Introduction to Arrays

Simple Examples

Math Examples

String Arrays

Sorting

Two-Dimensional Arrays
Why Arrays

**Task:** A company has 800 employees. Read their salaries, compute the average, and determine how many salaries are above the average.

**Observation:** First need to determine average. Then # salaries above it. Need to go over numbers twice. Must remember all.

**Perhaps:**

```c
double salary1, salary2, ..., salary800;
```

❌ Typing 800 names (in addition to values)
❌ Storing 800 names (in addition to values)
❌ Cannot be processed in a loop

**Solution:** Arrays!
What are Arrays

Ordered list of variables of the same type

Declaration:
```c
double salaries[800];
```

Accessing an element:
```c
salaries[5], salaries[i], salaries[2*i+j]
```

Comments:
- n-element array declared
- First element has index 0
- Last element has index n-1
- Array elements always stored in consecutive memory cells

```
124  -85  913  666
```

```
```
Assigning Values

- Array cells have arbitrary initial values
- Values can be assigned in a loop:
  
  ```c
  for (i = 0; i < 800; i++)
      salaries[i] = 3.5 * i;
  ```

- **Fixed-size** arrays can also be initialized at declaration
  
  ```c
  ```

- **Missing values** filled with 0’s
  
  ```c
  int grades[5] = {100, 97, 79};
  ```

- Can determine array size from initialization
  
  ```c
  int grades[] = {100, 97, 79, 0, 0};
  ```

- Different:
  
  ```c
  int grades[] = {100, 97, 79};
  ```

  - 5-element array
  - 3-element array
  - all 0
  - versus uninitialized
Run-time Array Declaration

❖ Can declare an array whose size is determined at run time

```c
int array_size;
scanf("%d", &array_size);
int my_array[array_size];
```

❖ Cannot initialize at declaration

```c
int my_array[array_size]={}; // ❌
```

❖ Can initialize later

```c
for (int i=0; i<array_size; i++)
    my_array[i]=0;
```
Outline

❖ Introduction to Arrays

❖ Simple Examples

❖ Math Examples

❖ String Arrays

❖ Sorting

❖ Two-Dimensional Arrays
#include<stdio.h>
#define NUM_EMPLOYEES 800

int main() {
    double salaries[NUM_EMPLOYEES], sum = 0.0, average;
    int i, above_average = 0;

    for (i = 0; i < NUM_EMPLOYEES; i++) {
        scanf("%lf", &salaries[i]);
        sum += salaries[i];
    }
    average = sum/NUM_EMPLOYEES;

    for (i = 0; i < NUM_EMPLOYEES; ++i)
        above_average += (salaries[i] > average);

    printf("Average is: %f\n", average);
    printf("%d salaries are above it\n", above_average);
    return 0;
}
Backwards (numbers)

❖ Read numbers and print them in reverse

```c
#include <stdio.h>

int main() {
    int num_numbers, i;
    printf("Number of numbers: ");
    scanf("%d", &num_numbers);
    int array[num_numbers];

    for (i=0; i<num_numbers; i++)
        scanf("%d", &array[i]);

    for (i=num_numbers-1; i>=0; i--)
        printf("%d", array[i]);

    printf("\n");
    return 0;
}
```

backwards_numbers.c
Which Digits

❖ Read digits
❖ Print each seen digit once
❖ In order of appearance
❖ \[3134177 \rightarrow 3147\]
❖ seen[10] Array for all digits
❖ Initially, all marked unseen
❖ Read digits
  ▶ If unseen - print, mark seen
  ▶ If seen - do nothing

#include <stdio.h>
#include <stdbool.h>

int main() {
    int num_digits, i, digit;
    bool seen[10] = {0};

    printf("# digits: ");
    scanf("%d", &num_digits);
    printf("%d digits: ", num_digits);

    for (i = 0; i < num_digits; i++) {
        scanf("%d", &digit);
        if (!seen[digit]) {
            printf("%d ", digit);
            seen[digit] = true;
        }
    }

    printf("\n");
    return 0;
}
Outline

❖ Introduction to Arrays

❖ Simple Examples

❖ Math Examples

❖ String Arrays

❖ Sorting

❖ Two-Dimensional Arrays
Decimal to Binary - with Arrays

❖ What is decimal 10 in binary?

```plaintext
   10
  0 0 1 0 1 0
```

❖ Same answer, different method:
```c
#include <stdio.h>  

dec2bin2.c  

int main() {  
    int decimal, i=0, bits[32];  
    printf("Nonnegative decimal: ");  
    scanf("%d", &decimal);  
    printf("(%d)_10 = (", decimal);  
    do {  
        bits[i++]=decimal%2;  
        decimal /= 2;  
    } while (decimal);  
    while(--i >= 0)  
        printf("%d", bits[i]);  
    printf(")\_2\n");  
    return 0;  
}  
```
The median of a set of numbers is the number that cuts the set in half: Half the numbers are above the median, and half are below it.

