330 Final

Brian Gunnarson

1.) myArray[i].j (myArray) $(4 \cdot 20) \cdot (8 \cdot 10) = 80^2 = 6400$

b.) A

c.) D

2.)
```
void rotate_arr(int **mat, int m) {
    for (int i=0; i<m/2; i++) {
        for (int j=i; j<m-i-1; j++) {
            int tmp = mat[i][j];
            mat[i][j] = mat[j][m-i-1];
            mat[j][m-i-1] = mat[m-i-1][m-j-1];
            mat[m-i-1][m-j-1] = mat[m-j-1][i];
            mat[m-j-1][i] = tmp;
        }
    }
}
```

3.)
```
void prefix_sum(int *A, int m){
    for (int i=1; i<m; i++) {
        A[i] += A[i-1];
    }
}
```

4.) a.)
```
My_Int()
My_Int()
My_NNInt()
~My_NNInt()
~My_Int()
~My_Int()
```

b.) E

c.) A

d.) 15 15
15 16

e.) A

5.) a.) ~Test();
Test(const Test& t);

b.) Test::~Test() {
delete [] array;
}

c.) Test::Test(const Test &t): array(t.array), n(t.n) {};

6.) a.) Declaration: Pixel(uchar c1, uchar c2, uchar c3);
Definition:
Pixel::Pixel(uchar c1, uchar c2, uchar c3) {
r=c1;
g=c2;
b=c3;
}

b.) Pixel::~Pixel() {};

Next Page

c.)
```
uchar Pixel::addSamples(const uchar c1, const uchar c2) const {
    unsigned int total = c1 + c2;   // use unsigned int since uchar goes to 255
    unsigned int avg = (unsigned int) total / 2;
    if (avg > 255)
        uchar result = 255;
    else if (avg < 0)
        uchar result = 0;
    else
        uchar result = (uchar) avg;
    return result;
}
```

d.)
```
const Pixel Pixel::operator+(const Pixel& p) const {
    uchar r = add_samples(p.r, this->r);
    uchar g = add_samples(p.g, this->g);
    uchar b = add_samples(p.b, this->b);
    return Pixel(r, g, b);
}
```

7.) a.)
```
unsigned int Vec<T>::get_size() const {
    return size;
}
T* Vec<T>::get_elem(unsigned int index) const {
    T* ret_val = &(array[i]);
    return ret_val;
}
```

b.)
```
template <class T>
void convert_matrix(T** in, T** out, unsigned int m, unsigned int n) {
    unsigned int size = m*n;
    *out = new T[size];
    int count=0;
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++) {
            (*out)[count] = in[i][j];
            count++;
        }
    }
}
```

c.) Definition: `Mat(unsigned int m, unsigned int n, T* arr);`

Implementation:

```
Mat<T>::Mat(unsigned int m, unsigned int n, T* arr): Vect(m*n, arr) { m=m;}
```

d.)
```
T* Mat<T>::getelem(unsigned int I, unsigned int J) {
    if (I==0)        //first row
        unsigned int idx = J;
    else
        unsigned int idx = J+ (this->get_size()/m);  // J+ # of columns
    T* ret_val = this->get_elem(idx);
    return ret_val;
}
```

e.)
```
void Mat<T>::print() const {
    unsigned int cols = (unsigned int) this->get_size()/m;
    for (int i=0; i<m; i++) {
        T* tmp_arr = new T[cols];
        for (int j=0; j<cols; j++) {
            tmp_arr[j] = array[j];
        }
        Vect tmp_vect(cols, tmp_arr);
        tmp_vect.print();
        delete [] tmp_arr;
    }
}
```

Extra Credit

Done in pt 2