

# **Functions**

1

## **Contents**



- Predefined functions
- User-defined functions
- Scope rules
- Parameters
- Function overloading
- Default arguments

fit@hcmus | Programming 1 | 2023



### **Functions**

- 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

fit@hcmus | Programming 1 | 2023



1

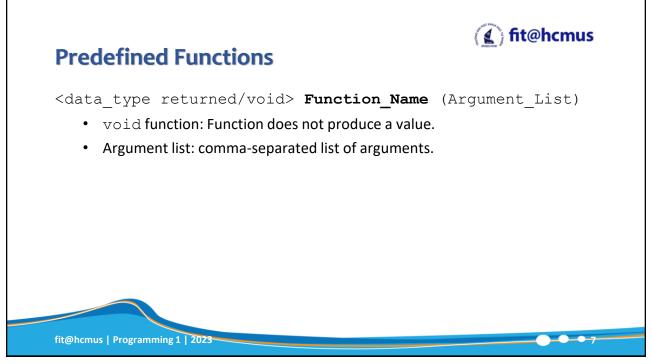
## **Functions**



- Other names:
  - Procedure
  - Subprogram
  - Method
- Types:
  - Pre-defined functions
  - User-defined (Programmer-defined) functions









### **Some Predefined Functions**

Some of the predefined mathematical functions are:

```
sqrt(x)
pow(x, y)
floor(x)
```

- Predefined functions are organized into separate libraries
- o I/O functions are in iostream header
- Math functions are in cmath header
- Some functions are in cstdlib header.

fit@hcmus | Programming 1 | 2023



Q

### **Some Predefined Functions**



- o pow(x,y) calculates  $x^y$ 
  - pow(2, 3) = 8.0
  - Returns a value of type double
  - x and y are the **parameters** (or **arguments**). The function has two parameters.
  - 8.0 is value returned.
- o sqrt(x) calculates the nonnegative square root of x, for x>=0.0
  - sqrt(2.25) is 1.5
  - Returns the value of type double





### **Some Predefined Functions**

- o The floor (x) function calculates largest whole number not greater than  ${\tt x}$ 
  - floor(48.79) is 48.0
  - Type double
  - Has only one parameter
- The abs (x), labs (x), fabs (x) functions calculate the absolute value of x (x is integer, long or floating-point number).

fit@hcmus | Programming 1 | 2023



10





### **Some Predefined Functions**

- o cos(x), cmath: return cosine of angle x.
- o tolower(c), cctype: return lowercase of c.
- o toupper(c), cctype: return UPPERCASE of c.

fit@hcmus | Programming 1 | 2023



12

## **void Predefined Functions**



- o exit(integer)
  - Library cstdlib
  - Program ends immediately.
  - argument value:
    - 1: caused by error
    - 0: other cases.

```
1 #include <iostream>
2 #include <cstdlib>
                                                   This is just a toy example. It
                                                   would produce the same
              espace std;
                                                   output if you omitted
 4
    int main( )
    {
 5
        cout << "Hello Out There!\n";
        exit(1);
        cout << "This statement is pointless, \n"
              << "because it will never be executed.\n"
             << "This is just a toy program to illustrate exit.\n";
10
11
        return 0;
```





### **User-defined Functions**

fit@hcmus | Programming 1 | 2023

14

#### fit@hcmus **Example** 1 #include <iostream> using namespace std; 3 double totalCost(int numberParameter, double priceParameter); //Computes the total cost, including 5% sales tax, //on numberParameter items at a cost of priceParameter each. Function declaration int main( ) also called the function { prototype double price, bill; int number; cout << "Enter the number of items purchased: ";</pre> cin >> number; 12 cout << "Enter the price per item \$"; 13 cin >> price; bill = totalCost(number, price); cout.setf(ios::fixed); 15 cout.setf(ios::showpoint); 16 19 20 << endl; 22 return 0; 23 } fit@hcmus | Programming 1 | 2023

#### 🚺 fit@hcmus **Example** cout << "Enter the number of items purchased: "; 10 11 cin >> number; 12 cout << "Enter the price per item \$";</pre> cin >> price; Function call bill = totalCost(number, price); 15 cout.setf(ios::fixed); 16 cout.setf(ios::showpoint); 17 cout.precision(2); cout << number << " items at " 18 << "\$" << price << " each.\n" 19 << "Final bill, including tax, is \$" << bill 20 22 return 0; 23 } 24 double totalCost(int numberParameter, double priceParameter) 25 { const double TAXRATE = 0.05; //5% sales tax 26 double subtotal: 27 subtotal = priceParameter \* numberParameter; return (subtotal + subtotal\*TAXRATE); fit@hcmus | Programming 1 | 2023





