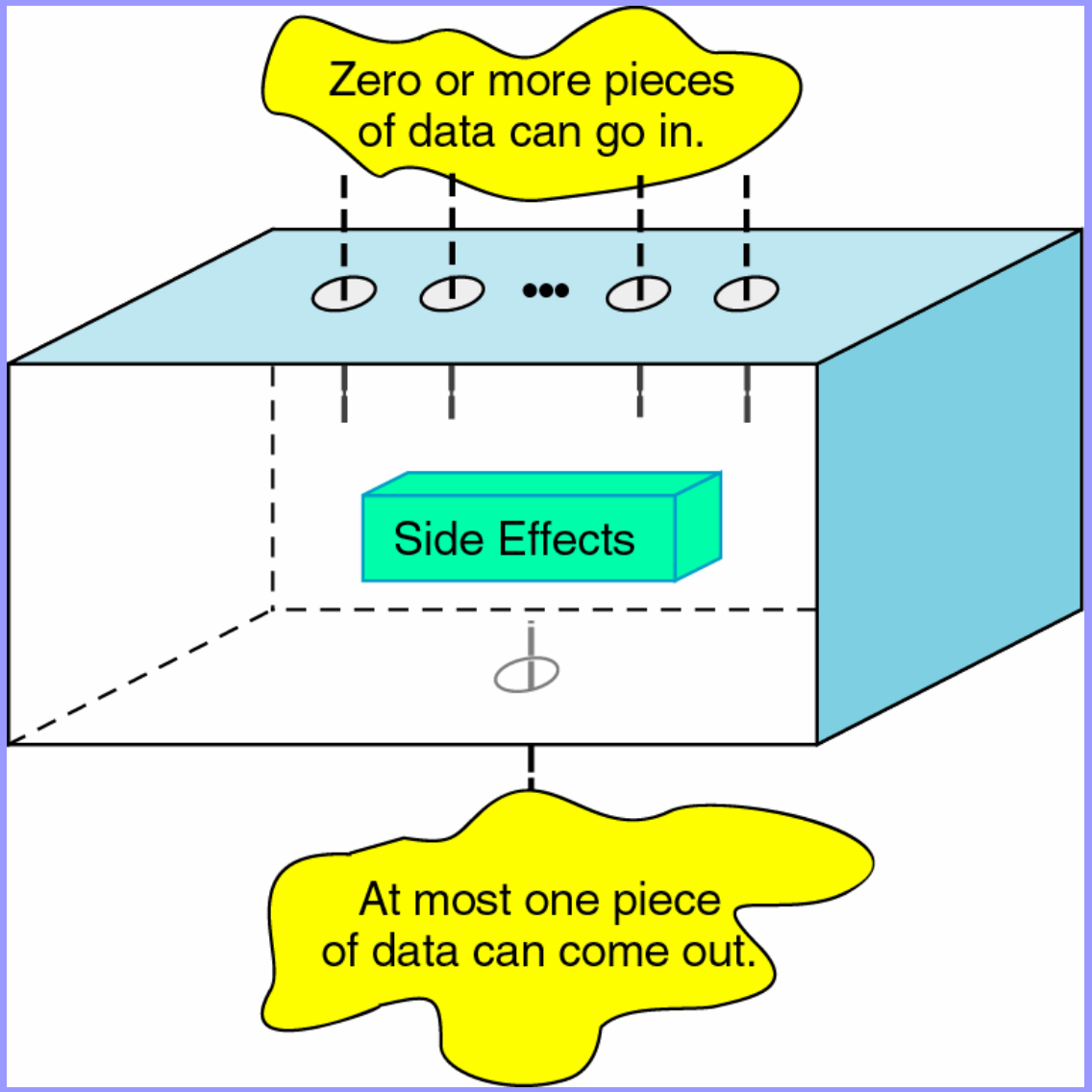
Lecture 10
Functions

C Programming

Lecture 10 Topics

- Writing a Program Using Functional Decomposition
- Writing a Void Function for a Task
- Using Function Arguments and Parameters
- Differences between Value Parameters and Reference Parameters

Functions


Zero or more pieces
of data can go in.
Side Effects
At most one piece
of data can come out.

Functions

- every C program must have a function called main
- program execution always begins with function main
- any other functions are subprograms and must be called

When a function is called,

temporary memory is set up (for its value parameters and any local variables, and also for the function's name if the return type is not void).

