### +Queue:

A queue is another special kind of list, where items are inserted at one end called the rear and deleted at the other end called the front. Another name for a queue is a “FIFO” or “First-in-first-out”list.

The operations for a queue are analogues to those for a stack, the difference is that the insertions go at the end of the list, rather than the beginning. We shall use the following operations on queues:

* + - *enqueue*: which inserts an element at the end of the queue.
    - *dequeue*: which deletes an element at the start of the queue.

### Representation of Queue:

Let us consider a queue, which can hold maximum of five elements. Initially the queue is empty.

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

F R

Queue E mp t y

F RO NT = RE A R = 0

Now, insert 11 to the queue. Then queue status will be:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 |  |  |  |  |

F R

RE A R = RE A R + 1 = 1 FRO NT = 0

Next, insert 22 to the queue. Then the queue status is:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 22 |  |  |  |

F R

RE A R = RE A R + 1 = 2 FRO NT = 0

Again insert another element 33 to the queue. The status of the queue is:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 22 | 33 |  |  |

F R

RE A R = RE A R + 1 = 3 FRO NT = 0

Now, delete an element. The element deleted is the element at the front of the queue. So the status of the queue is:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 22 | 33 |  |  |

RE A R = 3

F RO NT = F R O NT + 1 = 1

F R

Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The queue status is as follows:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 33 |  |  |

F R

RE A R = 3

F RO NT = F R O NT + 1 = 2

Now, insert new elements 44 and 55 into the queue. The queue status is:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 33 | 44 | 55 |

F R

RE A R = 5 FRO NT = 2

Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 33 | 44 | 55 |

RE A R = 5 FRO NT = 2

F R

Now it is not possible to insert an element 66 even though there are two vacant positions in the linear queue. To over come this problem the elements of the queue are to be shifted towards the beginning of the queue so that it creates vacant position at the rear end. Then the FRONT and REAR are to be adjusted properly. The element 66 can be inserted at the rear end. After this operation, the queue status is as follows:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33 | 44 | 55 | 66 |  |

F R

RE A R = 4 FRO NT = 0

This difficulty can overcome if we treat queue position with index 0 as a position that comes after position with index 4 i.e., we treat the queue as a **circular queue.**

### Source code for Queue operations using array:

In order to create a queue we require a one dimensional array Q(1:n) and two variables *front* and *rear*. The conventions we shall adopt for these two variables are that *front* is always 1 less than the actual front of the queue and rear always points to the last element in the queue. Thus, front = rear if and only if there are no elements in the queue. The initial condition then is front = rear = 0. The various queue operations to perform creation, deletion and display the elements in a queue are asfollows:

* + - 1. insertQ(): inserts an element at the end of queueQ.
      2. deleteQ(): deletes the first element ofQ.
      3. displayQ(): displays the elements in thequeue.

# include <conio.h> # define MAX 6

int Q[MAX]; int front,rear;

voidinsertQ()

{

int data;

if(rear == MAX)

{

}

else

{

}

}

printf("\n Linear Queue is full"); return;

printf("\n Enter data: "); scanf("%d", &data); Q[rear] = data;

rear++;

printf("\n Data Inserted in the Queue ");

void deleteQ()

{

if(rear == front)

{

}

else

{

}

}

printf("\n\n Queue is Empty.."); return;

printf("\n Deleted element from Queue is %d", Q[front]); front++;

void displayQ()

{

int i;

if(front == rear)

{

}

else

{

printf("\n\n\t Queue is Empty"); return;

printf("\n Elements in Queue are: "); for(i = front; i < rear; i++)

{

printf("%d\t", Q[i]);

}

}

}

int menu()

{

int ch; clrscr();

printf("\n \tQueue operations using ARRAY.."); printf("\n-----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Insert "); printf("\n 2. Delete "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter your choice: "); scanf("%d", &ch);

return ch;

}

void main()

{

int ch; do

{

ch = menu(); switch(ch)

{

case1:

case2:

case3:

case4:

}

insertQ(); break;

deleteQ(); break;

displayQ(); break;

return;

getch();

} while(1);

}

### Linked List Implementation of Queue:

We can represent a queue as a linked list. In a queue data is deleted from the front end and inserted at the rear end. We can perform similar operations on the two ends of a list. We use two pointers *front* and *rear* for our linked queue implementation.

