Functions

MMJ12503 – Computer programming

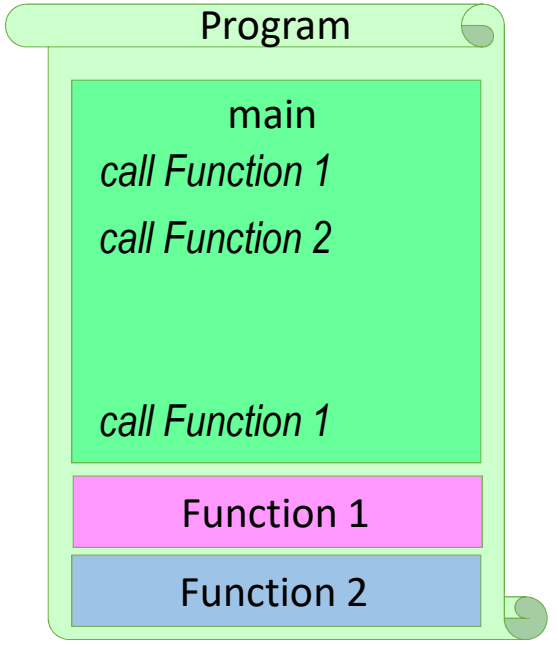
function()

Refresh of previous lecture

- C program decomposes a program into its component functions.
- Function is a series of statements that have been grouped together and given a name.

Problem

Function



Refresh of previous lecture

- The name of a function is used in three ways:
  1. for declaration
  2. in a call
  3. for definition
- A function must be first declared and defined.
- The argument names in the function declaration and function definition need not be the same. However, the data types of the arguments must match with that specified in function declaration as well as function definition.

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

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

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

Function 02*

```
#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

Refresh of previous lecture

- 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;
}
```

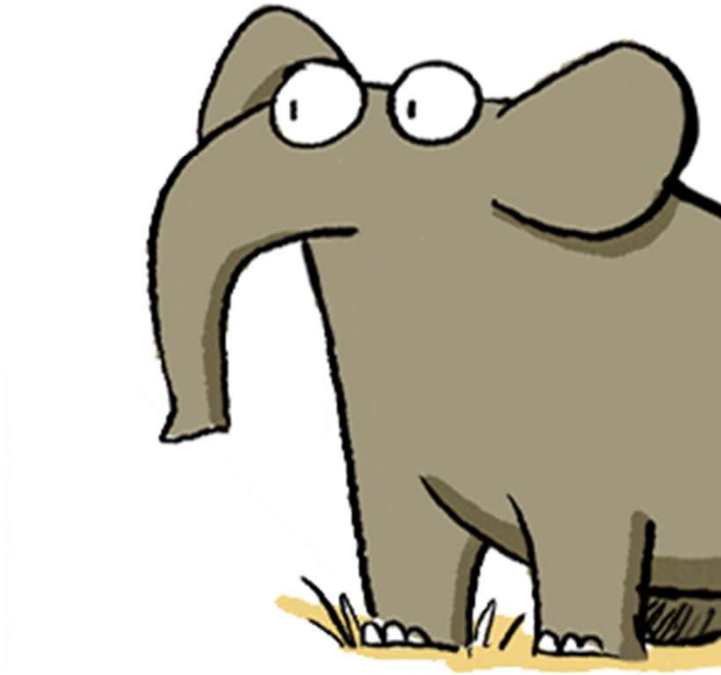
```
int square(int a)
{
   return (a*a);
}
```

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

- The syntax of a function declaration and function definition.

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

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



Return statement

- When a return statement in a function is executed, the function returns to its caller function.
- The return statement is optional in a function that does not return a value but, if used it is written as:

```
return;
```

- If the return is not specified, C will assume it is of type integer.
- If a function that does not have a return statement, the called function returns to the caller function after the last statement in the function's body is executed.

- A function that returns a value must have at least one return statement which may be written as:

`return(expression);` or `return expression;`

```
return(30*6+20);
```

```
return 30*6+20;
```

- A function can have any number of return statements. Of course only one return statement is executed per invocation, because the return statement returns control and a value to the caller function.
- As a rule of thumb keep the number of return statements small; otherwise the function becomes hard to understand, hard to debug and hard to alter.

