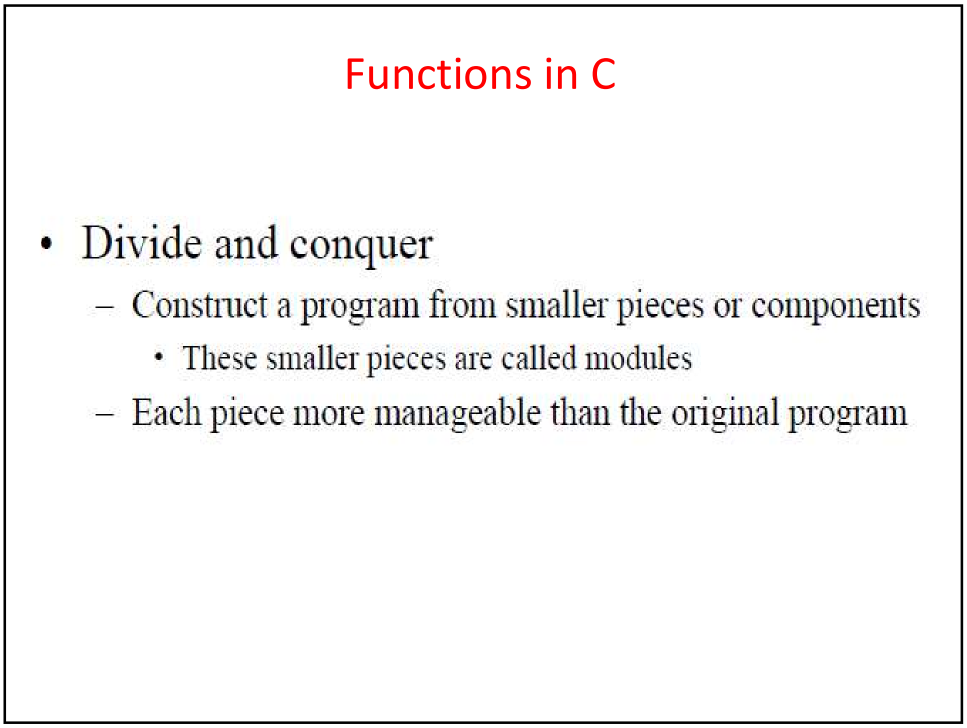
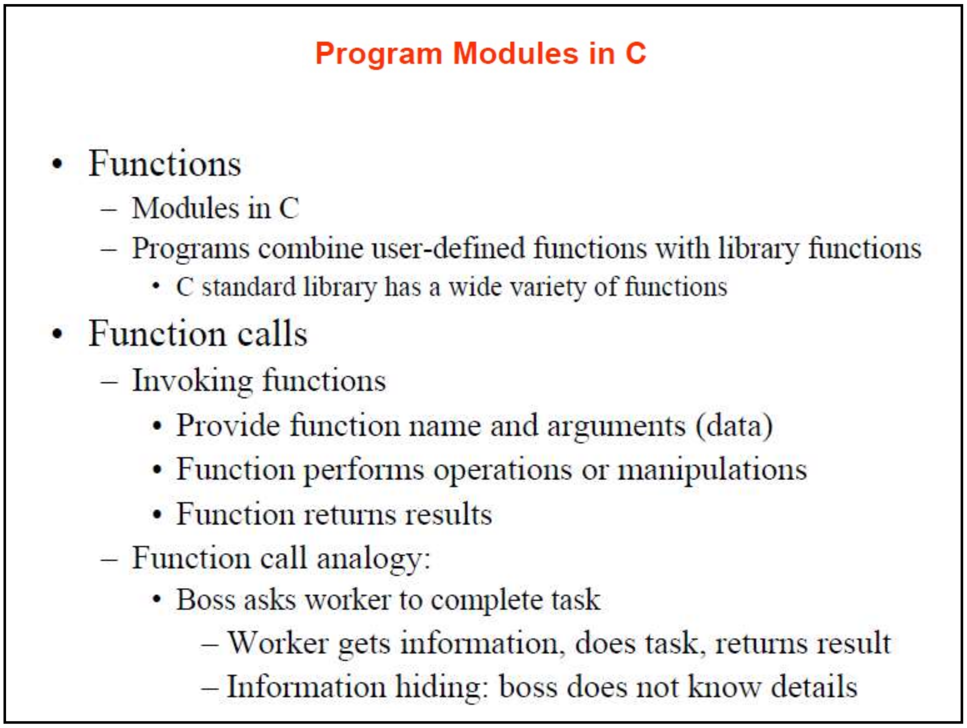