- Function declaration/Function prototype:
  - determines the kind of function
  - · tells the name of the functions
  - tells number and types of arguments
  - · ends with a semicolon.
- Function definition: describes how the function works.
  - **Function header:** same as function declaration (without semicolon at the end).
  - **Function body:** consists of declaration and executable statements enclosed within a pair of braces.

fit@hcmus | Programming 1 | 2023





## **Terminologies**

- Function call/Function invocation
- Formal argument (argument): variable declared in function header, or function prototype.
- Actual argument: variable or expression listed in a function call.

fit@hcmus | Programming 1 | 2023



18

### **Function Declaration**



Syntax:

Type\_Returned/void FunctionName(Parameter\_List);

- Parameter\_List can be empty (function with no arguments).
- Normally placed before the main function.





### return Statement

- Once a value-returning function computes the value, the function returns this value via the <u>return</u> statement
  - It passes this value outside the function via the return statement
- Syntax:

return expression;

fit@hcmus | Programming 1 | 2023



20

### return Statement



- When a return statement executes
  - Function immediately terminates
    - · Control goes back to the caller
- o A return statement in void function simply ends the function call.
  - void function needs not contain
- When a return statement executes in the function main, the program terminates.





## **Examples**

```
double larger(double x, double y)
    double max;
    if (x \ge y)
       max = x;
    else
        max = y;
    return max;
You can also write this function as follows:
double larger (double x, double y)
                                                           double larger (double x, double y)
    if (x >= y)
                                                               if (x >= y)
       return x;
                                                                   return x;
        return y;
                                                               return y;
```

fit@hcmus | Programming 1 | 2023



22

# Examples

```
fit@hcmus
```

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





### Function main

- o The main part of a program is the definition of a function called main.
- When program runs, the function main is automatically called.
- Some compiler requires return 0; in the main function.

fit@hcmus | Programming 1 | 2023



24

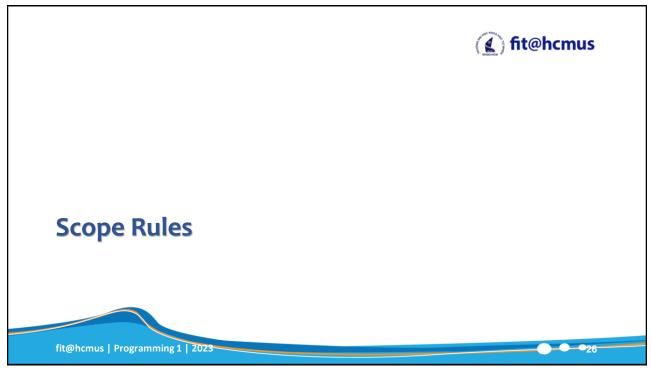
## **Recursive Functions**

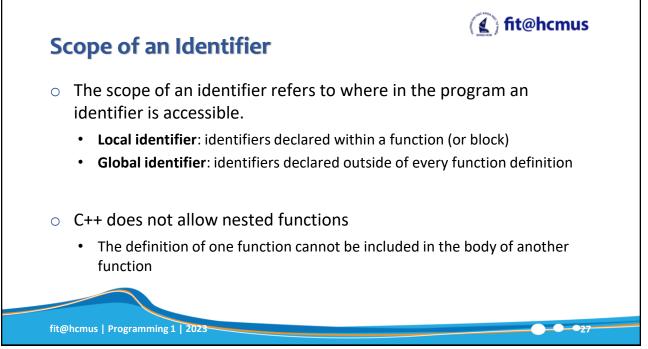


- The function calls itself.
- Example:

```
int Factorial(int N)
{
   if (N == 0)
       return 1;
   return N * Factorial(N-1);
}
```









### **Local Variables**

- Variables are declared within the body of a function.
  - Local to that function.
- Two same name (local) variables in two different functions are different.

fit@hcmus | Programming 1 | 2023



28

### **Local Variables**



- Variables are declared within the body of a function.
  - Local to that function.
- Two same name (local) variables in two different functions are different.



