Abstract Data Types (Moved from paper 4)

- An abstract data type is a collection of data and set of operations on that data

- Abstraction: To remove any unwanted data.

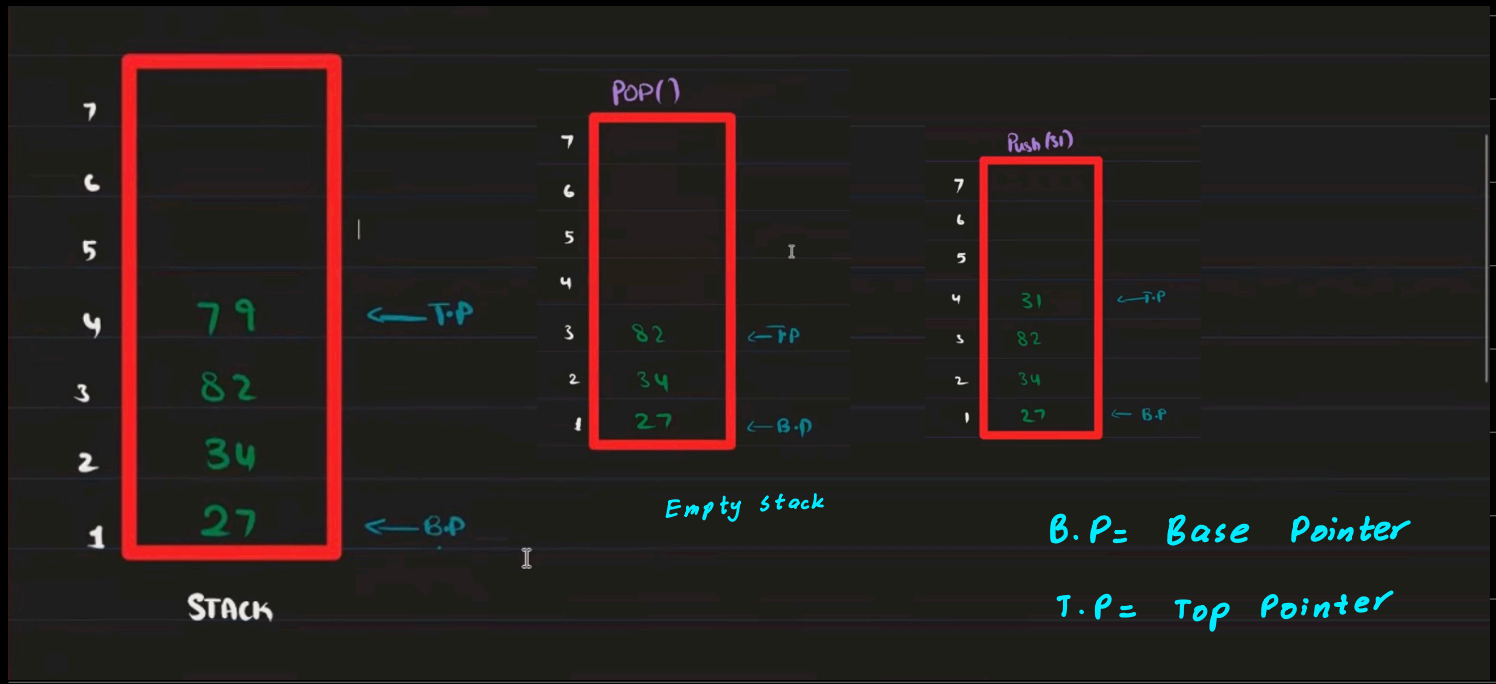
- Three types

① STACKS ② Queues ③ LINKED LIST

Data structure: data organization, management, and storage format that enables efficient access and modification of data.

STACKS

- A list containing several items operating on the last-in first-out (LIFO) principle. Items can be added to the stack (Push) and removed from the stack (Pop). The first item added to the stack is the last item to be removed from the stack.

