Tishk International University
Faculty of Applied Science
Information Technology Department

Functions

Lecture 2

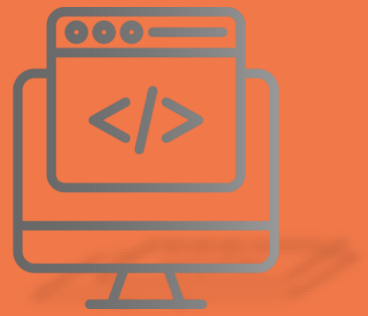
Spring 2026

Course Code: IT118

Grade 1

Islam Abdulazeez
islam.abdulaziz@tiu.edu.iq

April 28, 2026



Programming II

Outlines

- ✓ Introduction to Functions
- ✓ Function Definition & Calling
- ✓ Parameters & Return Values
- ✓ Function Prototypes
- ✓ Arrays with Functions
- ✓ Overloading Function

Learning Outcomes

- **At the end of today's session, you will be able to:**

  - ✓ Define functions and their role in programming.
  - ✓ Explain parameters, return values, and prototypes.
  - ✓ Write simple functions in C++.
  - ✓ Design basic modular programs using functions.

Introduction to Functions

- **Modular programming:** Breaking a program up into smaller units.
- They promote modularity, making code more organized, readable, reusable and maintainable.
- **Function:** Is a set of statements that work together to perform a specific task

Advantages for modular programming

- ✓ Simplifies the process of writing programs
- ✓ Improves debugging of programs
- ✓ Reducing redundancy

Introduction to Functions

- Functions consist of a **name**, **parameters** (if any), a **return type** (if any), and a **body** containing the executable code.
- **name:** name of the function. Function names follow the same rules as variable names.
- **parameter list:** variables that hold the values passed to the function.
- **body:** statements that perform the function's task .
- **Return type:** the data type of the value that the function returns to the part of the program that called it.

Introduction to Functions

This program has one long, complex function containing all of the statements necessary to solve a problem.

```
int main()
{
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
}
```

In this program the problem has been divided into smaller problems, each handled by a separate function.

```
int main()
{
   statement;
   statement;
   statement;
}
```

main function

```
void function2()
{
   statement;
   statement;
   statement;
}
```

function 2

```
void function3()
{
   statement;
   statement;
   statement;
}
```

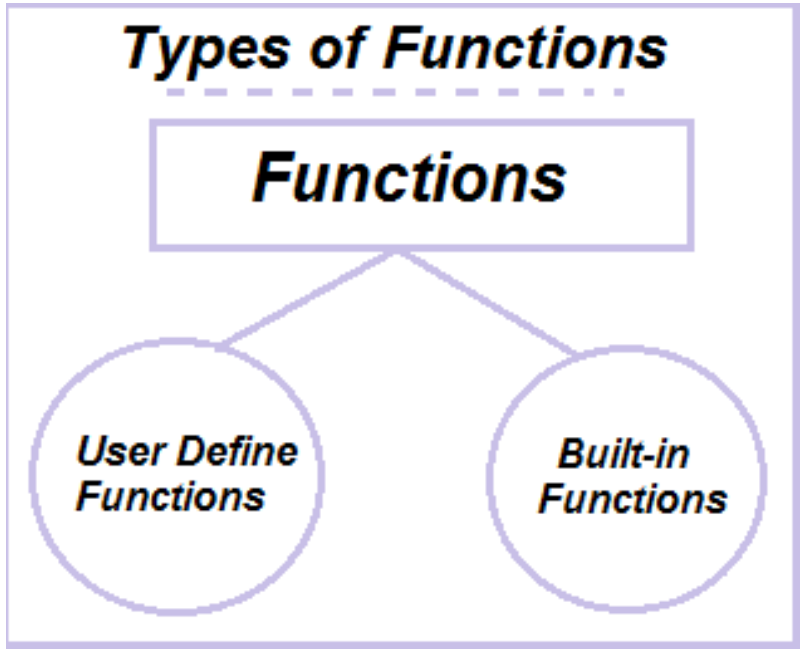
function 3

```
void function4()
{
   statement;
   statement;
   statement;
}
```

