Functions

What is Function?

- A function is a group of statements that perform a specific task.
- It divides a large program into smaller parts.
- A function is something like hiring a person to do a specific job for you.
- Every C program can be thought of as a collection of these functions.
- Program execution in C language starts from the main function.

Syntax

```
void main()
{
    // body part
}
```

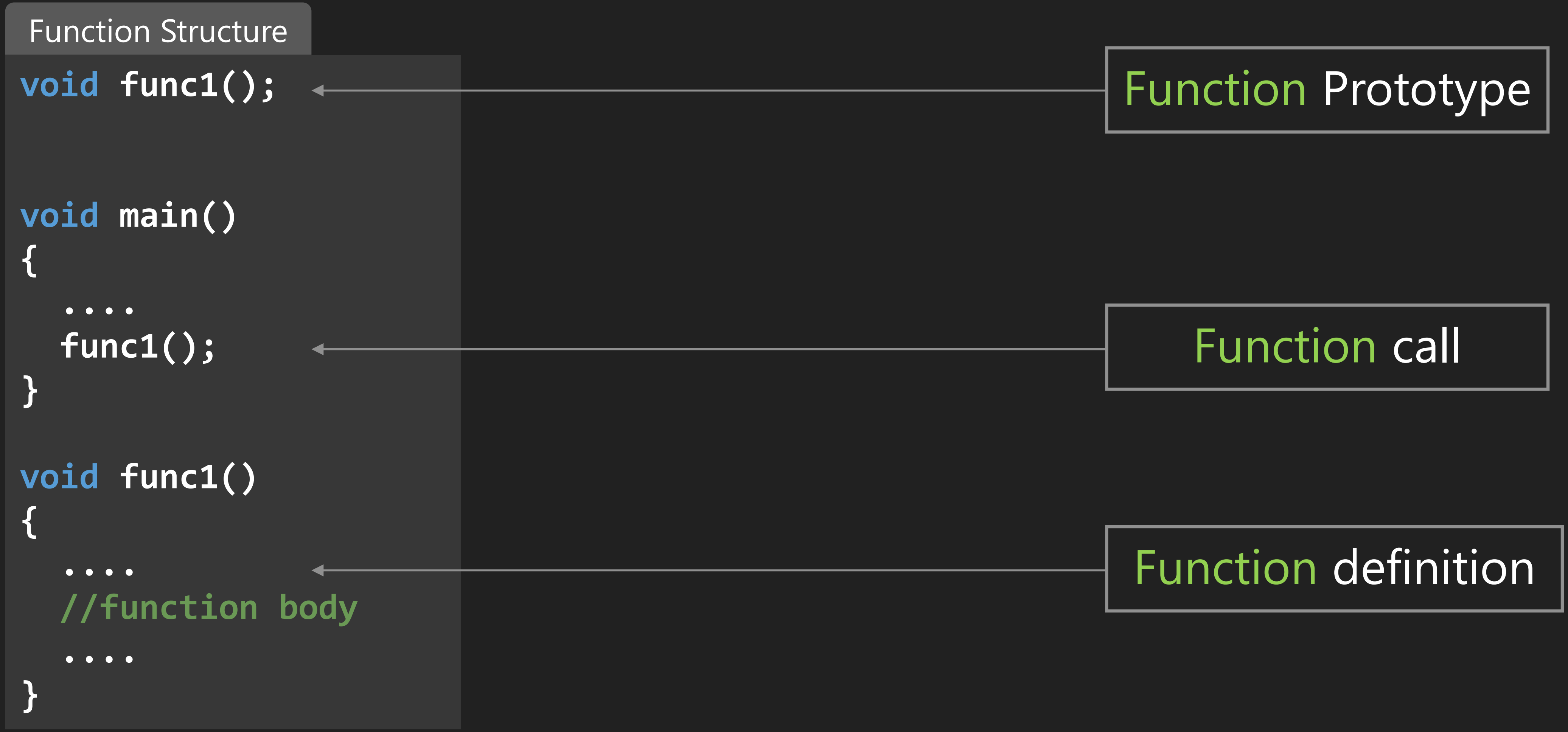
- Why function ?
  - Avoids rewriting the same code over and over.
  - Using functions it becomes easier to write programs and keep track of what they doing.