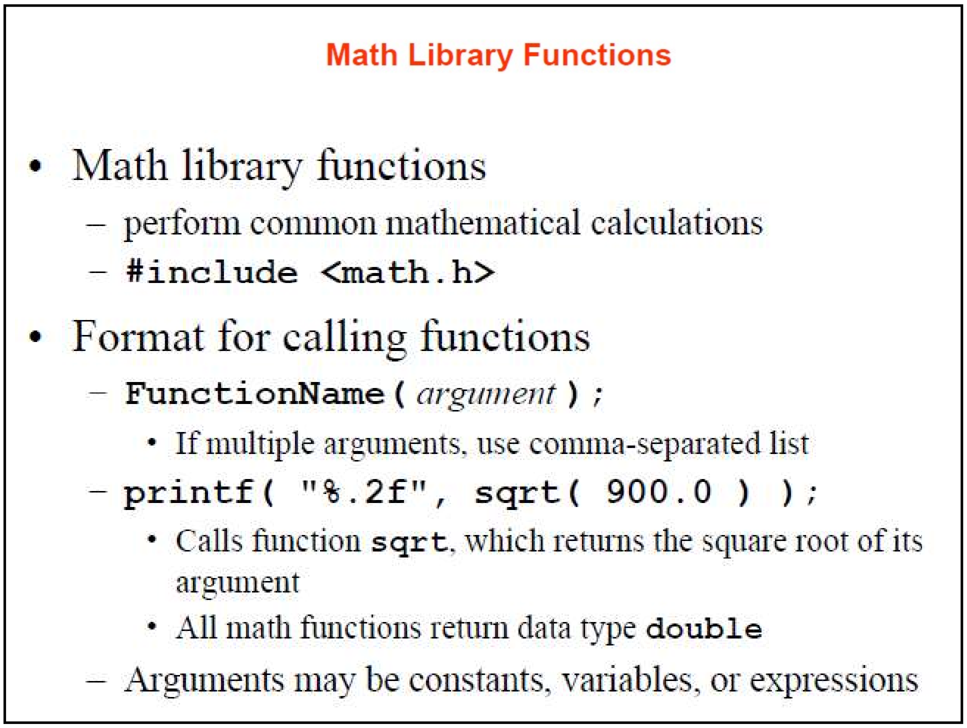
## Functions in C

- Divide and conquer
  - Construct a program from smaller pieces or components
    - These smaller pieces are called modules
  - Each piece more manageable than the original program