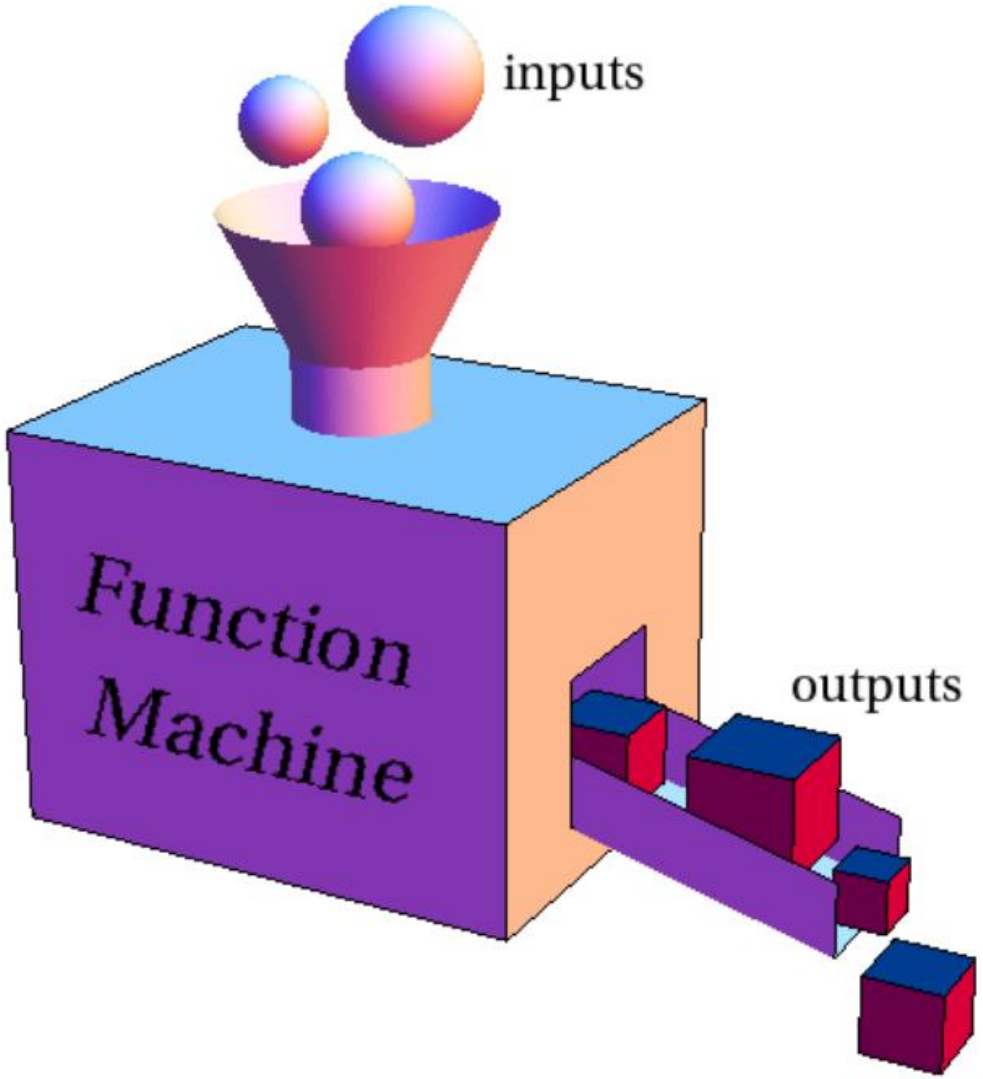
function 4

Types of Functions

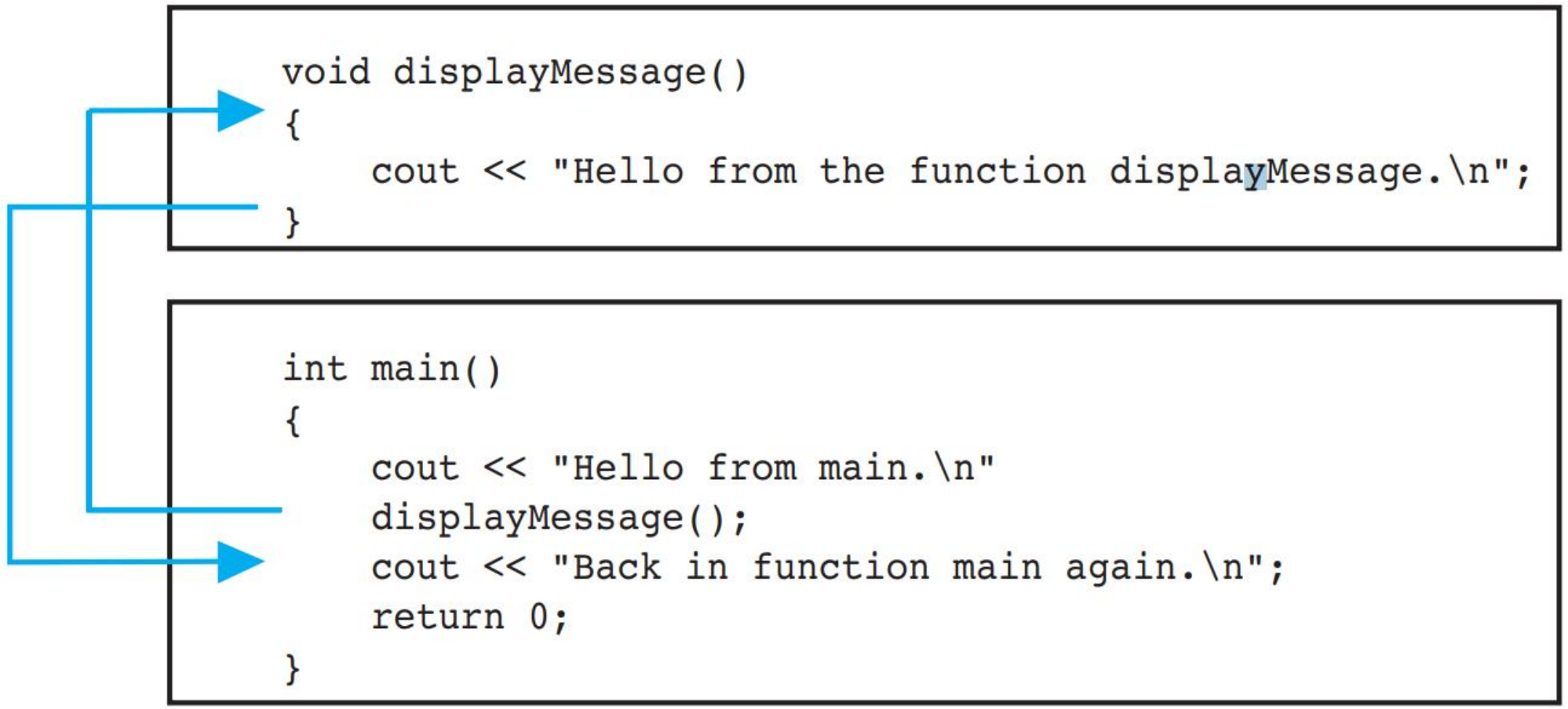
- **User-Defined Functions:** Functions created by the programmer to perform specific tasks.
- **Built-in Functions:** Functions that are already provided by the programming language (you don't need to write them).



