Functions

I. Standard math functions
II. User defined functions
III. Scope
IV. Case studies
V. Common Errors

Acknowledgment: These slides are based on Dr Obada Al Khatib's ones.

I. Standard `math` functions

TABLE 3.1 Some Mathematical Library Functions

Function	Standard Header File	Purpose: Example	Argument(s)	Result
`abs(x)`	`<stdlib.h>`	Returns the absolute value of its integer argument: if `x is -5`, `abs(x)` is 5	`int`	`int`
`ceil(x)`	`<math.h>`	Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0`	`double`	`double`
`cos(x)`	`<math.h>`	Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0`	`double` (radians)	`double`
`exp(x)`	`<math.h>`	Returns e^x where e = `2.71828...`: if `x` is `1.0`, `exp(x)` is `2.71828`	`double`	`double`
`fabs(x)`	`<math.h>`	Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432`	`double`	`double`
`floor(x)`	`<math.h>`	Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0`	`double`	`double`
`log(x)`	`<math.h>`	Returns the natural logarithm of `x` for `x > 0.0`: if `x` is `2.71828`, `log(x)` is `1.0`	`double`	`double`
`log10(x)`	`<math.h>`	Returns the base-10 logarithm of `x` for `x > 0.0`: if `x` is `100.0`, `log10(x)` is `2.0`	`double`	`double`
`pow(x, y)`	`<math.h>`	Returns x^y. If `x` is negative, `y` must be integral: if `x` is `0.16` and `y` is `0.5`, `pow(x,y)` is `0.4`	`double, double`	`double`
`sin(x)`	`<math.h>`	Returns the sine of angle `x`: if `x` is `1.5708`, `sin(x)` is `1.0`	`double` (radians)	`double`
`sqrt(x)`	`<math.h>`	Returns the nonnegative square root of `x` ($\sqrt{x}$) for `x` $\geq$ `0.0`: if `x` is `2.25`, `sqrt(x)` is `1.5`	`double`	`double`
`tan(x)`	`<math.h>`	Returns the tangent of angle `x`: if `x` is `0.0`, `tan(x)` is `0.0`	`double` (radians)	`double`

Function identifier (name)

```
pow( x, y);
```

function parameters

Data type of the returned value

Standard `math` functions

- If you need to use a math function, the `math` library must be linked to your program as follows:
  - #include <math.h>

- When you call a function, replace its formal parameters with the actual parameters in your program: *constants*, *variables*, or *expressions*

sin(x)

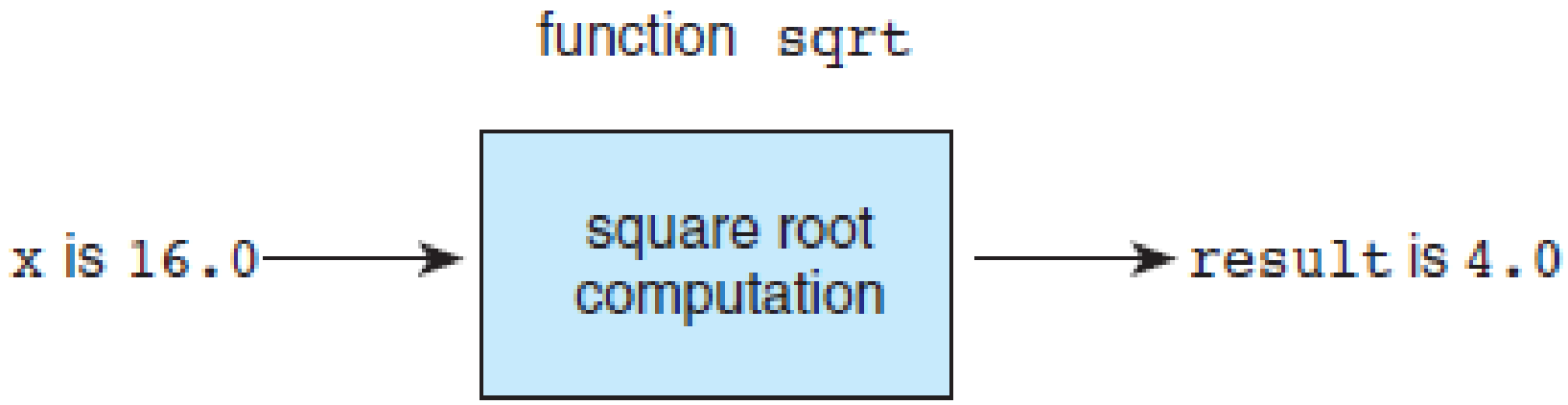
x is a formal parameter indicating that sin() requires one actual parameter

```
#include <stdio.h>
#include <math.h>
#define PI  3.1415926
int main(void)
{
      double signal, f=30.0,  t=1.0;

      signal = 5.0*sin( 1.5 );
         ….
      singal = 10.0*sin( 2.0*PI*f*t );
}        …
```

Actual function parameters Or arguments

Function sqrt(..) as a "Black Box"

- The sqrt() function is a library function computes the square root of a number.



