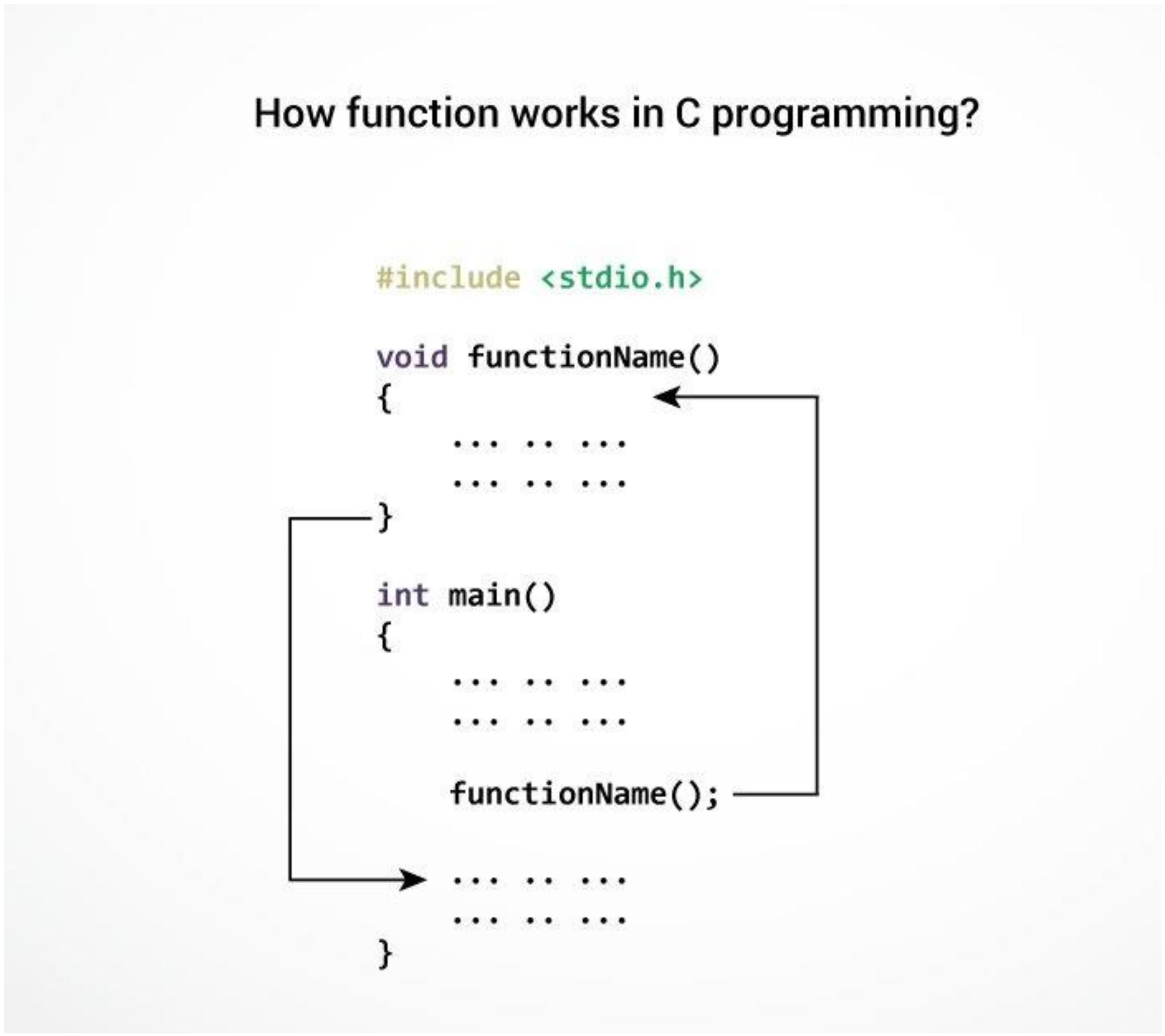
Unit 6

Function

- Introduction
- Function components (function declaration, function call, function definition)
- Types of function (library/built-in function and user-defined function)
- Category of function according to return value and arguments
- Parameter passing in C (call by value and call by reference)
- Recursion (recursive function)
- Passing array to function
- Passing string to function



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

Function

- A function is a block of statements that performs a specific task.
- Every C program has at least one function, which is **main().**
- The **main()** function in C is the entry point for all C programs. It is where the program starts executing.
- We can divide large and complex problems into small chunks using function.
- Suppose, you need to create a program to create a circle and color it. We can create two functions to solve this problem:
    - create a circle function
    - create a color function

Example:

```c
#include <stdio.h>

void introduction()
{
    printf("Hi\n");
    printf("My name is Ramesh.\n");
    printf("How are you?");
}

int main()
{
     /*calling function*/
     introduction();
     return 0;
}
```

Advantages of function:

1. Modularity: Functions help break down code into smaller, self-contained units.
2. Reusability: Functions can be reused in different parts of a program or other projects.
3. Abstraction: Functions provide a simpler interface, hiding complex operations.
4. Code organization: Functions improve code organization and make it more readable.
5. Testing and debugging: Functions enable isolated testing and easier debugging.

Function components

SN	C function components	Syntax

1	Function declaration	returnType functioName (parameters); // it tells the compiler that the function is being declared.

2	Function call	functionName (argumentList)
		
		// runs the function
		// fucntion can be used anywhere

3	Function definition	returnType functionName (argument list) {function body;} // function statements which are executed

Example:

```c
#include <stdio.h>

void introduction(char name[], char address[]); //function declaration

int main()
{
     // function call
    introduction("John david", "Kathmandu");

    return 0;
}

// function definition
void introduction(char name[], char address[])
{
    printf("Hi\n");
    printf("My name is %s.\n", name);
    printf("I am from %s.", address);
}
```

Types of Functions

1. Library/pre-defined Functions: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts() etc.
2. User-defined functions: are the functions which are created by the C programmer, and can be used it many times.

