# Functions

MMJ12503 – Computer programming

function()

# Refresh of previous lecture

- Function call statement has the following syntax.

```
function_name(variable1, variable2,…);
```

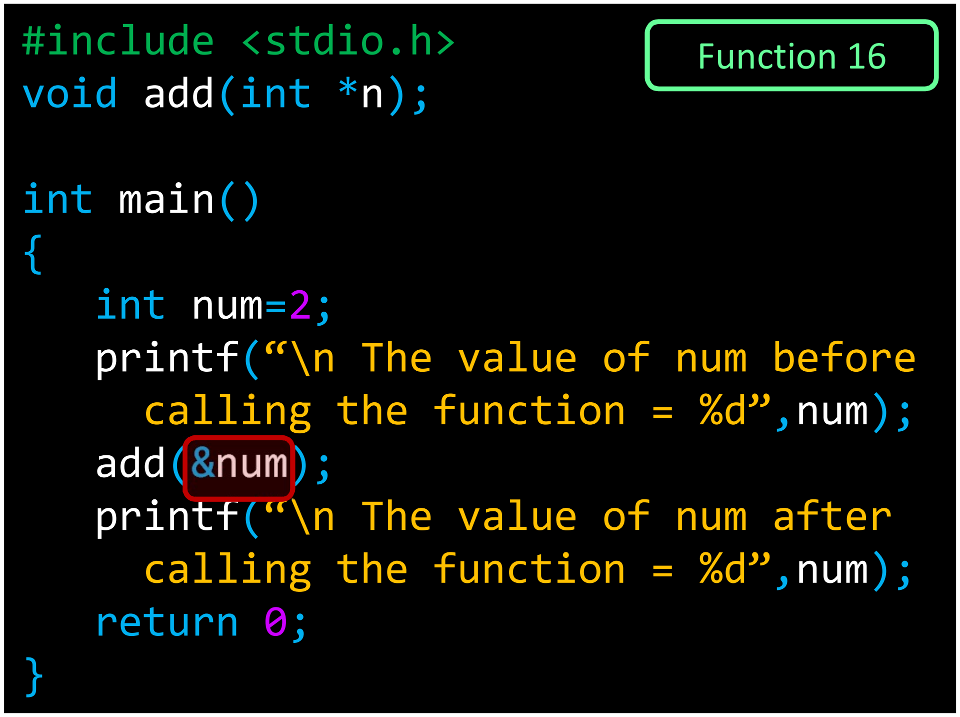
Don’t forget about semicolon at end of statement for function declaration

Function 02*

```
#include <stdio.h>

int square(int m);

void main(void)
{
   int x = 5;
   int y;
   y=square(x);
   return 0;
}

int square(int a)
{
   return (a*a);
}
```

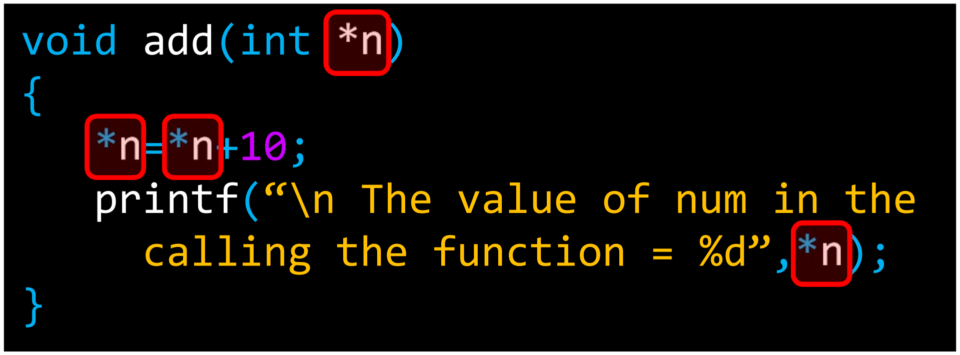
# Refresh of previous lecture

- To indicate that an argument is passed using call by reference, an ampersand sign (&) is placed after the type in the parameter list.
  - & - the address operator.
  - * - indirection operator.