The linked queue looks as shown in figure 4.4:

**front**

100

**rear**

400

10 200

100

20 300

200

30 400 40 **X**

300 400

Figure 4.4. Linked Queue representation

### Source code for queue operations using linkedlist:

# include <stdlib.h> # include <conio.h>

struct queue

{

int data;

struct queue \*next;

};

typedef struct queue node; node \*front =NULL;

node \*rear =NULL;

node\* getnode()

{

node \*temp;

temp = (node \*) malloc(sizeof(node)) ; printf("\n Enter data ");

scanf("%d", &temp -> data); temp -> next = NULL;

return temp;

}

void insertQ()

{

node \*newnode; newnode = getnode(); if(newnode == NULL)

{

printf("\n Queue Full"); return;

}

if(front == NULL)

{

}

else

{

}

front = newnode; rear =newnode;

rear -> next = newnode; rear =newnode;

printf("\n\n\t Data Inserted into the Queue..");

}

void deleteQ()

{

node \*temp; if(front == NULL)

{

printf("\n\n\t Empty Queue.."); return;

}

temp = front;

front = front -> next;

printf("\n\n\t Deleted element from queue is %d ", temp -> data); free(temp);

}

void displayQ()

{

node \*temp; if(front == NULL)

{

}

else

{

printf("\n\n\t\t Empty Queue ");

temp = front;

printf("\n\n\n\t\t Elements in the Queue are: "); while(temp != NULL )

{

printf("%d \n", temp -> data); temp = temp -> next;

}

}

}

char menu()

{

char ch; clrscr();

printf("\n \t..Queue operations using pointers.. "); printf("\n\t-----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Insert "); printf("\n 2. Delete "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter your choice: "); ch = getche();

return ch;

}

void main()

{

char ch; do

{

ch = menu(); switch(ch)

{

case '1' :

insertQ(); break;

case '2' :

deleteQ(); break;

case '3' :

displayQ(); break;

case '4':

return;

}

getch();

} while(ch != '4');

}

### Applications ofQueue:

1. It is used to schedule the jobs to be processed by theCPU.
2. When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO)basis.
3. Breadth first search uses a queue data structure to find an element from a graph.

### CircularQueue:

A more efficient queue representation is obtained by regarding the array Q[MAX] as circular. Any number of items could be placed on the queue. This implementation of a queue is called a circular queue because it uses its storage array as if it were a circle instead of a linear list.

There are two problems associated with linear queue. They are:

* Time consuming: linear time to be spent in shifting the elements to the beginning of the queue.
* Signaling queue full: even if the queue is having vacantposition.

For example, let us consider a linear queue status as follows:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 33 | 44 | 55 |

F R

RE A R = 5 FRO NT = 2

Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 33 | 44 | 55 |

F R

RE A R = 5 FRO NT = 2

This difficulty can be overcome if we treat queue position with index zero as a position that comes after position with index four then we treat the queue as a **circular queue.**

In circular queue if we reach the end for inserting elements to it, it is possible to insert new elements if the slots at the beginning of the circular queue are empty.

### Representation of CircularQueue:

Let us consider a circular queue, which can hold maximum (MAX) of six elements. Initially the queue is empty.