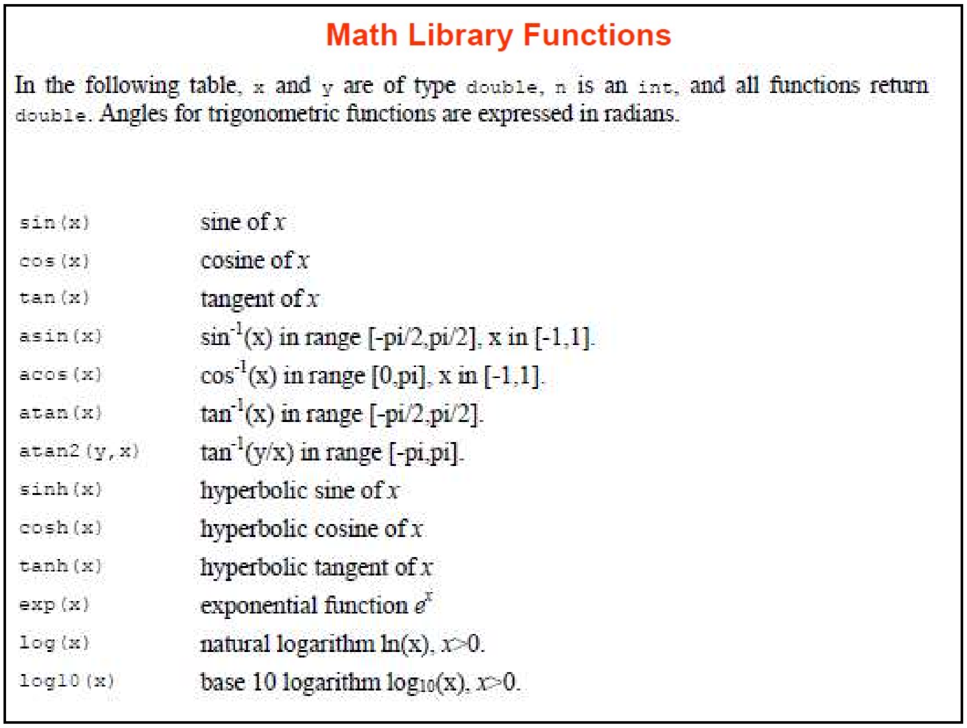
## Program Modules in C

- Functions
  - Modules in C
  - Programs combine user-defined functions with library functions
    - C standard library has a wide variety of functions
- Function calls
  - Invoking functions
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
  - Function call analogy:
    - Boss asks worker to complete task
      - Worker gets information, does task, returns result
      - Information hiding: boss does not know details



## Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
- Format for calling functions
  - **`FunctionName(`** *argument* **`);`**
    - If multiple arguments, use comma-separated list
  - **`printf( "%.2f", sqrt( 900.0 ) );`**
    - Calls function **`sqrt`**, which returns the square root of its argument
    - All math functions return data type **`double`**
  - Arguments may be constants, variables, or expressions

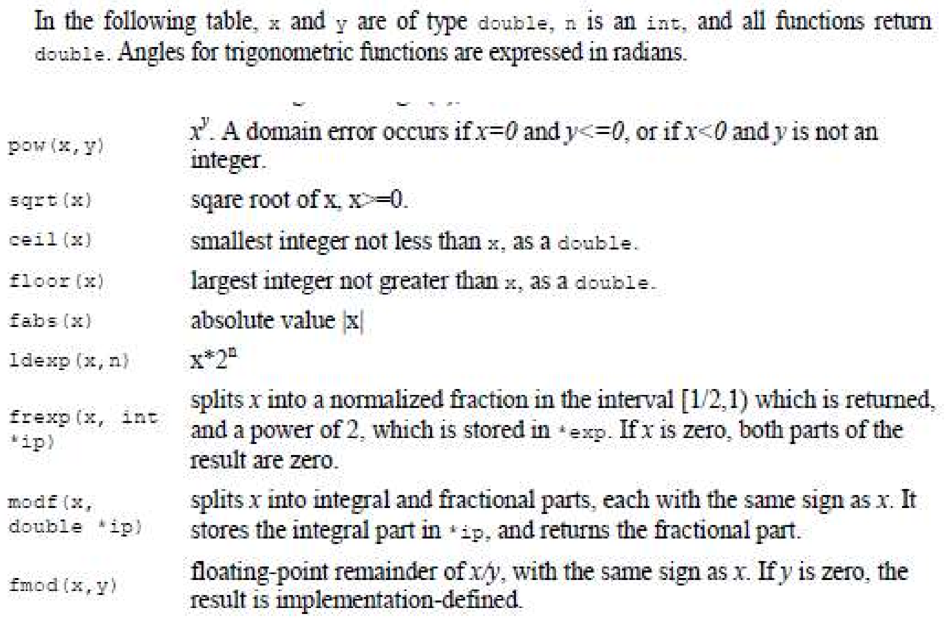


## Math Library Functions

In the following table, `x` and `y` are of type `double`, `n` is an `int`, and all functions return `double`. Angles for trigonometric functions are expressed in radians.

| | |
|---|---|
| `sin(x)` | sine of *x* |
| `cos(x)` | cosine of *x* |
| `tan(x)` | tangent of *x* |
| `asin(x)` | $\sin^{-1}(x)$ in range [-pi/2,pi/2], x in [-1,1]. |
| `acos(x)` | $\cos^{-1}(x)$ in range [0,pi], x in [-1,1]. |
| `atan(x)` | $\tan^{-1}(x)$ in range [-pi/2,pi/2]. |
| `atan2(y,x)` | $\tan^{-1}(y/x)$ in range [-pi,pi]. |
| `sinh(x)` | hyperbolic sine of *x* |
| `cosh(x)` | hyperbolic cosine of *x* |
| `tanh(x)` | hyperbolic tangent of *x* |
| `exp(x)` | exponential function $e^x$ |
| `log(x)` | natural logarithm ln(x), $x>0$. |
| `log10(x)` | base 10 logarithm $\log_{10}(x)$, $x>0$. |

