

CHAPTER 4

MODULAR PROGRAMMING WITH FUNCTIONS

Modularity

- A program may also contain other functions, and it may refer to functions in another file or in a library. These **functions**, or **modules**, are sets of statements that perform an operation or compute a value
- To maintain simplicity and readability in long and complex programs, we use a short main, and other functions instead of using one long main function.
- By separating a solution into a group of modules, each module is easier to understand, thus adhering to the basic guidelines of structured programming

Modularity

- Braking a problem into a set of modules has many advantages:
 1. Every module can be written and tested separately from the rest of the program
 2. A module is smaller than a complete program, so testing is easier
 3. Once a module has been tested, it can be used in new program without having to retest it (**reusability**)
 4. Use of modules (**modularity**) usually reduces the overall length of programs
 5. Several programmers can work on the same project if it is separated into modules

Modularity

Main

Modules

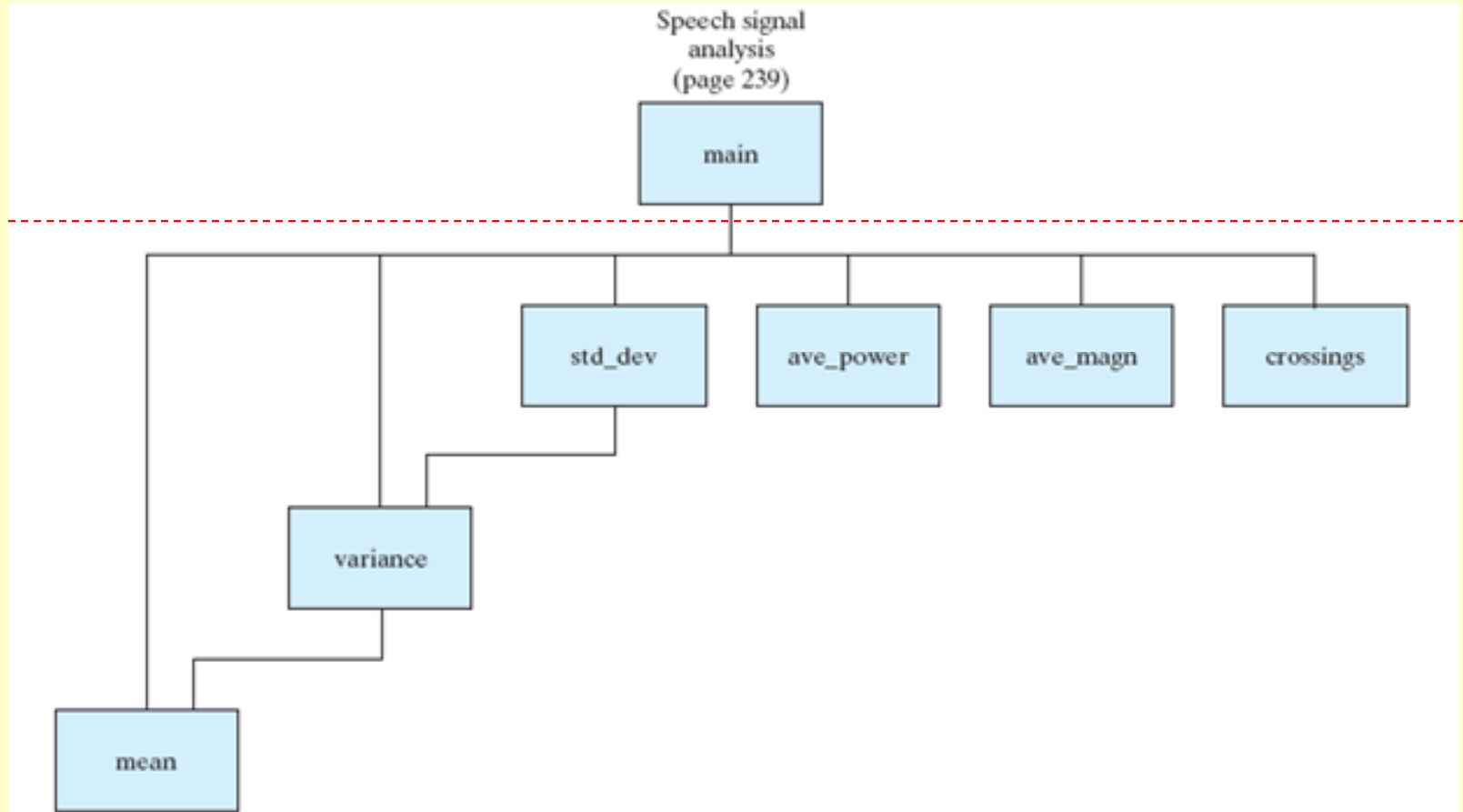


Figure 4.1 Examples of structure charts.