F R

1 QueueEmpty

5

0

3

2

4 MA X =6

F RO NT = RE A R = 0 CO U NT = 0

Circ u lar Q u e u e

Now, insert 11 to the circular queue. Then circular queue status will be:

F

R

5

0

11

1

3

2

FRO NT = 0

4 RE A R = ( RE A R + 1) % 6 =1

CO U NT = 1

Circ u lar Q u e u e

Insert new elements 22, 33, 44 and 55 into the circular queue. The circular queue status is:

4 1 FRONT =0

F

R

5

0

11

55

22

44

33

3

2

REAR = (REAR + 1) % 6 = 5

COUNT = 5

Circular Queue

Now, delete an element. The element deleted is the element at the front of the circular queue. So, 11 is deleted. The circular queue status is as follows:

R

F

5

0

55

22 1

44

33

3

2

F RO NT = (F R O NT + 1) % 6 = 1

4 RE A R =5

CO U NT = CO U NT - 1 = 4

Circ u lar Q u e u e

Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The circular queue status is as follows:

R

F RO NT = (F R O NT + 1) % 6 = 2

5

0

55

1

44

33

3

2

4 RE A R =5

CO U NT = CO U NT - 1 = 3

F

Circ u lar Q u e u e

Again, insert another element 66 to the circular queue. The status of the circular queue is:

R

4 FRO NT =2

5

0

66

55

1

44

33

3

2

RE A R = ( RE A R + 1) % 6 = 0 C O U NT = C O U NT + 1 =4

F

Circ u lar Q u e u e

Now, insert new elements 77 and 88 into the circular queue. The circular queue status is:

5 0

66 77

4 55

3

88 1

44 33

2 R

F

F RO NT = 2, RE A R = 2 RE A R = RE A R % 6 = 2 CO U NT = 6

Circ u lar Q u e u e

Now, if we insert an element to the circular queue, as COUNT = MAX we cannot add the element to circular queue. So, the circular queue is*full*.

### Source code for Circular Queue operations, usingarray:

# include <stdio.h> # include <conio.h> # define MAX 6

int CQ[MAX];

int front = 0; int rear = 0; int count = 0;

void insertCQ()

{

int data;

if(count == MAX)

{

}

else

{

}

}

printf("\n Circular Queue is Full");

printf("\n Enter data: "); scanf("%d", &data); CQ[rear] = data;

rear = (rear + 1) % MAX; count ++;

printf("\n Data Inserted in the Circular Queue ");

void deleteCQ()

{

if(count == 0)

{

}

else

{

}

}

printf("\n\nCircular Queue is Empty..");

printf("\n Deleted element from Circular Queue is %d ", CQ[front]); front = (front + 1) % MAX;

count --;

void displayCQ()

{

int i, j; if(count ==0)

{

}

else

{

printf("\n\n\t Circular Queue is Empty ");

printf("\n Elements in Circular Queue are: "); j = count;

for(i = front; j != 0; j--)

{

printf("%d\t", CQ[i]); i = (i + 1) % MAX;

}

}

}

int menu()

{

int ch; clrscr();

printf("\n \t Circular Queue Operations using ARRAY.."); printf("\n-----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Insert "); printf("\n 2. Delete "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter Your Choice: "); scanf("%d", &ch);

return ch;

}

void main()

{

int ch; do

{

ch = menu(); switch(ch)

{

case1:

case2:

case3:

case 4: default:

}

insertCQ(); break;

deleteCQ(); break;

displayCQ(); break;

return;

printf("\n Invalid Choice ");

getch();

} while(1);

}

### Deque:

In the preceding section we saw that a queue in which we insert items at one end and from which we remove items at the other end. In this section we examine an extension of the queue, which provides a means to insert and remove items at both ends of the queue. This data structure is a *deque*. The word *deque* is an acronym derived from *double-ended queue*. Figure 4.5 shows the representation of a deque.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 36 | 16 | 56 | 62 | 19 |

Figure 4.5. Representation of a deque.

Deletion

Insertion

Insertion

Deletion

front rear

A deque provides four operations. Figure 4.6 shows the basic operations on a deque.