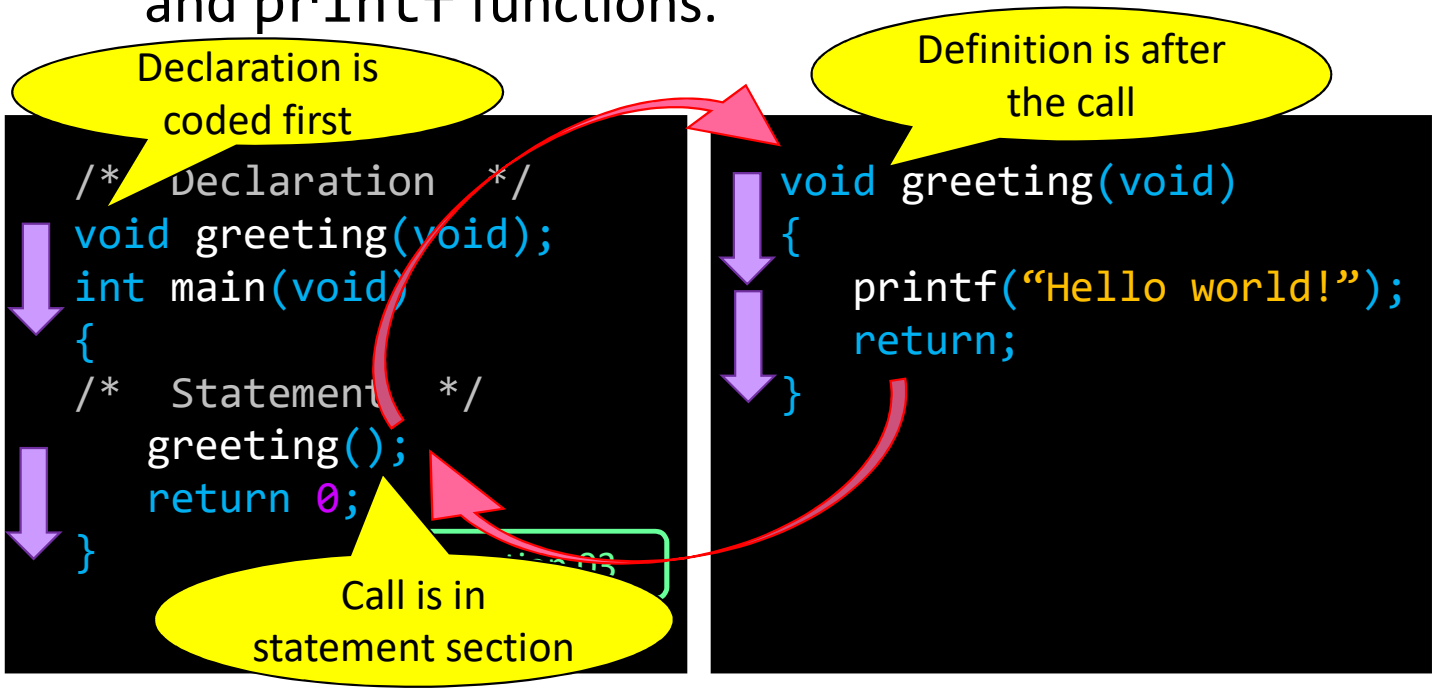
Function call

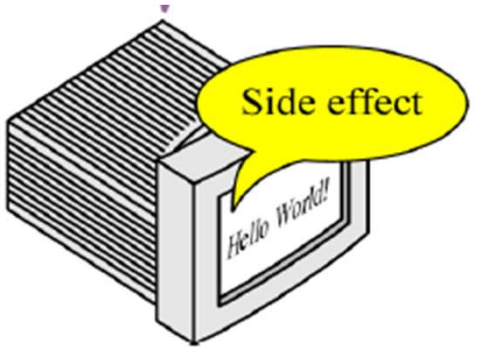
- The function call statement invokes the function.
- When the function is invoked the compiler jumps to the called function to execute the statements that are a part of that function. Once the called function is executed, the program control passes back to the calling function.
- Function call statement has the following syntax

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

- When the function declaration is present before the function call, the compiler can check if the correct number and type of arguments are used in the function call and the returned value, if any, is being used reasonably.

- Function definitions are often placed in a separate header file which can be included in other C source files that wish to use the functions.
- For example, the header file `stdio.h` contains the definition of `scanf` and `printf` functions.





Review of today lecture – part 1

- When a return statement in a function is executed, the function returns to its caller function.
- The return statement is optional in a function.
- The function call statement invokes the function.
- When the function is invoked the compiler jumps to the called function to execute the statements that are a part of that function. Once the called function is executed, the program control passes back to the calling function.

Review of today lecture – part 1

- 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*

```c
#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);
}
```

```c
/*  example of using function  */
/*  convert degree F to degree C */
#include <stdio.h>

float F2C(float far);

int main(void)
{
   float far,cen;
   printf("Enter temp in degree F : ");
   scanf("%f",&far);
   cen = F2C(far);
   printf("Temp in degree C is %f\n",cen);
   return 0;
}

float F2C(float far)
{
   return((5.0/9.0)*(far-32.0));
}
```

Function 04

Please try to identify the components of function in the given program:

1. Function declaration
2. Call function
3. Function definition

```c
/*  Add two numbers using functions  */
#include <stdio.h>
float addnumber(float num1, float num2);

void main(void)
{
   float n1,n2,sum;
   printf("please enter two numbers");
   scanf("%f %f",&n1,&n2);
   sum = addnumber(n1,n2);
   printf("%f + %f = %f\n",n1,n2,sum);
   return;
}

float addnumber(float num1, float num2)
{
   float result;
   result= num1 + num2;
   return(result);
}
```

Function 05

Please try to identify the components of function in the given program:

1. Function declaration
2. Call function
3. Function definition

```c
/*  Right angled triangle  */
#include <stdio.h>
#include <math.h>

float hypotenuse(float side1, float side2);
float areatri(float side1, float side2);
float perimeter(float side1, float side2);

void main(void)
{
   float s1,s2,area,hypo,peri;
   printf("please input the two sides ");
   scanf("%f %f",&s1,&s2);
   area = areatri(s1, s2);
   peri = perimeter(s1, s2);
   hypo = hypotenuse(s1, s2);
   printf("The sides of the triangle is %f
     %f\n",s1,s2);
   printf("The area = %f\n",area);
   printf("The perimeter = %f\n",peri);
   printf("The hypotenuse = %f\n",hypo);
   return;
}

float hypotenuse(float side1, float side2)
{
   return(sqrt(side1*side1+side2*side2));
}

float areatri(float side1, float side2)
{
   return(0.5*side1*side2);
}

float perimeter(float side1, float side2)
{
   float side3;
   side3=hypotenuse(side1,side2);
   return(side1+side2+side3);
}
```

Function 06

Side effect in a function

- A change of state in the program takes place due to the action of a function, which is termed as a side effect of a function.
- The side effect may be:
  - accepting data from outside the program.
  - sending data out of the program to a file or to the monitor.
  - changing the value of a variable in the calling function.

