# Functions

"Fun" but not really

Prepared and presented by Usman Mohammed

## What is a function?

- Functions are a group/ block of code to perform a specific task when called upon.
- Functions can be predefined or created by a user
- Predefined function are library functions
  - Example sqrt(), in the <cmath> library
- User can create their own function to do a specific task.

## Why use a function?

- Reusability

  Easy to call or invoke with out re-writing code

- Abstraction

  Can call it with out knowing the exact workings of the function, just it’s operation and return type

- Modularity

  Function allow the code to be more segmented.

- Visually Looks Better

  Allows for a easier readability which can help with testing and debugging

# Functions – Getting Started

- Here's the basic syntax

```
return_type function_name( parameter list ) {
   //body of the function
}
```

- return_type - data type function returns(e.g., `<int>`, `<double>` or `<void>`)
- function name - name that you choose, used to call the function
- parameter list - A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.(Tutorials Point)
- function Body – The function body contains a collection of statements that define what the function does.

## Functions – Examples

- Predefined Example

```
#include<iostream>
#include<cmath>
int main() {
 double number = 0.0, square_root = 0.0;
 std::cout << "Enter a number ";
 std::cin >> number;
 //sqrt() is a function from the library cmath to calculate
 //the square root of a +ve number. Note how we only know how to
 //use the function, we don't know how it works
 square_root = sqrt(number); std::cout << "Square root of " <<
 number << " = " << square_root;
 return 0;
}
```

## Functions – Examples

- User Defined Example

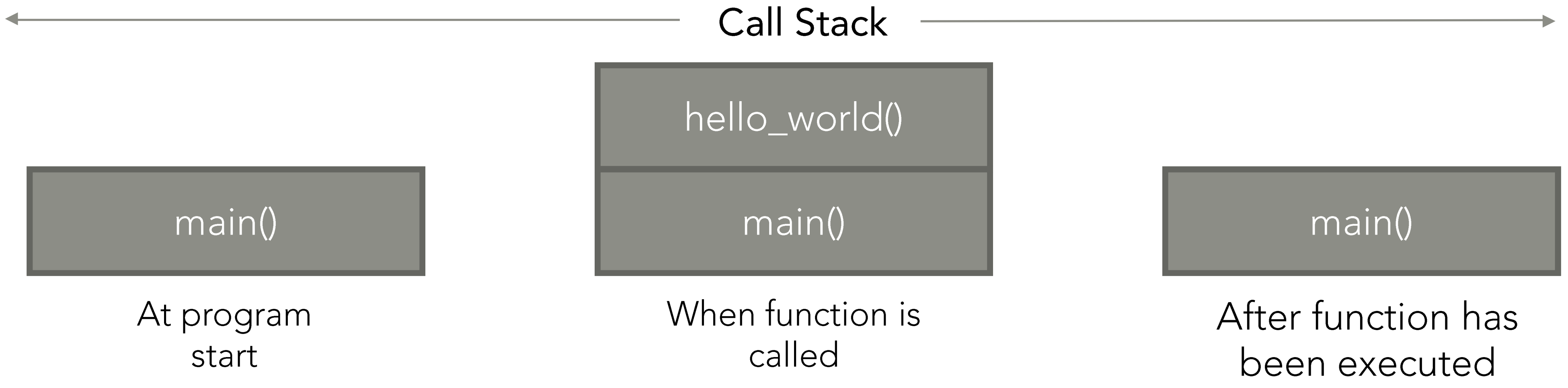
```
#include<iostream>
void hello_world() {
   std::cout << "I got called!" << std::endl;
   // Doesn’t return a value, function type is void
}

int main() {
   hello_world(); //calls the function hello_world
   return 0; //Main returns 0 because it's of type int
   //Final ouput "I got called!"
}
```

## Let’s break that down

- The function hello_world() was declared and defined before the main function
- It was of type void so, no return value
- It had no input parameters
- But how exactly do functions execute? What’s the flow?

