FUNCTIONS

QUICK EXERCISE: FIND THE OUTPUT

```
#include <stdio.h>
int sub(int a)
{
return a-4;
}
int add(int a, int b)
{
return (5 + sub (a+b));
}
int main()
{
printf("%d", add(3,6));
return 0;
}
```

Output:

10

Point to note:

- A function returns to the line of call once it finishes executing all it's statements.

GUESS… WHAT IS THIS?

RUSSIAN DOLLS

- At first, you see just one big doll, usually painted wood.
- You can remove the top half of the first doll, and what do you see inside? Another, slightly smaller, Russian doll!

RUSSIAN DOLLS

- You can remove that doll and separate its top and bottom halves.
- And you see yet another, even smaller, doll:

RUSSIAN DOLLS

- And you can keep going.
- Eventually you find the teeniest Russian doll.
- It is just one piece, and so it does not open.

WHY RUSSIAN DOLLS?

- We started with one big Russian doll, and we saw smaller Russian dolls, until we saw one that was so small that it could not contain another.
- Just as one Russian doll can go all the way down to a tiny Russian doll that is too small to contain another..
- we'll see how to design an algorithm to solve a problem by solving a smaller instance of the same problem...
- unless the problem is so small that we can just solve it directly. We call this technique **recursion**.

QUICK EXERCISE: MAKE OBSERVATIONS

```
#include <stdio.h>
int add(int a)
{
int d;
d= add(a);
return d;
}
int main()
{
printf("%d", add(5));
return 0;
}
```

Output:
Infinite loop

The function calls *"itself"*.
It endlessly goes into an infinite loop of function calls.

QUICK EXERCISE: FIND THE OUTPUT

```
#include <stdio.h>
int add(int a)
{
int d;
d= add(a-1);
return d;
}
int main()
{
printf("%d", add(5));
return 0;
}
```

Output:
Infinite loop

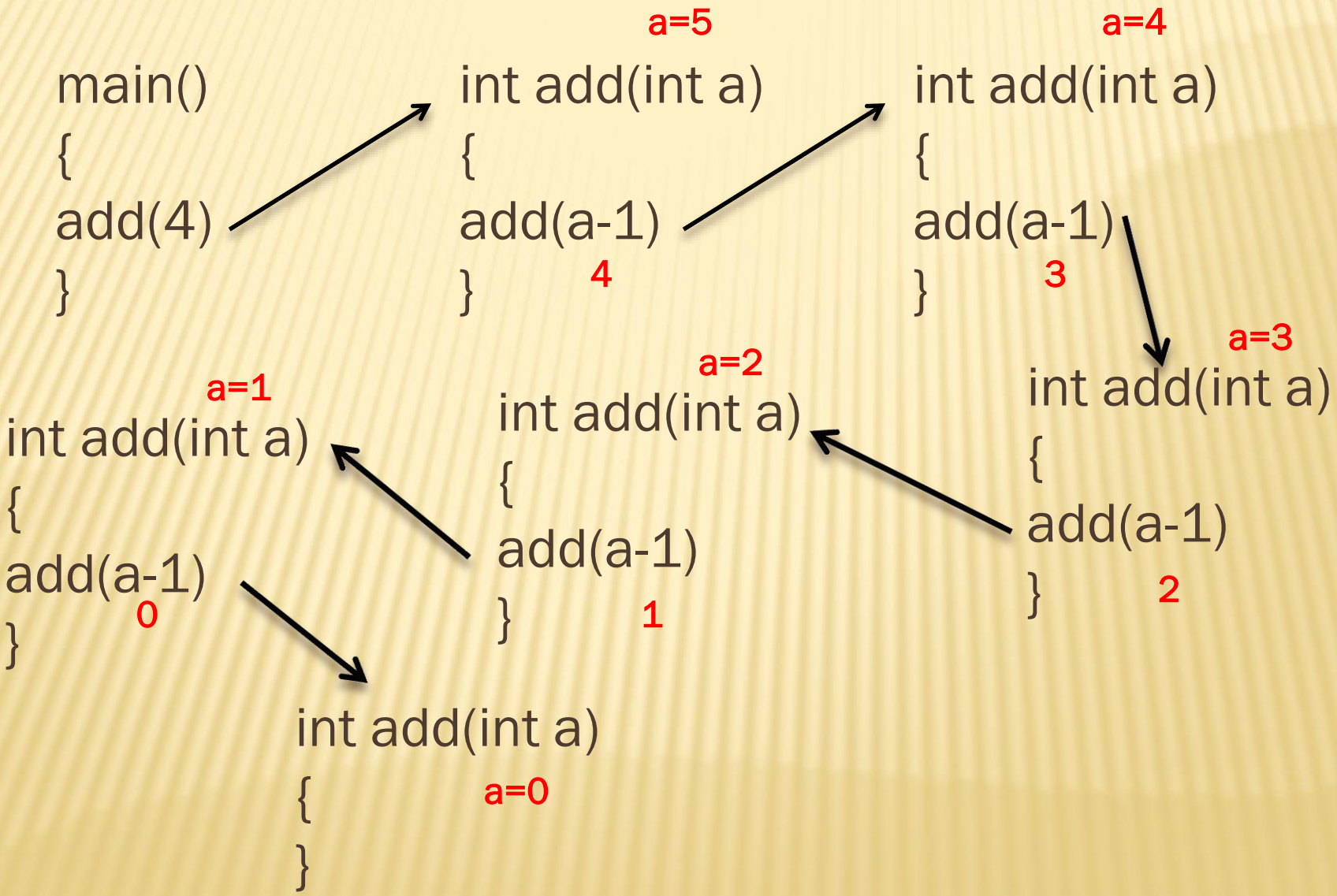
The function calls *"itself"*. It endlessly goes into an infinite loop of function calls.

