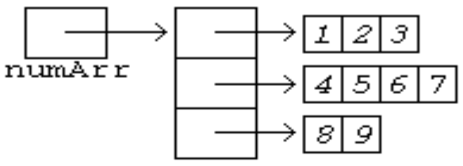
Q. # 1.

Create the Matrix class where each element is Rational object.

```
class Matrix
{
private:
  Rational * * mat;
  int rows, cols;
public:
  Matrix()
  Matrix(int r, int c );
  //all the functions which you implemented in Matrix of integers class.
};
```

Q. # 2.

In computer science a jagged array, also known as a ragged array, is a type of multidimensional array whose elements consist of one-dimensional arrays, hence it is an "array of arrays". The array is called "jagged" because the row/array can have different sizes, producing rows with different number of elements when visualized as output. In contrast, C++-styled arrays are always rectangular (all rows of same sizes).



Layout of the same array

We want to create an ADT named 'JaggedArray', which supports creation of jagged arrays. The operations needed for JaggedArray class and its data representation is given below.

To implement JaggedArray, we have used class Array that we implemented in a home-task; its complete implementation is given at the end of the question so you don't need to define or recall any of Array class functions. Following is the 'JaggedArray' class, whose detail is given below and also a sample run of JaggedArray class: You are required implement all the functions given in the class.

```
class JaggedArray
{
    Array * * data;
    int rows;
    bool isValidIndex( int i, int j);
public:
    JaggedArray();
    JaggedArray(int r, ...);
    JaggedArray( const JaggedArray & ref );
    int & at(int i, int j);
    const int & at(int i, int j) const;
    int getRows();
    int getColumn(int i);
    ~JaggedArray();
};
```

Jagged Array Data Representation

- data

    'data' will point to an array of pointers of type 'Array' whose each location will point to an Array object.

- rows

It contains the number of rows of jagged array (number of array objects / size of array pointed by ‘data’).

Jagged Array Operations

- `bool isValidIndex( int i, int j) const;`

  return true if the index received row and column are in range otherwise returns false.

- `JaggedArray();`

  Assigns data and rows equal to zero

- `JaggedArray(int r, ...);`

  It first argument ‘r’ receives the number of rows of jagged array and variable argument list receives the size of each array in jagged array.

  For Example if we create object like:

  ```
  JaggedArray ja(3,3,4,2);
  ```

  Then, it creates the same shape in memory as given in above diagram.

- `JaggedArray( const JaggedArray & ref );`

  Produces a deep copy of the received object.

- `int & at(int i, int j);`

  returns an alias of the element stores at row ‘i’ and column ‘j’ of jagged array

- `const int & at(int i, int j) const;`

  returns an alias of the element stores at row ‘i’ and column ‘j’ of jagged array

- `int getRows()const;`

  return the number of rows.

- `int getColumn(int i) const;`

  return the size of array at row ‘i’

- `~JaggedArray();`

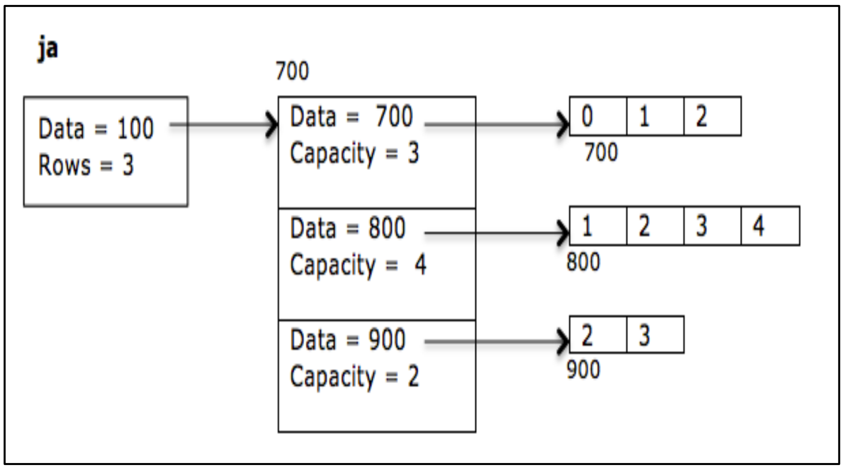
  It deallocates the resources captured by jagged array.

