Programming Fundamentals

LECTURE 03: FUNCTIONS

BY: ZUPASH AWAIS

WEEK 03

Memory

- Each variable uses space on the computer's memory to store its value.
- When we use the term **allocate**, we indicate that the variable has been given a space on the computer's memory.
- **Deallocations** means the space has been reclaimed by computer and the variable cannot be accessed now.
- Memory is divided into two parts. The first is called **Stack memory** and other is **Heap memory**.

Static and Dynamic Memory

Memory allocation in C++ is done by two methods.

- One of them is ***Static Memory Allocation*** which is also called as Compile Time Allocation.

- And the other one is called as ***Dynamic Memory Allocation*** which is also know as Run Time Allocation.

Stack and Heap Memory

- **Stack** memory *store variables declared inside function call and is generally smaller than heap memory.*

- **Heap** memory *is used in dynamic memory allocation and is generally larger than stack memory.*

Modular Programming

- **Modular programming:** breaking a program up into smaller, manageable functions or modules
- **Function:** a collection of statements to perform a task
- Motivation for modular programming:
  - Improves maintainability of programs
  - Simplifies the process of writing programs
  - Make part of program reusable

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;
}
```

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

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

main function

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

function 2

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

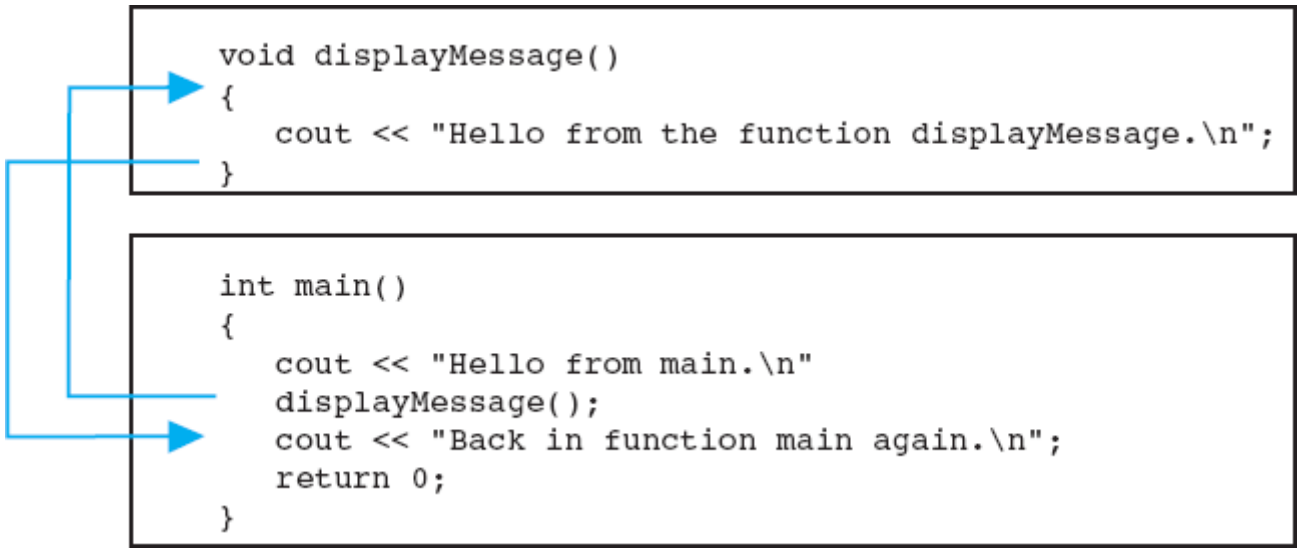
function 3

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

function 4

Defining and Calling Functions

- **Function call:** statement causes a function to execute
- **Function definition:** statements that make up a function



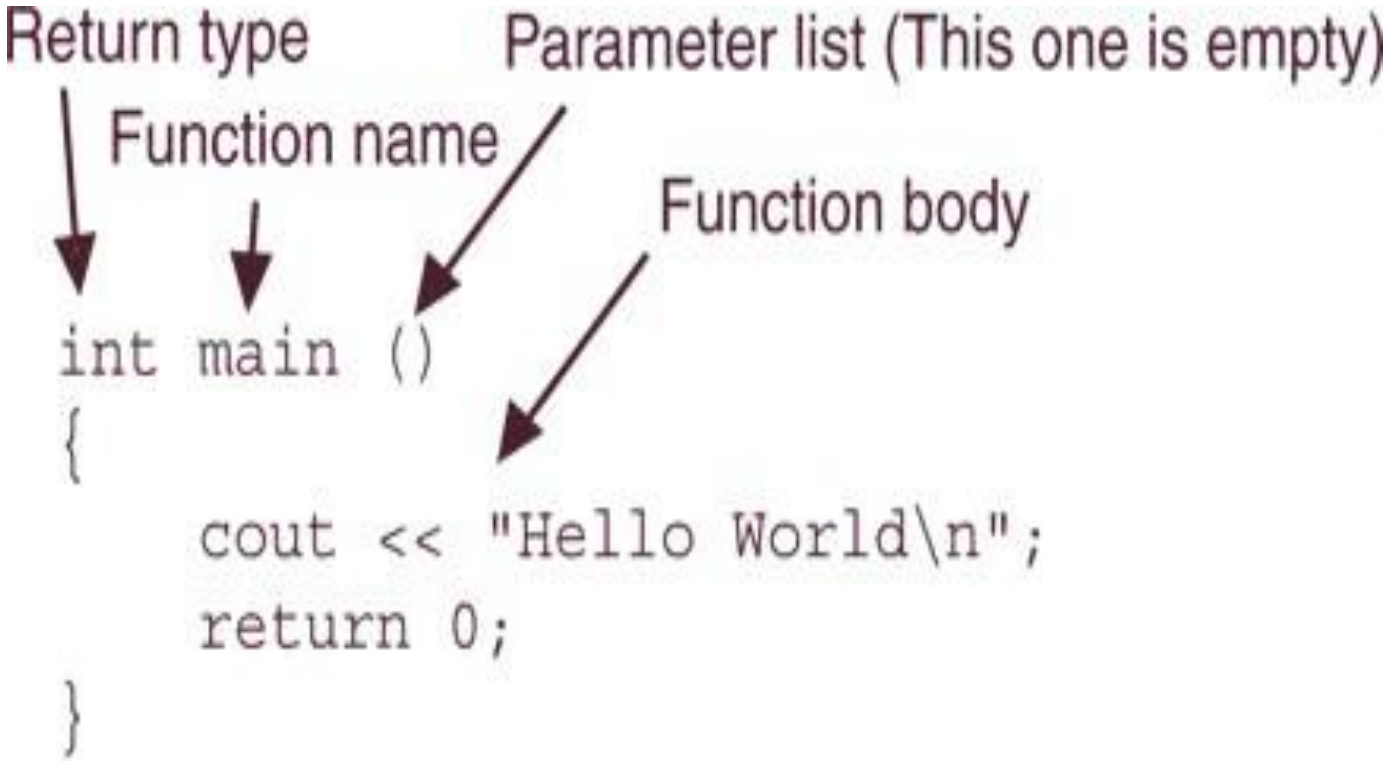
```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}

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

Function Definition

- Definition includes:
  - **return type:** data type of the value that function returns to the part of the program that called it
  - **name:** name of the function. Function names follow same rules as variables
  - **parameter list:** variables containing values passed to the function
  - **body:** statements that perform the function's task, enclosed in {}

Function Definition

Note: The line that reads `int main()`is the *function header*.



Function Return Type

- If a function returns a value, the type of the value must be indicated:

```
int main()
```

- If a function does not return a value, its return type is `void`:

```
void printHeading()
{
      cout << "Monthly Sales\n";
}
```

Calling a Function

- To call a function, use the function name followed by `()` and `;`

```
printHeading();
```

- When called, program executes the body of the called function

- After the function terminates, execution resumes in the calling function at point of call.