### **Global Constants**

- O Use the const modifier to name constant value.
  const double TAX RATE = 0.1; //VAT tax: 10%
- If the declaration is outside of all functions, the named constant is global named constant.

fit@hcmus | Programming 1 | 2023



30

### **Global Variables**



- Global variable:
  - same as global named constant, without using const modifier.
  - accessible to all function definitions in a file.



### **Global Variables**

- Some compilers initialize global variables to default values
- The operator :: is called the scope resolution operator
- By using the scope resolution operator
  - A global variable declared before the definition of a function (block) can be accessed by the function (or block) even if the function (or block) has an identifier with the same name as the variable.

fit@hcmus | Programming 1 | 2023

-0-0-32

32

#### ( fit@hcmus Global Variables | Example #include <iostream> int a; void PrintA() 5 6 ₩ 7 int a = 7; std::cout << "Local variable: " << a << std::endl;</pre> 8 std::cout << "Global variable: " << ::a << std::endl;</pre> 9 } 10 🛦 12 int main() 13 ▼ a = 15;14 PrintA(); 15 16 return 0; 17 ▲ } fit@hcmus | Programming 1 | 2023



### **Side Effects**

- Using global variables has side effects
  - · A function that uses global variables is not independent
  - If more than one function uses the same global variable and something goes wrong
    - · It is difficult to find what went wrong and where
    - Problems caused in one area of the program may appear to be from another area
- Global named constants have no side effects.

fit@hcmus | Programming 1 | 2023



34

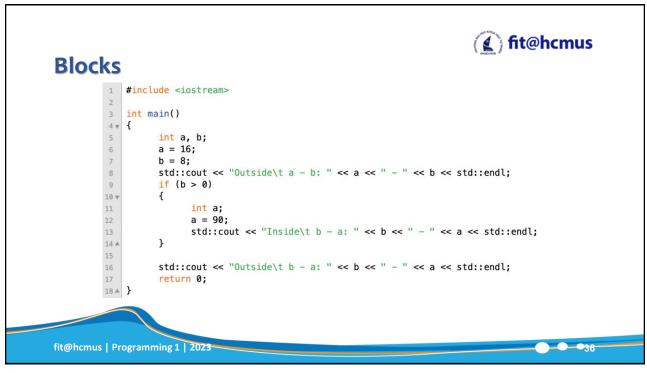
## Blocks



- A block is some C++ code enclosed in braces.
- Variables declared in a block are local to that block.

fit@hcmus | Programming 1 | 2023

**──**─────35

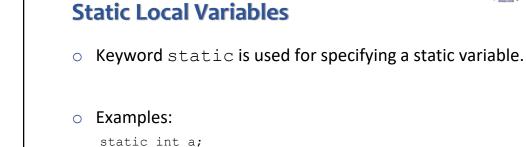






o If one identifier is declared in two blocks (nested), they are different variables with the same name.

```
(1) fit@hcmus
Nested Blocks | #include <iostream>
                                  int main()
                               4 w {
                                        int a, b;
                               5
                                        a = 16;
                                        b = 8;
                                        std::cout << "Outside\t a - b: " << a << " - " << b << std::endl;
                               9
                                        if (b > 0)
                              10 w
                                              int a;
                                              a = 90;
                              12
                                              if (a % 2 == 0)
                              13
                              14 w
                                              {
                              15
                                                    int a = 24:
                                                    std::cout << "Nested\t b - a: " << b << " - " << a << std::endl;
                              16
                              17 ▲
                                              }
                                              else
                              18
                                                    std::cout << a << " is odd\n";
                              19
                                              std::cout << "Inside\t b - a: " << b << " - " << a << std::endl;
                              20
                              21 4
                                        std::cout << "Outside\t b - a: " << b << " - " << a << std::endl;
                              23
                              24
                              25 ▲ }
fit@hcmus | Programming 1 | 2023
```



static float b;

fit@hcmus | Programming 1 | 2023

● ● ●39

fit@hcmus



### **Static Local Variables**

- A static local variable exists only inside a function where it is declared (like a local variable) but its lifetime starts when the function is called and ends only when the program ends.
- The main difference between local variable and static variable is that the value of static variable persists the end of the program.

