

Functions

Lecture Objectives

- To create functions, invoke functions, and pass arguments to a function
- To determine the scope of local and global variables
- To understand the differences between *pass-by-value* and *pass-by-reference*
- To use function overloading and understand ambiguous overloading
- To use function prototypes for declaring function headers
- To know how to use default arguments
- Static Variables

Functions in C++

- Experience has shown that the **best way to develop and maintain large programs** is to construct it from **smaller pieces** (modules)
- This technique Called “**Divide and Conquer**”

Bad Development Approach

```
main()
{
    -----
    -----
    -----
    -----
    .
    .
    .
    -----
    -----
    -----
    return 0;
}
```

Easier To >>

- ✓ Design
- ✓ Build
- ✓ Debug
- ✓ Extend
- ✓ Modify
- ✓ Understand
- ✓ Reuse

Wise Development Approach

```
main()
{
    -----
    -----
}

function f1()
{
    ---
    ---
}

function f2()
{
    ---
    ---
}
```

C++ Functions

- A function is a block of code that performs a specific task.
- Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:
 - a function to draw the circle
 - a function to color the circle
- Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.

Functions in C++(Cont.)

- In C++ **modules** Known as **Functions & Classes**
- Programs may use **new** and “**prepackaged**” or built-in modules
 - **New**: programmer-defined **functions** and **classes**
 - **Prepackaged**: from the *standard library*

Calling Functions

- Examples (built-in, and user-defined functions)

```
int n = getPIValue( ) ; //Takes no argument
```

← **A user-defined function**

```
cout << sqrt(9); //Takes one argument, returns square-root
```

```
cout<<pow(2,3); //Calculates 2 power 3
```

```
cout<<SumValues(myArray); //Returns sum of the array
```

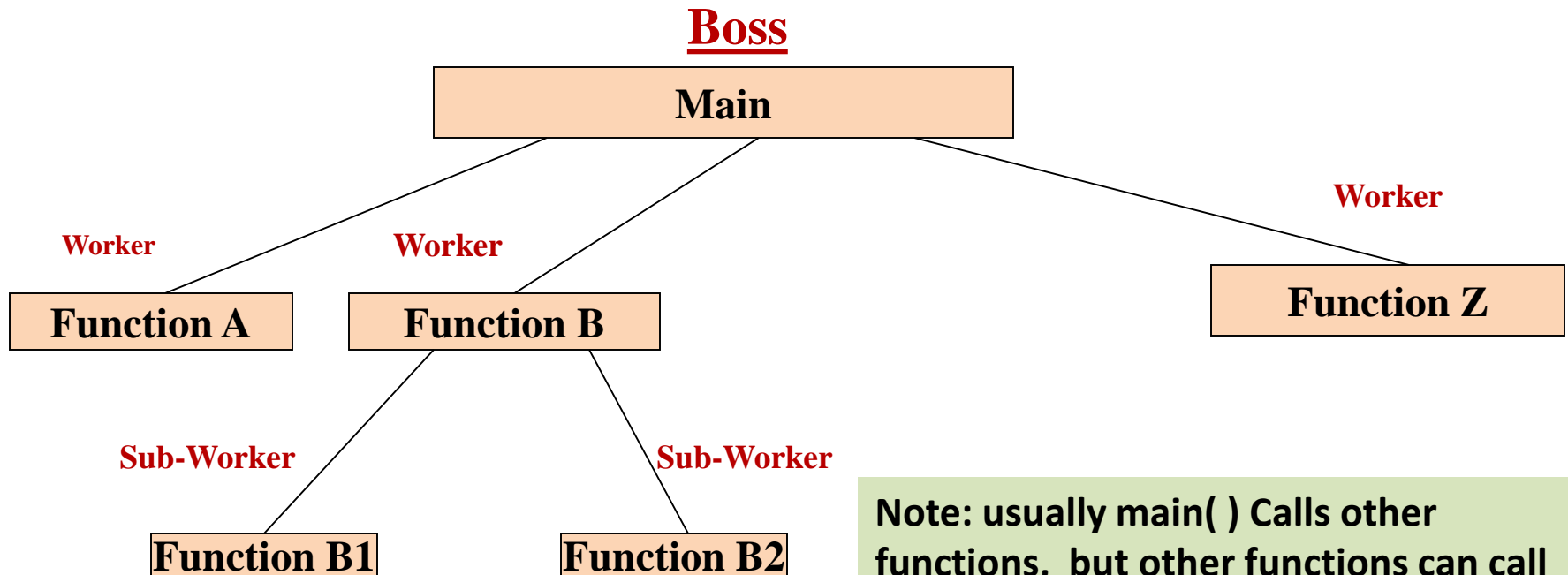
← **A user-defined function**

1. User-defined Function

- C++ allows the programmer to define their own function.
- A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
- When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

About Functions in C++

- Functions invoked by a function–call-statement which consist of it's name and information it needs (*arguments*)
- Boss To Worker Analogy:

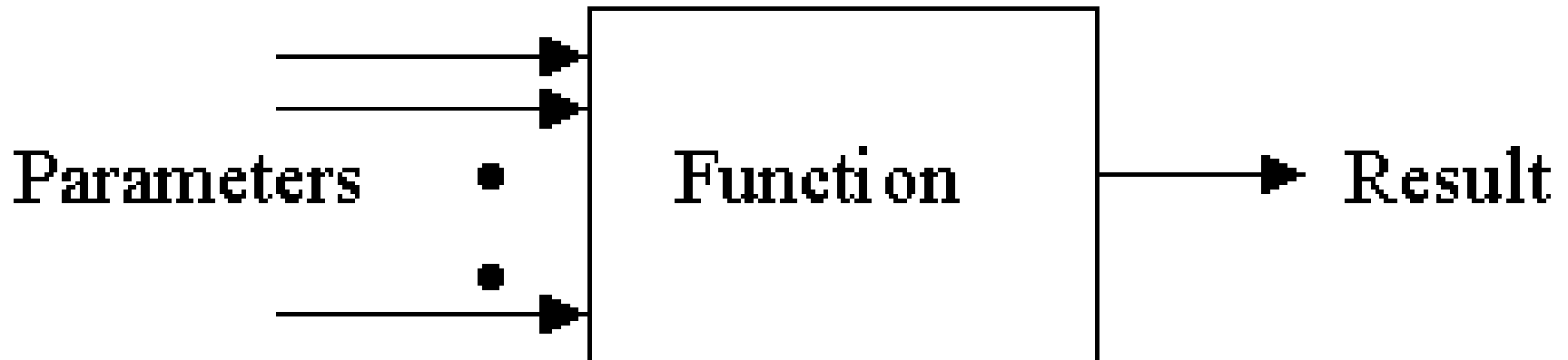


Note: usually `main()` Calls other functions, but other functions can call each other

Calling Function

- **Function calls:**

- Provide **function name** and **arguments (data)**:
Function performs operations and
Function returns results



Calling Functions

- Functions calling (Syntax):

<function name> (<argument list>);

E.g.,

FunctionName();

or

FunctionName(argument1);

or

FunctionName(argument1, argument2, ...);

Function Definition

- Syntax format for function definition

```
returned-value-type  function-name (parameter-list)
{
    Declarations of local variables and Statements;
    ...
}
```

- Parameter list

- Comma separated list of arguments
 - Data type needed for each argument
- If no arguments → leave blank

- Return-value-type

- Data type of result returned (use **void** if nothing will be returned)

Function Prototype

- Before a function is called, it must be declared first.
- Functions cannot be defined inside other functions
- A function prototype is a function declaration without implementation (the implementation can be given later in the program).

```
int multiplyTwoNums (int, int) ;
```



A Function prototype (declaration
without implementation)

Function Declaration

- The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {  
    // function body  
}
```

```
// function declaration  
void greet() {  
    cout << "Hello World";  
}
```

- the name of the function is `greet()`
- the return type of the function is `void`
- the empty parentheses mean it doesn't have any parameters
- the function body is written inside `{ }`

Calling a Function

- The syntax to declare a function is:

In the above program, we have declared a function named `greet()`. To use the `greet()` function, we need to call it.

Here's how we can call the above `greet()` function.

```
int main() {  
  
    // calling a function  
    greet();  
  
}
```

How Function works in C++

- The syntax to declare a function is:

```
#include<iostream>
```

```
void greet() {
```

```
    // code
```

```
}
```

```
int main() {
```

```
    ... ..
```

```
    greet();
```

```
    ... ..
```

```
}
```

**function
call**

Example 1: Display a Text

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

// declaring a function
void greet() {
    cout << "Hello there!";
}

int main() {

    // calling the function
    greet();

    return 0;
}
```

Output

```
Hello there!
```


Function Parameters

- A function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.
- For example, let us consider the function below:

```
void printNum(int num) {  
    cout << num;  
}
```

Here, the int variable num is the function parameter.

```
int main() {  
    int n = 7;  
  
    // calling the function  
    // n is passed to the function as argument  
    printNum(n);  
  
    return 0;  
}
```

Example 2: Function with Parameters

```
// program to print a text

#include <iostream>
using namespace std;

// display a number
void displayNum(int n1, float n2) {
    cout << "The int number is " << n1;
    cout << "The double number is " << n2;
}

int main() {

    int num1 = 5;
    double num2 = 5.5;

    // calling the function
    displayNum(num1, num2);

    return 0;
}
```

Output

```
The int number is 5
The double number is 5.5
```

Continue.....

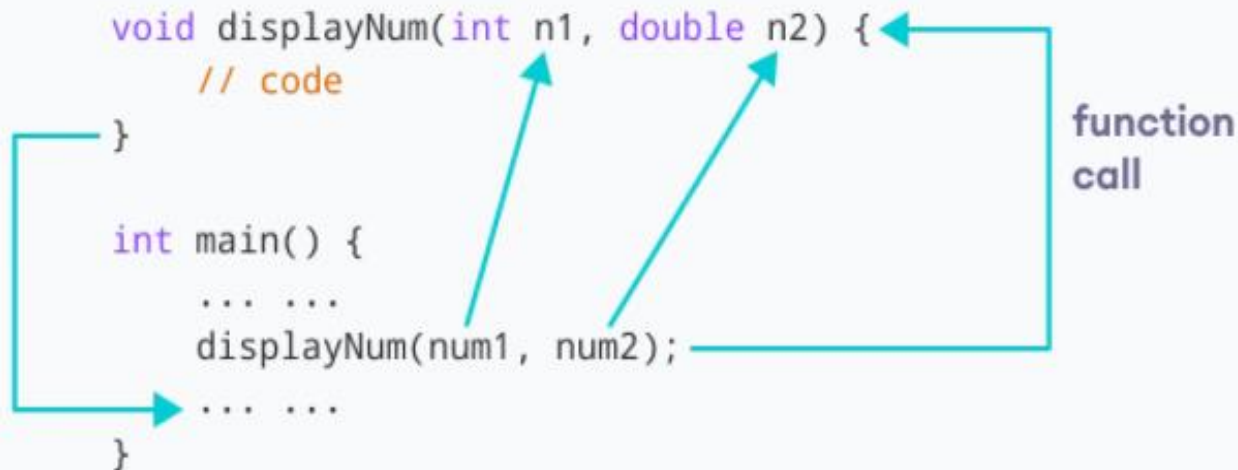
In the above program, we have used a function that has one `int` parameter and one `double` parameter.