Example

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

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

int main()
{
	// calling the function
	display();
	return 0;
}
```

Calling Functions

- main() can call any number of functions
- Functions can call other functions
- Compiler must know the following about a function before it is called:
  - ✓ name
  - ✓ return type
  - ✓ number of parameters
  - ✓ data type of each parameter

Example

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

void display1()
{
        cout << "Hello";
}

void display2()
{
        display1();
        cout << “ World!";
}

int main()
{
        // calling the function
        display2();
        return 0;
}
```

Function Prototypes

Ways to notify the compiler about a function before a call to the function:

- Place function definition before calling function's definition
- Use a function prototype (function declaration) – like the function definition without the body

Header: `void printHeading()`
Prototype: `void printHeading();`

Function Prototypes Notes

- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- When using prototypes, can place function definitions in any order in source file

Example

```
#include <iostream>
using namespace std;
// declaring a function
prototype void display();
// declaring a function
void display()
{
	cout << "Hello!";
}

int main()
{
	// calling the function
	display();
	return 0;
}
```

Sending Data into a Function

- Can pass values into a function at time of call:
  c = pow(a, b);

- Values passed to function are arguments

- Variables in a function that hold the values passed as arguments are parameters

A Function with a Parameter Variable

```
void displayValue(int num)
{
   cout << "The value is " << num << endl;
}
```

- The integer variable `num` is a parameter.
- It accepts any integer value passed to the function.

Parameters, Prototypes, and Function Headers

- For each function argument,
  - ✓ the prototype must include the data type of each parameter inside its parentheses
  - ✓ the header must include a declaration for each parameter in its ()

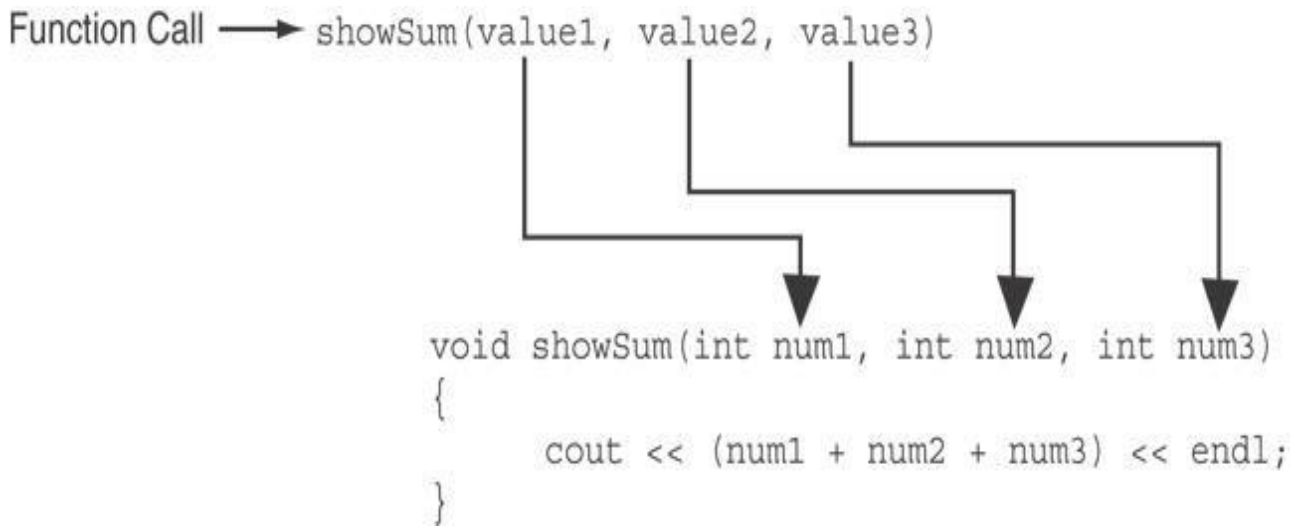
```
void evenOrOdd(int);  //prototype
void evenOrOdd(int num) //header
evenOrOdd(val);       //call
```

Function Call Notes

- Value of argument is copied into parameter when the function is called
- A parameter's scope is the function which uses it
- Function can have multiple parameters
- There must be a data type listed in the prototype () and an argument declaration in the function header () for each parameter
- Arguments will be promoted/demoted as necessary to match parameters

Passing Multiple Arguments

- When calling a function and passing multiple arguments:
- the number of arguments in the call must match the prototype and definition
- the first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.