Function Definition

- A function consists of a definition statement followed by declarations and statements. The general form of a function is:

```
return_type function_name(parameter_declarations) {  
    declarations;  
    statements;  
    return expression;  
}
```

- The parameter declarations represent the information passed to the function
- Additional variables used by the function are defined in declarations statement
- All functions should include a return statement

Storage Class and Scope

- The variables can be declared within the main function or user-defined functions are called **local variables**
- Variables that are declared outside the main program or user-defined functions are called **global variables**
- A local variable has a value when the function is being executed, but its value is not retained when the function is completed
- A global variable can be accessed by any function within the program

Problem Solving Applied: Computing the Boundaries of the Iris

- Most iris recognition techniques start with a segmentation operation that identifies the iris/pupil boundary and boundary between iris and sclera (the white of the eye)

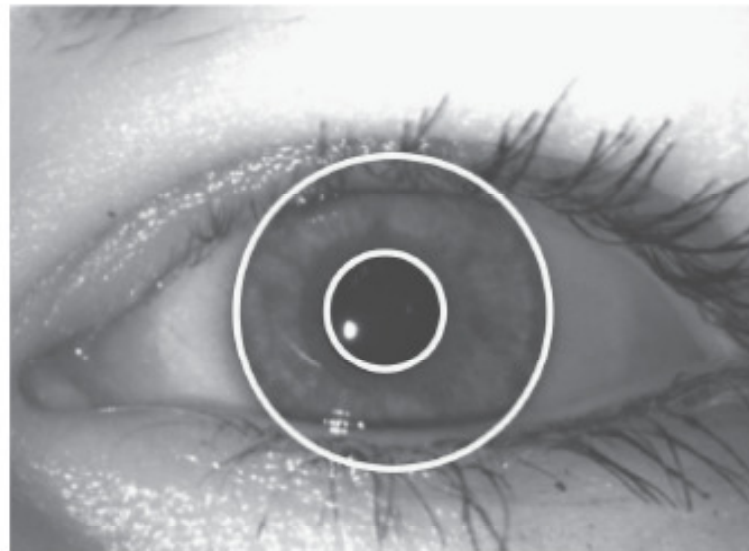


Figure 4.4 *Image of an eye with iris boundaries identified.*

Problem Solving Applied: Computing the Boundaries of the Iris

- Segmentation is a very complex process to carry out automatically with a computer algorithm. To simplify the process we will have a user click on three points on the pupil boundary and three points on the boundary between the iris and the sclera.
- With three points the computer can compute the equation of a circle, therefore we can compute the equation of the circle through these points, and the location of the center of the circle

Problem Solving Applied: Computing the Boundaries of the Iris

- The technique we will use to find the equation of a circle is based on finding the equation of a line through points P_1 and P_2 , and the equation of the line through points P_2 and P_3
- The lines perpendicular to the two line segments (P_1P_2 and P_2P_3) intersect in the center of the circle

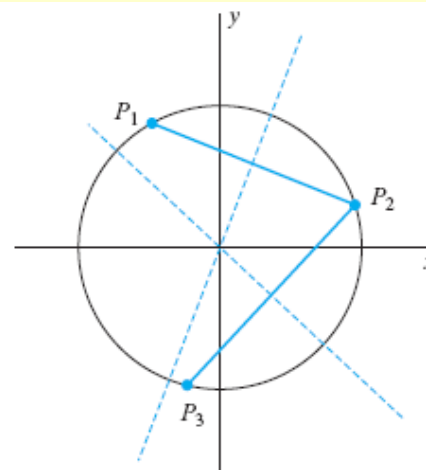


Figure 4.5 Relationship between points on a circle and the center.