Types of Function

- Function
  - Library Function
    - Predefined or inbuilt
    - Declarations inside header files
    - Eg. printf() – stdio.h
      pow() – math.h
      strcmp() – string.h
  - User Defined Function (UDF)
    - Created by User
    - Programmer need to declare it
    - Eg. findSimpleInterest()
      areaOfCircle()

Program Structure for Function

- When we use a user-defined function program structure is divided into three parts.



Function Structure

```c
void func1();


void main()
{
   ....
   func1();
}

void func1()
{
   ....
   //function body
   ....
}
```

Function Prototype

- A function Prototype also know as function declaration.
- A function declaration tells the compiler about a function name and how to call the function.
- It defines the function before it is being used or called.
- A function prototype needs to be written at the beginning of the program.

Syntax

```
return-type function-name (arg-1, arg 2, …);
```

Example

```
void addition(int, int);
```

Function Definition

- A function definition defines the functions header and body.
- A function header part should be identical to the function prototype.
  - Function return type
  - Function name
  - List of parameters
- A function body part defines function logic.
  - Function statements

Syntax

```
return-type function-name (arg-1, arg 2, …)
{
        //... Function body
}
```

Example

```
void addition(int x, int y)
{
 printf("Addition is=%d“,(x+y));
}
```

WAP to add two number using add(int, int) Function

Program

```c
#include <stdio.h>
void add(int, int); // function declaration

void main()
{
    int a = 5, b = 6;
    add(a, b); // function call
}

void add(int x, int y) // function definition
{
   printf("Addition is = %d", x + y);
}
```

Output

```
Addition is = 11
```

Actual parameters and Formal parameters

- Values that are passed to the called function from the main function are known as Actual parameters.
- The variables declared in the function prototype or definition are known as Formal parameters.
- When a method is called, the formal parameter is temporarily "bound" to the actual parameter.

Actual parameters

```
void main()
{
    int a = 5, b = 6;
    add(a, b); // a and b are the
    actual parameters in this call.
}
```

Formal parameters

```
void add(int x, int y) // x and y are
formal parameters.
{
   printf("Addition is = %d", x + y);

}
```

Return Statement

- If function is returning a value to calling function, it needs to use the keyword **return**.
- The called function can only return one value per call.

Syntax

```
return;
            Or

return (expression);
```

WAP to find maximum number from two number

Program

```c
#include <stdio.h>
int max(int a, int b);
void main()
{
    int a = 100;
    int b = 200;
    int maxvalue;
    maxvalue = max(a, b);
    printf("Max value is : %d\n",
    maxvalue);
}
int max(int a, int b)
{
    if (a > b)
        return a; // return a
    else
        return b; // return b
}
```

Output

```
Max value is : 200
```

WAP to calculate the Power of a Number

Program

```c
#include <stdio.h>
int power(int, int);
void main()
{
    int num, pow, res;
    printf("Enter any number : ");
    scanf("%d", &num);
    printf("Enter power of number : ");
    scanf("%d", &pow);
    res = power(num, pow);
    printf("%d's power %d = %d", num, pow, res);
}
int power(int n, int p)
{   int r = 1;
    while (p >= 1)
    {
        r = r * n;
        p--;
    }
    return r;}
```

Output

```
Enter any number : 5
Enter power of number : 3
5's power 3 = 125
```

WAP to find Factorial of a Number

Program

```c
#include <stdio.h>
int fact(int);
int main()
{
    int n, f;
    printf("Enter the number :\n");
    scanf("%d", &n);
    f = fact(n);
    printf("factorial = %d", f);
}
int fact(int n)
{
    int i, fact = 1;
    for (i = 1; i <= n; i++)
        fact = fact * i;
    return fact;
}
```

Output

```
Enter the number :
5
factorial = 120
```

WAP to check Number is Prime or not

Program

```c
#include <stdio.h>
int checkPrime(int);
void main()
{
    int n1, prime;
    printf("Enter the number :");
    scanf("%d", &n1);
    prime = checkPrime(n1);
    if (prime == 1)
        printf("The number %d is a prime
        number.\n", n1);
    else
        printf("The number %d is not a prime
        number.\n", n1);
}
```

Program contd.