- To use include <math.h> in your program
- Function prototype:

double sqrt(double arg);

A sqrt(..) based Program

```
1.  /*
2.   * Performs three square root computations
3.   */
4.
5.  #include <stdio.h> /* definitions of printf, scanf */
6.  #include <math.h>  /* definition of sqrt           */
7.
8.  int
9.  main(void)
10. {
11.       double first, second,    /* input - two data values        */
12.              first_sqrt,       /* output - square root of first  */
13.              second_sqrt,      /* output - square root of second */
14.              sum_sqrt;         /* output - square root of sum    */
15.
16.       /* Get first number and display its square root. */
17.       printf("Enter the first number> ");
18.       scanf("%lf", &first);
19.       first_sqrt = sqrt(first);
20.       printf("The square root of the first number is %.2f\n", first_sqrt);
```

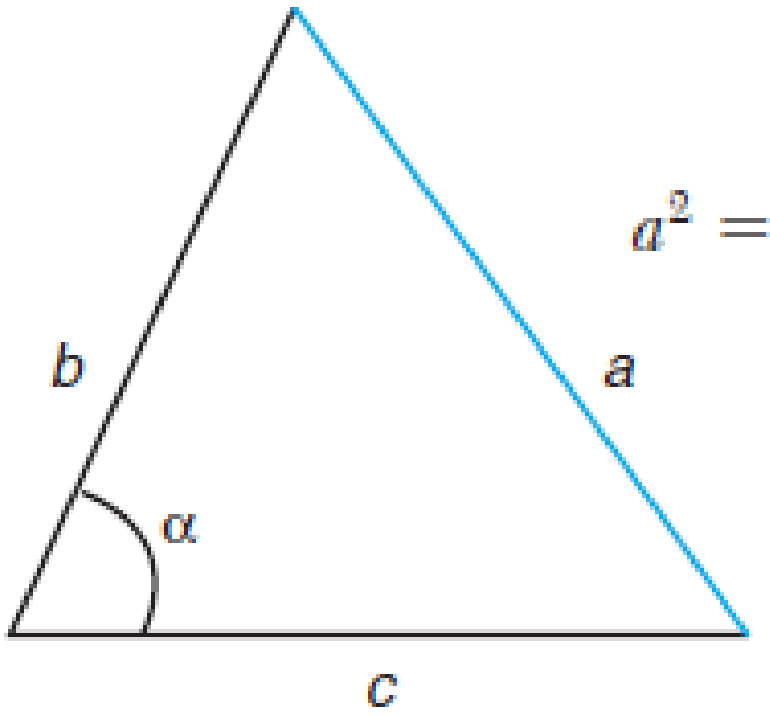
(continued)

...Continued

```
21.       /* Get second number and display its square root. */
22.       printf("Enter the second number> ");
23.       scanf("%lf", &second);
24.       second_sqrt = sqrt(second);
25.       printf("The square root of the second number is %.2f\n", second_sqrt);
26.
27.       /* Display the square root of the sum of the two numbers. */
28.       sum_sqrt = sqrt(first + second);
29.       printf("The square root of the sum of the two numbers is %.2f\n",
30.              sum_sqrt);
31.
32.       return (0);
33. }

Enter the first number> 9.0
The square root of the first number is 3.00
Enter the second number> 16.0
The square root of the second number is 4.00
The square root of the sum of the two numbers is 5.00
```

Example

- Give the C expression to calculate the value of the side a in the triangle below



$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

Answer:

```
a = sqrt(pow(b,2) + pow(c,2) - 2 * b * c * cos(alpha * PI / 180.0));
```

Angle must be in radian, see slide 6

II. User defined functions

1. Implement a function

- Make decision about data input
  - what values (if any) need to be passed to the function

    Example: two values of type `int`

- what value (if any) is returned by the function

  Example: a value of type `int`

- Choose a meaningful name for the function

  Example: `findMaxValue`

1.1 Provide the **function prototype**

Example:
```
int findMaxValue( int val1, int val2);
```

1.2 Provide the **function definition**

Example:
```
int findMaxValue( int val1, int val2)
{ . . . .
}
```

User defined functions

2. Use a function

- Call the function by specifying its name and passing the values (if required)

```
Example: printTime( hours, minutes );
```

- If the function returns a value, you can assign it to a variable

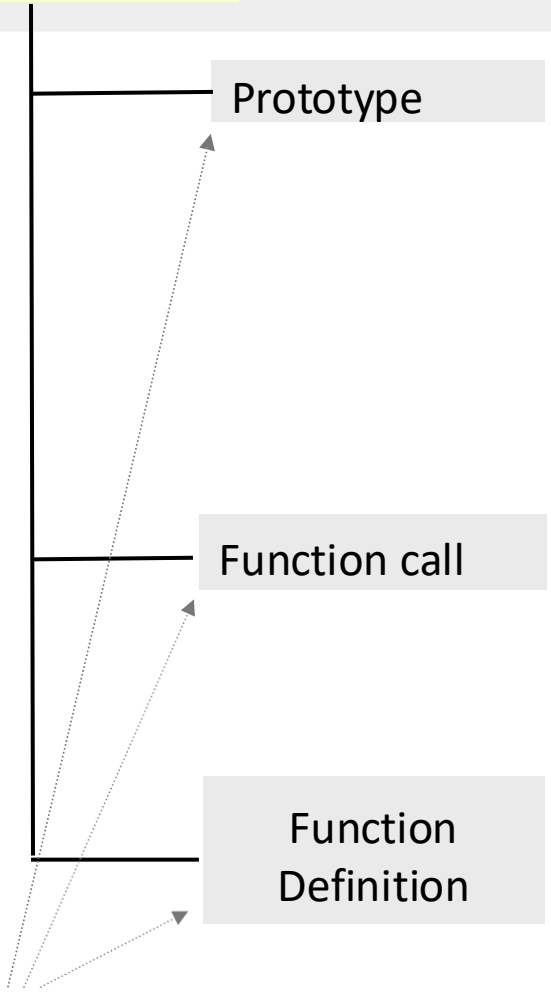
```
Examples: maxvalue = findMaxValue( 15, 94 );
          number = getRandomNumber();
```

