શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

Assignment - 1

1) What is a data structure? Explain types of Data structures.

→ Data structure is a method of organizing large amounts of data more efficiently so their easy operation on that data becomes easy.

Any processed data is also named as information.

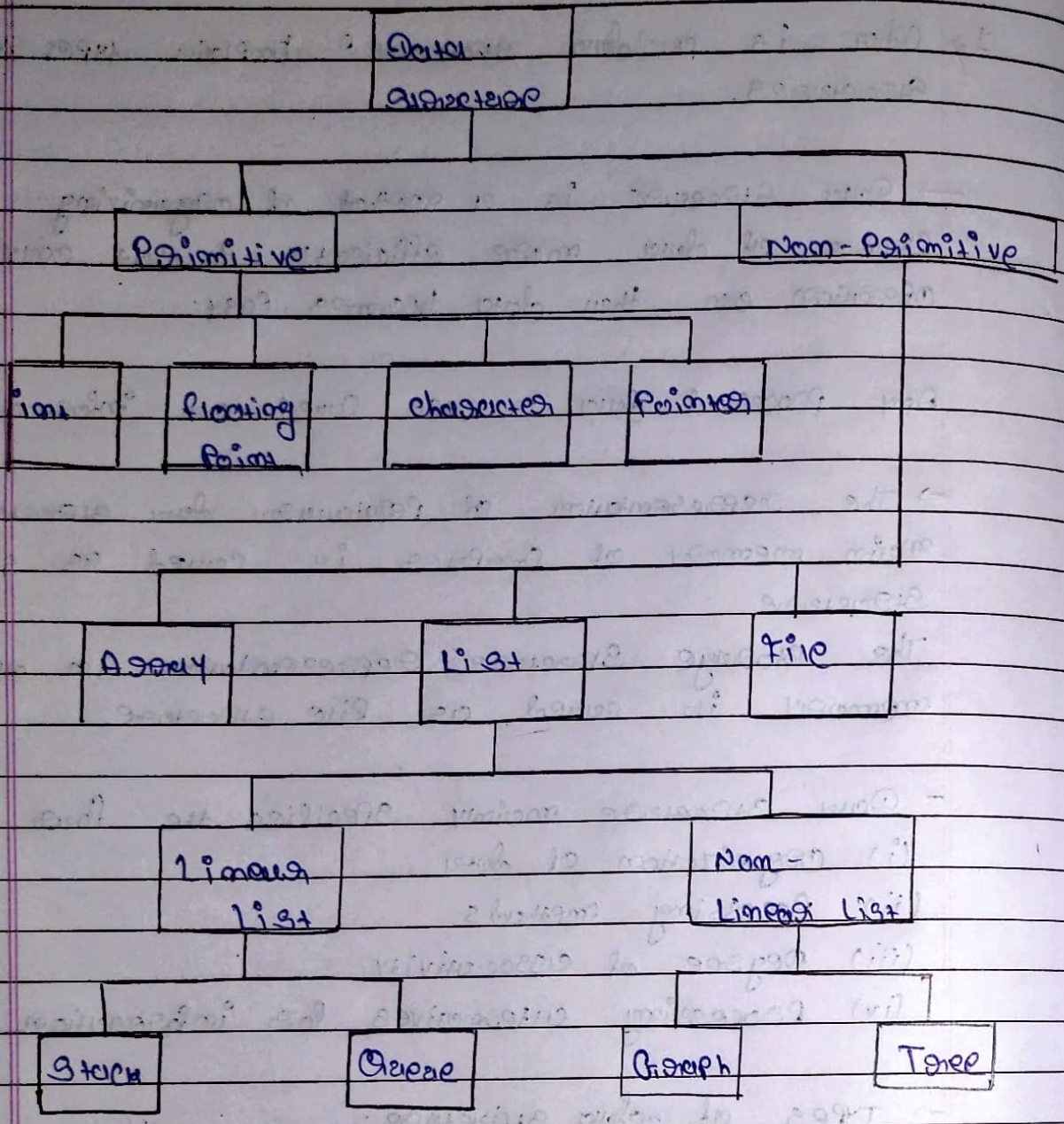
→ The representation of particular data structure in main memory of computer is called as storage structure.
The storage structure representation in auxiliary memory is called as file structure.