Example:

\[ 3, 4, 5, 5, 6, 7, 7, 9, 9, 10 \]

\[
\begin{align*}
9 & \quad 7 & \quad 5 & \quad 5 & \quad 6 & \quad 9 \\
3 & \quad 7 & \quad 5 & \quad 4 & \quad 6 & \quad 10
\end{align*}
\]

Definition: Arrange \( n \) numbers in increasing order:

\[ x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_{n-1} \leq x_n \]

If \( n \) is odd, then the median is the middle element of the ordered sequence: \( x_{(n+1)/2} \). If \( n \) is even, the median is the average of the two middle elements: \( (x_{n/2} + x_{n/2+1}) / 2 \).

Observation: The median is often more useful than average. For example, if salaries are 35, 40, 45, 50, 55, 60, 65, 70, 1000000

\[ \begin{align*}
\text{average} & = 1,111,157.7 \\
\text{median} & = 55
\end{align*} \]
Computing the Median

**Method A:** Sort the \( n \) numbers in increasing order, then take the middle one (or average middle two).

Sorting \( n \) numbers requires about \( n \log_2 n \) operations, hence inefficient.

**Method B:** Suppose the numbers can take at most \( m \) values. Make a histogram of the values observed (\( n \) operations). Then sum the histogram bars from left to right, till the sum exceeds \( n/2 \) (at most \( m \) operations).
Example: The input consists of \( n = 85 \) exam grades, each an integer in 0, 1, ..., 100. Count # times each grade occur

```c
#include <stdio.h>
#define NUM_GRADES 85
#define MAX_GRADE 100
int main() {
    int histogram[MAX_GRADE + 1] = {};
    int i, grade;

    for (i = 0; i < NUM_GRADES; i++) {
        scanf("%d", &grade);
        histogram[grade]++;
    }
    
    return 0;
}
```

Example: The input consists of \( n = 85 \) exam grades, each an integer in 0, 1, ..., 100. Count # times each grade occur
#include<stdio.h>
#define NUM_GRADES 85    // Assumed to be odd!
#define MAX_GRADE 100

int main() {
    int histogram[MAX_GRADE + 1] = {0};
    int i, grade, bar_sum;

    for (i = 0; i < NUM_GRADES; i++) {
        scanf("%d", &grade);
        histogram[grade]++;
    }

    bar_sum = grade = 0;

    while (bar_sum <= NUM_GRADES/2) {
        bar_sum += histogram[grade++];
    }

    printf("Median is %d\n", grade-1);
    return 0;
}
All the Primes up to $N$

**Problem:** Find all primes between 2 and a large positive integer $N$

**Possible Solution:** Test all integers in the range 2,3,...,$N$ for primality. This is very inefficient even with the best primality testing methods.

**The Eratosthenes Sieve:** First, pretend that all the integers in the range 2,3,...,$N$ are prime. Then proceed as follows:

- 2 is prime, mark all multiples of 2 as **not prime**
- 3 is prime, mark all multiples of 3 as **not prime**
- 4 was already marked as not prime, just skip it
- 5 is prime, mark all multiples of 5 as **not prime**
- 6 was already marked as not prime, just skip it

... and so on ...

After at most square root of $N$ such steps, all the numbers in the range 2,3,...,$N$ that remain unmarked **must be prime**.
Computing the Sieve of Eratosthenes for primes up to 120
Program

Integers represented in array, 1 indicates prime, 0 non-prime

```c
int main() {
    int n, i, p, sqrt_n=0;

    printf("Find primes up to: ");
    scanf("%d", &n);

    char prime[n+1];

    while ((sqrt_n+1)*(sqrt_n+1)<=n)
        sqrt_n++;

    for (i = 2; i <= n; i++)
        prime[i] = 1;

    for (p = 2; p <= sqrt_n; p++)
        if (prime[p])
            for (i = 2; i <= n/p; i++)
                prime[i*p] = 0;

    printf("Primes between 2 and %d:
", n);
    for (i = 2; i <= n; i++)
        if (prime[i])
            printf("%d
", i);

    return 0;
}
```

So `sieve[n]` represents `n`

`floor(sqrt(n))`

Start by assuming all prime

Not eliminated - indeed prime

All multiples of `p` are not prime
Comparison with Primality Testing

A short program can compare Eratosthenes’ Sieve with testing every integer in the range using the improved primality testing algorithm.

