جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني

Department of Cyber Security

Subject:

Object Oriented Programming (OOP)

Class:

Second

Lecturer:

Dr. Abdulkadhem A. Abdulkadhem

Lecture: (2)

Functions and Parameter Transmission

1. Introduction to Functions

What is a Function?

A function is a block of code designed to perform a specific task, reusable throughout a program. C++ allows flexible function definitions and parameter handling.

Parameter Transmission in Functions:

In C++, parameters can be passed in multiple ways:

- **Pass by Value**: A copy of the argument is passed.

- A **copy of the variable** is passed to the function.
- Any changes inside the function **do not affect the original variable**.

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

void squareByValue(int n) {
    n = n * n;                              // modifies only the copy
    cout << "Inside function (by value): " << n << endl;
}

int main() {
    int x = 5;
    squareByValue(x);
    cout << "Outside function: " << x << endl;
    return 0;
}
```

Expected Output:

```
Inside function (by value): 25
Outside function: 5
```

 The original `x` remains unchanged.

- **Pass by Reference**: The actual argument is passed, allowing the function to modify the argument.

- The **actual variable (its memory address)** is passed to the function.
- Any changes inside the function **directly affect the original variable**.

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

void squareByReference(int &n) {
    n = n * n;                                  // modifies the original variable
    cout << "Inside function (by reference): " << n << endl;
}

int main() {
    int x = 5;
    squareByReference(x);
    cout << "Outside function: " << x << endl;
    return 0;
}
```

Expected Output:

```
Inside function (by reference): 25
Outside function: 25
```

- The original `x` is modified.

Benefits: Saves memory (no copy is made) and allows modification of the caller's variables.

2. Function Overloading

Definition: Function overloading allows multiple functions to have the same name with different signatures (i.e., different parameter types or numbers of parameters).

Benefits: Provides better code readability and reusability.

Example Code:

```
#include <iostream>
using namespace std;
// Function to calculate the area of a rectangle
int area(int length, int width) {
   return length * width;
}
// Overloaded function to calculate the area of a circle
double area(double radius) {
   return 3.1415 * radius * radius;
}
```

```
int main() {
   int length = 5, width = 10;
   double radius = 7.5;

   cout << "Area of rectangle: " << area(length, width) << endl;
   cout << "Area of circle: " << area(radius) << endl;

   return 0;
}
```

Explanation:
The 'area' function is overloaded: one version takes two 'int' parameters, and the other takes a 'double' parameter. The correct function is selected based on the argument type.

3. Inline Functions

Definition: Inline functions are expanded in line where they are called, which can reduce function call overhead, especially for small functions.

When to Use: Inline functions should be used for small, frequently called functions.

Example Code:

```
#include <iostream>
using namespace std;
// Inline function to add two numbers
inline int add(int a, int b) {
   return a + b;
}

int main() {
   int x = 10, y = 20;
   cout << "Sum: " << add(x, y) << endl;

   return 0;
}
```

Explanation:
The 'add' function is declared as 'inline'. When called, the compiler replaces the function call with the function's code, which can improve performance for small functions.

4. Default Arguments

Definition: Default arguments allow function parameters to have default values if no arguments are passed during the function call.

Advantages: Simplifies function calls and improves code readability.

Example Code:

```
#include <iostream>
using namespace std;
// Function to print a message with a default argument
void greet(string name = "Guest") {
   cout << "Hello, " << name << "!" << endl;
}

int main() {
   greet();          // Uses default argument
   greet("Alice");    // Passes 'Alice' as argument

   return 0;
}
```

Explanation:
The 'greet' function has a default argument 'Guest'. If no argument is passed, the default value is used. Otherwise, the provided argument overrides the default value.

5. Return by Reference

Definition: Returning by reference allows a function to return a reference to a variable rather than a copy, enabling the caller to modify the returned variable directly.

When to Use: Used when you want the function to return a variable that can be modified by the caller.