Function Concept



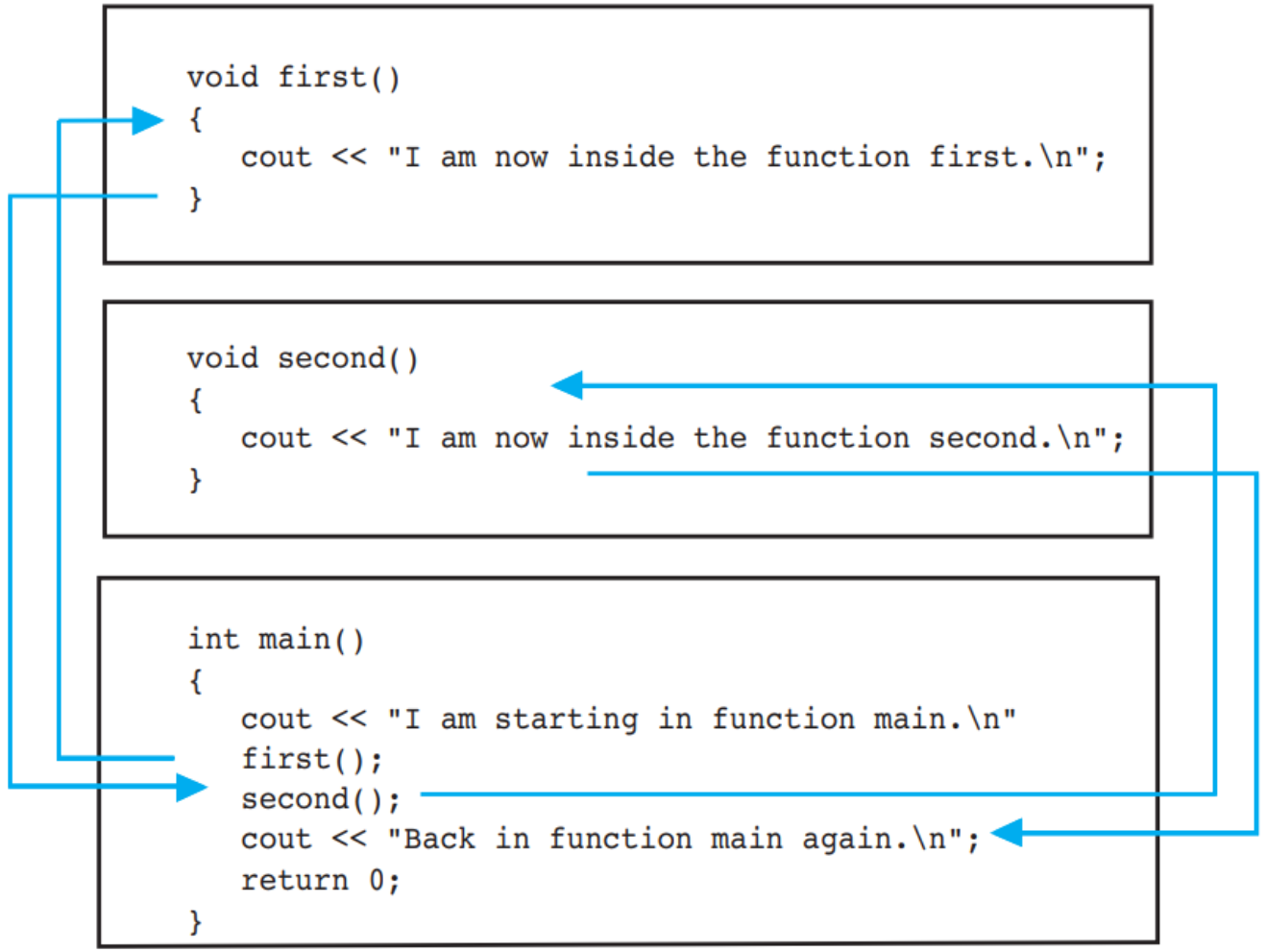
Define and Call Function in C++



```
void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}

int main()
{
    cout << "Hello from main.\n"
    displayMessage();
    cout << "Back in function main again.\n";
    return 0;
}
```

Calling a Function

- main function is automatically called when the program starts
- main can call any number of functions



Calling a Function

- Functions can call other functions

```
void greetProgrammers() {
    cout << "programmers" << endl;
}

void welcomeMessage() {
    cout << "Welcome!" << endl;
    greetProgrammers();
}

int main() {

    welcomeMessage();

    return 0;
}
```

Function Calling Inside Loop

```
void displayMessage() {
    cout << "Hello from the function displayMessage." << endl;
}

int main() {
    cout << "Hello from main." << endl;

    for (int count = 0; count < 3; count++) {
        displayMessage();
    }

    cout << "Back in function main again." << endl;
```

```
Hello from main.
Hello from the function displayMessage.
Hello from the function displayMessage.
Hello from the function displayMessage.
Back in function main again.
```

Pass by Value

- When an argument is passed to a function, a **copy of its value** is stored in the parameter.