**Parameters:**
- \( N = 1,000,000 \)
- Repeated 100 times in a loop

**Results:**
- Eratosthenes’ Sieve: **1.5 sec**
- Primality testing: **16.0 sec**

**Sieve of Eratosthenes 10 times faster**

```c
int prime = 1, n, i, sqrt_n;
...
if (n == 1 || (n != 2 && n%2 == 0))
    prime = 0;
else {
    while ((sqrt_n+1)*(sqrt_n+1)<=n)
        sqrt_n++;
    for (i = 3; i <= sqrt_n; i += 2)
        if (n%i == 0) {
            prime = 0;
            break;
        }
}
printf("%d is %sa prime\n", prime ? "" : "not ", n);
...
```
Outline

❖ Introduction to Arrays
❖ Simple Examples
❖ Math Examples
❖ String Arrays
❖ Sorting
❖ Two-Dimensional Arrays
Strings of characters

"ece15" string of 5 characters
"" empty string

Stored in character arrays

... // Better:
char class1[]="ece15"; // array
// Possible, but risky as we'll soon see:
char class2[] = {'e','c','e','1','5'};
...
#include <stdio.h>

int main() {
    printf("My fav class: %s\n", "ece15");

    char word1[50], word2[50];
    printf("One word: ");
    scanf("%s", word1);
    printf("word1: %s\n", word1)

    printf("Two words: ");
    scanf("%s%s", word1, word2);
    printf("word1: %s\n", word1);
    printf("word2: %s\n", word2);
    return 0;
}

Read & write -- %s
Printed strings before

Never read! - why? Need to store string. Array!

array a must
%s reads till space

No &
How does `printf` know when to stop printing?
(string had length 50, word was shorter)

Strings stored with '\0' at end (Null char whose ASCII value is 0)

```c
char class1[]="ECE15";
```

Print string without %s

```c
for(int i=0; string[i]; i++)
    printf("%c", string[i]);
printf("\n");
```

Correct array definition

```c
char class2[]={"e","c","e","1","5","\0"};
```

<table>
<thead>
<tr>
<th>Char</th>
<th>Desc</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>SOH</td>
<td>Start of Heading</td>
<td>1</td>
</tr>
<tr>
<td>STX</td>
<td>Start of Text</td>
<td>2</td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text</td>
<td>3</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Transmission</td>
<td>4</td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry</td>
<td>5</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledge</td>
<td>6</td>
</tr>
<tr>
<td>BEL</td>
<td>Bell</td>
<td>7</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
<td>8</td>
</tr>
</tbody>
</table>
Two Things that Require Work

❖ Copying

▸ For numbers and characters can write: var2=var1;

▸ Does not work for strings

▸ Need loop

```c
#include <stdio.h>

int main() {
    char word1[50], word2[50];
    int i=0;
    printf("First word: ");
    scanf("%s", word1);
    for (i=0; word1[i]; i++)
        word2[i]=word1[i];
    word2[i]=\0;
    printf("Second word: %s\n", word2);
    return 0;
}
```

Later see a function for this
Two Things that Require Work (2)

❖ Comparing

- For numbers and characters can write: `if(var2==var1)`
- Does not work for strings
- Need loop

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    char word1[50], word2[50];
    int i=0;
    bool same=true;
    printf("First word:  ");
    scanf("%s", word1);
    printf("Second word: ");
    scanf("%s", word2);
    do
        if (word2[i] != word1[i])
            same = false;
    while (word1[i] && word2[i++]);
    printf("%s\n", same ? "same" : "different");
    return 0;
}
```

Later see a function for this too..
Backwards (characters)

❖ Read word and print it in reverse