User defined function

Prototype

Function call

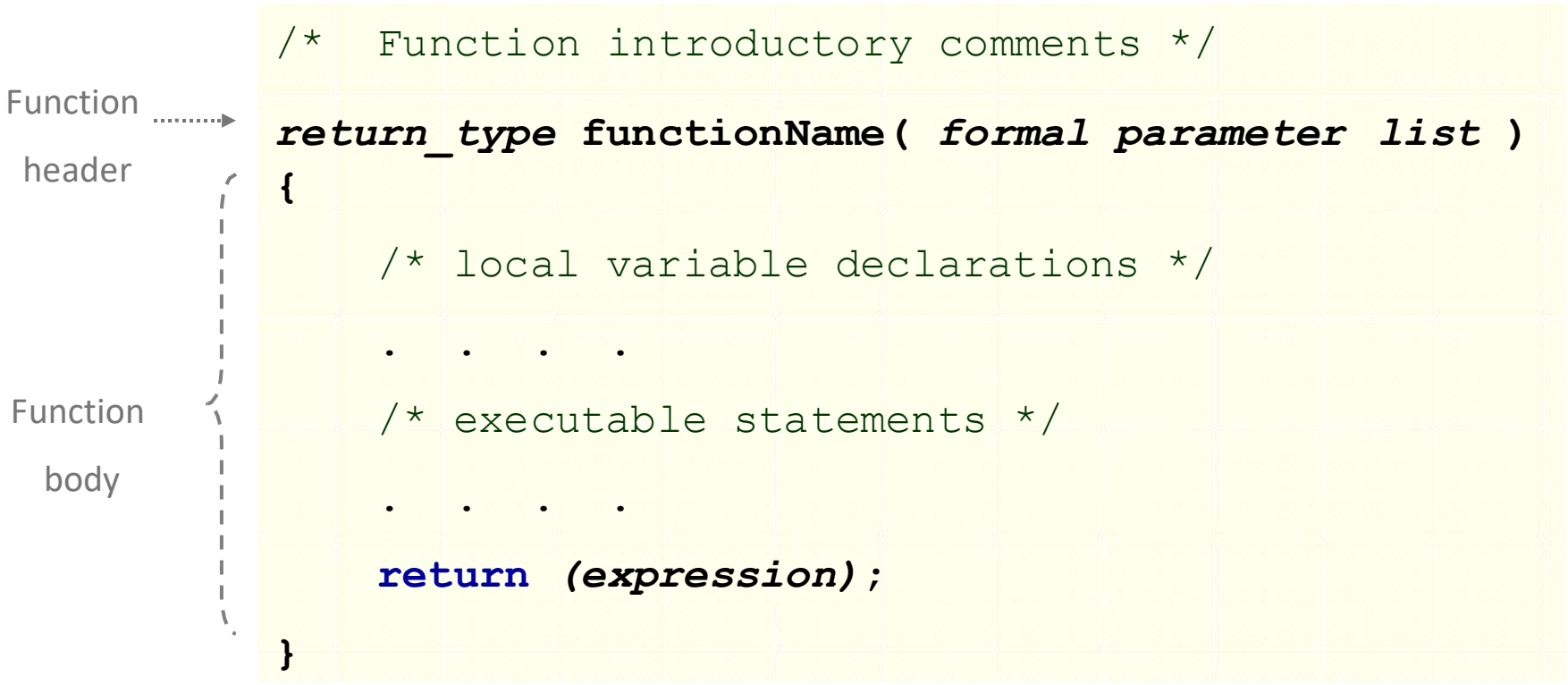
Function Definition



This order is important

```
/* preprocessor directives */
#include <stdio.h>

/* Function prototypes */
float getDistance(float velocity,  float theta);
. . .

int main(void)
{
    /* local variables */
    ……

   /* Calculate distance */
    distance = getDistance( 250.0, 30.0 );
    ……
    return (0);
}


/* Function Definitions */
float getDistance(float velocity, float  theta)
{
    theta *= RADIANS_PER_DEGREE;
    dist = velocity*velocity* sin(2.0*theta)/G;
}
……
```

Function Definition



Function return type

```
return_type functionName( formal parameter list )
{
   .   .   .   .   .
}
```

- Function return type has to be defined explicitly
  - otherwise C assumes `int`
- If function doesn't return anything , use `void` as return type
- Preferably, A `return` statement should be used even if the function return type is `void` (it doesn't return a value)

```
int firstF(…)
{
  int x;
  x = …
  return x;
}
```

```
double secondF (…)
{
   double time;
   time =…
   return (2.0*time);
}
```

```
void thirdF (…)
{
  …
  …
  return ;
}
```

Formal parameter list

```
return_type functionName( formal parameter list )
{
   .  .   .  .   .
}
```

- The list contains parameters that will be assigned with actual values passed to the function when it is called. Parameters in the list are separated by commas
- Each parameter must have a data type
- If no parameters are passed to the function, always specify **`void`**

C assumes an empty parameter list **( )** as "unknown fixed number of parameters".

```
return_type functionName(type par1, type par2, type par3) or
return_type functionName( void )
```

Examples:

```
int calcAverage(int x1, int x2)
double getHeight(double velocity, double theta)
char getSelection( char lt, int index, int ptr )
void prompt( void )
```

Quiz

- Find errors in the following function definitions

```
double multiply( int x,     y);
{
    int z;

    z = x*y + w;

    return z;
}
```

Missing data type for y
Variable w is not declared.
Semicolon at the end of the function header
CAUTION: Return type is double but z is int

```
getAverage( float x, float y);
{
    float sum;

    sum = x +y;

    return (sum/2.0);
}
```

No return type for the function, int will be assumed
Semicolon at the end of the function header
CAUTION: Return type is float

User defined function

- Function Prototypes
- calling
- definition

```
/* preprocessor directives */
……

/* Function prototypes */
double getDistance(double velocity, double theta);
…..

int main(void)
{
   /* local variables */
   ……

   /* statements */
   ……
   distance = getDistance(velocity, theta);
   ……
   return 0;
}
/* Calculate distance */
double getDistance(double velocity, double theta)
{
   theta *= RADIANS_PER_DEGREE;
   return (velocity*velocity* sin(2.0*theta)/G);
}
……
```