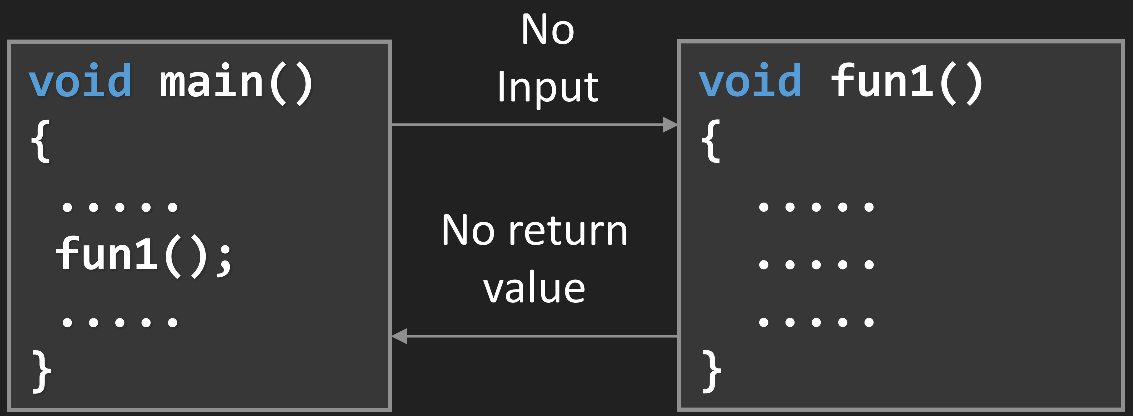
```c
int checkPrime(int n1)
{
    int i = 2;
    while (i <= n1 / 2)
    {
        if (n1 % i == 0)
                return 0;
        else
                i++;
    }
    return 1;
}
```

Output

```
Enter the number :7
The number 7 is a prime number.
```

Category of Function

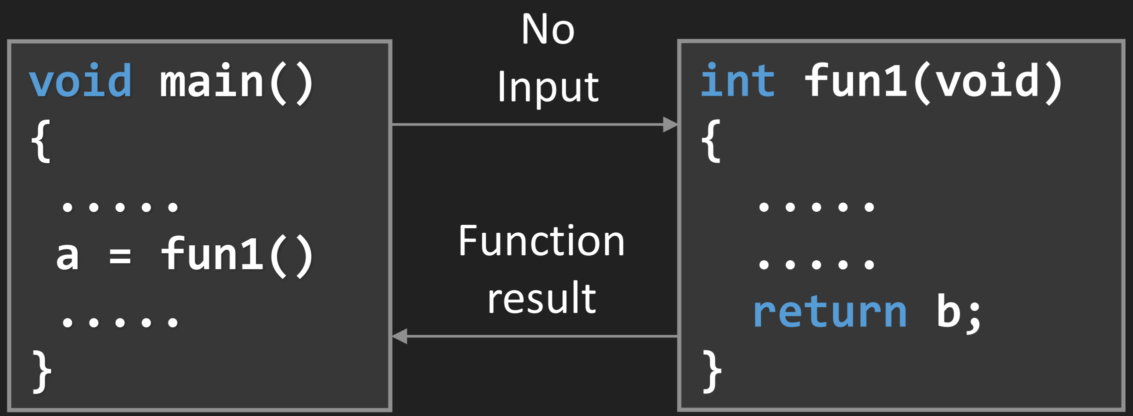
(1) Function with no argument and but no return value



```
void main()
{
 .....
 fun1();
 .....
}
```

```
void fun1()
{
  .....
  .....
  .....
}
```

(2) Function with no argument and returns value

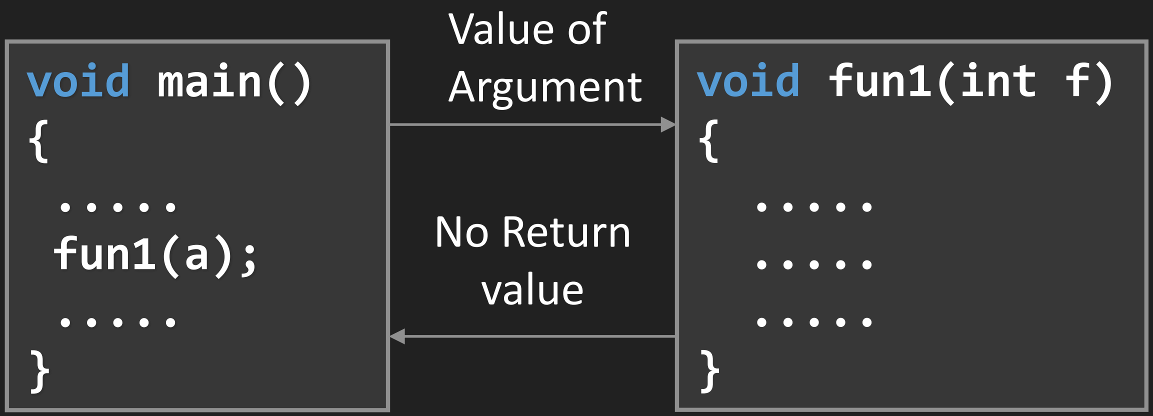


```
void main()
{
 .....
 a = fun1()
 .....
}
```

```
int fun1(void)
{
  .....
  .....
  return b;
}
```

