“Advanced C Programming”

eDESD-2021

Storage Classes

Defines the scope (visibility) and life-time of variables and/or functions

Auto

- Default storage class for local variables

```
int count;
auto int count;
```

Register

- used to define local variables that should be stored in a register instead of RAM.
- Max size = size of register (=1 word)
- can't have unary operator applied to it
- Should be used for variables that need quick access e.g counters

  Ex: register int count;

static

- Exists during the lifetime of the program
- Enables variables to maintain their values between the function calls
- Can also be applied to global variable but the scope is restricted to file in which is declared

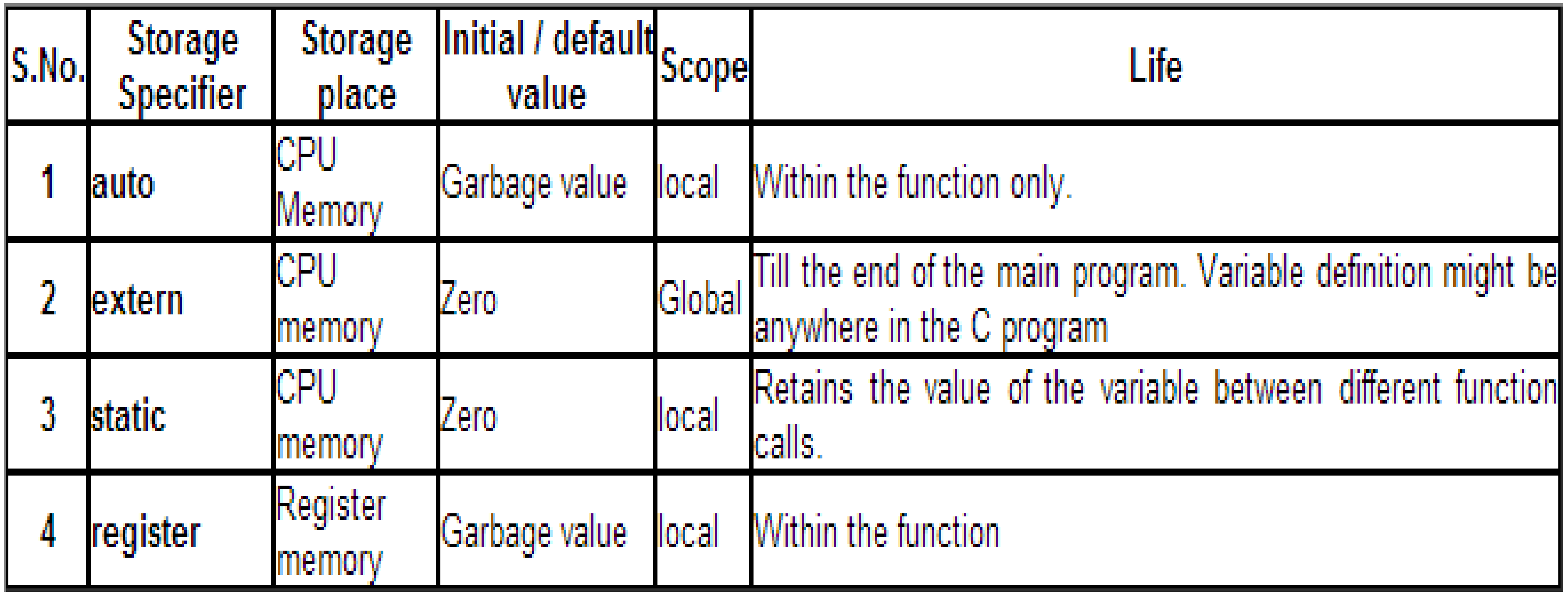
```
static int count=2;
```

extern

- Provides reference of global variable that is visible to ALL the program files
- When you use 'extern', the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined
- extern modifier is most commonly used when there are two or more files sharing the same global variables or functions

Storage Classes

Defines the scope (visibility) and life-time of variables and/or functions



S.No.	Storage Specifier	Storage place	Initial / default value	Scope	Life
1	auto	CPU Memory	Garbage value	local	Within the function only.
2	extern	CPU memory	Zero	Global	Till the end of the main program. Variable definition might be anywhere in the C program
3	static	CPU memory	Zero	local	Retains the value of the variable between different function calls.
4	register	Register memory	Garbage value	local	Within the function

Example auto

```
#include <stdio.h>
void increment(int);
int main()
 {
        auto int count=10 ;
        increment(count);
      printf("count is %d\n", count);
        increment(count);
      printf("count is %d\n", count);
        increment(count);
      printf("count is %d\n", count);
}
```

```
void increment(int count)
{
      count++;
      printf("count is %d\n", count);
}
```

- The scope of this auto variable is within the function only. It is equivalent to local variable. All local variables are auto variables by default.
- Life time of auto variable is within the block.
- Default value: garbage value

Example static

```
#include <stdio.h>
void increment();
int main()
 {
         increment();
         increment();
         increment();
}
```

```
void increment( )
{
     static int count=0;
     count++;
     printf("count is %d\n", count);
}
```

- The scope of this auto variable is within the function only.
- Life time of static variable until completion of program.
- Default value: 0

Example extern

```
#include<stdio.h>
int x = 10 ;
int main( )
{
    extern int y ;  //Declaration
    printf ( "The value of x is %d\n", x ) ;
    printf ( "The value of y is %d\n",y ) ;
    return 0;
}

int y = 50 ;   //Definition
```

- The scope of this variable is throughout the program
- Life time of extern variable until completion of program.
- Default value: 0

Example register

```
#include <stdio.h>
void increment(int);
int main()
 {
        register count = 1;
        increment(count);
        increment(count);
        increment(count);
}
```

```
void increment(int count)
{
      count++;
      printf("count is %d\n", count);
}
```

- The scope of this variable is within the block/function
- Life time of this variable is within the block/function.
- Default value: garbage value

- It's same as auto variable only difference is it's value stores in registers instead of RAM

Constants

Constants

Constants refer to fixed values that the program may not alter during its execution.

Constants:
integer constant,
floating constant,
character constant,
string constant.

```
#include <stdio.h>

int main() {
int area;
const int r = 10   ;
const float pi = 3.14;
const char nl='\n';

   area = 2*pi*r;
//pi=pi+0.1;
   printf("area : %d", area);
   printf("%c", nl);

   return 0;
}
```

```
#include <stdio.h>
#define R 10
#define PI  3.14
#define NEWLINE '\n'

int main() {
   int area;

   area = 2*PI*R;
   printf("area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

THANK YOU