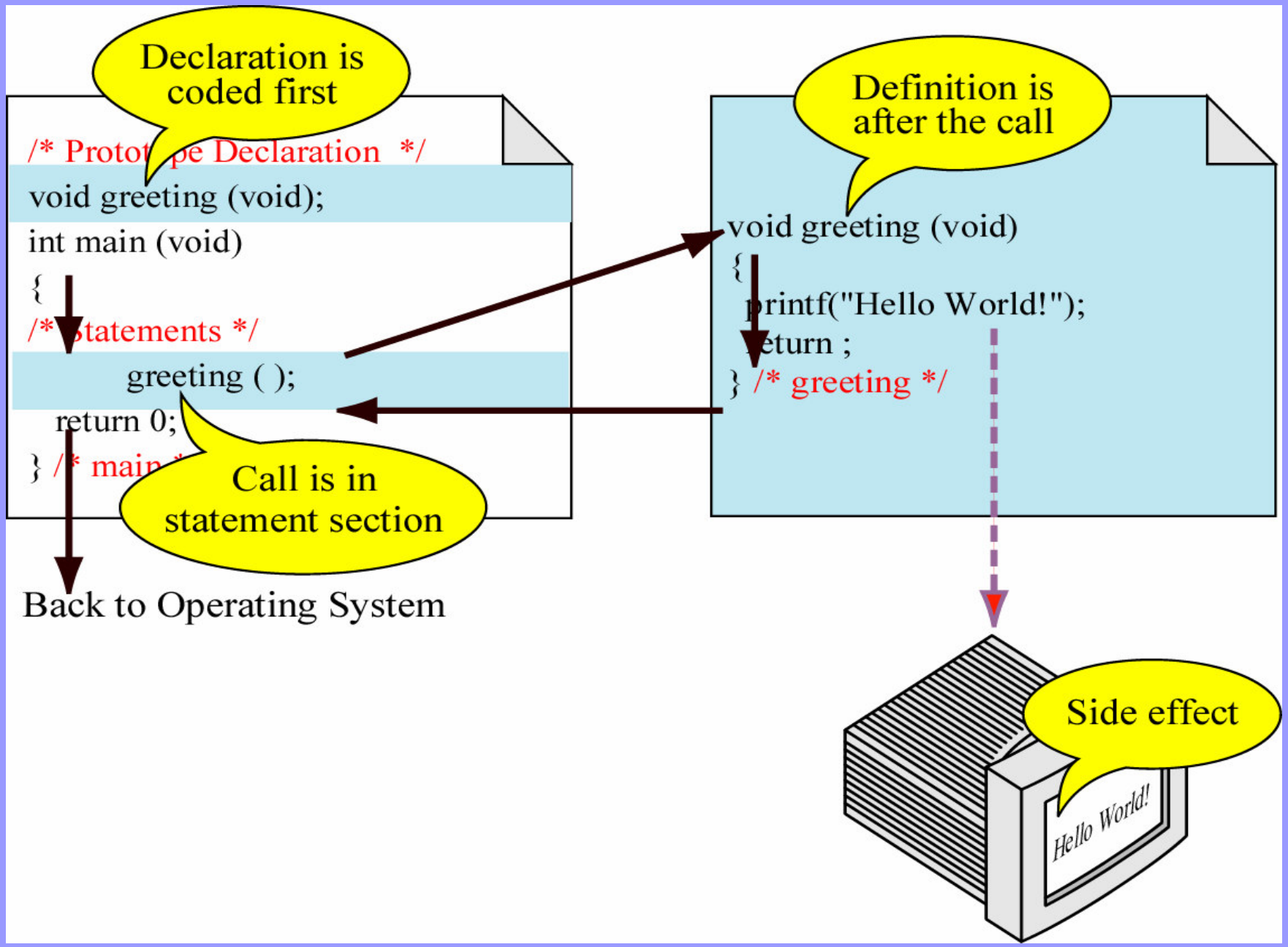
Then the flow of control passes to the first statement in the function's body. The called function's body statements are executed until one of these occurs:

return statement (with or without a return value),

or,

closing brace of function body.

Then control goes back to where the function was called.


Declaration is coded first
/* Prototype Declaration */
void greeting (void);
int main (void)
{
/* statements */
greeting ( );
return 0;
} /* main */
Call is in statement section
Back to Operating System
Definition is after the call
void greeting (void)
{
printf("Hello World!");
return ;
} /* greeting */
Side effect
Hello World!

Function Concepts

- Function Prototype
- Function Call
- Function Definition

What is in a prototype?

A prototype looks like a heading but must end with a semicolon;
and its parameter list just needs to contain the type of each parameter.

```
int  Cube( int  );      /* prototype */
```

Function Call

- **a function call temporarily transfers control to the called function's code**
- **when the function's code has finished executing, control is transferred back to the calling block**