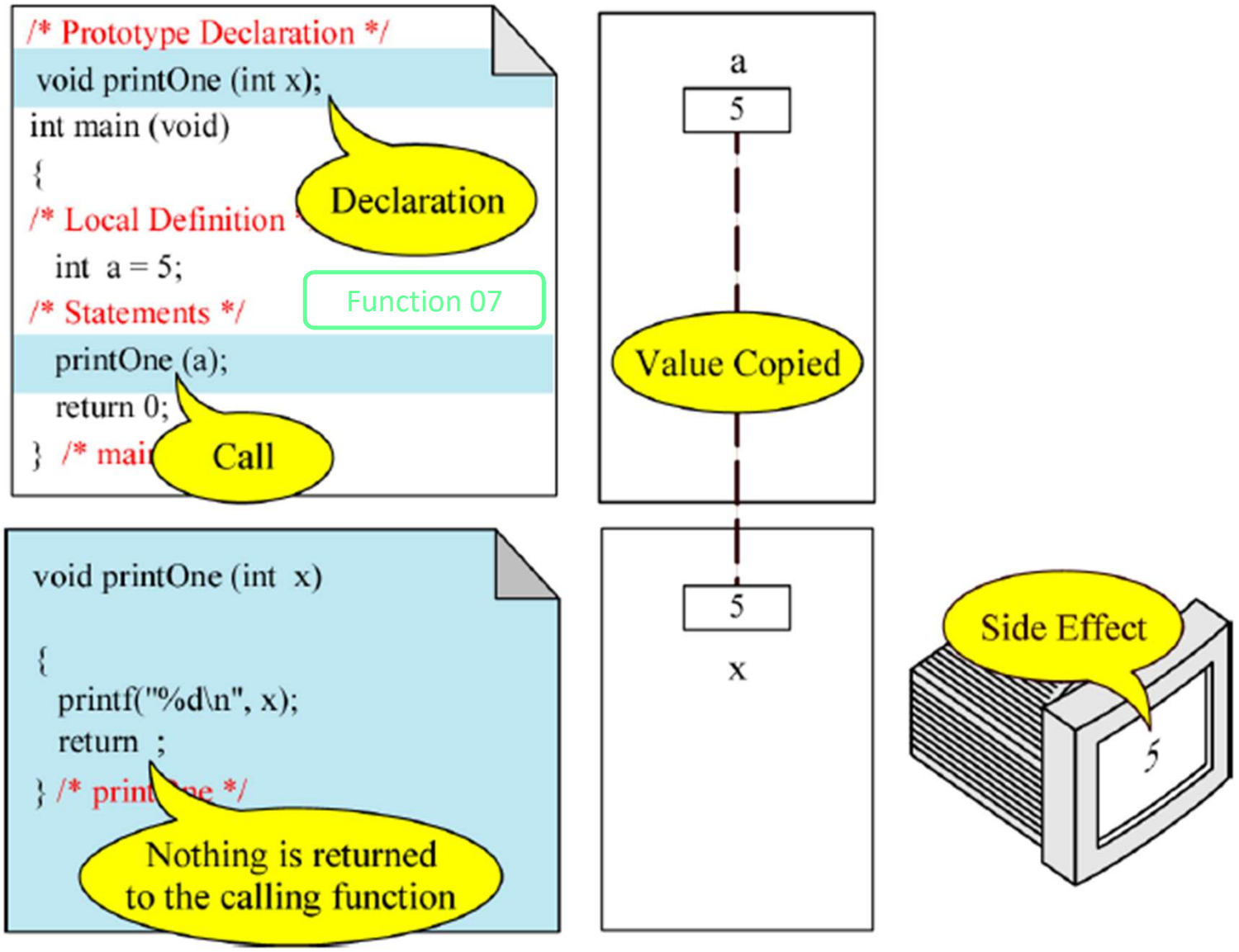
Passing parameters to the function

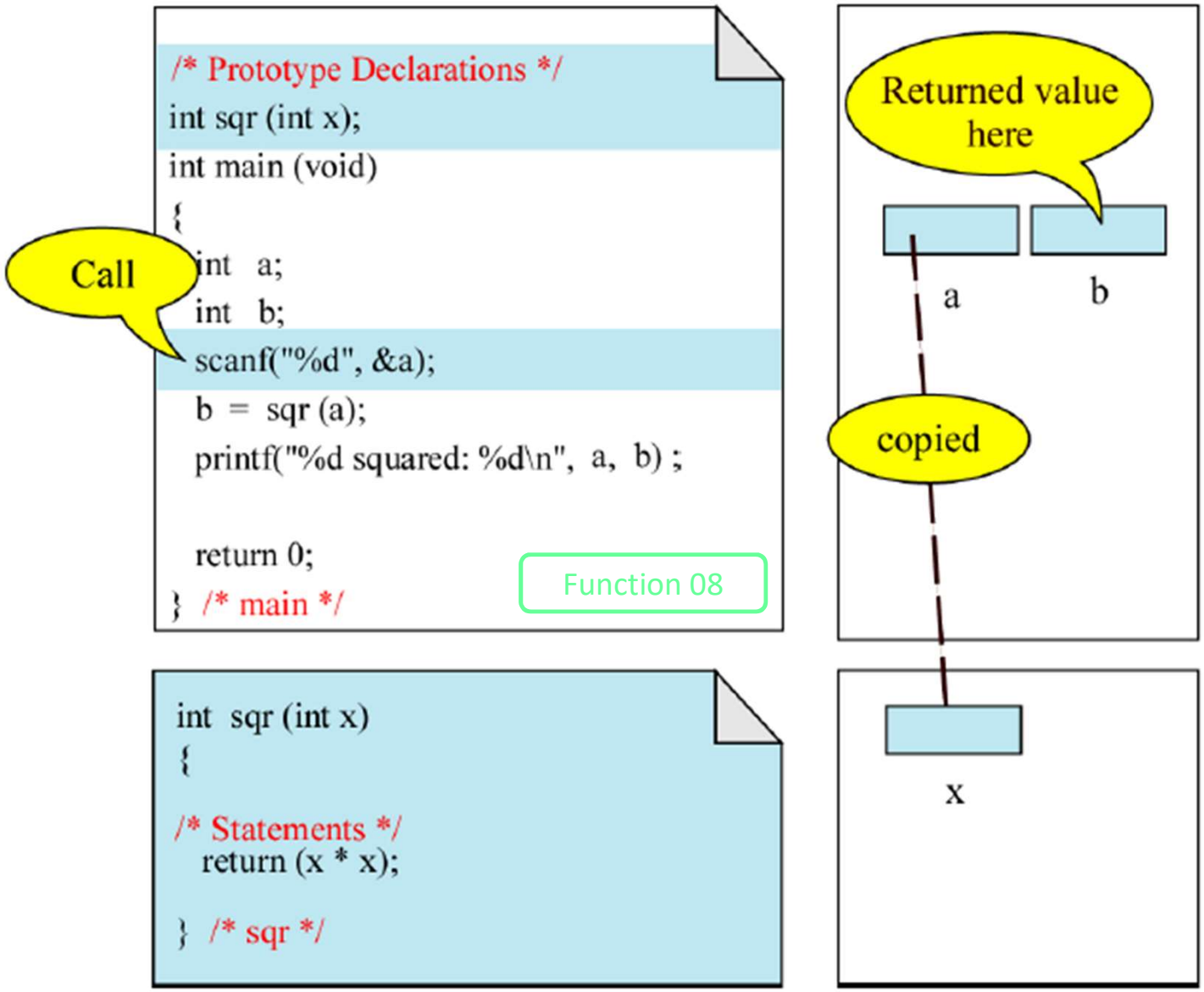
- When a function is called, the calling function may have to pass some values to the called function.
- There are two ways in which arguments or parameters can be passed to the called function. They include:
  - Call by value in which values of the variables are passed by the calling function to the called function.
  - Call by reference in which address of the variables are passed by the calling function to the called function.