We then pass `num1` and `num2` as arguments. These values are stored by the function parameters `n1` and `n2` respectively.

```
#include<iostream>

void displayNum(int n1, double n2) {
    // code
}

int main() {
    ... ..
    displayNum(num1, num2);
    ... ..
}
```



The diagram illustrates the function call process. Two teal arrows originate from the arguments `num1` and `num2` in the `displayNum(num1, num2);` line within the `main()` function. These arrows point to the parameters `int n1` and `double n2` in the `displayNum` function definition. A third teal arrow originates from the `displayNum(num1, num2);` line and points to the opening curly brace of the `displayNum` function, indicating the transfer of control to the function. The text "function call" is placed to the right of this arrow.

Function Prototype (cont.)

- Before actual implementation of a function, a function prototype can be used.
- **Why it is needed?**
 - It is required to declare a function prototype before the function is called.

Function Prototype (cont.)

```
void main()
```

```
{
```

```
    int sum = AddTwoNumbers (3,5) ;
```

```
    cout<<sum;
```

```
}
```

Error: Un-defined function



```
int AddTwoNumbers (int a, int b)
```

```
{
```

```
    int sum = a+b;
```

```
    return sum;
```

```
}
```

Function Prototype (cont.)

Solution-1

```
int AddTwoNumbers(int a, int b)
{
    int sum = a+b;
    return sum;
}

void main()
{
    int sum = AddTwoNumbers(3,5);
    cout<<sum;
}
```

Solution-2

```
int AddTwoNumbers(int, int);

void main()
{
    int sum = AddTwoNumbers(3,5);
    cout<<sum;
}

int AddTwoNumbers(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

Function Prototype

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example,

```
// function prototype
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

Continue....

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.

The syntax of a function prototype is:

```
returnType functionName(dataType1, dataType2, ...);
```


Example 4: C++ Function Prototype

```
// using function definition after main() function
// function prototype is declared before main()

#include <iostream>

using namespace std;

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

int main() {
    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}

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

Output

100 + 78 = 178

Function signature and Parameters

- **Function signature** is the combination of the function name and the parameter list.
- Variables defined in the function header are known as **formal parameters**.
- When a function is invoked, you pass a value to the parameter. This value is referred to as **actual parameter or argument**.

Function's return values

- A function may return a value:
 - *returnValueType* is the **data type** of the **value** the function returns.
- If function does not return a value, the *returnValueType* is the keyword **void**.
- For example, the *returnValueType* in the main function is **void**.

Return Statement

In the above programs, we have used void in the function declaration. For example,

```
void displayNumber() {  
    // code  
}
```

This means the function is not returning any value.

Continue.....

It's also possible to return a value from a function. For this, we need to specify the `returnType` of the function during function declaration.

Then, the `return` statement can be used to return a value from a function.

For example,

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

Here, we have the data type `int` instead of `void`. This means that the function returns an `int` value.

The code `return (a + b);` returns the sum of the two parameters as the function value.

The `return` statement denotes that the function has ended. Any code after `return` inside the function is not executed.

Example 3: Add Two Numbers

```
// program to add two numbers using a function

#include <iostream>

using namespace std;

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

int main() {

    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}
```

Output

```
100 + 78 = 178
```

Continue.....

In the above program, the `add()` function is used to find the sum of two numbers.

We pass two `int` literals `100` and `78` while calling the function.

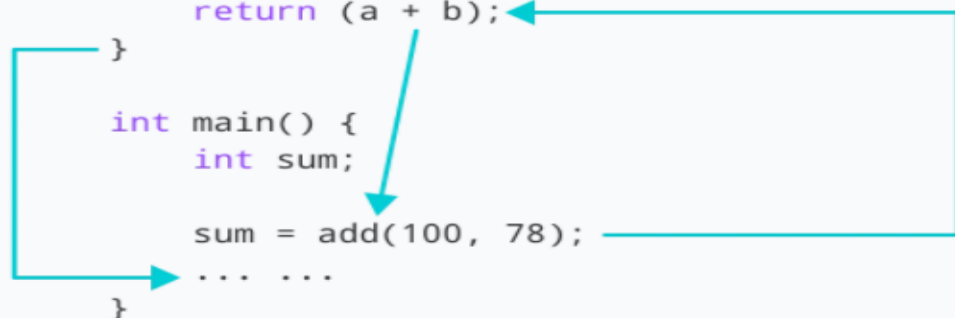
We store the returned value of the function in the variable `sum`, and then we print it.

```
#include<iostream>

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

int main() {
    int sum;