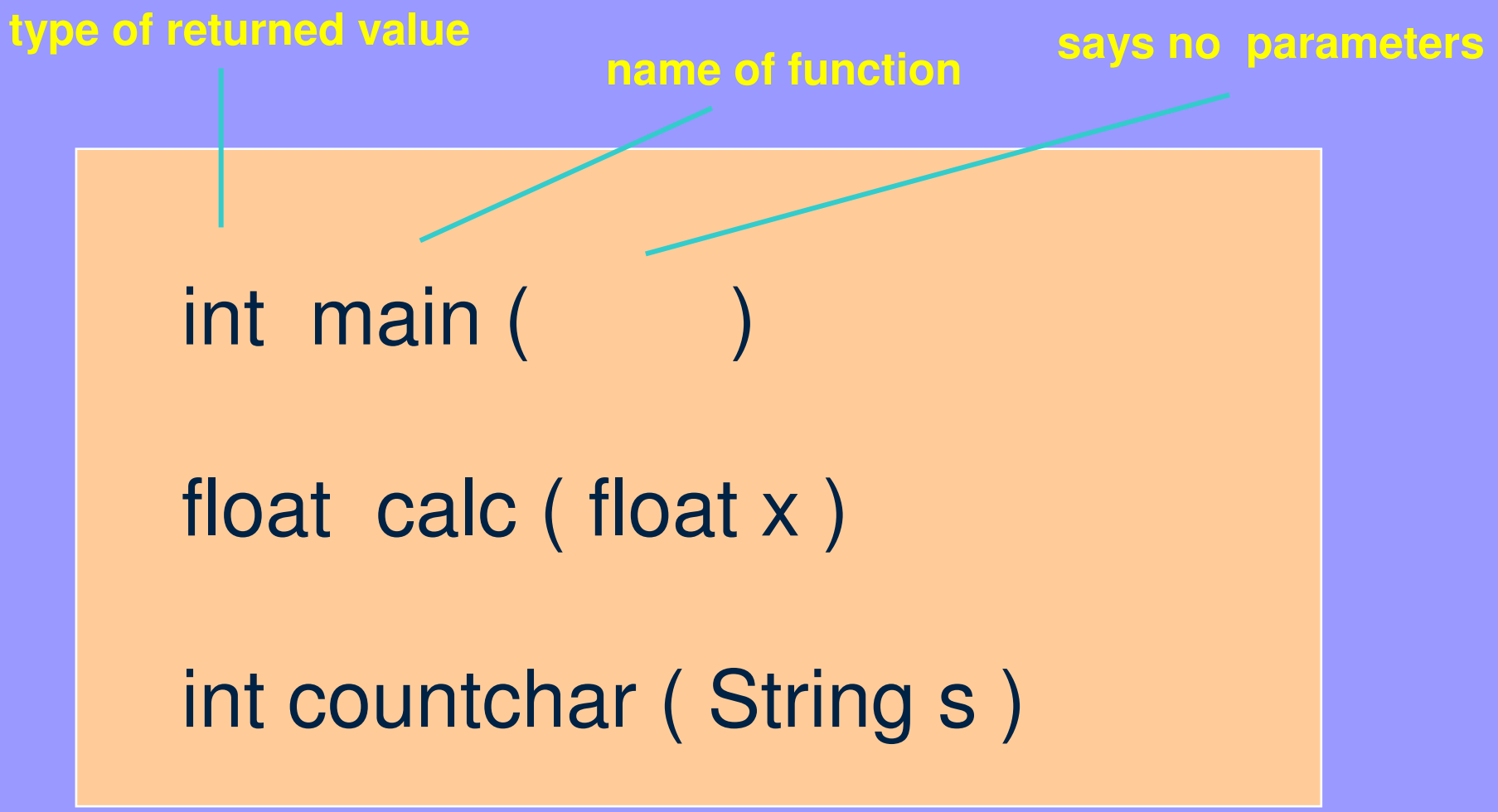
Function Call Syntax

FunctionName (Argument List)

The argument list is a way for functions to communicate with each other by passing information.

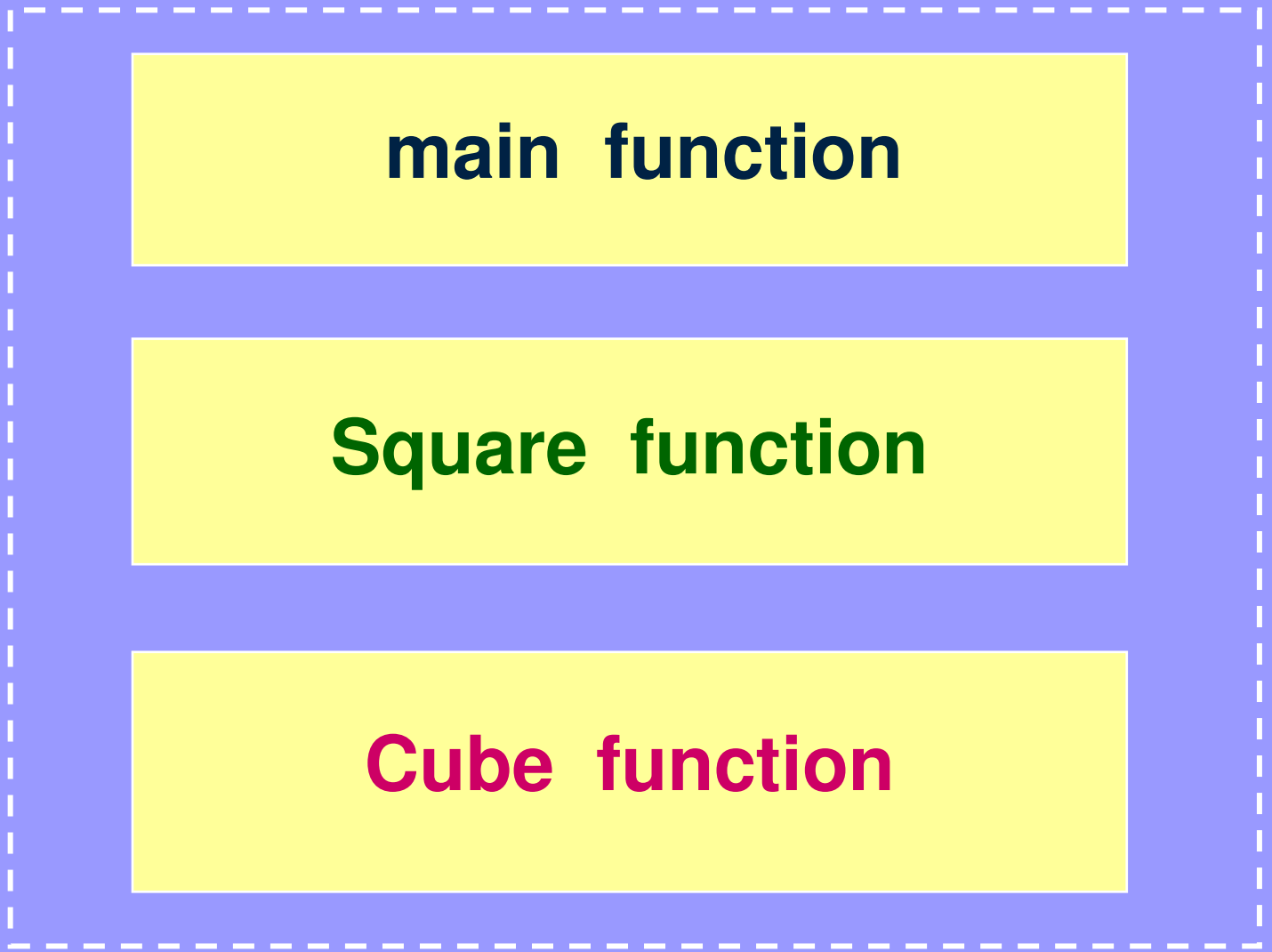
The argument list can contain 0, 1, or more arguments, separated by commas, depending on the function.