## Math Library Functions



In the following table, x and y are of type double, n is an int, and all functions return double. Angles for trigonometric functions are expressed in radians.

| | |
|---|---|
| pow(x,y) | $x^y$. A domain error occurs if $x=0$ and $y<=0$, or if $x<0$ and $y$ is not an integer. |
| sqrt(x) | sqare root of x, x>=0. |
| ceil(x) | smallest integer not less than x, as a double. |
| floor(x) | largest integer not greater than x, as a double. |
| fabs(x) | absolute value \|x\| |
| ldexp(x,n) | $x*2^n$ |
| frexp(x, int *ip) | splits *x* into a normalized fraction in the interval [1/2,1) which is returned, and a power of 2, which is stored in *exp. If *x* is zero, both parts of the result are zero. |
| modf(x, double *ip) | splits *x* into integral and fractional parts, each with the same sign as *x*. It stores the integral part in *ip, and returns the fractional part. |
| fmod(x,y) | floating-point remainder of *x/y*, with the same sign as *x*. If *y* is zero, the result is implementation-defined. |

## Functions

- Functions
  - Modularize a program
  - All variables declared inside functions are local variables
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - Local variables
- Benefits of functions
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoid code repetition

## Function Definitions

- Function definition format

```
return-value-type  function-name( parameter-list )
  {
    declarations and statements
  }
```

  - Function-name: any valid identifier
  - Return-value-type: data type of the result (default `int`)
    - `void` – indicates that the function returns nothing
  - Parameter-list: comma separated list, declares parameters
    - A type must be listed explicitly for each parameter unless, the parameter is of type `int`

## Function Definitions

- Function definition format (continued)

```
return-value-type  function-name( parameter-list )
  {
    declarations and statements
  }
```

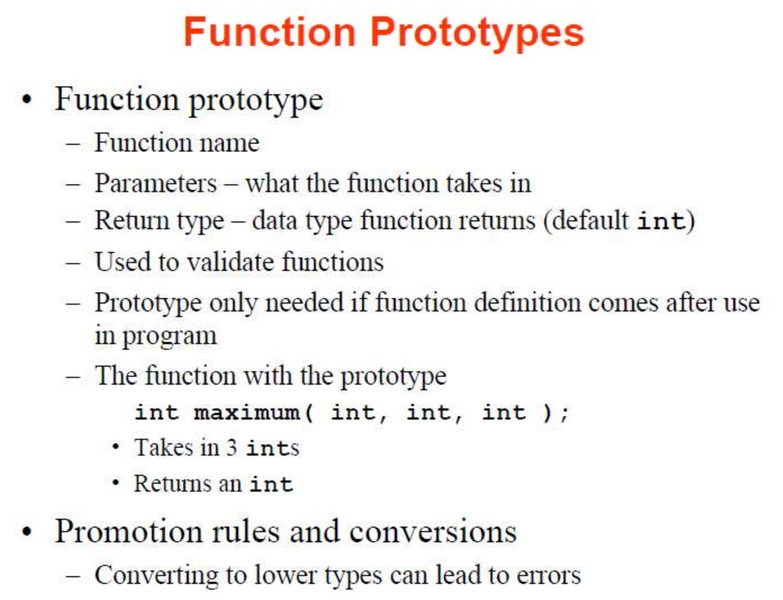
  - Declarations and statements: function body (block)
    - Variables can be declared inside blocks (can be nested)
    - Functions can not be defined inside other functions
  - Returning control
    - If nothing returned
      - `return;`
      - or, until reaches right brace
    - If something returned
      - `return` *expression* `;`

```
/* Finding the maximum of three integers */
#include<stdio.h>
Int maximum( int, int, int);    /* function prototype */
Int main()
  {int  a, b, c;
   printf( "Enter three integers: " );
   scanf( "%d%d%d", &a, &b, &c );
   printf( "Maximum is: %d\n", maximum( a, b, c ) );
return 0;
  }

/* Function maximum definition */
Int maximum( int x, int y, int z )
  { int max = x;
     if( y > max )
        max = y;
      if( z > max )
         max = z;
    return max;
}
```