* + - enqueue\_front: insert an element atfront.
    - dequeue\_front: delete an element atfront.
    - enqueue\_rear: insert element atrear.
    - dequeue\_rear: delete element atrear.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 11 | 22 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ) | 33 | 11 | 22 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 33 | 11 | 22 | 44 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5) |  | 11 | 22 | de |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 11 | 22 | 44 |  |

Figure 4.6. Basic operations on deque

enqueue\_front(33

enqueue\_rear(44)

dequeue\_front(33)

enqueue\_front(5

queue\_rear(44)

22

11

55

There are two variations of deque. They are:

* + - Input restricted deque(IRD)
    - Output restricted deque(ORD)

An Input restricted deque is a deque, which allows insertions at one end but allows deletions at both ends of the list.

An output restricted deque is a deque, which allows deletions at one end but allows insertions at both ends of the list.

### PriorityQueue:

A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processed according to the order in which they were added to the queue.

A prototype of a priority queue is time sharing system: programs of high priority are processed first, and programs with the same priority form a standard queue. An efficient implementation for the Priority Queue is to use heap, which in turn can be used for sorting purpose called heapsort.

### Exercises

1. What is a linear data structure? Give two examples of linear datastructures.
2. Is it possible to have two designs for the same data structure that provide the same functionality but are implementeddifferently?
3. What is the difference between the logical representation of a data structure and the physicalrepresentation?
4. Transform the following infix expressions to reverse polishnotation:
   1. A B \* C – D + E / F / (G +H)

b) ((A + B) \* C – (D – E)) (F + G)

c) A – B / (C \* D E)

d) (a + b c d) \* (e + f / d)) f) 3 – 6 \* 7 + 2 / 4 \* 5 – 8

g) (A – B) / ((D + E) \* F)

h) ((A + B) / D) ((E – F) \* G)

1. Evaluate the following postfix expressions: a) P1: 5, 3, +, 2, \*, 6, 9, 7, -, /,-

b) P2: 3, 5, +, 6, 4, -, \*, 4, 1, -, 2, ,+

c) P3 : 3, 1, +, 2, , 7, 4, -, 2, \*, +, 5,-

1. Consider the usual algorithm to convert an infix expression to a postfix expression. Suppose that you have read 10 input characters during a conversion and that the stack now contains thesesymbols:

bottom

+ (

\*

Now, suppose that you read and process the 11th symbol of the input. Draw the stack for the case where the 11th symbol is:

1. Anumber:
2. A leftparenthesis:
3. A rightparenthesis:
4. A minussign:
5. A divisionsign:
6. Write a program using stack for parenthesis matching. Explain what modifications would be needed to make the parenthesis matching algorithm check expressions with different kinds of parentheses such as (), [] and{}'s.
7. Evaluate the following prefix expressions: a) + \* 2 + / 14 2 51

b) - \* 6 3 – 4 1

c) + + 2 6 + - 13 2 4

1. Convert the following infix expressions to prefix notation: a) ((A + 2) \* (B + 4))-1

b) Z – ((((X + 1) \* 2) – 5) / Y)

c) ((C \* 2) + 1) / (A + B)

d) ((A + B) \* C – (D - E)) (F + G)

1. A – B / (C \* D E)
2. Write a “C” function to copy one stack to anotherassuming
   1. The stack is implemented usingarray.
   2. The stack is implemented using linkedlist.
3. Write an algorithm to construct a fully parenthesized infix expression from its postfix equivalent. Write a “C” function for youralgorithm.
4. How can one convert a postfix expression to its prefix equivalent andvice-versa?
5. A double-ended queue (deque) is a linear list where additions and deletions can be performed at either end. Represent a deque using an array to store the elements of the list and write the “C” functions for additions anddeletions.
6. In a circular queue represented by an array, how can one specify the number of elements in the queue in terms of “front”, “rear” and MAX-QUEUE-SIZE? Write a “C” function to delete the K-th element from the “front” of a circularqueue.
7. Can a queue be represented by a circular linked list with only one pointer pointing to the tail of the queue? Write “C” functions for the “add” and “delete” operations on such aqueue
8. Write a “C” function to test whether a string of opening and closing parenthesis is well formed ornot.
9. Represent N queues in a single one-dimensional array. Write functions for “add” and “delete” operations on the ithqueue
10. Represent a stack and queue in a single one-dimensional array. Write functions for “push”, “pop” operations on the stack and “add”, “delete” functions on the queue.

