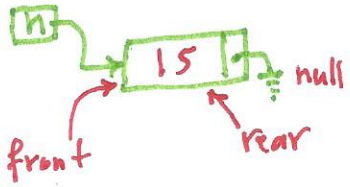
```
int main()
{  const int MAX_VALUES = 5;
   IntQueue iQ(MAX_VALUES);
   cout << "Enqueuing" << MAX_VALUES << "item ... \n";
   for (int x = 0, x < MAX_VALUES; x++)
         iQ.enqueue(x);  //
   Cout << "Now enqueue again --\n";
   iQ.enqueue(MAX_VALUES); // Queue is full.
   // dequeue and retrieve all elements
   while (! iQ.isEmpty())
   {  int value;
      iQ.dequeue(value);
      cout << value << endl;
    }
    return 0;
} //end main
```

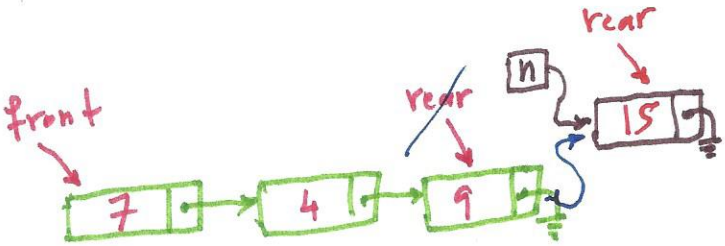
Week 11:

Dynamic Queue: A queue can be implemented as a linked list and can grow and shrink with each enqueue and dequeue call. We are not worry about the queue is full.


```
class DynQueue
{  private:
          struct Node
          {  int value;
             Node *next;
          }
          Node *front;
          Node *rear;
          int numItems;
   public:  DynQueue(); // prototype for default Constructor
```

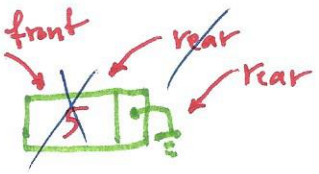
```
    ~DynQueue();//prototype
    // Queue Operation
    void enqueue(int); //prototype
    void dequeue(int &); //  "
    bool isEmpty() const; //  "
    void display() const; //  "
    void clear(); //  "
};//end class
// func. enqueue
void DynQueue::enqueue(int num)   // num = 15
{  Node *n = new Node; // new Node
   n->value = num;
   n->next = null;
   if(isEmpty())
   { front = n;
     rear = n;
   }
   else { rear->next = n;
          rear = n;
        }
   numItems++;
}//end enqueue class
// destructor
DynQueue::~DynQueue()
{  clear(); }
// Constructor
DynQueue::DynQueue()
{  front = nullptr; //or NULL
   rear = NULL; //or nullptr
   numItems = 0;
}
```

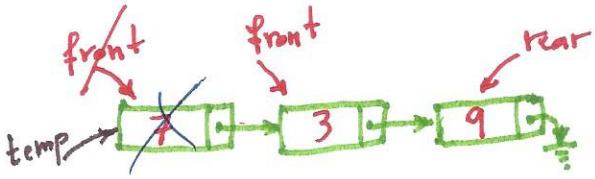




```
// Clear func. dequeues all the elements in a queue
Void DynQueue:: clear( )
{  int val;
   while ( !isEmpty())
         dequeue (val);
}
// isEmpty() func.
bool DynQueue:: isEmpty( ) const
{   if ( numItems>0)
              return false;
       else   return true;
}
// dequeue func removes the front element
void DynQueue:: dequeue ( int &num)
{  Node *temp;
   if ( isEmpty())
        cout <<" The Queue is empty.\n";
   else { // if we only have one node
          if (numItems ==1)
          { num = front->value;
            delete front;
            rear =NULL;
            numItems --;
          }
          else { num = front->value;
                 // remove the front node
                 // and delete it
                 temp = front;
                 front = front->next;
                 delete temp;
                 numItems --;
               } //endelse
        } //endelse
} //end func.
```

