Today our intention is to learn STORAGE CLASS.

But BEFORE defining STORAGE CLASS, Let us define some NEW **terms**. Those may not be NEW to you, but at least NEW to me. Okay..

We can **classify variables** depending on their place of declaration.
Thus we can classify variables as **Local** (internal) Variables and **Global** Variables.

Local variables: Variables declared within a function are called Local Variables.

Global variables: Variables declared outside of any function are called Global Variables.

Let me define two more NEW terms – **Scope** and **Longevity** of a variable.

The **scope** of a variable determines the portion of a program over which the variable is actually available for use (i.e. variable is Active).

The **longevity** refers to the period during which a variable retains a given value during execution of a program (i.e. variable remains alive).

Thus

- ✓ SCOPE refers to VISIBILITY of a variable
- ✓ LONGEVITY refers to LIFETIME of a variable.

What will be the **scope** and **longevity** of LOCAL and GLOBAL variables?

YOU NEED NOT TO GUESS, JUST GO TO NEXT PAGE.

Local Variable

1. Scope: Within the function in which it is declared.
2. Longevity: Within the function in which it is declared.

Global Variable

1. Scope: Entire program.
2. Longevity: Enter program.

Think of a C PROGRAM that contains LOCAL & GLOBAL variables **with same name**.

Is it possible?

If possible, which one will get preference?

YES, it is possible.

Rules/Properties regarding handling SUCH situation are discussed below.

Rules/Properties:

1. In case global variable and local variable have same name, the local variable will have the precedence over the global variable.

2. Once the global variable has been declared, any function can use it and change its value. Then subsequent functions will get only that new value.

Example to CLEAR the concept:

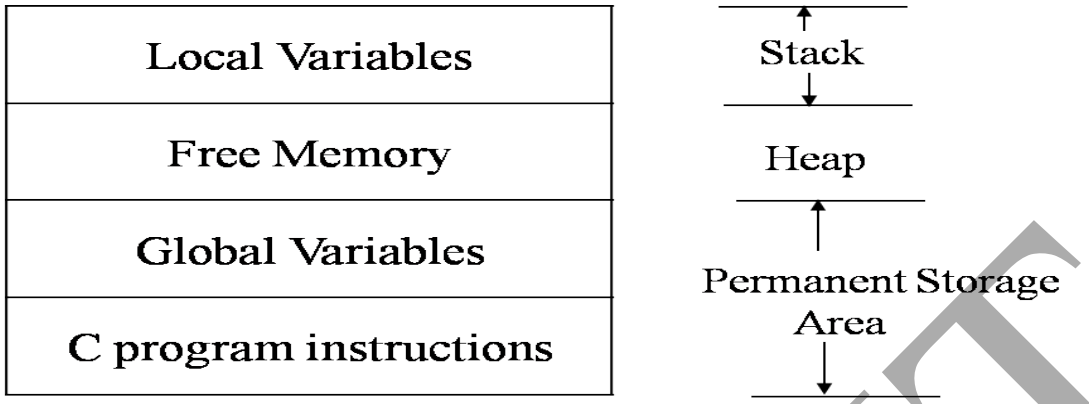
```
#include <stdio.h>
int n;                          // n is a GLOBAL variable by definition
void main()
{
        void fun1();            //  declaration of fun1( )
        void fun2();            //  declaration of fun2( )
        n=10;
        printf("\n%d  ", n);
        fun1( );                        // fun1( ) is called
        printf("%d  " , n);
        fun2( );                        // fun2( ) is called
        printf("%d  ", n);
        fun1( );                        // fun1( ) is called again
        printf("%d  ", n);
}

void fun1( )                            //Definition of fun1( )
{
        n=n+10;
        return;
 }

void fun2( )                            //Definition of fun2( )
{
        int n;                          //Here n is a Local variable by definition
        n=1;
        return;
 }
```

WHAT will be the OUTPUT of the program when executed?

