Storage Class In C

SDC OSW CSE 3541

Department of Computer Science & Engineering
ITER, Siksha 'O' Anusandhan Deemed To Be University
Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

Text Book(s)

- **Jeri R. Hanly & Elliot B. Koffman**

  Problem Solving and Program Design in C

  Seventh Edition, Pearson Education

- **Robert Love**

  LINUX System Programming

  Second Edition, SPD, O'REILLY

Talk Flow

Introduction

- Two different ways to characterize variables: by **data type** and by **storage class**.
- **Variable: Local variables** that are recognized only within a single function and **global variables** that are recognized in two or more functions.
- **Data type** refers to the type of information represented by a variable, e.g., integer number, floating-point number, character, etc.
- **Storage class** refers to the permanence of a variable, and its scope within the program, i.e., the portion of the program over which the variable is recognized.
- **Scope of a Name:** The scope of a name refers to the region of a program where a particular meaning of a name is **visible** or can be referenced.
- There are **four** storage classes in C:

① automatic (identified by the keyword **auto**)
② register (identified by the keyword **register**)
③ static (identified by the keyword **static**)
④ external (identified by the keyword **extern**)

Storage Classes in C

- If we don't specify the storage class of a variable in its declaration, the compiler will assume a storage class depending on the context in which the variable is used. Thus, variables have certain default storage classes.
- It is the variable's storage class that determines in which of two locations (i.e. memory and CPU register) the value is stored.
- A variable's storage class tells us:
  - Where the variable would be **stored**?
  - What will be the **initial value** of the variable, if initial value is not specifically assigned? (i.e. the default initial value.)
  - What is the **scope** of the variable? (i.e. in which functions the value of the variable would be available.)
  - What is the **life** of the variable?
    (i.e. how long would the variable exist.)

Automatic Storage Class

The features of a variable defined to have an automatic storage class are as under:

Storage	:	Memory
Default initial value	:	An unpredictable value, which is often called a 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

Discussion:

The **auto** storage class is the default storage class for all local variables including the formal argument declarations.

```
{
    int day;
    auto int month;
}
```

- Automatic variables defined in different functions will therefore be independent of one another, even though they may have the same name.
- Any variable declared within a function is interpreted as an automatic variable unless a different storage class specification is shown within the declaration.

Example: Automatic Storage Class

```
int main(void)
{
 auto int i, j ;
 printf("\n%d %d", i, j ) ;
}
output- Unpredicted value (garbage)
```

```
int main(void)
{
 auto int i = 1 ;
 {
   {
      {
           printf ( "\n%d ", i ) ;
      }
      printf ( "%d ", i ) ;
   }
   printf ( "%d", i ) ;
 }
}
Output- 1 1 1
```

```
int main(void)
{
   auto int i = 1 ;
   {
      auto int i = 2 ;
      {
           auto int i = 3 ;
           printf ( "\n%d ", i ) ;
      }
      printf ( "%d ", i ) ;
    }
    printf ( "%d", i ) ;
}
Output- 3 2 1
```

Register Storage Class

The features of a variable defined to have a register storage class are as under:

Storage	:	CPU Register
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.

Discussion:

```
int main(void){
  register int i ;
  for ( i = 1 ; i <= 10 ; i++ )
     printf ("\n%d", i ) ;
}
```

- A value stored in a CPU register can always be accessed faster than the one that is stored in memory.
- Therefore, if a variable is used at many places in a program it is better to declare its storage class as **`register`**. A good example of frequently used variables is loop counters.

Static Storage Class

The features of a variable defined to have a static storage class are as under:

Storage	:	Memory
Default initial value	:	Zero
Scope	:	Local to the block in which the variable is defined.
Life	:	Value of the variable persists between different function calls.

Discussion:

```
static int i ;
for (  ; i <= 10 ; i++ )
   printf ("\n%d", i ) ;
```

- Static variables are defined within a function in the same manner as automatic variables, except that the variable declaration must begin with the **`static`** keyword.
- Static variables can be utilized within the function in the same manner as other variables. They cannot, however, be accessed outside of their defining function.

