BORG AL ARAB TECHNOLOGICAL UNIVERSITY

CODE: ITS501

ADVANCED PROGRAMMING IN C

Third Year –Information Technology Program Fall **2026**

FUNCTIONS IN C PROGRAMMING:
VARIABLE SCOPE AND BEST PRACTICES

UNDERSTANDING VARIABLE SCOPE, BEST PRACTICES, AND COMMON MISTAKES

INTRODUCTION TO VARIABLE SCOPE

- **Definition:** Variable scope refers to the visibility and lifetime of variables within different parts of a program.

- **Importance:** Understanding scope is important for managing memory and avoiding errors.

TYPES OF VARIABLE SCOPE

- 1. Function Scope
- 2. Block Scope
- 3. File Scope
- 4. Global Scope

FUNCTION SCOPE

- **Definition:** Variables declared inside a function are only accessible within that function.
- **Example:**

```c
#include <stdio.h>

void example() {
    int a = 5; // Function-scoped variable
    printf("Inside function: %d\n", a); // Output: 5
}

int main() {
    example();
    // printf("Outside function: %d\n", a); // Error: 'a' not declared in this scope
    return 0;
}
```

```
Inside function: 5
```

FUNCTION SCOPE

```c
#include <stdio.h>

void example() {
    int a = 5; // Function-scoped variable
    printf("Inside function: %d\n", a); // Output: 5
}

int main() {
    example();
    printf("Outside function: %d\n", a); // Error: 'a' not declared in this scope
    return 0;
}
```

```
prog.c: In function ‘main’:
prog.c:10:38: error: ‘a’ undeclared (first use in this function)
   10 |     printf("Outside function: %d\n", a); // Error: 'a' not declared in this scope
      |                                      ^
prog.c:10:38: note: each undeclared identifier is reported only once for each function it appears in
```

BLOCK SCOPE

- **Definition:** Variables declared **inside a block** (enclosed by {}) **are only accessible** within that block.
- **Example:**

```c
#include <stdio.h>

void function() {
    int x = 10; // Block-scoped variable
    if (x > 5) {
        int y = 20; // 'y' is only accessible within this if block
        printf("Inside block: %d\n", y); // Output: 20
    }
    //printf("Outside block: %d\n", y); // Error: 'y' not declared in this scope
}

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

```
Inside block: 20
```

BLOCK SCOPE

- **Example:**

```c
#include <stdio.h>

void function() {
    int x = 10; // Block-scoped variable
    if (x > 5) {
        int y = 20; // 'y' is only accessible within this if block
        printf("Inside block: %d\n", y); // Output: 20
    }
        printf("Outside block: %d\n", y); // Error: 'y' not declared in this scope
}

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

```
prog.c: In function ‘function’:
prog.c:9:39: error: ‘y’ undeclared (first use in this function)
    9 |         printf("Outside block: %d\n", y); // Error: 'y' not declared in this scope
      |                                       ^
prog.c:9:39: note: each undeclared identifier is reported only once for each function it appears in
```

BLOCK SCOPE

- **Example:**

```c
#include <stdio.h>

void function() {
    int x = 10; // Block-scoped variable
    if (x > 5) {
        int y = 20; // 'y' is only accessible within this if block
        printf("Inside block: %d\n", y); // Output: 20
    }
        //printf("Outside block: %d\n", y); // Error: 'y' not declared in this scope
}

int main() {
    function();
    printf("In main function: %d\n", y); // Error: 'y' not declared in this scope
    return 0;
}
```

```
prog.c: In function ‘main’:
prog.c:14:40: error: ‘y’ undeclared (first use in this function)
   14 |       printf("In main function: %d\n", y); // Error: 'y' not declared in this scope
      |                                        ^
prog.c:14:40: note: each undeclared identifier is reported only once for each function it appears in
```

BLOCK SCOPE

- **Example:**

```c
#include <stdio.h>

int main() {


  for (int i = 1; i <= 2; ++i)
    {
    printf("print i inside : %d\n", i);
    }

    printf("print  i outside: %d\n", i);

  return 0;
}
```

```
prog.c: In function ‘main’:
prog.c:11:38: error: ‘i’ undeclared (first use in this function)
   11 |     printf("print  i outside: %d\n", i);
      |                                      ^
prog.c:11:38: note: each undeclared identifier is reported only once for each function it appears in
```

```c
#include <stdio.h>

int main() {

  int i;
  for ( i = 1; i <= 2; ++i)
    {
    printf("print i inside : %d\n", i);
    }

    printf("print  i outside: %d\n", i);

  return 0;
}
```

```
print i inside : 1
print i inside : 2
print  i outside: 3
```

FILE SCOPE

- **Definition:** Variables **declared outside** of any function are accessible throughout the file.
- **Example:**

```c
#include <stdio.h>

int globalVar = 100; // File-scoped variable

void display() {
    printf("File scope variable: variable called from display fnc %d\n", globalVar); // Output: 100
}

int main() {
    display();
    printf("File scope variable: variable called from main fnc %d\n", globalVar); // Output: 100
    return 0;
}
```

```
File scope variable: variable called from display fnc 100
File scope variable: variable called from main fnc 100
```

FILE SCOPE

- **Example:**

```c
#include <stdio.h>

int globalVar = 100; // File-scoped variable

void display() {
    printf("File scope variable: variable called from display fnc %d\n", globalVar); // Output: 100
}

int main() {
    display();
    globalVar=200; // changing value of global variable within file
    printf("File scope variable: variable called from main fnc %d\n", globalVar); // Output: 200
    return 0;
}
```

```
File scope variable: variable called from display fnc 100
File scope variable: variable called from main fnc 200
```



