IMAM ABDULRAHMAN BIN FAISAL UNIVERSITY
College of Computer Science & IT
Department of CS

Welcome to CS 221:Fundamentals of Programming

Weeks (5): Functions – Par 1

Chapters 9 and 10 (Zak Textbook)

Chapter 4 (Savitch Textbook)

Objectives

- Discover the Top-Down design and its relationship to functions
- Explain what is functions in C++, and type of functions.
- Introduce standard pre-defined functions and discover how to use them in a program.

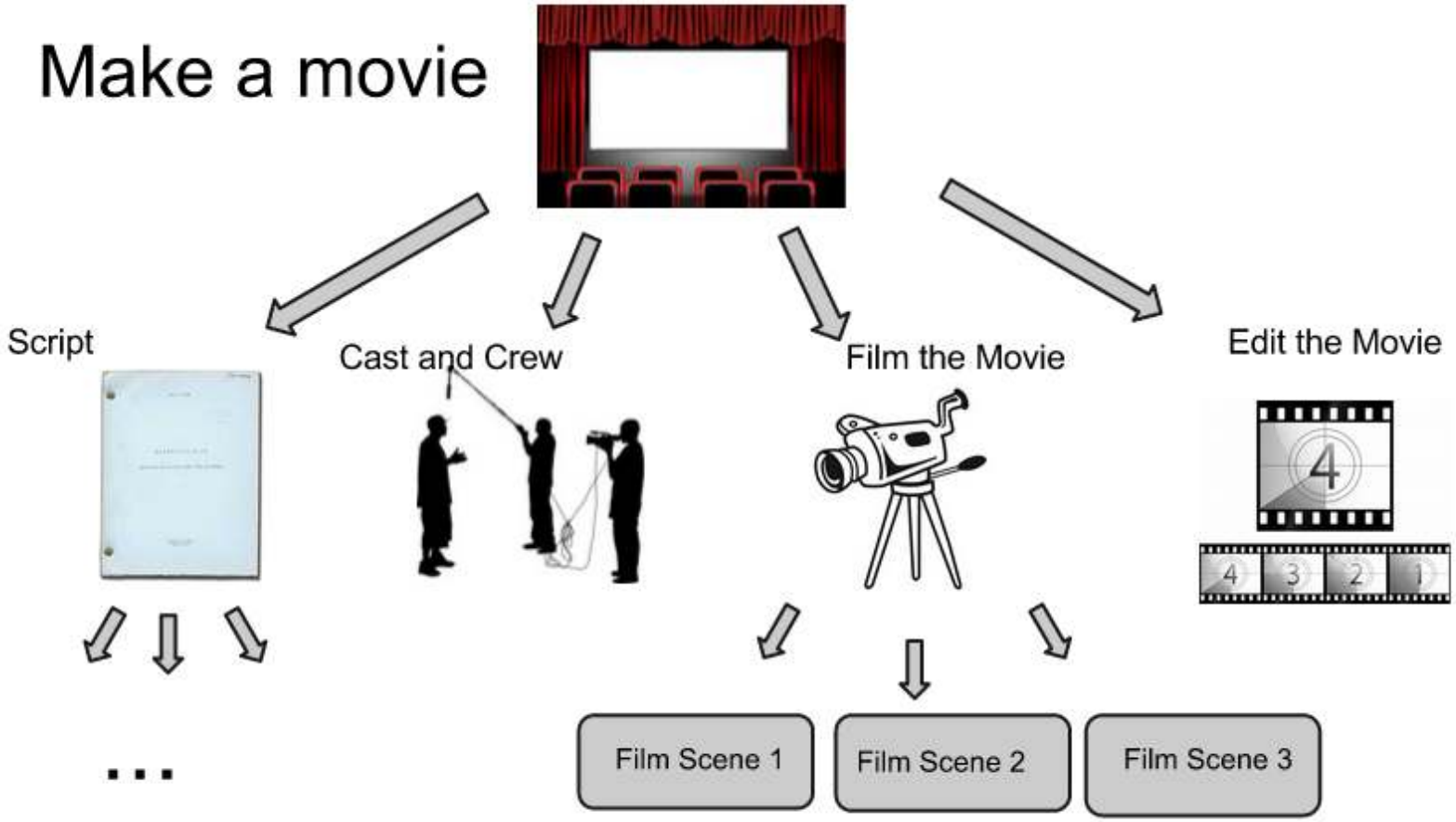
TOP-DOWN DESIGN

Top Down Design

- **To write a program**
  - Develop the algorithm that the program will use
  - Translate the algorithm into the programming language

- **Top Down Design**
  (also called stepwise refinement)
  - Break the algorithm into subtasks
  - Break each subtask into smaller subtasks
  - Eventually the smaller subtasks are trivial to implement in the programming language

