Arrays

Kornchawal Chaipah, PhD

Computer Engineering Department,
Khon Kaen University

Agenda

- 1D Array
- Passing an array to a function
- Array search algorithm
  - Linear search
  - Selection sort
- Multidimensional array
  - 2D array
  - 3D array

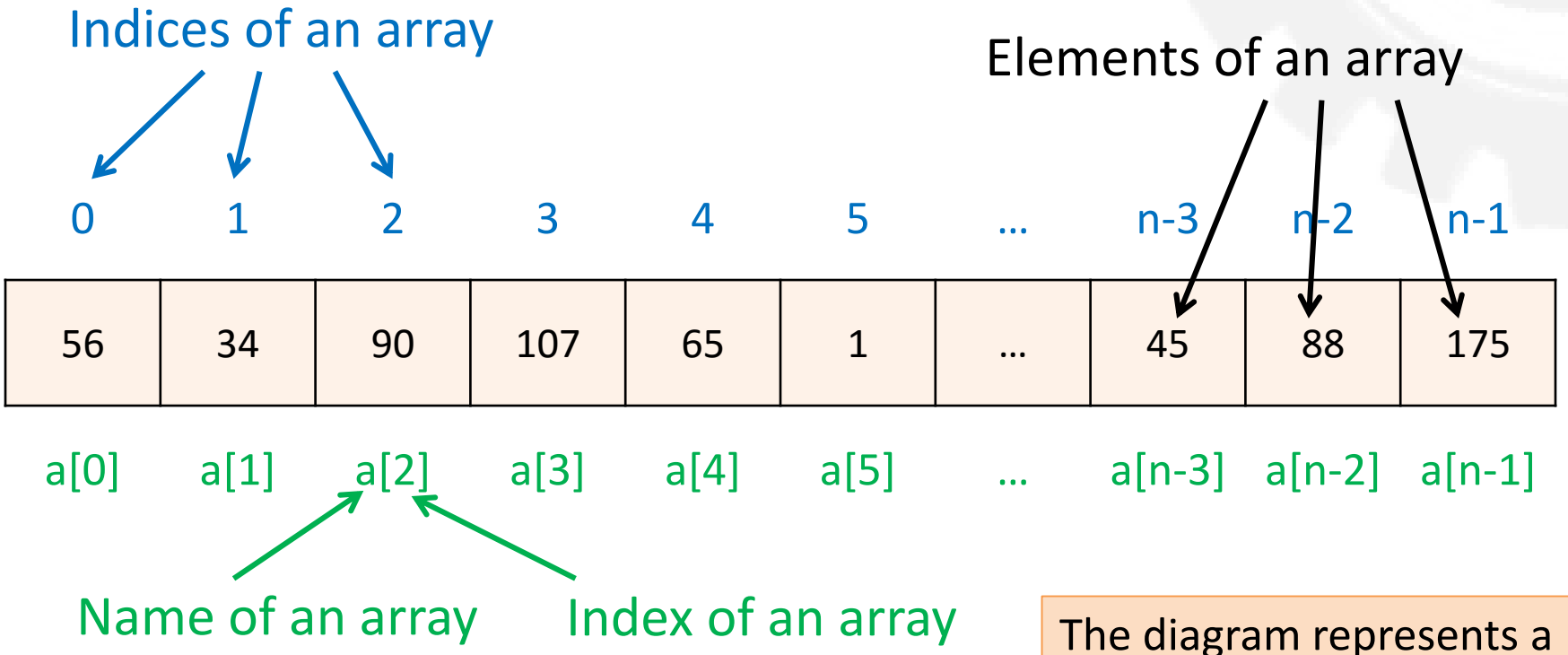
Array

- A sequence of object of the same type
  - a series of adjacent storage compartments that are numbered by their index value
- Use an array when several objects of the same type are to be used in the same way
- Element: each object in an array
- Index or a subscript: a position number of each element
  - Labeled as 0, 1, 2, …, n-1 where n is an array size

Zero-based indexing

- We can access each element in an array directly using an index
- An index starts from 0
  - i^{th} element is in the position i-1
- 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]

Array a[ ]



Accessing an array: name[index]

The diagram represents a region of the computer's memory, because an array is stored with its elements in a contiguous sequence.