After the preprocessor directives

Before the main() function

Function prototypes

Syntax:

```
return_type functionName(type var1, …, type var3);
```

Examples:

```
#include <stdio.h>   /* header files needed for your program */
 .    .    .    .
# define ALFA 5.67   /* definition of constants */
 .    .    .    .
int average(int x, int y);
double calHeight(double velocity, double theta);
char getSelection(void);

int main (void)
{
    …
    return (expression)
}
```

Prototypes of all functions defined by you must be here

Function prototypes

- Where:
  - A function prototype must appear before any call of the function
- Syntax:
  - Assuming that you have defined a function

```
double wage(int hours, double rate)
{
    return (hours*rate);
}
```

The prototype consists only of the function header (no code) and ends with **;**

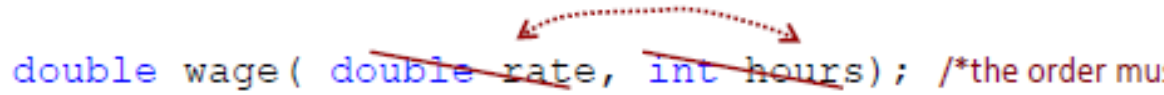
```
double wage( int hours, double rate);
```

  - C does not require identifiers in the parameter list, but it is easier to interpret the code if they are included

```
double wage( int, double ); /* possible, but not recommended */
```

  - Different parameter names from those in the function definition may be used (not recommended), but the order of the parameters in the declaration must match the parameter order in the definition

```
double wage( int hr, double payRate); /*not recommended */
```



Quiz

- Presuming the following definition

```
float process(int a, float b)
{
    return (a+b);
}
```

- Find errors in the following function prototypes

```
float process(int a, b);
float process(int a, int b);
float process(int a);
float process(float b, int a);
float process(int a, float b);
```

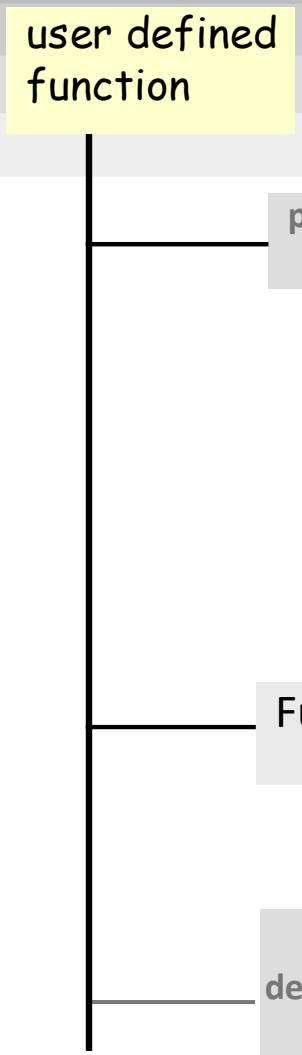
Missing data type for b

Different data type for b

Missing parameter b

Different data types for both parameters

No errors




```
/* preprocessor directives */
……

/* prototypes */
float getAverage(float a, float b);
…..

int main(void)
{
    /* local variables */
    double minPrice = 34.0;
    double maxPrice = 45.0;

    /* executable statements */
    ……
    distance = getAverage( minPrice,  maxPrice );
    …….
   return (0);
/* Calculate distance */
float getAverage( float a, float b)
{
   float average = (a + b)/2.0;
   return (average);
}
……
```

Function call

Syntax:

```
functionName(actual parameter list);
```

Examples:

```
int x=10, y=20, avg, prod;
. .   . .
prompt();   /* no parameters */
```

Actual parameters

- ✓constants
- ✓variables
- ✓expressions
- ✓function call

Don't specify data types of the actual parameters in function calls

```
avg = average( int x, int y );   /* two variables */

avg = average( x, y );           /* two variables */
prod = product( 2, avg-3 );      /* a constant and an expression */
avg = average( 15, prod );       /* a constant and a variable */
avg = average(  product( 2, 3), 8 ); /* a function and a constant */
```

UNIVERSITY OF WOLLONGONG IN DUBAI

Formal and Actual parameters

```
/* Function prototype */
double findAverage(double num1, double num2);
.   .   .   .
int main(void)
{
  double price1 = 10.00, price2 = 20.00;
  double avgPrice;
  /* Function call */
  avgPrice = findAverage( price1 , price2);
  . . . . .
}
```

10.00 **20.00**

Actual parameters

price1 and price2

```
   /* Function definition */
 double findAverage(double num1, double num2)
  {
15.00 return ((num1+num2)/2.0) ;
  }
```

Formal parameters

num1 and num2

UNIVERSITY OF WOLLONGONG IN DUBAI

Function call examples

```
#include <stdio.h>

int multiply(int x, int y);     /* function prototype */
.    .    .    .    .
int main( void )
{
    int a, b = 6, product, result;
    .   .    .
    a = 5;  /* make sure that variables are initialized before they are used
               in function calls */

    product = multiply(a, b);
    printf("a x b = %d \n", product );
    printf("6 x 7 = %d \n",  multiply( 6, 7 ) );
    . . .
    product = multiply(a+6, b);
    .   .   .
    result = multiply( a/2, b ) + multiply( 6, b/3 );
    product = multiply( multiply(a, b), a );
    .   .   .
}
```

