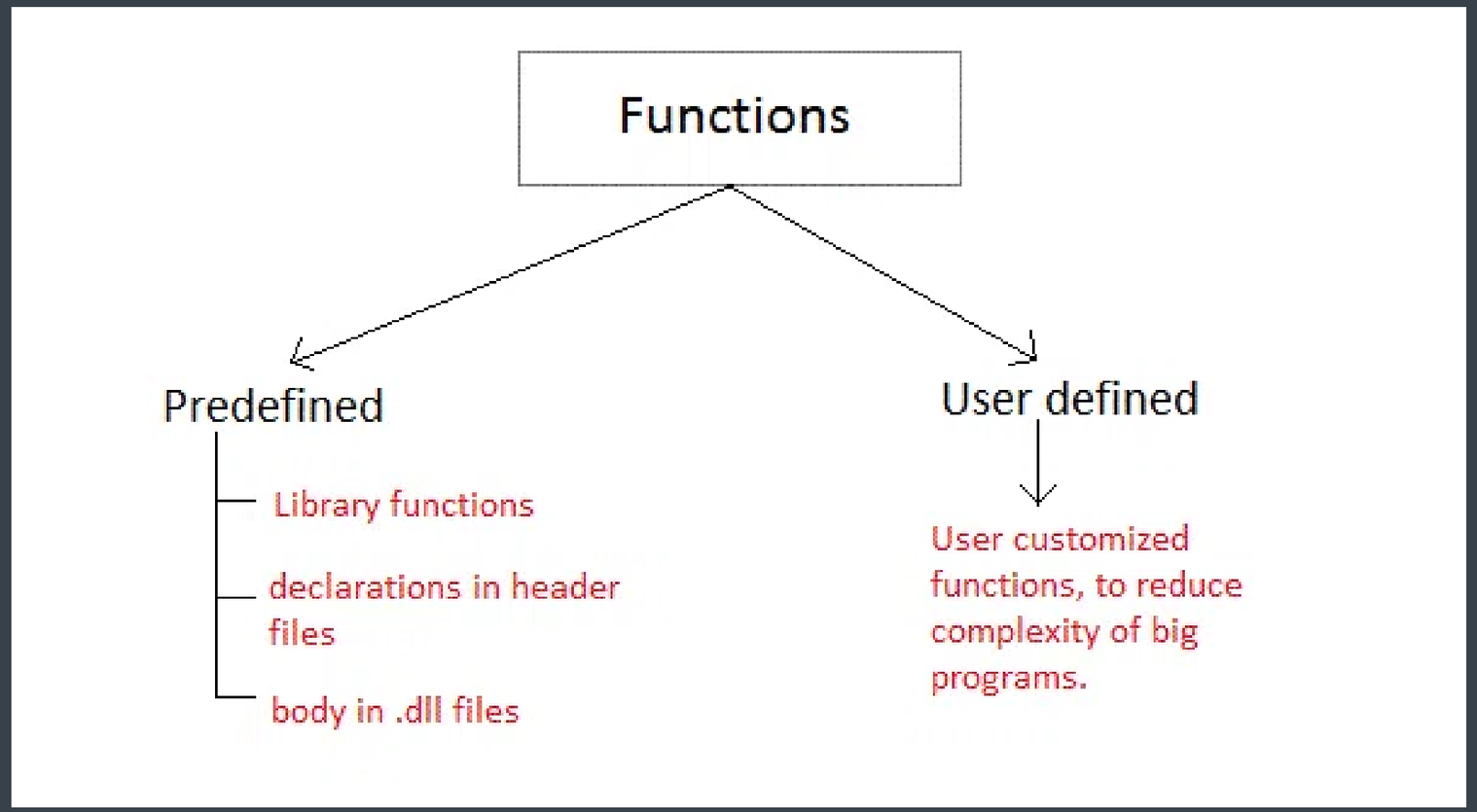
Functions

Function

Functions are blocks of organised, reusable code that perform a specific task

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Types


Functions
Predefined
Library functions
declarations in header files
body in .dll files
User defined
User customized functions, to reduce complexity of big programs.

Declaration/Defining

functions are defined using a specific syntax that specifies the function's name, return type, parameters, and the code to be executed within the function.

Syntax

```
return_type function_name(parameters) {
    // Function body: Code to be executed
    // ...
    // Optional return statement to return a value
}
```

Declaration/Defining

return_type:

- int
- float
- char
- If a function does not return any value, its return type is specified as void.

Example

```
int add(int a, int b) {
    int sum = a + b;
    return sum;
}
```

- int is the return type.
- add is the function name.
- (int a, int b) are the parameters.
- return statement returns the sum.

Function Call

To execute a function and run the code inside it, you call the function by using its name followed by parentheses ().

