The Queue Data Structure in C++

By Eric Suh

http://www.cprogramming.com/tutorial/computersciencetheory/queue.html

Queues are data structures that, like the stack, have restrictions on where you can add and remove elements. To understand a queue, think of a cafeteria line: the person at the front is served first, and people are added to the line at the back.

Thus, the first person in line is served first, and the last person is served last. This can be abbreviated to **F**irst **I**n, **F**irst **O**ut (**FIFO**).

The cafeteria line is one type of queue. *Queues are often used* in programming networks, operating systems, and other situations in which *many different processes must share resources* such as CPU time.

One bit of terminology: the addition of an element to a queue is known as an **enqueue**, and removing an element from the queue is known as a **dequeue**.

Although the concept may be simple, **programming a queue is not as simple as programming a stack**.

Let's go back to the example of the cafeteria line. Let's say one person leaves the line. Then what? Everyone in line must step forward, left?

Now, imagine if only one person could move at a time. So, the second person steps forward to fill the space left by the first person, and then the third person steps forwards to fill the space left by the second person, and so on.

Now imagine that **no one can leave or be added to the line until everyone has stepped forward**. You can see the line will move **very slowly**. It is not difficult to program a queue that works, but it is *quite* a proposition to make a queue that is ***fast***!!

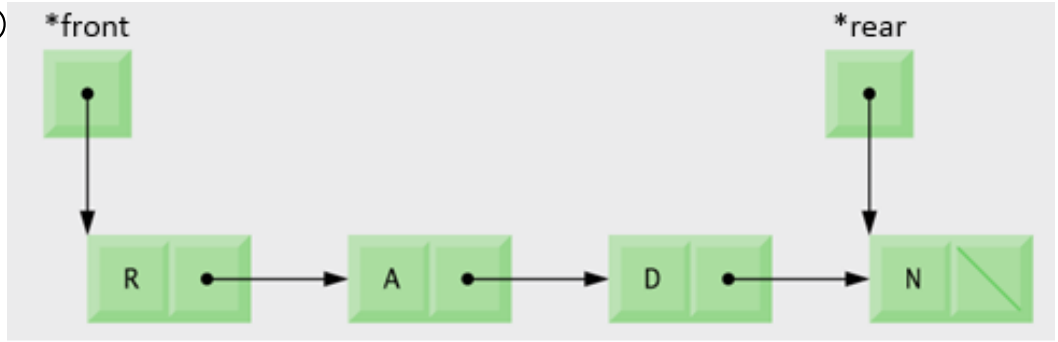
There are a couple of basic ways to implement a queue. The first is to just make an array and shift all the elements to accommodate enqueues and dequeues. This, as we said above, is slow. Click for an example.

The other way to implement a Queue is using Data Structure. Click for an example.

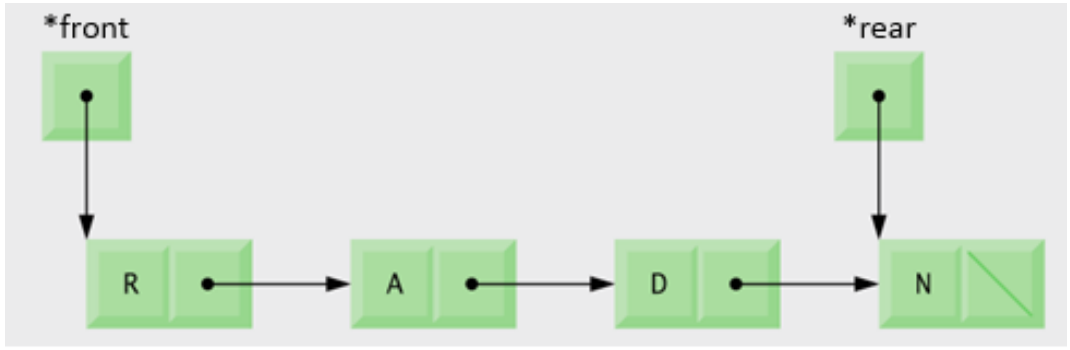
In the following section, we shall explore details of a program employing a queue data structure using linked list. The program is divided into 5 sections

Section 1	Program Description and declaration of prototypes
Section 2	Programs main function
Section 3	void Insert(int info)
Section 4	int Delete()
Section 5	void Display()

```
/* Implementation of Queue using Linked List.
 * The program has four options, which are:
 *
 * 1-Insert a Node into the Queue,
 * 2-Delete a Node from the Queue,
 * 3-Display all Nodes in the Queue,
 * 4-Exit
 */
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   int data;
   struct node *link;
} NODE;  // NODE IS AN ALIAS (shorthand writing) of struct node.
         // In other words, NODE is a new data type, just as int

// THESE ARE FUNCTION PROTOTYPES TO
// INSERT, DELETE & DISPLAY NODES
void Insert(int);
int Delete();
void Display();

NODE *front, *rear; //*front and *rear are 2 variables of type NODE
```

