CHAPTER 3

CONTROL STRUCTURES AND DATA FILES

Algorithm Development

- **Top-down design**

This approach presents a “big picture” description of the problem solution in sequential steps. The description of the problem is **refined** until the steps are language statements.

- **Refinement with Pseudocode and Flowcharts**

The pseudocode uses English-like statements to describe the steps of the program.
The flowchart uses a diagram to describe the steps

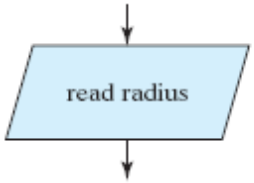
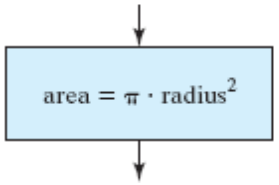
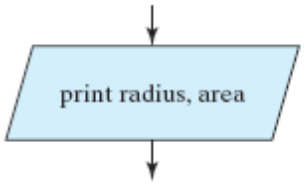
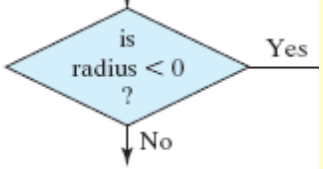
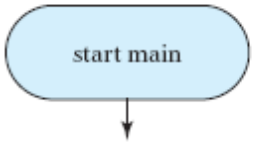
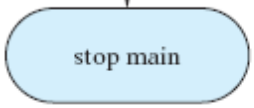
Basic Operation	Pseudocode Notation	Flowchart Symbol
Input	read radius	
Computation	set area to $\pi \cdot \text{radius}^2$	
Output	print radius, area	
Comparisons	if radius < 0 then ...	
Beginning of algorithm	main:	
End of algorithm		

Figure 3.1 *Pseudocode notation and flowchart symbols.*

Structured Programming

- **Sequence**

A sequence contains steps that are performed one after another.

For example the flowchart for a program that is used to compute the velocity and acceleration of an aircraft in described next

Execution of program

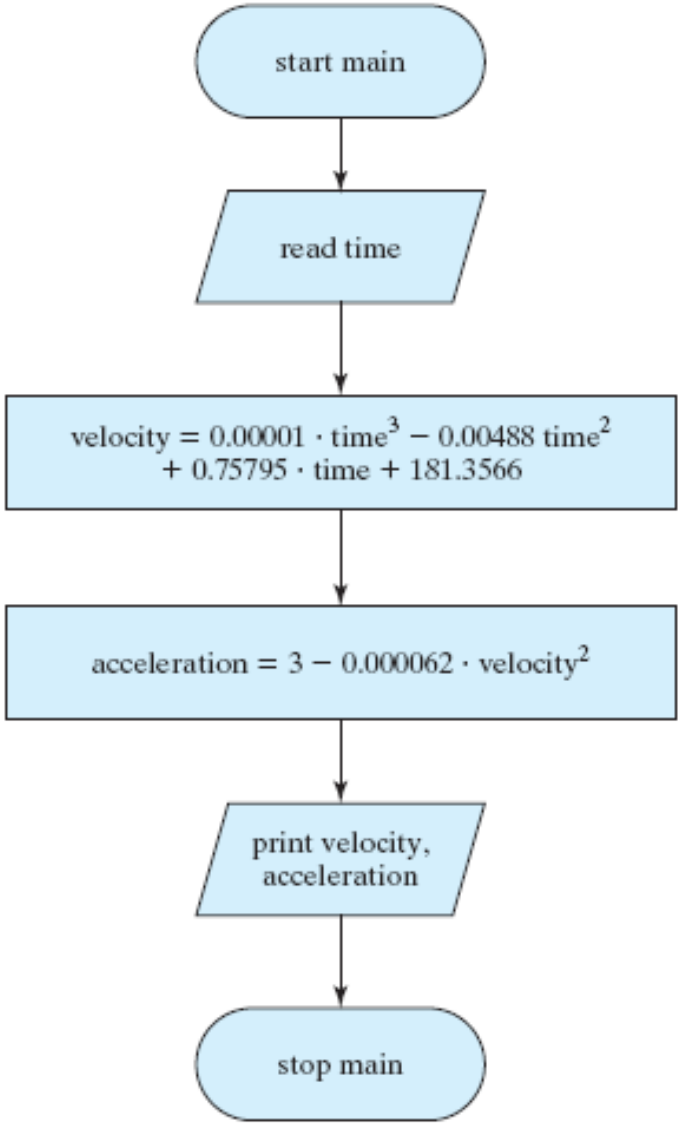


Figure 3.2 *Flowchart for open-rotor problem solution from Section 2.10.*

Structured Programming

- **Selection**

A selection structure contains a condition that can be evaluated as either **true** or **false**:

If the condition is true one statement(s) are executed;
if the condition is false another statement(s) are executed

For example suppose we have values for a division operation. Before we compute the division we want to make sure that the denominator is not zero