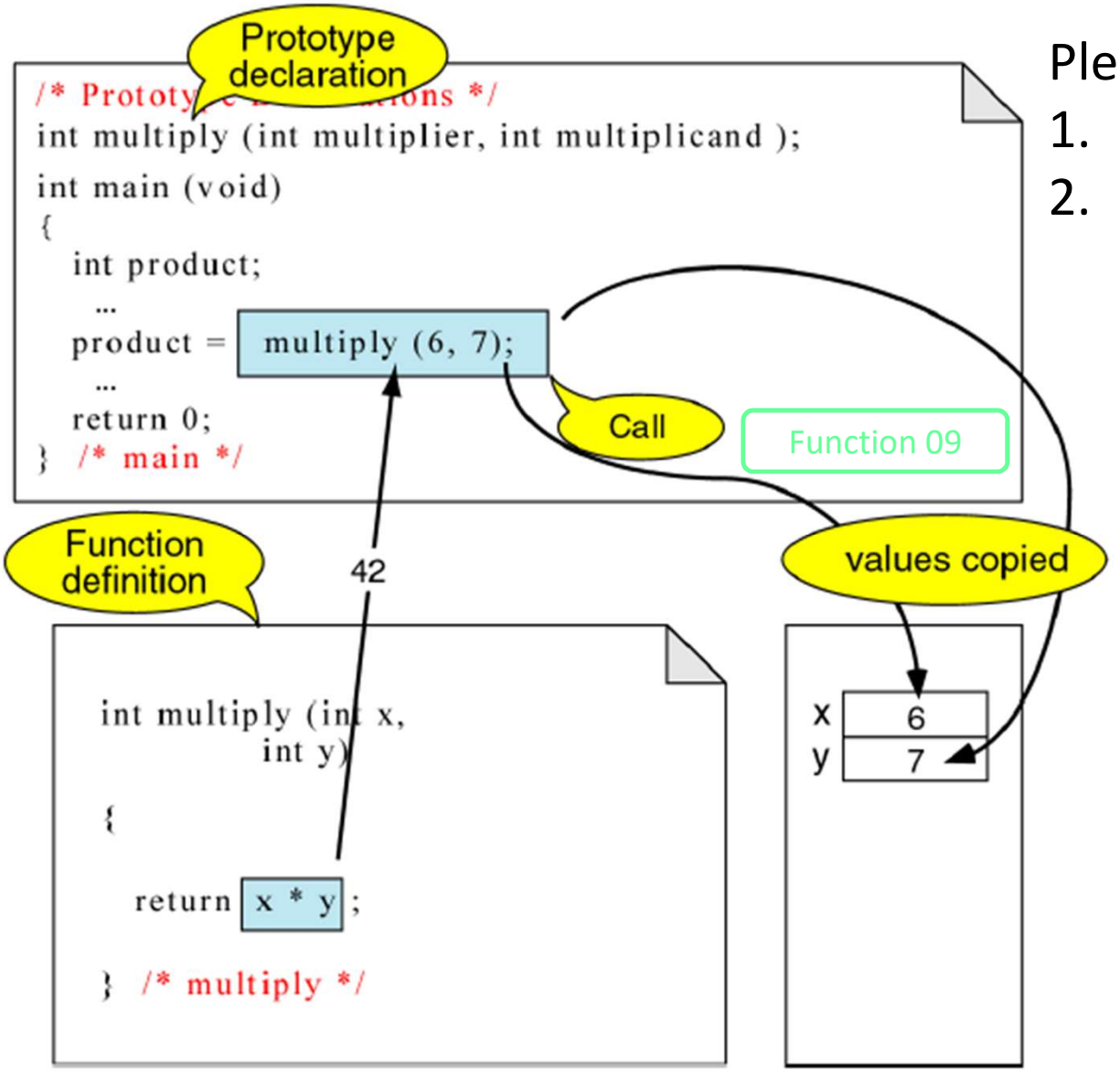
Call by Value

- Every argument to a function is an expression, which has a value.
- C passes an argument to an called function by making a copy of the expression value, storing it in a temporary cell.
- Only the copy of the value is passed to the function argument.
- The original data in the calling function are safe and unchanged.
- As only the copy of the values are passed to the function, this method is called call by values.

- The biggest advantages of using call by value technique to pass arguments to the called function is that arguments can be variables (e.g. x), literals (e.g. 6), or expressions (e.g. X+1).
- The disadvantage is that copying data consumes additional storage space.
- It can take a lot of time to copy, thereby resulting in performance penalty, especially if the function is called many times.


/* Prototype Declaration */
void printOne (int x);
int main (void)
{
/* Local Definition
int a = 5;
/* Statements */
printOne (a);
return 0;
} /* mai
Declaration
Function 07
Call
a
5
Value Copied
void printOne (int x)
{
printf("%d\n", x);
return ;
} /* print ne */
Nothing is returned
to the calling function
5
x
Side Effect
5


/* Prototype Declarations */
int sqr (int x);
int main (void)
{
int a;
int b;
scanf("%d", &a);
b = sqr (a);
printf("%d squared: %d\n", a, b) ;
return 0;
} /* main */
Call
Function 08
int sqr (int x)
{
/* Statements */
return (x * x);
} /* sqr */
Returned value here
a
b
copied
x



Please try the given programs:

1. Find square of a number
2. Modify a number

Call by reference

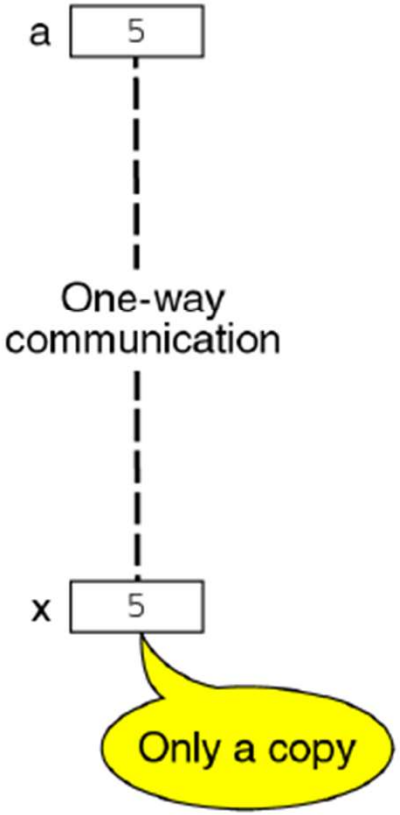
- It links the variable identifiers in the calling function to their corresponding parameters in the called function.
- When the called function changes a value in a variable, then it actually changes the variables in the calling function. This is done by passing an address to the called function.
- 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.
- If 'addr' is a variable that contains an address then *addr means the value pointed by the address stored in the variable 'addr'. Here 'addr' is a pointer variable.