Output will be
10 20 20 30

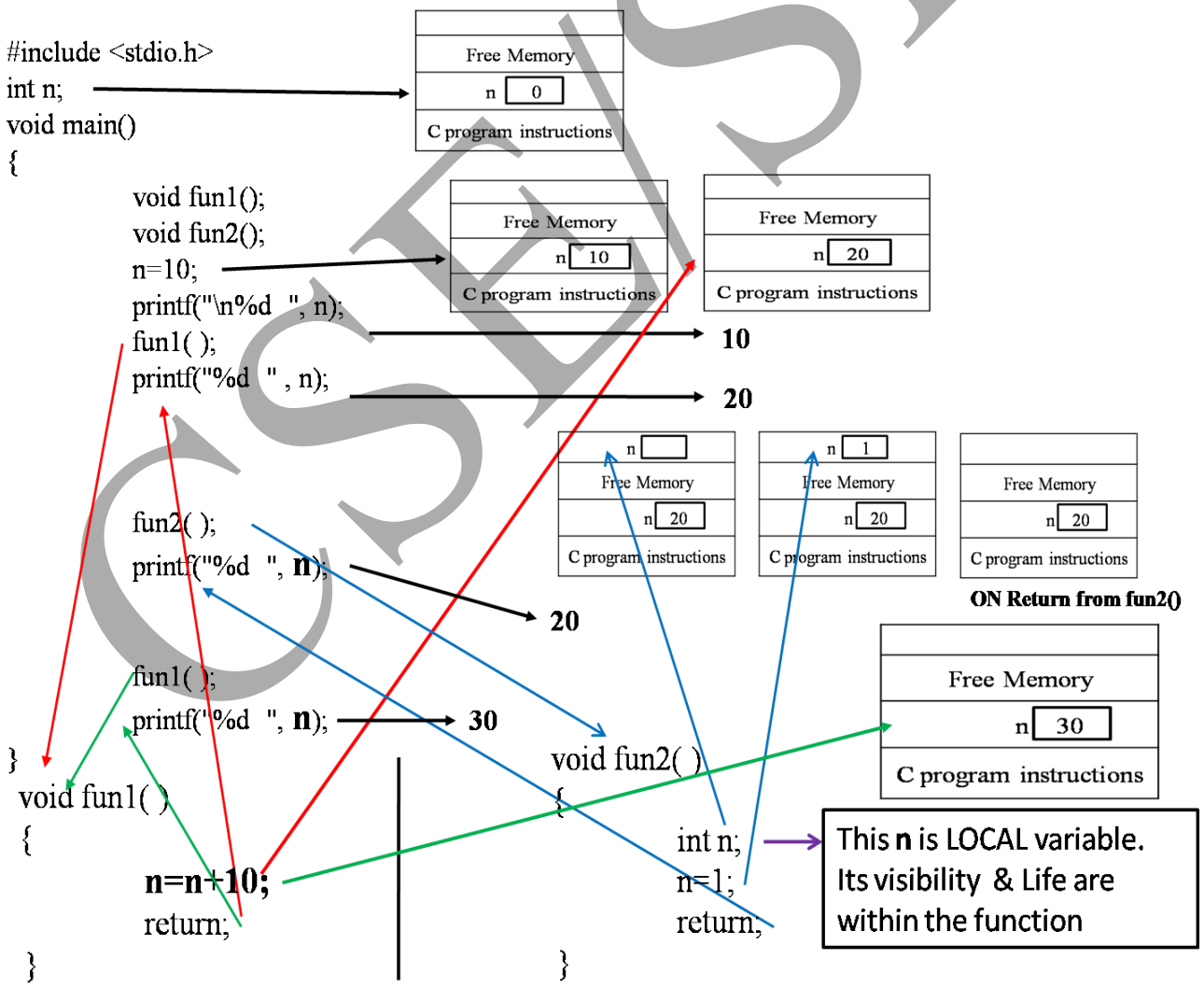
WHY?

Before Explanation let us OBSERVE the following Figure CAREFULLY.



Storage of a C program in memory

One thing is at least clear that in MEMORY, the place for LOCAL variables & GLOBAL variables are DIFFERENT.



Now if we modify the definition of fun2() function as follows:

```
void fun2()
{
      n=1;
      return;
 }
```

THEN the output will be

10 20 1 11

Because now **n** is not declared within the function so value **1** is assigned to global **n**. So printf() in main() after return from fun2( ) will print value **1**. Next, fun1( ) will be called from main( ) and in fun1() global **n** will be increased by **10**. That is **n** becomes 1 + 10 = **11** and after return from fun1(), printf() in main() will print value **11**.

What do you mean by Storage Class?

Answer:

Storage class refers to the **scope** and **longevity** of a variable.

The **scope** of a variable determines the portion of a program over which the variable is **active**.

The **longevity** refers to the period during which a variable remains **alive**.

T**o declare a variable** we need to mention **not only data type** but **also storage class**.

Example :

auto int n;

Here **auto** is a **storage class** and **int** is a **data type**.

Details about **storage Class** will be dealt in **Second Note**.