Syntax

```
return_type result = function_name(arguments);
```

```
int sum = add(5, 3);
```

Parameters and Arguments

Functions can take input values known as parameters or arguments.
Parameters are defined in the function's parentheses during the function definition.
Arguments are the actual values passed to the function when it is called.

Parameters and Arguments

```
int add(int a, int b) {
    int sum = a + b;
    return sum;
}
```

int a and int b are parameters of the add function.

```
int sum = add(5, 3);
```

Parameters and Arguments

```
int sum = add(5, 3);
```

When calling this function, you would provide two integer arguments, such as add(5, 3). These arguments are assigned to the parameters a and b.

Return Statement

Functions can return a value using the return statement. The returned value can be assigned to a variable or used in expressions.

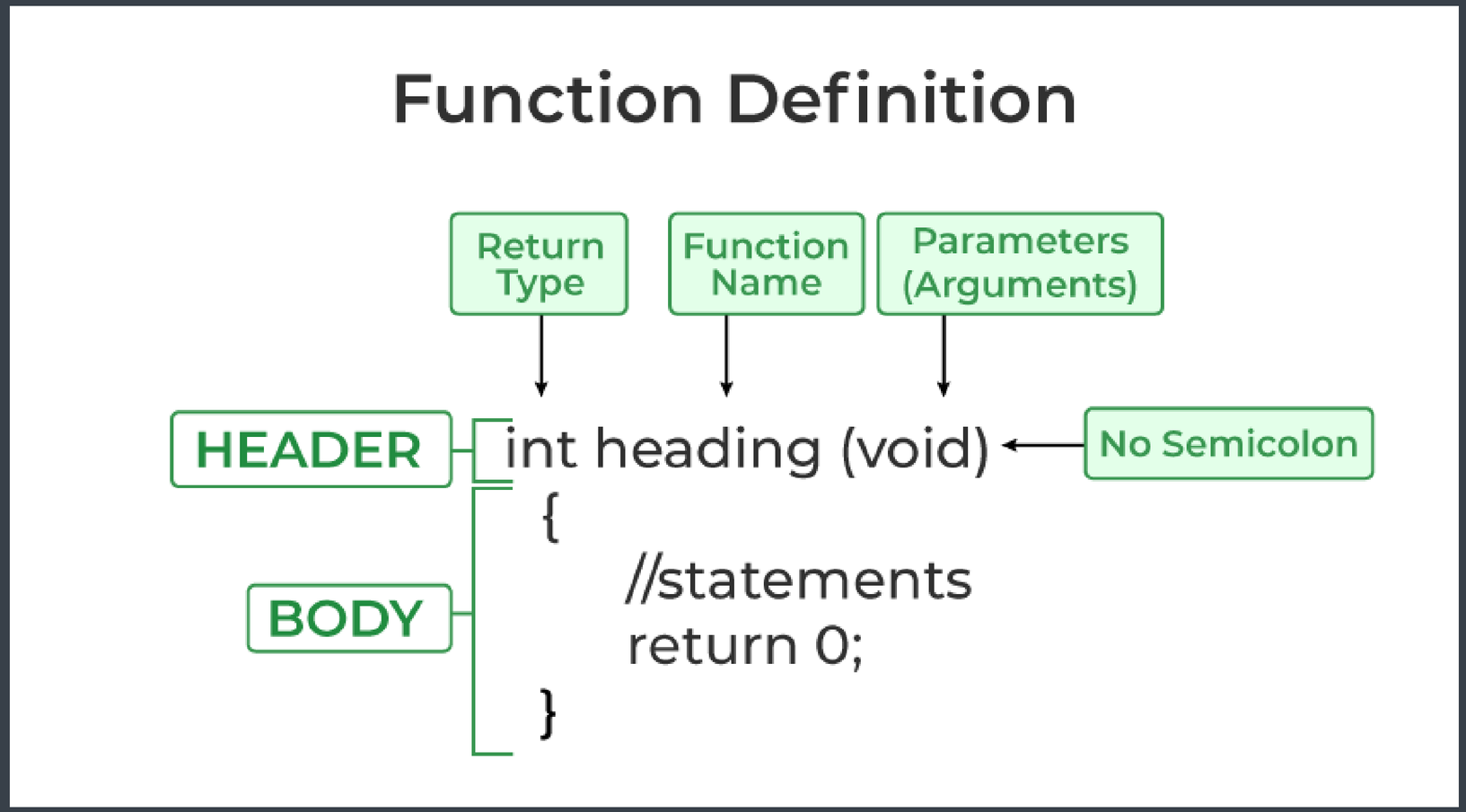
Syntax

return expression;

```
int add(int a, int b) {
    int sum = a + b;
    return sum; // Returns the sum to the caller
}
```

Return Statement: 4 Types

All at One Place



Positional Arguments

- Positional arguments are values passed to a function based on their position or order within the argument list. The position of an argument determines which parameter in the function it corresponds to.
- In C, arguments are matched to parameters based on their orde

```
int result = add(5, 3);

// 5 is the first argument (a), and 3 is the second argument (b)
```

Keyword Arguments

- Keyword arguments are a feature available in some programming languages (not in C). With keyword arguments, you can specify the name of the parameter to which each argument should be assigned.