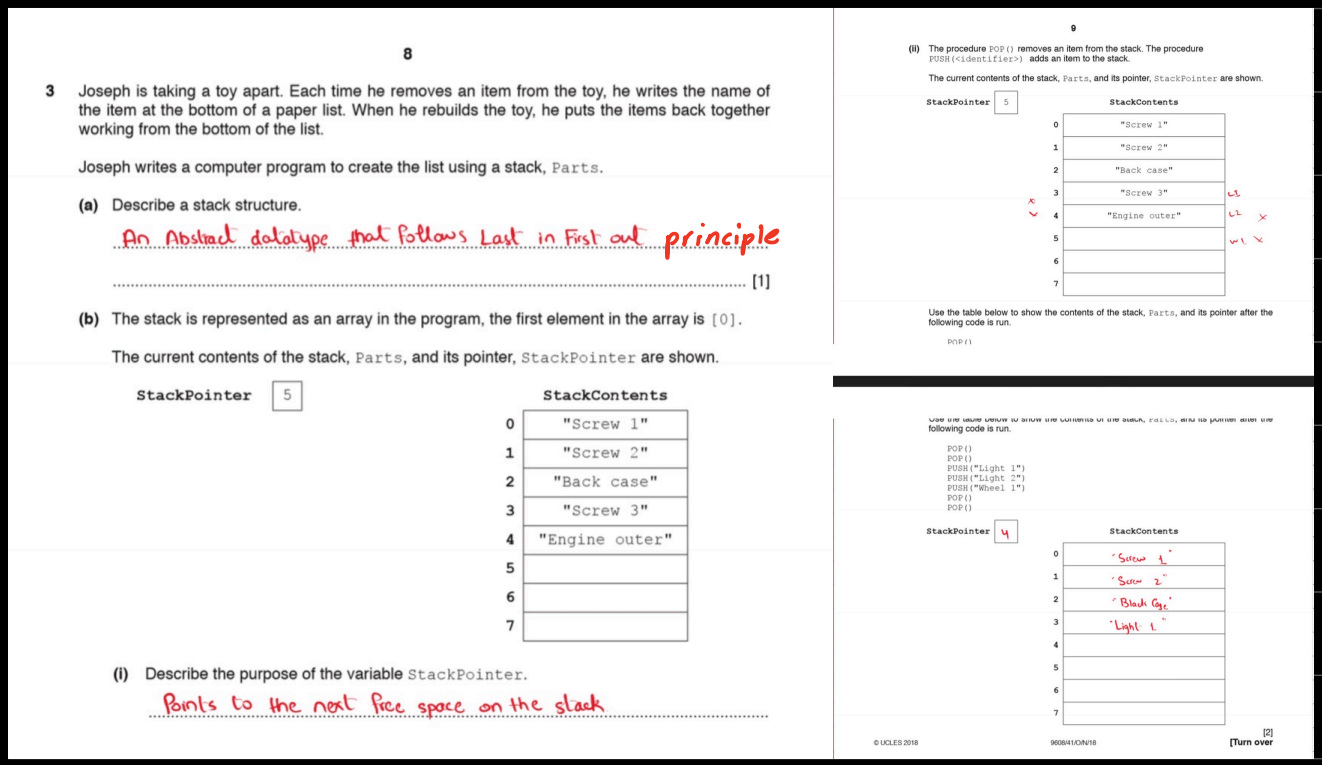


use

- Stack can ↑ 2 pointers
  - Value of Base Pointer always remain the same.

- Base Pointer always points to the first element in the stack.

- Top pointer always points to the last item in the stack, changes after PUSH() and POP() operations



8

3 Joseph is taking a toy apart. Each time he removes an item from the toy, he writes the name of the item at the bottom of a paper list. When he rebuilds the toy, he puts the items back together working from the bottom of the list.

Joseph writes a computer program to create the list using a stack, `Parts`.

(a) Describe a stack structure.

An Abstract datatype that follows Last in First out principle

..... [1]

(b) The stack is represented as an array in the program, the first element in the array is `[0]`.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

StackPointer	5

	StackContents
0	"Screw 1"
1	"Screw 2"
2	"Back case"
3	"Screw 3"
4	"Engine outer"
5	
6	
7	

(i) Describe the purpose of the variable `StackPointer`.