Function 16

```
#include <stdio.h>
void add(int *n);

int main()
{
   int num=2;
   printf(“\n The value of num before
     calling the function = %d”,num);
   add(&num);
   printf(“\n The value of num after
     calling the function = %d”,num);
   return 0;
}
```



```
void add(int *n)
{
   *n=*n+10;
   printf(“\n The value of num in the
     calling the function = %d”,*n);
}
```

- The syntax of call a function declaration and function definition.

```
function_name(variable1, variable2,…);
```

- The syntax of call by reference.

Function 16

```
#include <stdio.h>
void add(int *n);

int main()
{
   int num=2;
   printf(“\n The value of num before
     calling the function = %d”,num);
   add(&num);
   printf(“\n The value of num after
     calling the function = %d”,num);
   return 0;
}
```

```
void add(int *n)
{
   *n=*n+10;
   printf(“\n The value of num in the
     calling the function = %d”,*n);
}
```

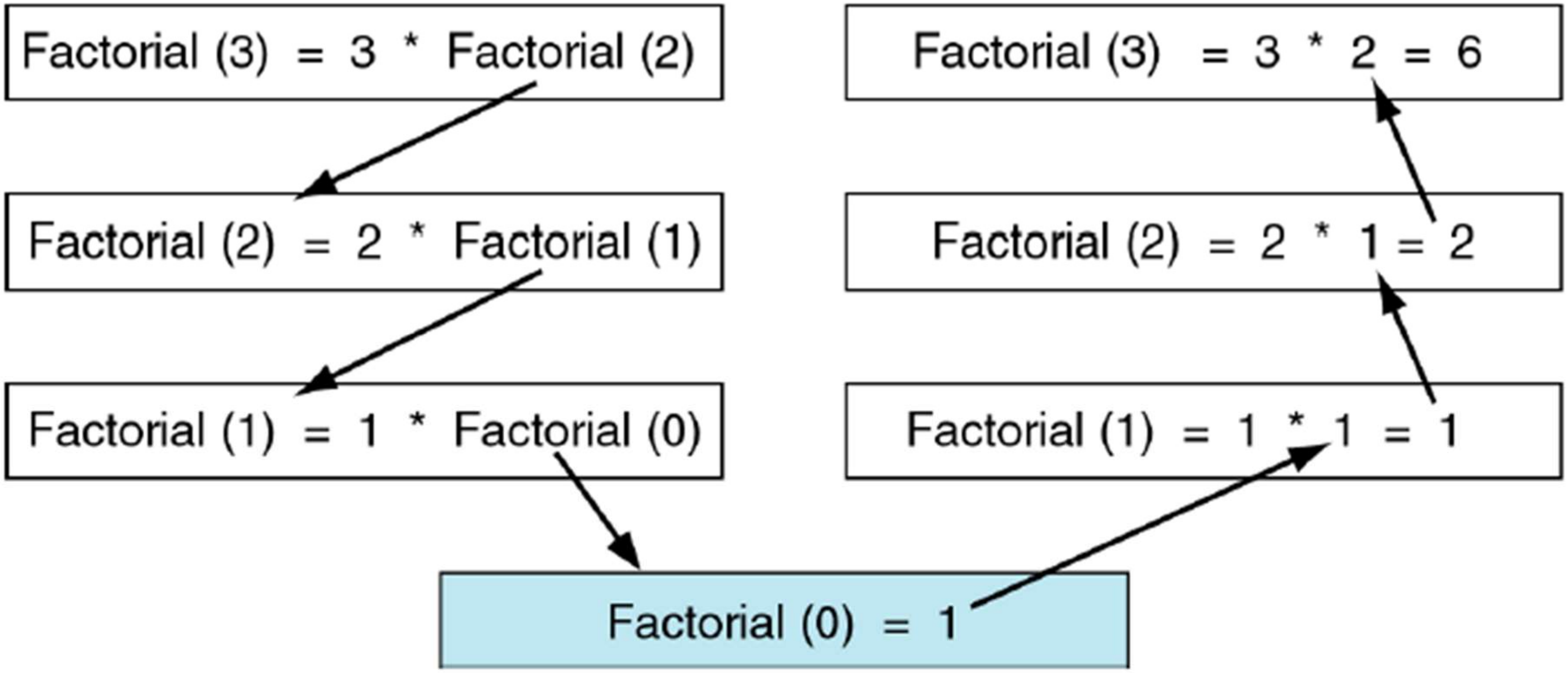
# Recursive Functions

- Recursion is a term describing functions which are called by themselves (A function that calls itself).
- Recursive function has two elements:
  - ➢Each call either solves one part of the problem or it reduces the size of the problem.
  - ➢The statement that solves the problem is known an base case. Every recursive function must have a base case. The rest of the function is known as the general case.
- Recursion is very useful in mathematical calculations and in sorting of lists.