- This can make function calls more readable and self-explanatory, as the argument names indicate their purpose.

Keyword Arguments

- C does not support keyword arguments natively; arguments are always matched to parameters by position. Keyword arguments are more commonly associated with languages like C.

```
result = add(a=5, b=3)  # Arguments are explicitly named
```

Default Arguments

C does not provide a built-in mechanism for defining default arguments when declaring functions.

```c
int add(int a, int b) {
    return a + b;
}

int add_with_default(int a) {
    return add(a, 0); // Default value of b is 0
}
```

Variable-Length Arguments

In C, variable-length arguments are often implemented using the <stdarg.h> library, which provides a set of macros and functions to work with variable argument lists.

```c
int main() {
    int total = sum(4, 10, 20, 30, 40);
    printf("Total: %d\n", total);
    return 0;
}
```

the sum function accepts a variable number of integer arguments

Scope of Variables

Global Scope

Variables defined outside of any function or block have a global scope. They can be accessed from anywhere in the program, including inside functions.

```
int globalVar = 100; // globalVar is a global variable

void function1() {
    globalVar += 10; // Accessing and modifying the global variable
}

void function2() {
    int localVar = globalVar; // Accessing the global variable
}
```

Local Scope

Variables defined inside a function have a local scope. They are accessible only within the function where they are defined and not from outside the function.

```
void someFunction() {
    int localVar = 42; // localVar is a local variable
    // localVar can be used within this function
}
```