### Multiple Choice Questions

1. Which among the following is a lineardatastructure: [ D ]
   1. Queue
   2. Stack
   3. LinkedList
   4. all theabove
2. Which among the following is a Dynamicdatastructure: [ A]
   1. Double LinkedList
   2. Queue
   3. Stack
   4. all theabove
3. Stack isreferred as: [ A]
   1. Last in first outlist
   2. First in first outlist
   3. both A andB
   4. none of the above
4. A stack is a data structure in which all insertions and deletions of entries are madeat:

[ A ]

* 1. Oneend
  2. In themiddle
  3. Both theends
  4. At anyposition

1. A queue is a data structure in which all insertions and deletions are made respectivelyat:

[ A ]

* 1. rear andfront
  2. front andfront
  3. front andrear
  4. rear andrear

1. Transform the following infix expression to postfix form: (A + B) \* (C – D) /E

[ D]

* 1. A B \* C + D /-
  2. A B C \* C D / -+
  3. A B + C D \* - /E
  4. A B + C D - \* E/

1. Transform the following infix expression to postfix form: A - B / (C \*D)

[ B]

* 1. A B \* C D -/
  2. A B C D \* /-
  3. / - D C \* BA
  4. - / \* A B CD

1. Evaluate the following prefix expression: \* - + 4 3 5 / + 243 [ A]
   1. 4
   2. 8
   3. 1
   4. none of the above
2. Evaluate the following postfix expression: 1 4 18 6 / 3 + + 5/+ [ C]
   1. 8
   2. 2
   3. 3
   4. none of the above
3. Transform the following infix expression to prefix form: ((C \* 2) + 1) / (A +B)

[ B]

A. A B + 1 2 C \* +/

B. / + \* C 2 1 + AB

C. / \* + 1 2 C A B +

D. none of the above

1. Transform the following infix expression to prefix form: Z – ((((X + 1) \* 2) – 5) /Y)

[ D]

A. / - \* + X 1 2 5Y

B. Y 5 2 1 X + \* -/

C. / \* - + X 1 2 5 Y

D. none of the above

1. Queue is alsoknownas: [ B]
   1. Last in first outlist
   2. First in first outlist
   3. both A andB
   4. none of the above
2. One difference between a queue and astackis: [ C]
   1. Queues require dynamic memory, but stacks donot.
   2. Stacks require dynamic memory, but queues donot.
   3. Queues use two ends of the structure; stacks use onlyone.
   4. Stacks use two ends of the structure, queues use onlyone.
3. If the characters 'D', 'C', 'B', 'A' are placed in a queue (in that order), and then removed one at a time, in what order will they beremoved?

[ D]

* 1. ABCD
  2. ABDC
  3. DCAB
  4. DCBA

1. Suppose we have a circular array implementation of the queue class, with ten items in the queue stored at data[2] through data[11]. The CAPACITY is 42. Where does the push member function place the new entry in thearray?

[ D]

* 1. data[1]
  2. data[2]
  3. data[11]
  4. data[12]

1. Consider the implementation of the queue using a circular array. What goes wrong if we try to keep all the items at the front of a partially-filled array (so that data[0] is always thefront).
   1. The constructor would require lineartime.
   2. The get\_front function would require lineartime.
   3. The insert function would require lineartime.
   4. The is\_empty function would require lineartime.
2. In the linked list implementation of the queue class, where does the push member function place the new entry on the linkedlist?
   1. At thehead
   2. At thetail
   3. After all other entries that are greater than the newentry.
   4. After all other entries that are smaller than the newentry.
3. In the circular array version of the queue class (with a fixed-sized array), which operations require linear time for their worst-casebehavior?

[ B ]

[ A ]

[ ]

* 1. front
  2. push
  3. empty
  4. None ofthese.