Problem Solving Applied: Computing the Boundaries of the Iris

- Here are the equations needed for this program

- Slope of line P1P2

$$m_{12} = (y_2 - y_1) / (x_2 - x_1)$$

- Slope of line P2P3

$$m_{23} = (y_3 - y_2) / (x_3 - x_2)$$

- Equation of line P1P2

$$y_{12} = m_{12}(x - x_1) + y_1$$

Problem Solving Applied: Computing the Boundaries of the Iris

- Equation of line P2P3

$$y_{23} = m_{23}(x-x_2)+y_2$$

- Equation of line perpendicular to line P1P2 that bisects the line segment

$$y_{p12} = (-1/m_{12})(x-(x_1+x_2)/2)+((y_1+y_2)/2)$$

- Equation of line perpendicular to line P2P3 that bisects the line segment

$$y_{p23} = (-1/m_{23})(x-(x_2+x_3)/2)+((y_2+y_3)/2)$$

Problem Solving Applied: Computing the Boundaries of the Iris

- Equation for x coordinate of the center of circle

$$x_c = (m_{12}m_{23}(y_1 - y_3) + m_{23}(x_1 + x_2) - m_{12}(x_2 - x_3)) / (2(m_{23} - m_{12}))$$

- Equation for y coordinate of the center of circle

$$y_c = (-1/m_{12})(x_c - (x_1 + x_2)/2) + (y_1 + y_2)/2$$

- Equation for radius of the circle

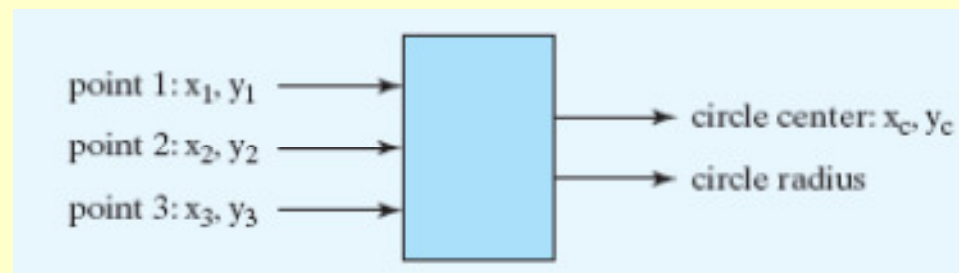
$$r = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2}$$

Problem Solving Applied: Computing the Boundaries of the Iris

1. Problem Statement:

Given three points in a plane, determine the coordinates of the center of the circle and the radius of the circle that contains the three points

2. Input/Output Description:



Problem Solving Applied: Computing the Boundaries of the Iris

3. Hand example:

Using the equations provided earlier in this section, we now compute the center of the corresponding circle and its radius. Assume we start with three points:

$$P1 = (-1,-1), \quad P2 = (1,1), \quad P3 = (3,-1)$$

The slopes for lines P1P2 and P2P3 are:

$$m_{12} = \frac{2}{2} = 1, \quad m_{23} = \frac{-2}{2} = -1$$

Problem Solving Applied: Computing the Boundaries of the Iris

3. Hand example:

The equations for the line P1P2 and P2P3 are:

$$y_{12} = x, \quad y_{23} = -x + 2$$

We can now compute the equations for the perpendicular lines through P1P2 and P2P3 that bisect the line segments:

$$y_{p12} = -x, \quad y_{p23} = x - 2$$

The coordinates of the center point are:

$$x_c = 1, \quad y_c = -1$$

Problem Solving Applied: Computing the Boundaries of the Iris

3. Hand example:

The radius of the circle is:

$$r = 2$$

The equation for the circle is:

$$(x - 1)^2 + (y + 1)^2 = 4$$

Problem Solving Applied: Computing the Boundaries of the Iris

4. Algorithm Development:

Because there are several equations to evaluate to compute the coordinates of the center of the circle, this is a good candidate for using different function(s)

Decomposition Outline of the main program

1. Read the coordinates of the three points
2. Determine the x and y coordinate of the center of the circle
3. Compute the radius of the circle
4. Print the coordinates of the center of the circle and radius