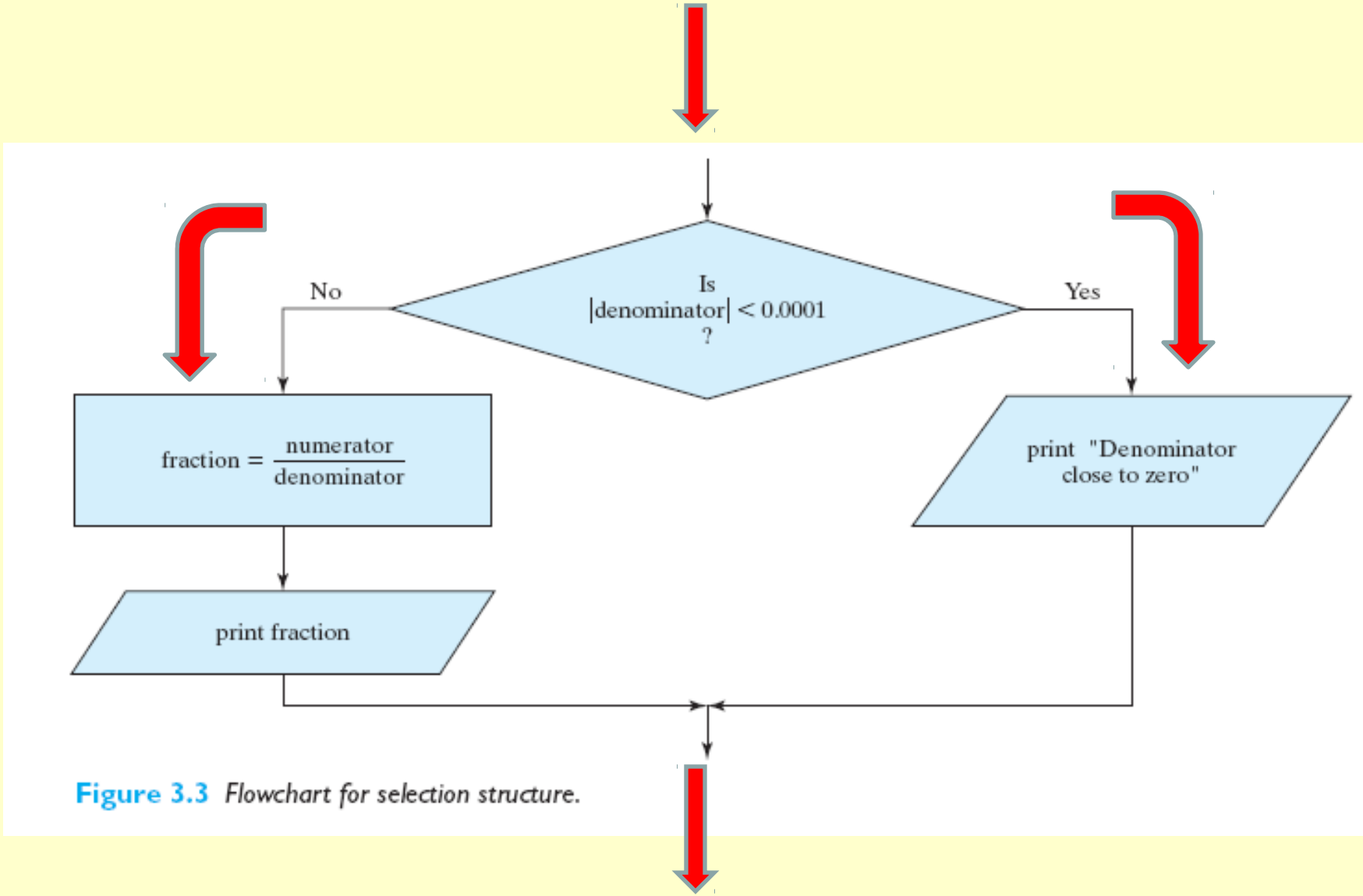


Figure 3.3 *Flowchart for selection structure.*

Structured Programming

- **Repetition**

The repetition structure allows us to repeat (or loop through) a set of steps as long as a condition is true

For example we might want to compute a set of velocity values that correspond to different times, i.e. 0, 1, 2,... 10 seconds.

We do not want to develop a sequential structure that has a statement to compute the velocity for time of 0, another for time of 1, and so on.

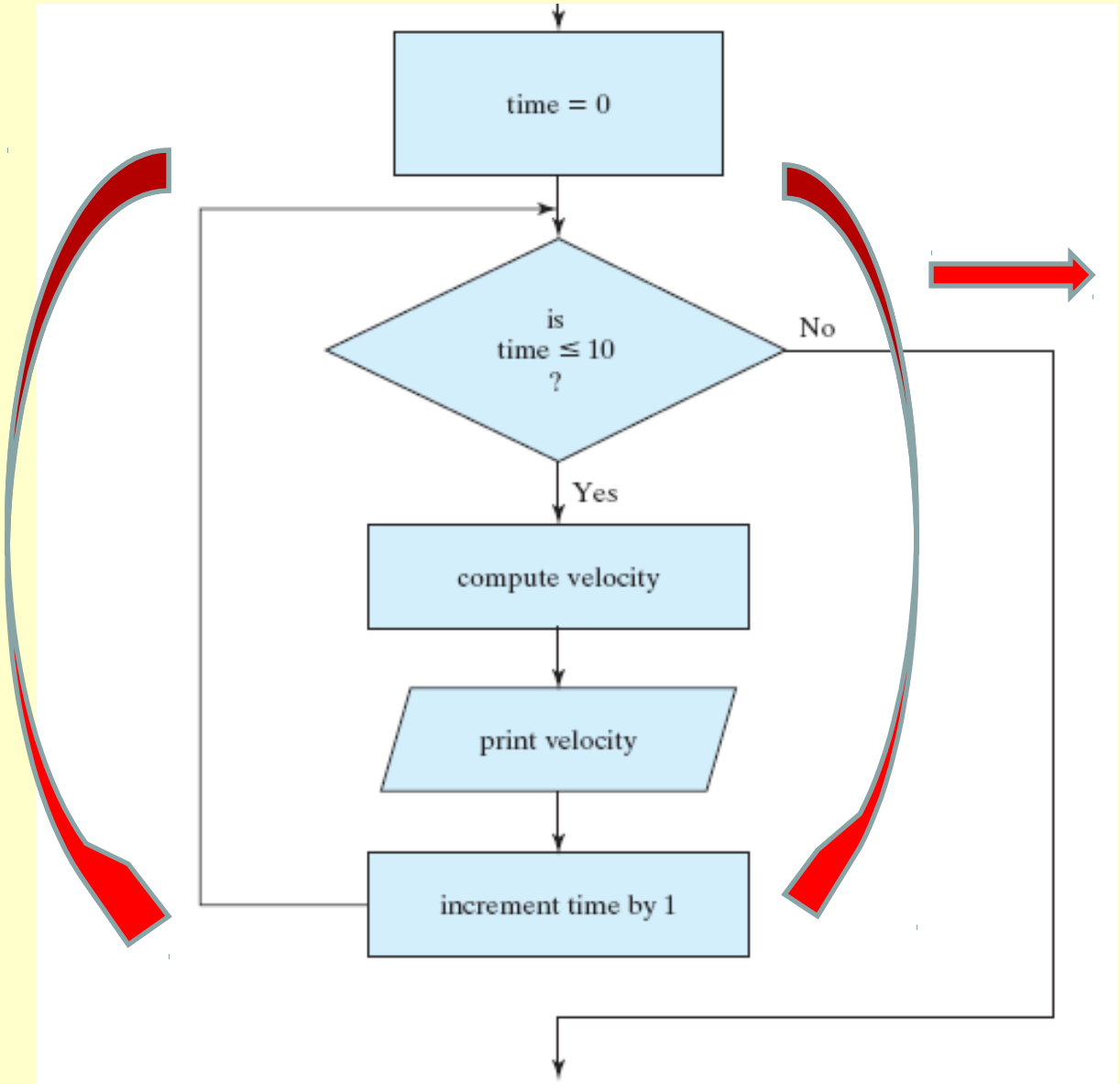


Figure 3.4 *Flowchart for repetition structure.*

Conditional Expressions

- **Relational Operators**

The relational operators that can be used to compare two expressions in C are:

<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Conditional Expressions

•In C, a true condition is assigned a value of 1 and a false condition is assigned a value of zero. Therefore, the following statement is valid:

```
d = b > c;
```

If b>c, the the value of d is 1; otherwise the value of d is zero.