What is in a heading?



HEADER FILE	FUNCTION	EXAMPLE OF CALL	VALUE
<stdlib.h>	abs(i)	abs(-6)	6
<math.h>	pow(x,y)	pow(2.0,3.0)	8.0
	fabs(x)	fabs(-6.4)	6.4
<math.h>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<math.h>	log(x)	log(2.0)	.693147
<stdio.h>	printf( )		

Program with Several Functions



Program with Three Functions

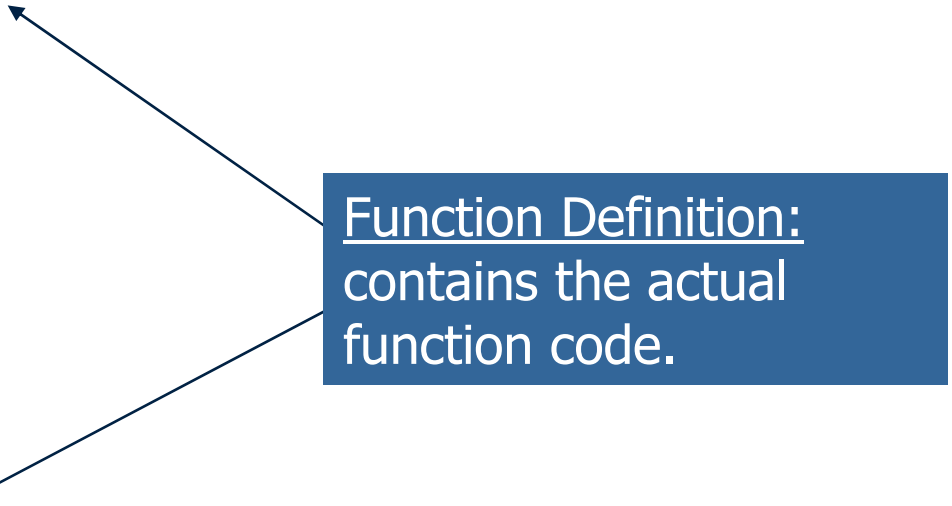
```
#include <stdio.h>

int Square( int );     /* declares these functions */
int Cube( int );

int main( )
{
      int num = 3 ;
      int sq , cb ;

      sq = Square(num);
      printf(" The square of 3 is %d \n ", sq );
      cb = Cube(num);
      printf(" The cube of 3 is %d \n ", cb );

      return 0;
}
```

Function Prototype: must be declared at top of program.

Function Call: invokes the function

Rest of Program

```
int Square( int n )
{
   int s = n * n ;
   return s;
}


int Cube( int n )
{
    int c = n * n * n ;
    return c;
}
```



A void function call stands alone

```
#include <stdio.h>

void  DisplayMessage ( int  n ) ;      /*declares function */


int main(  )
{
     DisplayMessage( 15 ) ;              /* function call */

     printf(  “Good Bye \n “ ) ;

     return 0 ;
}
```

A void function does NOT return a value

```
void  DisplayMessage ( int  n )
{
     printf( " I have liked math for %d years \n " , n );
}
```

Two Kinds of Functions

Value-Returning	Void
Always returns a single value to its caller and is called from within an expression.	Never returns a value to its caller, and is called as a separate statement.

```
#include <stdio.h>
int Cube ( int ) ;                          /* prototype */
void  main (  )
{
   int      yourNumber = 14;
   int      myNumber = 9 ;

printf(  "My Number = %d \n " ,  myNumber ) ;
   printf( " its cube is  %d \n " , Cube (myNumber) ) ;

printf(  "Your Number = %d \n " ,  yourNumber ) ;
   printf( " its cube is  %d \n " , Cube (yourNumber) ) ;

}
```

arguments

Write a void function

called DisplayMessage ( ) which you can call from main ( ) to describe the pollution index value it receives as a parameter.