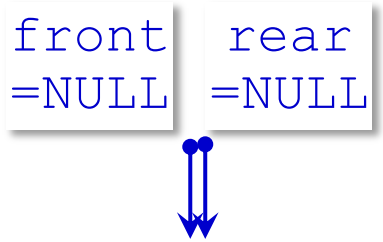


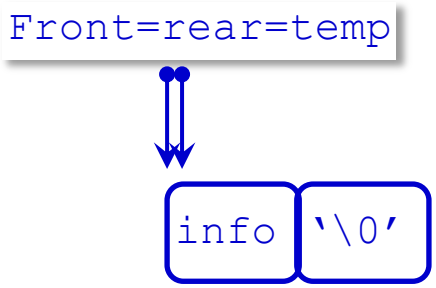
```
void main()
{
    int option, data;
    front = rear = NULL;

    do
    {
        printf("Linked List Implementation of QUEUE Operations \n");
        printf("\nChoose Your Option \n");
        printf("\n1-Insert, \n2-Delete, \n3-Display,\n4-Exit\n");
        printf("\n\nYour option: ");
        scanf("%d", &option);

        switch (option)
        {
        case 1:
            printf("\n\nRead the data to be Inserted ?");
            scanf("%d", &data);
            Insert(data);
            break;
        case 2:
            data = Delete();
            if (data != -1)
              printf("Deleted Node(From Front)with Data: %d\n", data);
            break;
```

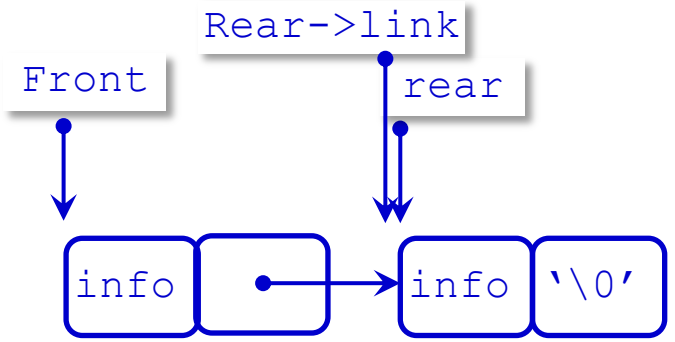


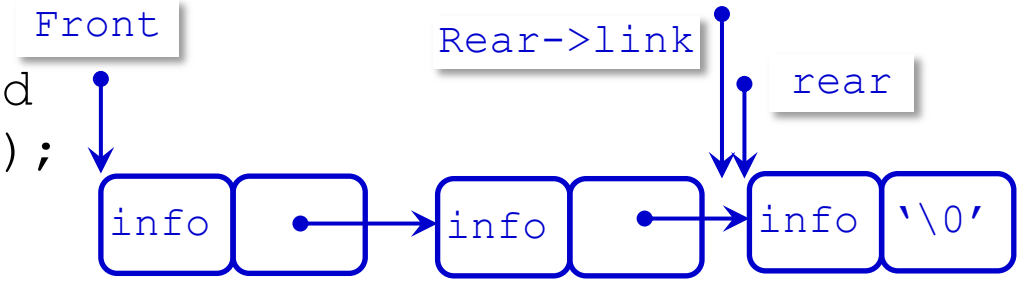
```
      case 3:
         printf("Linked List Implementation of Queue: Status:\n");
         Display();
         break;
      case 4:
         printf("\nTerminating\n");
         break;
      default:
         printf("\n\nInvalid Option!!! Try Again!! \n\n");
         break;
      }

      printf("\n\nPress a Key to Continue...");
      getchar();
   } while (option != 4);
}
```

```
void Insert(int info)
{
    NODE *temp;
    temp = (NODE *) malloc(sizeof(NODE));

    if (temp == NULL)
       printf("Out of Memory!! Overflow!!!");
    else
    {
       temp->data = info;
       temp->link = NULL;

       if (front == NULL)
       {
          front = rear = temp;
       } /* First Node? */
       else
       {
          rear->link = temp;
          rear = temp;
       } /* Insert End */
       printf(" Node has been inserted
              at End Successfully !!");
    }
}
```

