FUNCTIONS

QUICK EXERCISE

```
#include<stdio.h>
int add(int a, int b)
{
int c,d;
c=a+b;
d=a+b+1;
return c;
return d;
}
int main()
{
printf("%d\n", add(5,4));
printf("%d", add(8,6));
}
```

Output:

9

14

RETURN KEYWORD

- The return keyword will take the control back to the line of call.
- A function can only return ONE value through the return keyword.
- A function can have multiple return statements, but any ONE return statement will be executed.
- A function may or may not always return a value.
- If a function does NOT return any value, the return type is said to be “void”
- A simple return statement is allowed without actually returning a value.
- If a explicit return type is not present, by default the return type is said to be “int”.

QUICK EXERCISE

```
#include <stdio.h>
void print(int x)
{
if(x<4)
{
printf("Hello\n");
return;
}
printf("Inside Function\n");
}
int main()
{
print(2);
print(8);
printf("End of main\n");
return 0;
}
```

Output:
Hello
Inside Function
End of main

Points to note:

1. A function may not have a return statement at all. In such a case, when a function is over, control automatically goes to the caller.
2. This is because, a program must always begin as well as end with the main() function.

QUICK EXERCISE

```
#include<stdio.h>
void add ( int a, int b)
{
int c;
c=a+b;
printf("%d", c);
}
void main()
{
int x,y,z;
x=10; y=15;
z = add ( x,y );
printf("Sum = %d", z);
}
```

Output:

Error.

Since the return type is void, no value is being returned from the function that can be stored in z.

QUICK EXERCISE

```
#include<stdio.h>
void add ( int a, int b)
{
int c;
c=a+b;
printf("%d", c);
}
void main()
{
int x,y,z;
z=add ( 10,15 ) +add ( 4,6 );
printf("Sum = %d", z);
}
```

Output:

Error.

Since the return type is void, no value is being returned from the function that can be added.

QUICK EXERCISE

```
#include<stdio.h>
void add ( int a, int b)
{
int c;
c=a+b;
printf("%d", c);
}
void main()
{
int x,y,z;
z=add ( add (12, 3), 8 );
printf("Sum = %d", z);
}
```

Output:

Error.

Since the return type is void, no value is being returned from the function that can be passed as parameter to the outer call.

QUICK EXERCISE

```
#include<stdio.h>
void add ( int a, int b)
{
int c;
c=a+b;
printf("%d", c);
}
void main()
{
add(3,4);
}
```

Output:
Sum = 7

This is the ONLY correct way to call a function that returns no value.

QUICK EXERCISE

Given the following function, write the main() function with the function call for the same.

(Note: This example is for understanding purposes ONLY. Follow parameter passing and returning values for real programs)

```
void increment ()
{
int x;
scanf("%d", &x);
x=x+1;
printf("%d", x);
}
```

Solution:

```
int main()
{
increment();
}
```

The function has no parameters and returns no values.

WRITE A FUNCTION THAT SWAPS TWO INTEGER VALUES. PRINT THE VALUES BEFORE THE SWAP AND AFTER THE SWAP.

```
#include <stdio.h>
void swap (int , int );
int main(void)
{
int a=10, b=20;
printf("Before swap a=%d b=%d\n",a,b);
swap (a,b);
printf("After swap a=%d b=%d\n",a,b);
return 0;
}
void swap(int x,int y)
{
int z;
z=x;
x=y;
y=z;
}
```

Output:

Before swap a=10 b=20

After swap a=10 b=20

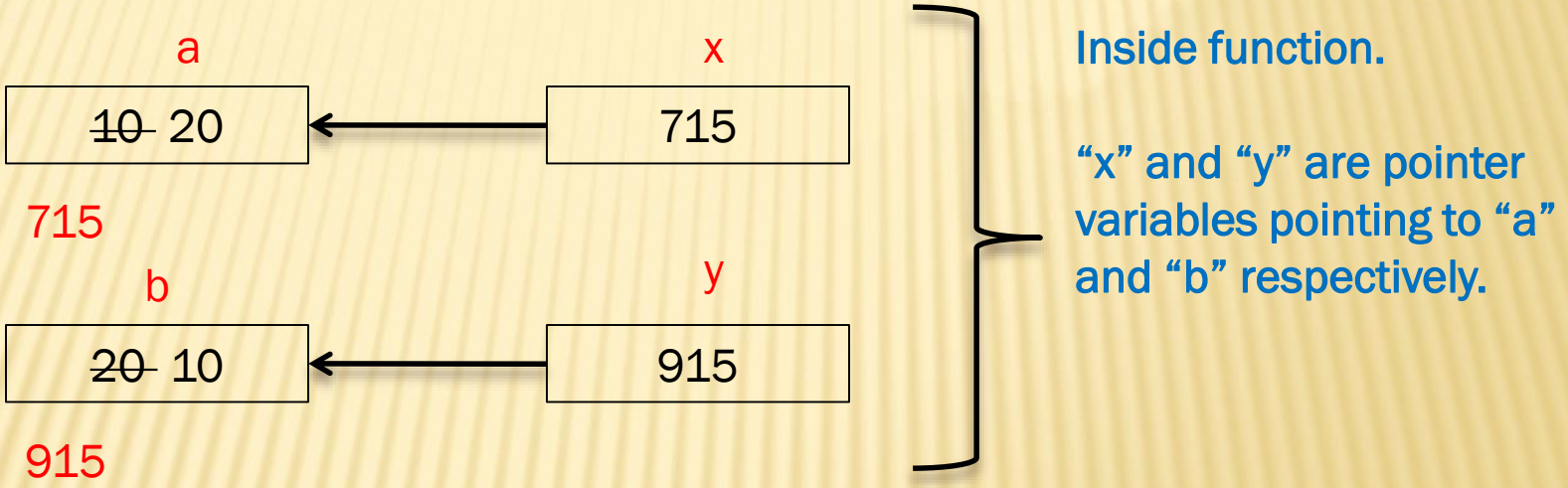
WHY THE PREVIOUS PROGRAM DOES NOT WORK…

- When we pass the values of “a” and “b” to the function, it gets copied inside “x” and “y”.
- So consider the following memory representation,

- When we swap “x” and “y” inside the function, there is no effect on “a” and “b”.
- The variables “a” and “x”, “b” and “y” have different addresses .
- This method of passing parameters is called as “PASS BY VALUE”.
- Any changes made to the formal parameters in the function will not be reflected on the actual parameters

PASS/CALL BY ADDRESS

- Instead of passing “values”, what if we can pass addresses to the function?



- If we pass “addresses” of a and b to the function, then through the pointer variables, we can make any changes to the variables in the main() function.

SWAPPING USING PASS BY ADDRESS

```
#include <stdio.h>
_____ //Function declaration
int main(void)
{
int a=10, b=20;
printf("Before swap a=%d b=%d\n",a,b);
_____ // Function call
printf("After swap a=%d b=%d\n",a,b);
return 0;
}
void swap( _____ )
{
// Three steps of swapping
}
```

SWAPPING USING PASS BY ADDRESS

```
#include <stdio.h>
void swap ( int *, int *) //Function declaration
int main(void)
{
int a=10, b=20;
printf("Before swap a=%d b=%d\n",a,b);
swap(&a, &b);  // Function call
printf("After swap a=%d b=%d\n",a,b);
return 0;
}
void swap( int *x, int *y )
{
int t;
t= *x;
*x=*y;
*y=t;
}
```

Output:

Before swap a=10 b=20

After swap a=20 b=10

DIFFERENCE BETWEEN PASS BY ADDRESS AND PASS BY VALUE

PASS BY VALUE	PASS BY ADDRESS
The **values** of the actual parameters are passed as parameters to the function definition.	In this case, **addresses** are passed as parameters to the function definition.
The formal parameters are simply **copies** of the actual arguments.	The formal parameters are **pointer** variables, pointing to the addresses specified in the function call.
Therefore, any changes made through the formal arguments, will NOT be reflected on the actual parameters	Any changes made through the formal arguments WILL be reflected at the corresponding addresses in the line of call.
This method can return only ONE value using the return keyword.	This method does not need the return statement. This method is used when the function needs to return multiple values.
Example with output (Refer previous slides)	Example with output (Refer previous slides)

QUICK EXERCISE

```
#include <stdio.h>
void sum (int a, int b, int c)
{
a = b + c;
b = a + c;
c = a + b;
}
int main()
{
int x=3,y=4,z=5;
sum (x,y,z);
printf(" x=%d y=%d z=%d\n",x,y,z);
return 0;
}
```

Output:
x=3 y=4 z=5

QUICK EXERCISE

```
#include <stdio.h>
void sum (int *a, int *b, int *c)
{
*a = *b + *c;
*b = *a + *c;
*c = *a + *b;
}
int main()
{
int x=3,y=4,z=5;
sum (&x, &y,&z);
printf(" x=%d y=%d z=%d\n",x,y,z);
return 0;
}
```

Output:
x=9 y=14 z=23

PROGRAM IT…

- Write a function that calculates and returns the area and circumference of a circle.