```c
#include <stdio.h>
int main() {
    int i, j;
    char word[50];

    printf("Word: ");
    scanf("%s", word);
    for (i=0; word[i]; i++) ;
    printf("Reverse: ");
    for (j=i-1; j>=0; j--)
        printf("%c", word[j]);
    printf("\n");
    return 0;
}
```

backwards_letters.c
Outline

❖ Introduction to Arrays
❖ Simple Examples
❖ Math Examples
❖ String Arrays
❖ Sorting
❖ Two-Dimensional Arrays
Two variables: \( x \) and \( y \)

Exchange their values

\[
\begin{array}{cc}
\text{before} & \text{after} \\
0 & 1 \\
1 & 0 \\
\end{array}
\]

Attempt

\[
\begin{array}{ccc}
\text{x=y;} & \text{y=x;} & \text{before} \\
\text{before} & \text{after} \\
0 & 1 \\
1 & 1 \\
1 & 1 \\
\end{array}
\]

Correct

\[
\begin{array}{ccc}
\text{temp=x;} & \text{x=y;} & \text{y=temp;} \\
\text{before} & \text{after} \\
0 & 1 & ? \\
0 & 1 & 0 \\
1 & 1 & 0 \\
1 & 0 & 0 \\
\end{array}
\]
Can we exchange two numbers without an extra variable?

Yes

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>x=x+y</td>
<td>a+b</td>
<td>b</td>
</tr>
</tbody>
</table>
y=x-y| a+b | a |
x=x-y|   | a |

Risky
n numbers: $a_0, a_1, \ldots, a_{n-1}$

Sort in increasing order

Bubble sort

move largest # to right

- Compare $a_0$ with $a_1$
  - if $a_0 > a_1$, exchange them

- Compare $a_0 = 1$ with $a_2$
  - if $a_1 > a_2$, exchange them

etc.

move remaining largest to right

... 

Demo: Hungarian, Indian, Swedish

```c
#include <stdio.h>

int main() {
    int number, i, j, temp;
    printf("Number of elements: ");
    scanf("%d", &number);
    int array[number];
    for (i=0; i<number; i++)
        scanf("%d", &array[i]);
    for (i=0; i<number-1; i++)
        for (j=1; j<number-i; j++)
            if (array[j-1]>array[j]) {
                temp = array[j-1];
                array[j-1] = array[j];
                array[j] = temp;
            }
    for (i=0; i<number; i++)
        printf("%d ", array[i]);
    printf("\n");
    return 0;
}
```

sort.c
Outline

❖ Introduction to Arrays
❖ Simple Examples
❖ Math Examples
❖ String Arrays
❖ Sorting
❖ Two-Dimensional Arrays
Two-dimensional arrays

Array of 2 rows and 3 columns

Initialization

```
int ar[2][3] = {{1,0,2},{3,9,7}};
int ar[][3] = {{1,0,2},{3,9,7}};
```

3 necessary

Stored consecutively row by row

```
int ar[][3] = {1,0,2,3,9,7};
```

Addressing

```
int ar[i][j];
ar[0][0]=1;
ar[1][2]=7;
```

Element in row i, column j

Row Column
Applications of 2 (and 3) dim arrays

- Pictures
- Cars
- Planes
- Phones
- Video games
Matrix transposition

❖ **Matrix** - two dimensional array of numbers

❖ $M[i][j]$ - element at row $i$, column $j$
  ‣ $M[0][0]=1$, $M[1][2]=6$

❖ **Transposition** - flip around diagonal
  ‣ $M[i][j] \leftrightarrow M[j][i]$

❖ Loop over $i$ and $j$
  ‣ Swap $M[i][j]$ and $M[j][i]$
```
#include <stdio.h>
#define SIZE 3

int main() {
    int ar[][SIZE] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
    int i, j, temp;

    printf("Matrix:\n");
    for (i=0; i<SIZE; i++)
        for (j=0; j<SIZE; j++)
            printf("%d %s", ar[i][j], j==SIZE-1? "\n" : "");

    for (i=1; i<SIZE; i++)
        for (j=0; j<i; j++) {
            temp = ar[i][j];
            ar[i][j]=ar[j][i];
            ar[j][i]=temp;
        }

    printf("Transpose:\n");
    for (i=0; i<SIZE; i++)
        for (j=0; j<SIZE; j++)
            printf("%d %s", ar[i][j], j==SIZE-1? "\n" : "");
    return 0;
}
```
Outline

❖ Introduction to Arrays
❖ Simple Examples
❖ Math Examples
❖ String Arrays
❖ Sorting
❖ Two-Dimensional Arrays