## Functions – Call Stack

- When your program starts executing it starts an event queue or stack.
- The first function that gets **pushed** (added) to the stack is main
- Then we called hello_world(), so that function got **pushed** onto the stack
- When it finishes executing, it gets **popped** off and **returns** to the place where it was called



## Functions – Parameters and Arguments

- Information can be passed to functions as a parameter. Parameters act as variables inside the function.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:
- Syntax:

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

## Functions – Examples

- Passing Parameters

```
#include<iostream>
#include<string>
void print_name(std::string fname) {
 std::cout<<fname<<std::endl;
}
int main() {
 print_name("Prakash"); // used
 print_name("Usman");   // multiple
 print_name("Walnut");  // times

 return 0;
}
```

## Functions – Parameters and Arguments

- The arguments that are passed in have to obey the order that you defined in your function

```
#include<iostream>
#include<string>
void print_fullname(std::string fname, std::string lname) {
  std::cout << "My name is: " << fname<<" "<< lname<<std::endl;
}
int main() {
  print_fullname("James", "Bond"); //My name is James Bond
  print_fullname("Bond", "James"); //My name is Bond James
  return 0;
}
```

## Functions – Return Values

- As previously mentioned the return type,  determines the type of data returned by the function.
- Previously, “void” was used, therefore nothing was returned.
- Recall Data Types: int, string, double, float

## Functions – Examples

- Return Value

```
#include<iostream>
int add_two_numbers(int x, int y = 1) {
  //note we can assign default parameters
  return x + y;
}
int main() {
  //store the returned value as variable
  int answer = add_two_numbers(5, 7);
  std::cout << answer << std::endl;

  //use the default parameter
  answer = add_two_numbers(5);
  std::cout << answer << std::endl;
  return 0;
}
```

## Functions – Scope

- By now you've noticed that in order to use variables from `main` in another function we have to pass them in
- To use variables or values from a function they have to be `returned`
- Variables have a scope, they are only accessible within that scope. A scope is defined by a function or class – essentially curly braces

## Functions – Scope

```
#include<iostream>
#include<string>
void print_name(std::string fname) {
  std::string stringInFunction = "";
  std::cout<<fname <<std::endl;
}
int main() {
 print_name("Prakash");
 print_name("Usman");
 print_name("Walnut");
//cannot access ‘stringInFunction’, error
 std::cout << stringInFunction << endl;
 return 0;
}
```

## Recursive Functions

- A function that calls itself
- It is useful for some tasks, such as sorting elements, or calculating the factorial of numbers.
- Let’s look at factorial, the mathematical formula is given below :

  $n! = n * (n-1) * (n-2) * (n-3) \dots * 1$
- Recursive functions need a base or terminating case to avoid an infinite loop or a stack overflow

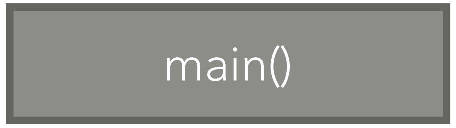
## Recursive Functions – Examples

```
#include<iostream>
long factorial(long num) {
  if (num == 1) { //stop when num becomes 1
   return 1;
  }
  else {
   return num * factorial(num - 1);
  }
}

int main() {
  int number = 9;
  std::cout << number << "! = " << factorial(number) << std::endl;
  return 0;
}
```

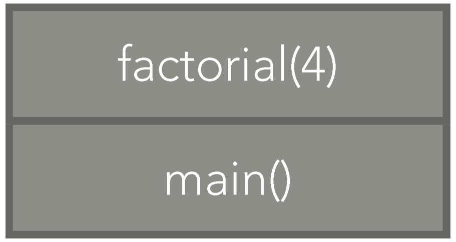
## Let’s break that down

- The result of a number’s factorial can be large, hence the prefix long which increases the number of bytes a variable can occupy
- When we first call `factorial(4)`
  - We pass 4 to the `if` statement, 4 > 1 so the call stack gets `4*factorial(4-1)`= 4* factorial(3)
- So now we call factorial(3)… let’s visualize this on a call stack

