Content 36

Storage Classes In C Auto, Extern Static & Register Storage Class in C

**Declaration:-** Telling the compiler about the variable. (no Space is reserved).

**Definition:-** declaration + Space Reservation.

***"A storage class defines scope, default initial value, and a lifetime of a variable."***

A Storage Class Defines Following attributes about a variable in C;

1. **Scope:-**  Were this Variable be available or Existance.
2. **Default Initial value:**- Here int b; is not initialized which system will print or display an deault value(/garbage Value).
3. **Lifetime:**- Life of that Variable.

Here, scope refers to the variable's availability at places. The initial default value refers to the value present in the variables as default before being initialized, and the lifetime refers to the variable's duration of life.

Now, as we are familiar with the basic concept of storage classes, let us move to its types. There are four types of storage classes depending upon the type of variables they store. The following are the type of storage classes. You can guess their stored variables by their names:

* Automatic Variables
* External Variables
* Static Variables
* Register Variables

**Note:** Each storage class differs, with respect to scope, default initial value, and a lifetime of a variable.

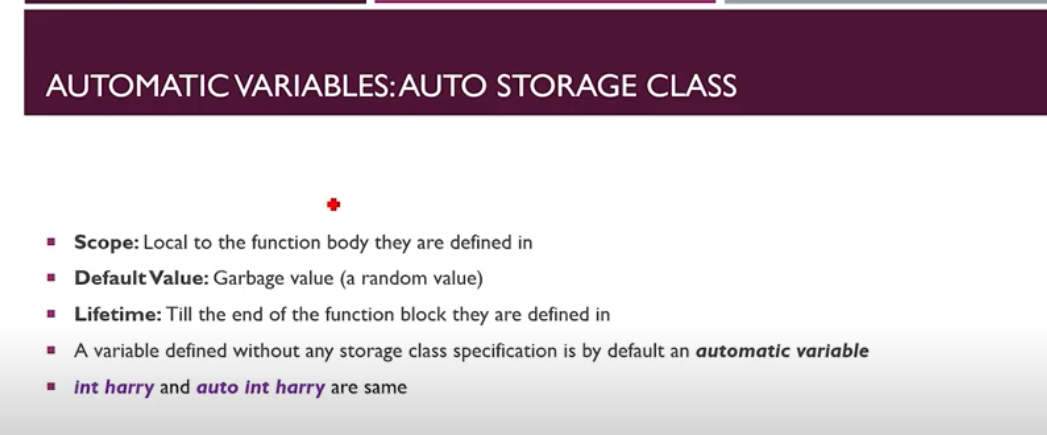
#### Auto Storage Class: (There working is like Local Variables)

Variables being formed in a function and whose storage class has not been defined initially fall in this category automatically. Its scope is minimum as it can only be accessed inside the function it is initialized in. No other function can access it. Until the variable has been assigned some value, it stores garbage value as default. Their lifetime depends upon the function block's length as the lifetime is until the function block's end.

int a;

auto int a;

//Both are the same.



**Code1 for Auto Variable/Auto Storage Class:**

1. #include<stdio.h>
2. int func(int a,int b)
3. {   auto int sum;
4. return a+b;
5. }
6. int main()
7. {   int sum=func(3,2);
8. printf("My sum is %d",sum);
9. return 0;
10. }

**Output:**

My sum is: 8

#### If I comment out line no 5 it would not affect the output. But if also comment out 7 it would be.

**Code1 for Auto Variable/ Auto Storage Class:** I added at line 1

1. #include <stdio.h>
2. int sum;
3. int func(int a, int b)
4. {
5. // auto int sum;
6. return a + b;
7. }
8. int main()
9. {
10. int sum = func(3, 2);
11. printf("My sum is %d", sum);
12. return 0;
13. }

**Output:**

My sum is: 5

So it is giving reference to the Local variable. If I uncomment line no 5 then it will give output as 5.