Top Down Design
Example: Make Movie



Top Down Design
Example: Main Function in program

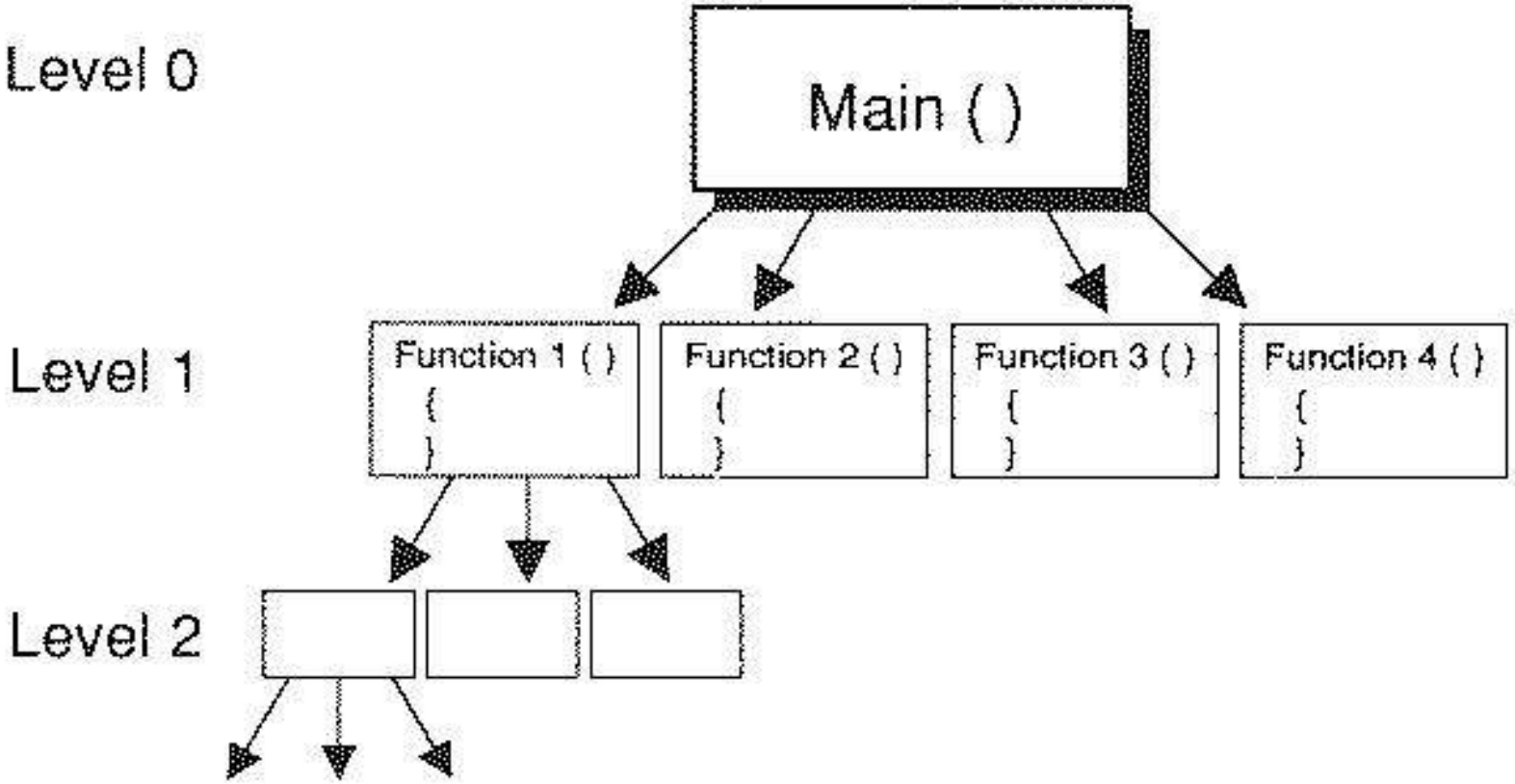


Figure 7.1. C Programs are built from functions.

Benefits of Top Down Design

- Subtasks, or functions in C++, make programs
  - Easier to understand
  - Easier to change
  - Easier to write
  - Easier to test
  - Easier to debug
  - Easier for teams to develop

FUNCTIONS

Motivation

- Suppose that you need to find the sum of integers from **1** to **10**, from **20** to **30**, and from **35** to **45**, respectively.
- So, how do you solve this problem?

```
int sum = 0;
for (int i = 1; i <= 10; i++)  sum += i;
cout << " Sum from 1            to 10 is " << sum ;

sum = 0;
for (int i = 20; i <= 30; i++)  sum += i;
cout << " Sum from 20 to 30 is " << sum;

sum = 0;
for (int i = 35; i <= 45; i++)  sum += i;
cout << "Sum from 35 to 45 is " << sum ;
```

What is Function?

- Functions are like building blocks
- They allow complicated programs to be divided into manageable pieces
- Some advantages of functions:
  - A programmer can focus on just that part of the program and construct it, debug it, and perfect it.
  - Different people can work on different functions simultaneously.
  - Can be re-used (even in different programs).
  - Enhance program readability.