Your city describes a pollution Index

less than 35 as “Pleasant”,

35 through 60 as “Unpleasant”,

and above 60 as “Health Hazard.”

```
#include <stdio.h>
void DisplayMessage (int);                    /* prototype */


int main ( )                                  argument
{
        int pollutionIndex;

        printf(" Enter pollution index : ");
        scanf( " %d", &pollutionIndex );
        DisplayMessage(pollutionIndex);       /* call */
        return 0;
}
```

parameter

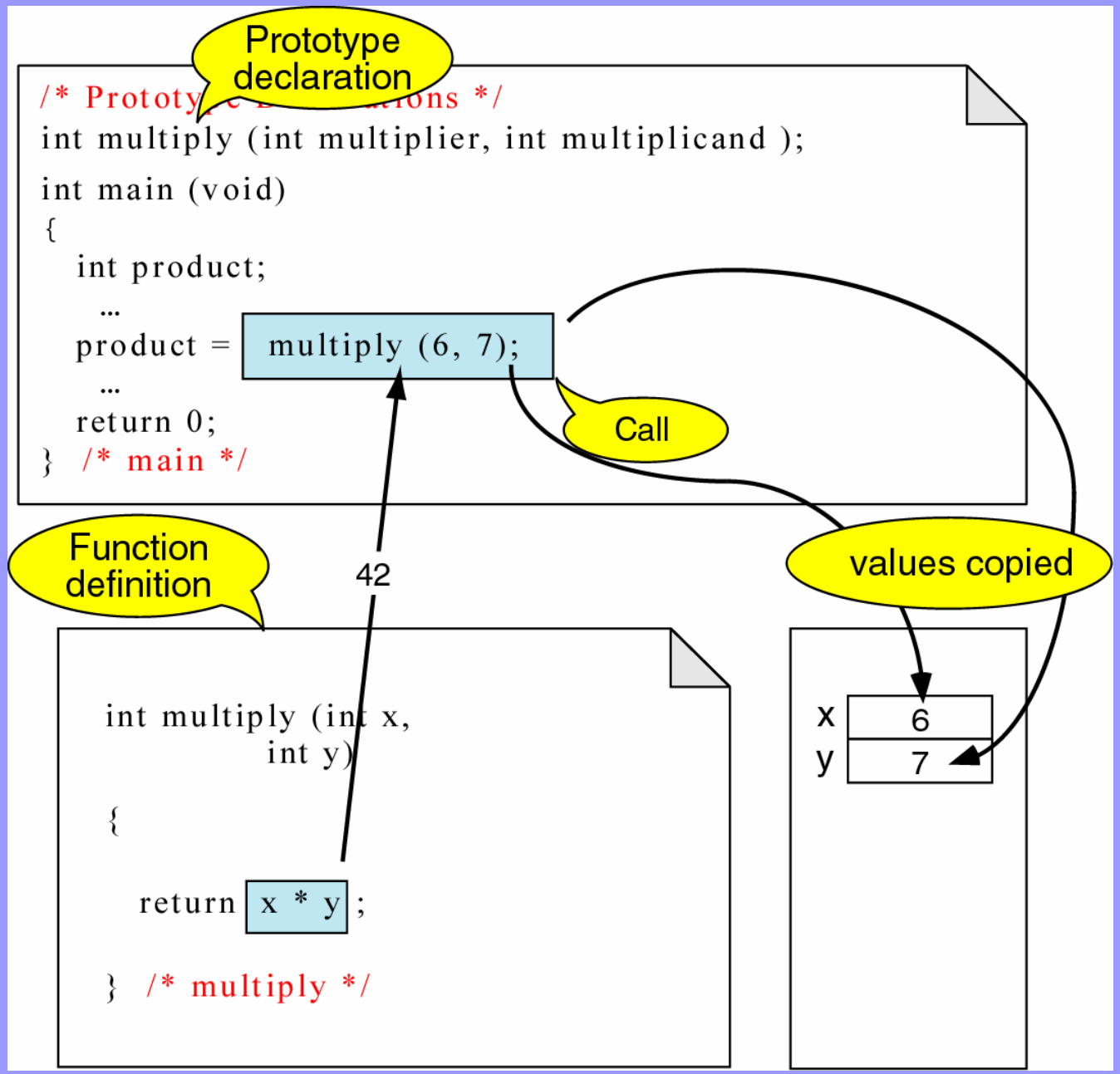
```
void DisplayMessage( int index )

{
        if ( index < 35 )
                printf( “Pleasant” ) ;
        else if ( index <= 60 )
                printf(  “Unpleasant” ) ;
        else
                printf ( “Health Hazard” );
}
```

Parameter List

- is the means used for a function to share information with the block containing the call

