Storage Class

C Storage Classes

Scope of variables

Lifetime of variables

Local and Global variables

Static and register variables

- CPU Registers – collection of bits
- Closer to CPU
- Faster access

- Memory – collection of bits
- slower

Type and Storage Class

Properties of variables – Type and storage class.

Type and Storage Class

Type

Data type of a variable

storage class

Scope, visibility and lifetime of a variable

Types of Storage Class

There are 4 types of storage class:

external X
static
register
local Variable/automatic
Global variables

Automatic and local variables

The variables declared inside a block are automatic or local variables.

The local variables exist only inside the block in which it is declared.

```
#include <stdio.h>
int main(void) {
  int j;
  for (int i = 0; i < 5; ++i) {
     int k;
     printf("C programming");
  }
printf("%d", i);
  return 0;
}
```

Error in the code -

Error: undeclared identifier i.

It's because i is declared inside the for loop block.

Outside of the block, it's undeclared.

```
int main() {
    int n1; // n1 is a local variable to main()
}
void func() {
   int n2;  // n2 is a local variable to func()
}
```

n1 is local to main() and
n2 is local to func().

Cannot access the n1 variable inside func() as it only exists inside main().

Cannot access the n2 variable inside main() as it only exists inside func().

Global Variable

Variables that are declared outside of all functions are known as external or global variables.

They are accessible from any function inside the program.

```
#include <stdio.h>
void display();
int n = 5;
// global variable

int main()
{
   ++n;
   display();
……..
   return 0;
print n …
}
```

```
void display()
{
   int i;
   ++n;
   printf("n = %d", n);
}
```

```
#include <stdio.h>
void display();
int n = 5;
// global variable

int main()
{
    ++n;
    display();
    return 0;
}
```

```
void display()
{
    ++n;
    printf("n = %d", n);
}
```

Output

n = 7

Global variable / External variable

Assume, a global variable is declared in file1

If you try to use that variable in a different file file2, there will be compilation problem

keyword **extern** is used in file2 to indicate that the external variable is declared in another file.

Register variables

The register keyword is used to declare **register** variables.

register int I;

Register variables were supposed to be faster than local variables.

Rare chance that using register variables will make your program faster with todays compiler

Unless you are working on embedded systems where you know how to optimize code for the given application, there is no use of register variables.

Static variables

A static variable is declared by using the **static** keyword.

static int i;

The value of a static variable persists until the end of the program.

```
#include <stdio.h>
void display();

int d = 1;

int main()
{
            .....
   display(); - 8 times
……
            print(d);
            print(c);  XXX
}

void display()
{
static      int c = 1;

            d = d + 1
            print(d);
c += 5;
   printf("%d  ",c);
}
```

```c
#include <stdio.h>
void display();

int main()
{
   display();
   display();
}
void display()
{
   static int c = 1;
   c += 5;
   printf("%d  ",c);
}
```

Output

6 11

Explanation

During the first function call, the value of c is initialized to 1. Its value is increased by 5. Now, the value of c is 6, which is printed.

During the second function call, c is not initialized to 1 again. It's because c is a static variable. The value c is increased by 5. Now, its value will be 11, which is printed.