Consider the statement:

```
if(a)
count++;
```


If the condition value is zero, then the condition is

Logical Operators

- Logical operators compare conditions, not expressions. C supports three logical operators:

and	&&
or	||
not	!

- For example consider the following condition:

```
a<b && b < c
```

This condition is read “a is less than be, and b is less than c”. If a=1, b=5, c=8, then the condition is true.

Simple if Statements

- An if statement allows us to test condition and them perform statements based on whether the conditions are true or false

- The simple if statement has the form:

```
If(condition) {
        statement 1;
        statement 2;
                ...
}
```

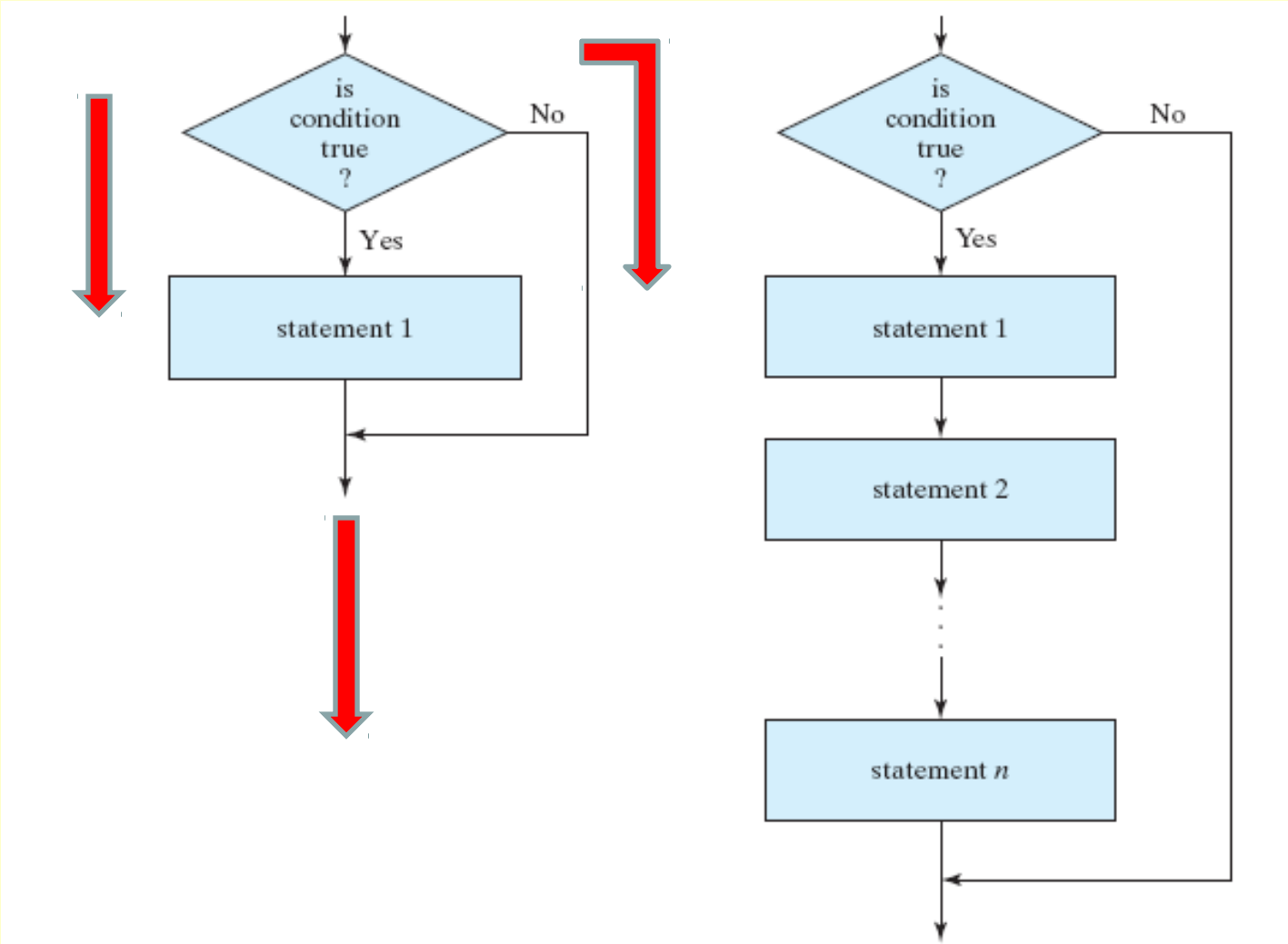


Figure 3.5 *Flowcharts for selection statements.*

If/else Statements

- An if/else statement allows us to execute one set of statements if a condition is true and a different set if the condition is false

- The if/else statement has the form:

```
If(condition) {
        statement 1;
        statement 2;
}
else
{
        statement 3;
        statement 4;
{
```