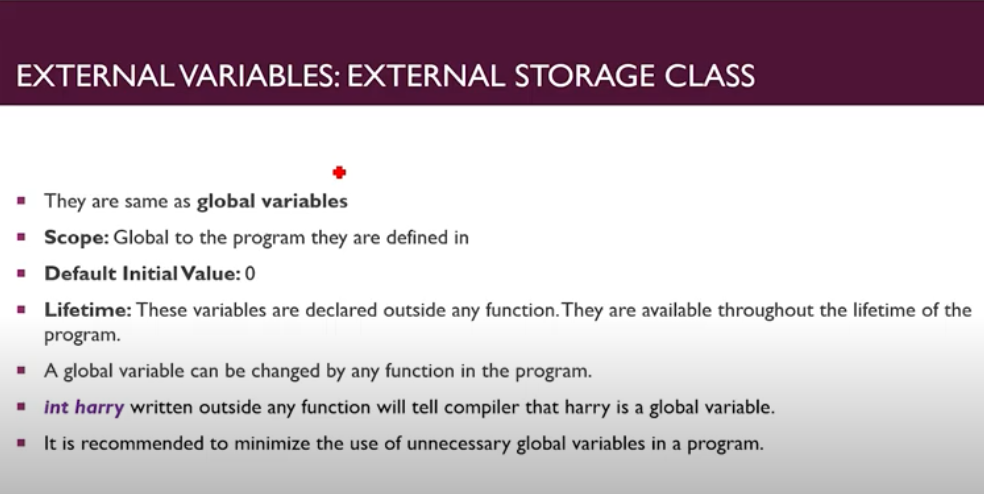
Global will not have presidency until there is Local Variable.

#### External Storage Class: (This are same as global Variable)

These sorts of variables are defined outside the function, hence can be used inside any function, meaning that they can be used globally. Their initial value is set to 0. As they can be used throughout the program, so their lifetime equals the lifetime of the program. Too many global variables in a program can cause security issues and also are not usually recommended.

extern int a;

#### “Take the minimize use of global Variable because it hold space alongthe lifetime of program”.



**Code For Understanding the External Variable/ External Storage class:**

1. // For Understanding External  Vsrisbles
2. #include <stdio.h>
3. int sum;
4. int func(int a, int b)
5. {
6. // auto int sum;
7. return a + b;
8. }
9. int main()
10. {
11. // int sum = func(3, 2);
12. printf("My sum is %d", sum);
13. return 0;
14. }

**Output:**

My sum is: 0

I had commented out both the lines (6 and 14) so here Global Variable takes the presidency. And also prints its default initial value as 0.

#### Extern Keyword:

Using the extern keyword, we inform our compiler that the variable is already declared at some other place. By doing so, we can use the same variable with the same space, without allocating its new memory and accessing the same variable in some other file. Its syntax is simple as we have to use the extern keyword, and it will automatically access it from the other file.

first.c

int main()

{ int a=10;

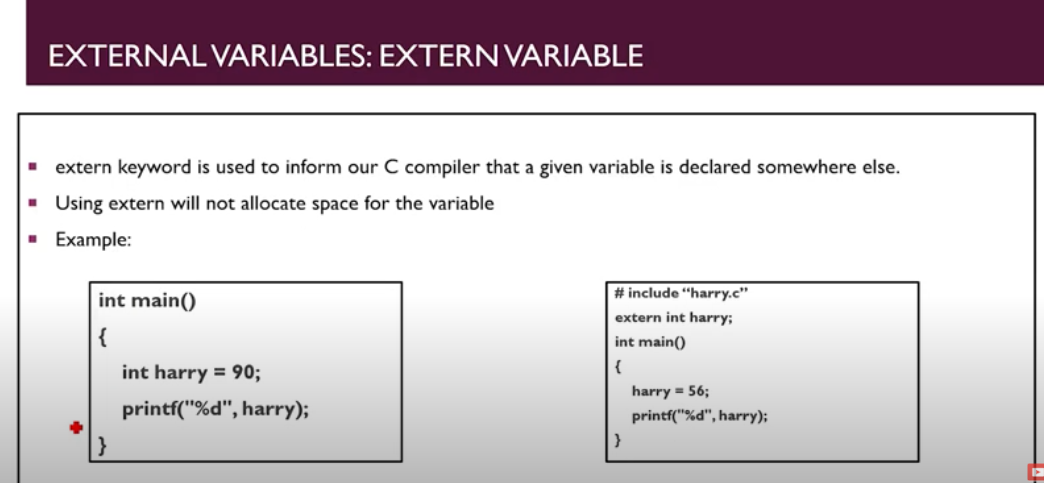
Printf(“%d”, a); }

#include “first.c”