Classified by Location

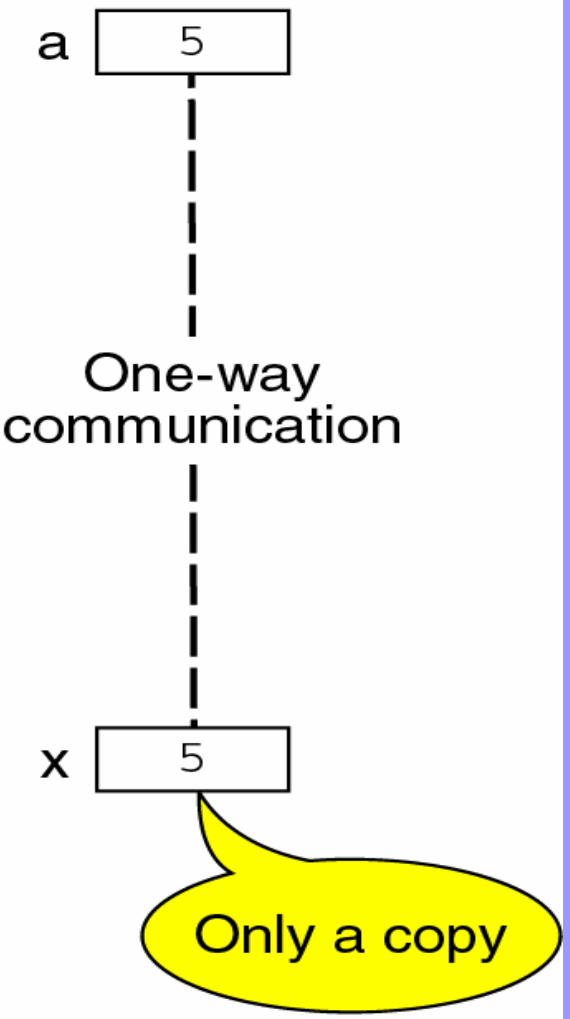
Arguments	Parameters
Always appear in a function call within the calling block.	Always appear in the function heading, or function prototype.


Prototype declaration
/* Prototype Declarations */
int multiply (int multiplier, int multiplicand);
int main (void)
{
int product;
...
product = multiply (6, 7);
...
return 0;
} /* main */
Call
Function definition
42
values copied
int multiply (int x,
int y)
{
return x * y ;
} /* multiply */
x
6
y
7

```c
/* Prototype Declarations */
void fun (int num1);


int main (void)
{
/* Local Definitions */
   int  a = 5;

/* Statements */
   fun (a)
   printf("%d\n", a);

   return 0;
}  /* main */
```

prints 5

```c
void fun (int x)
{
/* Statements */
   x = x + 3;
   return;
}  /* fun*/
```



MAIN PROGRAM MEMORY

If you pass only a copy of 25 to a function, it is called "pass-by-value" and the function will not be able to change the contents of age. It is still 25 when you return.

MAIN PROGRAM MEMORY

BUT, if you pass 4000, the address of age to a function, it is called "pass-by-reference" and the function will be able to change the contents of age. It could be 23 or 90 when you return.

4000

25

age

Argument in Calling Block

Value Parameter

The value (25) of the argument
is passed
to the function when
it is called.

Reference Parameter

The memory address (4000)
of the argument
is passed
to the function when
it is called.

By default, parameters

are always value parameters, unless you do something to change that.

To get a reference parameter you need to place & after the type in the function heading and prototype.

When to Use Reference Parameters

- reference parameters should be used when you want your function to give a value to, or change the value of, a variable from the calling block.

- Return more than one variables.

Example of Pass-by-Reference

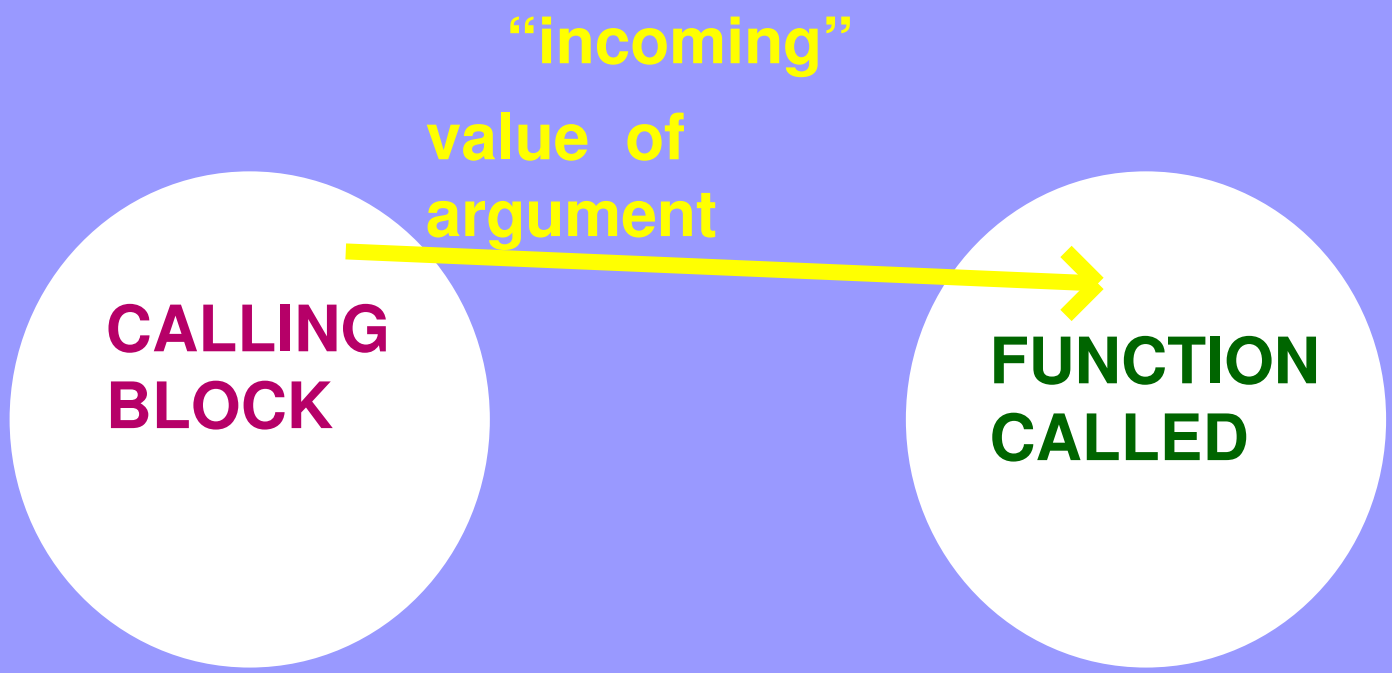
We want to find 2 real roots for a quadratic equation with coefficients a,b,c. Write a prototype for a void function named GetRoots( ) with 5 parameters. The first 3 parameters are type float. The last 2 are reference parameters of type float.

```
#include <stdio.h>
#include <math.h>
void GetRoots(float, float, float, float&, float&);

void main ( )
{
     float a, b, c, first, second;
      printf(" Enter a b c : ");
      scanf("%f %f %f", &a , &b, &c );
     GetRoots(a, b, c, first, second);
     printf(" Roots are %.2f %.2f \n " , first , second
   );

}

// function GetRoots goes here
.....
```

