Scope of Variables in C

Scope = Lifetime

The area under which a variable is applicable.

Strict definition : A block or a region where a variable is declared, defined and used and when a block or a region ends, variable is automatically destroyed.

C

```c
#include <stdio.h>

int main()
{
    int var = 34;    // Scope of this variable is within main() function only.
                    // Therefore, called LOCAL to main() function.

    printf("%d", var);

    return 0;
}
```

Output

```
34
```

Scope of an identifier is the part of the program where the identifier may directly be accessible. In C, all identifiers are lexically(or statically) scoped. C scope rules can be covered under the following two categories.

There are basically 4 scope rules:

Scope	Meaning
File Scope	Scope of a Identifier starts at the beginning of the file and ends at the end of the file. It refers to only those Identifiers that are declared outside of all functions. The Identifiers of File scope are visible all over the file Identifiers having file scope are global

Scope	Meaning
Block Scope	Scope of a Identifier begins at opening of the block / '{' and ends at the end of the block / '}'. Identifiers with block scope are local to their block
Function Prototype Scope	Identifiers declared in function prototype are visible within the prototype
Function scope	Function scope begins at the opening of the function and ends with the closing of it. Function scope is applicable to labels only. A label declared is used as a target to goto statement and both goto and label statement must be in same function

1. File Scope

These variables are usually declared outside of all of the functions and blocks, at the top of the program and can be accessed from any portion of the program. These are also called the global scope variables as they can be globally accessed.

Example 1:

C

```c
// C program to illustrate the global scope

#include <stdio.h>

// Global variable
int global = 5;

// global variable accessed from
// within a function
void display()
{
    printf("%d\n", global);
}

// main function
int main()
{
    printf("Before change within main: ");
    display();

    // changing value of global
```

Output:

```
Before change within main: 5
After change within main: 10
```

Example 2:

C

```c
// filename: file1.c

int a;

int main(void)
{
a = 2;
}
```

C

```c
// filename: file2.c
// When this file is linked with file1.c, functions
// of this file can access a

extern int a;

int myfun()
{
a = 2;
}
```

Note: To restrict access to the current file only, global variables can be marked as static.

2. Block Scope: A Block is a set of statements enclosed within left and right braces i.e. '{' and '}' respectively. Blocks may be nested in C(a block may contain other blocks inside it). A variable declared inside a block is accessible in the block and all inner blocks of that block, but not accessible outside the block. Basically these are local to the blocks in which the variables are defined and are not accessible outside.

C

```c
#include <stdio.h>

// Driver Code
int main()
{
    {
        int x = 10, y = 20;
        {
            // The outer block contains
            // declaration of x and
            // y, so following statement
            // is valid and prints
            // 10 and 20
            printf("x = %d, y = %d\n", x, y);
            {
                // y is declared again,
                // so outer block y is
                // not accessible in this block
                int y = 40;

                // Changes the outer block
                // variable x to 11
```

Output

```
x = 10, y = 20
x = 11, y = 41
x = 11, y = 20
```

3. Function Prototype Scope: These variables range includes within the function parameter list. The scope of the these variables begins right after the declaration in the function prototype and runs to the end of the declarations list. These scopes don’t include the function definition, but just the function prototype.

Example:

C

```c
// C program to illustrate
// function prototype scope
```

```c

#include <stdio.h>

// function prototype scope
//(not part of a function definition)
int Sub(int num1, int num2);

// file scope
int num1;

// Function to subtract
int Sub(int num1, int num2)
{
    return (num1-num2);
}

// Driver method
int main(void)
{
```

Output

```
5
```

4. Function Scope: A Function scope begins at the opening of the function and ends with the closing of it. Function scope is applicable to labels only. A label declared is used as a target to go to the statement and both goto and label statement must be in the same function.

Example:

C

```c
void func1()
{
    {
        // label in scope even
        // though declared later
        goto label_exec;

    label_exec:;
    }

    // label ignores block scope
    goto label_exec;
}

void funct2()
{

    // throwserror:
    // as label is in func1() not funct2()
    goto label_exec;
}
```

What if the inner block itself has one variable with the same name?
If an inner block declares a variable with the same name as the variable declared by the outer block, then the visibility of the outer block variable ends at the point of the declaration by inner block.

What about functions and parameters passed to functions?
A function itself is a block. Parameters and other local variables of a function follow the same block scope rules.

Can variables of the block be accessed in another subsequent block?
No, a variable declared in a block can only be accessed inside the block and all inner blocks of this block.

C

```c
// C program to illustrate scope of variables

#include<stdio.h>

int main()
{
    // Initialization of local variables
    int x = 1, y = 2, z = 3;

    printf("x = %d, y = %d, z = %d\n",
    x, y, z);
    {

        // changing the variables x & y
        int x = 10;
        float y = 20;

        printf("x = %d, y = %f, z = %d\n",
        x, y, z);
        {
```

Output

```
x = 1, y = 2, z = 3
x = 10, y = 20.000000, z = 3
x = 10, y = 20.000000, z = 100
```