fit@hcmus | Programming 1 | 2023

**9**-**9**-40

40

#### fit@hcmus **Static Local Variables | Example** #include <iostream> void Test() 3 4 ▼ { static int count = 0; count++; 6 7 std::cout << "Call of " << count << std::endl;</pre> 8 🛦 int main() 9 10 ▼ { Test(); 11 12 Test(); return 0; 13 14 ▲ } fit@hcmus | Programming 1 | 2023



## **Example | Classifying Numbers**

- We use **functions** to write the program that determines the number of odds and evens from a given list of integers.
- Main algorithm remains the same:
  - Initialize variables, zeros, odds, evens to 0
  - · Read a number.
  - · If number is even, increment the even count
    - If number is also zero, increment the zero count; else increment the odd count
  - Repeat Steps 2-3 for each number in the list.

fit@hcmus | Programming 1 | 2023



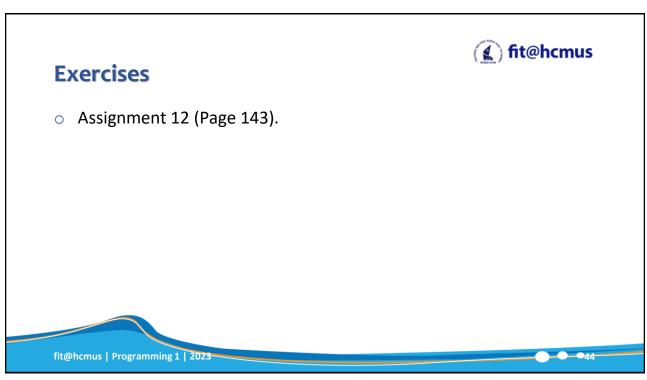
42

## **Example | Classifying Numbers**



- The program functions include:
  - initialize: initialize the variables, such as zeros, odds, and evens
  - getNumber: get the number
  - **classifyNumber**: determine if number is odd or even (and whether it is also zero); this function also increments the appropriate count
  - printResults: print the results









### **Value vs Reference Parameters**

- Call-by-Value parameter: a formal parameter that receives a copy of the content of corresponding actual parameter.
  - Can be variables or expressions.
- Call-by-Reference parameter: a formal parameter that receives the location (memory address) of the corresponding actual parameter.
  - Only be variables.

fit@hcmus | Programming 1 | 2023



46

## **Call-by-Value Parameters**



- Is actually a local variable.
- The value of the corresponding actual parameter is copied into it.
- The parameter has its own copy of the data.
- During program execution
  - The parameter manipulates the data stored in its own memory space.





## **Call-by-Reference Parameters**

- It receives the memory address of the corresponding actual parameter.
- The parameter stores the address of the corresponding actual parameter.
- During program execution to manipulate data
  - The address stored in the parameter directs it to the memory space of the corresponding actual parameter.

fit@hcmus | Programming 1 | 2023



48





- Indicating the call-by-reference parameters by attaching the ampersand sign & at the of the type name in formal parameter list.
- Example:

```
void getInput(double& N);
void sum(int N, int& s);
```





## **Call-by-Reference Parameters**

- o Call-by-Reference parameters can:
  - Pass one or more values from a function
  - Change the value of the actual parameter
- o Call-by-Reference parameters are **useful** in three situations:
  - · Returning more than one value
  - Changing the actual parameter
  - When passing the address would save memory space and time

fit@hcmus | Programming 1 | 2023



50

## **Example**



Write a function to swap the value of two integer variables a, b.





## **Example**

- Write a function to swap the value of two integer variables a, b.
- O Version 01:

```
void swap(int a, int b)
{
   int temp;
   temp = a;
   a = b;
   b = temp;
}
```

fit@hcmus | Programming 1 | 2023



54

## Example



- Write a function to swap the value of two integer variables a, b.
- Version 02:

```
void swap(int& a, int& b)
{
   int temp;
   temp = a;
   a = b;
   b = temp;
}
```





fit@hcmus | Programming 1 | 2023

57



### **Overloaded Functions**

**Function Overloading** 

- Two or more functions having same name but different argument(s) are known as overloaded functions.
- The signature of a function consists of the function name and its formal parameter list.
- Examples:

```
int test();
void test(int);
void test(float);
int test(float);
float test();
```