1. In the linked-list version of the queue class, whichoperationsrequire [ ] linear time for their worst-casebehavior?
   1. front
   2. push
   3. empty
   4. None of theseoperations.
2. To implement the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during an insertion into a NONEMPTYqueue?

[ B]

* 1. Neitherchanges
  2. Only front\_ptrchanges.
  3. Only rear\_ptrchanges.
  4. Both change.

1. To implement the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during an insertion into an EMPTYqueue?

[ D]

* 1. Neitherchanges
  2. Only front\_ptrchanges.
  3. Only rear\_ptrchanges.
  4. Both change.

1. Suppose top is called on a priority queue that has exactly two entries with equal priority. How is the return value of topselected?
   1. The implementation gets to choose eitherone.
   2. The one which was insertedfirst.
   3. The one which was inserted mostrecently.
   4. This can never happen (violates theprecondition)

[ B]

1. Entries in a stack are "ordered". What is the meaning ofthisstatement? [ D]
   1. A collection of stacks can besorted.
   2. Stack entries may be compared with the '<'operation.
   3. The entries must be stored in a linkedlist.
   4. There is a first entry, a second entry, and soon.
2. The operation for adding an entry to a stack istraditionallycalled: [ D ]
   1. add
   2. append
   3. insert
   4. push

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. The operation for removing an entry from a stack is traditionallycalled:    1. delete C.pop    2. peek D.remove 2. Which of the following stack operations could result in stackunderflow?    1. is\_empty C.push    2. pop D. Two or more of the aboveanswers | | [  [ | C A | ]  ] |
| 1. Which of the following applications may use astack?    1. A parentheses balancingprogram.    2. Keeping track of local variables at runtime.    3. Syntax analyzer for acompiler.    4. All of theabove. | | [ | D | ] |
| 28. Here is an infix expression: 4 + 3 \* (6 \* 3 - 12). Suppose that we are using the usual stack algorithm to convert the expression from infix to postfix notation. What is the maximum number of symbols that will appear on the stack AT ONE TIME during the conversion ofthis  expression? | | [ | D | ] |
| A. 1 | C. 3 | | | |
| B. 2 | D. 4 | | | |
| 1. What is the value of the postfix expression 6 3 2 4 + -\*    1. Something between -15 and-100    2. Something between -5 and-15    3. Something between 5 and-5    4. Something between 5 and15    5. Something between 15 and100 | | [ | A | ] |
| 30. If the expression ((2 + 3) \* 4 + 5 \* (6 + 7) \* 8) + 9 is evaluated with\*  having precedence over +, then the value obtained is same as the value of which of the following prefix expressions? | | [ | A | ] |
| A. + + \* + 2 3 4 \* \* 5 + 6 7 8 9 | C. \* + + + 2 3 4 \* \* 5 + 6 7 8 9 | | | |
| B. + \* + + 2 3 4 \* \* 5 + 6 7 8 9 | D. + \* + + 2 3 4 + + 5 \* 6 7 8 9 | | | |

1. Evaluate the following prefixexpression:

+ \* 2 + / 14 2 5 1

[ B]

* 1. 50
  2. 25
  3. 40
  4. 15

1. Parenthesis are never needed prefix orpostfixexpression: [ A ]
   1. True
   2. False
   3. Cannot beexpected
   4. None of theabove
2. A postfix expression is merely the reverse of theprefixexpression: [ B ]
   1. True
   2. False
   3. Cannot beexpected
   4. None of theabove
3. Which among the following data structure may give overflow error, even though the current number of elements in it, is less than itssize:

[ A ]

* 1. SimpleQueue
  2. CircularQueue
  3. Stack
  4. None of theabove

1. Which among the following types of expressions does not require precedence rules forevaluation:
   1. Fully parenthesized infixexpression
   2. Prefixexpression
   3. both A andB
   4. none of the above

[ C]

1. Conversion of infix arithmetic expression to postfixexpressionuses: [D ]
   1. Stack
   2. circularqueue
   3. linkedlist
   4. Queue