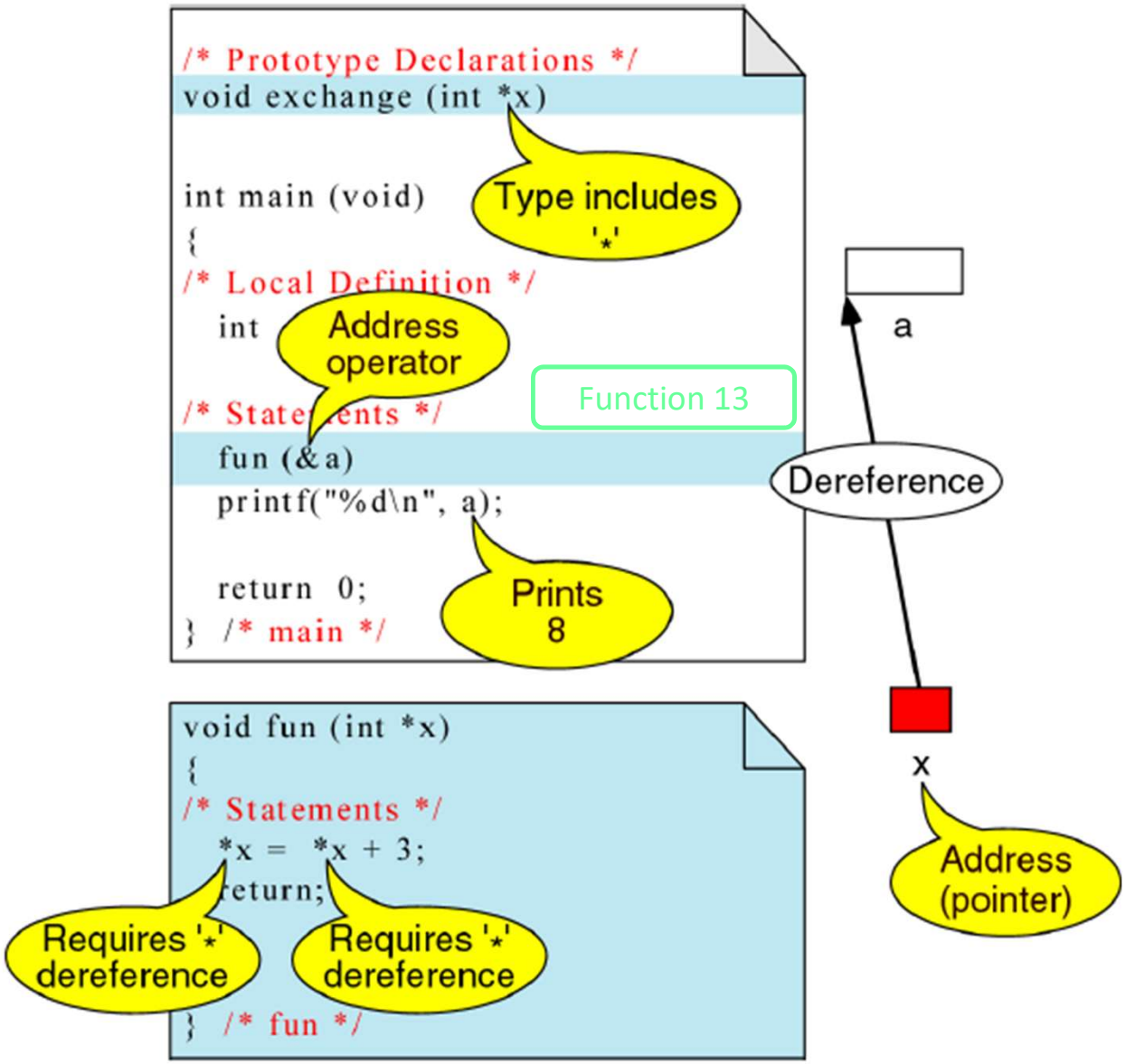
```c
/* Prototype Declarations */
void fun (int num1);

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

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

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

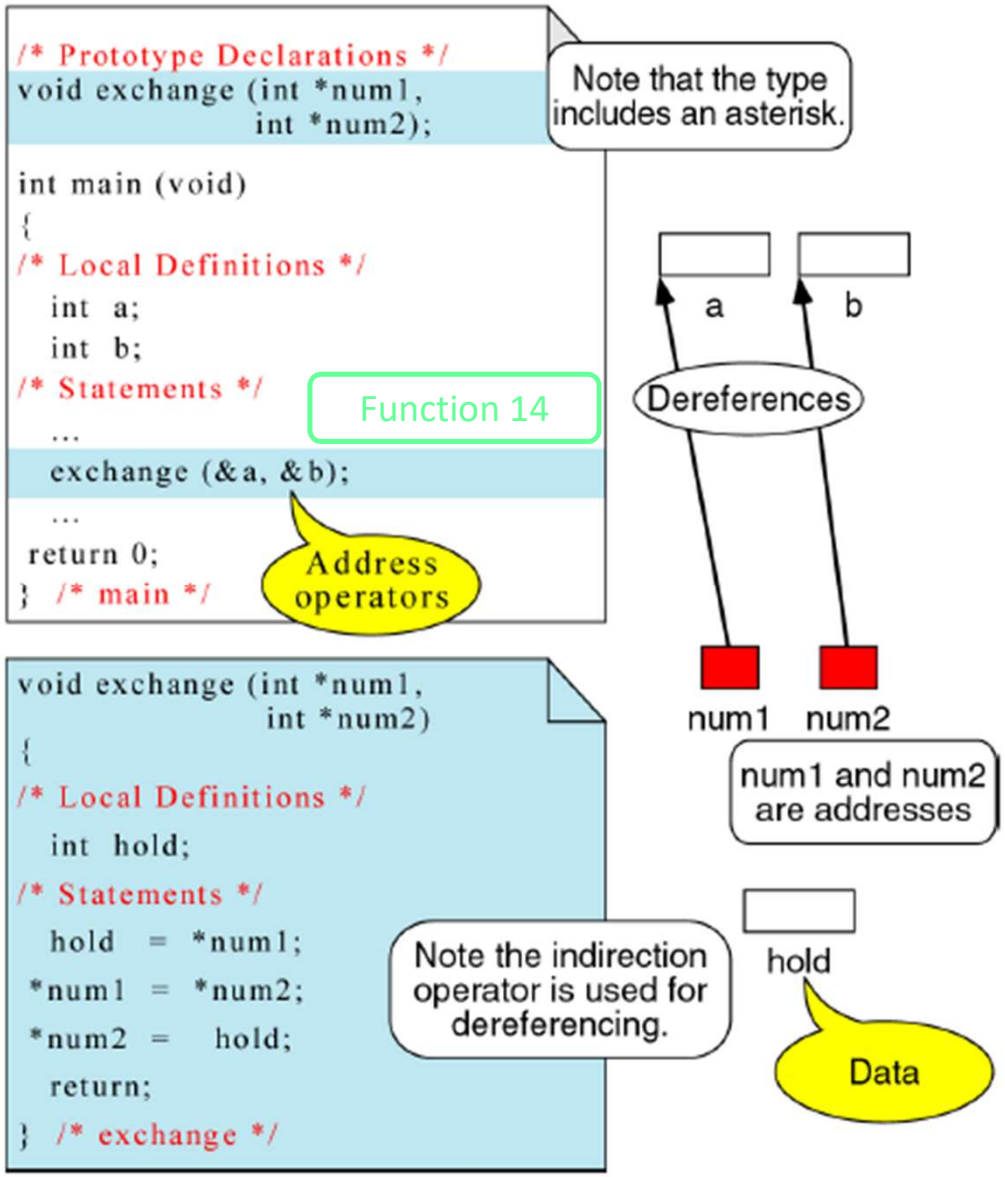
prints 5

Function 12

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



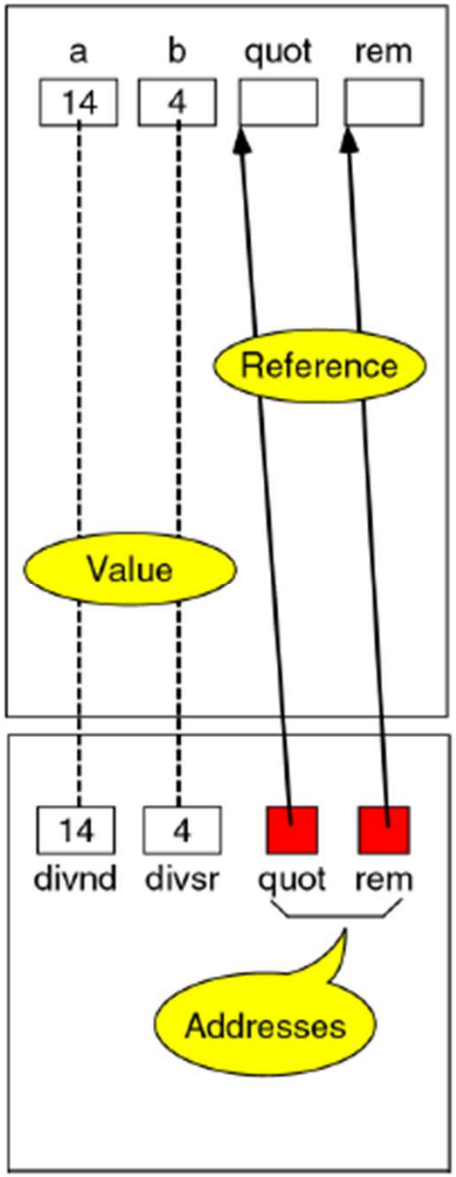

/* Prototype Declarations */
void exchange (int *x)
int main (void)
{
/* Local Definition */
int
/* Statements */
fun (&a)
printf("%d\n", a);
return 0;
} /* main */
Type includes '*'
Address operator
Function 13
Prints 8
a
Dereference
x
Address (pointer)
void fun (int *x)
{
/* Statements */
*x = *x + 3;
return;
} /* fun */
Requires '*' dereference
Requires '*' dereference


/* Prototype Declarations */
void exchange (int *num1,
int *num2);
int main (void)
{
/* Local Definitions */
int a;
int b;
/* Statements */
...
exchange (&a, &b);
...
return 0;
} /* main */
Note that the type includes an asterisk.
Function 14
Address operators
a
b
Dereferences
num1
num2
num1 and num2 are addresses
void exchange (int *num1,
int *num2)
{
/* Local Definitions */
int hold;
/* Statements */
hold = *num1;
*num1 = *num2;
*num2 = hold;
return;
} /* exchange */
Note the indirection operator is used for dereferencing.
hold
Data