- To understand recursive functions, let take an example of calculating factorial of a number.
- Factorial:
  - ➢$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$
  - ➢$n! = n \times (n-1)!$
- Base case $1! = 1$.

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

$$n! = n \times (n-1)!$$



Factorial (3) = 3 * Factorial (2)

Factorial (2) = 2 * Factorial (1)

Factorial (1) = 1 * Factorial (0)

Factorial (0) = 1

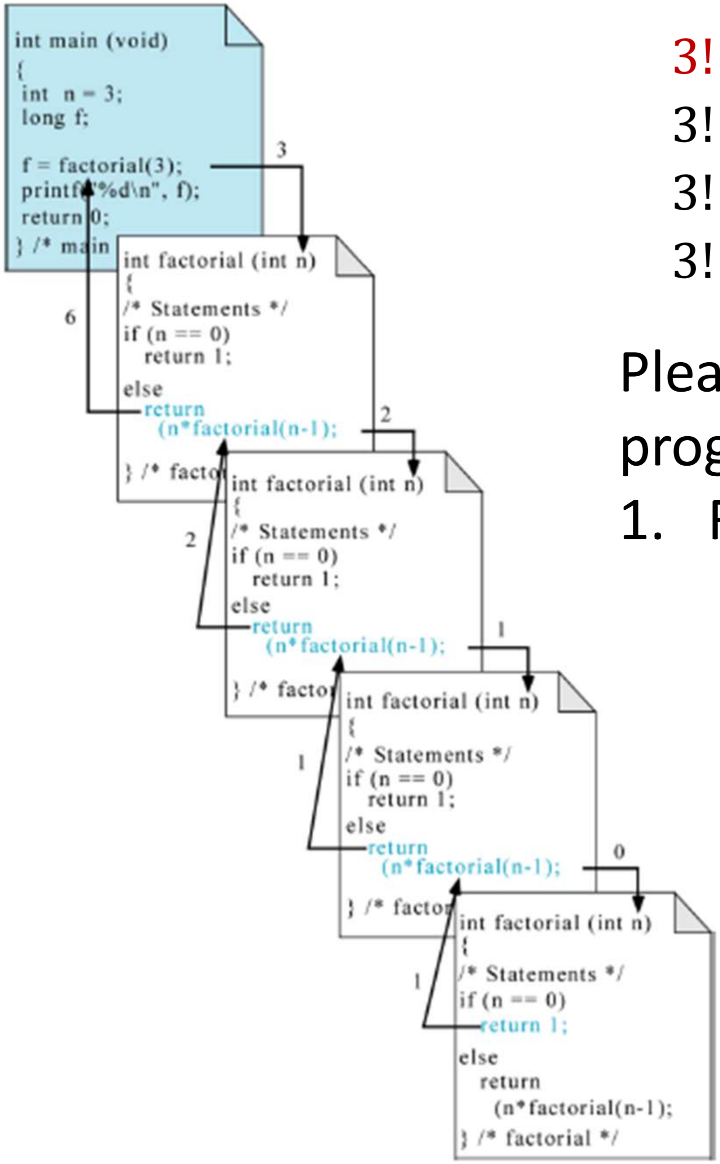
Factorial (1) = 1 * 1 = 1

Factorial (2) = 2 * 1 = 2

Factorial (3) = 3 * 2 = 6

```
/*  Recursive functions  */
#include <stdio.h>
int factorial(int num);

int main(void)
{
   int n=3;
   long f;
   f=factorial(n);
   printf (“%d\n”,f);
   return 0;
}

int factorial(int n)
{
/*  statement  */
   if (n==0)
     return 1;
   else
     return(n*factorial(n-1));
}
```