Quiz

- Which function calls are incorrect?

```
int sum(int x, int y);
.   .    .    .

int main (void)
{
  int total, a=5, b=6, y;

  y = sum( 10, 20 );
  a = sum( a, b )
  total = sum( a+1, 20 );
  total = sum( int a, int 10 );
  sum( a+b, 20 );
  y = sum( 10+a );
  total = sum( a, b ) + sum( 10, 20 );

  return (0);
}
```

incorrect : missing ;

incorrect

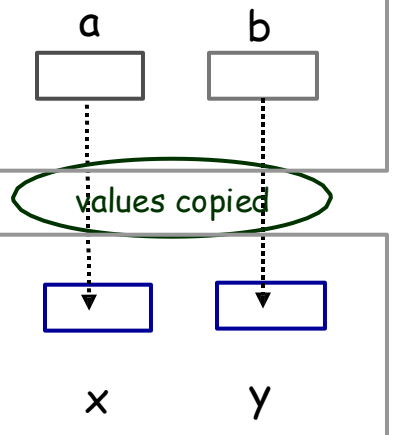
incorrect

```
int sum(int x, int y)
{
    return (x+y);
}
```

Void functions with parameters

what are the values of `a` and `b` ?

```
#include <stdio.h>
void printTwo(int x, int y);

int main(void)
{
  int a=5, b=10;

  printTwo(a, b);
  return 0;
}
```




```
void printTwo(int x, int y)
{
  printf("x=%d, y=%d!\n", x,y);
  return;
}
```

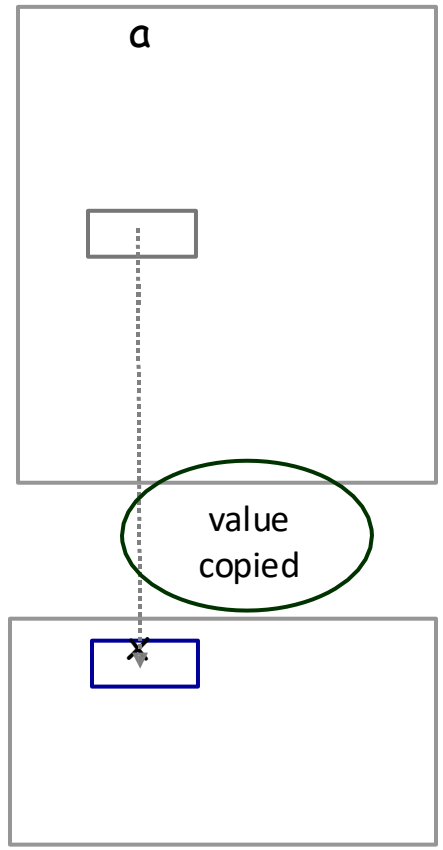
what are the values of `x` and `y` *after the function call?*

Functions returning a value

result returned and then assigned to b

```
#include <stdio.h>

int sqr(int x);

int main(void)
{
   int a, b;
   scanf("%d", &a);
   b = sqr( a );

   printf("%d squared is %d\n", a,b);
   return 0;
}
```

```
int sqr(int x)
{
    int result;
    result = x*x;

    return result;
}
```



III. Scope

- Global scope
  - Anything (constants, variables, functions, etc) defined in the global scope area is visible to the entire program
  - Global variables are placed in memory when the program is started and released from memory only when it is terminated
- Local scope
  - Local scope variables are visible only within a function
  - Local automatic variables are placed in memory when the function is called and they are destroyed when the function returns
  - All subsequent function calls cannot retrieve values of local automatic variables from previous calls of the same function

Quiz

```
/* this is a sample to demonstrate scope */
#include <stdio.h>

int fun(int a, int b);
```

global area

Automatic variable `y` is local to `main()`

```
int main(void)
{
    int a, b, c;                    main's area
    float y = 0.0;
    .   .   .
    y = 5.0;            /* OK */
    c = fun(a, b);      /* OK */
    printf( " y = %f ", y );
    return 0;
}
```

y = ? y is 5

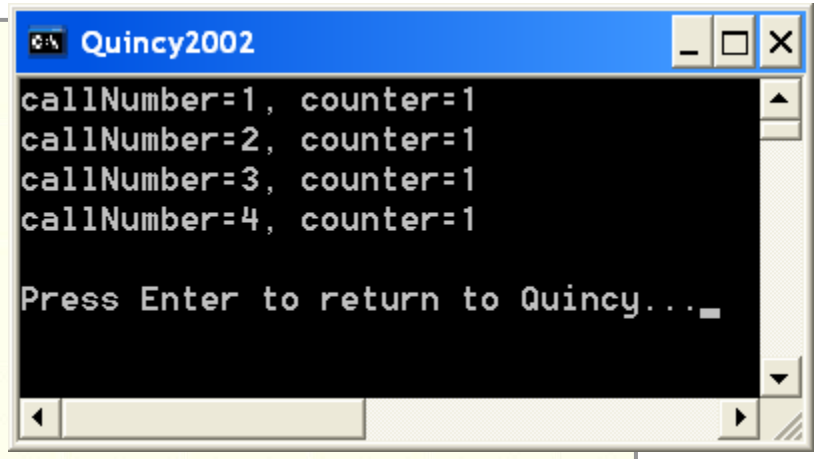
Automatic variable y is local to `fun()`

It's different from the variable `y` declared in `main()`

```
int fun(int j, int k)                fun's area
{
    int y;
    y = 10;             /* OK */
    return (y)
}   .   .   .
```

