UST
Faculty of Engineering
First Year [Sem2]

Data Structure and Algorithms

Lecture 6

Functions in C++

Introduction to Functions

- When writing a program, we often need to perform the same task several times.
- For example, imagine a program that calculates the area of a circle.

Without functions, we might write:

</> C++

```cpp
#include <iostream>
using namespace std;

int main() {
    double radius1 = 5;
    double radius2 = 10;

    double area1 = 3.14159 * radius1 * radius1;
    double area2 = 3.14159 * radius2 * radius2;

    cout << area1 << endl;
    cout << area2 << endl;

    return 0;
}
```

The formula:

```
π x radius x radius
```

Instead, we can create a function:

```C++
double calculateArea(double radius)
```

What is a function?

- A **function** is a named block of code designed to perform a specific task.

For example:

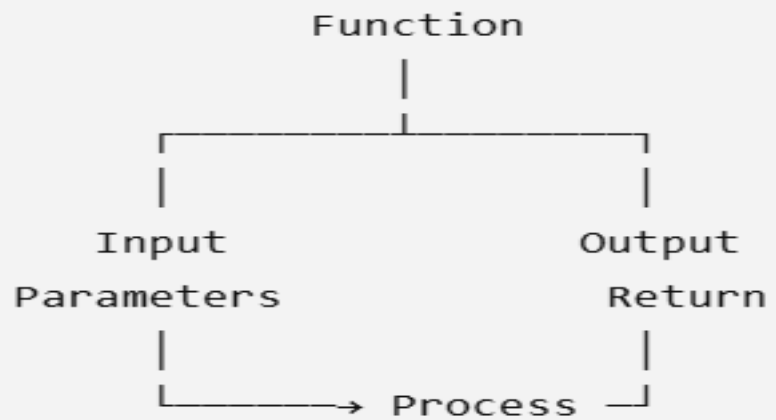
```
</>  C++

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

This function performs one task:

Add two numbers.

A function can:

- Receive data.
- Process data.
- Return a result.



Why Do We Use Functions?

Functions are important because they make programs:

1. **Easier to understand:**

Instead of putting everything inside main(), we divide the program into smaller tasks.

2. Reusable:

A function can be called many times.

3. Easier to debug:

We can test each function independently.

4. Easier to maintain:

Changing one function does not necessarily require changing the entire program.

5. Less repetitive:

- We avoid writing the same code multiple times.

6. Modular:

- A large program can be divided into smaller modules.

Function Structure

- A simple function looks like this:

```C++
returnType functionName(parameters)
{
    // statements
    return value;
}
```

Example

```
</>  C++

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

```
int         add        (int a, int b)
|           |                  |
|           |                  └── Parameters
|           └── Function name
└── Return type
```

Parts of a Function

A function generally consists of:

1.Return type

2.Function name

3.Parameter list

4.Function body

5.Return statement, when required

Example:

```C++
int multiply(int x, int y)
{
    int result = x * y;
    return result;
}
```

Return Type:

```
</>  C++
int
```

Function Name:

```
</>  C++
multiply
```

Parameters:

```
</>  C++
int x, int y
```

Function Body:

```
</>  C++
{
    int result = x * y;
    return result;
}
```

Defining a Function

- A function definition specifies what the function does.

Example:

```C++
void sayHello()
{
    cout << "Hello Engineering Students!" << endl;
}
```

This function does not receive any input and does not return a value.

Calling a Function

- Defining a function does not execute it.
- We must **call** it.

Example:

```cpp
</> C++

#include <iostream>
using namespace std;

void sayHello()
{
    cout << "Hello!" << endl;
}

int main()
{
    sayHello();

    return 0;
}
```

Output:

```
Hello!
```

Execution flow

```
main()
   |
   ▼
sayHello()
   |
   ▼
Print "Hello!"
   |
   ▼
Return to main()
```

```C++
sayHello();
```

```C++
add(5, 3);
```

Example:

```cpp
#include <iostream>
using namespace std;

void displayMessage()
{
    cout << "Welcome to C++ Programming" << endl;
}

int main()
{
    displayMessage();
    displayMessage();

    return 0;
}
```

Output:

```
Welcome to C++ Programming
Welcome to C++ Programming
```

Functions with Parameters

- A function can receive information through **parameters**.

Example:

```cpp
void greet(string name)
{
    cout << "Hello " << name << endl;
}
```

Output:

```
Hello Ahmed
```

Calls:

```cpp
greet("Ahmed");
```

Output:

```
Hello Sara
```

```cpp
greet("Sara");
```

Parameters vs Arguments

- These two terms are related but different.

Parameter

- A variable declared in the function definition.

```
</> C++

void greet(string name)
```

Here:

Name is a parameter.

Argument

- The actual value passed to the function.

```C++
greet("Ahmed");
```

Here:

Ahmed is an **Argument**.

Simple distinction

```
Parameter → placeholder
Argument  → actual value
```

Multiple Parameters

A function can have several parameters.

Example:

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

The values are passed as:

```
a = 10
b = 20
```

Call:

```C++
int result = add(10, 20);
```

Result:

```
30
```

Functions That Return a Value

- A function can calculate something and return the result.

Example:

```cpp
int square(int number)
{
    return number * number;
}
```

Call:

```cpp
int result = square(5);
```

The returned value is:

```
25
```

The return Statement

The return statement sends a value back to the calling function.

Example:

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

```
result = 10
```

If we call:

```cpp
int result = add(4, 6);
```

Function Returning double

- Functions can return different data types.

Example:

```
</> C++

double calculateAverage(double a, double b)
{
    return (a + b) / 2;
}
```

Call:

```
</> C++

double avg = calculateAverage(80, 90);
```

Result:

```
85
```

Function Returning char

Example:

```C++
char getGrade()
{
    return 'A';
}
```

Call:

```C++
char grade = getGrade();
```

Function Returning bool

Example:

```
</> C++

bool isEven(int number)
{
    return number % 2 == 0;
}
```

Call:

```
</> C++

if (isEven(10))
{
    cout << "Even";
}
```

Output:

```
Even
```

void Functions

A void function does not return a value.

Example:

```cpp
void display()
{
    cout << "Hello";
}
```

We do **not** write:

```cpp
int x = display();
```

We call it:

```cpp
display();
```

void vs Value-Returning Functions

Type	Returns value?	Example
`void`	No	`void display()`
`int`	Yes	`int add()`
`double`	Yes	`double average()`
`char`	Yes	`char getGrade()`
`bool`	Yes	`bool isEven()`

Example

```cpp
#include <iostream>
using namespace std;

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

int main()
{
    int x, y;

    cout << "Enter two numbers: ";
    cin >> x >> y;

    int result = add(x, y);

    cout << "Sum = " << result << endl;

    return 0;
}
```

```
Enter two numbers: 15 25

Sum = 40
```