QUICK EXERCISE: FIND THE OUTPUT

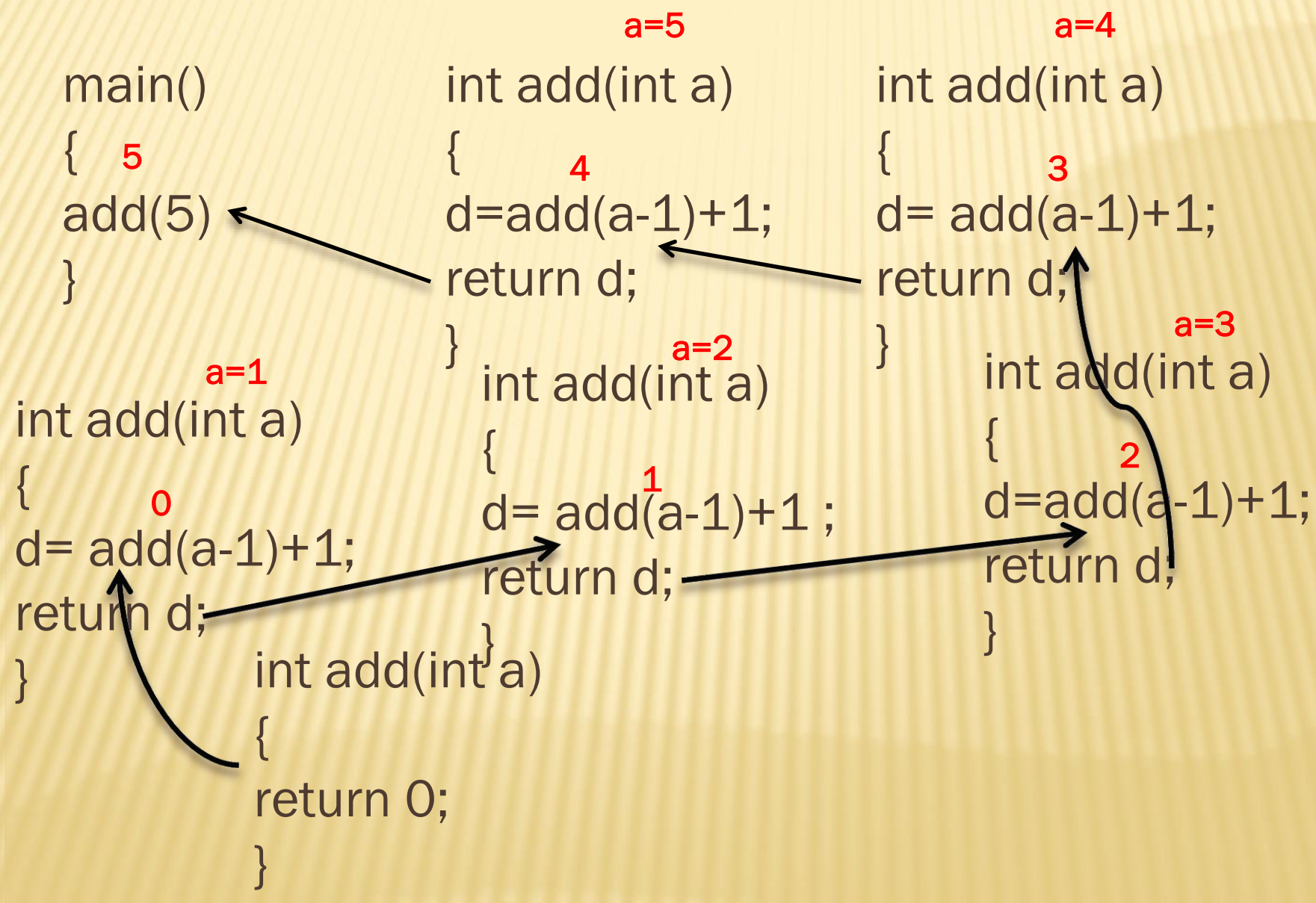
```
#include <stdio.h>
int add(int a)
{
int d;
if(a==0)
return 0;
d = add(a-1)+1;
return d;
}
int main()
{
printf("%d", add(5));
return 0;
}
```