Static variables in functions

```c
/* comments */
#include <stdio.h>

void function( void );

int main(void)
{
    function();
    function();
    function();
    function();

    return (0);
}
/* function */
void function( void )
{
     static int callNumber = 0;
     int counter=0;
     callNumber++;
     counter++;
     printf("callNumber=%d, counter=%d\n",  callNumber, counter);
     return;
}
```



value of a static variable is preserved between function calls

Initialized only once

IV. Case study: Projectile Trajectory

- A projectile fired at an angle θ with an initial velocity v_0 travels a distance d given by

$$d = \sin(2*\theta) * v_o^2/g$$

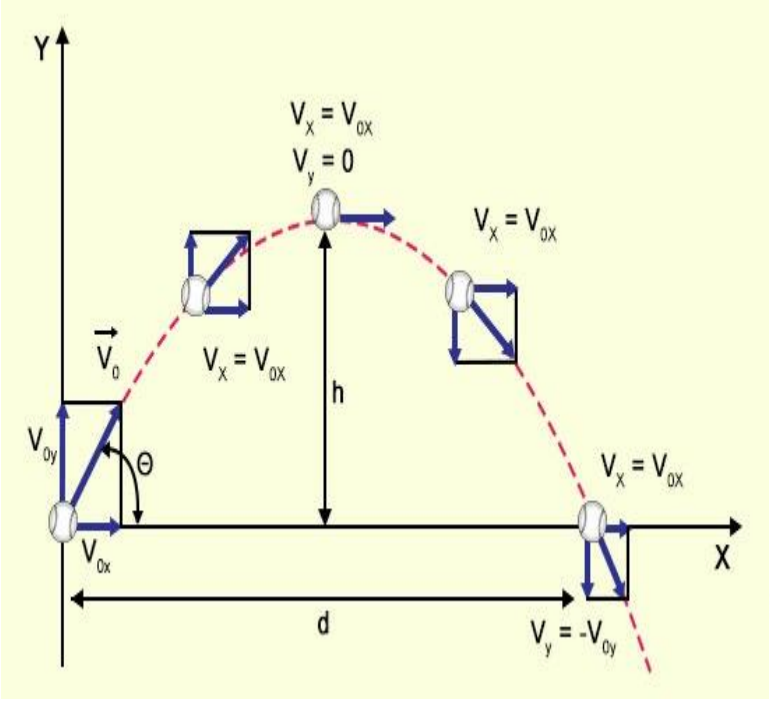
where g is the acceleration constant of $9.8\ m/sec^2$