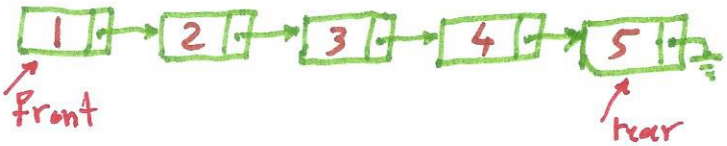
→ return numItems == 0;





```
// display function
void DynQueue :: display ( )
{  if (isEmpty())
        cout << "Queue is empty." <<endl;
   else{  while ( front != NULL)
          { cout << front->value << " ";
            front = front->next;
          } //endwile
       }//end else
} //end func.
```

```
// Driver for DynQueue
#include <iostream>
using . . . . ;
int main()
{
  const int MAX_VALUE = 5;
  DynQueue q1 ;
  Cout << "Enqueuing " << MAX_VALUE << "...items...\n";
  for (int X=1; X<=MAX_VALUE ; X++)
          q1.enqueue(X);
  int val ;
  Cout << "The values in the queue are :\n";
  while (! q1.isEmpty())
  {
     q1.dequeue(val);
     cout << val <<endl;
  }
  return 0;
} //end main
```



```
// A Dynamic Queue Template
template <class T>
class DynQueue
{  private:   struct Node
              { T value;
                Node *next;
              }
     . . .
     void enqueue(T); //prototype
     void dequeue(T&); //  "
     :       :
};//end class
// Constructor
template <class T>
DynQueue<T>::DynQueue()
{   front = nullptr;
    rear = nullptr;
    numItems = 0;
}
// destructor
template <class T>
DynQueue<T>::~DynQueue()
{   clear(); }
// enqueue
template <class T>
void DynQueue<T>::enqueue(T num)
{   . . .
    . . .
}
. . .
. . .
```

```
// Driver for Dynamic Queue Template
#include <iostream>
#include "DynQueue.h"
using namespace std;

int main()
{
   DynQueue <double> myQueue;
   DynQueue <float> yourQueue;
   DynQueue <int> herQueue;

   myQueue.enqueue (2.7);
    "       "    (5.4);
    "       "    (12.9);
   myQueue.display ();
    .       .       .
    .       .       .
    .       .       .
}
```

The STL deque (doubly Queue) and queue Containers.

STL = Standard Template Library

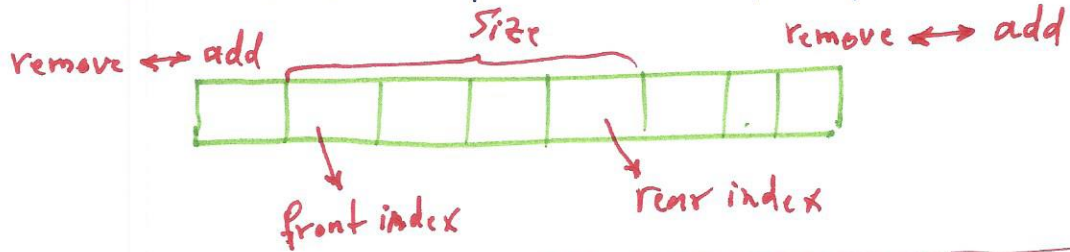
Member Function	
push_back	q1.push_back(5); // push 5 to the back of queue
pop_front	q1.pop_front(); // removes the first element
front	cout << q1.front(); // returns a reference to first element

The queue Container Adaptor:

It can be built upon vectors, lists, or deques. By default, it uses deque as its base. It uses: push(7), pop(), ...

```
iQ.push(num);
iQ.pop();
cout<<iQ.front;
```

- Deque: It is a doubly queue.
  - Insertions at both front and rear
  - Removals " " " " "
  - Stack and Queue are special cases of Deque.

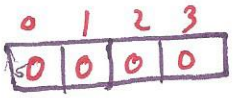


C++ Vectors: A vector is similar to an array but it can grow and shrink during run time (Dynamic Data structure or Dynamic Memory Allocation).

We need to include the vector library: `#include <vector>`

In order to use vector and its member functions.

```
Vector <int> ivec (4);
```
datatype → int, name → ivec, initial size → 4




```
vector <int> ivec( 4, 3);
```
size → 4, initial value → 3


```
ivec [2] =35;
ivec [0] = ivec [1] + ivec [3]; // 6

Vector <int> V(8);
for( int i=0; i<8; i++)
        V [i] =i;
 cout << V.size() << endl; // 8
// push_back ()
 ivec.push_back (100)
 ivec.push_back ( 200);
 cout << ivec. size();
//empty vector
vector <int> myvec ; // empty vector
myvec.push_back(12);
  "     "     (15);
```

```
// pop_back() removes the last element of the vector
   myvec.pop_back(); //15
   myvec.pop_back(); //12
   cout<<myvec.size(); //0
// Using vectors as parameters
int total (vector<int> V)
{  int sum=0;
   for(int i=0; i<V.size(); i++)
         sum+=V[i];

   return sum;
} //end func.
```

Test #2, Ch. 17, 18 (Linked List, stack, and Queue), Sat. 4-27-24.

Multiple Choice + Written (4 Questions)