LECTURE NOTE 18

STORAGE CLASSES

To completely define a variable one needs to mention its type along with its **storage class**. In other words we can say not only variables have a data type but also they have ‘storage classes.

Till now the storage class of any variable is not mentioned because storage classes have defaults. If someone doesn’t specify the storage class of a variable during its declaration, the compiler assumes a storage class depending upon the situation in which the variable is going to be used. Therefore we can now say every variable have certain default storage class.

Compiler identifies the physical location within the computer where the string of bits which represents the variable’s values are stored from the variable name itself.

Generally there are two kinds of locations in a computer where such a value can be present, these are Memory and CPU registers. The storage class of a particular variable determines in which of the above two locations the variable’s value is stored.

There are four properties by which storage class of a variable can be recognized.These are scope, default initial value, scope and life.

A variable’s storage class reveals the following things about a variable

(i) Where the variable is stored.
(ii) What is the initial value of the variable if the value of the variable is not specified?
(iii)What is the scope of the variable (To which function or block the variable is available).
(iv) What is the life of particular variable (Up to what extent the variable exists in a program).

There are four storage classes in C, these are Automatic, Static, Register and External. The keywords auto, static, registers and exter are used for the above storage classes respectively.

We can specify a storage class while declaring a variable. The

general syntax is

storage_class datatype variable_name;

The following table shows different properties of a variable which according to its storage class.

Properties / Storage Class	**Storage**	**Default Initial Value**	**Scope**	**Life**
Automatic	Memory	Garbage Value	Local to the block in which the variable is defined	Till the control remains within the block in which the variable is defined
Register	CPU Registers	Garbage Value	Local to the block in which the variable is defined	Till the control remains within the block in which the variable is defined
Static	Memory	Zero	Local to the block in which the variable is defined	Value of the variable continues to exist between different function calls
External	Memory	Zero	Global	Till the program's execution doesn't come to an end

Automatic Storage Class

Syntax to declare automatic variable is:

auto datatype variablename;

Example:

auto int i;

Features of Automatic Storage Class are as follows

Storage: Memory

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 the variable is defined

All the variables which are declared inside a block or function without any storage class specifier are automatic variables. So by default every variable is automatic variable. We can use auto keyword to declare automatic variables but generally it is not done. Since automatic variables are known inside a function or block only, we may have variables of same name in different functions or blocks without any doubt. Look at the following two functions:

```
void fun1()

{

  int x,y;

/* Some statements*/

}
```

```
void fun1()

{

   auto int x,y;

   /*Some statements*/

   }
```

In the above two functions declaration statements are equivalent as both declare variables x and y as automatic variables.

The following program illustrates the work of automatic variables.

```
void test();
void main()
{
test();
test();
test();
}
 void test()
{
auto int k=10;
printf("%d\n",k);
k++;
}
```

Output:

```
10
10
10
```

In the above program when the function test() is called for the first time ,variable k is created and initialized to 10. When the control returns to main(), k is destroyed. When function test() is called for the second time again k is created , initialized and destroyed after execution of the function. Hence automatic variables came into existence each time the function is executed and destroyed when execution of the function completes.

Register Storage Class

Syntax to declare register variable is:

register datatype variablename;

Features of Register Storage Class are as follows:

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 the variable is defined

Register storage class can be applied only to automatic variables. Its scope, life and default initial value are same as that of automatic variables. The only difference between register and automatic variables is the place where they are stored. Automatic variables are stored in memory but register variables are stored in CPU registers. Registers are small storage units present in the processor. The variables stored in registers can be accessed much faster than the variables stored in registers. Hence the variables which are frequently used in a program can be assigned register storage class for faster execution. For example loop counters are declared as register variables, which are defined as follows:

```
int main()
{
 register int a;
for(a=0;i<50000;i++)
printf("%d\t",a);
return 0;
 }
```

In the above program, variable **a** was used frequently as a loop counter so the variable a is defined as a register variable. Register is a not a command but just a request to the compiler to allocate the memory for the variable into the register. If free registers are available than memory will be allocated in to the registers. And if there are no free registers then memory will be allocated in the RAM only (Storage is in memory i.e. it will act as automatic variable).

Every type of variables can be stored in CPU registers. Suppose the microprocessor has 16 bit registers then they can't hold a float or a double value which requires 4 and 8 bytes respectively. But if you use register storage class for a float or double variable then you will not get any error message rather the compiler will treat the float and double variable as be of automatic storage class(i.e. Will treat them like automatic variables).

Static Storage Class

Syntax to declare static variable is:

static datatype variablename;

Example:

static int i;

Features of Static Storage Class are as follows

Storage: Memory

Default Initial Value: Zero

Scope: Local to the block in which the variable is defined

Life: Value of the variable continues to exist between different function calls

Now look at the previous program with k is declared as static instead of automatic.

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

```
static int k=10;
 printf("%d\n",k);
 k++;
}
```

 Output:

```
10
11
12
```

Here in the above program the output is 10, 11, 12, because if a variable is declared as static then it is initialized only once and that variable will never be initialized again. Here variable k is declared as static, so when test() is called for the first time k value is initialized to 10 and its value is incremented by 1 and becomes 11. Because k is static its value persists. When test() is called for the second time k is not reinitialized to 10 instead its old value 11 is available. So now 11 is get printed and it value is incremented by 1 to become 12. Similarly for the third time when test() is called 12(Old value) is printed and its value becomes 13 when executes the statement 'k++;'

So the main difference between automatic and static variables is that static variables are initialized to zero if not initialized but automatic variables contain an unpredictable value(Garbage Value) if not initialized and static variables does not disappear when the function is no longer active , their value persists, i.e. if the control comes back to the same function again the static variables have the same values they had last time.