Points to the next free space on the stack

9

(ii) The procedure `POP()` removes an item from the stack. The procedure `PUSH(<identifier>)` adds an item to the stack.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

StackPointer	5

	StackContents
0	"Screw 1"
1	"Screw 2"
2	"Back case"
3	"Screw 3"
4	"Engine outer"
5	
6	
7	

Use the table below to show the contents of the stack, `Parts`, and its pointer after the following code is run.

```
POP()
POP()
PUSH("Light 1")
PUSH("Light 2")
PUSH("Wheel 1")
POP()
POP()
```

StackPointer	4

	StackContents
0	"Screw 1"
1	"Screw 2"
2	"Black Case"
3	"Light 1"
4	
5	
6	
7	

[2]

© UCLES 2018 9608/41/O/N/18 [Turn over

- When B.P = T.P, there is only one item in the stack.

Queues

- A list containing several items operating on the FIFO (First-in First-out) principle. Items can be added (enqueue) and removed (dequeue)
- The first item added is the first item to be removed from Queue.

Q- Describe a Queue

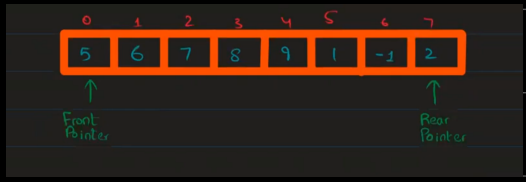
- Linear data structure

- First - in first out logic → An item is added to the end of the queue and removed from the front.
- All items are kept in the order they are

Front pointer: pointing towards first item in queue, and a rear pointer: pointing towards last item in queue

- Insertion is done at rear pointer
- Deletion is done at front pointer

- Condition for a queue being full is that rear pointer should point towards upperbound / max index

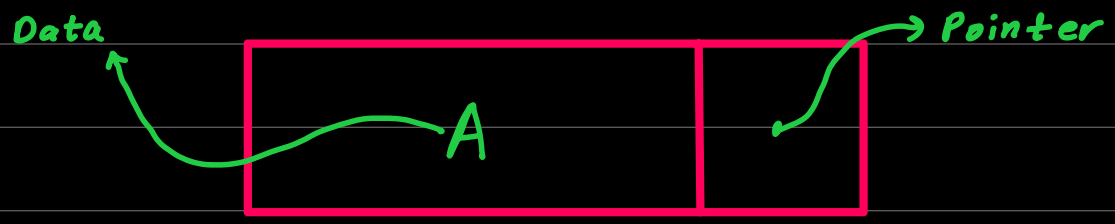


Note: Refer to the end of the notes for Circular Queue

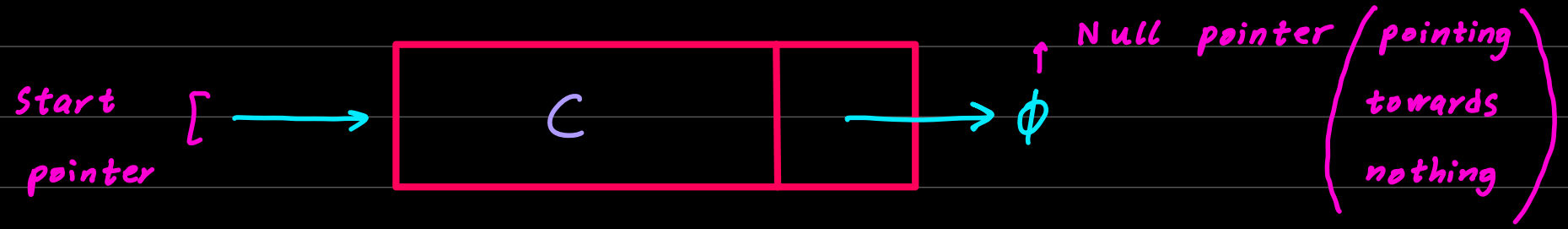
Linked List

· A list containing several items in which each item in the list points to the next item in the list. In a linked list, a new item is always added to the start of the list.

