

Arrays

Arrays

- An *array* is a sequence of objects all of which have the *same* type.
- The objects are called the *elements* of array and are numbered consecutively, i.e., 0, 1, 2, 3, These numbers are called *index values* or *subscripts* of the array.
 - The term subscript is used as a mathematical sequence, e.g., a_0, a_1, a_2, \dots
- The subscripts locate the element's position within an array, thereby giving *direct access* into the array.

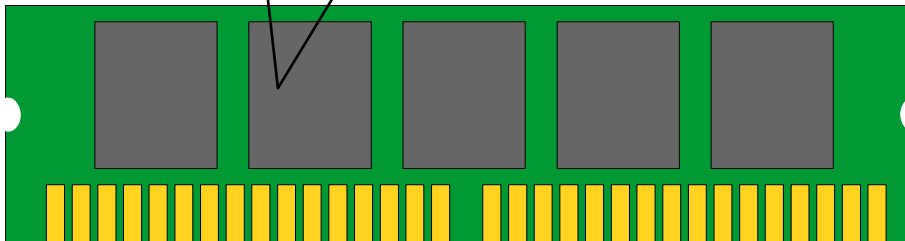
(cont'd.)

- If the name of the array is a , then $a[0]$ is the name of the element that is in position 0, $a[1]$ is the name of the element that is in position 1, and so on.
- In general, the i th element is in position $i - 1$.
- So, if the array has n elements, their names are $a[0], a[1], \dots, a[n-1]$.
- An array is visualized as a series of adjacent storage compartments that are numbered by their index value.

Example

a	
0	11.11
1	33.33
2	55.55
3	77.77
4	99.99

- `a[0]` contains 11.11
- `a[1]` contains 33.33
- ...
- The diagram actually represents a region of the computer's memory because an array is always stored this way with its elements in a contiguous sequence.

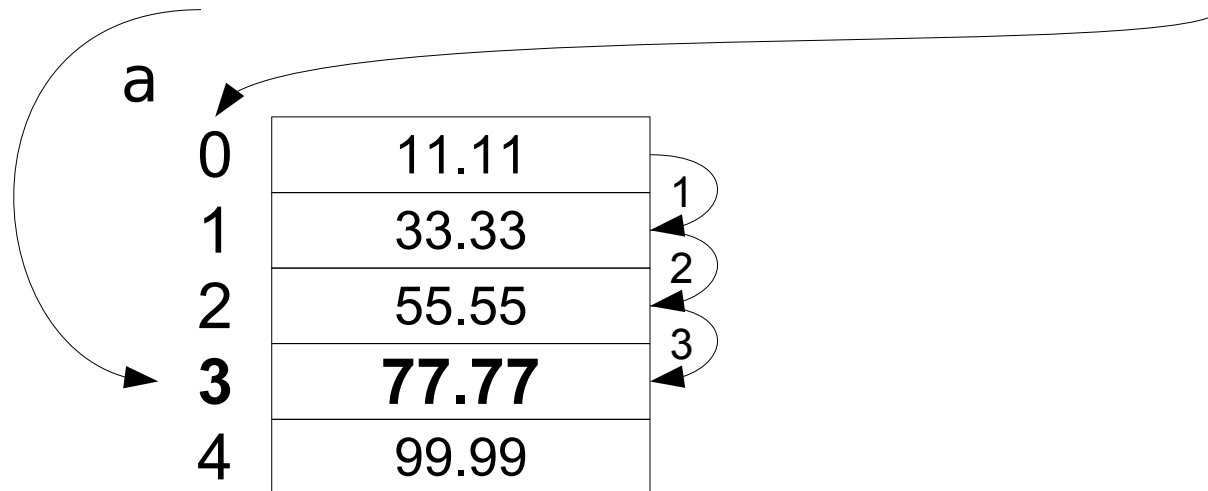


(cont'd.)

- The method of numbering the i -th element with index $i - 1$ is called *zero-based indexing*.
- It guarantees that the index of each array element is equal to the number of steps from the initial element `a[0]` to that element.
 - e.g., element `a[3]` is 3 steps from element `a[0]`.
- If several objects of the same type are to be used in the same way, it is usually simpler to encapsulate them into an array.