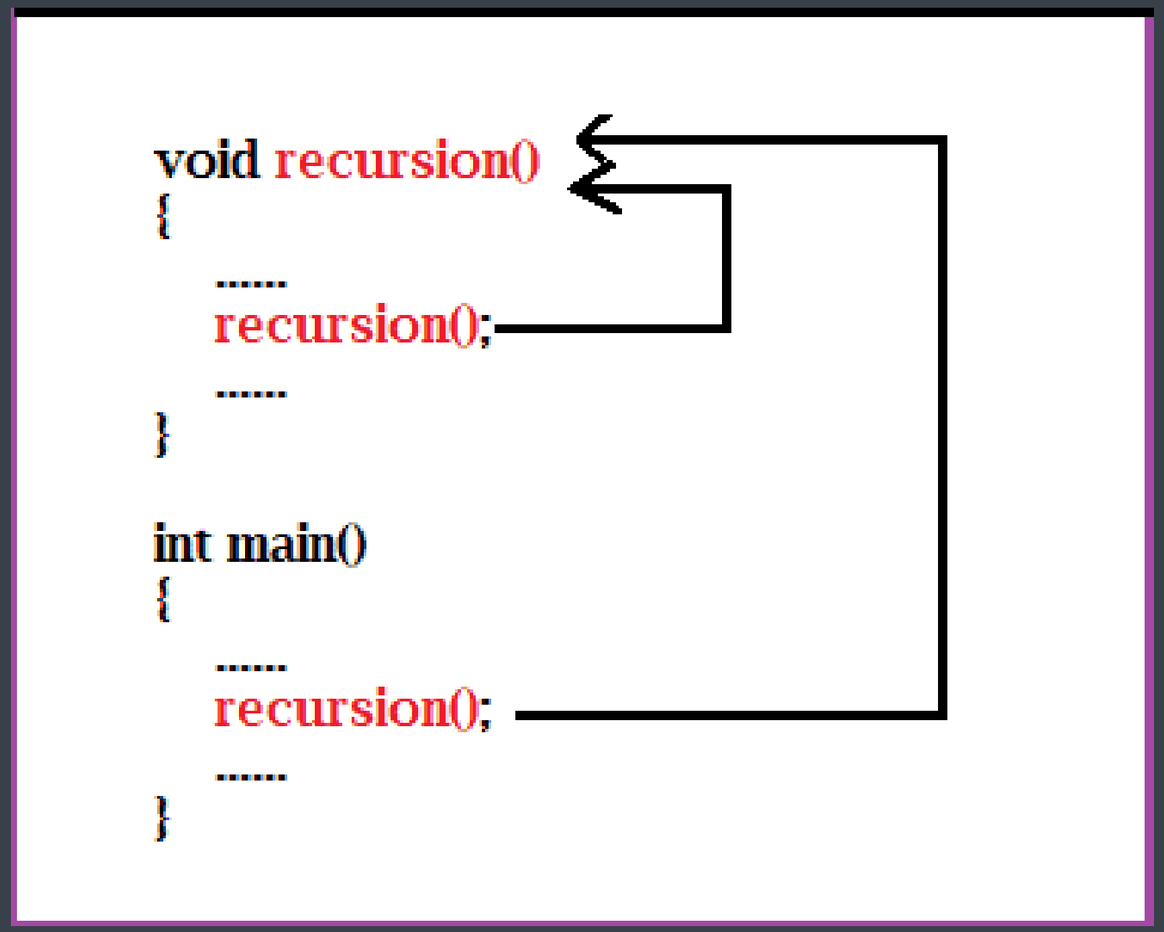
Recursion

Recusive function

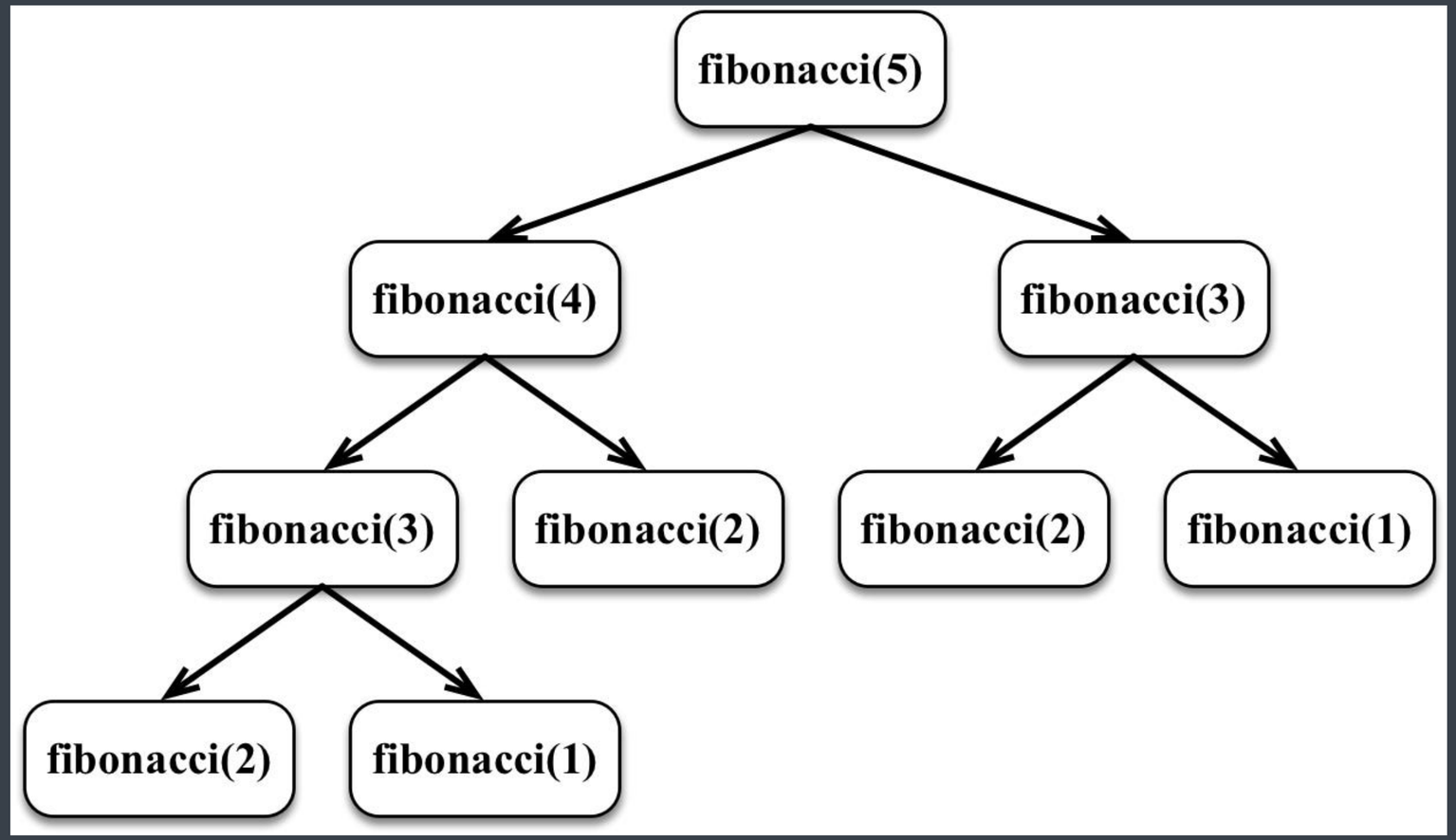
Recursion: A recursive function is a function that calls itself, either directly or indirectly, to solve a problem

Base Case: A base case is a condition that defines when the recursion should stop. It provides the termination point for the recursive calls.

Recusive function



Recusive function for fibonacci



Recusive function for factorial