General advice is avoid using static variables in a program unless you need them, because their values are kept in memory when the variables are not activc which means they occupies space in memory that could otherwise be used by other variables.

External Storage Class

Syntax to declare static variable is:

extern datatype variablename;

 Example:

```
extern int i;
```

Features of External Storage Class are as follows

Storage: Memory

Default Initial Value: Zero

Scope: Global

Life: Till the program's execution doesn't come to an end

External variables differ from automatic, register and static variables in the context of scope, external variables are global on the contrary automatic, register and static variables are local. External variables are declared outside all functions, therefore are available to all functions that want to use them.

If the program size is very big then code may be distributed into several files and these files are compiled and object codes are generated. These object codes linked together with the help of linker and generate ".exe" file. In the compilation process if one file is using global variable but it is declared in some other file then it generate error called undefined symbol error. To overcome this we need to specify global variables and global functions with the keyword extern before using them into any file. If the global variable is declared in the middle of the program then we will get undefined symbol error, so in that case we have to specify its prototype using the keyword extern.

So if a variable is to be used by many functions and in different files can be declared as external variables. The value of an uninitialized external variable is zero. The declaration of an external variable declares the type and name of the variable, while the definition reserves storage for the variable as well as behaves as a declaration. The keyword **extern** is specified in declaration but not in definition. Now look at the following four statements

1. auto int a;
2. register int b;
3. static int c;
4. extern int d;

Out of the above statements first three are definitions where as the last one is a declaration.
Suppose there are two files and their names are File1.c and File2.c respectively. Their contents are as follows:

File1.c
int n=10;
void hello()
{
printf("Hello");
}

```
File2.c
extern int n;
extern void hello();
void main()
{
 printf("%d", n);
hello();
}
```

In the above program File1.obj+ File2.obj=File2.exe and in File2.c value of n will be 10.

Where to use which Storage Class for a particular variable:

We use different storage class in appropriate situations in a view to

1. Economise the memory space consumed by the variables
2. Improve the speed of execution of the program.

The rules that define which storage class is to be executed when are as follows:

(a) Use static storage class if you want the value of a variable to persist between different function calls.

(b) Use register storage class for those variables that are being used very often in a program, for faster execution. A typical application of register storage class is loop counters.

(c) Use extern storage class for only those variables that are being used by almost all the functions in a program, this avoids unnecessary passing of these variables as arguments when making a function call.

(d) If someone do not have any need mentioned above, then use auto storage class.

Exercise:

[A] Answer the following Questions.

(a) What do you mean by storage class of a variable?

(b) Why register storage class is required?

(c) How many storage classes are there in c and why they are used?

(d) Why someone needs static storage variable in a program?

(e) Which storage class is termed as default storage class, if the storage class of a variable is not mentioned in a program? Explain its working with the help of a suitable example.

(f) Can you store a floating point variable in CPU Registers, Explain?

(g) What do you mean by scope and life of a variable?

(h) Justify where to use which storage class.

(i) What do you mean by external variable and where it is defined?

(j) What is the difference between auto and static variables in a program?

(k) What do you mean by definition and declaration of a variable?

[B] What is the output of the following programs:

(a)
```
void main()
{
static int a=6;
printf("\na=%d",a--);
 if(a!=0)
main();
}
```

(b)
```
void main()
{
 int i,j;
for(i=1;i<5;i++)
{
j=fun1(i);
printf("\n%d",j);
}
}
 int fun1(int a)
```

```
{
 static int b=2;
   int c=3;
   b=a+b;
   return(a+b+c);
  }
```

(c)

```
void main()
{
fun1();
fun1();
}
void fun1()
{
 auto int x=0;
  register int y=0;
 static int z=0;
 x++;
 y++;
 z++;
 printf("\n%d%d%d",x,y,z);
}
```

```
(d) int a=10;
void main()
    {
    int a=20;
    {
    int a=30;
printf("%d\n",a);
     }
    printf("%d",a);
    }
```

[C] State whether the following statements are True or False:

(a) The register storage class variables cannot hold float values.

(b) The value of an automatic storage class variable persists between various function invocations.

(c) If the CPU registers are not available, the register storage class variables are treated as static storage class variables.

(d) The default value for automatic variable is zero.

(e) If a global variable is to be defined, then the extern keyword is necessary in its declaration.
(f) If we try to use register storage class for a float variable the compiler will flash an error message.

(g) Storage for a register storage class variable is allocated each time the control reaches the block in which the variable is present.

(h) An extern storage class variable is not available to the functions that precede its definition, unless the variable is explicitly declared in the above functions.

(i) The life of static variable is till the control remains within the block in which it is defined.

(j) The address of a register variable is not accessible.

[D] Following program calculates the sum of digits of the number 25634. Go through it and find out why is it necessary to declare the storage class of the variable sum as static.

```
#include<stdio.h>

 void main()

{ int m;

m=sumdigit(25634);

 printf("\n%d",m);

}

int sumdigit(int n)

{

static int sum;

 int p,q;

p=n%10;

q=(n-p)/10;

sum=sum+p;

if(q!=0) sumdigit(q);

else

return(sum);
 }
```