(cont'd.)

- The method of numbering the i -th element with index $i - 1$ is called *zero-based indexing*.
- It guarantees that the index of each array element is equal to the number of steps from the initial element $a[0]$ to that element.
 - e.g., element **$a[3]$** is 3 steps from element $a[0]$.



(cont'd.)

- If several objects of the same **type** are to be used in the same way, it is usually simpler to encapsulate them into an array.
 - e.g., arrays of float `a[7]`, int `b[6]`, char `c[5]`

a 0	2.2
1	1.55
2	444.44
3	33.333
4	99.99
5	55.501
6	7.0

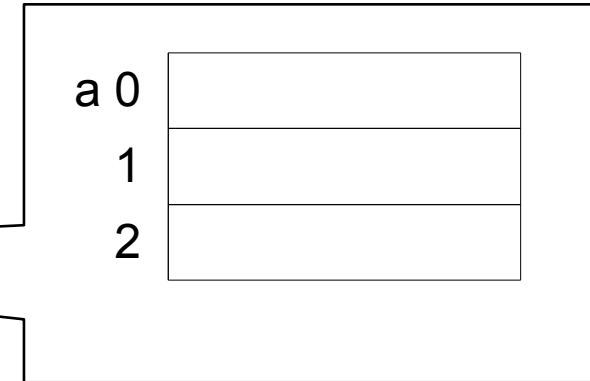
b 0	10
1	15
2	22
3	-4
4	0
5	99

c 0	'H'
1	'e'
2	'l'
3	'l'
4	'o'

Processing Arrays

- Using direct access on arrays

```
int main() {  
    double a[3];  
    a[2] = 55.55;  
    a[0] = 11.11;  
    a[1] = 33.33;  
    cout << "a[0] = " << a[0] << endl;  
    cout << "a[1] = " << a[1] << endl;  
    cout << "a[2] = " << a[2] << endl;  
}
```



Exercise #1

- Write a program to read a set of integers, then print them in reverse order.
 - Read the number of integers first.
c:>How many integer? 5
c:>1 3 5 7 9
c:>9 7 5 3 1
- Can you write this program without using array ?

Printing a Sequence in Reverse

```
int main() {  
    const int SIZE = 5; //fix to 5 numbers  
    double a[SIZE];  
    cout << "Enter " << SIZE << " numbers: \t";  
    for (int i = 0; i < SIZE; i++)  
        cin >> a[i];  
    cout << "In reverse order: ";  
    for (int i = SIZE - 1; i >= 0; i--)  
        cout << "\t" << a[i];  
    cout << endl;  
    Return 0;  
}
```

Initializing an array

- An array can be initialized with an optional *initializer list*:

```
float a[] = {22.2, 44.4, 66.6};
```

- The value in the list are assigned to the elements of the array in the order that they are listed.

