

**QUESTION:**

**Write short notes on the following: 1. Local, Global and Static variables 2. Scope of a variable 3. Inline and Recursive functions.**

**SOLUTION:**

1. **Local Variables:**

Local variables are a specific type of variable that are only available within the context of a particular expression and can only be **accessed within the function that defines** them. Local variables are useful when you only need that data within a particular expression.

**For Example:**

#include <stdio.h>

int main ()

{

/\* local variable declaration \*/

int a, b;

int c;

/\* actual initialization \*/

a = 10;

b = 20;

c = a + b;

printf ("value of a = %d, b = %d and c = %d\n", a, b, c);

return 0;

}

1. **Global Variables:**

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

**For Example:**

#include <stdio.h>

/\* global variable declaration \*/

int g;

int main () {

/\* local variable declaration \*/

int a, b;

/\* actual initialization \*/

a = 10;

b = 20;

g = a + b;

printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

return 0;

}

1. **Static Variables:**

In programming, a static variable is the one allocated “statically,” which means **its lifetime is throughout the program run**. It is declared with the 'static' keyword and persists its value across the function calls.

Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve their previous value in their previous scope and are not initialized again in the new scope.

**Syntax:**

static data\_type var\_name = var\_value;

**For Example:**

#include<stdio.h>

int fun()

{

static int count = 0;

count++;

return count;

}

int main()

{

printf("%d ", fun());

printf("%d ", fun());

return 0;

}

**Output:**

1 1

1. **Scope of a variable:**

When we declare a variable in a program, it cannot be accessed against the scope rules. Variables can be accessed based on their scope. The scope of a variable decides the portion of a program in which the variable can be accessed. The scope of the variable is defined as follows-

**Scope of a variable is the portion of the program where a defined variable can be accessed.**

**For example:**

If you live in an apartment building, you have a single key that allows you access to your apartment. The owner/manager of the apartment may also have a master key that allows access to all apartments.

A similar concept applies to variables in C. **Variable scope** refers to the accessibility of a variable in a given program or function. A variable may only be available within a specific function (your apartment key), or it may be available to the entire C program (the shared access key).

1. **Inline and Recursive Functions:**

In programming terms, a recursive function can be defined as **a routine that calls itself directly or indirectly**. Using the recursive algorithm, certain problems can be solved quite easily. ... Moreover, every recursive program can be written using iterative methods.

**Inline Function** are those function whose definitions are small and be substituted at the place where its function call is happened. Function substitution is totally compiler choice.

An **inline function** is a function for which the compiler replaces a call to the function with the code for the function itself. The process of replacing a function call with the function's code is called **inlining**. When the compiler performs inlining for a function, the function has been **inlined**.

**For Example:**

1. **INLINE FUNCTION:**

#include <stdio.h>

// Inline function in C

static inline int foo()

{

return 2;

}

// Driver code

int main()

{

int ret;

// inline function call

ret = foo();

printf("Output is: %d\n", ret);

return 0;

}

1. **RECURSIVE FUNCTION:**

#include <stdio.h>

int fact (int);

int main()

{

    int n,f;

    printf("Enter the number whose factorial you want to calculate?");

    scanf("%d",&n);

    f = fact(n);

    printf("factorial = %d",f);

}

int fact(int n)

{

    if (n==0)

    {

        return 0;

    }

    else if ( n == 1)

    {

        return 1;

    }

    else

    {

        return n\*fact(n-1);

    }

}