C - Storage Classes

Presented by,

LIVIN MATHEW S3 CSB ROLL NO.: 41

SOOLU THOMAS S3 CSB ROLL NO.: 60

FEW TERMS

1. **Scope:** the scope of a variable determines over what part(s) of the program a variable is actually available for use (active).

2. **Local (internal) variables:** are those which are declared within a particular function.

3. **Global (external) variables:** are those which are declared outside any function.

Scope

A scope in any programming is the region of the program where a defined variable can have its existence and beyond that variable can not be accessed. There are 3 places where variables can be declared in a C programming Language;

1. Inside a function - called **Local Variables**.
2. Outside of **all** functions - called **Global Variables.**
3. In the definition of function parameters - called **Formal Parameters**.

LOCAL VARIABLES

1. The variables which are declared inside a function or a block is called **Local Variables.**
2. They can be used only within the function in which it is declared / initialized.
3. These variables cannot be accessed by any other functions

```
void main()
{
    //Declaration of local variables
    int a, b;
    float c;

    //Initialization of Local Variables
    a=3;
    b=5;
    c=3.95;

    printf(“%d%d%d”, a,b,c);
}
```

GLOBAL VARIABLES

```
#include <stdio.h>

/* global variable declaration */
int g;

int main ()
{
  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

FORMAL PARAMETERS

```
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main ()
{
  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);

  return 0;
}

/* function to add two integers */
int sum(int a, int b)
{
    printf ("value of a in sum() = %d\n",  a);
    printf ("value of b in sum() = %d\n",  b);

    return a + b;
}
```

STORAGE CLASS

A storage class defines the scope (visibility) and lifetime of variables and/or of functions in a program

Topics

- Automatic variables
- External variables
- Static variables
- Register variables

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.
- This variable are therefore 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.
- By Default, garbage Value is assigned

EXAMPLE PROGRAM # 1

```
int main()
{ int m=1000;
  function2();
  printf("%d\n",m);
}
function1()
{
  int m = 10;
  printf("%d\n",m);
}
function2()
{  int m = 100;
   function1();
   printf("%d\n",m);
}
```

Output

10

100

1000

STATIC VARIABLES

- The value of static variables persists until the end of the program.

- It is declared using the keyword ***static*** like

  static int x;

  static float y;

- It may be of **external or internal** type depending on the place of the declaration.

- Static variables are initialized only once, when the program is compiled.

INTERNAL STATIC VARIABLE

- Are those which are declared inside a function.
- Scope of *Internal static* variables extend upto the end of the program in which they are defined.
- *Internal static* variables are almost same as *auto* variable except they remain in existence (alive) throughout the remainder of the program.
- *Internal static* variables can be used to retain values between function calls.

Example of Internal Static Variable

- Internal static variable can be used to count the number of calls made to function. eg.

```
int main()
{
  int I;
  for(i =1; i<=3; i++)
   stat();
 }
void stat()
{
  static int x=0;
  x = x+1;
  printf("x = %d\n",x);
}
```

Output

x=1

x=2

x=3

The register Storage Class

- stored in one of the machine's register
- declared using register keyword
- eg. **register int count;**

- Since register access are much faster than a memory access keeping frequently accessed variables in the register lead to faster execution of program.

- Since only few variable can be placed in the register, it is important to carefully select the variables for this purpose. However, C will automatically convert register variables into non register variables once the limit is reached.

- Don't try to declare a global variable as register. Because the register will be occupied during the lifetime of the program.

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 are 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*** used to declare these variable.
- It is visible only from the point of declaration to the end of the program.

External variable (examples)

```
int number;
float length=7.5;
main(){
  . . .
  . . .
}
funtion1(){
  . . .
  . . .
}
funtion2(){
  . . .
  . . .
}
```

The variable number and length are available for use in all three function

```
int count;
main(){
 count=10;
  . . .
  . . .
}
funtion(){
 int count=0;
 . . .
 . . .
 count=count+1;
}
```

When the function references the variable count, it will be referencing only its local variable, not the global one.

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);
 }
```

Once a variable has been declared global any function can use it and change its value. The subsequent functions can then reference only that new value.

Output

x=10

x=20

x=1

x=30

External declaration(examples)

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

Note that extern declaration does not allocate storage space for variables

Multifile Programs and *extern* variables

file1.c

```
int main()
{
    extern int m;
    int i;
    . . .
    . . .
}
function1()
{
     int j;
     . . .
     . . .
}
```

file2.c

```
int m;
function2()
{
    int i;
    . . .
    . . .
}
function3()
{
     int count;
     . . .
     . . .
}
```

Multifile Programs and *extern* variables

file1.c

```
int m;
int main()
{
    int i;
    . . .
    . . .
}
function1()
{
     int j;
     . . .
     . . .
}
```

file2.c

```
extern int m;
function2()
{
    int i;
    . . .
    . . .
}
function3()
{
     int count;
     . . .
     . . .
}
```

Thank you... :)

**** Soolu Thomas & Livin Mathew ****