Storage class in c and Scope of variables

- Storage classes are used to define scope and life time of a variable. There are four storage classes in C programming.
  - auto
  - extern
  - static
  - register

1) auto

- The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable.

```
#include <stdio.h>
void main(){
      int a=10;
      auto int b=10;//same like above
      printf("%d %d",a,b);
}
```

- Output:

```
10 10
```

2) register

- The register variable allocates memory in register than RAM.
- Its size is same of register size. It has a faster access than other variables.
- It is recommended to use register variable only for quick access such as in counter.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 register int a=10;
printf(“%d”,a)
}
```

Note: We can't get the address of register variable.

3) static

- The static variable is initialized only once and exists till the end of the program. It retains its value between multiple functions call.
- The static variable has the default value 0 which is provided by compiler.

```
#include <stdio.h>
void func() {
  static int i=0;//static variable
  int j=0;//local variable
  i++;
  j++;
  printf("i= %d and j= %d\n", i, j);
}
```

```
void main() {
  func();
  func();
  func();
}
```

Output:

```
i= 1 and j= 1
i= 2 and j= 1
i= 3 and j= 1
```

4) extern

- The extern variable is visible to all the programs. It is used if two or more files are sharing same variable or function.
- extern int counter=0;

```
#include<stdio.h>
void check();
extern int a=5;
void main()
{
    a+=4;
    check();
}
```

```
void check()
{
    ++a;
    printf(“a=%d\n”,a);
}
Output: a=10
```

Storage Classes	**Storage Place**	**Default Value**	**Scope**	**Life-time**
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of main program, May be declared anywhere in the program
static	RAM	Zero	Local	Till the end of main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within function

C - Scope Rules

- A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language −
  - Inside a function or a block which is called local variables.
  - Outside of all functions which is called global variables.
  - In the definition of function parameters which are called formal parameters.
- Let us understand what are local and global variables, and formal parameters.

Local Variables

- Variables that are declared inside a function or block are called local variables.
- They can be used only by statements that are inside that function or block of code.
- Local variables are not known to functions outside their own.
- The following example shows how local variables are used.
- Here all the variables a, b, and c are local to main() function.

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;
}
```

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.

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;
}
```

- A program can have same name for local and global variables but the value of local variable inside a function will take preference. Here is an example −

```
#include <stdio.h>

/* global variable declaration */
int g = 20;
int main () {
  /* local variable declaration */
  int g = 10;
  printf ("value of g = %d\n",  g);
  return 0;
}
```

- When the above code is compiled and executed, it produces the following result −
- value of g = 10

Formal Parameters

- Formal parameters, are treated as local variables with-in a function and they take precedence over global variables. Following is an example −

```
#include <stdio.h>
/* global variable declaration */
int a = 20;
int main () {
  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;
  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  return 0;
}
```

```c
 /* function to add two integers */
int sum(int a, int b) {
   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);
   return a + b;
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30
```

Initializing Local and Global Variables

- When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows −

Data Type	Initial Default Value
int	0
char	'\0'
float	0
double	0
pointer	NULL