Storage class in C

Topics

- Automatic variables
- External variables
- Static variables
- Register variables
- Scopes and longevity of above types of variables.

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. Longevity: it refers to the period during which a variables retains a given value during execution of a program(alive)
3. Local(internal) variables: are those which are declared within a particular function.

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

Automatic variables

- Are declare 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.

Example program

```
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
```

Few observation abt auto variables

- Any variable local to main will normally live throughout the whole program, although it is active only in main.
- During recursion, the nested variables are unique **auto** variables.
- Automatic variables can also be defined within blocks. In that case they are meaningful only inside the blocks where they are declared.
- If automatic variables are not initialized they will contain garbage.

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()
{. . .
 . . .
}
funtion1()
{. . .
 . . .
}
```

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

```
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
  1. Define y before main.
  2. Declare y with the storage class ***extern*** in main before using it.

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;
     . . .
     . . .
}
```

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 there 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.

Examples (internal static)

- 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
```

External static variables

- An external static variable is declared outside of all functions and is available to all the functions in the program.
- An external static variable seems similar simple external variable but their difference is that static external variable is available only within the file where it is defined while simple external variable can be accessed by other files.

Static function

- Static declaration can also be used to control the scope of a function.
- If you want a particular function to be accessible only to the functions in the file in which it is defined and not to any function in other files, declare the function to be static. eg.

```
static int power(int x inty)
           {
             .   .   .
             .   .   .
           }
```

Register Variable

- These variables are stored in one of the machine's register and are 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 nonregister 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.