- Data structure mainly specifies the four things :-
  (i) Organization of data
  (ii) Accessing methods
  (iii) Degree of associativity
  (iv) Processing alternatives for information.

→ Types of data structure :

(1.) Primitive Data structure

(2.) Non - Primitive Data structure



(1.) Primitive data structure :

- These are basic structures and directly operated upon by machine instructions.
- They have different representations on different computers
- Ex. integer, float, characters & Pointers

શ્રી ઉમરગુજરાત બારગામ
DATE :
PAGE :

(2.) Non - Primitive data structure :

- These are sophisticated data structure.
- These are derived from primitive data structure.
- The non - primitive data structure emphasize on storing of a group of homogeneous or heterogeneous data items.

- Array : fixed size sequenced collection of elements of the same type.
- List : An ordered set containing variable number of elements is called as list.
- File : A collection of logically related information.

2.) Difference between primitive & non - primitive data - structure.

→	Primitive data - structure	non - primitive data structure
	- Primitive Data - types are pre - defined by the language.	- non - primitive data - type need to be defined by the user.
	- When declared primitive data stored in the stack.	- The reference variables are stored in the stack but original object is in the heap.

- When a copy is created for the data, a completely separate allocation is done for copy.	- Copies have create new reference variables but they both point to the same object in the heap.
- The primitive data structure will contain some value	- The non-primitive data structure can be NULL
- The size depends on the type of OS	- Size is not fixed.
- It can be used to call the methods	- It can't be used to call the method.

3.) Differentiate Linear and non-linear Data Structures.

Li. Linear DS	Non-Linear DS
- Every item is related to it's previous and next item	- Every items is attached with many other items.
- Data is arranged in linear sequence	- Data is not arranged in sequence
- Data items can be traversed in a single run	- Data cannot be traversed in a single run

શ્રી ઉમિયા ગુજરાતી વાચનાલય
DATE :
PAGE :

- Implementation is easy	- Implementation is difficult
- Ex. Array, Stack, Queue, Linked list	- Ex. Graph, Tree

4.) Differentiate static data-structure and dynamic data-structure

Static	Dynamic
- Memory allocated at compile time	- Memory allocated at run-time.
- Size it is fixed and cannot be modified	- Size can be modified during run time
- Some times memory utilization is inefficient	- Memory utilization is efficient as memory can be reused.
- Access time is faster as it is fixed	- Access time may be slower due to indexing and pointer usage
- Ex. Arrays, Stack, Queue Trees. (with fixed size)	- List, tree (with variable size), Hash tables.

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

Q:) List the various operations which can be performed on a data structure.

→ Create :
The create operation results in reserving memory for program elements.

Destroy :
Destroy operation destroy memory space allocated for specified data structure.

Selection :
Selection deals with accessing a particular data.

Updation :
Update or modify the data in DS

Searching :
It find the presence of desired data item in the list of data items

Sorting :
Arrange all items in particular order

merging : Combine the data item of two different sorted list.

Traversal : visiting each and every node of list.

Assignment 2

1) Create a SPARSE matrix of the following matrix.

→

$$\begin{bmatrix} 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 6 \\ 9 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 \end{bmatrix}_{4 \times 4}$$

→ Using Array.

Row	0	0	1	2	3
column	2	3	3	0	0
value	5	2	6	9	7

2) List various operations which can be performed on a STACK and explain in brief.

→ Operations which are perform on a stack

- Push
- Pop
- Peep
- To change

→ PUSH (S, top, x)

- Insert an element x to the top of a stack.

```
1. [Check for stack overflow]
   if (top >= N-1) then
   write ("Stack overflow")
   return;
```

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

2. [Increment top pointer by 1]
   TOP ← TOP + 1

3. [Insert Element]
   S[TOP] ← X

4. [Finished]
   Return

→ POP (S, top)
- Removes the top element from stack.