Parameter

```
void welcomeUser(string name) {
    cout << "Welcome " << name << "!" << endl;
}

int main() {
    string userName = "Baran";
    welcomeUser(userName);

    return 0;
}
```

Parameter

```
void welcomeUser(string name) {
    cout << "Welcome, " << name << "!" << endl;
}

int main() {
    string userName;
    cout << "Input the name: ";
    cin >> userName;

    welcomeUser(userName);
```

Returning a Value From a Function

- Return statement can be used to return a value from the function to the module that made the function call.

Return Type

```
int sum(int x, int y) {
    return x + y;
}

int main() {

    cout << sum(3, 6);

    return 0;
}
```

Returning a Value From a Function

- Calling function should use return value, e.g.,
  - Assign it to a variable
  - Send it to cout
  - Use it in an arithmetic computation
  - Use it in a relational expression

```
int square(int x) {
    return x * x;
}

int main() {
    int number = 5;
    int squaredValue = square(number);
    cout << "The square of " << number << " is: " << squaredValue << endl;

    return 0;
}
```

Returning a Value From a Function

- Calling function should use return value, e.g.,
  - Assign it to a variable
  - Send it to cout
  - Use it in an arithmetic computation
  - Use it in a relational expression

```
int square(int x) {
    return x * x;
}

int main() {
    int number = 5;
    cout << "The square of " << number << " is: " << square(number) << endl;

    return 0;
}
```

Returning a Value From a Function

- Calling function should use return value, e.g.,
  - Assign it to a variable
  - Send it to cout
  - Use it in an arithmetic computation
  - Use it in a relational expression

```
int square(int x) {
    return x * x;
}

int main() {
    int number = 5;
    int squaredValue = square(number);
    int multipliedValue = squaredValue * 2;

    cout << "The square of " << number << " and * by 2 is: " << multipliedValue << endl;

    return 0;
}
```

Returning a Value From a Function

- Calling function should use return value, e.g.,
  - Assign it to a variable
  - Send it to cout
  - Use it in an arithmetic computation
  - Use it in a relational expression

```
int square(int x) {
    return x * x;
}

int main() {
    int number = 5;
    int squaredValue = square(number);

    if (squaredValue > 20) {
        cout << "The square of " << number << " is greater than 20." << endl;
    }
    else {
        cout << "The square of " << number << " is not greater than 20." << endl;
    }

    return 0;
}
```

A function with Boolean return

```
bool isEven(int number) {
    if (number % 2 == 0) {
        return true;
    }
    else {
        return false;
    }
}

int main() {
    int val;

    cout << "Enter an integer and I will tell you ";
    cout << "if it is even or odd: ";
    cin >> val;

    if (isEven(val)) {
        cout << val << " is even.\n";
    }
    else {
        cout << val << " is odd.\n";
    }

    return 0;
}
```

It can be written like this too

```
if (isEven(val) == true) {
```

A function with Boolean return

All the same

```
bool isEven(int number){
    if (number % 2 == 0){
        return true;
    } else {
        return false;
    }
}
```