It stays in motion for a time t given by

$$t = \sin(\theta) * 2 * v_o/g$$

and attains the maximum hight h given by

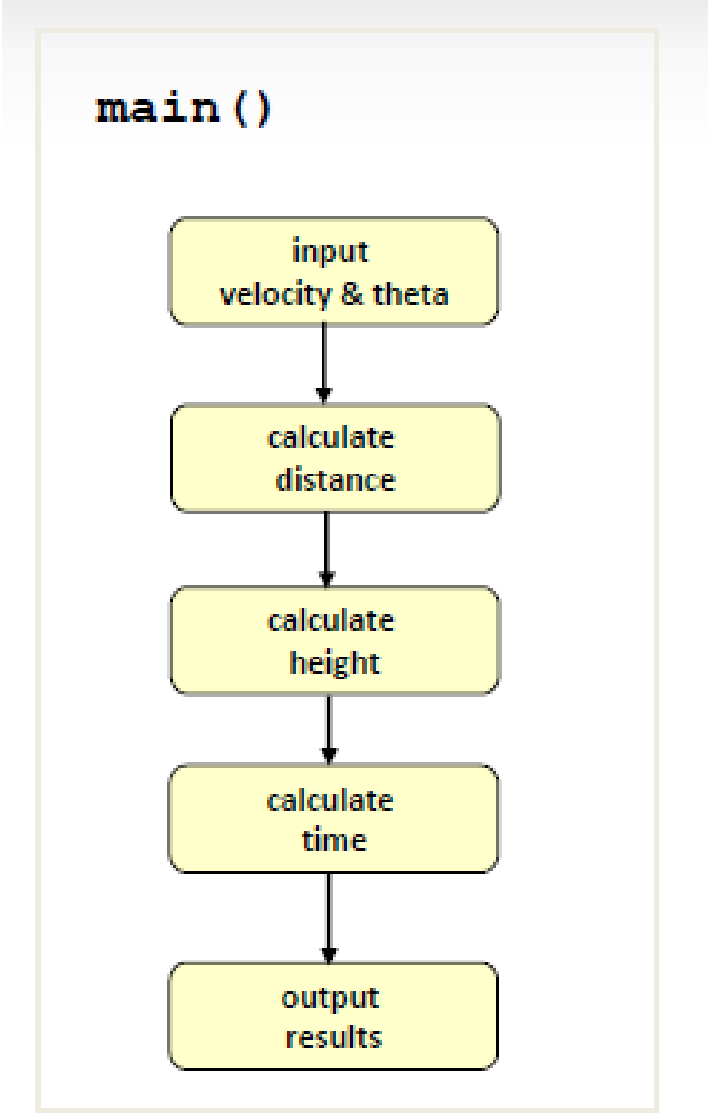
$$h = \sin(\theta) * v_o^2/g$$



C Program with main() function only

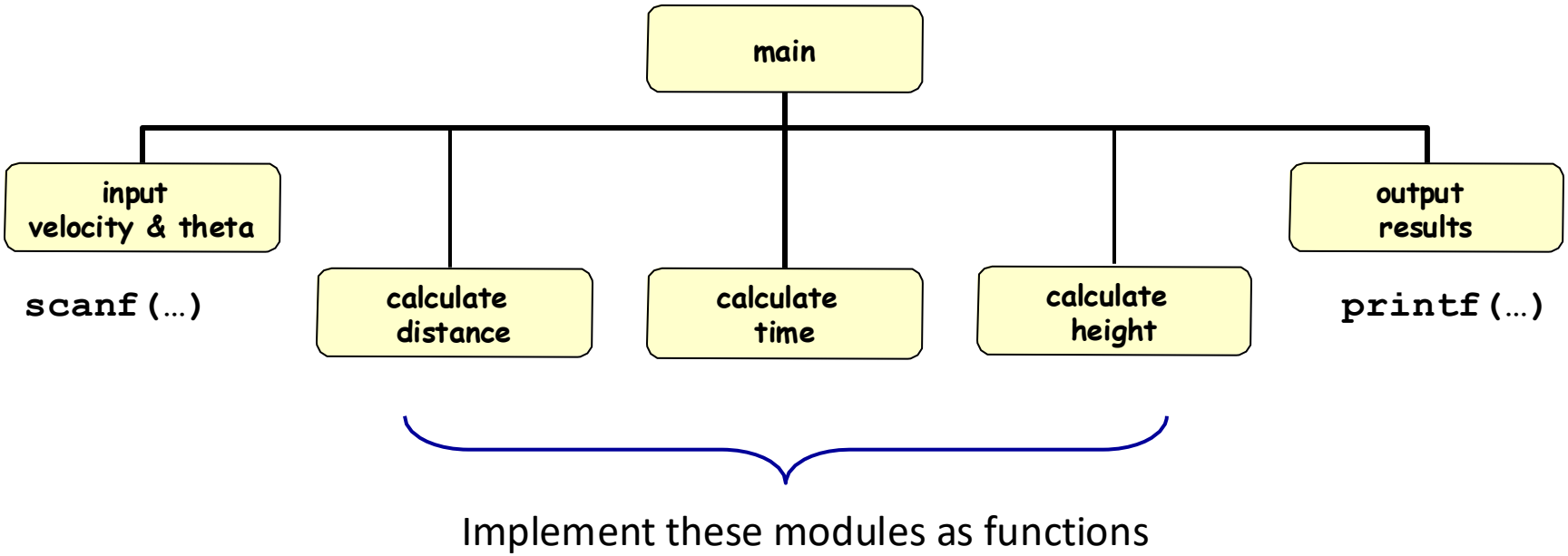
```
#include <stdio.h>
#include <math.h>
#define G       9.8
#define PI      3.141592
#define RADIANS_PER_DEGREE    (PI/180)

int main(void)
{
        float velocity, theta;
        float distance, time, height;

        scanf("%f %f", &velocity, &theta);

        /* convert degrees to radians */
        theta *= RADIANS_PER_DEGREE;

        distance = velocity*velocity* sin(2.0*theta)/G;
        time = 2.0*velocity*sin(theta)/G;
        height = pow(velocity, 2.0)*sin(theta)/G;

        printf("distance = %f\n", distance);
        printf("time = %f\n", time);
        printf("height = %f\n", height);

        return (0);
}
```

Single-module Solution



Looks simple, but what if you need to calculate the distance, time and height several times with various parameters?

Task decomposition



- To implement a function you need to consider:
  - function data input
  - an algorithm that is needed to produce the result
  - function data output

```c
#include <stdio.h>
#include <math.h>
#define G     9.8
#define PI    3.14159265
#define RADIANS_PER_DEGREE    (PI/180)

/* Function prototypes */
float getDistance(float velocity, float theta);
float getTime(float v, float t);
float getHeight(float velocity, float theta);

int main(void)
{
    float velocity, theta;
    float distance, time, height;

    scanf("%lf %lf", &velocity, &theta);

    distance = getDistance (velocity, theta);

    time = getTime (velocity, theta);

    height = getHeight (velocity, theta);

    printf("distance = %f\n", distance);
    printf("time = %f\n", time);
    printf("height = %f\n", height);

    return (0);
}
```

User defined functions

```c
/* Calculate distance */
float getDistance( float vel, float theta )
{
    float dist;
    theta *= RADIANS_PER_DEGREE;
    dist = vel*vel*sin(2.0*theta)/G;
    return dist;
}

/* Calculate time */
float getTime( float vel, float theta )
{
    float time;
    theta *= RADIANS_PER_DEGREE;
    time = 2.0*vel*sin(theta)/G;
    return time;
}

/* Calculate height */
float getHeight( float vel, float theta )
{
    float ht;
    theta *= RADIANS_PER_DEGREE;
    ht = pow(vel, 2.0)*sin(theta)/G;
    return ht;
}
```

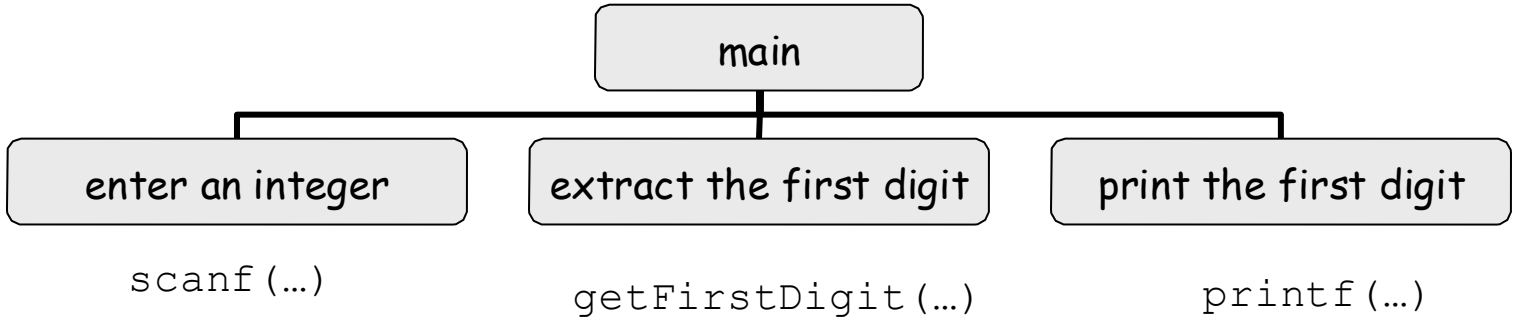
IV. Case study: Integer Digits

- Print the least significant digit of an integer entered from the keyboard

  Example:

  ```
  24 -> 4
  138 -> 8
  ```

  One of the key questions that you should answer: How many functions do I need?