Categories of function

There are 4 categories of functions based on arguments and return value

a. Function without arguments and without return value
b. Function without arguments and with return value
c. Function with arguments and without return value
d. Function with arguments and with return value

a. Function without arguments and without return value

Example

```c
#include <stdio.h>

void printName()
{
    printf("Good Morning");
}

void main()
{
    printf("Hello everyone! ");
    printName();
}
```

b. Function without arguments and with return value

Example:

```c
#include <stdio.h>

int area()
{
    int l = 10, b = 8;
    return l*b;
}

int main()
{
    int result  = area();

    return 0;
}
```

c. Function with arguments and without return value

Example:

```c
#include <stdio.h>

void area(int l, int b)
{
    printf("The area is %d", l * b);
}

int main()
{
    area(10, 8);

    return 0;
}
```

d. Function with arguments and with return value

Example:

```c
#include <stdio.h>

int area(int l, int b)
{
    return l * b;
}

int main()
{
    int result = area(10, 8);
    printf("The area is %d", result);

    return 0;
}
```

Pass by value

- Pass by value is also known as call by value.
- In pass by value, actual value is copied and passed to the function.

- Changes made to the parameter inside the function have no effect on the argument.
- By default, C programming uses *call by value* to pass arguments.

Example: Swapping two numbers using call by value

```
#include <stdio.h>

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
    printf("After swap: a = %d, b = %d\n", a, b);
}

int main()
{
    int a = 10;
    int 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;
}
```

Pass by reference

- Pass by reference is also known as call by reference.
- In pass by reference, memory address of the value is passed to the function rather than actual value.
- Changes made to the parameter inside the function have effect on the argument.

Example: Swapping two numbers using call by reference

```
#include <stdio.h>
void swap(int *, int *);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main \na = %d \nb = %d\n",
a, b);
```

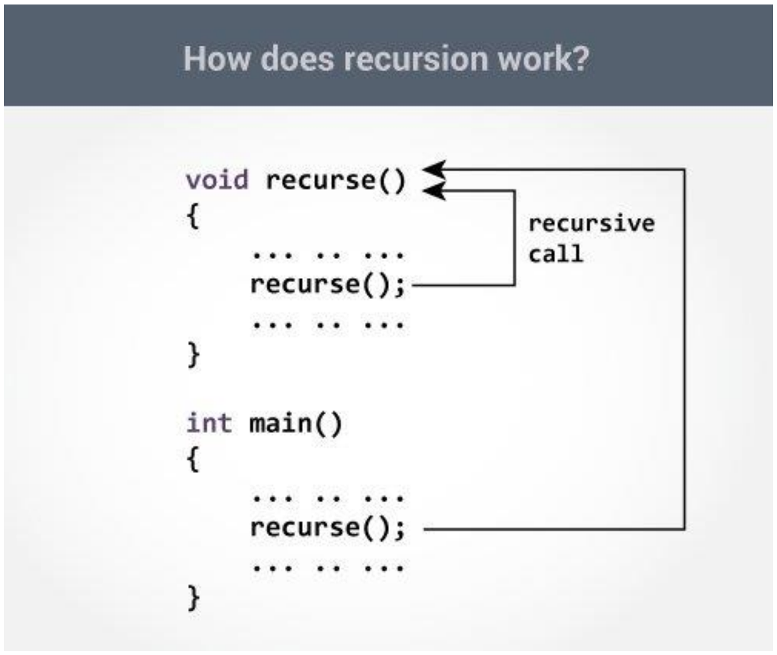
```
    swap(&a, &b);
    printf("After swapping values in main \na = %d \nb = %d\n", a, b);

    return 0;
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    printf("After swapping values in function \na = %d \nb = %d\n",
*a, *b);
}
```

Recursion

- Recursion is a programming concept where a function that calls itself.
- Such function is called recursive function.
- The recursion continues until some condition is met to prevent it.
- In recursion, a function solves a problem by breaking it down into smaller instances of the same problem.



```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Example: Sum of Natural Numbers Using Recursion

```c
#include <stdio.h>
int sum(int n);

int main()
{
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}

int sum(int n)
{
    if (n != 0)
        // sum() function calls itself
        return n + sum(n - 1);
    else
        return n;
}
```

<u>Example 2: Factorial of given number using recursion</u>

```c
#include <stdio.h>

long fact(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}

int main()
{
    int number;
    long factorial;

    printf("Enter a number: ");
    scanf("%d", &number);

    factorial = fact(number);

    printf("The factorial of %d is %ld\n", number, factorial);

    return 0;
}
```

Passing array to function

We can pass an array item or whole array to the function using array name as an function argument.

Example

```c
#include <stdio.h>
void calculateSum(float num[]);

int main()
{
    float num[] = {23.4, 55, 22.6, 3, 40.5, 18};
    calculateSum(num);

    return 0;
}

void calculateSum(float num[])
{
    float sum = 0.0;
    for (int i = 0; i < 6; ++i)
    {
        sum += num[i];
    }
    printf("Result = %.2f", sum);
}
```

Passing string to function

We can pass a string item or whole string to the function using string name as an function argument.

Example:

```c
#include <stdio.h>
void displayString(char str[]);

int main()
{
    char str[50];
    printf("Enter your name: ");
    gets(str);
    displayString(str);
    return 0;
}
void displayString(char str[])
{
    printf("Hello! ");
    puts(str);
}
```