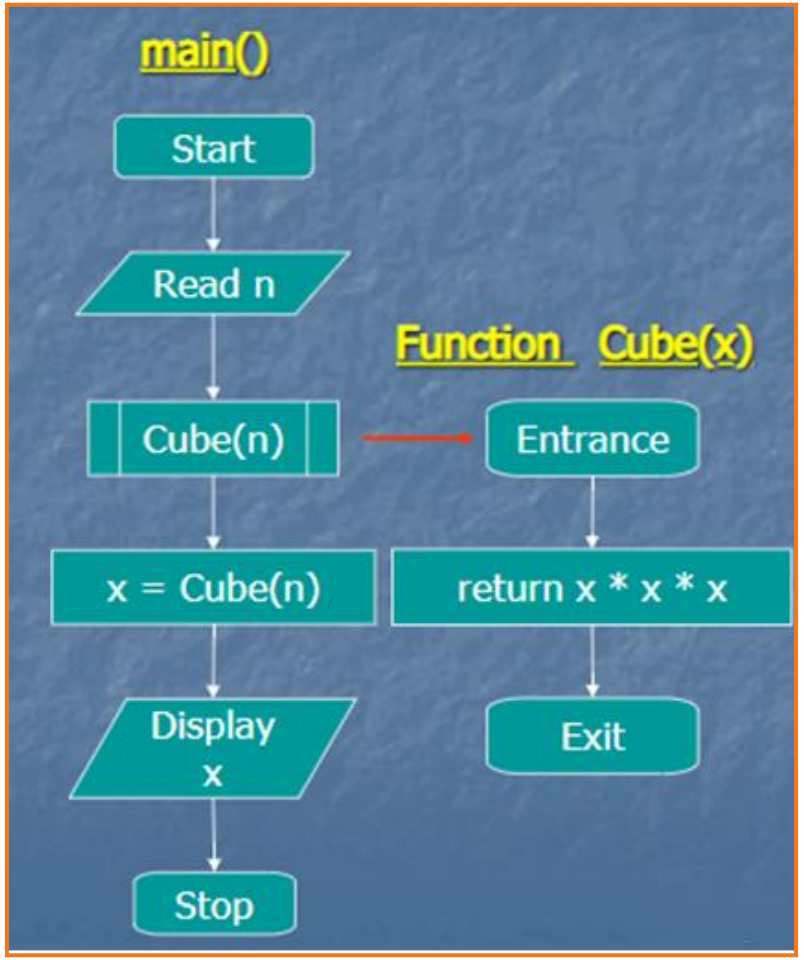
```
bool isEven(int number){
    bool answer;
    if (number % 2 == 0){
        answer = true;
    }
    return answer;
}
```

```
bool isEven(int number){
    bool answer = false;
    if (number % 2 == 0){
        answer = true;
    }
    return answer;
}
```

Example: Cube Function, using Parameters

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

double Cube(double a) {
    return a*a*a;
}

int main(){
    double v, a;
    cout << "Give the cube side ";
    cin >> a;
    v = Cube(a);
    cout << " The Volume is " << v;
    cout << endl;
    return 0;
}
```

Parameter



Example: Sum Function

- Write a function "Sum" that will take two integer numbers n and m and return the sum of all the numbers from n to m.

i.e.

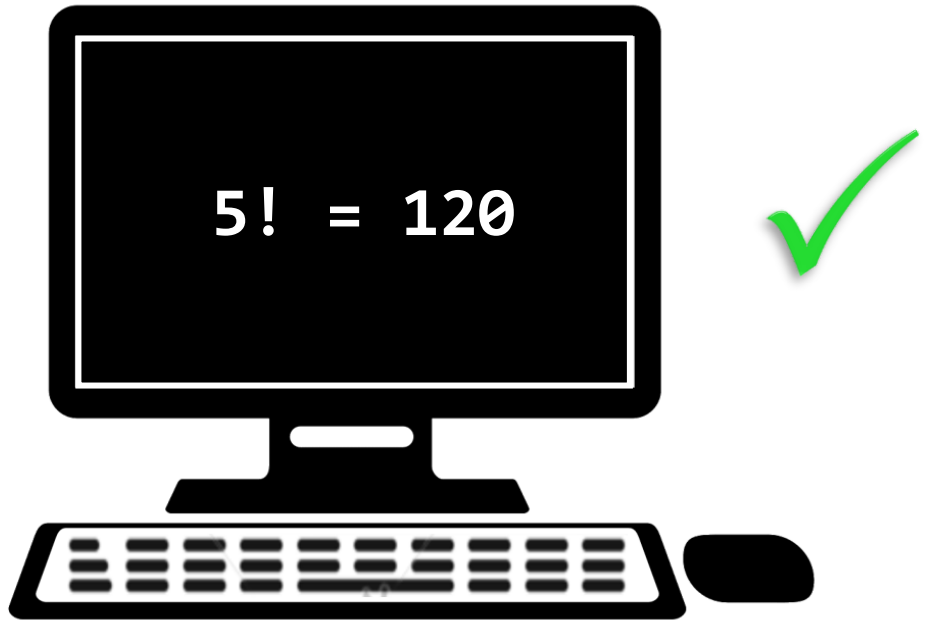
- Sum(1,4) = 1 + 2 + 3 + 4 = 10
- Sum(4,9) = 4 + 5 + 6 + 7 + 8 + 9 = 39
- Sum(7,7) = 7
- Sum(7,2) = 0

Example: Sum Function