a

0	22.2
1	44.4
2	66.6

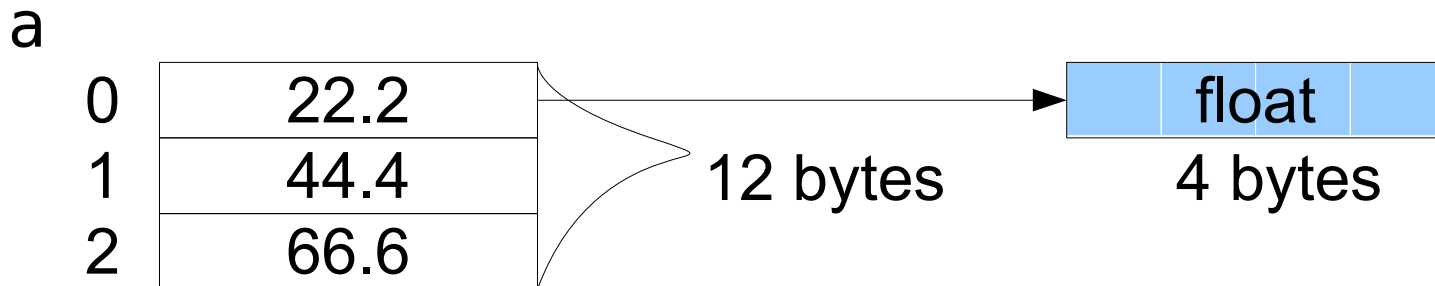
b

0	-2
1	1
2	0
3	1

_____ b[] = { _____ };

(cont'd.)

```
int main() {  
    float a[] = {22.2, 44.4, 66.6};  
    int size = sizeof(a)/sizeof(float);  
    for (int i = 0; i < size; i++)  
        cout << "\ta[" << i << "] = "  
            << a[i] << endl;  
}
```

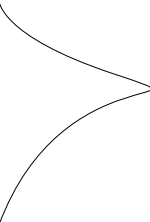


* Operator `sizeof()` = returns size in bytes of the object or type

Initialize an Array with Trailing Zeros

```
int main() {  
    float a[7] = {22.2, 44.4, 66.6};  
    int size = sizeof(a)/sizeof(float);  
    for (int i = 0; i < size; i++)  
        cout << "\ta[" << i << "] = "  
            << a[i] << endl;  
}
```

a	0	22.2
	1	44.4
	2	66.6
	3	0.0
	4	0.0
	5	0.0
	6	0.0



Trailing zeros for the rest of array

(cont'd.)

- The number of values in an array's initializer list cannot exceed its size.

```
float a[3] = {22.2, 44.4, 66.6, 88.8}; // !!
```

- An array can be initialized to be all zeros by using an empty initializer list.

```
float a[ ] = {0, 0, 0, 0, 0, 0, 0, 0, 0};  
float a[9] = {0, 0};  
float a[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
```

- This is **not** the same as using no initializer list.

An Uninitialized Array

- If an array is not initialized, it will contain garbage.

```
int main() {  
    const int SIZE = 4;  
    float a[SIZE];  
    for (int i = 0; i < SIZE; i++)  
        cout << "\ta[" << i << "] = "  
              << a[i] << endl;  
}
```

Arrays can be initialized ..

- But they **cannot** be assigned, so:
`float a[7] = {22.2, 44.4, 66.6};`
`float b[7] = {33.3, 55.5, 77.7};`
`b = a; // Error !!`
- **Nor** can array be used to initialize another:
`float a[7] = {22.2, 44.4, 66.6};`
`float b[7] = a; // Error !!`

Array Index to Exceed its Bounds

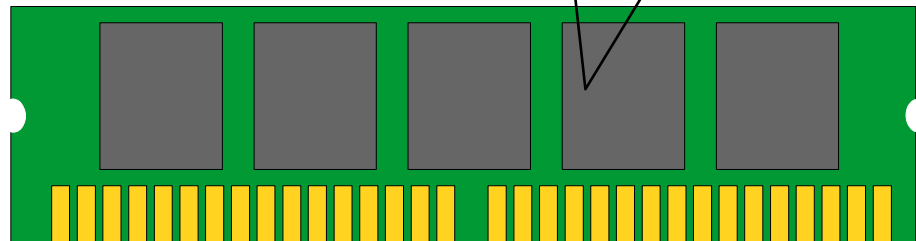
```
int main() {  
    const int SIZE = 4;  
    float a[SIZE] = {33.3, 44.4, 55.5, 66.6};  
    for (int i = 0; i < 7; i++)  
        cout << "\ta[" << i << "] = "  
            << a[i] << endl;  
}
```

a	?
0	33.3
1	44.4
2	55.5
3	66.6
	?
	?
	?
	?