Output:
5

HOW DOES THE FUNCTION CALL WORK?



HOW DOES THE RETURN WORK?

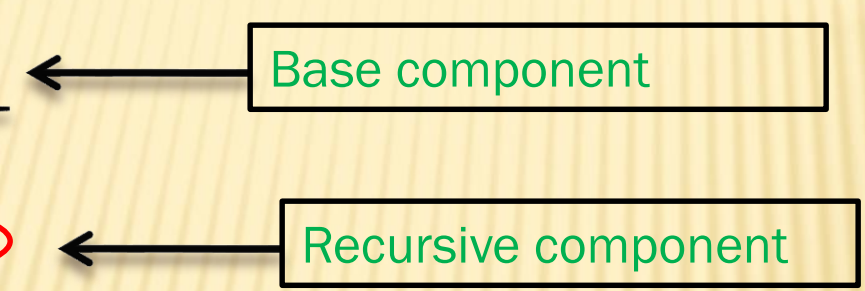


RECURSION

- Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself.
- A recursive function is the one that calls itself, with a more simplified parameter.
- The line of call is called as the *recursive component.*
- The parameters ultimately reach a value where we get a direct solution for the problem.
- This is called as the *base component*.
- The base component typically consists of a
  i. condition
  ii. return statement

RECURSION

```
#include <stdio.h>
int add(int a)
{
int d;
if(a==0)
return 0;
d = add(a-1);
return d;
}
int main()
{
printf("%d", add(5));
return 0;
}
```



QUICK EXERCISE: FIND THE OUTPUT

```
#include <stdio.h>
int add(int a)
{
int d;
if(a==0)
return 0;
d = add(a);
return d;
}
int main()
{
printf("%d", add(5));
return 0;
}
```

Output:

Infinite Loop

The recursive component must modify the parameter in some way.

Else the base component condition will not be reached.

QUICK EXERCISE: FIND THE OUTPUT

```
#include <stdio.h>
int add(int a)
{
int d;
d = add(a-1);
return d;
if(a==0)
return 0;
}
int main()
{
printf("%d", add(5));
return 0;
}
```

Output:

Infinite Loop

The base component must always be placed BEFORE the recursive component.

QUICK EXERCISE: FIND THE OUTPUT

```
#include <stdio.h>
int add(int a, int b)
{
int d;
if(b==0)
return 0;
d = a + add(a,b-1);
return d;
}
int main()
{
printf("%d", add(5,8));
return 0;
}
```

Output:
40
Can you predict what this program is doing?
Adding “a” , “b” times or in other words,
a*b

QUICK EXERCISE

```
#include <stdio.h>
int fun(int n);
int main()
{
printf("%d", fun(1));
return 0;
}
int fun(int n)
{
int d;
if(n==4)
return n;
else
{
d = 2 * fun(n+1);
return d;
}
}
```

Output:
32

QUICK EXERCISE

```
int fun(int n)
{
int d,e;
if(n==0)
return 2;
else
{
d = fun(n/2);
e = fun(n/3);
return (d+e);
}
}
int main()
{
printf("%d", fun(6));
}
```

Output:
14

PROGRAM IT

- Write a recursive function which calculates and returns the factorial of a number.
- Write a recursive function that returns the nth fibonacci number.