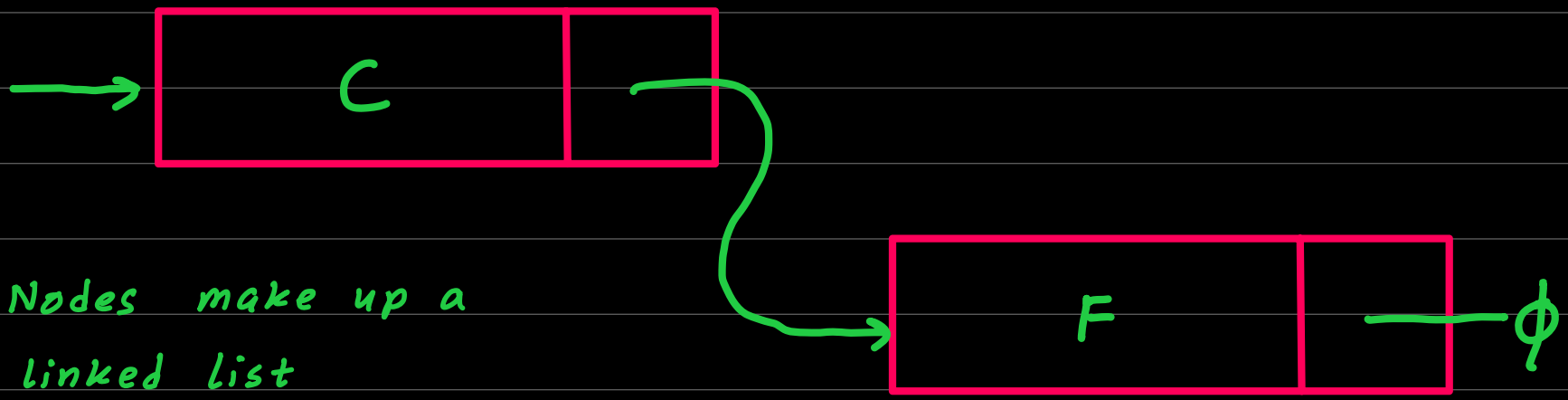
NODE



· A node contains data and a pointer which points towards memory location.

