Data types Revisited (Storage Classes)

- We have seen earlier that variables are declared with data type. But to fully define a variable, one needs to mention not only its type but also its storage class.

Data types.

Data type	Range	Bytes	Format
Signed char	-128 to +127	1	%c
unsigned char	0 to 255	1	%c
Signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	—	4	%ld
long unsigned int	—	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	—	8	%lf
long double	—	10	%Lf

A variable's storage class tells us :-

(a) Where the variable would be stored
(i.e Memory or CPU Registers)

(b) What will be the initial value of the variable, if initial value is not specifically assigned :
(i.e default initial value).

(c) What is the scope of the variable, i.e in which functions the value of the variable would be available.

(d) What is the life of the variable i.e how long would the variable exist.

There are 4 storage classes in C :-

(a) Automatic Storage Class (auto)

(b) Static " " (static)

(c) Register " " (register)

(d) External " " (extern)

---

1. Automatic storage Class

- Storage — Memory
- Default Initial value — An unpredictable value, or called a garbage value.
- Scope — Local to the block in which the variable is defined.
- Life — Till the control remains within the block in which the variable is defined.

Eg (1)
```
main()
{
auto int i;
printf("%d", i);
}
```
→ 12234 (garbage value) ∵ not initialized

Eg (2)
```
main()
{
auto int i=1;
{
auto int i=2;
{
auto int i=3;
printf("%d", i);
}
printf("%d", i);
}
printf("%d", i);
}
```
O/P → 3 2 1

(2) Register Storage Class

- Storage — CPU Registers
- Default Initial value — Garbage Value
- Scope — Local to the block in which the variable is defined.
- Life — Till the control remains within the block in which variable is defined.

⇒ A value stored in CPU register can always be accessed faster than the one that is stored in memory. Therefore, if a variable is used at many places in a program, it is better to declare it as a register. like → loop counters.

Eg
```
main()
{
    register int i;
    for (i=1; i<=10; i++)
        printf("%d", i);
}
```

⇒ Even if it is declared as register, it may not be stored in CPU registers because CPU Registers are limited. So in that case, variable will be used as auto.

3. Static Storage Class

- Storage — Memory
- Default Initial value — Zero
- Scope — Local to the block in which the variable is defined.
- Life → Value of the variable persists between different function calls.

⟹ Static variables are the variables that retain its value between function calls.
Means when control will be returned from function, it will retain its value when next time that function is called.

---

eg. Diff. b/w Automatic / static storage class

```
main()
{
  increment();
  increment();
  increment();
}
increment()
{
  auto int i=1;
  printf("%d", i);
  i=i+1;
}
```

O/P

```
1
1
1
```

```
main()
{
  increment();
  increment();
  increment();
}
increment()
{
  static int i=1;
  printf("%d", i);
  i=i+1;
}
```

O/P

```
1
2
3
```

In this program, when variable i is auto, each time increment() is called, it is re-initialized to 1. Even if i becomes 2 in that function call that value will be lost.

On the other hand, if i is static, it is initialized to 1 only once. It is never initialized again.
During first call to increment(), i is incremented to 2.
Because i is static, this value persists.
The next time increment() is called, i is not reinitialized to 1. So it will print 2. — — — and so on.

---

(4) External Storage Class.

- Storage — Memory
- Default Initial value — Zero.
- Scope — Global
- Life — As long as the program's execution doesn't come to an end.

Eg:
```
int x=21;
main()
{ extern int y;
  printf("%d %d", x, y);
}
int y=31;
```

Here x and y both are external or global variables.
and their scope will be till end of program.