1. [Check for stack underflow]
   if (top = -1) then
   write ("stack is underflow")
   return;

2. [Remove the top element]
   x ← S[top]
   top ← top - 1

3. [Return top element of the stack]
   write (x)

4. [Finished]
   Return

→ PEEP (S, top, index)
- ~~Peep~~ to read the element of stack at given index.

શ્રી ઉમરા ગુજરાતી બાલ્ગામ
DATE :
PAGE :

1. [check for stack underflow]
   if (top - i+1 <= 0) then
   write ("Stack Underflow")
   return;

2. [Return index element from top of stack]
   ~~write (x)~~
   x ← S[top - index + 1]
   write (x)

3. [Finished]
   Return.

-> EX_CHANGE (S, top, value, index)
   - to overwrite the value at given index.

1. [check for stack underflow]
   if (top - index + 1 < 0) then
   write ("Stack Underflow")
   return.

2. [Change the index element with value f
   top of stack]
   S[top - index + 1] ← x

3. [Finished]
   Return

3.) Convert the expression $(a+b\hat{}c\hat{}d)*(e+f/d)$ from infix to postfix.

→ Expression : $(a+b\hat{}c\hat{}d)*(e+f/d)$

Postfix : $abc\hat{}d\hat{}+efd/+*$

Symbol	Stack	Top of the Stack	Postfix string	Rank
(				0
a	(	(	a	0
+	(	(	a	1
b	(+	+	a	1
^	(+^	^	ab	2
c	(+^	^	ab	2
^	(+^	^	abc	3
d	(+^	^	abc^	2
)	(+^	^	abc^d	3
*			abc^d^+	1
(	*	*	abc^d^+	1
e	*	*	abc^d^+	1
+	*+	+	abc^d^+e	2
f	*+	+	abc^d^+e	2
/	*+	+	abc^d^+ef	3
d	*+/	/	abc^d^+ef	3
)	*+/	/	abc^d^+efd	4
			abc^d^+efd/+*	1

શ્રી ઉત્તર ગુજરાત બારડોલી
DATE :
PAGE :

4.) Convert the expression AB-CDE+↑FG+- from Postfix to infix

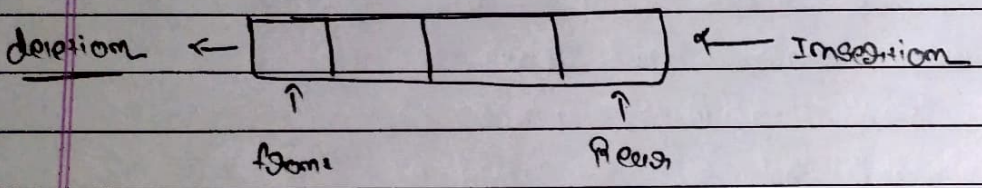
Q.) List various operations which can be performed on a QUEUE and explain them in brief.

→ Enqueue – To insert the elements in queue

Dequeue – To remove the element in queue.

- The Queue is a linear list which permits deletion to be performed at one end of the list and insertion at the other end.

- List processed 'FIFO pattern.



- Enqueue (Q, rear, size, value, front)

1. [Check another Queue is full or not]
   if rear = size - 1 then
   Write ("Queue is full")
   Return.

2. [Increment rear pointer]
   rear ← rear + 1

3. [Insert element]
   Q[rear] ← value

શ્રી ઉમિયા પ્રસાદ બારડાય
DATE :
PAGE :

4. [Is front property set or not]
if front = = -1 then
front ← 0
return

5. [Finished]
Return

- Dequeue (Q, front, size, value)

1. [Check for Underflow]
if Front = = value then
write ("Queue is empty")
return

2. [Read front element]
x ← Q [front]

3. [increment front pointer by 1]
front ← front + + 1

4. [write deleted element]
write (x)

5. [Finished]
Return

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

6) Explain ENQUEUE and DEQUEUE operation on a circular Queue using algorithm / code

→ Enqueue :

1. ~~check~~ [check queue is ~~em~~ full]
   if ((front = 0 and rear = size - 1) OR (front = rear + 1)) then write ("Queue is Full")

- Here, first front and rear both are set at -1 that means empty when a first element insert by rear then front set to 0. when.
  (i) when you enter all elements by size
  (ii) If any free space available ~~to~~ by deleting element then we have to fill it to be considered as full so, In circular every when front pointer is near to rear pointer then it consider as Full Queue.

