QUEUE WITH ARRAYS

```
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>
using namespace std;

template <class DT>
class Queue
{
      public:
      Queue(DT max=0);
      bool IsEmpty();
      bool IsFull();
      void Put(DT i);
      DT Get();
private:
      DT front;
      DT rear;
      DT size;
      DT *arr;
};
#endif

```

CPP

```
#include"queue.h"

template <class DT>
Queue<DT>::Queue(DT max=0)
{
      size=max;
      arr=new DT[size];
      front=-1;
      rear=-1;
      for(int i=0; i<size; i++)
      {
                arr[i]=0;
      }
}

template <class DT>
bool Queue<DT>::IsEmpty()
{
      if(front==-1 && rear==-1)
                return true;
      else
                return false;
}

template <class DT>
bool Queue<DT>::IsFull()
{
      if(front==(rear+1)%size)
                return true;
      else
                return false;
}

```

```
34. template <class DT>
35. void Queue<DT>::Put(DT item)
36. {
37.       if (IsFull())
38.       {
39.                 return;
40.       }
41.       if(IsEmpty())
42.       {
43.                 front=0;
44.                 rear=0;
45.       }
46.       else
47.       {
48.                 rear=(rear+1)%size;
49.       }
50.       arr[rear]=item;
51. }
52.
53. template <class DT>
54. DT Queue<DT>::Get()
55. {
56.       if(IsEmpty())
57.       {
58.                 return 0;
59.       }
60.       DT temp= arr[front];
61.       if(front==rear)
62.       {
63.                 front=rear=-1;
64.       }
65.       else
66.       {
67.                 front=(front+1)%size;
68.       }
69.       return temp;
70. }
71.
```

A queue is a First-In-First-Out (FIFO) data structure where elements are inserted at the rear and removed from the front.

Line 48: If the queue is not empty, the rear index is incremented. The % size ensures that if the rear reaches the end of the array, it wraps around to the beginning (circular behavior).

```
1. #include"queue.h"
2. #include"queue.cpp"
3. using namespace std;
4. int main()
5. {
6. Queue<int> *q =new Queue<int>(3);
7. if(q->IsEmpty())
8. cout<<"Queue is currently empty"<<endl;
9. q->Put(1);
10. q->Put(2);
11. q->Put(3);
12. while (!q->IsEmpty())
13. {
14. int value=q->Get();
15. cout<<value<<endl;
16. }
17. system("pause");
18. return 0;
19. }
20.
```

QUEUE USING LINKED LIST

```
#ifndef QUEUE_H
#define QUEUE_H
#include<iostream>
using namespace std;
template<class DT>
class Node
{
private:
	int data;
	Node<DT> *link;
public:
	Node(int d=0)
	{
		data=d;
		link=NULL;
	}
	int getdata()
	{
		return data;
	}
	Node<DT> *getlink()
	{
		return link;
	}
	void setdata(int d)
	{
		data=d;
	}
	void setnext(Node<DT> *l)
	{
		link=l;
	}
};
template<class DT>
class List
{
private:
	Node<DT> *first;
public:
	List();
	void Insert(Node<DT>* prev,Node<DT>* newnode);
	void Delete(Node<DT>* node);
};
template<class DT>
class Queue
{
private:
	List<DT> *l;
	Node<DT> *Head;
	Node<DT> *Tail;
public:
	Queue();
	bool Empty();
	void Put(int element);
	int Get();
};
#endif

```

```
1.   CPP

   #include"Queue.h"
2.
3. template<class DT>
4. List<DT>::List()
5.       {
6.                 first = NULL;
7.       }
8.
9. template<class DT>
10. void List<DT>::Insert(Node<DT>* prev,Node<DT>* newnode)
11.       {
12.                if (first == NULL)
13.                {
14.                          first = newnode;
15.                }
16.                else
17.                {
18.                          newnode->setnext(prev->getlink());
19.                          prev->setnext(newnode);
20.                }
21.       }
22.
23. template<class DT>
24. void List<DT>::Delete(Node<DT>* node)
25.       {
26.                delete node;
27.       }
28.

29. template<class DT>
30. Queue<DT>::Queue()
31.       {
32.                Head=NULL;
33.                Tail=NULL;
34.                l=new List<DT>();
35.       }
36.

37. template<class DT>
38. bool Queue<DT>::Empty()
39.       {
40.                return (Head==NULL && Tail==NULL);
41.       }
42.

43. template<class DT>
44. void Queue<DT>::Put(int element)
45.       {
46.                Node<DT> *N=new Node<DT>();
47.                N->setdata(element);
48.                l->Insert(Tail,N);
49.                if (Empty())
50.                {
51.                          Head=N;
52.                          Tail=N;
53.                }
54.                else
55.                {
56.                          Tail=N;
57.                }
58.       }
59.
```

->If the list is empty, new node becomes the first node.
->If the list is not empty, the new node is inserted after the node prev.

```
template<class DT>
void List<DT>::Delete(Node<DT>* prev,
Node<DT>* node)
{
   if (prev != NULL)
   {
      prev->setnext(node->getlink()); // Link the
previous node to the next node
   }

   else
   {
      first = node->getlink(); // If there's no
previous, node is the first node
   }
   delete node; // Now safely delete the node
}
```

Line 48: The new node is inserted into the linked list at the end (after the current tail). l->Insert (Tail, N) adds the new node N after the current Tail. Here l seems to represent an instance of a linked list class managing the nodes, and Insert places the new node into that structure.

// If not empty, update the Tail to point to the new node

```
60. template<class DT>
61. int Queue<DT>::Get()
62.       {
63.                 Node<DT> *N=Head;
64.                 int x=N->getdata();
65.                 if (Head==Tail)
66.                 {
67.                           Head=NULL;
68.                           Tail=NULL;
69.                 }
70.                 else
71.                 {
72.                           Head=Head->getlink();
73.                 }
74.                 l->Delete(N);
75.                 return x;
76.       }
77.
```

```
 1. #include<iostream>
 2. using namespace std;
 3. #include"Queue.h"
 4. #include"Queue.cpp"
 5. int main()
 6. {
 7.       Queue<int> *q =new Queue<int>();
 8.
 9.       if(q->Empty())
10.                 cout<<"Queue is currently empty"<<endl;
11.
12.       q->Put(1);
13.       q->Put(2);
14.       q->Put(3);
15.
16.       while (!q->Empty())
17.       {
18.                 int value=q->Get();
19.                 cout<<value<<endl;
20.       }
21.
22.       system("pause");
23.       return 0;
24. }
25.
```