    sum = add(100, 78);
    ... ..
}
```



The diagram illustrates the execution flow. A teal arrow points from the `add(100, 78);` line in the `main()` function to the `return (a + b);` line in the `add()` function. Another teal arrow points from the `return` statement back to the `sum =` assignment in `main()`. A large teal box encloses the `add()` function definition and the call to it, with the label "function call" to its right.

Working of C++ Function with return statement

Notice that `sum` is a variable of `int` type. This is because the return value of `add()` is of `int` type.

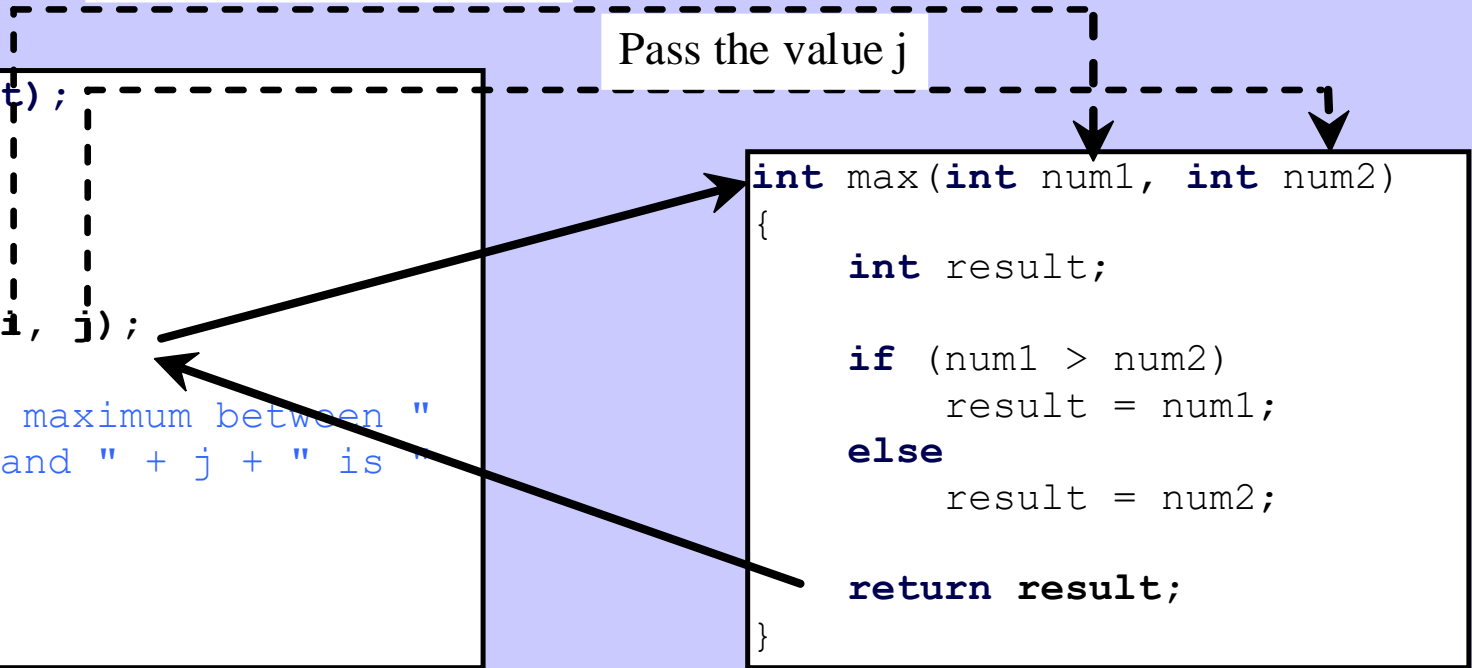
Calling Functions

Pass the value i

Pass the value j

```
int max(int,int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```



Calling Functions

i is now 5

Pass the value i

Pass the value j

```
int max(int num1, int num2);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

j is now 2

Pass the value i

Pass the value j

```
int max(int, int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

Call `max(5, 2)`

Pass the

Pass the value j

```
int max(int,int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

Now num1=5 num2=2

Pass the value i

Pass the value j

```
int max(int,int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

A new variable **result** will be created (local to the max function)

Pass the value i

Pass the value j

```
int max(int,int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

5 > 2 → true condition

Pass the value i

Pass the value j

```
int max(int,int);
int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);

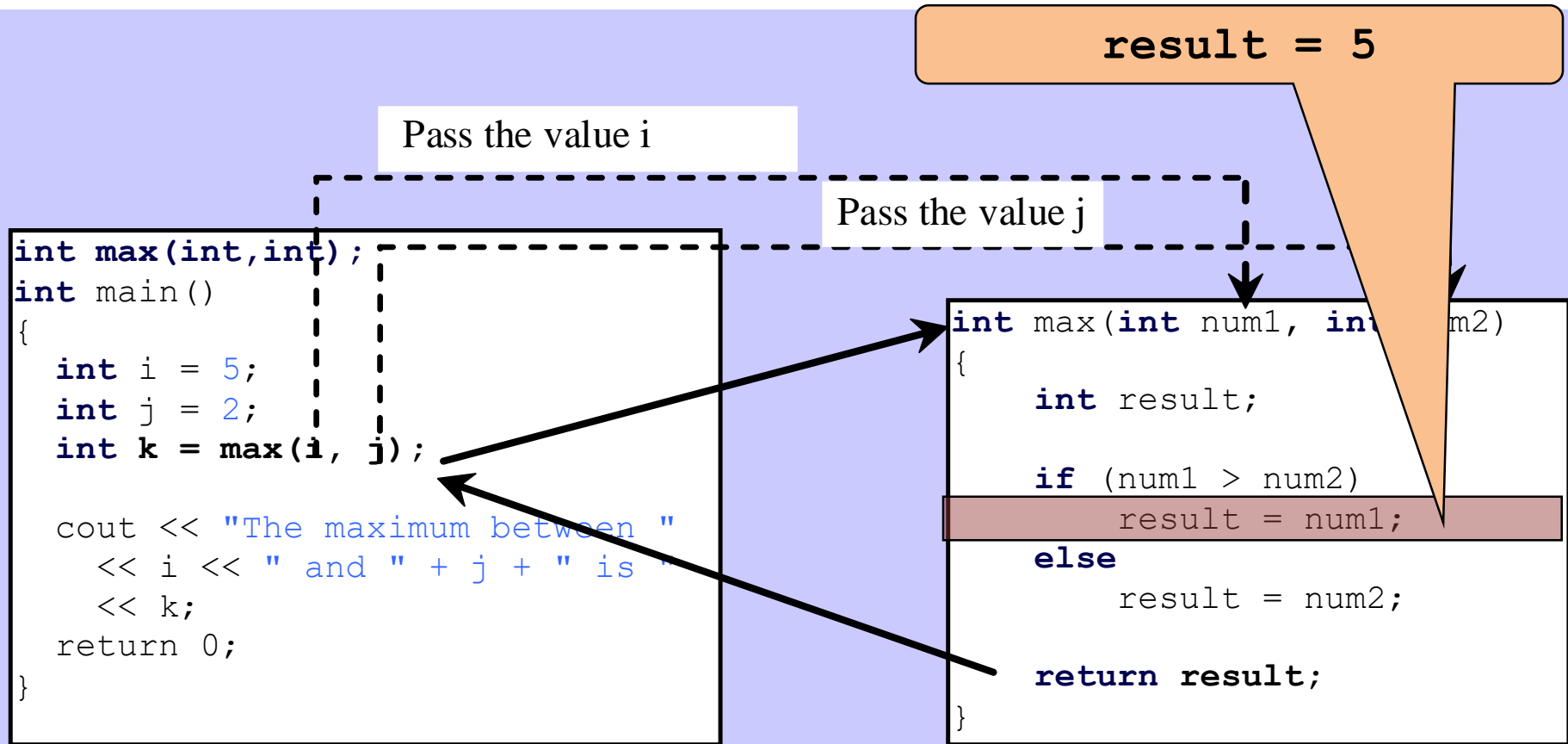
    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;
    return 0;
}
```