2. [Set rear Pointer]
   rear = (rear + 1) % size

- This expression work as if rear = size - 1 then rear = 0 else rear = rear + 1.
- It add element in circular way.

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

3. [Insert element]
Q [rear] ← value

4. [Set front pointer if it is not set]
if front = -1 then
front = 0

- For, first element front = -1 we set front 0 for deletion function

5. [Finished]
Return

→ Dequeue :

1. [Check whether Queue is empty or not]
if front = -1 then
write ("Queue is empty")
- when we insert first element in queue we set front -1 to 0 if front is -1 which means there is no element in queue

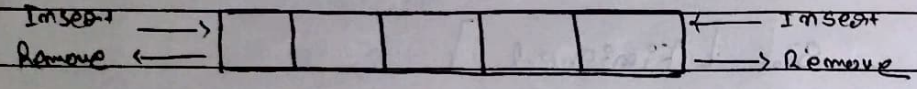
2. [Take front element and write it]
x ← Q [Front]
write (x)

3. [Set front pointer]
front = (front + 1) % size
- As queue Pointer same only increment front Pointer with this expression

4. [Finished]
Retrain

7.) List the types of Double Ended Queue and explain them in brief.

→ Double Ended Queue is a type of queue which insertion and removal of elements can either be performed from the front or the rear



→ Types of Double Ended Queue

- Input Restricted Queue
  ↳ Input restricted at single end but deletion allow at both ends

- Output Restricted Queue
  ↳ Deletion restricted at single end but insertion allow at both ends

Q.) Write an algorithm / code to create singly linked list.

→ ~~Insert at first~~ Insertion

- Infirst (list_ref , value)

1. new_node = get node ()
2. new_node (info) ← value
3. new_node (next) ← list_ref
4. list_ref ← new_node
5. Return.

- In_End (list_ref , value)

1. new_node = get node ()
2. new_node (info) ← value , new_node (next) = N

3. [create temp node to traverse to last node]
   ~~node~~ temp ← list_ref

4. [check if temp is NULL]
   if temp = NULL then
   ~~insert~~ Infirst (list_ref , value)
   return.

5. [traverse to the last node]
   (i) while (temp (next) != NULL)
   temp ← temp (next)
   (ii) temp (next) ← new_node

6. Return.

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

• InPosition (list_ref , value , Position)

1. new_node = get node ()
2. [check it's first element or not]
   if (list_ref = NULL) then
   InFirst (list_ref , value)
   return.
3. [Traverse to the Insertion at Position]
   (i) temp ← list_ref
   (ii) while (i < Position - 1)
   temp ← temp (next)
   i ← i + 1
   (iii) next Temp ← temp (next)
4. [get Pointers]
   (i) temp (next) ← new_node
   (ii) new_node (next) ← next Temp
   (iii) new_node (info) = value
5. [Finished]
   Return.

→ Deletion :-

• Re RemoveFirst (list_ref)

1. [Check: List is empty or not]
   if (list_ref = NULL) then
   write ("List is empty")
   return.