OUTPUT:

```
Enter three integers: 22 85 17
Maximum is: 85
```



## Function Prototypes

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default `int`)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype

    `int maximum( int, int, int );`
    - Takes in 3 `int`s
    - Returns an `int`
- Promotion rules and conversions
  - Converting to lower types can lead to errors

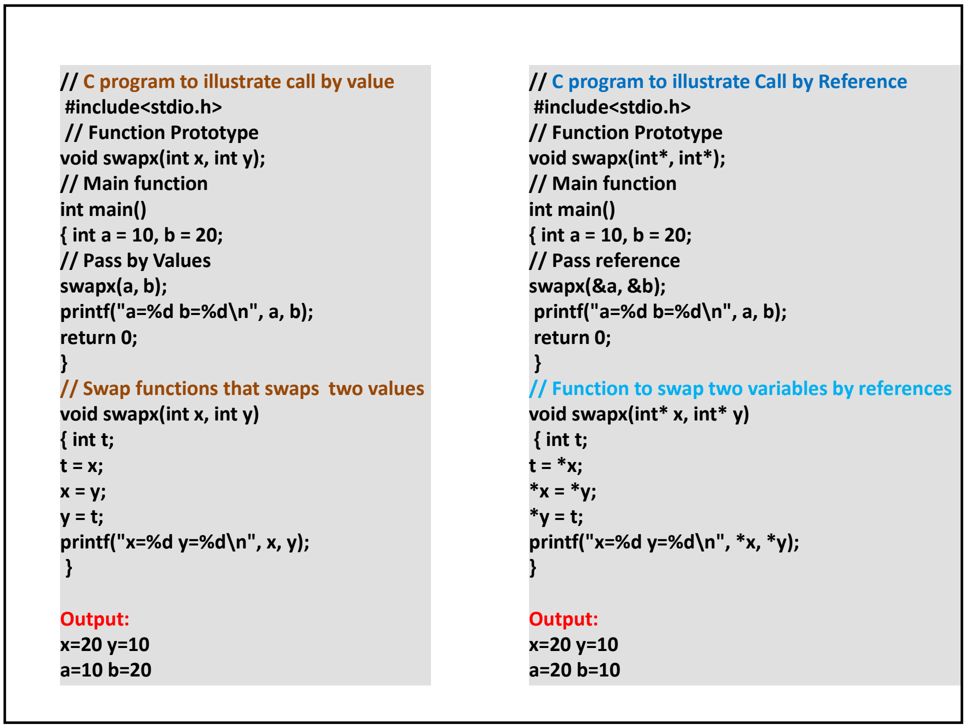
## Header Files

- Header files
  - Contain function prototypes for library functions
  - **`<stdlib.h>`** , **`<math.h>`** , etc
  - Load with **`#include <filename>`**

    **`#include <math.h>`**
- Custom header files
  - Create file with functions
  - Save as **`filename.h`**
  - Load in other files with **`#include "filename.h"`**
  - Reuse functions

## Calling Functions: Call by Value and Call by Reference

- Used when invoking functions
- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value



```
// C program to illustrate call by value
 #include<stdio.h>
 // Function Prototype
void swapx(int x, int y);
// Main function
int main()
{ int a = 10, b = 20;
// Pass by Values
swapx(a, b);
printf("a=%d b=%d\n", a, b);
return 0;
}
// Swap functions that swaps  two values
void swapx(int x, int y)
{ int t;
t = x;
x = y;
y = t;
printf("x=%d y=%d\n", x, y);
 }
```

**Output:**

```
x=20 y=10
a=10 b=20
```

```
// C program to illustrate Call by Reference
 #include<stdio.h>
// Function Prototype
void swapx(int*, int*);
// Main function
int main()
{ int a = 10, b = 20;
// Pass reference
swapx(&a, &b);
 printf("a=%d b=%d\n", a, b);
 return 0;
 }
// Function to swap two variables by references
void swapx(int* x, int* y)
 { int t;
t = *x;
*x = *y;
*y = t;
printf("x=%d y=%d\n", *x, *y);
}
```

**Output:**

```
x=20 y=10
a=20 b=10
```