```
int Sum(int n, int m) {
    int s = 0;
    for (int i = n; i <= m; i++) {
        s = s + i;
    }
    return s;
}

int main() {
    int x, y, z;

    cout << "Input First number: ";
    cin >> x;

    cout << "Input Second number: ";
    cin >> y;

    z = Sum(x, y);

    cout << "The sum is " << z << endl;

    return 0;
}
```

Example: Factorial Calculation

```
int myFact(int a) {
    int fact = 1;
    for (int i = 1; i <= a; i++) {
        fact = fact * i;
    }
    return fact;
}

int main() {
    int x = 5;
    cout << x << "! = " << myFact(x) << endl;
    return 0;
}
```



Function Prototype

- A **function prototype** in C++ is a declaration of a function that tells the compiler:
  - the function name
  - the return type
  - the number and types of parameters
- It is written before the function is used, usually at the top of the program.
- Why is it important?
  - Allows using functions before defining them
  - Checks correct arguments (number & type)
  - Keeps code organized

Function Prototype

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

// function prototype
int add(int x, int y);

int main() {
    int a = 10, b = 20;
    int z = add(a, b);
    cout << "The sum of " << a << " and " << b << " is " << z << endl;
    return 0;
}

// function definition
int add(int x, int y) {
    return x + y;
}
```

Function without Prototype

- The function (add) is coming after main function, and it hasn't defined as a function prototype

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

int main() {
    int a = 10, b = 20;
    int z = add(a, b);
    cout << "The sum of " << a << " and " << b << " is " << z << endl;
    return 0;
}

int add(int x, int y) {
    return x + y;
}
```

Function without Prototype

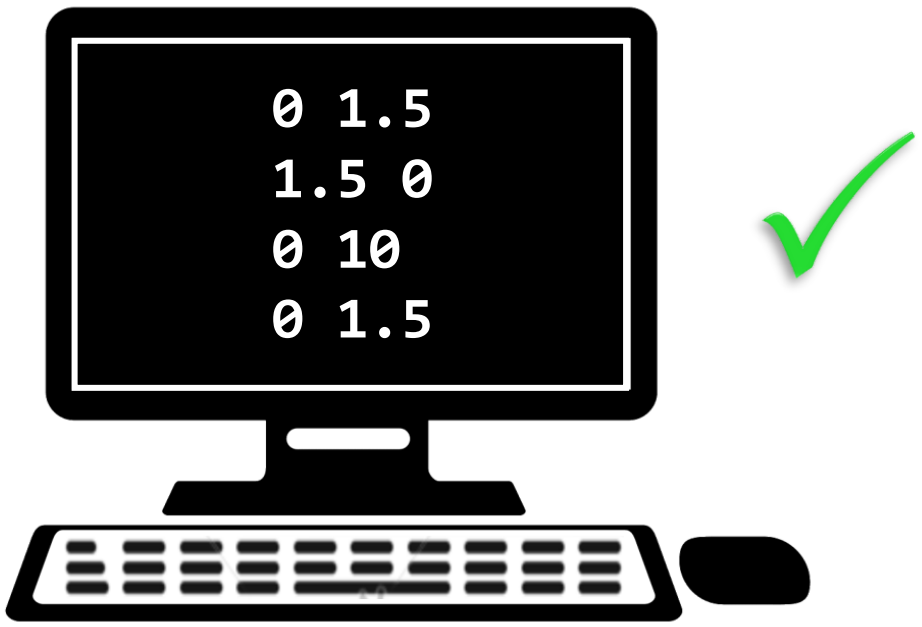
- The function (add) is coming before main function, and it is correct and doesn't need a function prototype

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

// function definition without prototype

int add(int x, int y) {
    return x + y;
}

int main() {
    int a = 10, b = 20;
    int z = add(a, b);
    cout << "The sum of " << a << " and " << b << " is " << z << endl;
    return 0;
}
```