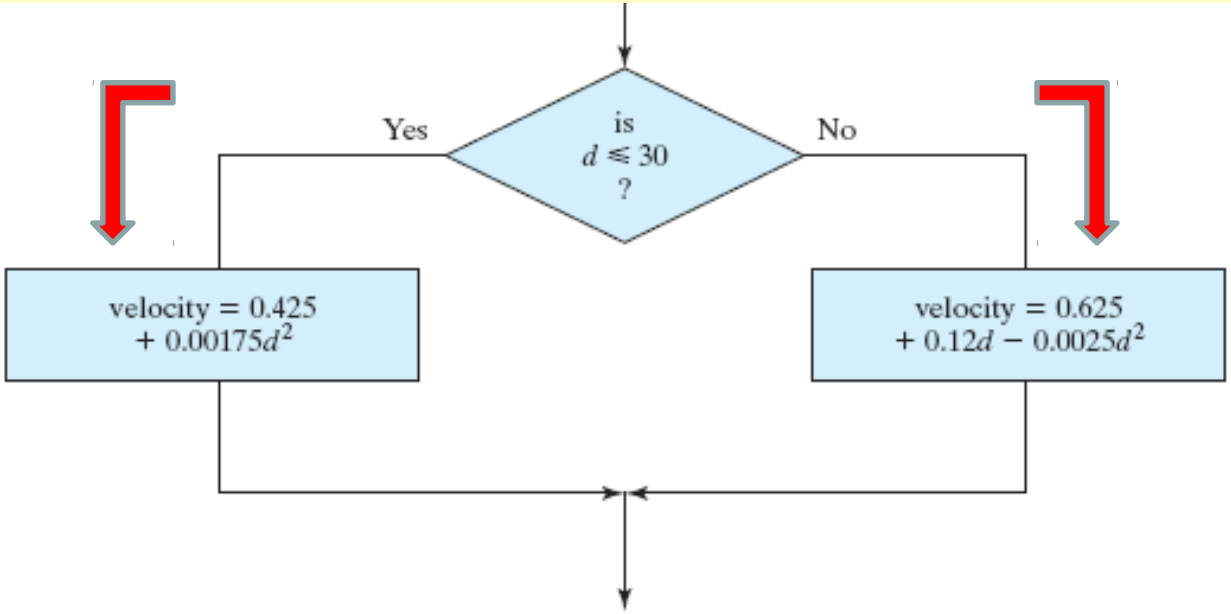


Figure 3.6 *Flowchart for* `if/else` *statement.*

Switch Statement

- The switch statement is used for multiple-selection decision making

- The switch statement has the form:

```
switch(condition) {
        case label_1:
                statement s;
        case label_2:
                statement s;
                ...
        default:
                statement s;
{
```

Problem Solving Applied: Face Recognition

One technique for comparing faces uses ratios of distances between key points on a face

distar

distar

chin

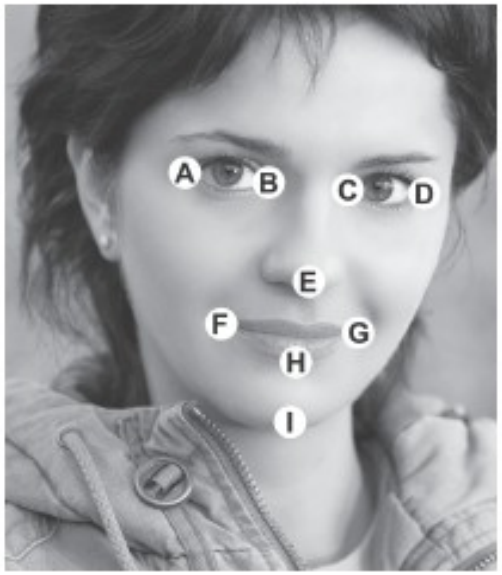


Figure 3.7 *Key points for face recognition.*

The ratios might include

between eyes divided by

between the nose and the

they c

images of

Because they are **ratios**, be computed from different sizes.

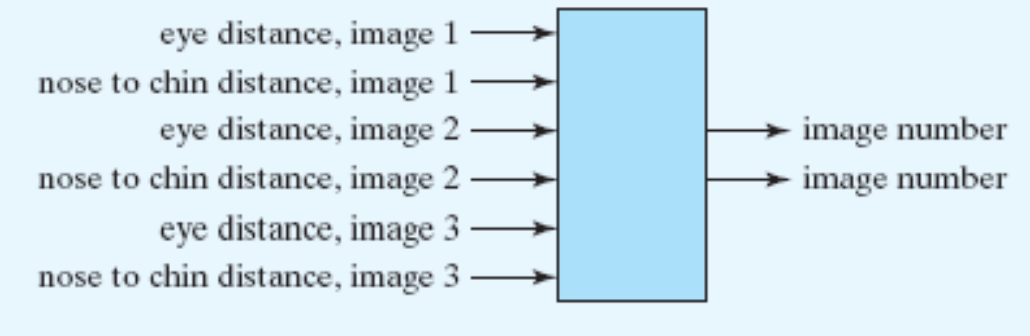
There are other challenges if the

Problem Solving Applied: Face Recognition

1.Problem Statement:

Given information on three faces, use ratios to determine the two faces that are the most similar

2.Input/Output Description:



Problem Solving Applied: Face Recognition

3.Hand Example:

Assume the following distances are measured from three images (in cm):

	Image 1	Image 2	Image 3
Outer eye distance	5.7	6.0	6.0
Nose to chin distance	5.3	5.0	5.6

The ratios of outer eye distance to nose to chin are:

Ratio_1 = 5.7/5.3 = 1.08, Ratio_2 = 6.0/5.0 = 1.20

Ratio_3 = 6.0/5.6 = 1.07

Problem Solving Applied: Face Recognition

3.Hand Example:

Assume the following distances are measured from three images (in cm):

	Image 1	Image 2	Image 3
Outer eye distance	5.7	6.0	6.0
Nose to chin distance	5.3	5.0	5.6

We then compute the absolute difference, so that it is positive:

diff_1_2 = |1.08-1.20| = 0.12, diff_1_3 = |1.08-1.07| = 0.01

Problem Solving Applied: Face Recognition

3.Hand Example:

Assume the following distances are measured from three images (in cm):

We then compute the absolute difference, so that it is positive:

diff_1_2 = |1.08-1.20| = 0.12, diff_1_3 = |1.08-1.07| = 0.01

diff_2_3 = |1.20 – 1.07| = 0.13

The difference with smallest value then determines the two images that are the most similar, using these two