Function 18



```
int main (void)
{
 int  n = 3;
 long f;

 f = factorial(3);
 printf("%d\n", f);
 return 0;
} /* main
```

```
int factorial (int n)
{
/* Statements */
if (n == 0)
   return 1;
else
   return
     (n*factorial(n-1);

} /* facto
```

```
int factorial (int n)
{
/* Statements */
if (n == 0)
   return 1;
else
   return
     (n*factorial(n-1);

} /* facto
```

```
int factorial (int n)
{
/* Statements */
if (n == 0)
   return 1;
else
   return
     (n*factorial(n-1);

} /* factor
```

```
int factorial (int n)
{
/* Statements */
if (n == 0)
   return 1;

else
   return
     (n*factorial(n-1);
} /* factorial */
```

$3! = 3 \times 2 \times 1$

$3! = 3 \times 2!$

$3! = 3 \times 2 \times 1!$

$3! = 3 \times 2 \times 1$

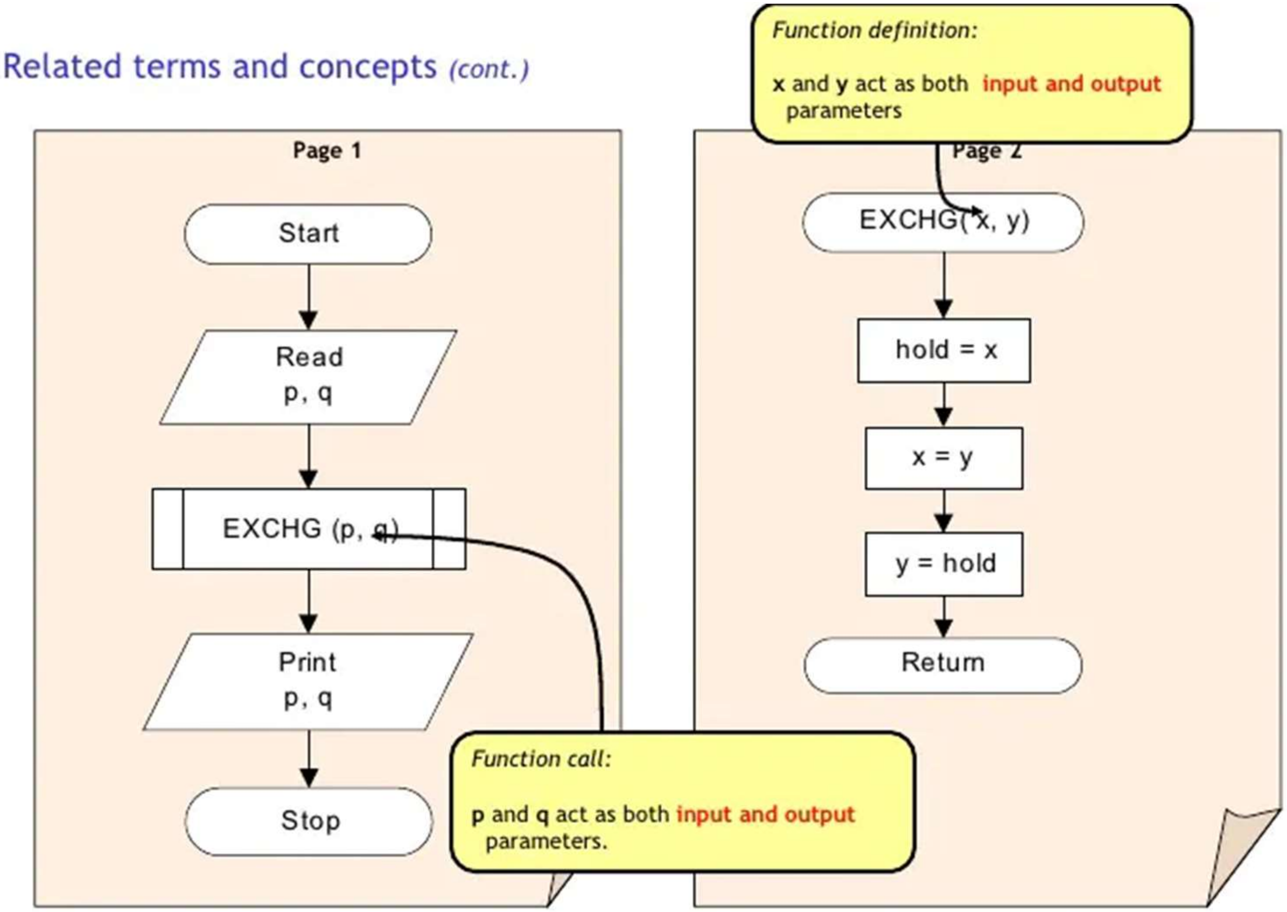
Please try the given programs:

1. Finding exponents

```
/*  Finding exponents x^y */
#include <stdio.h>
int exp_rec(int, int);

main()
{
   int num1, num2, res;
   printf(“\n Enter the value for x and y
     for x^y : ”);
   scanf(“%d %d”, &num1, &num2);
   res=exp_rec(num1, num2);
   printf("\n x^y is : %d",res);
   return 0;
}

int exp_rec(int x, int y)
{
   if (y==0)
     return 1;
   else
     return (x*exp_rec(x,y-1));
}
```

Function 19

# Functions in flowchart



Related terms and concepts *(cont.)*

*Function definition:*

**x** and **y** act as both input and output parameters

*Function call:*

**p** and **q** act as both input and output parameters.

This flowchart exchanges or swaps the value of x and y each other

# Re-cap

1. Function with no return value
2. Function with return value
3. Functions return more than one value

```
main()
{
    ......
    ......
    func1();
    ......
    ......
    return 0;
}
```

```
func1()
{
    ......
    ......
    ......
    ......
}
```

```
main()
{
    ......
    ......
    func1();
    ......
    ......
    return 0;
}
```

```
func1()
{
    ......
    ......
    ......
    ......
    return var;
}
```

```
main()
{
    ......
    ......
    func1(&var1,&var2);
    ......
    ......
    return 0;
}
```

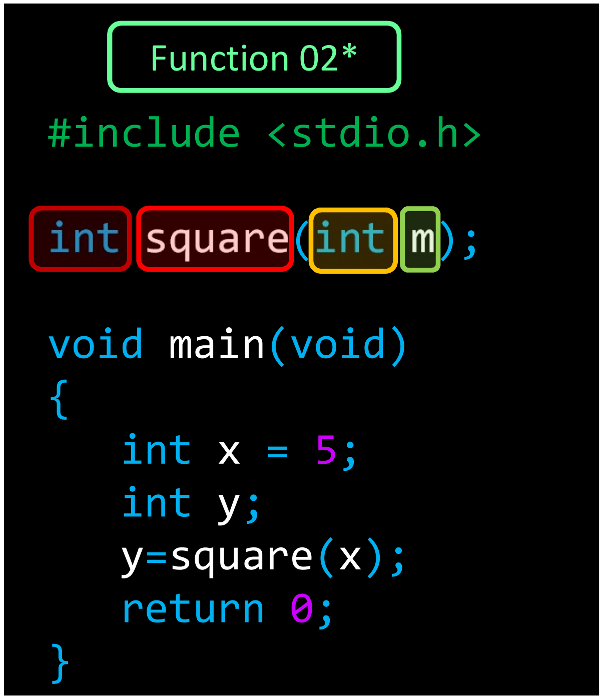
```
func1(*var1,*var2)
{
    ......
    ......
    *var1
    *var2
    ......
    return;
}
```

ARE YOU OKAY?

# Review of this chapter

- The syntax of a function declaration can be given as:

```
Return_data_type function_name(data_type variable1, data_type variable2,…);
```



Function 02*

```c
#include <stdio.h>

int square(int m);

void main(void)
{
    int x = 5;
    int y;
    y=square(x);
    return 0;
}
```

Don’t forget about semicolon at end of statement for function declaration

# Review of this chapter

- The syntax of a function definition can be given as:

```
Return_data_type function_name(data_type variable1, data_type variable2,…)
{
  ……
  statements
  ……
  return(variable);
}
```

Function 02*

```
#include <stdio.h>

int square(int m);

void main(void)
{
   int x = 5;
   int y;
   y=square(x);
   return 0;
}
```

```
int square(int a)
{
   return (a*a);
}
```

Don’t write semicolon at end of statement for function defination

# Review of this chapter

- Function call statement has the following syntax.

```
function_name(variable1, variable2,…);
```