Array Type Examples

`float a[7]`

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

`int b[6]`

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

`char c[5]`

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

Direct Access on an Array

```
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;
}
```

Declaring an array of size 3

Assigning values to elements

Reading values from an array

Simple Array Processing

- Problem: read a set of integers, then print them in a reverse order
- Input: several values of integers
- Output: values of integers in a reverse order
- Can we write this without an array?
  - Probably if the set size is not big and is not changed
- Let's use array

Printing a Sequence in Reverse

```
int main() {
   const int SIZE = 5;
   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;
}
```

Declaring an array

Reading values into an array

Printing an array in a reverse order

Initializing an Array

- Direct initializing (as seen before)

```
a[2]= 55.55;
a[0]= 11.11;
a[1]= 33.33;
```

- Using initializer list

```
float a[] = {22.2, 44.4, 66.6}
float a[7] = {22.2, 44.4, 66.6};
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};
```

Initializing an Array w/o a Size

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

a[0]	22.2
a[1]	44.4
a[2]	66.6

- Values in the list are assigned to the elements in order
- Size of array (the number of elements) = the number of elements in the list

Size of an Array

Code:

```
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;
    }
    return 0;
}
```

Size in bytes

Output:

sizeof(array) = size of type * number of elements

a[0]	22.2
a[1]	44.4
a[2]	66.6

4 bytes
4 bytes
4 bytes

sizeof(a) = 4*3 = 12 bytes

Initializing an Array with Trailing Zeros

```
float a[7] = {22.2, 44.4, 66.6}
```

a[0]	22.2
a[1]	44.4
a[2]	66.6
a[3]	0
a[4]	0
a[5]	0
a[6]	0

- Values are assigned in order
- The rest is filled with zeros
- Size of array = declared size
- Number of elements can't exceed the declared size

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

Initializer an all-zero array

```
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};
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
0	0	0	0	0	0	0	0	0

Initialize to zeros != uninitialize

An Uninitialized Array

- Uninitialized array contains garbage

Code:

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

Output:

Don't do these with arrays

- An array can't be assigned to

X

```
float a[7] = {22.2, 44.4, 66.6};
float b[7] = {33.3, 55.5, 77.7};
b = a;
```

- An array can't be used to initialize another

X

```
float a[7] = {22.2, 44.4, 66.6};
float b[7] = a;
```

Array Index to Exceed its Bounds

Code:

```
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;
   }
    return 0;
}
```

Output:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
33.3	44.4	55.5	66.6	?	?	?

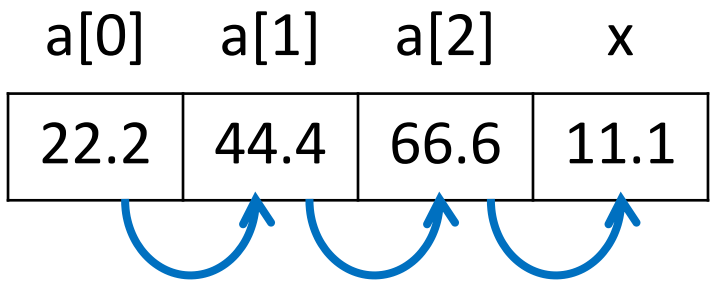
Causing Side Effects

Code:

```
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;
   return 0;
}
```

Output:

a[3] = 3 steps from a[0]



** This might or might not happen, depending on how a compiler allocate memory **

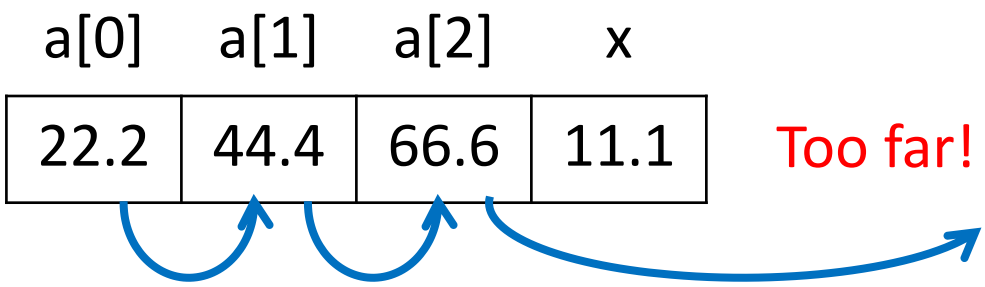
Causing Unhandled Exceptions

Code:

```
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;
   return 0;
}
```

Output:

a[3333] = 3333 steps from a[0]



Searching in an Array

- Finding an object in an array
- Comparing a value with each element
  - E.g. x == a[i]
- Linear search
  - Starts at the beginning (a[0])
  - Inspects each element, one after the other, until the object is found
  - Usually "returns" the index of the found object

Linear Search

Code:

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

Loop to search one by one

If found, 1) print index and 2) stop searching by terminating the program

If not found, the loop will end and the program proceeds to the regular termination

Sorting an Array

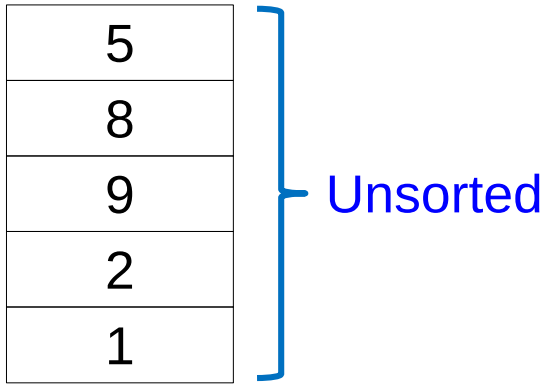
- The linear search is not efficient
- To use a more efficient searching algorithm, the data must be sorted
- Many algorithms to sort an array
- Selection sort is one of the simplest
  - Although not as efficient as most others

Selection Sort

1) Include all elements of an array in the unsorted set
2) Find min/max in the given unsorted set
3) Swap the min/max with the first element in the unsorted set
4) Shrink the unsorted set by one
   – Sorted set will be at the beginning of the array
5) Repeat 2)-4) until only one element left in the array