```c
/* Prototype Declarations */
void divide (int  divnd, int  divsr,
             int *quot,  int *rem);
int  main (void)
  {
/* Local Definitions */
  int  a;
  int  b;
  int  quot;
  int  rem
/* Statements */
    ...
  divide (a, b, &quot, &rem);
    ...
   return 0;
 } /* main */
```

Function 15

```c
void  divide (int  divnd,
              int  divsr,
              int *quot,
              int *rem)
{
/* Statements */
  *quot =  divnd / divsr;
  *rem  =  divnd % divsr;
   return  ;
}  /* divide */
```



Please try the given programs:

1. Call a number by reference
2. Find biggest of three integers

Review of today lecture – part 2

- A change of state in the program takes place due to the action of a function, which is termed as a side effect of a function.
  - accepting data from outside the program.
  - sending data out of the program to a file or to the monitor.
  - changing the value of a variable in the calling function.
- Call by value in which values of the variables are passed by the calling function to the called function.
- C passes an argument to an called function by making a copy of the expression value, storing it in a temporary cell.

Review of today lecture – part 2

- Call by reference in which address of the variables are passed by the calling function to the called function.
- 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.

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

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

```c
/*  Find the square of a number  */
#include <stdio.h>

int square(int num);

int main(void)
{
   int s1,result;
   scanf("%d",&s1);
   result = square(s1);
   printf("The square of %d is %d\n",s1,result);
   return 0;
}

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

Function 10

Is this a call by value or call by reference?

Please try to identify the input and output of function in the given program:

1. input
2. output

If user key in

a) 5
b) 6.0
c) 4/2
d) number

```c
/*  Modify a number  */
#include <stdio.h>

void modify(int x);

int main(void)
{
   int x=5;
   printf("value of x before calling modify is %d\n",x);
   modify(x);
   printf("value of x after calling modify is %d\n",x);
   return 0;
}

void modify(int x)
{
   printf("value of local variable x in modify is %d\n",x);
   x=10;
   printf("value of x after reassigning local variable x in modify is %d\n",x);
   return;
}
```

Result:

value of x before calling modify is 5
value of local variable x in modify is 5
value of x after reassigning local variable x in modify is 10;
value of x after calling modify is 5

```c
/*  Call a number by reference  */
#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);
}
```

Function 16

Is this a call by value or call by reference?

Please try to identify the input and output of function in the given program:

1. input
2. output

If number is update as below, what is out of program.

a) 6.0
b) 8/2
c) number

Result:

The value of num before calling the function = 2
The value of num in the calling the function = 12
The value of num after calling the function = 12

```c
/*  Find biggest of three integers  */
#include <stdio.h>
int greater(int a, int b, int c);

int main()
{
   int num1, num2, num3, large;
   printf("\n Enter the first number: ");
   scanf("%d", &num1);
   printf("\n Enter the second number: ");
   scanf("%d", &num2);
   printf("\n Enter the third number: ");
   scanf("%d", &num3);

   large=greater(num1,num2,num3);
   printf("\n Largest number = %d",large);
   return 0;
}

int greater(int a, int b, int c)
{
   if(a>b && a>c)
     return a;
   if(b>a && b>c)
     return b;
   else
     return c;
}
```


ARE
YOU
OKAY?

Review of today 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);
}
```

Review of today 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

```c
#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;
}
```

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