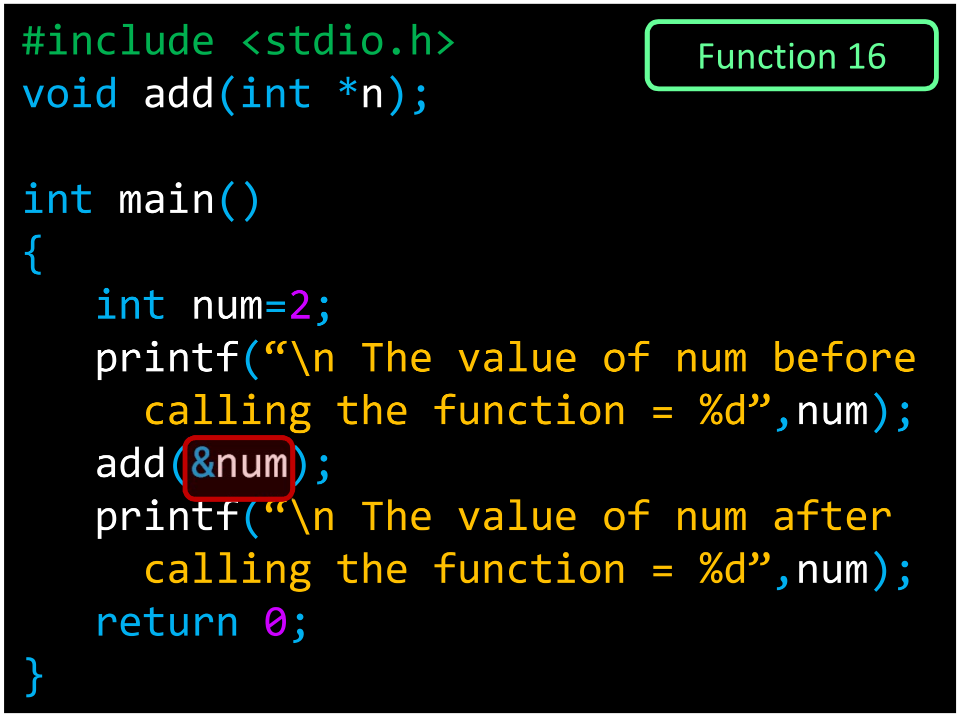
Don’t forget about semicolon at end of statement for function declaration

Function 02*

```
#include <stdio.h>

int square(int m);

void main(void)
{
   int x = 5;
   int y;
   y=square(x);
   return 0;
}

int square(int a)
{
   return (a*a);
}
```

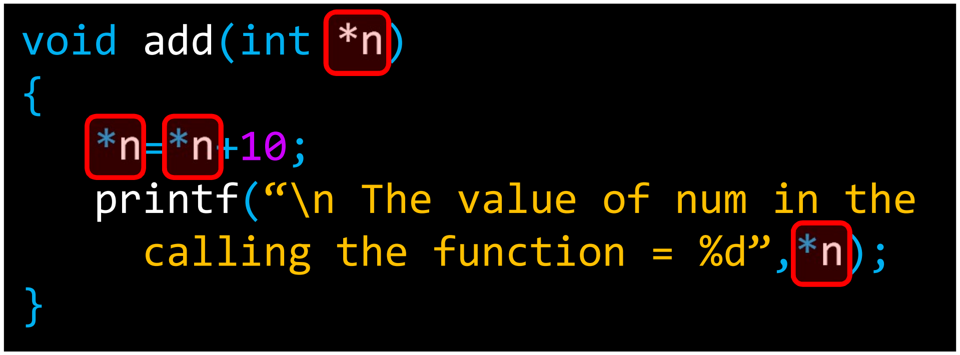
# Review of this chapter

- To indicate that an argument is passed using call by reference, an ampersand sign (&) is placed after the type in the parameter list.
  - & - the address operator.
  - * - indirection operator.



Function 16

```
#include <stdio.h>
void add(int *n);

int main()
{
   int num=2;
   printf(“\n The value of num before
     calling the function = %d”,num);
   add(&num);
   printf(“\n The value of num after
     calling the function = %d”,num);
   return 0;
}
```



```
void add(int *n)
{
   *n=*n+10;
   printf(“\n The value of num in the
     calling the function = %d”,*n);
}
```