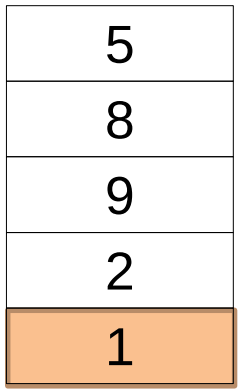
Selection Sort

1) Start

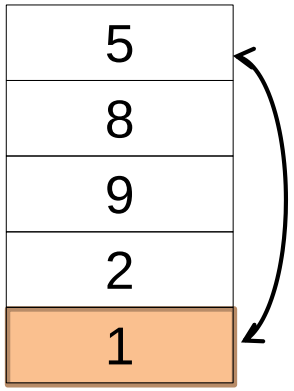


Selection Sort

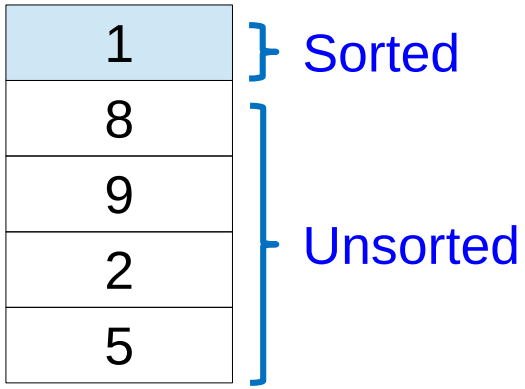
2) Find min



3) Swap min with the first element



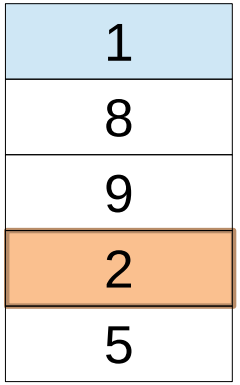
4) Shrink the unsorted set by one



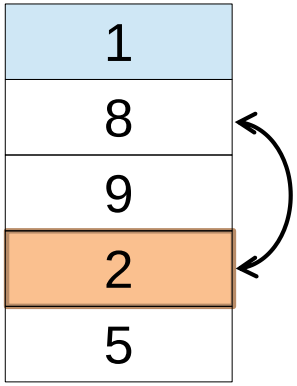
5) Repeat 2) - 4) until only one element left

Selection Sort

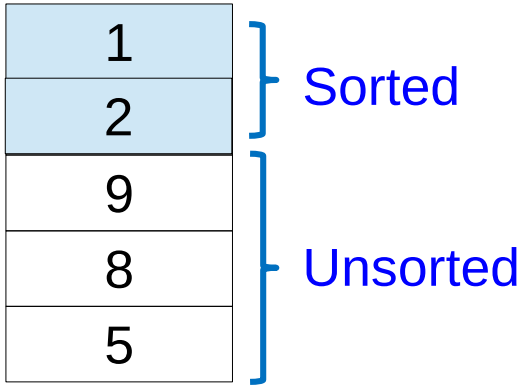
2) Find min



3) Swap min with the first element



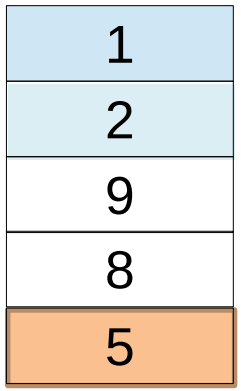
4) Shrink the unsorted set by one



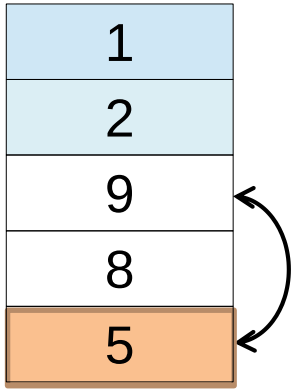
5) Repeat 2) - 4) until only one element left

Selection Sort

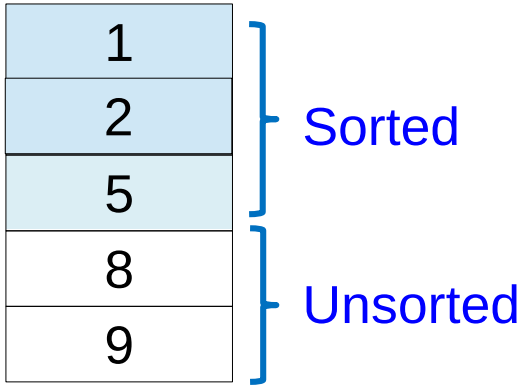
2) Find min



3) Swap min with the first element



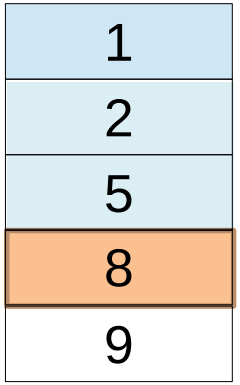
4) Shrink the unsorted set by one



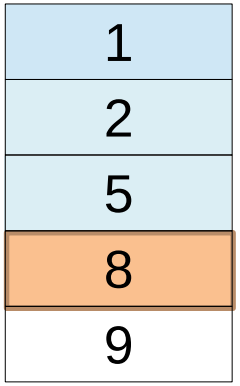
5) Repeat 2) - 4) until only one element left

Selection Sort

2) Find min

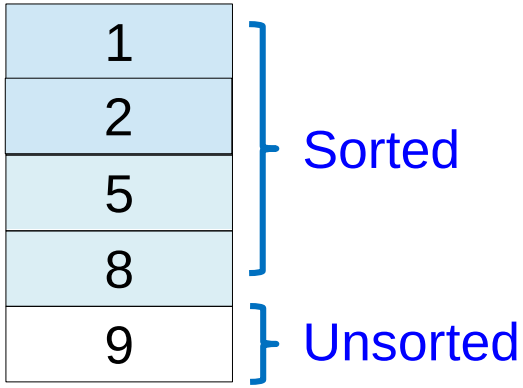


3) Swap min with the first element



No swap

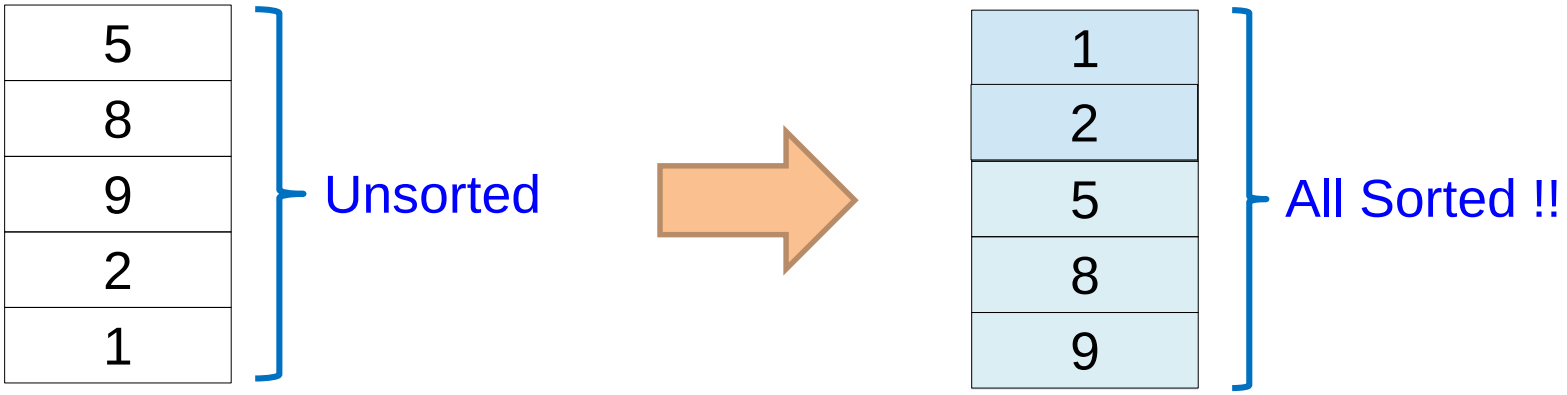
4) Shrink the unsorted set by one



5) Repeat 2) - 4) until only one element left

We have only one element left! So we are done!

Selection Sort



Selection Sort Code 1/2

Code:

```
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 (find min)

      …
```

For each round, start from index j and 1) find min 2) swap with a[j] 3) increase j

Loop to find min
iMin keeps the index of min

Selection Sort Code 2/2

Code:

```
    …
        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;
}
```

Swap min with j (the first one in the set)

Print the array

Median

- Median is the middle value of a sorted data set
- If there are 2 values in the middle, take an average
- E.g.
  - The median of {1,2,3,5,7,8,9} is 5
  - The median of {1,2,2,3,5,7,8,9} is (3+5)/2 = 4

Finding a Median

- Problem: read a set of integers (maybe unsorted) then print them in a sorted order with a median
- What are input(s) and output(s)?
- What do we know?
- Can we write this program without an array?

Finding a median

Code:

```
int main(){
   int n;
   cout << "How many integers? ";
   cin >> n;
   int a[n];
   // read array

   // sort array

   // find median!

   // print the sorted array and median

   return 0;
}
```

Passing an Array to a Function

- Similar to passing variable by reference
  - Can be changed in a function
  - But no & in the definition
- Pass just array name to a function
  - Actually passing the memory address of the first element of the array
- Need to specify array size in the definition
  - But can omit the size of the first dimension

Print() Function

Passing an array a[] to this function by omitting the first dimension size

Code:

```
void print(int a[], int n)
{
    for (int i=0; i<n ;i++){
        cout << a[i] << " ";
    }
    cout << endl;
}

int main( )
{
    int a[]={1,2,3};
    print(a,3);
    return 0;
}
```

Or you can put the array size

Code:

```
void print(int a[3], int n)
{
    for (int i=0; i<n ;i++){
        cout << a[i] << " ";
    }
    cout << endl;
}

int main( )
{
    int a[]={1,2,3};
    print(a,3);
    return 0;
}
```

Sum() Function

Code:

```
int sum(int [], int);

int main() {
  int a[] = {11, 33, 55, 77};
  int size = sizeof(a)/sizeof(int);
  cout << "sum(a, size) = " << sum(a, size)
       << endl;
}

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

Output:

I/O Functions for an Array (1/2)

Code:

```
const int MAXSIZE = 100;
void read(int[], int&);
void print(int[], int);

int main() {
   int a[MAXSIZE] = {0}, size;

   read(a, size);
   cout << "The array has " << size << " elements: ";

   print(a, size);
   cout << endl;

   return 0;
}
…
```

I/O Functions for an Array (2/2)

Code:

```
…
void read(int a[], int& n) {
    cout << "Enter integers. Terminate with 0:\n";
    n = 0;
    do {
        cout << "a[" << n << "]: ";
        cin >> a[n];
    } while (a[n++] != 0 && n < MAXSIZE);
    --n;
}

void print(int a[], int n) {
    for (int i = 0; i < n; i++){
        cout << a[i] << " ";
    }
}
```

Multi-dimensional Arrays

- Element type of an array can be almost any data type, including an array type!
- An array of arrays is called a multi-dimensional array
  - 2D: an array of an array in each "room"
  - 3D: an array of a 2D array in each "room"
  - ….

1D and 2D Arrays

- We knew a 1D array – think of it as a row of rooms

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]

- Now 2D – think of it as a table with rows and columns

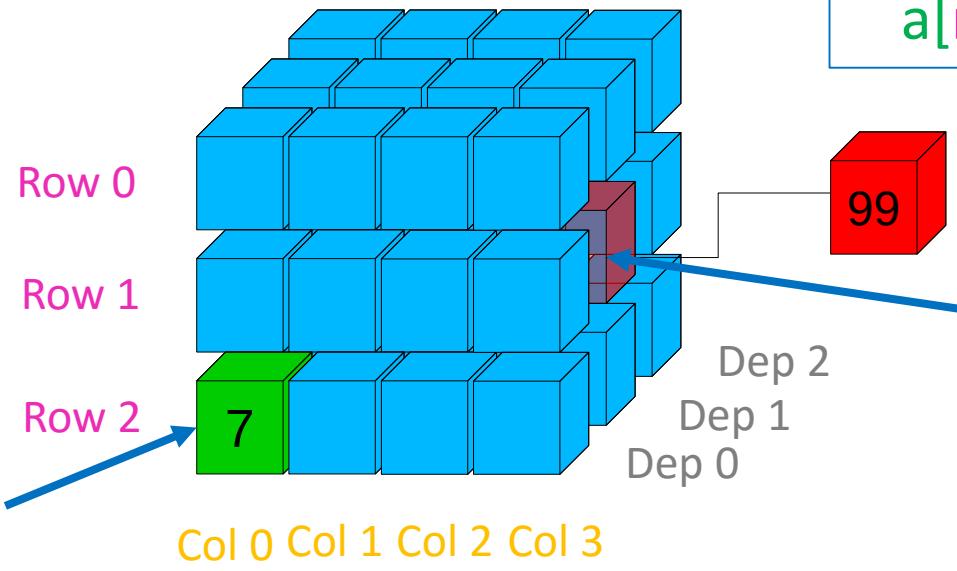
We can access each element by using a[row][col]

	Col 0	Col 1	Col 2	Col 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

3D Array

- Think of it as a cube box with rows, columns, and depth!