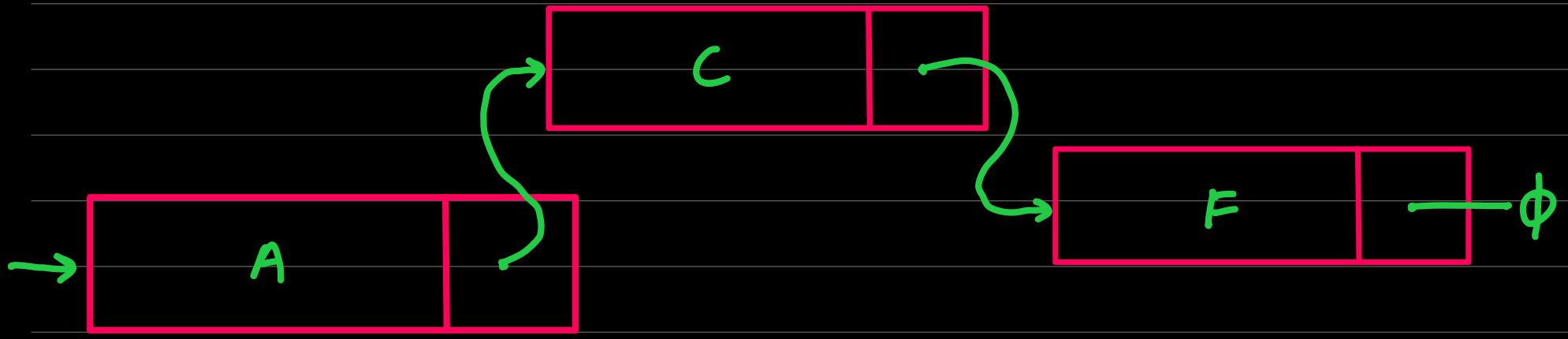


Q- Now connect one more element F

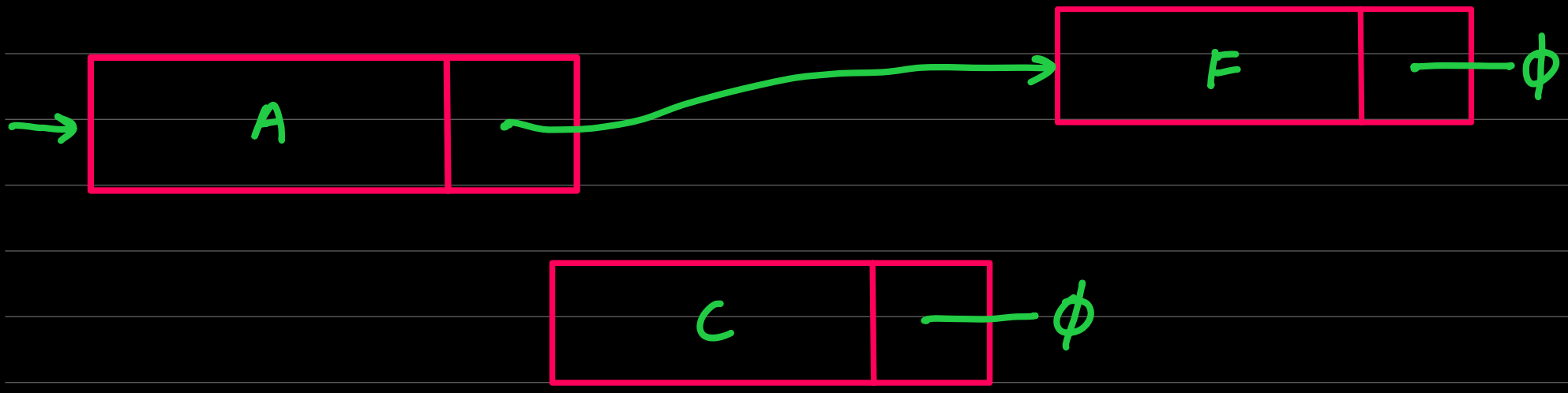


- Nodes make up a linked list

Q- Now connect one more Element A to the start.

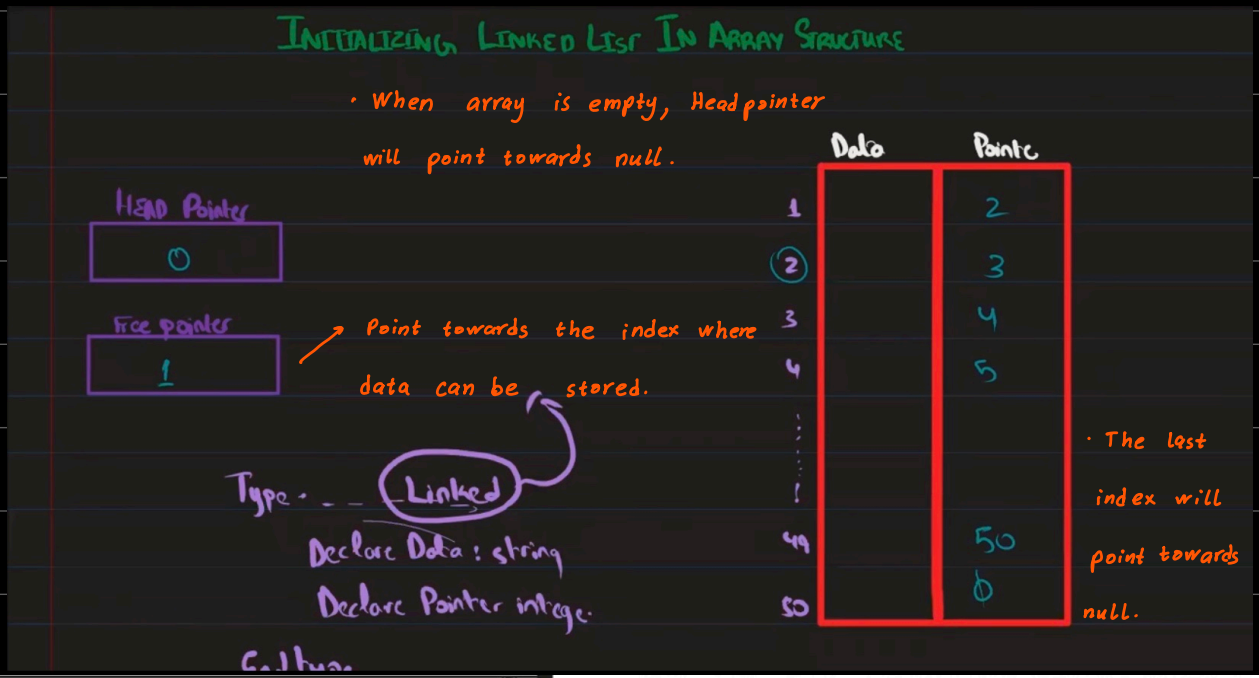


Q- Now remove element C



- Data item remains when removing it from a linked list.
- Now two seperate linked lists are present.