Problem Solving Applied: Face Recognition

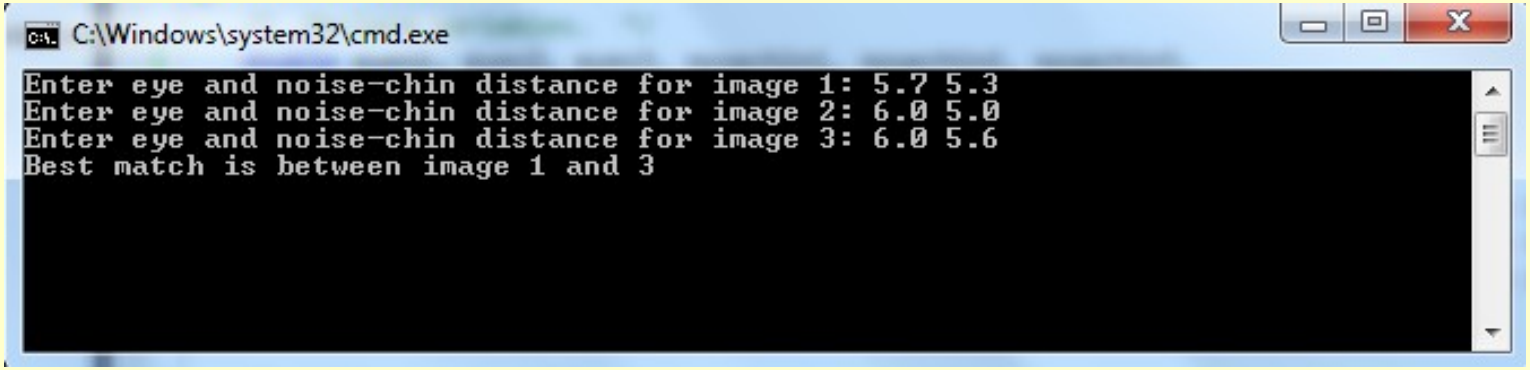
4. Algorithm Development:
   Decomposition Outline
   1. Read the distances for each image
   2. Compute the ratios for each image
   3. Compute the differences between ratios
   4. Find the minimum difference
   5. Print the corresponding image numbers as the best match

```c
/*
    Purpose: selects two face images that are similar, based on eyes and nose-chin ratios
    Input(s): outer eye and noise-chin distance
    Output(s): which faces are similar
    Written by: HE
    Date: 8/12
*/

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  Declare variables.  */
   double eyes1, eyes2, eyes3, nosechin1, nosechin2, nosechin3,
          ratio1, ratio2, ratio3, diff12, diff23, diff13;

   /*  Get user input from the keyboard.  */
   printf("Enter eye and noise-chin distance for image 1: "); scanf("%lf %lf",&eyes1,&nosechin1);
   printf("Enter eye and noise-chin distance for image 2: "); scanf("%lf %lf",&eyes2,&nosechin2);
   printf("Enter eye and noise-chin distance for image 3: "); scanf("%lf %lf",&eyes3,&nosechin3);

   /*  Compute ratios  */
   ratio1 = eyes1/nosechin1;    ratio2 = eyes2/nosechin2;   ratio3 = eyes3/nosechin3;

   /*  Compute differences  */
   diff12 = fabs(ratio1 - ratio2);  diff13 = fabs(ratio1 - ratio3); diff23 = fabs(ratio2 - ratio3);
   /*  Find and print the minimum difference   */
   if((diff12 <= diff13) && (diff12 <= diff23)) printf("Best match is between image 1 and 2 \n");
   if((diff13 <= diff12) && (diff13 <= diff23)) printf("Best match is between image 1 and 3 \n");
   if((diff23 <= diff13) && (diff23 <= diff12)) printf("Best match is between image 2 and 3 \n");
   getch();

   /*  Exit program.  */
   return 0;
}
/*-------------------------------------------------------------*/
```

Problem Solving Applied: Face Recognition

5.Testing

The answer matches the hand example, so we can then test the program with additional lenghts:



The answer matches the hand example

Loop Structures

- Loops are used to implement repetitive structures. C contains three different loop structures:

1.while loop
2.do/while loop
3.for loop

- In addition C allows the use of two statements that modify the flow of the loop structures:
- The **break** statement which causes the execution of the program to immediately break the loop structure
- The **continue** statement which cause the execution of the current loop to continue

Loop Structures

1.while loop

The general form of a while loop is as follows:

```
while(condition) {
    statement 1;
    statement 2;
        ...
}
```

Loop Structures

2. **do/while** loop

The general form of a do/while loop is as follows:

```
do {
    statement 1;
    statement 2;
        ...
} while (condition);
```

Loop Structures

3. **for** loop

The general form of a while loop is as follows:

```
for( expression 1 ;  expression 2 ;  expression 3 ) {
    statement 1;
    statement 2;
        ...
}
```

Where **expression 1** is used to initialize the **loop-control variable**

expression 2 specifies the condition that should be true to continue the loop repetition

expression 3 specifies the modification to the loop-control

Data Files

•Engineering problem solutions often involve large amounts of data. These data can be **output data** generated by the program, or **input data** that are used by the program

•It is not generally feasible to print large amounts of data on the screen, or read large amounts of data from the keyboard

I/O Statements

Each data file must have a file points associated with it. A file points is defined with the FILE declaration:

```
FILE *sensor 1;
```

Data Files

•After a file pointer is defined, it must be associated with a file name. The **fopen** function obtains the information need to relate the file pointer to the file:

```
sensor = fopen(“sensor1.txt”,”r”);
```

•To be sure that programs find data files, it is a good practice to whether a file exists before opening it

```
file1 = fopen(FILENAME,”r”);
if(file1 == NULL) printf(“Error opening file \n”);
else {
    ...
}
```

Data Files

- We can now read information from data files using the **fscanf** statement:

```
fscanf(sensor,”%lf %lf”, &t , &motion );
```

- Or write information to data files using the **fprinf** statement:

```
fprinf(sensor,”%f %f \n”, t , motion);
```

- The **fclose** statement is used to close a file after reading/writing to it:

```
fclose(sensor);
```

Reading Data Files

•In order to read information form a data file, we MUST know some details about the file:
1.Name
2.Order of data
3.Data type
4.Special information in the file?