What is Function? (continued)

- Some functions are built-in functions (part of C++): defined in language libraries
- Others, called programmer-defined functions, are written by programmers; defined in a program

- Typically, `main` is used to call other functions, but any function can call any other function.
- Functions
  - Called modules .
  - Like small programs.
  - Can be put together to form a larger program.

Function Syntax: Header and Definition

- Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

- **`functionType`** is also called the data type or return type
  - The function can be of type void, means it does not return any value:
    - Example: `void printName(string name)`
  - Or can be a value-retuned type, where it can returns value of the type specified by the functionType
    - Example: `int FindMax(int x, int y)`

Function Syntax: Header and Definition

- Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

- **Formal parameters** are written in the function prototype and function header of the definition. Formal parameters are local variables which are assigned values from the arguments when the function is called.

```
dataType identifier, dataType identifier, ...
```

- **Formal parameter** list can be empty:

```
functionType functionName()
```

Function Syntax: Function Call

```
functionName(actual parameter list)
```

Actual parameter:

- When a function is *called*, the values (expressions) that are passed in the call are called the *arguments* or *actual parameters* (both terms mean the same thing).
- At the time of the call **each actual parameter is assigned to the corresponding formal parameter** in the function definition.

- The syntax of the actual parameter list is:

```
expression or variable, expression or variable, ...
```

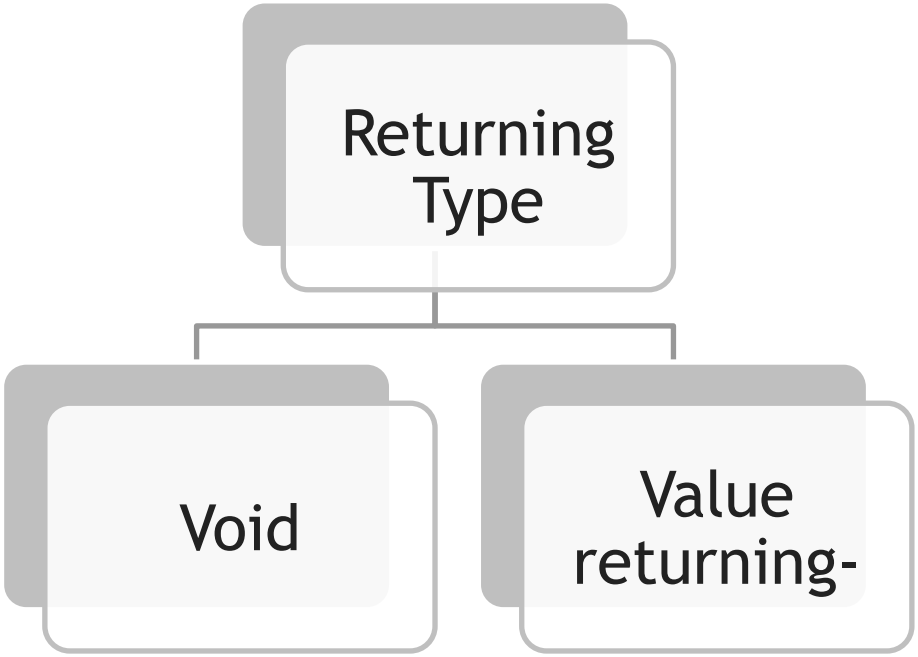
- A call to a function with an empty actual parameter list is:

```
functionName()
```

Function Syntax (Example)

- Heading: (Function Type, Function Name, Formal Parameters)
  - Example: `int abs(int number)`
- Formal Parameter: variable(s) declared in the heading
  - Example: `number // in the heading above`
- Actual Parameter: variable(s) or expression(s) listed in a call to a function
  - Example:
    - `x = pow(u, v) // u and v are actual parameters`
    - `int z = abs(-5) // -5 is actual parameter`

Functions Type OR Returning Type




```
functionType functionName(formal parameter list)
{
    statements
}
```

Functions Type OR Returning Type

- Value-returning functions: have a return type
  - Return a value of a specific data type using the `return` statement

- Void functions: do not have a return type
  - *Do not* use a `return` statement to return a value

Void Functions

- Void functions and value-returning functions have similar structures
  - Both have a heading part and a statement part

```
void functionName()
{
    statements
}
```

- A void function does not have a return type
  - `return` statement without any value is typically used to exit the function early
- `void` is a reserved word
- A call to a void function is a stand-alone statement