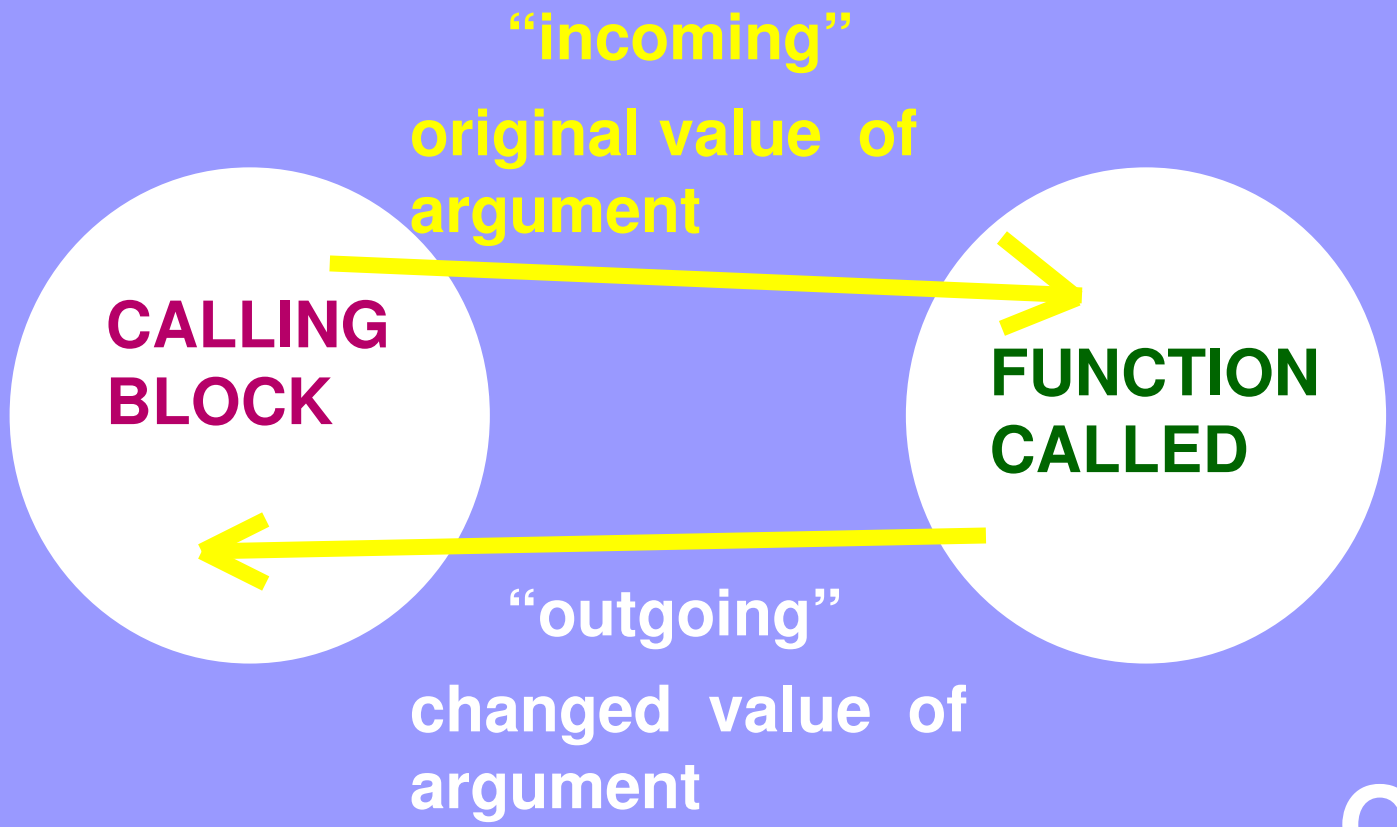
Function Definition

```
void  GetRoots( float a,  float b,  float c,
                float& root1, float& root2)
{
   float temp;                              // local variable

   temp = b * b - 4.0 * a * c;

   root1 = (-b + sqrt(temp) ) / ( 2.0 * a );

   root2 = (-b - sqrt(temp) ) / ( 2.0 * a );

   return;
}
```