Initializing Linked List in Array structure



(b) A programming language provides built-in array data structures. This linked list is to be implemented using these array data structures.

Define a record type, `ListNode`, for each node.

```
TYPE ListNode
    Declare Data : string
    Declare Pointer : integer
ENDTYPE
```

[3]

(c) Write an array declaration to reserve space for 50 nodes in array `NameList`.

```
Declare NameList[1:50] OF ListNodes
```

[2]

(d) (i) The `CreateLinkedList` operation links all nodes to form the free list and initialises the `HeadPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers.

NameList

HeadPointer: 0

FreePointer: 1

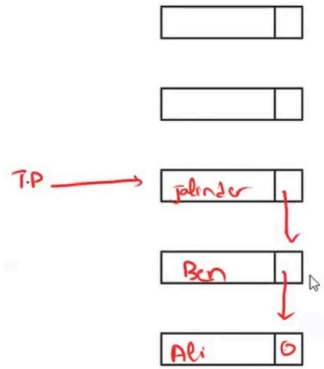
	Name	Pointer
[1]		2
[2]		3
[3]		4
[4]		5
:		
[49]		50
[50]		0

[4]

Stacks with Linked List

```
CreateStack
Push("Ali")
Push("Jack")
Pop
Push("Ben")
Push("Ahmed")
Pop
Push("Jatinder")
```

Add appropriate labels to the diagram to show the final state of the stack. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

```
TYPE Node
  DECLARE Name : STRING
  DECLARE Pointer : INTEGER
ENDTYPE
```

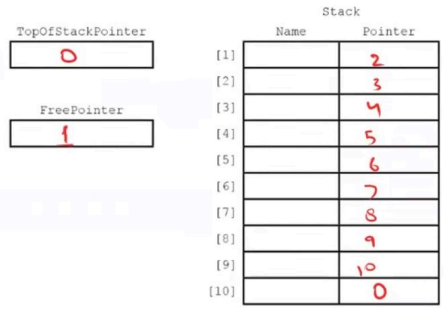
The statement

```
DECLARE Stack : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Stack`.

(i) The `CreateStack` operation links all nodes and initialises the `TopOfStackPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateStack` has been executed.



[4]

Queues with Linked List