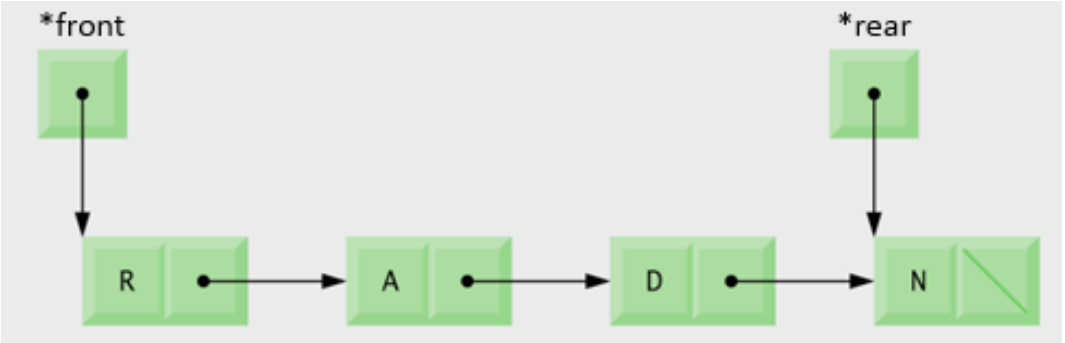


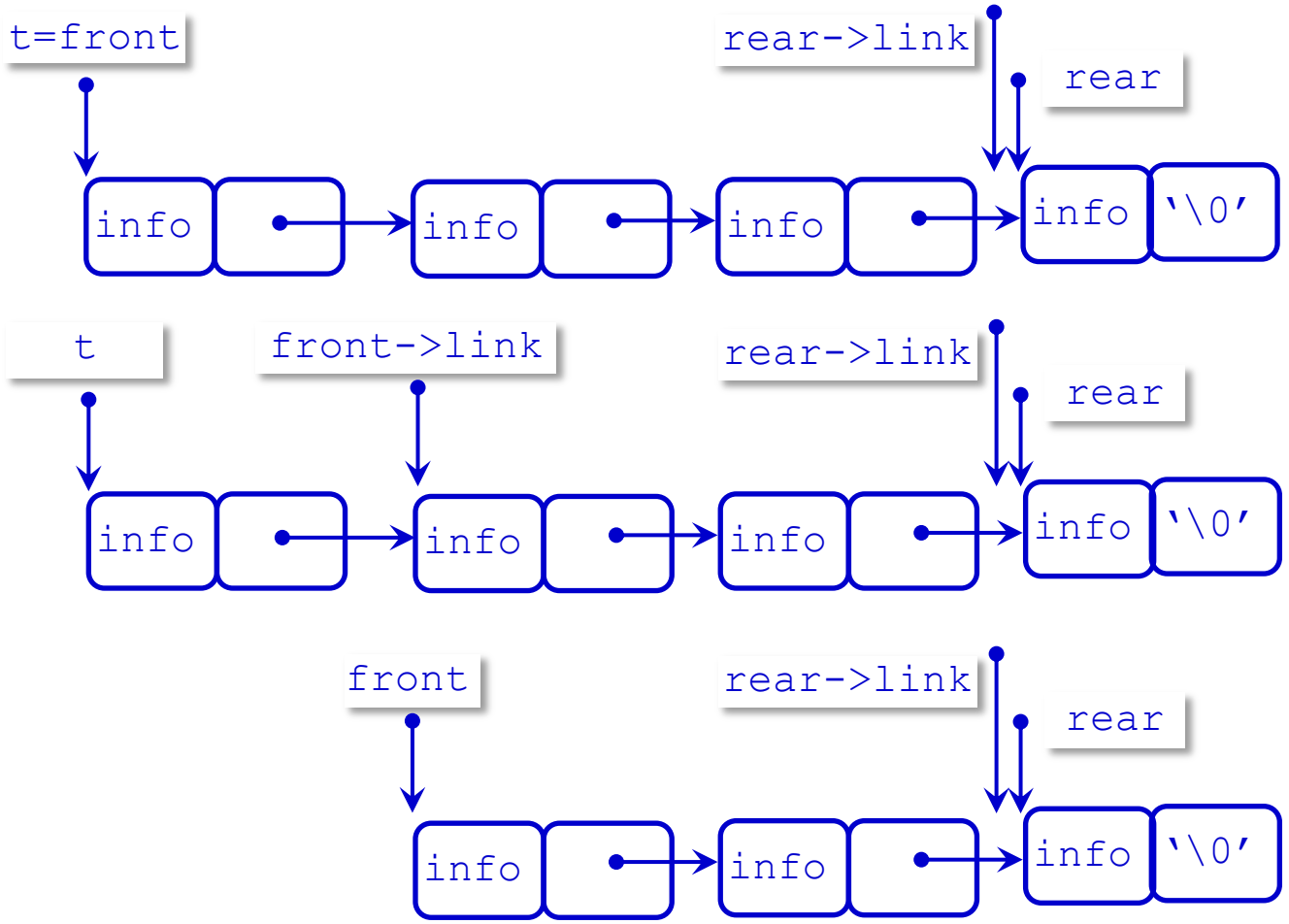






```
int Delete()
{
    int info;
    NODE *t;

    if (front == NULL)
    {
        printf(" Underflow!!!");
        return -1;
    }
    else
    {
        t = front;
        info = front->data;

        if (front == rear)
            rear = NULL;
        front = front->link;
        t->link = NULL;
        free(t);
        return (info);
    }
}
```





```
void Display()
{
   NODE *t;

   if (front == NULL)
      printf("Empty Queue\n");
   else
   {
      t = front;
      printf("Front->");

      while (t)
      {
         printf("[%d]->", t->data);
         t = t->link;
      }
      printf("Rear\n");
   }
}
```