Exter int first;

int main() { int a=34;

printf(“%d”, a); }



#### Code For understanding Extern Keyword:

1. #include <stdio.h>
2. int sum;
3. int func(int a, int b)
4. {   extern int sum;
5. // auto int sum;
6. return a + b;
7. }
8. int sum =10;
9. int main()
10. {
11. // int sum = func(3, 2);
12. printf("My sum is %d", sum);
13. return 0;
14. }

Here at line 8 I had declared a Global Variable But I am using it as Local Variable by Extern keyword.

**Output:**

My sum is 10

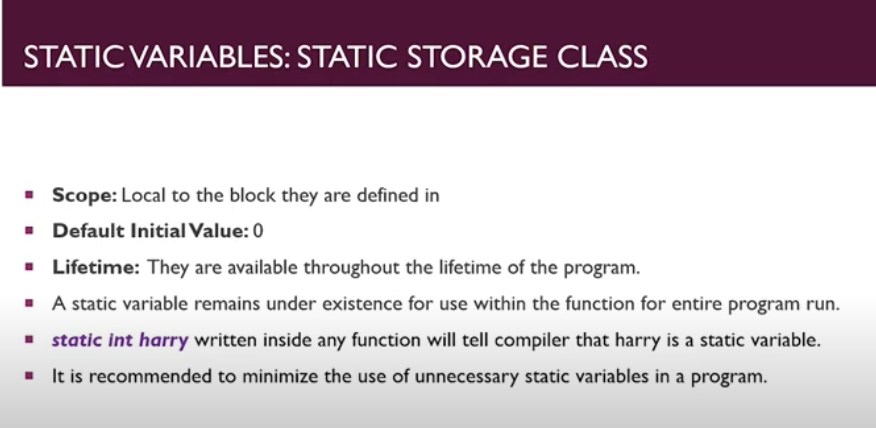
But here if line (11 and 5) are uncomment then the program will say which one I have to execute because here all will take presidency as Local Variables and it will put an error.

And If I uncomment line 11 then which declaration would compiler Except? Here compiler will give presidency to line 11 as it was an actual local Variable and line 8 is global variable but we had made it as equal to Local Variable by extern keyword.

#### Static Storage Class:

Static variables are a little bit technical as their lifetime is throughout the program, but their scope is limited to the function they are initialized in. It comes in handy when we are changing their value in the program as the program will store the new value, overwriting the previous one. Their initial default value is 0, and their syntax is very easy as we just have to use the Static keyword during initialization.

static int a;



“Minimize the use of static Variables as it also takes lifetime space as Global variable”.

#### Code for understanding Static Storage Class:

int func2(int c)

{

    static int myvar=1;

    myvar++;

    printf("\nThe value is: %d", myvar);

    return myvar;

}

int main()

{

    int myvar;

    myvar = func2(sum);

    myvar = func2(sum);

    myvar = func2(sum);

    return 0;

}

**Output:**

The value is: 2

The value is: 3

The value is: 4

#### Register Storage Class:

It is very similar to the Auto storage class as its scope is limited to the function it is defined in, the initial default value is 0, and lifetime is till the end of the function block. Now the major difference between it and the others is that it requests the CPU's register memory instead of the local memory for fast access. It is usually used for the programs that need to be accessed faster than the other or used frequently.

**Syntax:** register int a;;

**This Completely in the hands of computer to allow you to give the Register Storage class.**

**So there is 100 on 1 chance of allowment.**