•Data files generally have three common structures:
1.First line contains the number of lines (called records), i.e. 150 (if there are 150 records in file)
2.Special signals (called trailer or sentinel signal) that signal the last record, i.e. -999.0 (if all number in the file are positive)
3.Use an End-of-File indicator which is inserted at the end of every file

Numerical Technique: Linear Modeling

- Linear modeling is used to find the coefficients of a linear equation that is the best fit to a set of data points in terms of minimizing the sum of squared distances between the line and the data points

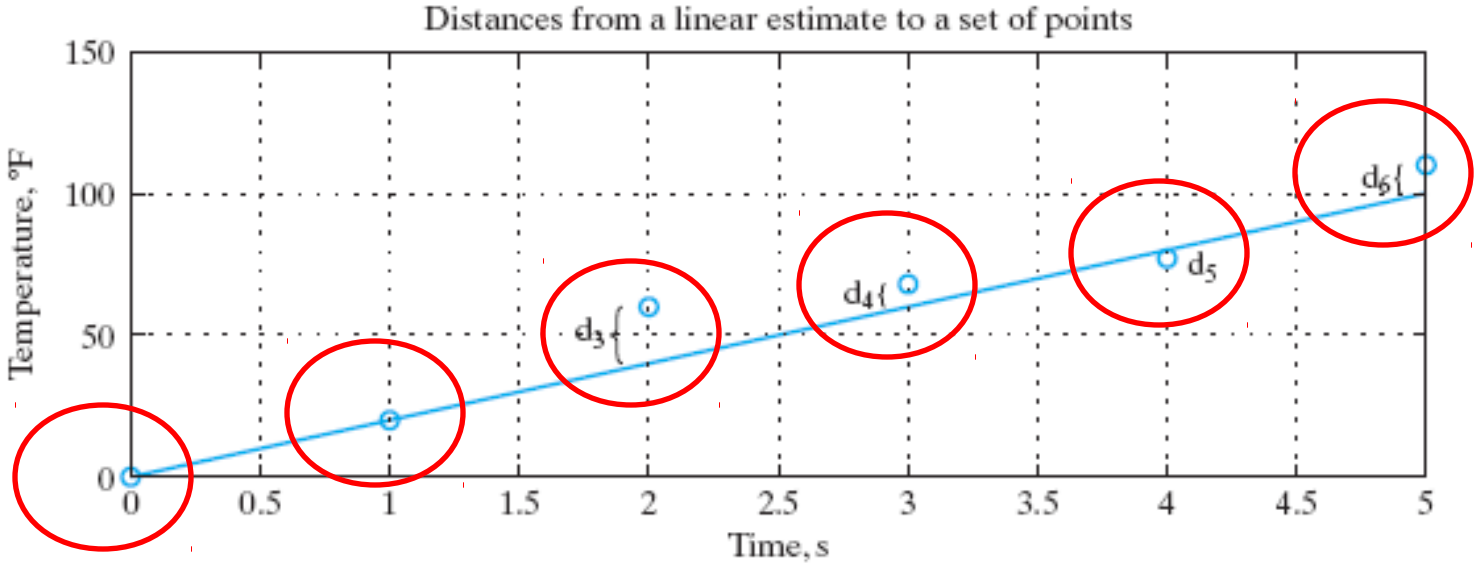


Figure 3.13 *Distances between points and the linear estimate.*

Numerical Technique: Linear Modeling

- The best linear fit for the example at hand is:

$$y = mx + b = 20.83x + 3.76$$

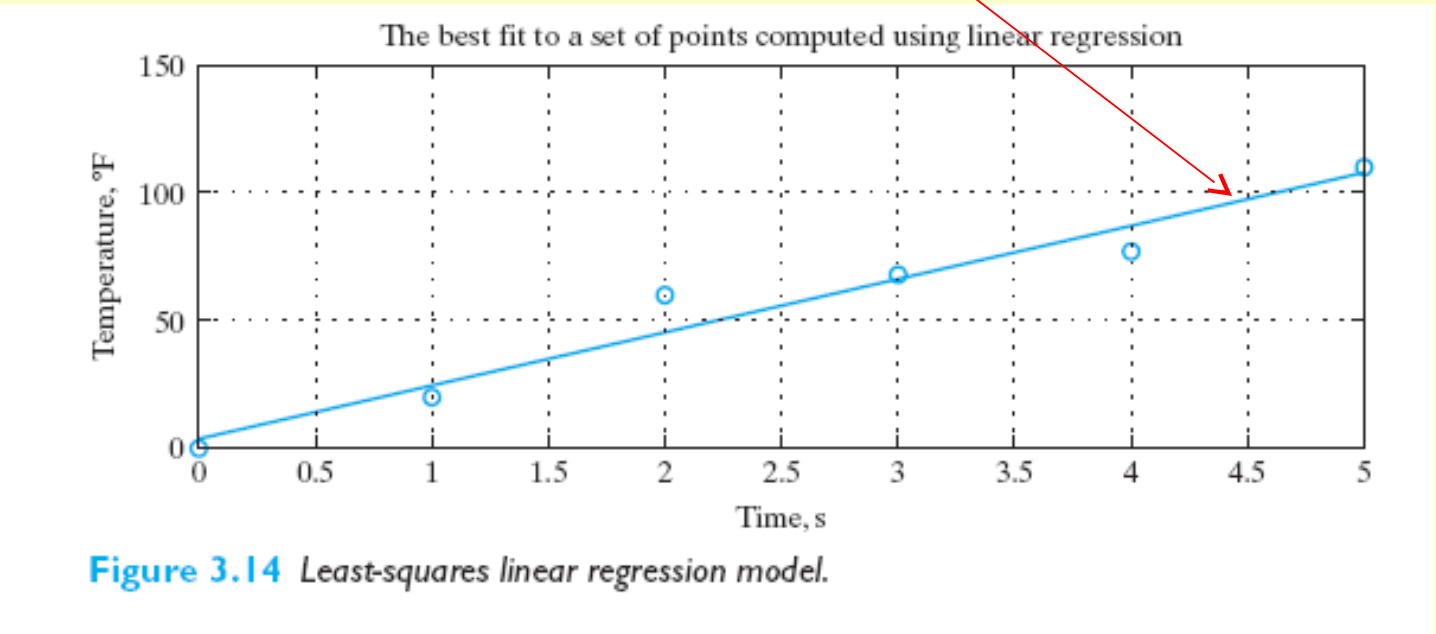


Figure 3.14 *Least-squares linear regression model.*

Numerical Technique: Linear Modeling

•With an equation model, we can compute estimates that we could not compute with linear interpolation. For example we can compute an estimate for temperature at t = 3.3 seconds, and t = 8.0 seconds (this is called extrapolation)