Another Example of Queue Using Structures

```
/* Full Program: queue.c
 * This example shows that AddItem and RemoveItem functions
 * returns the pointer of added or removed node
 *
 * A queue is basically a special case of a linked list where one
 * data element joins the list at the left end and leaves in a
 * ordered fashion at the other end.
 *
 * Demo of dynamic data structures in C
 */
#include <stdio.h>

#define FALSE 0
#define NULL 0

typedef struct {
    int     dataitem;
    struct listelement *link;
} listelement;

void Menu (int *choice);

listelement * AddItem (listelement * listpointer, int data);
```

```
listelement * RemoveItem (listelement * listpointer);

void PrintQueue (listelement * listpointer);

void ClearQueue (listelement * listpointer);

main ()
{
    listelement listmember, *listpointer;
    int data, choice;

    listpointer = NULL;

    do
   {
      Menu (&choice);

      switch (choice)
      {
         case 1:
         printf ("Enter data item value to add  ");
         scanf ("%d", &data);

         listpointer = AddItem (listpointer, data);
```

```
         break;

         case 2:
            if (listpointer == NULL)
               printf ("Queue empty!\n");
            else
               listpointer = RemoveItem (listpointer);
         break;

         case 3:
            PrintQueue (listpointer);
         break;

         case 4:
         break;

         default:
            printf ("Invalid menu choice - try again\n");
         break;
      }
    } while (choice != 4);

   ClearQueue (listpointer);

} /* end of main */
```

```
void Menu (int *choice)
{
    char local;

    printf ("\nEnter\t1 to add item,\n\t2 to remove item\n\ \t3 to
print queue\n\t4 to quit\n");
    do
   {
      local = getchar ();

      if ((isdigit (local) == FALSE) && (local != '\n'))
      {
         printf ("\nyou must enter an integer.\n");
         printf ("Enter 1 to add, 2 to remove, 3 to print, 4 to
quit\n");
      }
    } while (isdigit ((unsigned char) local) == FALSE);

   *choice = (int) local - '0';
}

listelement * AddItem (listelement * listpointer, int data)
{
    listelement * lp = listpointer;
```

```
    if (listpointer != NULL)
   {
      while (listpointer -> link != NULL)
         listpointer = listpointer -> link;

      listpointer -> link = (struct listelement  *) malloc (sizeof
(listelement));
      listpointer = listpointer -> link;
      listpointer -> link = NULL;
      listpointer -> dataitem = data;

      return lp;
    }
    else
   {
      listpointer = (struct listelement  *) malloc (sizeof
(listelement));
      listpointer -> link = NULL;
      listpointer -> dataitem = data;

      return listpointer;
    }
}
```

```
listelement * RemoveItem (listelement * listpointer)
{
    listelement * tempp;
    printf ("Element removed is %d\n", listpointer -> dataitem);
    tempp = listpointer -> link;
    free (listpointer);
    return tempp;
}

void PrintQueue (listelement * listpointer)
{
    if (listpointer == NULL)
      printf ("queue is empty!\n");
    else
      while (listpointer != NULL)
      {
         printf ("%d\t", listpointer -> dataitem);
         listpointer = listpointer -> link;
      }

      printf ("\n");
}

void ClearQueue (listelement * listpointer)
{
```

```
    while (listpointer != NULL)
   {
      listpointer = RemoveItem (listpointer);
    }
}
```