Example Code:

```
#include <iostream>
using namespace std;
// Function that returns a reference to a variable
int& getLargest(int &a, int &b) {
   return (a > b) ? a : b;
}

int main() {
   int x = 5, y = 10;
   cout << "Before modification: x = " << x << ", y = " << y << endl;

   getLargest(x, y) = 100;  // Modifying the largest number by reference
   cout << "After modification: x = " << x << ", y = " << y << endl;

   return 0;
}
```

Explanation:
The 'getLargest' function returns a reference to the largest of two integers. In 'main', the returned reference is used to directly modify the largest variable.

Summary

1. Function Overloading allows multiple functions with the same name but different parameters.
2. Inline Functions can reduce the overhead of small function calls by expanding them inline.
3. Default Arguments provide default values for function parameters.
4. Pass by Reference allows a function to modify the caller's arguments directly, saving memory and time.
5. Return by Reference enables a function to return a reference to a variable, allowing modifications outside the function.

Homework Assignment

Instructions:

- Complete each task as specified.
- Write clean, well-commented C++ code for each task.
- Ensure that your code compiles and runs correctly.
- Due Date: 1 week.
- Submit your homework to the google form [https://forms.gle/7M6B1iRqjNbQsVdV6].

Task 1: Function Overloading

Write an overloaded function named *calculate* to perform the following operations:

- For two integer inputs: Return the sum of the two numbers.
- For two floating-point inputs: Return the product of the two numbers.
- For three integer inputs: Return the largest of the three numbers.

```
cout << calculate(5, 10);      // Output: 15 (integer sum)
cout << calculate(2.5, 4.0);   // Output: 10.0 (floating-point product)
cout << calculate(3, 7, 2);    // Output: 7 (largest of three integers)
```

Task 2: Default Arguments

Write a function named *printMessage* that takes two parameters: message (a string) and times (an integer). The function should print the message the specified number of times. If no times argument is provided, the default value should be 3.

Example:

```
printMessage("Hello!");       // Prints "Hello!" 3 times
printMessage("Hi!", 5);       // Prints "Hi!" 5 times
```

Task 3: MCQ questions

Q1. Which of the following best describes *function overloading* in C++?
a) Having multiple functions with the same name and identical parameter list.
b) Defining multiple functions with the same name but different parameter lists.
c) Declaring the same function in different files.
d) Allowing a function to return multiple values.
e) Using default arguments in a function.

Q2. Consider the following code:

```
#include <iostream>
using namespace std;
inline int multiply(int a, int b) {
    return a * b;
}
int main() {
    int x = 3, y = 4;
    cout << multiply(x, y);
}
```

What is the main purpose of declaring `multiply` as `inline`?
a) It prevents the function from being called multiple times.
b) It forces the compiler to generate a new copy of the function each time.
c) It eliminates function call overhead by expanding the function code directly at the call site.
d) It makes the function automatically overloaded.
e) It allows the function to accept reference parameters.

Q3. Which of the following function definitions correctly demonstrates the use of a *default argument*?
a) `void show(int x, int y);`

b) `void show(int x = 10, int y = 20);`

c) `void show(int x; int y = 20);`

d) `int show(int x, int y = "Guest");`

e) `int show(int x, int y = default);`

Q4. What will be the output of the following code?

```
#include <iostream>
using namespace std;
void greet(string name = "Guest") {
    cout << "Hello, " << name << "!" << endl;
}
int main() {
```

```
    greet();
    greet("Alice");
}
```

a) Hello, Guest!
   Hello, Alice!
b) Hello, !
   Hello, Alice!
c) Hello, Guest!
   Hello, Guest!
d) Compile-time error
e) Runtime error

Q5. In the following code, what happens when `swap(x, y)` is executed?

```
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int x = 5, y = 10;
    swap(x, y);
    cout << x << " " << y;
}
```

a) Output: `5 10`
b) Output: `10 5`
c) Output: `0 0`
d) Output: `x y`
e) Compile-time error

Q6. Consider the following code:

```
int& getLargest(int &a, int &b) {
    return (a > b) ? a : b;
}
int main() {
    int x = 8, y = 12;
    getLargest(x, y) = 50;
    cout << x << " " << y;
}
```

What will be the output?
a) 8 12
b) 8 50
c) 50 12
d) 50 50
e) Compile-time error