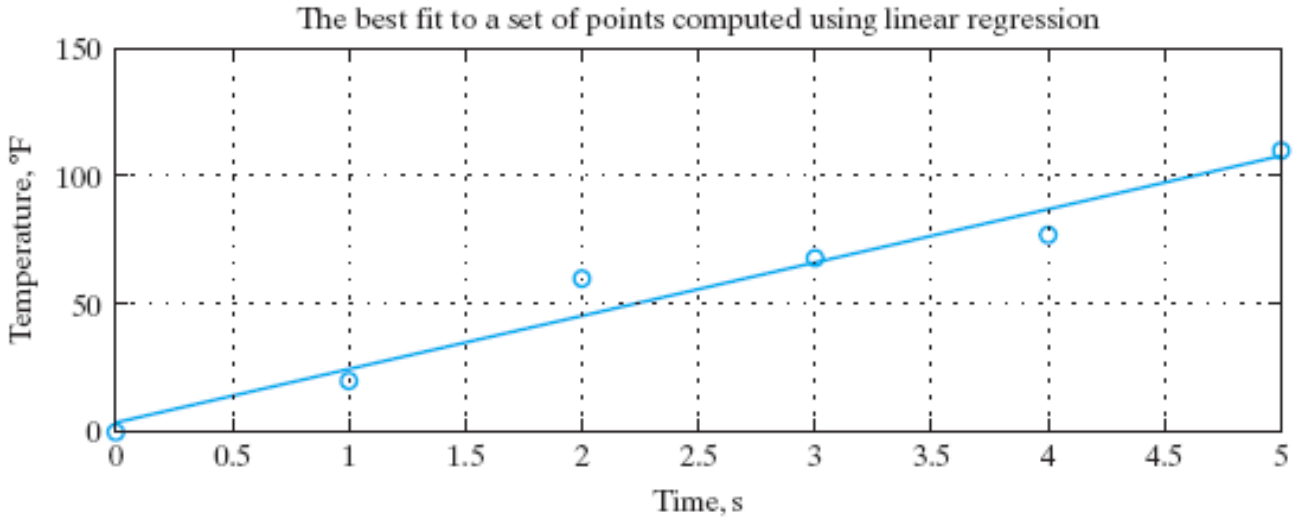


Figure 3.14 *Least-squares linear regression model.*

Numerical Technique: Linear Modeling

- With an equation model, we can compute estimates that we could not compute with linear interpolation. For example we can compute an estimate for temperature at t = 3.3 seconds, and t = 8.0 seconds (this is called extrapolation)

Numerical Technique: Linear Modeling

- We find the **slope** and **intercept** for the best linear fit to a set of n data points in a least squares sense using the following equations:

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2},$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}.$$

Problem Solving Applied: Ozone Measurements

•Satellite sensors can be used to measure many different pieces of information that help us understand more about the atmosphere, which is composed of layers around the earth.

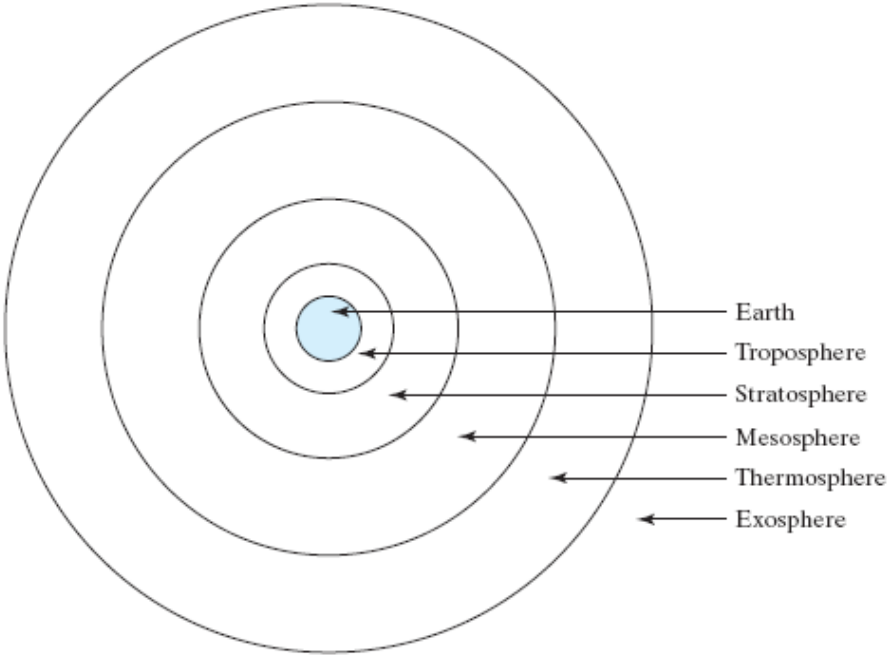


Figure 3.15 *Atmospheric layers around the earth.*

Problem Solving Applied: Ozone Measurements

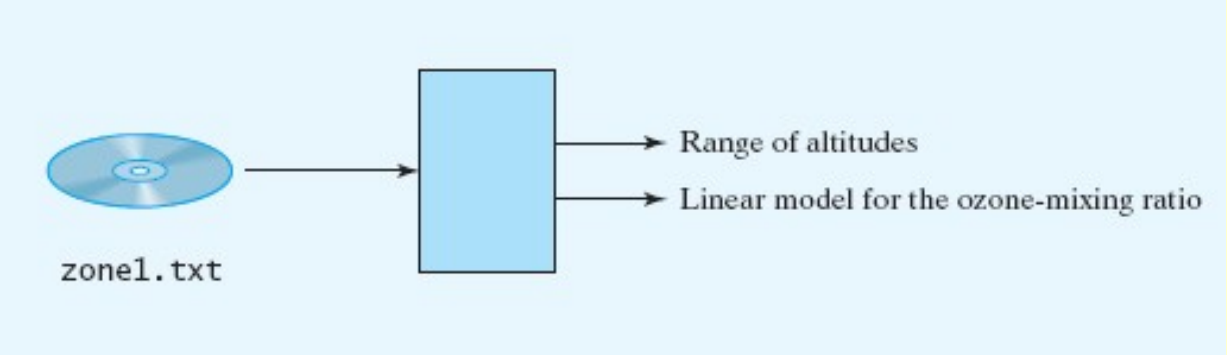
1.Problem Statement:

Consider a set of data measuring the ozone-mixing ratio in parts per million volume (ppmv). The data is nearly linear, and we can use a linear model to estimate the ozone at altitudes that then the ones for which we have data.