(head)

2. [set list_ref to the next node]
   (i) temp ← list_ref
   (ii) list_ref ← temp (next)

3. [free node]
   (i) write (temp (info))
   (ii) free (temp)

- RemoveEnd (list_ref)

1. [check list is empty]
   if (list_ref = NULL) then
   write ("List is empty")
   return

2. [set the last node]
   (i) temp ← list_ref
   (ii) ~~write~~ ~~temp~~ nextTemp ← temp (next)
   (iii) while (nextTemp (next) != NULL)
   ~~one~~ temp ← temp (next)
   (iv) temp (next) ← NULL
   (v) ~~write~~ & x ← nextTemp (info)
   ~~(vi)~~

3. [free node]
   (i) free (nextTemp)
   return

શ્રી ઉત્તર ગુજરાત બારગામ
DATE :
PAGE :

• Remove Position (list_ref , position):

1. [check list is empty]
   if list_ref = NULL then
   cout<<e ("list is empty")
   return

2. [Find the position node]
   (i) temp ← list_ref
   (ii) prevTemp ← temp
   (iii) while (i < position)
      a. if (i = position - 1) then
         prevTemp ← temp
      b. temp ← temp (next)
      c. i ++

3. [Remove the node]
   (i) prevTemp (next) ← temp (next)
   (ii) x ← temp (info)
   (iii) free (temp)
   (iv) return

4. [Finished]
   Return

શ્રી ઉમિયા ગુજરાતી બારગ્રામ
DATE :
PAGE :

10) Write the various ways to insert a node into a singly linked list. Explain any one using algorithm

→ In singly linked list we insert a node in three ways

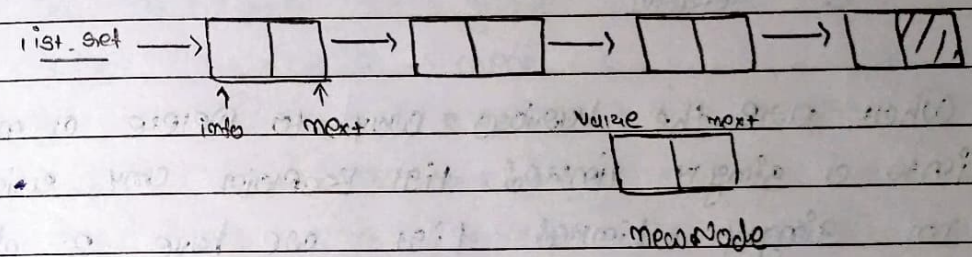
(i) Insert at start point of a list

(ii) Insert at End point of a list

(iii) Insert at given position in a list

→ Insert a node at start of list

- Let assume we have a list as following



- We have to add a new node to the start point

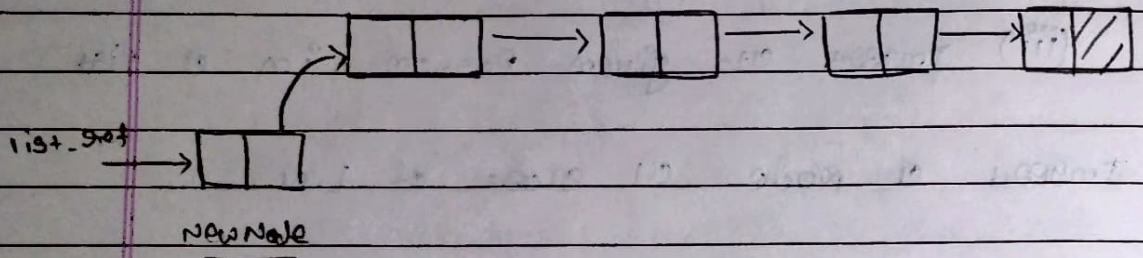
Algorithm :-

1. [check the list is empty or not]
   if (list_ref = NULL) then
   newNode (next) ← NULL
   list_ref ← newNode
   return.

- When there is no elements in List the starting and ending node be same so, we make newNode into End node

2. [Attach the node]
newNod (next) ← list_ref
list_ref ← newNode

- Simply, we set the pointer, newNode's next pointer points the first element of list and list_ref points new node to make it first node.



3. Return.

11) Check the working way to Delete a node into a singly linked list. Explain any using Algo.

→ In singly Linked List we have 3 different ways to remove a node.

(i) Remove from start point
(ii) Remove from End point
(iii) Remove from given Position.

- Remove from End point

Algorithm

1. [check the list is empty]
if (list_ref = NULL) then
write ("List is Empty")
return.