```
TYPE Node
   DECLARE Name    : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
```

The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.

HeadPointer: 0

TailPointer: 0

FreePointer: 1

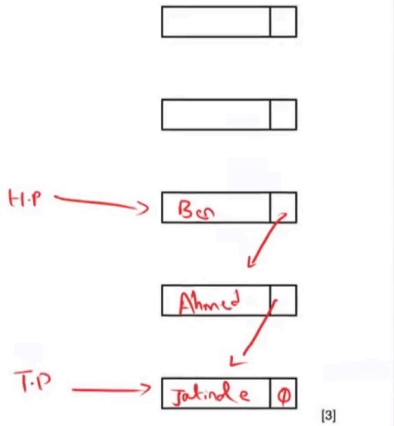
Queue

	Name	Pointer
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		7
[7]		8
[8]		9
[9]		10
[10]		0

[4]

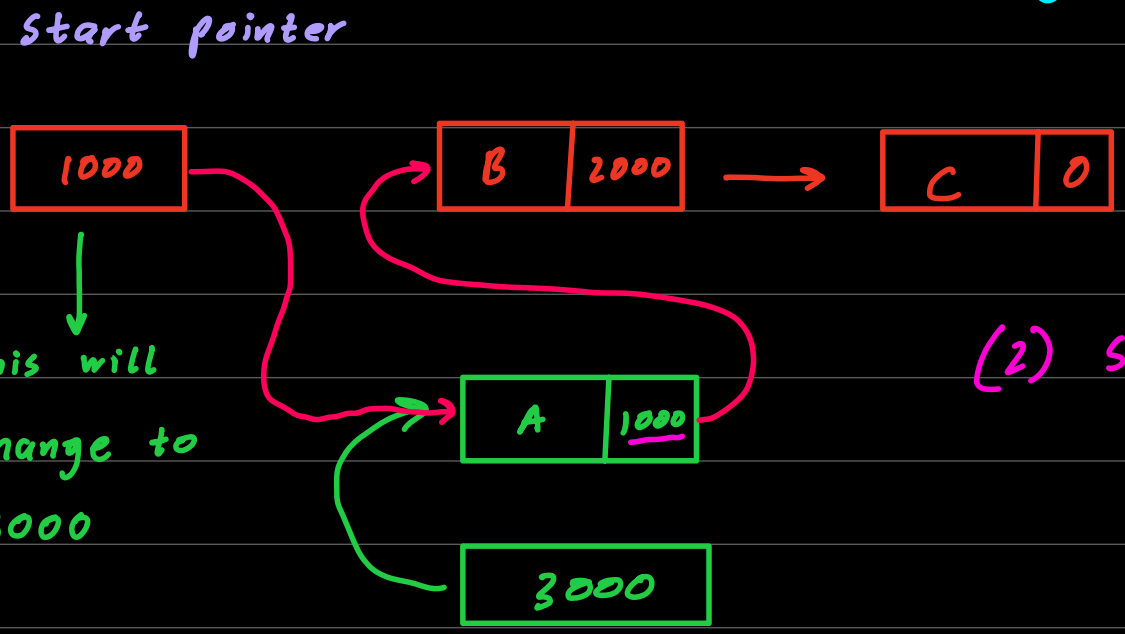
```
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

Insertion of node in the beginning



(2) Startpointer = Temppointer

· Previous pointer of a node contains the address of the next node.

(1) First we will connect pointer of A to pointer of B by using the address in the start pointer.

(2) Then we will point the head pointer to A by using Temp pointer.

Q- Why we can not point head pointer to A first?

or start pointer in above case

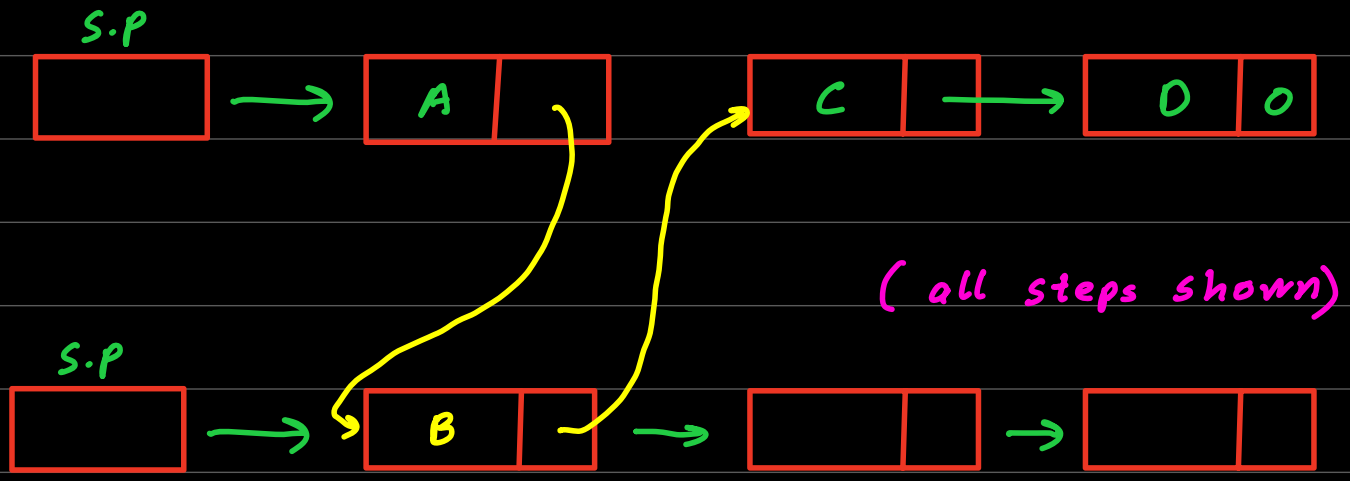
- Because after that we will not have another way to link pointer B to A, the reference of point B will be lost.