Example

```
/*
 * Print the first digit of an integer
*/
#include <stdio.h>

int getFirstDigit( int num );

int main(void)
{
  int number, digit;
   printf("Enter an integer:");
   scanf("%d", &number);

   digit = getFirstDigit( number );

  printf("The digit is: %d", digit);
  return 0;
}
```

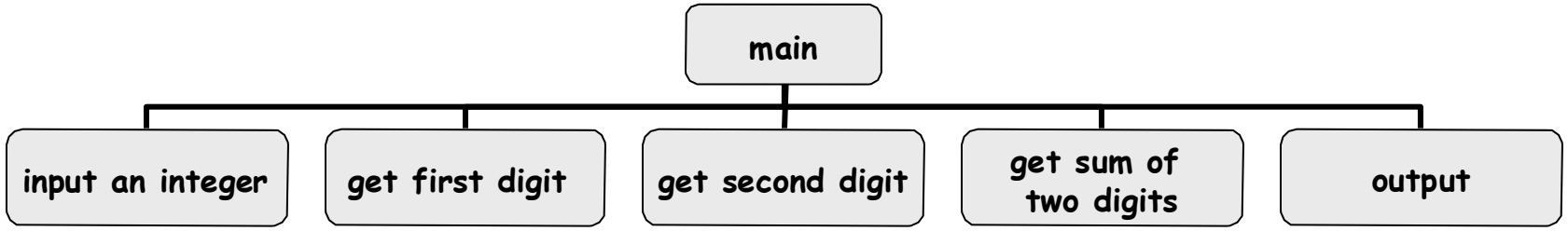
```
/*
 * Extracts the least significant
 * digit of an integer
 * pre: num – integer
 * post: the least digit of num
 */
int getFirstDigit( int num )
{
   return (num % 10);
}
```

Example

- Design a program that extracts and adds two least significant digits of an integer

- Usually, there are several ways how to split a program into a set of interacting functions

A possible solution



Solution

```c
/*
 * This program extracts and adds the two least
 * significant digits of an integer
 */
#include <stdio.h>

/* prototypes */
int getFirstDigit(int num);
int getSecondDigit(int num);
int addTwoDigits(int d1, int d2);

int main(void)
{
    int number, digit1, digit2, sum;

    printf("Enter an integer: ");
    scanf("%d", &number);

   digit1 = getFirstDigit(number);
   digit2 = getSecondDigit(number);
   sum = addTwoDigits(digit1, digit2);

   printf("The sum is %d\n", sum);

   return (0);
}
```

```c
/* first digit of an integer
*/
int getFirstDigit( int num )
{
    return (num%10);
}

/* second digit of an
integer*/
int getSecondDigit( int num )
{
    return ( (num/10)%10 );
}
int addTwoDigits( int d1, int d2 )
{/* Sum of two digits */
    return (d1+d2);
}
```

IV. Common Errors

```
/* the function prototype */
double divide( int dividend, int divisor );
   …
/* the function definition */
double divide( float dividend, float divisor)
{
   . . .
}
```

Type mismatch

```
/* the function prototype */
double divide( float dividend, float divisor );
   …
/* the function definition */
double divide( float dividend )
{
   . . .
}
```

Mismatch in the number of parameters

Easy to find and fix as these errors are detected by the compiler

Common Errors

```
double divide( float dividend, float divisor );
   …
double divide( float divisor, float dividend )
{
   .  .  .
}
```

Incorrect order

(Not detected by the compiler)

```
double divide( float dividend, float divisor);
   …
double divide( float dividend, float divisor)
{
    float dividend;
    .  .  .
    return result;
}
```

A local variable is declared with the same name as a parameter

(Detected by the compiler)

UNIVERSITY OF WOLLONGONG IN DUBAI

Common Errors

```
float divide(float dividend, float divisor);
   …
int main(void)
{
    float dividend, divisor, res;
     …  …
    res = divide(dividend, divisor);
}

float divide(float dividend, float divisor)
{
    float result;
    .   .   .
    return;
}
```

No value is returned, while `float` is expected

(Warning issued by most compilers)

Common Errors

```
float divide(float dividend, float divisor);
   …
int main(void)
{
    float dividend, divisor, res;
     …  …
    res = divide( dividend ); /* two parameters should be passed */
}

float divide(float dividend, float divisor)
{
    float result;
    .   .   .
    return result;
}
```

Insufficient number of actual parameters

(detected by the compiler)

Bad Structuring

```
float getAverage( void ); /* prototype */
   …
int main(void)
{
    float res;
     …  …
    res = getAverage();
}

/* calculates the average of two numbers */
float getAverage( void )
{
    float num1, num2, avrg;

    printf("Enter two numbers:");
    scanf("%f %f", &num1, &num2 );

    avrg = (num1 + num2)/2.0
    return avrg;
}
```

Get average of what?

The function name suggests that this is a data processing function. The user input/output should not be here

How to reuse this function in other projects where I/O is not required ?

A Better Solution

```
float getAverage( float par1, float par2 );
   …
int main(void)
{
    float num1, num2, res;

    printf("Enter two numbers:");
    scanf("%f %f", &num1, &num2 );
    .  .  .
    res = getAverage( num1, num2 );
}

/* calculates the average of two numbers */
float getAverage( float par1, float par2 )
{
    float avrg;

    avrg = (par1 + par2)/2.0;
    return avrg;
}
```

User input (if it's needed at all) can be moved to the top level function, or to another function responsible exclusively for user input of numbers

Programming Style

```
  /* ---------------------------------------------
  * Description of the function
  *
  * Parameters: valid input parameters
  * Returned value: description on the return value
  ---------------------------------------------*/
retutn_type functionName( list of formal parameters )
{
    /* local variables */
    .  .  .

    /* block of executable statements */
    .  .  .

}
```