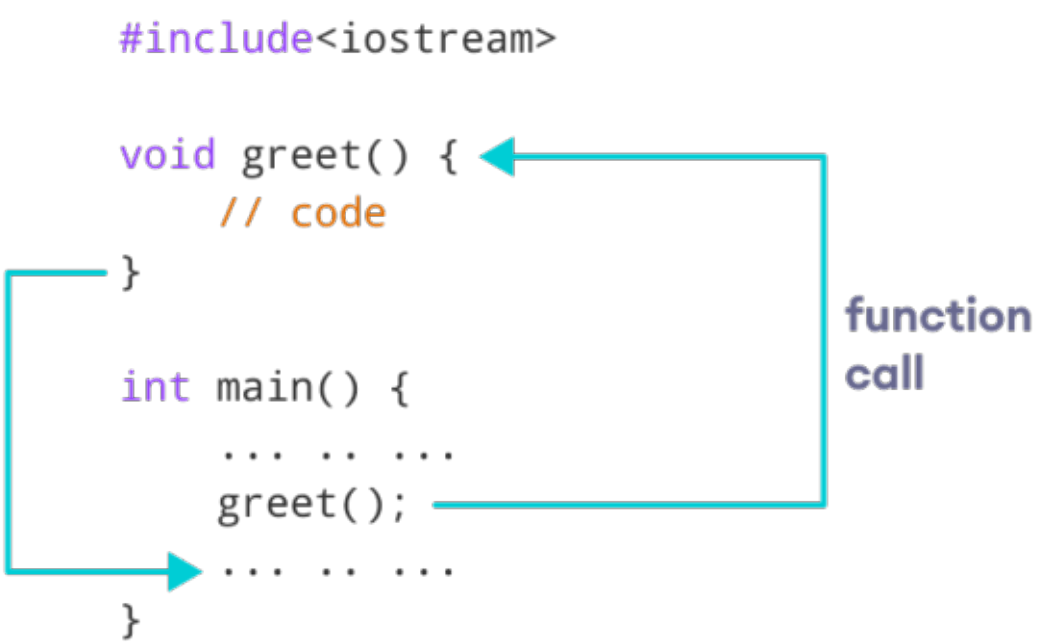
```
functionName();
```

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

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

int main() {
   // calling the function
   greet();

   return 0;
}
```



Void Functions with Parameters

- Function definition syntax:

```
void functionName(formal parameter list)
{
     statements
}
```

```
dataType& variable, dataType& variable, ...
```

```
functionName(actual parameter list);
```

```
expression or variable,expression or variable, ...
```

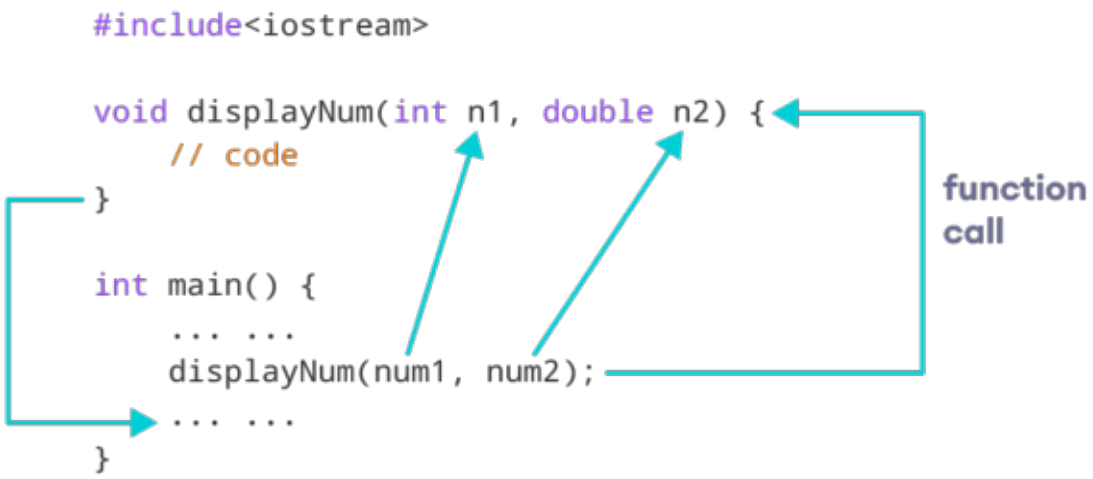
program to print a text

```
#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;}
```



Value-Returning Functions

- To use these functions you must:
  - if the function is pre-defined, then Include the appropriate header file in your program using the `#include` statement,
  - Know the following items:
    - Name of the function
    - Number of parameters, if any
    - Data type of each parameter
    - Data type of the value returned: called the type of the function

program to add two numbers using a function and Return the value

```
#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;  }
```

Value-Returning Functions (continued)

- Because the value returned by a value-returning function is unique, we must:
  - Save the value for further calculation
  - Use the value in some calculation
  - Print the value
- A value-returning function is used in an assignment or in an output statement

`return` Statement

- Once a value-returning function computes the value, the function returns this value via the `return` statement
  - It passes this value outside the function via the `return` statement

Syntax: `return` Statement

- The `return` statement has the following syntax:

```
return expr;
```

- Expression can be any expression that must evaluate to something compatible with the **returnValueType**
- When a return statement executes
  - Function immediately terminates
  - Control goes back to the caller
- When a `return` statement executes in the function `main`, the program terminates

Syntax: `return` Statement

The *body of a method* that **returns a value** must contain one or more return statements

A **void method** does not require a return statement, unless there is a situation that requires the method to end before all its code is executed.

In this context, since it does not return a value, a return statement is used without an expression:

```
return;
```

Value-Returning Functions (continued)

```
int abs(int number)
```

```
int abs(int number)
{
    if (number < 0)
        number = -number;

    return number;
}
```

```
double pow(double base, double exponent)
```

```
double u = 2.5;
double v = 3.0;
double x, y, w;

x = pow(u, v);                  //Line 1
y = pow(2.0, 3.2);              //Line 2
w = pow(u, 7);                  //Line 3
```

Does the following functions do the same thing?

