Storage Classes in C

Storage class in C helps us determine the default value, memory location, scope of variable and the lifetime of the variable.

Storage class is written before declaration.
ie, ______ datatype variablename;

Scope of a variable is of three types:
i) Block scope ii) Function scope iii) Program scope.

Eg:
```
    int a = 50 (iii)  <----  Program scope.
    int main ()
    { fun();   (i)                              Output
    int  a = 10; (iv)   <----  Function          30
                                scope.
    { int a = 20;  (ii)  <---- block scope.      20
       printf("%d", a);}                         50
    { printf("%d", a++);}                        10
    printf("%d", a);
    return 0;                       If int a = 10; is omitted,
    }
                                          Output:
                                           30
    void fun()        ⇓                    20
    { int a = 30;    (i). <- Function      50
                                 scope.
    printf("%d", a); }                     51
```

Date. No.

Based on default value, memory location, scope of variable and lifetime of variable, storage classes in C are of four types:

(i) Automatic:

Syntax: auto datatype variablename;

→ Default value = garbage value

→ Scope: local scope i.e, either block or function.

→ Lifetime: within the block or function depending where it has been declared.

→ Location: RAM in stack fragment.

(ii) Register:

Syntax = register datatype variablename;

→ Default value = garbage value

→ Scope: local scope in either block or function.

→ Lifetime: within the block or function where it is declared.

→ Location: CPU register.

We know, program is stored in harddisk, loaded into RAM and processing is done with CPU. So, while program execution, there's switching between RAM and registers.

Hence, for increasing efficiency, we store in registers to avoid switching.

We can't use pointer with register and we can't get the address of register variables with pointer.

Date. No.

iii) Static:

Syntax: static datatype variablename;

- Default value = 0 or null
- Location = RAM
- Scope = within the block.
- Lifetime = till the end of function

Eg:
```
void display();
void main()
{
    display();
    display();
}
void display()
{
    static int x;
    x += 10
    printf("\n X= %d", x);
}
```

Output:
```
10
20
```

Here,
on 1st call, when declared value is 0, 10 is added to 0 and printed. ie, 10.
On second call, since lifetime is throughout the program, the value when declared is 10, 10 is added and printed. Output = 20.

Here, if we print a in main, it will give error as x has been declared outside the block of function display.

We reduce the use of static as it holds memory throughout the program.

(iv) External

Syntax: extern datatype variablename;

→ Default value = 0
→ Location = RAM
→ ~~Sto~~ Scope = Global
→ Lifetime = till the end of program.

We reduce the use of extern as it holds memory throughout the program.
All global and only global variables come under extern category.

- We can use extern in function too.
- Declaration required before usage.
- It helps us links to files in same ~~proje~~ project.

To include another file, #include "filename"

Eg:

1.c

```
#include <stdio.h>
int x=10;
extern void display ();
void main ()
 { display; }
```

2.c

```
#include<stdio.h>
void display ()
{ extern int x;
  x++;
  printf ("Hello from 2.c ");
  printf ("X=%d , x);
}
```