Example: Static Storage Class

```
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   auto int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

Example: Static Storage Class

```
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   auto int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

Output- 1 1 1

Example: Static Storage Class

```
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   auto int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

```
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   static int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

Output- 1 1 1

Example: Static Storage Class

```c
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   auto int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

Output- 1 1 1

```c
void increment(void);
int main(void)
{
   increment( ) ;
   increment( ) ;
   increment( ) ;
}
void increment(void)
{
   static int i = 1 ;
   printf ( "%d\n", i ) ;
   i = i + 1 ;
}
```

Output- 1 2 3

Guess the Output

```
void func(void);
static int count = 5; /* global variable */
main() {
   while(count--) {
      func();
   }
   return 0;
}
/* function definition */
void func( void ) {
   static int i = 5; /* local static variable */
   i++;
   printf("i is %d and count is %d\n", i, count);
}
```

Guess the Output

```
void func(void);
static int count = 5; /* global variable */
main() {
   while(count--) {
      func();
   }
   return 0;
}
/* function definition */
void func( void ) {
   static int i = 5; /* local static variable */
   i++;
   printf("i is %d and count is %d\n", i, count);
}
```

Output:

i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0

External Storage Class

The features of a variable whose storage class has been defined as external are as follows:

Storage	:	Memory
Default initial value	:	Zero
Scope	:	Global
Life	:	As long as the program's execution doesn't come to an end.

Discussion:

```
int i ;
int main(void){
   printf ( "\ni = %d", i ) ;
   .....
   .....
   return 0;
}
```

- External variables are declared outside all functions, yet are available to all functions that care to use them.

Example: External Storage Class

```
void increment(void);
void decrement(void);
int i ;
int main(void){
  printf ("\ni = %d", i ) ; /* 0 */
  increment( );  /* 1 */
  increment( );  /* 2 */
  decrement( );  /* 1 */
  decrement( );  /* 0 */
  return 0;
}
void increment(void){
i = i + 1;
printf("\non incrementing i=%d",i);
}
void decrement(void){
i = i - 1;
printf("\non decrementing i=%d",i);
}
```

Example: External Storage Class

```
void increment(void);
void decrement(void);
int i ;
int main(void){
  printf ("\ni = %d", i ) ; /* 0 */
  increment( );  /* 1 */
  increment( );  /* 2 */
  decrement( );  /* 1 */
  decrement( );  /* 0 */
  return 0;
}
void increment(void){
i = i + 1;
printf("\non incrementing i=%d",i);
}
void decrement(void){
i = i - 1;
printf("\non decrementing i=%d",i);
}
```

```
int x = 10 ;
int main(void)
{
 int x = 20 ;
 printf("\n%d",x); /* local */
 display( ) ;
 return 0;
}
void display(void){
printf("\n%d",x) ; /* global */
}
```

Example: External Storage Class

```
void increment(void);
void decrement(void);
int i ;
int main(void){
  printf ("\ni = %d", i ) ; /* 0 */
  increment( );  /* 1 */
  increment( );  /* 2 */
  decrement( );  /* 1 */
  decrement( );  /* 0 */
  return 0;
}
void increment(void){
i = i + 1;
printf("\non incrementing i=%d",i);
}
void decrement(void){
i = i - 1;
printf("\non decrementing i=%d",i);
}
```

```
int x = 10 ;
int main(void)
{
 int x = 20 ;
 printf("\n%d",x); /* local */
 display( ) ;
 return 0;
}
void display(void){
printf("\n%d",x) ; /* global */
}
```

```
int x = 21 ;
int main(void)
{
  extern int y ;
  printf("\n%d %d", x, y);
  return 0;
}
int y = 31;
```

Accessing Global Variable

A global variable can be accessed using the keyword **extern**, if we have a local variable with same name.

```
int val = 10; /* global variable */

int main(void)
{
    int val = 20; /* local variable */
    {
        extern int val;
        printf("global variable val=%d\n", val);
    }
    printf("local variable val=%d\n", val);
    return 0;
}
```

Which storage class to Use When

Some sort of rules for usage of different storage classes in different programming situations with a view to:

(a) economize the memory space consumed by the variables

(b) improve the speed of execution of the program

The rules are as under:

- Use **static** storage class only if you want the value of a variable to persist between different function calls.
- Use **register** storage class for only those variables that are being used very often in a program.
- Use **extern** storage class for only those variables that are being used by almost all the functions in the program.
- If you don't have any of the express needs mentioned above, then use the **auto** storage class.

Review Questions

```
int i = 0 ;
void val(void){
 i = 100 ;
 printf("\n val's i = %d",i);
 i++ ;
}
int main(void){
 printf("\n main's i = %d",i);
 i++ ;
 val();
 printf("\n main's i = %d",i) ;
 val();
}
```

```
int main(void)
{
 static int count = 5 ;
 printf("\ncount = %d", count--);
 if ( count != 0 )
     main( ) ;
}
```

```
void g(int x){
 static int v=1;int b = 3 ;
 v += x ;
 return ( v + x + b ) ;
}
int main(void){
 int i, j ;
 for(i = 1;i < 5;i++){
   j = g(i) ;
   printf("\n%d",j);
 }
}
```

```
void func(void){
auto int i=0;register int j=0;
static int k=0;i++ ; j++ ; k++;
printf("\n%d%d%d",i, j,k);
}
int main(void){
 func();
 func();
 return 0;
}
```

THANK YOU