```
int max(int num1, int num2)
{
    int result;

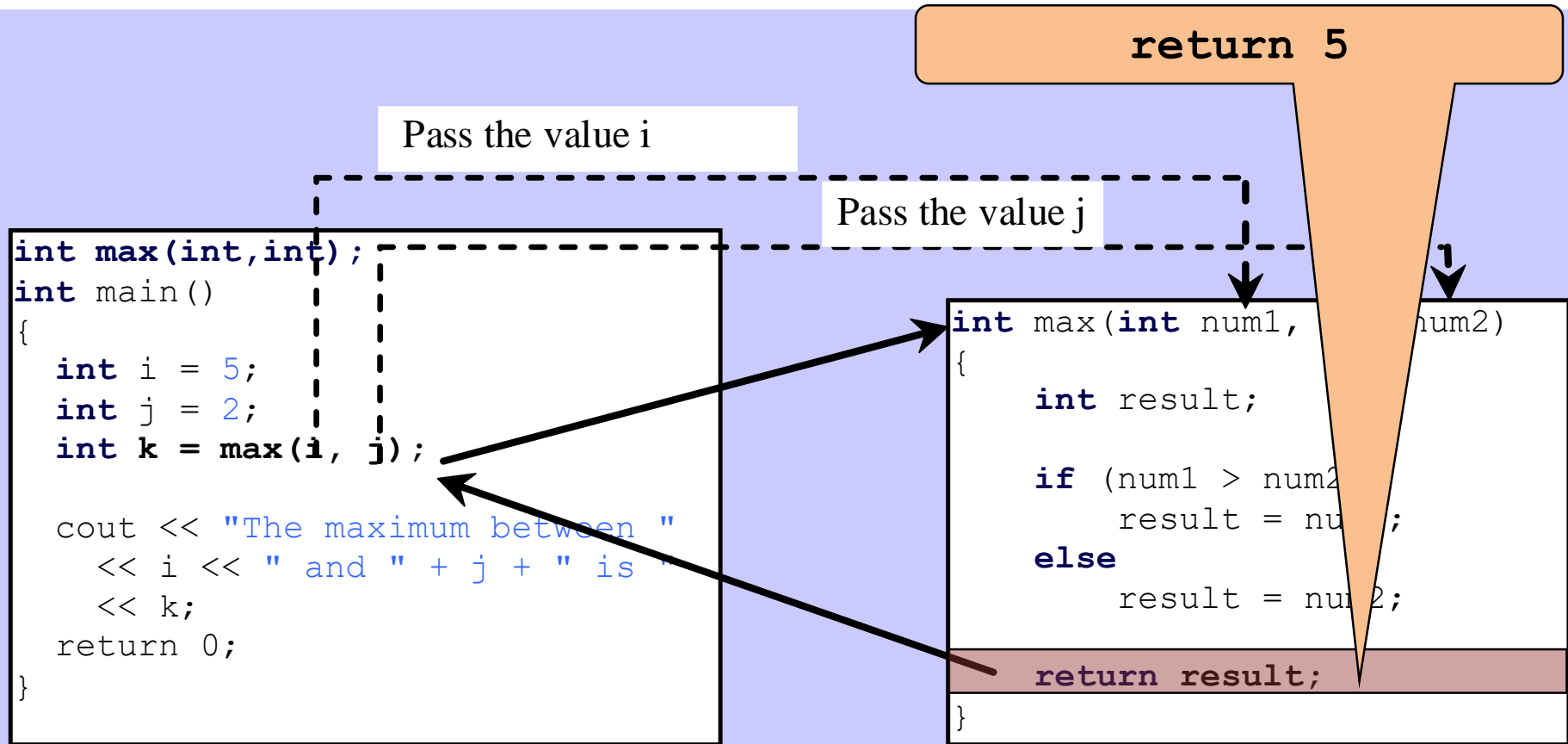
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Calling Functions



Calling Functions



Calling Functions

k = 5

Pass the value i

Pass the value j

```
int max(int, int);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

Display "The maximum between 5 and 2 is 5"

Pass the value i

Pass the value j

```
int max(int i, int j);  
int main()  
{  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);
```

```
    cout << "The maximum between "  
        << i << " and " + j + " is "  
        << k;  
    return 0;  
}
```

```
int max(int num1, int num2)  
{  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Calling Functions

main function ends

Pass the value i

Pass the value j

```
int max(int,int)
int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);

    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;

    return 0;
}
```

```
int max(int num1, int num2)
{
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

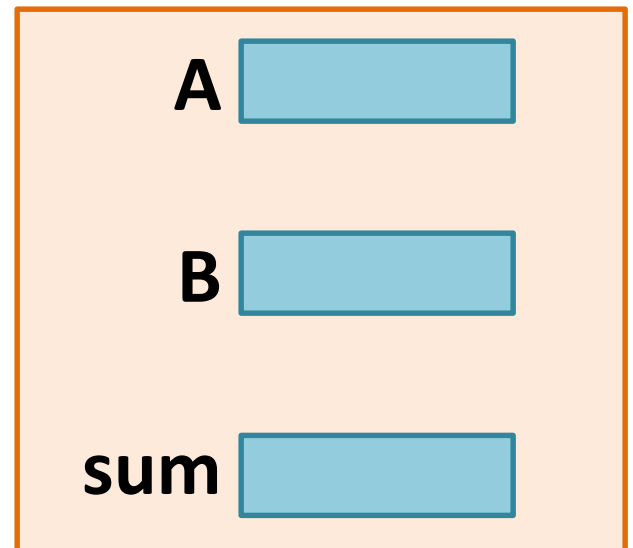
    return result;
}
```