Category of Function cont.

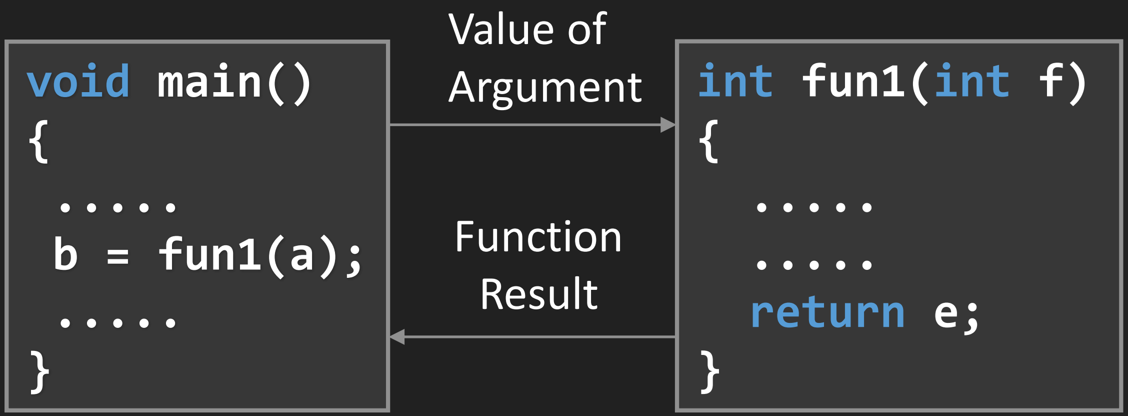
(3) Function with argument and but no return value



```
void main()
{
 .....
 fun1(a);
 .....
}
```

```
void fun1(int f)
{
  .....
  .....
  .....
}
```

(4) Function with argument and returns value



```
void main()
{
 .....
 b = fun1(a);
 .....
}
```

```
int fun1(int f)
{
  .....
  .....
  return e;
}
```

Storage Classes

- Storage class decides the scope, lifetime and memory allocation of variable.
- Scope of a variable is the boundary within which a variable can be used.

Storage Specifier	Storage	Initial Value	Scope	Life	Example
Automatic {auto}	Stack	Garbage	Within block	End of block	`int a;` `auto int a;`
Register {register}	CPU register	Garbage	Within block	End of block	`register int var;`
External {extern}	Data segment	Zero	Global Multiple file	Till end of program	`extern int var;`
Static {static}	Data segment	Zero	Within block	Till end of program	`static extern int var;` `static int var;`

Static Example

Program

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

void main()
{
    printf("Counter = %d \n", incrementCounter());
    printf("Counter = %d \n", incrementCounter());
}

int incrementCounter()
{
     static int count = 0; // static variable
    count++;
    return count;
}
```

Output

```
Counter = 1
Counter = 2
```

Advantages of Function

- Using function we can avoid rewriting the same logic or code again and again in a program.
- We can track or understand large program easily when it is divide into functions.
- It provides reusability.
- It help in testing and debugging because it can be tested for errors individually in the easiest way.
- Reduction in size of program due to code of a function can be used again and again, by calling it.

Practice Programs

1) WAP to count simple interest using function.
2) WAP that defines a function to add first *n* numbers.
3) WAP using global variable, static variable.
4) WAP that will scan a character string passed as an argument and convert all lowercase character into their uppercase equivalents.
5) Build a function to check number is prime or not. If number is prime then function return value 1 otherwise return 0.
6) Write a program to calculate nCr using user defined function. nCr = n! / (r! * (n-r)!)
7) Create a function to swap the values of two variables.
8) Write a function which takes 2 numbers as parameters and returns the gcd of the 2 numbers. Call the function in main().

Thank you