Week 10, Sat.:

```
// Driver for DynStack
#include <iostream>
using namespace std;
int main()
{
   DynStack st1;
   int num;
   char ch;
   do
   { cout << "Stack Menu:\n P for push \n O for pop \n D for display \n
            Q for quit";
     cin >> ch;  // We can convert ch to uppercase to avoid having 'p' or 'P' cases.
     switch (ch)
     {   case 'p':
         case 'P': cout << "Enter a number:";
                   cin >> num;
                   st1.push(num);
                   break;
         case 'o':
         case 'O': st1.pop(num);
                   break;
         case 'd':
         case 'D': st1.display();
                   break;
         default: cout << "Invalid Selection.";
     } while (ch != 'Q' || ch != 'q')
     return 0;
   } // end main
```

```
// Dynamic Stack Template:
// Stack template
template <class T>
class DynStack
{  private:   struct Node
              { T value;
```

```
        Node * next;
    }
public: ...
    void push(T); // func. prototype
    void pop(T&); //  "    "
    bool isEmpty(); // "    "
    void display(); // "    "
    T topNode() const; // "  "
    ...
}; // end class
template <class T>
DynStack<T>:: ~DynStack()
{  while (!isEmpty())
     { pop(num); }
}
template <class T>
void DynStack<T> :: pop(T &num)
{   ...
}
... // Same for isEmpty and other functions
```

// Driver for stack template

```
#include <iostream>
 ....
int main()
{  string fruit;
   DynStack<int> st2;
   st2.push(10);
   st2.push(7);
   st2.push(15);
    ...
   DynStack<string> st3;
   st3.push("Orange");
   st3.push("Apple");
```

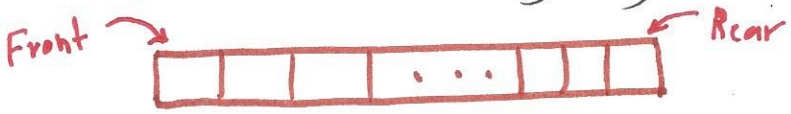
```
st3.push("Strawberry");
cout << st3.pop(fruit);
cout << "Top fruit in st3 is: " << st3.topNode() << endl;   // "Apple"
. . . .
}//end main
```

---

* The Queue Data structure: (FIFO), First in first out data structure.

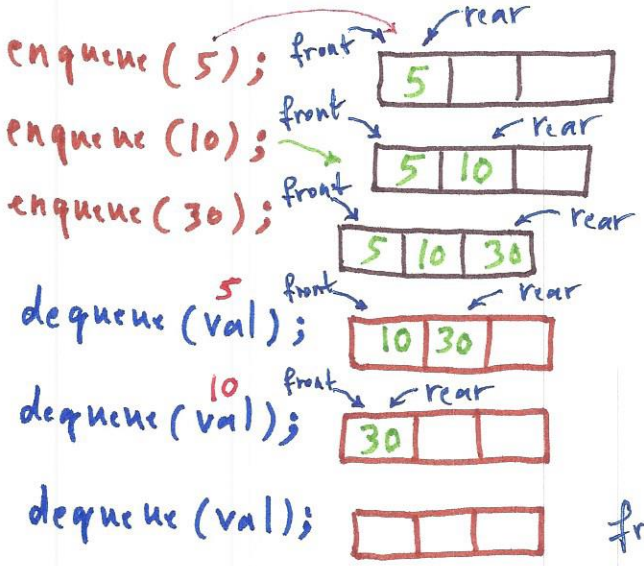
- Queues are data structures like stack, have restrictions on where we can add and remove elements.

Examples: 1- Cafeteria line
2- To see a teller in a bank
3- programming Network
4- O.S. (Multiuser, Multitasking, using printer, CPU time, ...)



Operations: - enqueue : Adding an element to queue
- dequeue: Removing " " from queue
- . . . . . .

Programming a queue is not as simple as stack. Because moving the first person in the queue, no one can leave or be added to the line until everyone has steped forward. So the line will move slowly.