Parameter Values

- Changes to the parameter in the function do not affect the value of the argument in the calling function.

```
void func1(double, int); // Function prototype

int main() {
    int x = 0;
    double y = 1.5;

    cout << x << " " << y << endl;
    func1(y, x);
    cout << x << " " << y << endl;

    return 0;
}

void func1(double a, int b) {
    cout << a << " " << b << endl;

    a = 0.0;
    b = 10;

    cout << a << " " << b << endl;
}
```



A function with Boolean return

```cpp
bool isValid(int val) {
    int min = 0, max = 100;
    bool status;
    if (val >= min && val <= max) {
        status = true;
    }
    else {
        status = false;
    }
    return status;
}

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if (isValid(num)) {
        cout << "The number is within the valid range (0 - 100)." << endl;
    }
    else {
        cout << "The number is NOT within the valid range." << endl;
    }
    return 0;
}
```

Same Result

```cpp
bool isValid(int val){
    if(val >= 0 && val <= 100){
        return true;
    } else {
        return false;
    }

}
```

Arrays with Functions

- Functions can accept arrays as parameters and perform operations on array elements.

void function type

```
void sumArray(int arr[], int size) {
    int sum = 0;

    for (int i = 0; i < size; i++) {
        sum = sum + arr[i];
    }

    cout << "The sum= " << sum;
}

int main() {
    int A[] = { 1, 2, 3, 4, 5 };
    int size = sizeof(A) / sizeof(A[0]);

    sumArray(A, size);

    return 0;
}
```

int function type

```
int sumArray(int arr[], int size) {
    int sum = 0;

    for (int i = 0; i < size; i++) {
        sum = sum + arr[i];
    }
    return sum;
}

int main() {
    int A[] = { 1, 2, 3, 4, 5 };
    int size = sizeof(A) / sizeof(A[0]);
    int sum = sumArray(A, size);
    cout << "The sum= " << sum;
    return 0;
}
```

Default Parameters

- **Default Parameters** are values a function uses automatically when no argument is given.

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

void myFunction(int a = 10, int b = 1) {
    cout << "a= " << a << endl;
    cout << "b= " << b << endl;
}

int main() {
    myFunction();
    cout << endl;

    myFunction(5);
    cout << endl;

    myFunction(7, 3);
    return 0;
}
```

Output

```
a= 10
b= 1

a= 5
b= 1

a= 7
b= 3
```

Overloading Function

- Two or more functions may have the same name, as long as their parameter lists are different.

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

int myTest(int x) {
    return x * x;
}

string myTest(string x) {
    return "Your name is " + x;
}

int main() {

    int x = 17;
    string name = "Ramyar";

    cout << myTest(name) << endl;
    cout << myTest(x) << endl;

    return 0;
}
```

Overloading Function

- How does the compiler know which version of “square() ” to call?
- It looks at the arguments passed in each case.
- Then It looks to find the right version of square() that match the correct argument.

```
/*overloaded function square. This function returns the square of the value
 passed into its double parameter. */
double square(double number){
 return number * number;
}

/*overloaded function square. This function returns the square of the value
 passed into its int parameter. */
int square(int number){
 return number * number;
}


int main(){
    int userInt;
    double userReal;

    cout << "Enter an integer value: ";
    cin >> userInt;
    cout << "Here are their squares: ";

    cout << square(userInt)<<endl;
    cout << "Enter double value: ";
    cin >> userReal;
    cout << "Here are their squares: ";
    cout << square(userReal) << endl;


    return 0;
}
```

Output

Output

```
Enter an integer value: 4
Here is its square: 16
Enter a double value: 5.5
Here is its square: 30.25
```

Activities and Next Lecture's Topic

Activities

- Review this lecture note
- Practice

Next Lecture's Topic

- Global Variables & Pointers

References

- Gaddis, T. (2014). Starting out with C++: Early objects (7th ed.). Pearson Education.

Thank You!