```
double larger(double x, double y)
{
    double max;

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}
```

You can also write this function as follows:

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

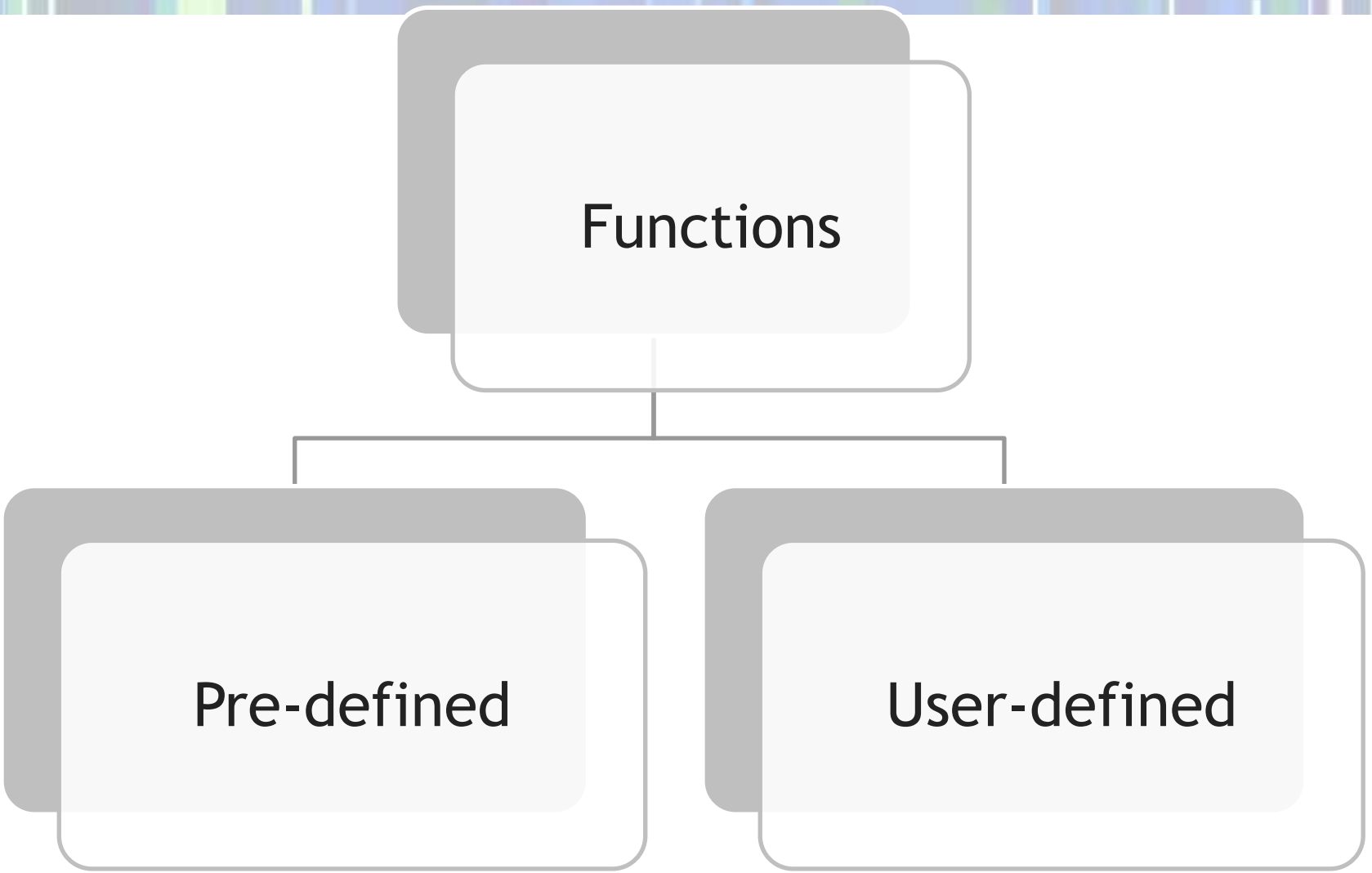
```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```

NOTE

1. In the definition of the function `larger`, `x` and `y` are formal parameters.
2. The `return` statement can appear anywhere in the function. Recall that once a `return` statement executes, all subsequent statements are skipped. Thus, it's a good idea to return the value as soon as it is computed.