Causing Side Effects

```
int main() {  
    const int SIZE = 4;  
    float a[] = {22.2, 44.4, 66.6};  
    float x = 11.1;  
    cout << "x = " << x << endl;  
    a[3] = 88.8;  
    cout << "x = " << x << endl;  
}
```

a	
0	22.2
1	44.4
2	66.6
3	88.8
Jump 3x4 bytes	

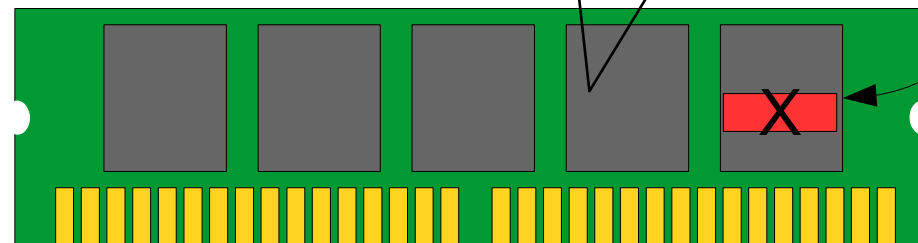


Causing Unhandled Exceptions

```
int main() {  
    const int SIZE = 4;  
    float a[] = {22.2, 44.4, 66.6};  
    float x = 11.1;  
    cout << "x = " << x << endl;  
    a[3333] = 88.8;  
    cout << "x = " << x << endl;  
}
```

- OS will terminate this program before it harm the system.

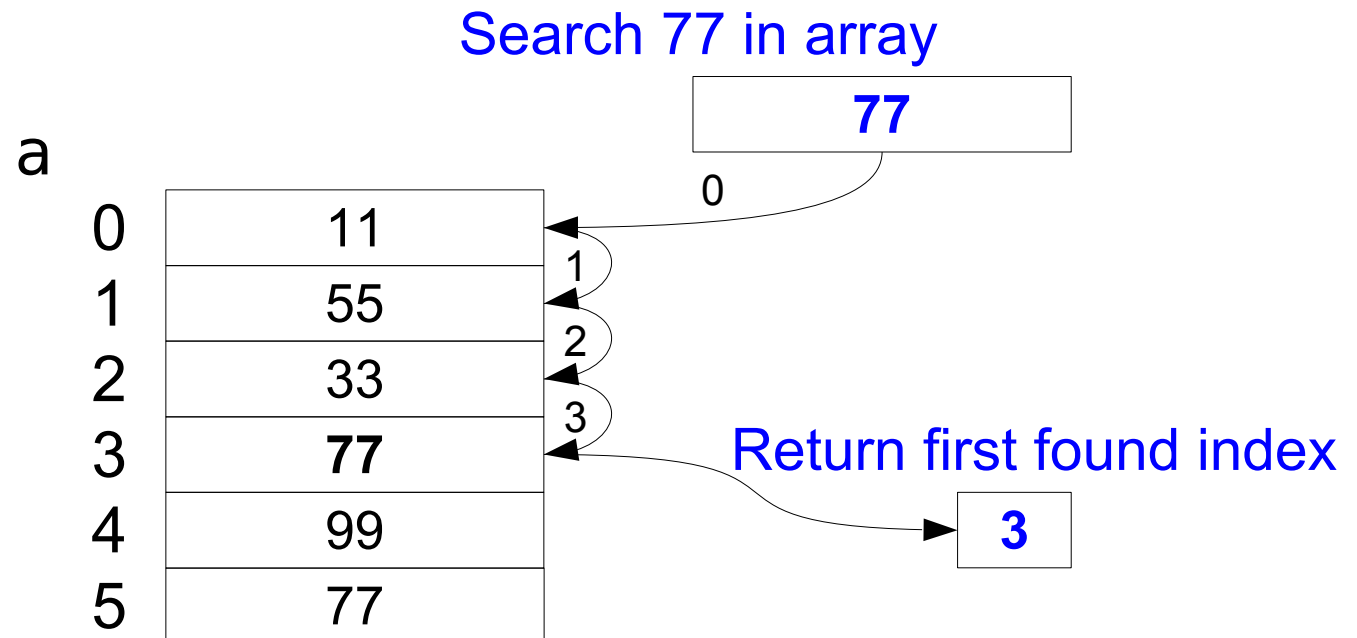
a	
0	22.2
1	44.4
2	66.6



Jump 3333x4 bytes

Linear Search Algorithm

- Find an object in an array.
- Start at the beginning
- Inspect each element, one after the other, until the object is found.
 - “One after the other”, that's why it is called *linear*.