Problem Solving Applied: Computing the Boundaries of the Iris

4. Algorithm Development:

Decomposition Outline of a function

1. Compute the equations for the lines connecting the points
2. Compute the equations for the lines perpendicular to the lines connecting the three points
3. Compute the x coordinate of the center of the circle (the y coordinate will be computed in the main)

Problem Solving Applied: Computing the Boundaries of the Iris

- The main program looks like this:

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

int main(void)
{
    /* Declare and initialize variables. */
    double x1,x2,x3,y1,y2,y3,m12,xc,yc,r;
    double circle_coord(double x1,double y1,double x2,double y2,double x3,double y3);

    /* Get user input form keyboard*/
    printf("Enter x,y coordinates for P1: "); scanf("%lf %lf",&x1,&y1);
    printf("Enter x,y coordinates for P2: "); scanf("%lf %lf",&x2,&y2);
    printf("Enter x,y coordinates for P3: "); scanf("%lf %lf",&x3,&y3);

    /* Use a function to compute x coordinate of the center */
    xc = circle_coord(x1,y1,x2,y2,x3,y3);

    /* Compute the y coordinate of the center and radius of circle */
    m12 = (y2-y1)/(x2-x1);  yc = -(1/m12)*(xc-(x1+x2)/2)+(y1+y2)/2;
    r = sqrt((x1-xc)*(x1-xc)+(y1-yc)*(y1-yc));

    /* Print circle parameters */
    printf("\n Center of circle: (%.1f,%.1f) \n",xc,yc);
    printf("Radius of circle: %.1f \n",r); getch();

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

Problem Solving Applied: Computing the Boundaries of the Iris

- The function looks like this:

```
double circle_coord(double x1,double y1,double x2,double y2,double x3,double y3)
{
    /* Declare variables */
    double m12, m23, xc_num, xc_den, xc;

    /* Compute the slopes of the lines between points */
    m12 = (y2-y1)/(x2-x1); m23 = (y3-y2)/(x3-x2);

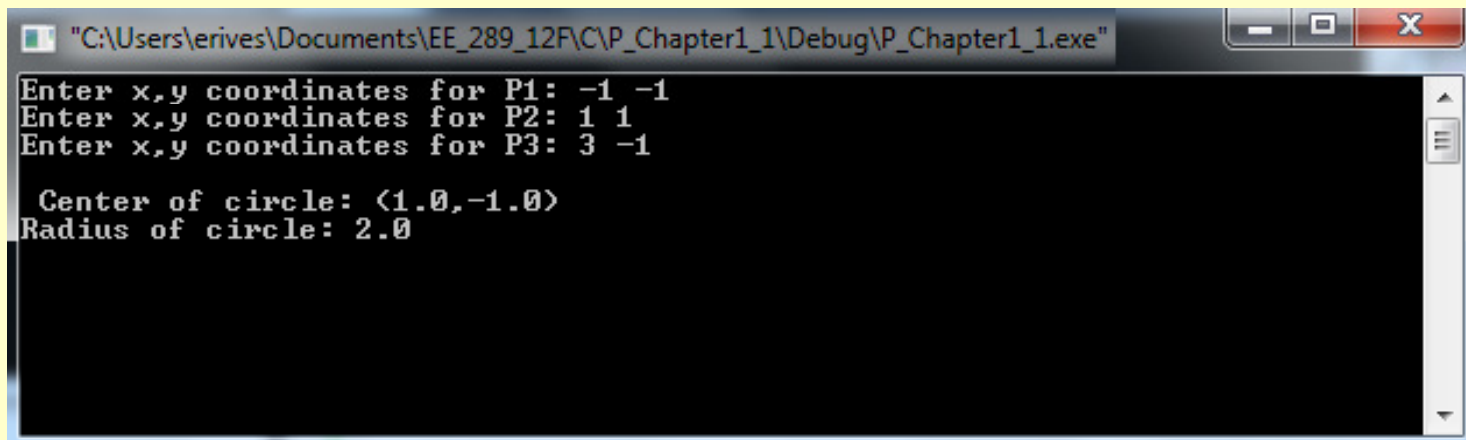
    /* Compute the x coordinate of the center of the circle */
    xc_num = m12*m23*(y1-y3)+m23*(x1+x2)-m12*(x2+x3);
    xc_den = 2*(m23-m12);
    xc = xc_num/xc_den;

    /* Return x coordinate of center */
    return xc;
}
```