# Let’s break that down



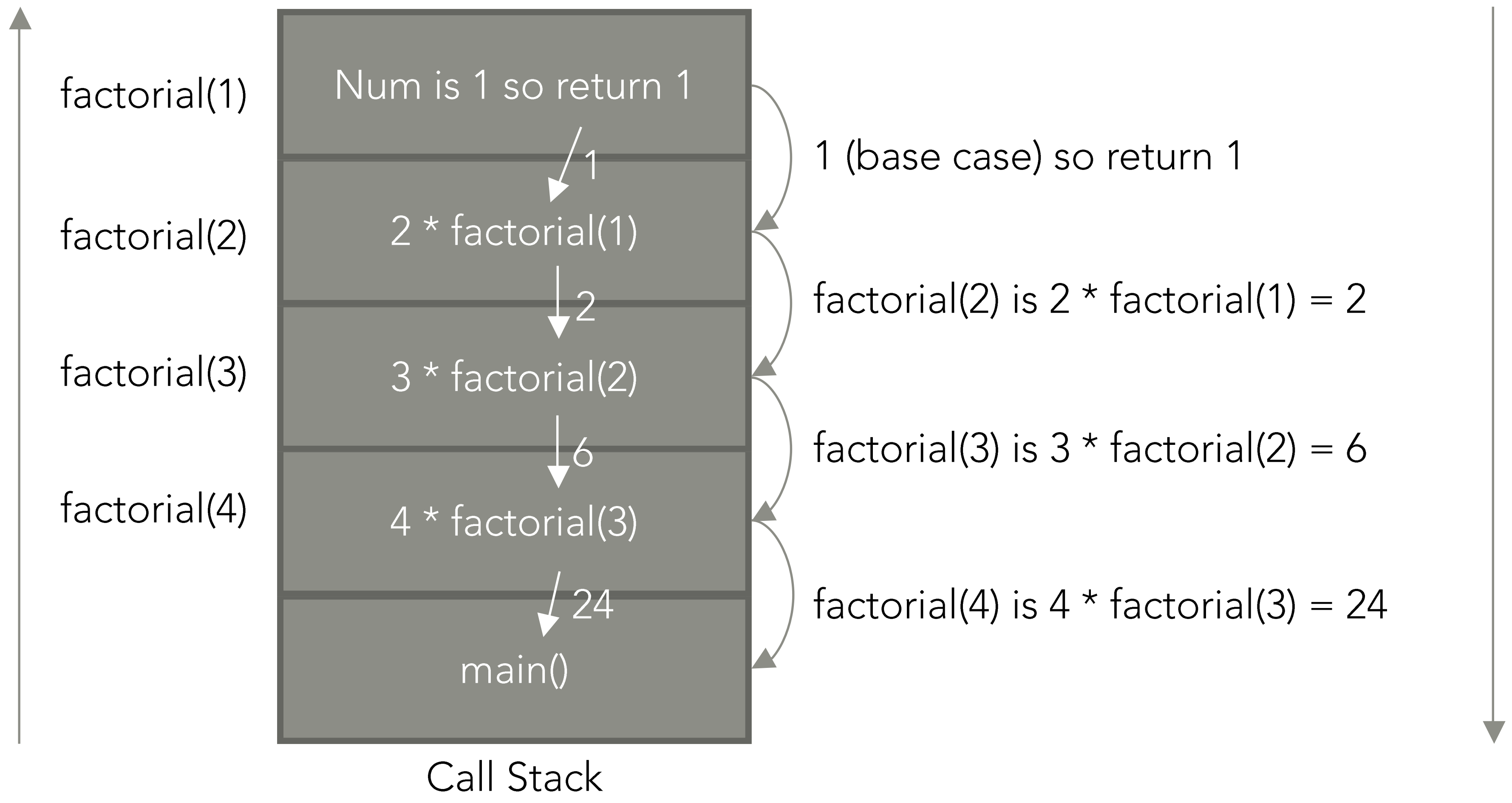
Execution start



Function invoked

# Let’s break that down

Function instances get pushed onto the stack



Until the base case resolves and then the instances get popped off, returning their value to the previous invocation

## Challenge

- Try getting the result of a user defined number from the Fibonacci sequence using recursion. Write your algorithm first!

## Questions?

simeon.ramjit@sta.uwi.edu

github.com/simeon9696/programmingworkshop

## References

- “C Functions.” Tutorialspoint. Accessed January 14, 2020. https://www.tutorialspoint.com/cplusplus/cpp_functions.htm.
- “C Functions Parameters.” C Function Parameters. Accessed January 14, 2020. https://www.w3schools.com/cpp/cpp_function_param.asp.