Variables scope

- Formal parameters and variables declared within a function body are local to that function:
 - Cannot be accessed outside of that function

```
int add(int A, int B)
{
    int sum = a+b;
    return sum;
}
```

Memory (for function **add**)



Variables scope

- Global variables with same name:

```
int sum=55;
```

```
void main()
```

```
{
```

```
...
```

```
}
```

```
void display()
```

```
{
```

```
int sum = 66;
```

```
cout<<sum; // Display 66
```

```
}
```

Global Memory

sum 55

Memory (for function display)

sum 66

Variables scope

- Global variables with same name:

```
int sum=55;
```

```
void main()
```

```
{
```

```
...
```

```
}
```

```
void display()
```

```
{
```

```
    int sum = 66;
```

```
    cout<<::sum; // Display 55
```

```
}
```

Global Memory

sum

55

Memory (for function **display**)

sum

66

Benefits of Using User-Defined Functions

- Functions make the code reusable. We can declare them once and use them multiple times.
- Functions make the program easier as each small task is divided into a function.
- Functions increase readability.

C++ Library Functions

- Library functions are the built-in functions in C++ programming.
- Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.
- Some common library functions in C++ are `sqrt()`, `isdigit()`, etc.
- In order to use library functions, we usually need to include the header file in which these library functions are defined.
- For instance, in order to use mathematical functions such as `sqrt()`, `isdigit()` we need to include the header file `cmath`.

Example 5: Program to Find the Square Root of a Number

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

int main() {
    double number, squareRoot;

    number = 25.0;

    // sqrt() is a library function to calculate the square root
    squareRoot = sqrt(number);

    cout << "Square root of " << number << " = " << squareRoot;

    return 0;
}
```

Output

```
Square root of 25 = 5
```

Continue.....

In this program, the `sqrt()` library function is used to calculate the square root of a number.

The function declaration of `sqrt()` is defined in the `cmath` header file. That's why we need to use the code `#include <cmath>` to use the `sqrt()` function.

Calling Functions

- Three ways to pass arguments to function
 1. Pass-by-value
 2. Pass-by-reference with reference arguments
 3. Pass-by-reference with pointer arguments

1. Pass by value – Example

```
void func (int num)
{
    cout<<"num = "<<num<<endl;
    num = 10;
    cout<<"num = "<<num<<endl;
}

void main()
{
    int n = 5;
    cout<<"Before function call: n = "<<n<<endl;
    func(n);
    cout<<"After function call: n = "<<n<<endl;
}
```

1. Pass by value – Example

```
#include <iostream>
void passByValue(int x) {
    x = 10;
    std::cout << "Inside passByValue function: " << x << std::endl;
}

int main() {
    int val = 5;
    std::cout << "Before passByValue function: " << val << std::endl;
    passByValue(val);
    std::cout << "After passByValue function: " << val << std::endl;
    return 0;
}
```

Output 5,10,5

1. Pass by Reference– Example

```
#include <iostream>

void passByReference(int &x) {
    x = 10;
    std::cout << "Inside passByReference function: " << x << std::endl;
}

int main() {
    int val = 5;
    std::cout << "Before passByReference function: " << val << std::endl;
    passByReference(val);
    std::cout << "After passByReference function: " << val << std::endl;

    return 0;
}

5,10,10
```

Practice Questions

Define two functions to print the maximum and the minimum number respectively among three numbers entered by user.

Write a program to print the circumference and area of a circle of radius entered by user by defining your own function.

A person is eligible to vote if his/her age is greater than or equal to 18. Define a function to find out if he/she is eligible to vote.

C++ Pass by Value vs. Pass by Reference

- In above examples, we learned about passing arguments to a function.
- This method used is called passing by value because the actual value is passed.
- However, there is another way of passing arguments to a function where the actual values of arguments are not passed. Instead, the reference to values is passed.

Using Reference Variables with Functions

- To create a **second name** for a **variable** in a program, you can generate an **alias**, or an **alternate name**
- In **C++** a **variable** that acts as an **alias** for another **variable** is called a **reference variable**, or simply a **reference**
- **Arguments** passed to function using reference arguments:
 - ***Modify original values of arguments***

2. Pass By Reference

- You can use a **reference variable** as a **parameter** in a **function** and **pass a regular variable** to invoke the function.
- The **parameter** becomes an **alias** for the **original variable**. This is known as *pass-by-reference*.
 - When you **change** the **value** through the **reference variable**, the **original value** is actually changed.

2. Pass by Reference – Example