Pass-by-value

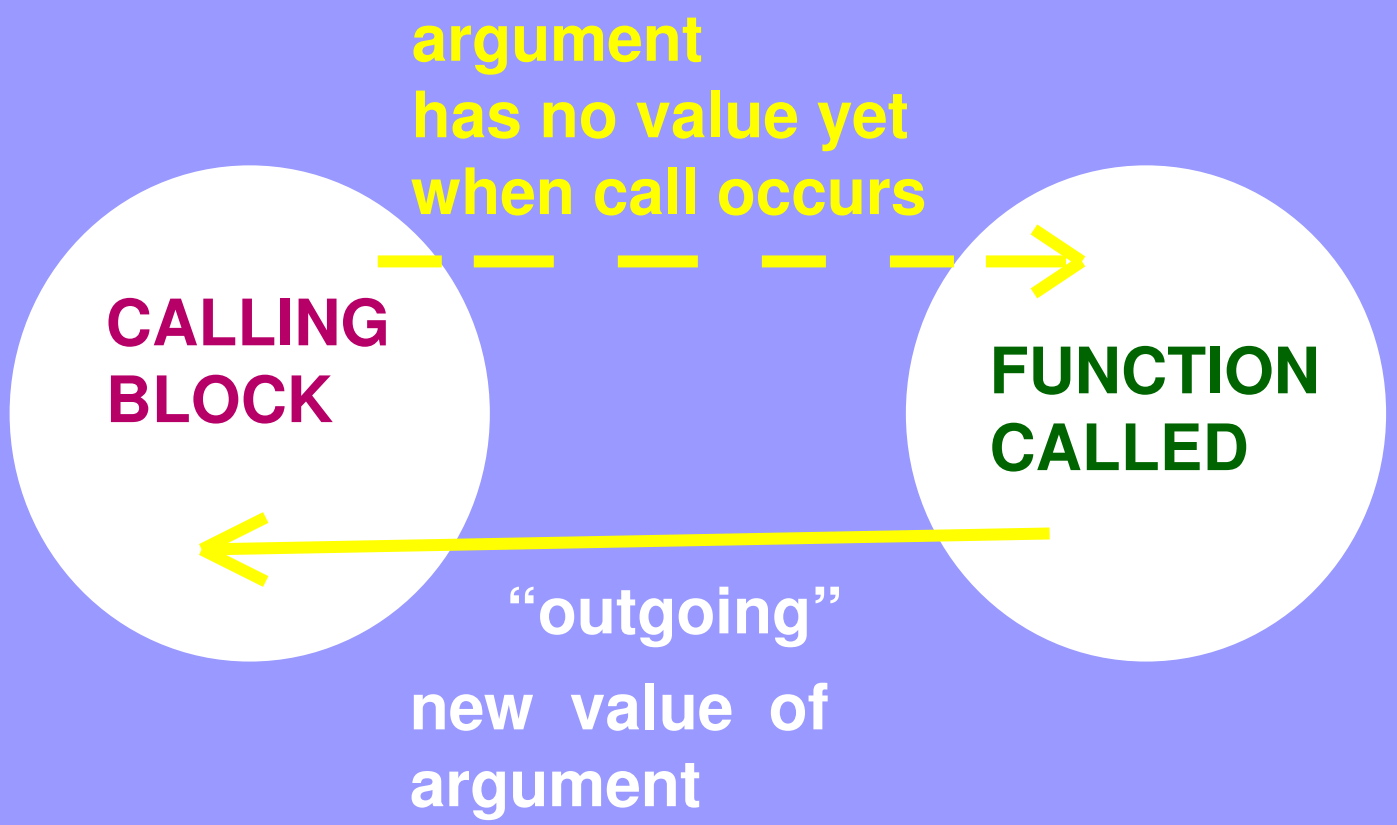


Pass-by-reference



OR,

Pass-by-reference



Write prototype and function definition for

- a void function called GetRating( ) with one reference parameter of type char
- the function repeatedly prompts the user to enter a character at the keyboard until one of these has been entered: E, G, A, P to represent Excellent, Good, Average, Poor

void GetRating(char&); *// prototype*

```
void GetRating (char&   letter)
{
        printf("Enter Employee rating \n");
        printf("Use E , G , A , or P : ");
        scanf("%c", &letter );

        while ( (letter  != 'E')  &&  (letter  != 'G')  &&
                          (letter  != 'A')  &&  (letter  !=
   'P') )
        {
                printf( "Rating invalid.  Try again: \n ");
                scanf("%c", &letter);
        }
}
```

Example 1

- Write a function that receives a positive integer & returns its factorial.
- Write a program that prints out the factorial of numbers 1 .. 20. (Use the function above)

Example 2

- Write a function that receives a positive integer & returns 1 if it is prime and 0 otherwise.
- Write a program that prints out all prime numbers 2-100 .