rust separate the interface and the implementation as given below:

```
struct Person {
    fname: String,
    lname: String,
    age: i32,
}
impl Person {
    pub fn new(first_name: &str, last_name: &str, age: i32)
   -> Person {
        Person {
            fname: first_name.to_string(),
            lname: last_name.to_string(),
            age,
        }
    }
    pub fn get_lname(&self) -> &String {&self.lname}
    pub fn set_lname(&mut self,
            new_lname: String) {
        self.lname = new_lname;
     }
}
```

IMPLEMENTATION OF A STACK

```
class Stack:
    def __init__(self):
        self.top = None

    def push(self, frame):
        frame.next = self.top
        self.top = frame

    def pop(self):
        if self.top is None:
            raise Exception("Stack is empty")
        frame = self.top
        self.top = frame.next
    return frame
```

CLASS STACKFRAME

```
class StackFrame:
    def __init__(self, return_address, local_variables,
   arguments):
        self.return_address = return_address
        self.local_variables = local_variables
        self.arguments = arguments
        self.next = None  # for chaining purpose
#Usage
if __name__ == '__main__':
    stack = Stack()
    # Simulate a function call with arguments and local
   variables
    frame1 = StackFrame(10, {"x": 5}, ["hello", "world"])
    stack.push(frame1)

    # Pop frames to simulate function returns
    stack.pop()
    print(stack.top.local_variables)
```

LINKED LIST IMPLEMENTATION

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next_node = None

class LinkedList:
    def __init__(self):
        self.head = None
# All the operations on a list
    def is_empty(self):
    #Complete it
    def append(self, data)
    #
    ...
```