શ્રી ઉત્તર ગુજરાત બારગ્રામ
DATE :
PAGE :

2. [Find the last node]

(i) temp ← list_ref

(ii) nextTemp ← temp(next)

(iii) while (nextTemp(next) != NULL)

temp ← temp(next)

(iv) temp(next) ← NULL

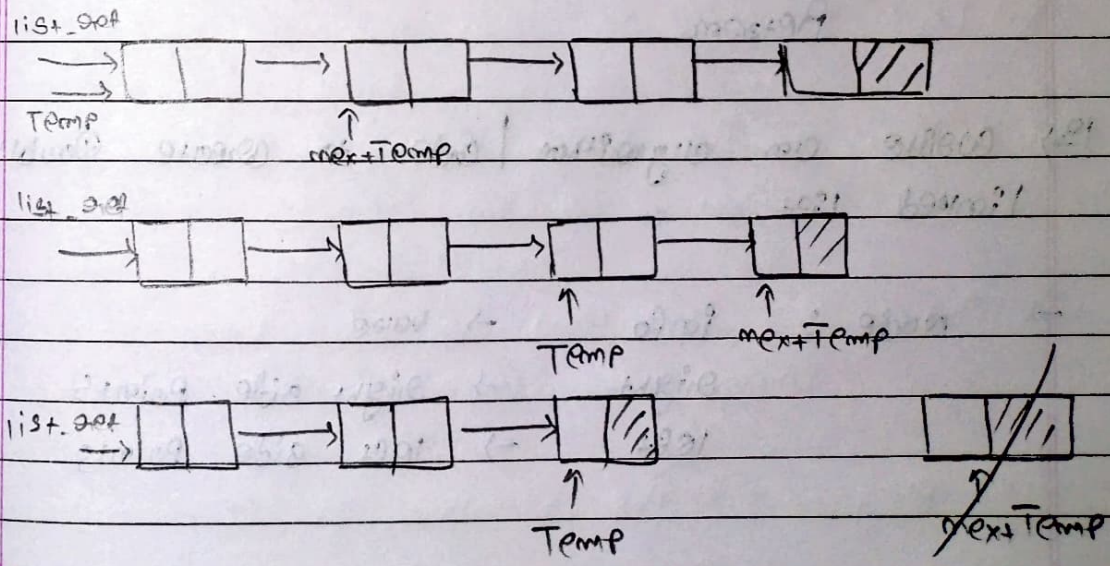
(v) x ← nextTemp(info)

3. [Free node]

free (temp nextTemp)

return x.

- We create a temp node same as list_ref and nextTemp which is the next node of a temp. The nextTemp is going ahead of temp node when nextTemp → next point is NULL means em. it's end node. We simply delete end node temp by temp(next) = NULL So, we have a free node nextTemp which had to be end node then remove it.



19) Write an algorithm / code to create circular Linked list

→ Algorithm to insert in circular Linked list

1. [check the list is empty]
   if (list_ref = NULL) then
   list_ref ← newnode.
   list_ref (next) ← list_ref
   return.

2. [Traverse the list to find last node]
   (i) temp ← list_ref
   (ii) while (temp (next) != NULL)
   temp ← temp (next).

3. [Insert the newnode]
   (i) temp (next) ← newNode
   (ii) newNode (next) ← list_ref

4. [Finished]
   Return

19) Write an algorithm / code to create Doubly Linked list.

→ node : info → value
right → right side points
left → left side points

→ Algorithm to insert in doubly linked list.

1. [Create node]
   (i) newnode = getnode ( )
   (ii) newnode (right) ← NULL
   (iii) newnode (info) ← value

2. [Traverse to the last node]
   ~~while (temp (right) !=)~~
   (i) temp ← list_ref
   (ii) while (temp (right) != NULL)
        temp ← temp (right)
   ~~(iii) temp (right) ← newnode~~

3. [Insert the node]
   (i) temp (right) ← newnode
   (ii) newnode (left) ← temp

4. [Finished]
   Return.