Sample Run

According to the sample-run/code given below: the object layout will be as follows:

```
int main()
{
    JaggedArray ja(3,3,4,2);
    for ( int i=0; i<3; i++)
    {
        for (int j=0; j<ja.getColumn(i); j++)
        {
            ja.at(i, j) = i+j;
        }
    }
    for ( int i=0; i<3; i++)
    {
```

```
        for (int j=0; j<ja.getColumn(i); j++)
        {
            cout<<ja.at(i,
j)<<" ";
        }
        cout<<"\n";
    }
    return 1;
}
```



```
class Array
{
    int *data;
    int  capacity;
    int isValidIndex( int index ) const
    {
        return index>=0 &&
index<capacity;
    }
public:
    ~Array()
    {
        if (data)
            delete [] data;
        data=0;
        capacity=0;
    }
    int & getSet(int index)
    {
        if (isValidIndex(index))
            return data[index];
        exit(0);
    }
    const int & getSet(int index) const
    {
        if (isValidIndex(index))
            return data[index];
        exit(0);
    }
    int getCapacity() const
    {
        return capacity;
    }
    void reSize ( int newCap )
    {
        if (newCap<=0)
        {
            this->~Array();
            return;
        }
        int * ptr = new int[newCap];
        memcpy(ptr, data,
(newCap<capacity?newCap:capacity)*sizeof
(int));

        this->~Array();

        capacity =
newCap<capacity?newCap:capacity;
        data=ptr;
    }
    Array ( const Array & ref)
    {
        if (ref.data==0)
        {
            data=0;
            capacity=0;
            return;
        }
        capacity=ref.capacity;
        data = new int[capacity];
        memcpy(data, ref.data,
capacity*sizeof(int));
    }
    Array(int argCount=5, ...)
    {
        if (argCount<=0)
        {
            capacity=0;
            data=0;
            return;
        }
        capacity = argCount;
        data = new int[capacity];
        va_list vl;
        va_start(vl, argCount);
        for ( int i=0; i<capacity; i++)
            data[i] = va_arg(vl, int);
    }
};
```

The Solution is given below, but do it yourself with full effort then you may consult/compare the following solution

```
class JaggedArray
{
    Array * * data;
    int rows;
    bool isValidIndex( int i, int j) const
    {
        return i>=0 && i<rows && j>=0 && j<=data[i]->getCapacity();
    }
public:
    JaggedArray()
    {
        data=0;
        rows=0;
    }
    JaggedArray(int r, ...)
    {
        if (r<=0)
        {
            rows=0;
            data=0;
            return;
        }
        rows  = r;
        data = new Array* [rows];
        va_list vl;
        va_start(vl, rows);
        for ( int i=0; i<rows; i++)
            data[i] = new Array(va_arg(vl, int));
    }
    JaggedArray( const JaggedArray & ref )
    {
        rows = ref.rows;
        data = new Array* [rows];
        for ( int i=0; i<rows; i++)
            data[i] = new Array(*(ref.data[i]));
    }
    int & at(int i, int j)
    {
        if (isValidIndex(i,j))
            return data[i]->getSet(j);
        exit(0);
```

```cpp
    }
    const int & at(int i, int j) const
    {
        if (isValidIndex(i,j))
            return data[i]->getSet(j);
        exit(0);
    }
    int getRows() const
    {
        return rows;
    }

    int getColumn(int i) const
    {
        if (i>=0 && i<rows)
            return data[i]->getCapacity();
        exit(0);
    }
    ~JaggedArray()
    {
        if(!data)
            return;
        for ( int i=0; i<rows; i++)
        {
            delete data[i];
        }
        delete [] data;
        data = 0;
        rows = 0;
    }
};
int main()
{
    JaggedArray ja(3,3,4,2);
    for ( int i=0; i<3; i++)
    {
        for (int j=0; j<ja.getColumn(i); j++)
        {
            ja.at(i, j) = i+j;
        }
    }
    for ( int i=0; i<3; i++)
    {
        for (int j=0; j<ja.getColumn(i); j++)
        {
            cout<<ja.at(i, j)<<" ";
        }
        cout<<"\n";
    }
    return 1;
}
```