Problem Solving Applied: Computing the Boundaries of the Iris

5. Testing

We test the program with the data from the hand example:



```
"C:\Users\erives\Documents\EE_289_12F\C\P_Chapter1_1\Debug\P_Chapter1_1.exe"  
Enter x,y coordinates for P1: -1 -1  
Enter x,y coordinates for P2: 1 1  
Enter x,y coordinates for P3: 3 -1  
  
Center of circle: (1.0,-1.0)  
Radius of circle: 2.0
```

The answer matches the hand example

Macros

- Before compiling a program, the preprocessor performs any actions specified by preprocessing directives, such as inclusions of header files
- A simple operation can also be specified by a preprocessing directive called a **macro**

```
#define macro_name(parameters) macro_text
```

The `macro_text` replaces references to the `macro_name` in the program.

- The macro can represent a simple function.

Macros

- Consider the following simple program:

```
#include <stdio.h>
#define degrees_C(x) (((x)-32)*(5.0/9.0))

int main(void) {
    /* Declare variables */ ...
    /* Get temperature in Fahrenheit */ ...

    /* Convert and print temp in Centigrade */
    printf(“%f Centigrade \n”,degrees_C(temp));

    /* Exit program */ ...
}
```

Recursion

- A function that invokes itself (or calls itself) is a **recursion function**. Recursion can be a powerful tool for solving certain classes of problems.
- A simple example of recursion can be shown using the factorial computation. Recall that factorial is defined as

$$k! = (k)(k-1)(k-2)\dots(3)(2)(1)$$

Where K is a nonnegative integer, and where $0! = 1$

- So $5! = 5*4*3*2*1 = 120$ or
 $5! = 5*4!$ And $4! = 4*3!$ And $3! = 3*2!$ And $2! = 2*1!$ And $1! = 1*0!$
and finally $0! = 1$

Recursion

- Consider the following simple program used to compute the factorial of a number recursively

```
// Purpose: This program computes a factorial
// Input(s): a positive number
// Output(s): factorial of a number
// Written by: He

#include <stdio.h>

int main(void)
{
    /* Declare and initialize variables. */
    int n;
    long factorial_r(int k);

    /* Get user input form keyboard*/
    printf("Enter a positive number: "); scanf("%i",&n);

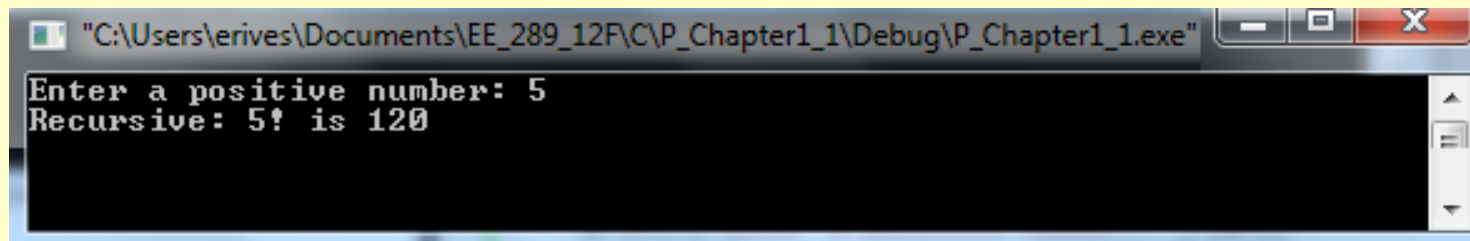
    /* Compute and print factorial */
    printf("Recursive: %i! is %li \n",n,factorial_r(n)); getch();

    /* Exit program. */
    return 0;
}

/* This function computes a factorial recursively */
long factorial_r(int k)
{
    if (k==0) return 1;
    else return k*factorial_r(k-1);
}
```

Recursion

- The output of the program is shown below for the hand example of 5!



```
"C:\Users\erives\Documents\EE_289_12F\C\P_Chapter1_1\Debug\P_Chapter1_1.exe"
Enter a positive number: 5
Recursive: 5! is 120
```