## **Overloaded Functions | Examples**

```
void display(int var) {
   std::cout << "Integer number: " << var << std::endl;
}

void display(float var) {
   std::cout << "Float number: " << var << std::endl;
}

void display(int var1, float var2) {
   std::cout << "Integer number: " << var1 << std::endl;
   std::cout << "Integer number: " << var2 << std::endl;
}</pre>
```

fit@hcmus | Programming 1 | 2023

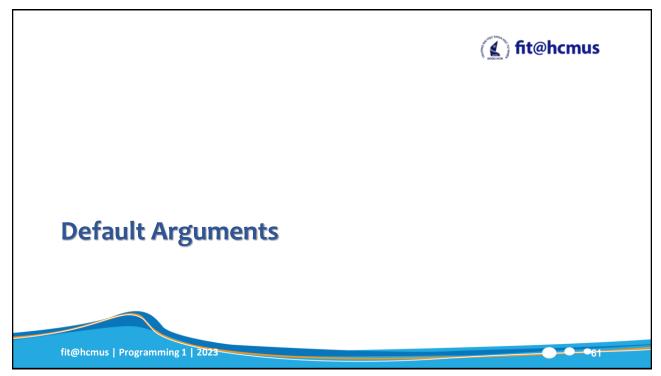
59

# **Overloaded Functions | Examples**

```
1 ▼ int absolute(int var) {
         if (var < 0)
3
             var = -var;
        return var;
4
5 A }
6
7 ▼ float absolute(float var){
        if (var < 0.0)
8
            var = -var;
9
10
        return var;
11 ▲ }
```

fit@hcmus | Programming 1 | 2023









- In a function call, the number of actual parameters and formal parameters must be the same.
  - C++ relaxes this condition for functions with default parameters
- You specify the value of a default parameter when the function name appears for the first time (e.g., in the prototype).
- If you do not specify the value of a default parameter, the default value is used.

fit@hcmus | Programming 1 | 2023

**\_\_\_\_**62



## **Default Arguments**

- All default parameters must be the **rightmost** parameters of the function.
- Default values can be constants, global variables, or function calls
  - Cannot assign a constant value as a default value to a reference parameter.
- In a function call where the function has more than one default parameter and a value to a default parameter is not specified:
  - You must omit all of the arguments to its right

fit@hcmus | Programming 1 | 2023



63

## **Examples**



```
#include <iostream>
2
   // A function with default arguments, it can be called with
3
   // 2 arguments or 3 arguments or 4 arguments.
   int sum(int x, int y, int z = 0, int w = 0)
5
6 ₹ {
          return (x + y + z + w);
7
8 4 }
   int main()
10
11 ▼ {
          std::cout << sum(10, 15) << std::endl;
12
          std::cout << sum(10, 15, 25) << std::endl;
13
          std::cout << sum(10, 15, 25, 30) << std::endl;
14
          return 0;
15
16 ▲ }
```

fit@hcmus | Programming 1 | 2023

-64



## **Examples**

Illegal function prototypes:

```
void funcOne(int x, double z = 23.45, char ch, int u = 45);
int funcTwo(int length = 1, int width, int height = 1);
void funcThree(int x, int& y = 16, double z = 34);
```

fit@hcmus | Programming 1 | 2023



fit@hcmus

65

# **Examples**

Consider the following function prototype:

```
void testDefaultParam(int a, int b = 7, char z = '*');
```

Which of the following function calls is correct?

- a. testDefaultParam(5);
- b. testDefaultParam(5, 8);
- c. testDefaultParam(6, '#');
- d. testDefaultParam(0, 0, '\*');



```
(1) fit@hcmus
Function as an Argument
                             1 #include <iostream>
                                  int add(int a, int b)
                              4 ₩ {
                              5
                                        return a + b;
                              6 A }
                              8
                                  int subtract(int a, int b)
                              9 ₩ {
                             10
                                        return a - b;
                             11 ▲ }
                             13
                                  void Print(int Func(int, int), int a, int b, char c)
                             14 ₹ {
                                        std::cout << a << c << b << " is " << Func(a, b) << std::endl;
                             15
                             16 ▲ }
                             17
                             18
                                  int main()
                             19 ₹ {
                                       int a = 16, b = 8;
Print(add, a, b, '+');
Print(subtract, a, b, '-');
                             20
                             21
                             23
                                        return 0;
                             24▲ }
fit@hcmus | Programming 1 | 2023
```

