Programming and Problem Solving through C Language
O Level / A Level

Chapter - 7 : Storage Classes

Storage Class

A storage class defined the scope, visibility and a life time(extent) of a variable.

A storage class is used to describe the following things:

- The variable scope.
- The location where the variable will be stored.
- The initial value of a variable.
- A lifetime of a variable.
- Who can access a variable?

Scope of a Variable

- The scope of a variable determines over what part(s) of the program a variable is actually available for use (active).
- Longevity: It refers to the period during which a variables retains a given value during execution of a program (alive).
- Local (internal) variables are those which are declared within a particular function.
- Global (external) variables are those which are declared outside any function.

Scope of a Declaration

Scope of a declaration is the region of C program text over which that declaration is active.

- Top-level identifiers – Extends from declaration point to end of file.
- Formal parameters in functions – Extends from declaration point to end of function body.
- Block/function (local) identifiers – Extends from declaration point to end of block/function.

Extent (Life Time of Variable)

- The extent of an object is the period of time that its storage is allocated.
- An object is said to have static extent when it is allocated storage at or before the beginning of program execution and the storage remains allocated until program termination.
- All functions have static extent, as do all variables declared in top-level declarations and variables declared with the static qualifier.
- Formal parameters and variables declared at the beginning of a block or function have local extent (dynamic, de-allocated on block/function exit).

Storage Class Types

- Auto
- Extern
- Static
- Register

Storage Specifier	**Storage**	**Initial Value**	**Scope**	**Life Time**
auto	Stack	Garbage	Within Block	End of Block
extern	Data Segment	Zero	Global, Multiple Files	Till end of program
static	Data Segment	Zero	Within Block	Till end of Program
register	CPU Register	Garbage	Within Block	End of Block

Automatic variables

- Are declared inside a function in which they are to be utilized.
- Are declared using a keyword auto. eg. **auto int number**;
- Are created when the function is called and destroyed automatically when the function is exited. These variables are private (local) to the function in which they are declared.
- Variables declared inside a function without storage class specification is, by default, an automatic variable.
- Auto is the default storage class for all local variables.

```
{
        int count;
        auto int month;
}
```

- The example above defines two variables with the same storage class.
- Auto can only be used within functions, i.e. local variables.
- During recursion, the nested variables are unique auto variables

```
#include <stdio.h>
int main( )
{
   auto int j = 1;
   {
         auto int j= 2;
         {
              auto int j = 3;
              printf ( " %d ", j);
         }
         printf ( "\t %d ",j);
   }
   printf( "%d\n", j);
}
```

The features of automatic variables are

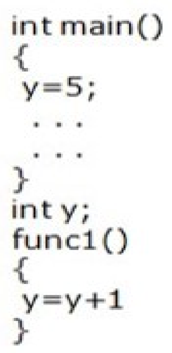
- Storage – memory.
- Default initial value - an unpredictable value, which is often a garbage value.
- Scope - local to the block in which the variable is defined.
- Life - till the control remains within the block variable is defined.

External Variables

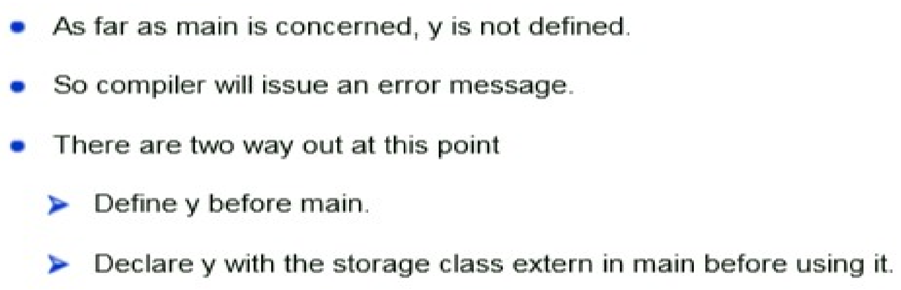
- These variables are declared outside any function.
- These variables are active and alive throughout the entire program.
- Also known as global variables and default value is zero.
- Unlike local variables they can be accessed by any function in the program.
- In case local variable and global variable have the same name, the local variable will have precedence over the global one.
- Sometimes the keyword **extern** is used to declare these variables.
- **Extern** is used to give a reference of a global variable that is visible to all the program files.
- It is visible only from the point of declaration to the end of the program.
- The extern variable cannot be initialized, as all it does is, point the variable name at a storage location that has been previously defined.
- When a programmer has multiple files and defines a global variable or function, which will be used in other files also, then extern will be used in another file to give reference of defined variable or function.
- **Extern** is used to declare a global variable or function in another file.

The features of external storage class variable are as follows:

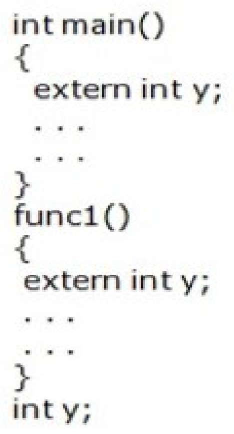
- Storage — memory.
- Default initial value — zero.
- Scope — global.
- Life — as long as the program execution does not end.



```
int main()
{
 y=5;
  . . .
  . . .
}
int y;
func1()
{
 y=y+1
}
```



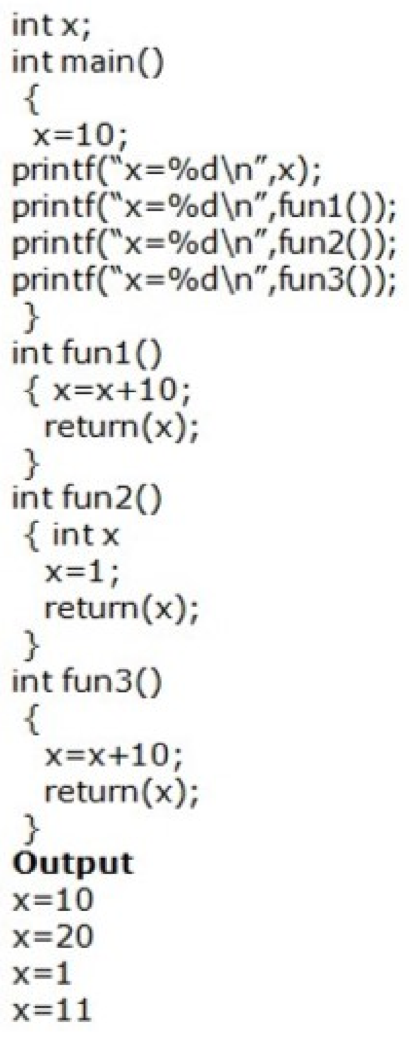
- As far as main is concerned, y is not defined.
- So compiler will issue an error message.
- There are two way out at this point
  - ➤ Define y before main.
  - ➤ Declare y with the storage class extern in main before using it.



```
int main()
{
  extern int y;
  . . .
  . . .
}
func1()
{
 extern int y;
 . . .
 . . .
}
int y;
```

- Note that extern declaration does not allocate storage space for variables.

Global Variable Example



```
int x;
int main()
 {
  x=10;
printf(“x=%d\n”,x);
printf(“x=%d\n”,fun1());
printf(“x=%d\n”,fun2());
printf(“x=%d\n”,fun3());
 }
int fun1()
 { x=x+10;
   return(x);
 }
int fun2()
 { int x
   x=1;
   return(x);
 }
int fun3()
 {
   x=x+10;
   return(x);
 }
```

Output

```
x=10
x=20
x=1
x=11
```