Activity

- Write a C++ program to design a calculator. Make a function for each given operation:

- Sum
- Subtract
- Multiply
- Divide
- Percentage

Overloading Functions

- Overloaded functions have the same name but different parameter lists
- Can be used to create functions that perform the same task but take different parameter types or different number of parameters
- Compiler will determine which version of function to call by argument and parameter lists

Overloading Functions

- Using these overloaded functions,

```
void getDimensions(int);            // 1
void getDimensions(int, int);       // 2
void getDimensions(int, double);    // 3
void getDimensions(double, double);// 4
```

- the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);              // 1
getDimensions(length, width);       // 2
getDimensions(length, height);      // 3
getDimensions(height, base);        // 4
```

Default Values in Function (Default Arguments)

A *Default argument* is an argument that is passed automatically to a parameter if the argument is missing on the function call.

- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype

- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

Default Values in Function (Default Arguments)

If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0);// OK
int getSum(int, int=0, int);  // NO
```

When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2);     // OK
sum = getSum(num1, , num3);   // NO
```

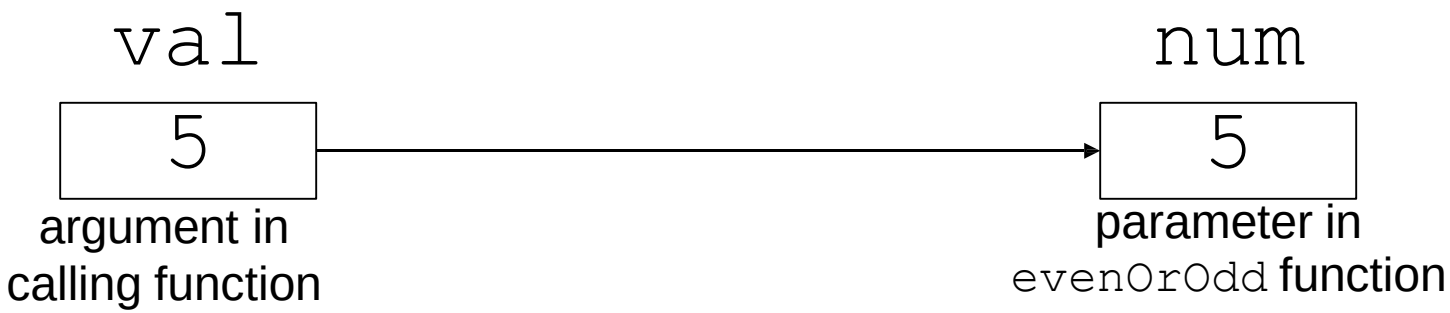
Types of Function Calls

- Call by Value [Pass by value]
- Call by Reference (& operator) [Pass by reference]

Passing Data by Value

- Pass by value: when an argument is passed to a function, its value is copied into the parameter.
- Changes to the parameter in the function do not affect the value of the argument
- Example: `int val=5;`

```
evenOrOdd(val);
```



- `evenOrOdd` can change variable `num`, but it will have no effect on variable `val`

& Operator (Reference Operator)

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting ‘&’ in the declaration.

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

int main()
{
  int x = 10;

  // ref is a reference to x.
  int& ref = x;

  // Value of x is now changed to 20
  ref = 20;
  cout << "x = " << x << endl ;

  // Value of x is now changed to 30
  x = 30;
  cout << "ref = " << ref << endl ;

  return 0;
}
```

Using Reference Variables as Parameters

- A mechanism that allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to ‘return’ more than one value

Passing by Reference

- A reference variable is an alias for another variable
- Defined with an ampersand (&)

```
void getDimensions(int&, int&);
```

- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing parameters by reference

Example

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

void func(int& a)
{
   a = 30;
}

int main()
{
   int x = 10;
   cou  << "Value of x: " << x;
   func(x);
   cout << "\nValue of x: " << x;
}
```

Activity

- Write a C++ program to swap two numbers using function pass by reference.