Use the least-squares technique to determine a linear model for estimating the ozone-mixing ratio at a specified altitude

Problem Solving Applied: Ozone Measurements

2.Input/Output Description:



3.Hand example:

Altitude (km)	Ozone Mixing Ratio (ppmv)
20	3
24	4
26	5
28	6

Problem Solving Applied: Ozone Measurements

- We need to evaluate the following equations:

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2},$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}.$$

- We compute the values:

m = 0.37 and b = -4.6

Problem Solving Applied: Ozone Measurements

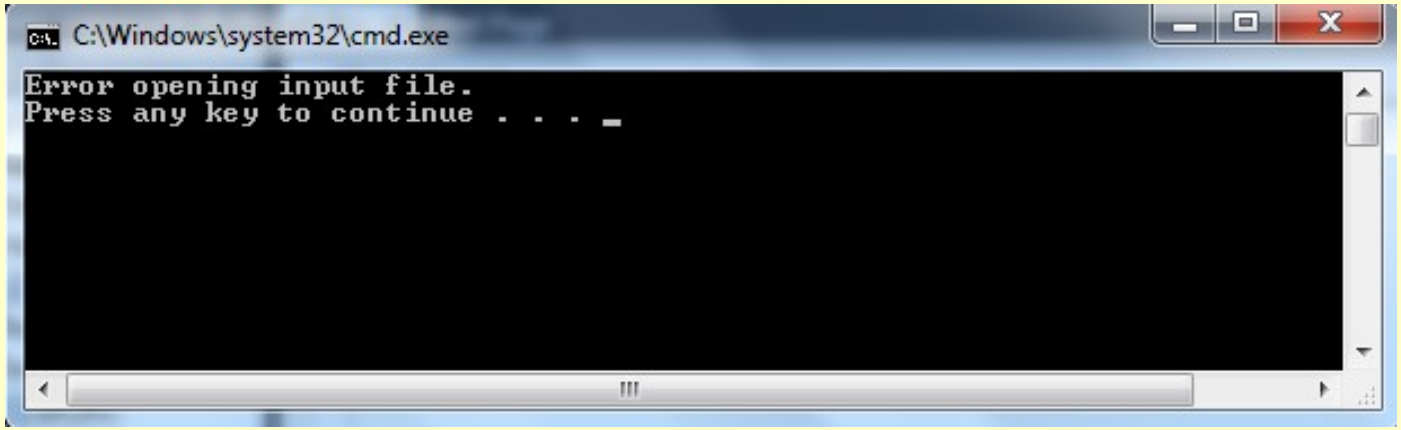
4. Algorithm Development:
   Decomposition Outline
   1. Read data file values
   2. Compute the slope and y-intercept
   3. Print range of altitudes and linear model

```c
#include <stdio.h>
#define FILENAME "zone1.txt"

int main(void)
{
   /*  Declare and initialize variables.  */
   int count=0;
   double x, y, first, last, sumx=0, sumy=0, sumx2=0, sumxy=0, denominator, m, b;
   FILE *zone;

   /*  Open input file.  */
   zone = fopen(FILENAME,"r");
   if (zone == NULL) printf("Error opening input file.  \n");
   else
   {
      /*  While not at the end of the file,  */
      /*  read and accumulate information.   */
      while ((fscanf(zone,"%lf %lf",&x,&y)) == 2)
      {
         ++count;
         if (count == 1) first = x;
         sumx += x; sumy += y; sumx2 += x*x; sumxy += x*y;
      }
      last = x;

      /*  Compute slope and y-intercept.  */
      denominator = sumx*sumx - count*sumx2;
      m = (sumx*sumy - count*sumxy)/denominator;
      b = (sumx*sumxy - sumx2*sumy)/denominator;

      /*  Print summary information.  */
      printf("Range of altitudes in km: \n");
      printf("%.2f to %.2f \n\n",first,last);
      printf("Linear model: \n");
      printf("ozone-mix-ratio = %.2f altitude + %.2f \n",m,b);

      /*  Close file.  */
      fclose(zone);
   }

   /*  Exit program.  */
   return 0;
}
/*--------------------------------------------------------------*/
```

Problem Solving Applied: Ozone Measurements

5.Testing

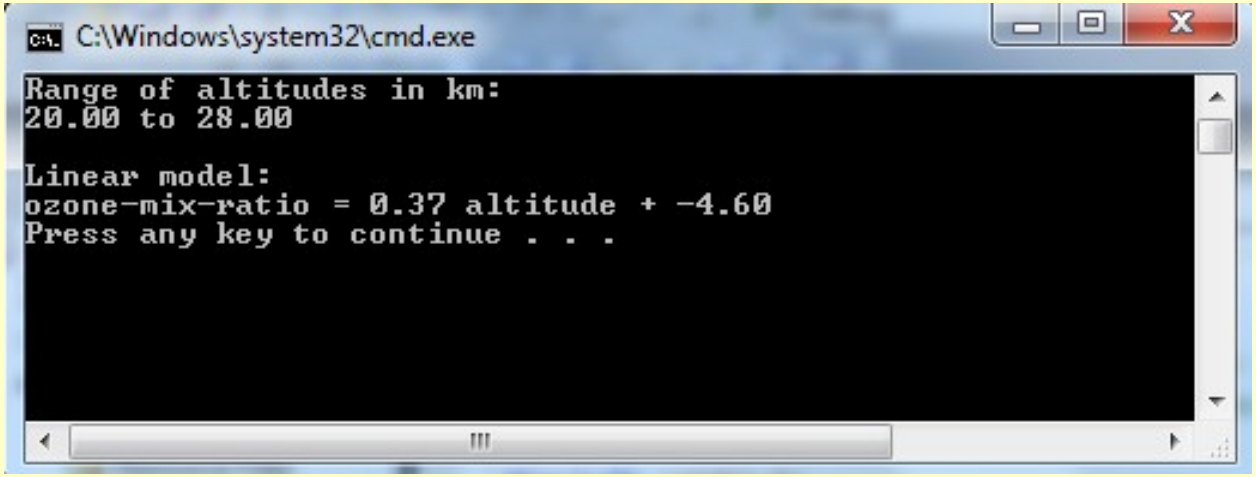
If we do not define a zone1.txt file, we should get an error message indicating that we could not open it



Problem Solving Applied: Ozone Measurements

5.Testing

Using the data from the hand example as the contents of the file zone1.txt, we get the following output:



The answer matches the hand example

Homework on Chapter 3 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due in a week