We can access each element by using a[row][col][dep]



Declaring and Initializing a Multidimensional Array

1D	`int a[size];`
	`int a[4];`
	`int a[] = {1,2,3,4};`

2D	`int a[row_size][col_size];`
	`int a[2][3];`
	`int a[2][3] =`
	`{ {1,2,3} ,`
	`{4,5,6} };`

3D	`int a[row_size][col_size][dep_size];`
	`int a[2][3][4];`
	`int a[2][3][4] =`
	`{ { {1,2,3,4} , {1,2,3,4} , {9,10,11,12} } ,`
	`{ {13,14,15,16} , {17,18,19,20} , {21,22,23,24} } };`

Accessing a Multidimensional Array

1D	`a[1] = 6;`
	`cout << a[1];`
	`cin >> a[1];`

2D	`a[1][2] = 6;`
	`cout << a[1][2];`
	`cin >> a[1][2];`

3D	`a[1][2][3] = 6;`
	`cout << a[1][2][3];`
	`cin >> a[1][2][3];`

Processing a Multidimensional Array

- Use nested loops to process a multidimensional array
  - 2D: use 2 nested loops
  - 3D: use 3 nested loops
  - nD: use n nested loops

Getting inputs for 2D Array

Code:

```
int main(){
   int a[3][4];
   for (int row = 0; row < 3; row++) {
      for (int col = 0; col < 4; col++){
            cin >> a[row][col];
      }
   }
   return 0;
}
```

Outer loop "controls" rows

Inner loop "controls" columns

	Col 0	Col 1	Col 2	Col 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

For each row, read in for each column.
row 0 -> col 0 -> col 1 -> col 2 -> col 3
row 1 -> col 0 -> col 1 -> col 2 -> col 3
row 2 -> col 0 -> col 1 -> col 2 -> col 3

Getting inputs for 2D Array

Code:

```
int main(){
   int a[3][4];
   for (int row = 0; row < 3; row++) {
      for (int col = 0; col < 4; col++){
            cin >> a[row][col];
      }
   }
   return 0;
}
```

Outer loop "controls" rows

Inner loop "controls" columns

	Col 0	Col 1	Col 2	Col 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

For inputs: 1 2 3 4 5 6 7 8 9 10 11 12 respectively

What if Col is in the outer loop?

Code:

```
int main(){
   int a[3][4];
   for (int col = 0; col < 3; col++) {
      for (int row = 0; row < 4; row++){
            cin >> a[row][col];
      }
   }
   return 0;
}
```

Outer loop "controls" columns

Inner loop "controls" rows

	Col 0	Col 1	Col 2	Col 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

For each col, read in for each row.
col 0 -> row 0 -> row 1 -> row 2
col 1 -> row 0 -> row 1 -> row 2
col 2 -> row 0 -> row 1 -> row 2
col 3 -> row 0 -> row 1 -> row 2

What if Col is in the outer loop?

Code:

```
int main(){
   int a[3][4];
   for (int col = 0; col < 3; col++) {
      for (int row = 0; row < 4; row++){
            cin >> a[row][col];
      }
   }
   return 0;
}
```

Outer loop "controls" columns

Inner loop "controls" rows

	Col 0	Col 1	Col 2	Col 3
Row 0	1	4	7	10
Row 1	2	5	8	11
Row 2	3	6	9	12

For inputs: 1 2 3 4 5 6 7 8 9 10 11 12 respectively

Read and Print Example

Code:

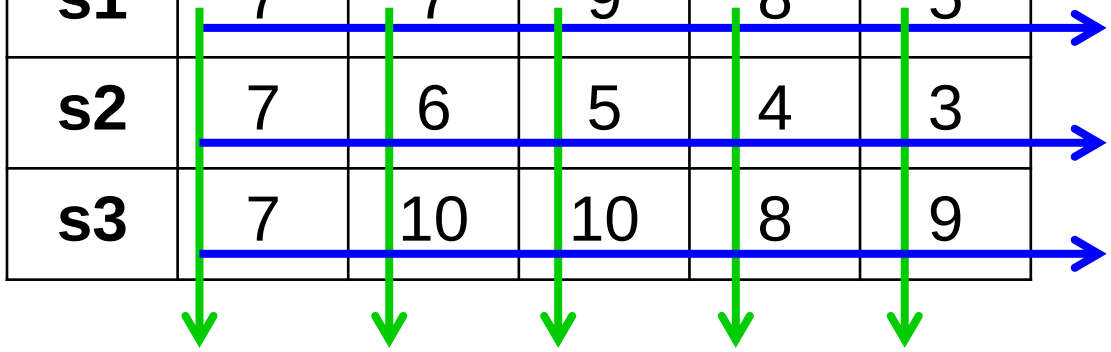
```
int main(){
   int a[3][5];
   cout << "Enter 15 integers, 5 per row:\n";
   for (int i = 0; i < 3; i++) {
        cout << "Row " << i << ": ";
        for (int j = 0; j < 5; j++){
           cin >> a[i][j];
         }
   }
   cout << "Print array 3x5" << endl;
   for (int i = 0; i < 3; i++) {
 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

Output:

Quiz Scores

- A "database" to keep scores of 5 quizzes for 3 students
- Then find
  - class average for each quiz
  - quiz average for each student

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



Quiz Scores: Global vars and func prototype (declaration)

Code:

```
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);

…
```

Quiz Scores: main()

Code:

```
…
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;
}
…
```

Quiz Scores: read() and printQuizAverages()

Code:

```
…
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];
                }
        }
}
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;
        }
}
…
```

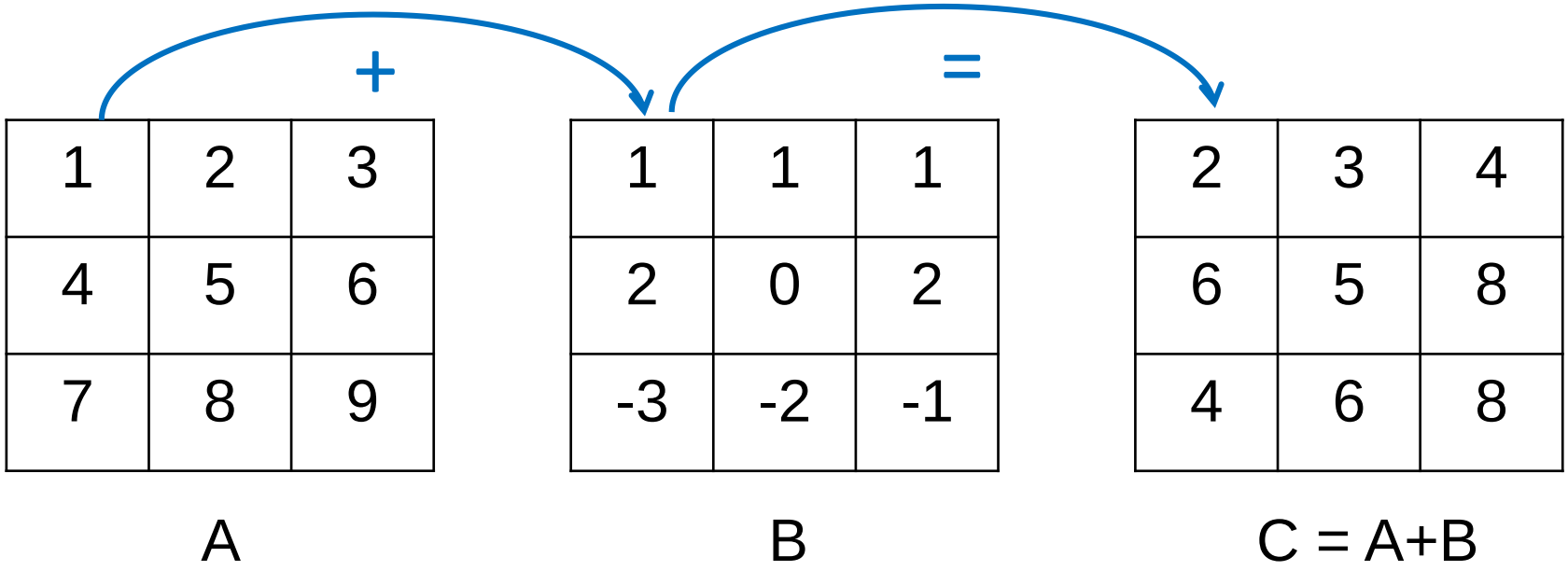
Quiz Scores: printClassAverages()

Code:

```
…
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;
        }
}
```

Adding Matrices

- Problem: write a program to add 2 matrices
- What you know:
  - Add values from the same positions



Adding Matrices: code

Code:

```
int main(){
   int A[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
   int B[3][3] = {{1,1,1}, {2,0,2},
                  {-3,-2,-1}};
   int C[3][3];