* Return size of array if not found

(cont'd.)

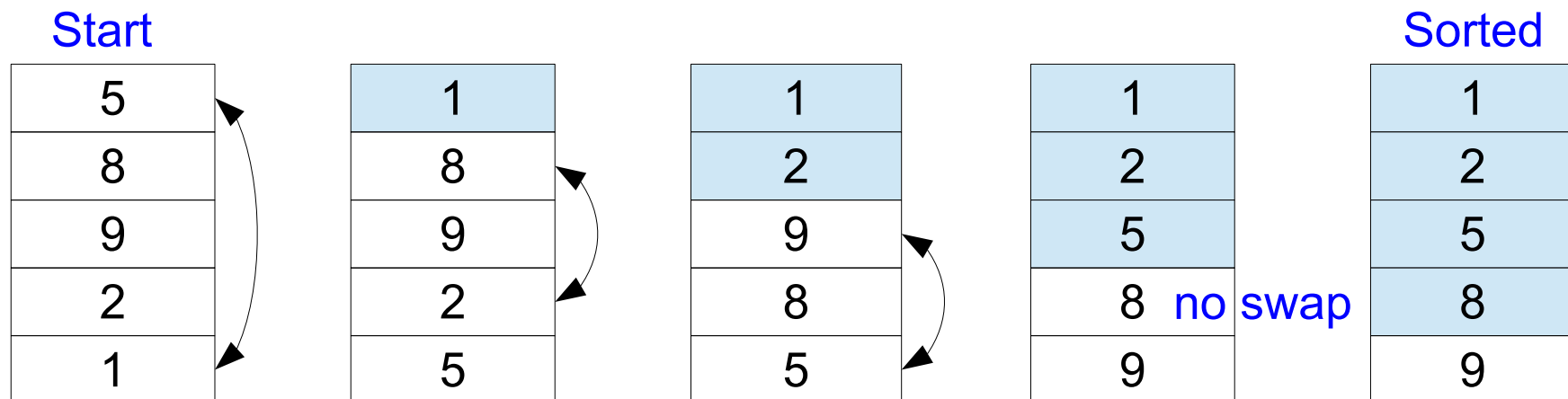
```
int main() {  
    int a[6] = {11, 55, 33, 77, 99, 77};  
    int target = 77;  
    int i = 0;  
    for ( ; i < 6; i++){  
        if (a[i] == target){  
            cout<<"found at a["<<i<<'] '<<endl;  
            return 0;  
        }  
    }  
    i++;  
    cout<<"not found"<<endl;  
    return 0;  
}
```

The Selection Sort

- The linear search algorithm is not very efficient.
- To use an efficient searching algorithm on a sequential data structure such as array, the structure must be sorted to put its element in order.
- There are many algorithms for sorting an array. The Selection Sort is one of the simplest.
 - But not as efficient as most others.

(cont'd.)

- The Selection Sort is very simple.
 1. Find the min/max
 2. Swap most left of unsorted with min/max
 3. Repeat 1-2 until last array (n-1)



(cont'd.)

```
int main() {
    int n =5;
    int a[] = {5, 8, 9, 2, 1};
    int i, j, iMin, temp;

    for (j = 0; j < n-1; j++) {
        iMin = j;
        for ( i = j+1; i < n; i++){
            if (a[i] < a[iMin]){
                iMin = i;
            }
        }
    } //end loop i
```

```
        if(iMin != j){
            temp= a[j];
            a[j] = a[iMin];
            a[iMin] = temp;
        }
    } //end loop j

    for (int i =0; i<n; i++){
        cout<<a[i]<<' ';
    }
    return 0;
}
```


Median

- Median in statistic is the **middle** value in the ***sorted*** data set.
- If there are two values in the middle, median will be calculated from the mean of this two middle values.
 - e.g. median of {1,2,3,**5**,7,8,9} is **5**
median of {1,2,2,**3,5**,7,8,9} is **4** from $(3+5)/2$

Exercise #2

- Write a program to read a set of integers, then print them in sorted order with median value.