GLOBAL SCOPE

- **Definition:** Global variables can be accessed by any function or file using the keyword “extern”.

```c
int globalVar = 50; // Global variable

void func1() {
    printf("Global variable in func1: %d\n", globalVar); // Output: 50
}

void func2() {
    globalVar = 100; // Modifies the global variable
    printf("Modified global variable in func2: %d\n", globalVar); // Output: 100
}

int main() {
    func1();
    func2();
    func1(); // Check the modified value
    return 0;
}
```

```
Global variable in func1: 50
Modified global variable in func2: 100
Global variable in func1: 100
```

VARIABLE SHADOWING

- **Definition:** Occurs when a local variable has **the same name** as a global variable, hiding the global one.

```c
#include <stdio.h>

int num = 10; // Global variable

void show() {
    int num = 20; // Local variable (shadows global)
    printf("Local num in show fnc: %d\n", num); // Output: 20 (prints local num)
}

int main() {
    printf("Global num in main fnc: %d\n", num); // Output: 10 (prints global num)
    show();
    printf("Global num in main fnc: %d\n", num); // Output: 10 (prints global num)
    return 0;
}
```

```
Global num in main fnc: 10
Local num in show fnc: 20
Global num in main fnc: 10
```

COMMON ERRORS RELATED TO SCOPE

- **Error Examples:** Accessing a variable outside its scope leads to compilation errors.
- Shadowing can cause confusion if not managed carefully.
- **Example of Error:**

```c
#include <stdio.h>

void anotherFunction() {
    printf("Accessing a: %d\n", a); // Error: 'a' not declared in this scope
}

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

```
prog.c: In function ‘anotherFunction’:
prog.c:4:34: error: ‘a’ undeclared (first use in this function)
    4 |       printf("Accessing a: %d\n", a); // Error: 'a' not declared in this scope
      |                                  ^
prog.c:4:34: note: each undeclared identifier is reported only once for each function it appears in
```

DEBUGGING SCOPE ISSUES

Tips:

- Use meaningful variable names to avoid shadowing.
- Utilize debugging tools or print statements to trace variable values.

```c
void debug() {
    int localVar = 20; // Local variable
    printf("Local variable: %d\n", localVar); // Output: 20
    printf("Global variable: %d\n", globalVar); // Output: 30
}

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

```
Local variable: 20
Global variable: 30
```

NOTES

- **Local Variables**: Variables declared inside a function or block. They are created when the function is called and destroyed when it exits.
- **Global Variables**: Declared outside any function and can be accessed from any function within the same source file. They exist for the lifetime of the program.
- In C, the value of an **uninitialized variable** is considered **undefined**. This means that it could contain **any value that happens to be in the memory** location assigned to it at that moment. Example int a;

QUESTIONS

- Can a variable declared inside a function be accessed from the main function in C?
- Answer:
  - No, a variable declared inside a function **cannot be accessed directly from the main function** (or any other function) in C.
  - Variables declared within a function have local scope, meaning they are only accessible within that function.

EXAMPLE

Explanation

In the example: localVar is declared in myFunction, so it is only accessible within that function.
Attempting to access localVar in main results in a compilation error because localVar is not visible there.

```c
void myFunction() {
    int localVar = 42; // Local variable
    printf("Inside myFunction: %d\n", localVar);
}

int main() {
    myFunction();
    printf("In main: %d\n", localVar); // Error: 'localVar' not declared in this scope
    return 0;
}
```

```
prog.c: In function ‘main’:
prog.c:10:29: error: ‘localVar’ undeclared (first use in this function)
   10 |     printf("In main: %d\n", localVar); // Error: 'localVar' not declared in this scope
      |                             ^~~~~~~~
prog.c:10:29: note: each undeclared identifier is reported only once for each function it appears in
```

EXAMPLE WITH LOCAL VARIABLE POINTER

```c
#include <stdio.h>

void myFunction() {
    int localVar = 42; // Local variable
    int *ptr = &localVar; // Pointer to local variable
    printf("Inside myFunction: value is %d    address is %d\n", *ptr, &ptr);
}

int main() {
    myFunction();
    // printf("In main: %d\n", *ptr); // Error: 'ptr' not declared in this scope
    // The pointer 'ptr' is not accessible here.
    return 0;
}
```

```
Inside myFunction: value is 42    address is 1733481520
```

A pointer ptr is declared and initialized with the address of localVar. The & operator gets the address of localVar, allowing ptr to point to it.

if you try to access a pointer to a local variable after the function has exited? It will cause a segmentation fault.

Pointer Validity:
While ptr holds the address of localVar, once myFunction exits, that address becomes invalid because localVar no longer exists.

If a function returns a pointer to a local variable, you expect It will cause undefined behavior when trying to access that pointer outside the function

```
prog.c: In function ‘main’:
prog.c:13:31: error: ‘ptr’ undeclared (first use in this function)
   13 |      printf("In main: %d\n", *ptr); // Error: 'ptr' not declared in this scope
      |                               ^~~
prog.c:13:31: note: each undeclared identifier is reported only once for each function it appears in
```

HOW TO ACCESS

If you need to access a variable's value in another function, you can:

- Return the variable's value from the function.
- Pass the variable as a parameter to another function.
- Use global variables (though this is generally not recommended).

BEST PRACTICES FOR VARIABLE SCOPE

- **Use Local Variables Whenever Possible**
- **Minimize Global Variables:** to reduce dependencies
- **Meaningful Naming:** Use descriptive names for variables to indicate their purpose and scope.

SUMMARY

Recap Key Points:

- Understand different types of scope: block, function, file, and global.
- Be cautious of variable shadowing.
- Practice debugging techniques to handle scope-related issues.
- Follow best practices and avoid bad practices for cleaner, maintainable code