   // Add: C = A+B




   // Print C




   return 0;
}
```

Output:

Transposing Matrix

- Problem: write a program to transpose a matrix
- What you know:
  - Row -> Col
  - Col -> Row

1	2	3
4	5	6
7	8	9

A

1	4	7
2	5	8
3	6	9

At

Transposing Matrix: code

Code:

```
int main(){
   int A[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
   int At[3][3];

   // Transpose A -> At




   // Print At




   return 0;
}
```

Output:

Finding Zeros in a 3D Array (1/2)

Code:

```
int numZeros(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 " << numZeros(a, 2, 4, 3)
        << " zeros.\n";

   return 0;
}

…
```

Finding Zeros in a 3D Array (2/2)

Code:

We can omit the first dimension size when passing a multidimensional array to a function

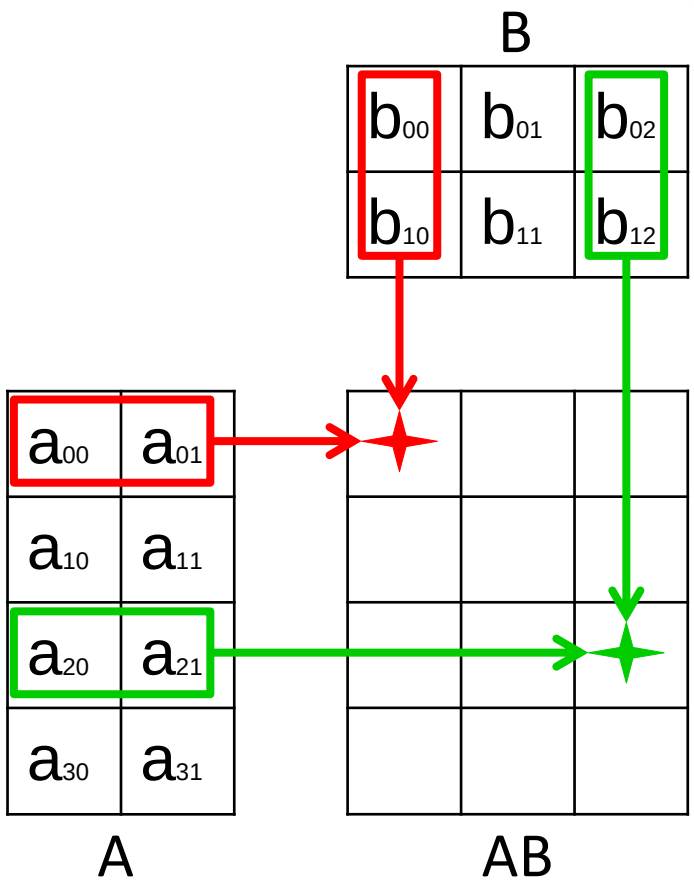
```
…

int numZeros(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;
}
```

Output:

Matrix Multiplication

- Problem: write a program to multiply 2 matrices
- What we know:
  - If A is an n x m matrix and B is an m x p matrix
  - AB is n x p

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



Multiplying Matrix: code

Code:

```
int main(){
   int A[4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
   int B[2][3] = {{2,2,2}, {3,3,3}};
   int AB[4][3];

   // AB = A x B (dot product)









   // Print AB



   return 0;
}
```

Output:

Take Home Messages 1/2

- Accessing a 1D array
  - Row houses with house numbers, starting from 0
  - Reading/printing forward and reverse
- Searching / sorting an array
- Passing array to the function
  - Work like passing by reference (passing the address of the first element)
  - But no &

Take Home Messages 2/2

- Accessing a 2D array
  - Table with row and column numbers
  - Double loop to read/write
- Accessing a 3D array
  - Box with row, column, and depth numbers
  - Triple loop to read/write

References

- คณาจารย์คณะวิศวกรรมศาสตร์. (n.d.). **เอกสารชุดนำเสนอภาพและบรรยายวิชาการเขียนโปรแกรม (ส่วนกลาง)**
- คณาจารย์คณะวิศวกรรมศาสตร์. (n.d.). **เอกสารประกอบการสอนวิชาการเขียนโปรแกรม (ส่วนกลาง)**
- รศ. วิโรจน์ ทวีปวรเดช. (2554). **การเขียนโปรแกรมคอมพิวเตอร์ Computer Programming.** พิมพ์ครั้งที่ 2 โรงพิมพ์มหาวิทยาลัยขอนแก่น
- Cplusplus.com. (n.d.). **C++ Documentation**. สืบค้นเมื่อ 18 กุมภาพันธ์ 2555, http://www.cplusplus.com/
- Cplusplus.com. (n.d.). **Library Reference**. สืบค้นเมื่อ 18 กุมภาพันธ์ 2555, http://www.cplusplus.com/
- ISO/IEC 14882 Programming Language – C++