- Read the number of integers first.

c:>How many integer? 7

c:>5 2 9 8 1 7 3

c:>1 2 3 5 7 8 9 median is 5

c:>How many integer? 10

c:>5 2 9 8 1 7 3 6 4 11

c:>1 2 3 4 5 6 7 8 9 11 median is 5.5

- Can you write this program without using array ?

Arrays and Type Definitions

- The syntax for the typedef of an array type:

```
typedef element-type alias[];
```

- Example

```
typedef float Sequence[];
```

(cont'd.)

```
void sort(Sequence, int);
```

```
int main() {  
    Sequence a = {55.5, 22.2, ....};  
    print(a, n);  
    sort(a, n);  
    print(a, n);  
}
```

```
void sort(Sequence a, int n) {  
    ...  
}
```

Multidimensional Arrays

- Element type of an array can be almost any type, including an array type.
- An array of arrays is called a *multidimensional array*.
- The simplest way to declare a multidimensional array is like:

```
double a[32][10][4];
```

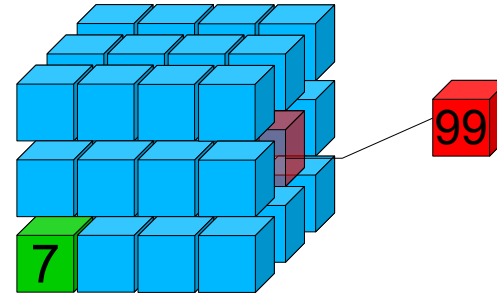
- An element can be identified by a multi-index., e.g.,

```
a[25][8][3] = 99.99;
```

(cont'd)

- Multidimensional array is an imagination world.

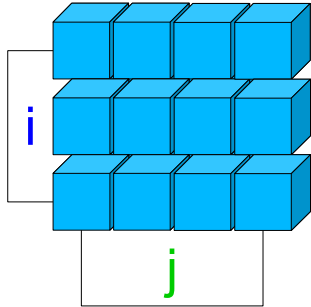
- `int a[3][4][3];`
`a[1][3][1] = 99;`
`a[2][1][0] = 7;`



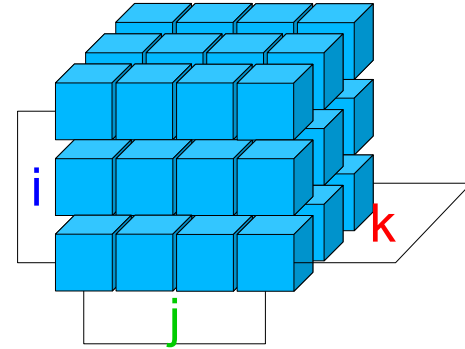
- Can you draw 4D array?
- `int b[3][3][3][2];`

Read and Print a 2-D Array

- Using nested loops to process arrays
 - 2-D array uses 2 nested loops.
 - n-D array uses n nested loops.



```
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 4; j++){  
        cin >> a[i][j];  
    }  
}
```



(cont'd.)

```
int main(){
    int a[3][5];
    cout << "Enter 15 integers, 5 per row:\n";
    for (int i = 0; i < 3; i++) {
        cout << "Row " << i << ": ";
        for (int j = 0; j < 5; j++){
            cin >> a[i][j];
        }
    }
    cout<< "Print array 3x5"<<endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 5; j++){
            cout << " " << a[i][j];
        }
        cout << endl;
    }
    return 0;
}
```


Process 2-D Array of Quiz Scores

```
const int NUM_STUDENTS = 3;  
const int NUM_QUIZZES = 5;
```