```
void func(int &num)
```

```
{
```

```
    cout<<"num = "<<num<<endl;
```

```
    num = 10;
```

```
    cout<<"num = "<<num<<endl;
```

```
}
```

```
void main()
```

```
{
```

```
    int n = 5;
```

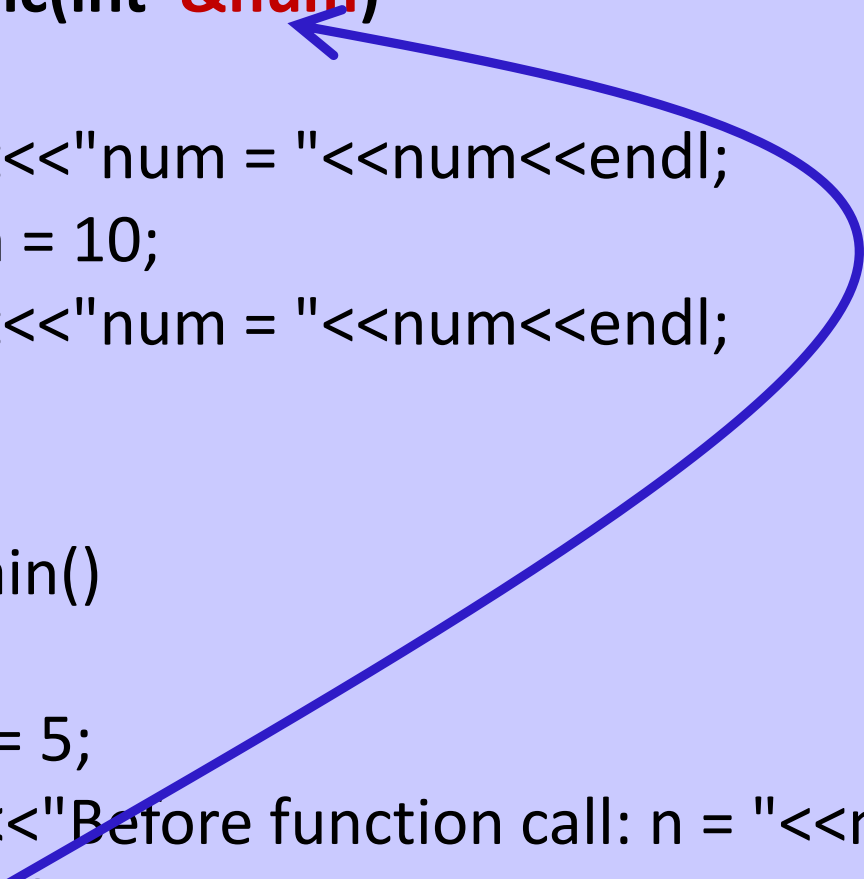
```
    cout<<"Before function call: n = "<<n<<endl;
```

```
    func(n);
```

```
    cout<<"After function call: n = "<<n<<endl;
```

```
}
```

num refers to variable
n (in main function)



C++ Pass by Value vs. Pass by Reference

```
// function that takes value as parameter

void func1(int numVal) {
    // code
}

// function that takes reference as parameter
// notice the & before the parameter
void func2(int &numRef) {
    // code
}

int main() {
    int num = 5;

    // pass by value
    func1(num);

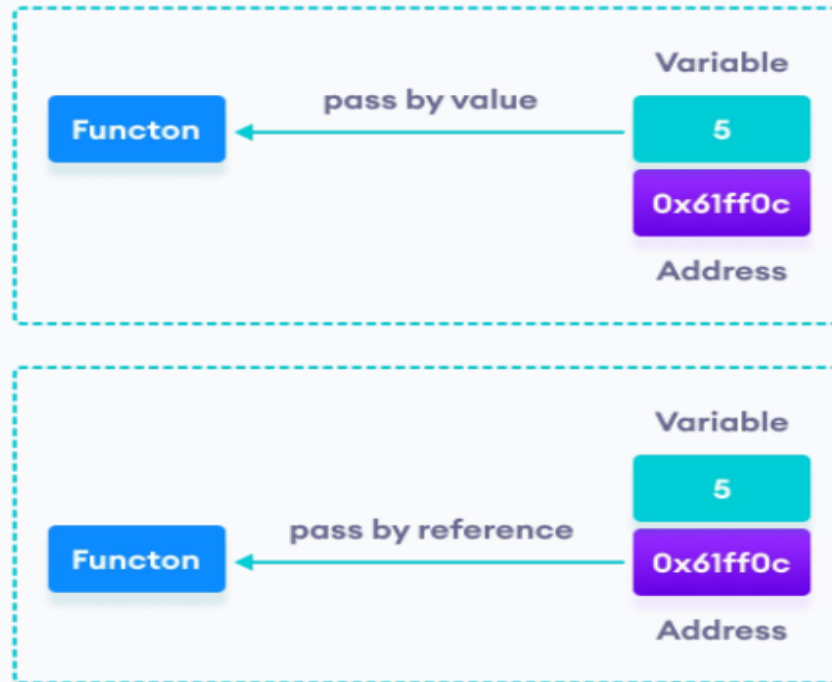
    // pass by reference
    func2(num);

    return 0;
}
```

C++ Pass by Value vs. Pass by Reference

Notice the `&` in `void func2(int &numRef)`. This denotes that we are using the address of the variable as our parameter.

So, when we call the `func2()` function in `main()` by passing the variable `num` as an argument, we are actually passing the address of `num` variable instead of the value `5`.



2. Pass by Reference - Example

- Swap two variable using a Pass-By reference

Example 6: Passing by reference

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

// function definition to swap values
void swap(int &n1, int &n2) {
```

Output

Before swapping

a = 1

b = 2

After swapping

a = 2

b = 1

```
    cout << "b = " << b << endl;
```

```
    // call function to swap numbers
```

```
    swap(a, b);
```

```
    cout << "\nAfter swapping" << endl;
```

```
    cout << "a = " << a << endl;
```

```
    cout << "b = " << b << endl;
```

```
    return 0;
```

```
}
```

Continue....

In this program, we passed the variables `a` and `b` to the `swap()` function. Notice the function definition,

```
void swap(int &n1, int &n2)
```

Here, we are using `&` to denote that the function will accept addresses as its parameters.

Hence, the compiler can identify that instead of actual values, the reference of the variables is passed to function parameters.

In the `swap()` function, the function parameters `n1` and `n2` are pointing to the same value as the variables `a` and `b` respectively. Hence the swapping takes place on actual value.

Passing Array to a Function in C++ Programming

- In this Lecture, we will learn how to pass a single-dimensional(1d) array as a function parameter in C++ with the help of examples.
- In C++, we can pass arrays as an argument to a function.
- Before you learn about passing arrays as a function argument, make sure you know about [C++ Arrays](#) and [C++ Functions](#).