- Pointers are variables

Insertion of node in the middle

Note: Use a free list

Q- Add data value B in between A and C



(all steps shown)

1. Check for a free node in a linked list
2. Search for a correct insertion point
3. Assign data value B to the first node in free list
4. Pointer from A will be changed to point to node containing B
5. Pointer from B will be changed to point to node containing C.

⑥ Start pointer in free list moved to point to next free node.

Q- How the linked list may be implemented by using arrays and variables?

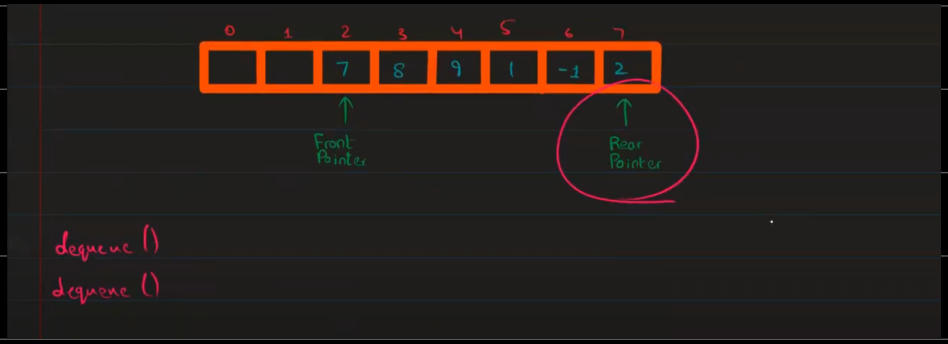
- Define a record data type containing a data element and a pointer and declare an array of this type.
- An integer variable to hold the start pointer and an integer variable to store the next free pointer.

- Start pointer :
  - Tells us from where linked list starts
  - If pointing towards null, then list is empty

- Free pointer :
  - Tells us the next index position where data can be stored

Circular Queue

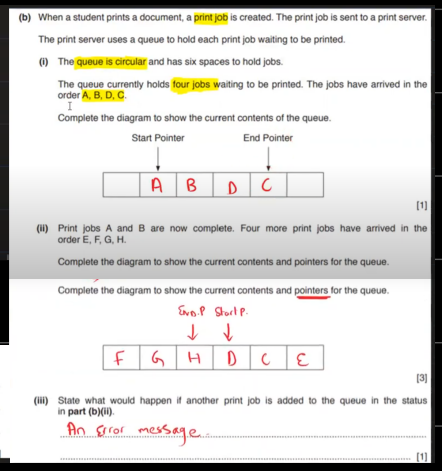
Problem:



- The condition for queue being full is met as Rear pointer points at max index but queue in the above case is not full b/c empty space is present.

Solution:

- When rear pointer reaches max index and data is enqueued, rear pointer points towards the lowest index in the Queue



(b) When a student prints a document, a print job is created. The print job is sent to a print server.

The print server uses a queue to hold each print job waiting to be printed.

(i) The queue is circular and has six spaces to hold jobs.

The queue currently holds four jobs waiting to be printed. The jobs have arrived in the order A, B, D, C.

Complete the diagram to show the current contents of the queue.

Start Pointer End Pointer

	A	B	D	C	

[1]

(ii) Print jobs A and B are now complete. Four more print jobs have arrived in the order E, F, G, H.

Complete the diagram to show the current contents and pointers for the queue.

Complete the diagram to show the current contents and pointers for the queue.

End P. Start P.

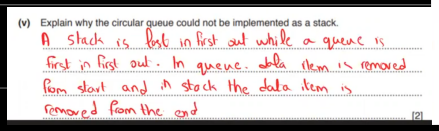
F	G	H	D	C	E

[3]

(iii) State what would happen if another print job is added to the queue in the status in part (b)(ii).

An error message.

[1]



(v) Explain why the circular queue could not be implemented as a stack.

A stack is first in first out while a queue is first in first out. In queue, data item is removed from start and in stack the data item is removed from the end

[2]