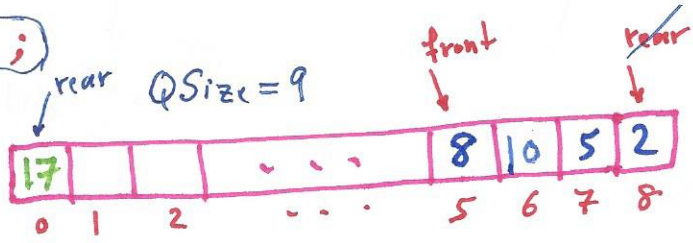
front = -1, rear = -1 (empty Queue)

Types of the Queue: 1- Static Queue (arrays)
2- Dynamic Queue (Linked List)

- Static Queue: We use array.

```
class IntQueue
{
  private: int *QArray; // points to QArray
           int QSize; // Max size of array
           int front; // index of Queue front
           int rear; //  "  "  "  rear
           int numItems;
  public : IntQueue(int); // Constructor
           IntQueue(const IntQueue &); // Copy Constructor
                                       // to initialize a new queue
                                       // with another existing queue
           ~IntQueue(); // destructor
           void enqueue(int); // adding a value at rear of a queue
           void dequeue(int &); // remove front element
           bool isEmpty() const;
           bool isFull() const;
           void clear();
           ...  // other funcs like display() func.
           ...  // Searching for a value in Queue
           ...
};//end class
// Constructor
IntQueue::IntQueue(int s)
{  QArray = new int[s]; // Dynamic memory allocation
   front = -1;
   rear = -1;
   numItems = 0;
   QSize = s;
}
// Destructor
IntQueue::~IntQueue()
{  delete [] QArray;
}
```

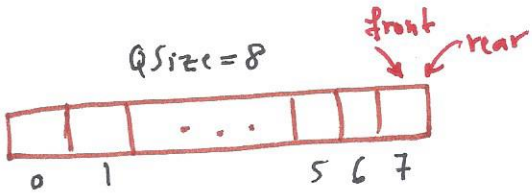
```
// enqueue func.
void IntQueue :: enqueue(int num)        // num = 17
{   if (isFull())
        cout<<" The Queue is full! "<<endl;
    else
        {  // we Calculate new rear position
           rear = (rear+1) % QSize;
           QArray [rear] = num;          // rear = 0, num = 17
           // update item count
           numItems ++;
        }
} // end enqueue
```



e.g. rear = 8
rear = (rear+1) % QSize
rear = (8+1) % 9 = 0
rear = 0

rear = 0
rear = (0+1) % 9 = 1

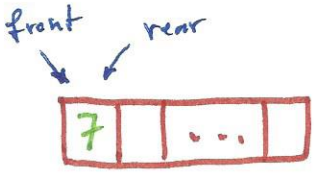
```
// Clear()
// Clear sets the front and rear indices
// and sets numItems = 0
void IntQueue :: Clear()
{   front = QSize-1;
    rear = QSize-1;
    numItems = 0;
} //end clear
```



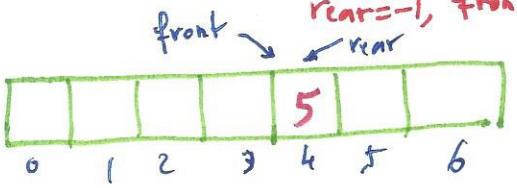

```
// isEmpty() func.
bool IntQueue:: isEmpty() Const
{    return  numItems == 0;
} //end isEmpty()
// isFull() func.
bool IntQueue:: isFull() Const
{   If (numItems < QSize)
            return false;
    else    return true;
} //end func.
```

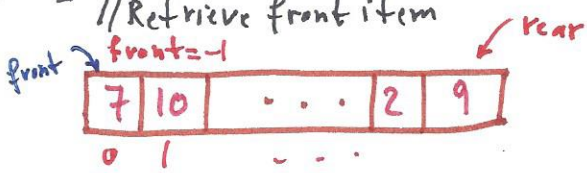
↓ ↓

```
// dequeue func., remove element from front
void IntQueue :: dequeue (int &num)
{   if (isEmpty())
        cout<<"The queue is empty!\n";
    else
        if( front == rear) // Queue with one element
        { num = QArray [front]
          front = -1;
          rear = -1;
          numItems = 0;
        }
        else //remove front element
        { front = (front+1) % QSize; // Because in Constructor front=-1
                                     // and in enqueue we dont change
                                     // the front value.
          num = QArray [front]; //Retrieve front item
          numItems--;
        }
}// end dequeue
```



rear=-1, front=-1





- With the static Queue, using dequeue, the values will remain in the array, the only thing changes are front and rear indices. With enqueue we may overwrite with existing value in a queue. So we cannot delete any value in a static queue.

```
// display() func.
void IntQueue :: display()
{   if(numItems == 0)
        cout << "The queue is empty!" <<endl;
    else
    {   for(int i=1; i <= numItems ; i++)
            cout << QArray [(front+i) % QSize] << " ";
    }//end else
}//end display()
// Driver for IntQueue class
#include <iostream>
#include "IntQueue.h"
  using namespace std;
```

```
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
```