Revisiting The Datatypes

Storage Classes in C

- To fully define a variable one needs to mention not only its ‘type’ but also its ‘storage class’. In other words, not only do all variables have a data type, but they also have a ‘storage class’.

- From the C compiler’s point of view, a variable name identifies some physical location within the computer where the string of bits representing the variable’s value is stored.

Storage Classes in C

- There are basically two kinds of locations in a computer where such a value may be kept — Memory and CPU registers.

- It is the variable's storage class that determines in which of these two locations the value is stored.

What we know from Storage Class?

A variable's storage class tells us:

a) Where the variable would be stored.

b) What will be the initial value of the variable, if the initial value is not specifically assigned? (i.e. the default initial value).

c) What is the scope of the variable; i.e. in which functions the value of the variable would be available?

d) What is the life of the variable; i.e. how long would the variable exist?

Storage Classes in C

There are four storage classes in C:

- Automatic storage class
- Register storage class
- Static storage class
- External storage class

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.

Automatic Storage Class

```
main( )
{
    auto int i, j ;
    printf ( "\n%d %d", i, j ) ;
}
```

The output of the above program could be...

```
1211 221
```

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

The output of the above program would be:

```
3 2 1
```

Register Storage Class

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

- 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

- A good example of frequently used variables is loop counters. We can name their storage class as a register.

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

- Here, even though we have declared the storage class of i as register, we cannot say for sure that the value of i would be stored in a CPU register.
- Why? Because the number of CPU registers are limited, and they may be busy doing some other task.
- What happens in such an event... the variable works as if its storage class is **auto**.

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 − The value of the variable persists between different function calls.

Static Storage Class

```
main( )
{
    increment( ) ;
    increment( ) ;
    increment( ) ;
}

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

```
main( )
{
    increment( ) ;
    increment( ) ;
    increment( ) ;
}

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

The output of the above programs would be:

```
1
1
1
```

```
1
2
3
```

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.

External Storage Class

```
int i ;
main( )
{
printf ( "\ni = %d", i ) ;
increment( ) ;
increment( ) ;
decrement( ) ;
decrement( ) ;
}
increment( )
{
i = i + 1 ;
printf ( "\non incrementing i = %d", i ) ;
}
decrement( )
{
```

External Storage Class

Look at the following program.

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

Output:

```
21 31
```

- Here, x and y both are global variables as both are defined externally.
- Note the difference between the following:

```
extern int y ;
int y = 31 ;
```

- Here the first statement is a declaration, whereas the second is the definition.

External Storage Class

- When we declare a variable no space is reserved for it, whereas, when we define it space gets reserved for it in memory.
- We had to declare y since it is being used in printf( ) before it's definition is encountered. There was no need to declare x since its definition is done before its usage.
- Also remember that a variable can be declared several times but can be defined only once.

External Storage Class

Another small issue—what will be the output of the following program?

```
int x = 10 ;
main( )
{
    int x = 20 ;
    printf ( "\n%d", x ) ;
    display( ) ;
}
display( )
{
    printf ( "\n%d", x ) ;
}
```

- Here x is defined at two places, once outside **main( )** and once inside it.
- Whenever such a conflict arises, it's the local variable that gets preference over the global variable.
- Hence the printf( ) within main() outputs 20.
- When display( ) is called and control reaches the printf( ) there is no such conflict. Hence this time the value of the global x, i.e. 10 gets printed.

Which to Use When

We can make a few ground rules for the 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

Which to Use When

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. The reason is, there are very few CPU registers at our disposal and many of them might be busy doing something else. Make careful utilization of scarce resources. A typical application of register storage class is loop counters, which get used a number of times in a program.

Which to Use When

- Use extern storage class for only those variables that are being used by almost all the functions in the program. This would avoid unnecessary passing of these variables as arguments when making a function call. Declaring all the variables as extern would amount to a lot of wastage of memory space because these variables would remain active throughout the life of the program.

Which to Use When

- If you don’t have any of the express needs mentioned above, then use the auto storage class. In fact most of the time we end up using the auto variables because often it so happens that once we have used the variables in a function we don’t mind losing them.

THANK YOU