Passing an Array to a Function

- When passing an array to a function, we need to tell the compiler what the type of the array is and give it a variable name, similar to an array declaration

`float a[]`

This would be a formal parameter

- We don't want to specify the size so function can work with different sized arrays.
- Size comes in as a second parameter

Syntax for Passing Arrays as Function Parameters

- The syntax for passing an array to a function is:

```
returnType functionName(dataType arrayName[arraySize]) {  
    // code  
}
```



Optional

Let's see an example,

```
int total(int marks[5]) {  
    // code  
}
```

Here, we have passed an `int` type array named `marks` to the function `total()`. The size of the array is `5`.

Following is a simple example to show how arrays are typically passed in C++

Passing array to function in C

Pointer a takes the base address of array arr

```
void func( int a[] , int size )  
{  
  
}  
  
int main( )  
{  
    int n=5;  
    int arr[5] = { 1, 2, 3, 4, 5 };  
    func( arr , n);  
    return 0;  
}
```

The length of arr is passed. It is compulsory to pass size as is just a pointer

Example 1: Passing Array to a Function

```
// C++ Program to display marks of 5 students
```

Output

```
Displaying marks:
```

```
Student 1: 88
```

```
Student 2: 76
```

```
Student 3: 90
```

```
Student 4: 61
```

```
Student 5: 69
```

```
// declare and initialize an array  
int marks[5] = {88, 76, 90, 61, 69};
```

```
// call display function  
// pass array as argument  
display(marks);
```

```
return 0;
```

```
}
```

Cont.....

Here,

1. When we call a function by passing an array as the argument, only the name of the array is used.

```
display(marks);
```

Here, the argument `marks` represent the memory address of the first element of array `marks[5]`.

2. However, notice the parameter of the `display()` function.

```
void display(int m[5])
```

Here, we use the full declaration of the array in the function parameter, including the square braces `[]`.

3. The function parameter `int m[5]` converts to `int* m;`. This points to the same address pointed by the array `marks`. This means that when we manipulate `m[5]` in the function body, we are actually manipulating the original array `marks`.

C++ handles passing an array to a function in this way to save memory and time.

Passing an Array to a Function

```
int Display(int data[], int N)
{ int k;
  cout<<"Array contains"<<endl;
  for (k=0; k<N; k++)
    cout<<data[k]<<" ";
  cout<<endl;
}
```

An int array
parameter
of unknown size

The size of
the array

```
int main()
{
  int a[4] = { 11, 33, 55, 77 };
  Display(a, 4);
}
```

The array
argument, no []

Passing an Array to a Function

```
#include <iostream.h>
int sum(int data[], int n); //PROTOTYPE
```

```
void main()
{
    int a[] = { 11, 33, 55, 77 };
    int size = sizeof(a)/sizeof(int);
    cout << "sum(a,size) = " << sum(a,size) << endl;
}
```

The array argument, no []



```
int sum(int data[], int n)
{
    int sum=0;
    for (int i=0; i<n;i++)
        sum += data[i];
    return sum;
}
```

An int array
parameter
of unknown size



The size of the array



Arrays are always Pass By Reference

- Arrays are automatically passed by reference.
- Do not use &.
- If the function modifies the array, it is also modified in the calling environment.

//Following function sets the values of an array to 0

```
void Zero(int arr[], int N)
{
    for (int k=0; k<N; k++)
        arr[k]=0;
}
```

Example 2: Print minimum number

```
#include <iostream>
using namespace std;
void printMin(int arr[5]);
int main()
{
    int arr1[5] = { 30, 10, 20, 40, 50 };
    printMin(arr1, 5);           //passing array to function
}
void printMin(int arr[5], int size)
{
    int min = arr[0];
    for (int i = 0; i < size; i++)
    {
        if (min > arr[i])
        {
            min = arr[i];
        }
    }
    cout<< "Minimum element is: "<< min <<"\n";
}
```

Output:

```
Minimum element is: 10
```

Function Overloading

- Function overloading
 - Functions with same name and different parameters
 - Should perform similar tasks:
 - i.e., function to square **ints** and function to square **floats**

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

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

Function Overloading

- At call-time C++ compiler selects the proper function by examining the number, type and order of the parameters

Function Overloading

```
void print(int i)
{ cout << " Here is int " << i << endl; }
```

```
void print(double f)
{ cout << " Here is float " << f << endl; }
```

```
void print(char* c)
{ cout << " Here is char* " << c << endl; }
```

```
int main()
{ print(10); print(10.10); print("ten"); }
```

Class Exercise 1 - Find the output

```
void main()
{
    int x = 20, y = 15;
    int z = Test(x, y);
    cout<<x<<" "<<y<<" "<<z;
}
```

```
int Test(int &a, int b)
{
    if(a >= 5)
    {
        a++;
        b--;
        return b;
    }
    else
    {
        --b;
        --a;
        return b;
    }
}
```

Class Exercise-2

- Write a program that calculates the area of a rectangle (width*length). The program should be based on the following functions:
 - int getLength()
 - int getWidth()
 - int CalculateArea()
 - void DisplayArea()

Class Exercise-3

- Write a C++ program that has a function called **zeroSmaller()** that is passed two int *arguments by reference* and then sets the smaller of the two numbers to 0.

Class Exercise-4

- Write a function called `swap()` that interchanges two int values passed to it by the calling program. (Note that this function swaps the values of the variables in the calling program, i.e., the original. You'll need to decide how to pass the arguments. Create a `main()` program to exercise the function.

Class Exercise-5

- Write a function that, when you call it, displays a message telling how many times it has been called: “I have been called 3 times”, for instance. Write a main() program that ask the user to call the function, if the user presses ‘y’ the function is called otherwise if ‘n’ is pressed the program terminates.