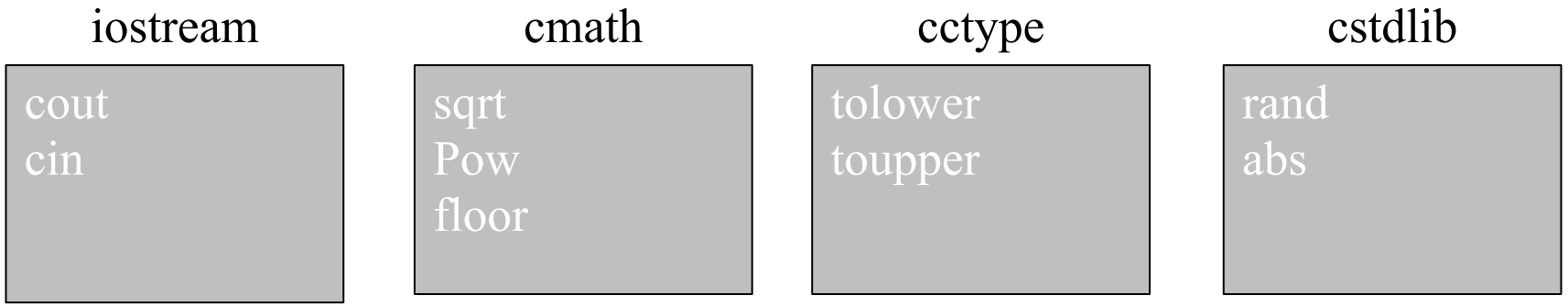
Types of Functions



PRE-DEFINED FUNCTIONS

Function Libraries

- Predefined functions are found in libraries
- The library must be “included” in a program to make the functions available
- An include directive tells the compiler which library header file to include.
  - You cannot use functions without calling its header.
  - In case you call function without includes its header, linking error will appear in compilation time.



Includes Predefined Functions

- To include the math library containing `sqrt()`:

```
#include <cmath>
```

- Newer standard libraries, such as `cmath`, also require the directive

```
using namespace std;
```

Function Call Syntax

- Function_name (Argument_List)
  - Argument_List is a comma separated list:

    (Argument_1, Argument_2, … , Argument_Last)
- Example:
  - `side = sqrt(area);`
  - `cout << "2.5 to the power 3.0 is " << pow(2.5, 3.0);`

Predefined Functions (continued)

- **`pow(x,y)`** calculates x^y
  - `x` and `y` are the parameters (or arguments)
  - Returns a value of type `double`
  - `pow(2, 3) = 8.0`
  - The function has two parameters : **2** and **3**

- **`sqrt(x)`** calculates the nonnegative square root of **`x`**, for `x >= 0.0`
  - `x` is the parameter (or argument)
  - Returns a value of type `double`
  - `sqrt(2.25)` is `1.5`
  - The function has only one parameter : **2.25**

Predefined Functions (continued)

- **`floor(x)`** calculates largest whole number not greater than **`x`**
  - `x` is the parameter (or argument)
  - Returns a value of type `double`
  - `floor(48.79)` is `48.0`
  - The function has only one parameter: **48.79**

Predefined Functions (continued)

TABLE 6-1 Predefined Functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than x: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle x: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where e = 2.718: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Predefined Functions (continued)

TABLE 6-1 Predefined Functions (continued)

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y; If `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, returns `x`	`int`	`int`

Predefined Functions (code)

EXAMPLE 6-1

```
//How to use predefined functions.
#include <iostream>
#include <cmath>
#include <cctype>
#include <cstdlib>

using namespace std;

int main()
{
    int    x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static_cast<char>(toupper('a'))
         << endl;                                          //Line 1

    u = 4.2;                                               //Line 2
    v = 3.0;                                               //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;               //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                           //Line 5

    u = u + pow(3.0, 3);                                   //Line 6
    cout << "Line 7: u = " << u << endl;                   //Line 7

    x = -15;                                               //Line 8
    cout << "Line 9: Absolute value of " << x
         << " = " << abs(x) << endl;                       //Line 9

    return 0;
}
```

Predefined Functions (code)

- Example 6-1 sample run:

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5: 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15
```

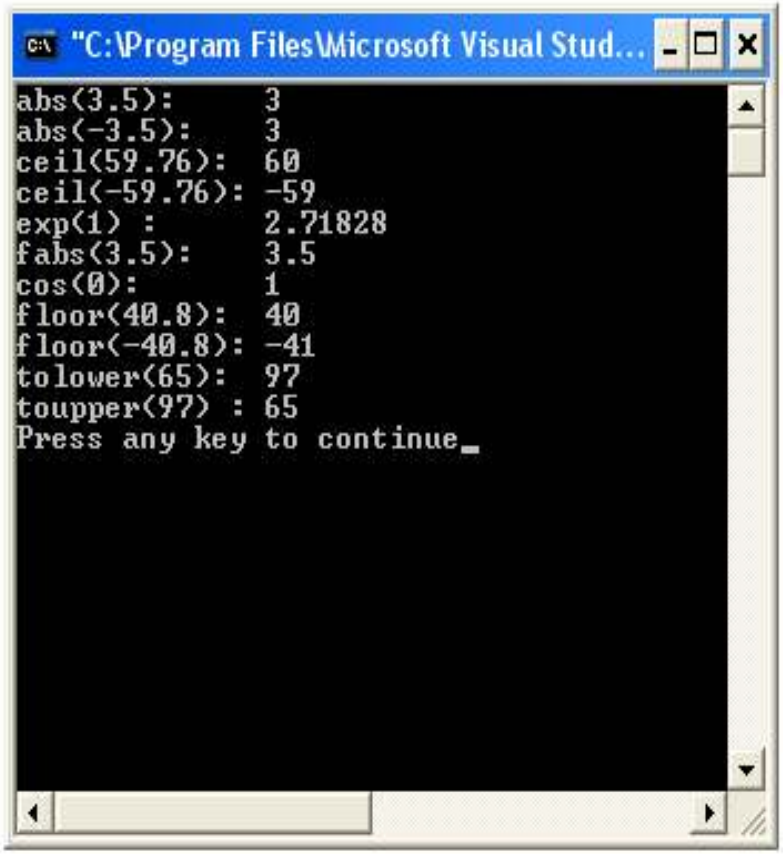
An Example Program

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<cctype>
using namespace std;
Int main()
{
cout<<"abs(3.5):     "<<abs(3.5)<<endl;
cout<<"abs(-3.5):    "<<abs(-3.5)<<endl;
cout<<"ceil(59.76):  "<<ceil(59.76)<<endl;
cout<<"ceil(-59.76): "<<ceil(-59.76)<<endl;
cout<<"exp(1) :      "<<exp(1)<<endl;
cout<<"fabs(3.5):    "<<fabs(3.5)<<endl;
cout<<"cos(0):       "<<cos(0)<<endl;
cout<<"floor(40.8):  "<<floor(40.8)<<endl;
cout<<"floor(-40.8): "<<floor(-40.8)<<endl;
cout<<"tolower(65):  "<<tolower(65)<<endl;
cout<<"toupper(97) : "<<toupper(97)<<endl;

Return 0;
}
```

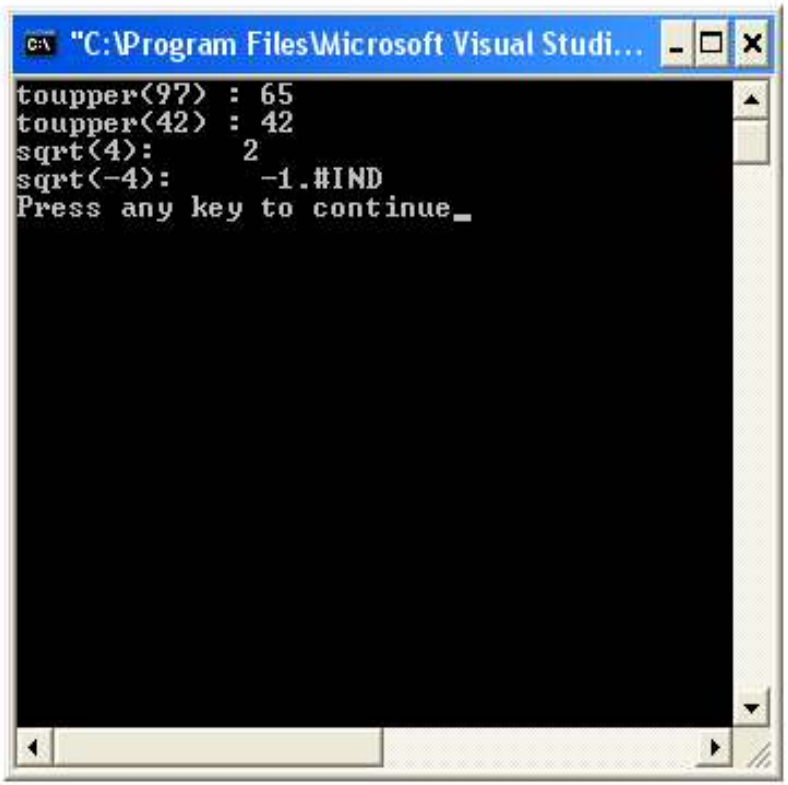
```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<cctype>
using namespace std;
void main()
{
cout<<"abs(3.5):      "<<abs(3.5)<<endl;
cout<<"abs(-3.5):     "<<abs(-3.5)<<endl;
cout<<"ceil(59.76):   "<<ceil(59.76)<<endl;
cout<<"ceil(-59.76):  "<<ceil(-59.76)<<endl;
cout<<"exp(1) :       "<<exp(1)<<endl;
cout<<"fabs(3.5):     "<<fabs(3.5)<<endl;
cout<<"cos(0):        "<<cos(0)<<endl;
cout<<"floor(40.8):   "<<floor(40.8)<<endl;
cout<<"floor(-40.8):  "<<floor(-40.8)<<endl;
cout<<"tolower(65):   "<<tolower(65)<<endl;
cout<<"toupper(97) :  "<<toupper(97)<<endl;

}
```



```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<cctype>
using namespace std;
void main()
{

cout<<"toupper(97) : "<<toupper(97)<<endl;
cout<<"toupper(42) : "<<toupper(42)<<endl;
cout<<"sqrt(4):      "<<sqrt(4)<<endl;
cout<<"sqrt(-4):     "<<sqrt(-4)<<endl;
}
```



