جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني
DEPARTMENT OF CYBER SECURITY

SUBJECT:

STRUCTURED PROGRAMMING

CLASS:

1ST STAGE

LECTURER:

DR. ABDULKADHEM A. ABDULKADHEM

LECTURE: (4)

Local and Global Variables, Actual and Formal Arguments, Recursive and Overloading in Functions

1. Local and Global Variables in Functions

A. Local Variables

Local variables are declared inside a function and can only be accessed within that function.

Example 1: Using Local Variables

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

void localExample() {
    int num = 10; // Local variable
    cout << "Local variable: " << num << endl;
}

int main() {
    localExample();
    cout << num;          // This would cause an error because num is local
                              to localExample()
    return 0;
}
```

Example 2: Another Local Variable Example

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

void printSum() {
    int a = 5, b = 10; // Local variables
    cout << "Sum: " << (a + b) << endl;
}

int main() {
    printSum();
    return 0;
}
```

B. Global Variables

Global variables are declared outside of all functions and can be accessed by any function in the program.

Example 1: Using a Global Variable

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

int globalVar = 100; // Global variable

void displayGlobal() {
    cout << "Global variable: " << globalVar << endl;
}

int main() {
    displayGlobal();
    return 0;
}
```

Example 2: Modifying a Global Variable

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

int count = 0; // Global variable

void increment() {
    count++;
    cout << "Count: " << count << endl;
}

int main() {
    increment();
    increment();
    return 0;
}
```

2. Actual and Formal Arguments

A. Actual Arguments

These are the values passed to a function when it is called.

B. Formal Arguments

These are the parameters defined in the function signature that receive values from actual arguments.

Example 1: Passing Values to a Function

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

void display(int num) { // num is a formal argument
    cout << "Number: " << num << endl;
}

int main() {
    int value = 10;
    display(value); // value is the actual argument
    return 0;
}
```

Example 2: Adding Two Numbers with Parameters

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

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

int main() {
    int sum = add(5, 7); // Actual arguments
    cout << "Sum: " << sum << endl;
    return 0;
}
```

3. Recursive Functions

A recursive function is a function that calls itself to solve a problem in a smaller scope.

Example 1: Factorial Calculation

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

int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}

int main() {
    cout << "Factorial of 5: " << factorial(5) << endl;
```

```
    return 0;
}
```

Example 2: Fibonacci Series

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

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    cout << "Fibonacci(6): " << fibonacci(6) << endl;
    return 0;
}
```

4. Function Overloading

Function overloading allows multiple functions to have the same name but different parameter lists or types.

Example 1: Overloading Addition Function

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

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

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

int main() {
    cout << "Sum (int): " << add(5, 10) << endl;
    cout << "Sum (double): " << add(2.5, 3.7) << endl;
    return 0;

}
```

Example 2: Overloaded Print Function

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

void print(int num) {
```

```
    cout << "Integer: " << num << endl;
}

void print(double num) {
    cout << "Double: " << num << endl;
}

void print(string text) {
    cout << "String: " << text << endl;
}

int main() {
    print(10);
    print(5.5);
    print("Hello");
    return 0;
}
```

Example 3: Overloaded Multiply Function

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

// Overloaded function to multiply two or three numbers
int multiply(int a, int b) {
    return a * b;
}

int multiply(int a, int b, int c) {
    return a * b * c;
}

int main() {
    cout << "Multiply (2 numbers): " << multiply(3, 4) << endl;
    cout << "Multiply (3 numbers): " << multiply(2, 3, 4) << endl;
    return 0;
}
```

Example 4: Overloaded Display Function

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

// Overloaded function to display different numbers of arguments
void display(int a) {
    cout << "Single integer: " << a << endl;
}

void display(int a, int b) {
    cout << "Two integers: " << a << " and " << b << endl;
}

int main() {
```

```
    display(5);
    display(7, 9);
    return 0;
}
```