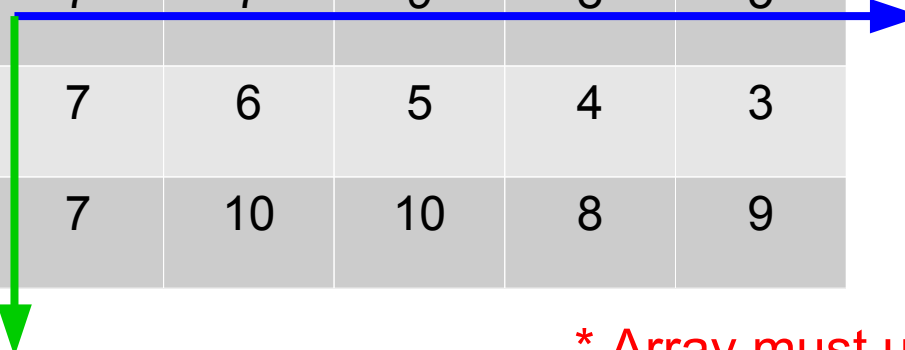
```
typedef int Score[NUM_STUDENTS][NUM_QUIZZES];
```

```
void read(Score);
```

```
void printQuizAverages(Score);
```

```
void printClassAverages(Score);
```

S\Q	q1	q2	q3	q4	q5
s1	7	7	9	8	5
s2	7	6	5	4	3
s3	7	10	10	8	9



* Array must use zero indexing

(cont'd.)

```
int main() {  
    Score score;  
    cout << "Enter " << NUM_QUIZZES  
        << " scores for each student:\n";  
    read(score);  
    cout << "The quiz averages are:\n";  
    printQuizAverages(score);  
    cout << "The class averages are:\n";  
    printClassAverages(score);  
    return 0;  
}
```

(cont'd.)

```
void read(Score score) {  
    for (int s = 0; s < NUM_STUDENTS; s++) {  
        cout << "Student " << s << ": ";  
        for (int q = 0; q < NUM_QUIZZES; q++)  
            cin >> score[s][q];  
    }  
}
```

(cont'd.)

```
void printQuizAverages(Score score) {  
    for (int s = 0; s < NUM_STUDENTS; s++) {  
        float sum = 0.0;  
        for (int q = 0; q < NUM_QUIZZES; q++)  
            sum += score[s][q];  
        cout << "\tStudent " << s << ": "  
             << sum/NUM_QUIZZES << endl;  
    }  
}
```

(cont'd.)

```
void printClassAverages(Score score) {  
    for (int q = 0; q < NUM_QUIZZES; q++) {  
        float sum = 0.0;  
        for (int s = 0; s < NUM_STUDENTS; s++)  
            sum += score[s][q];  
        cout << "\tQuiz " << q << ": "  
             << sum/NUM_STUDENTS << endl;  
    }  
}
```

Exercise #3

- Write a program to add two 3 x 3 matrices.

1	2	3
4	5	6
7	8	9

A

1	1	1
2	0	2
-3	-2	-1

B

2	3	4
6	5	8
4	6	8

C = A+B

Exercise #4

- Write a program to transpose a 3 x 3 matrix

1	2	3
4	5	6
7	8	9

A

1	4	7
2	5	8
3	6	9

A^t

Process a 3-D Array

```
int numZeroes(int[][4][3], int, int, int);
```

```
int main() {  
    int a[2][4][3] =  
        {{{5,0,2}, {0,0,9}, {4,1,0}, {7,7,7}},  
         {{3,0,0}, {8,5,0}, {0,0,0}, {2,0,9}}};  
    cout << "This array has "  
        << numZeroes(a, 2, 4, 3)  
        << " zeroes.\n";  
    return 0;  
}
```


(cont'd.)

```
int numZeroes(int a[][4][3],
               int n1, int n2, int n3) {
    int count = 0;
    for (int i = 0; i < n1; i++)
        for (int j = 0; j < n2; j++)
            for (int k = 0; k < n3; k++)
                if (a[i][j][k] == 0) ++count;
    return count;
}
```

Exercise #5

- Write a program to compute matrix multiplication.
 - $AB = \text{matrix multiplication}(\text{product})$
 - If A is an $n \times m$ matrix and B is an $m \times p$ matrix

$$(AB)_{ij} = \sum_{k=1}^m A_{ik} B_{kj}$$