Predefined Functions (example)

- **The `rand, srand,` and `time` Functions**

- **`Rand()`:** The function returns an integer that is greater than or equal to 0 but less than or equal to the value stored in the RAND_MAX constant, which is one of many constants built into the C++ language.

- **`RAND_MAX`**
  - Varies with different compilers:
    - Its value is always at least 32,767.
  - To display your compiler's RAND_MAX value:
    - cout << RAND_MAX;

Predefined Functions (example)

HOW TO Use the **rand** Function

Syntax

rand()

your compiler may require the `#include <cstdlib>` directive

Example 1

```
int randomNum = 0;
randomNum = rand();
```

The **rand** function generates a random integer that is greater than or equal to 0 but less than or equal to **RAND_MAX**. It then returns the random integer to the assignment statement, which assigns it to the **randomNum** variable.

Example 2

```
cout << rand();
```

The **rand** function generates a random integer that is greater than or equal to 0 but less than or equal to **RAND_MAX**. It then returns the random integer to the **cout** statement, which displays it on the computer screen.

Example 3

```
int tripleNum = 0;
tripleNum = rand() * 3;
```

The **rand** function generates a random integer that is greater than or equal to 0 but less than or equal to **RAND_MAX**. It then returns the random integer to the assignment statement, which multiplies it by 3 and assigns the result to the **tripleNum** variable.

Predefined Functions (example)

Generate Random value within a Specific range

HOW TO Generate Random Integers within a Specific Range

Syntax

lowerBound **+ rand() % (***upperBound* **–** *lowerBound* **+ 1)**

Example 1

```
cout << 1 + rand() % (6 - 1 + 1);
```

displays a random integer from 1 through 6 on the computer screen

Predefined Functions (example)

Generate Random value within a Specific range

Results using different **rand** values:

rand value: 27	`1 + 27 % (6 - 1 + 1)`
`6 - 1 + 1` is evaluated first and results in 6	`1 + 27 % 6`
`27 % 6` is evaluated next and results in 3	`1 + 3`
`1 + 3` is evaluated last and results in 4	`4`
rand value: 8	`1 + 8 % (6 - 1 + 1)`
`6 - 1 + 1` is evaluated first and results in 6	`1 + 8 % 6`
`8 % 6` is evaluated next and results in 2	`1 + 2`
`1 + 2` is evaluated last and results in 3	`3`
rand value: 324	`1 + 324 % (6 - 1 + 1)`
`6 - 1 + 1` is evaluated first and results in 6	`1 + 324 % 6`
`324 % 6` is evaluated next and results in 0	`1 + 0`
`1 + 0` is evaluated last and results in 1	`1`

Random Number Generator (code)

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int main()
{
cout << RAND_MAX;
srand(time(0));
int no;
for (int j=0; j<5; j++)
{
    no=rand();
    cout<<endl<<"No genrated: "<<no;
}}
```

Predefined Functions (example)

- You should initialize the random number generator in each program in which it is used. Otherwise, it will generate the same series of numbers each time the program is executed.

- Typically, the initialization task is performed at the beginning of the program. You initialize the generator using the **`srand`** function.

- A more common way to initialize the generator is to use the C++ **`time()`** function with **`srand()`** function.

Predefined Functions (example)

How To Use the **srand** Function

Syntax

srand(*seed***)**

your compiler may require the `#include <cstdlib>` directive

Example 1

```
int x = 0;
cout << "Enter an integer: ";
cin >> x;
srand(x);
cout << rand() << endl;
```

The `srand` function initializes the random number generator using the integer entered by the user. The `cout` statement displays a random integer that will be greater than or equal to 0 but less than or equal to `RAND_MAX`.

the `time` function requires the `#include <ctime>` directive

Example 2

```
srand(static_cast<int>(time(0)));
cout << rand() << endl;
```

The `srand` function initializes the random number generator using the value returned by the `time` function after it has been converted to the `int` data type. The `cout` statement displays a random integer that will be greater than or equal to 0 but less than or equal to `RAND_MAX`.

References

- https://www.w3schools.in/cplusplus-tutorial/functions/
- https://www.youtube.com/watch?v=j-idGfFDsAU
- https://www.tutorialspoint.com/cplusplus/cpp_functions.htm

Any Questions

- Thanks for Listening ☺️