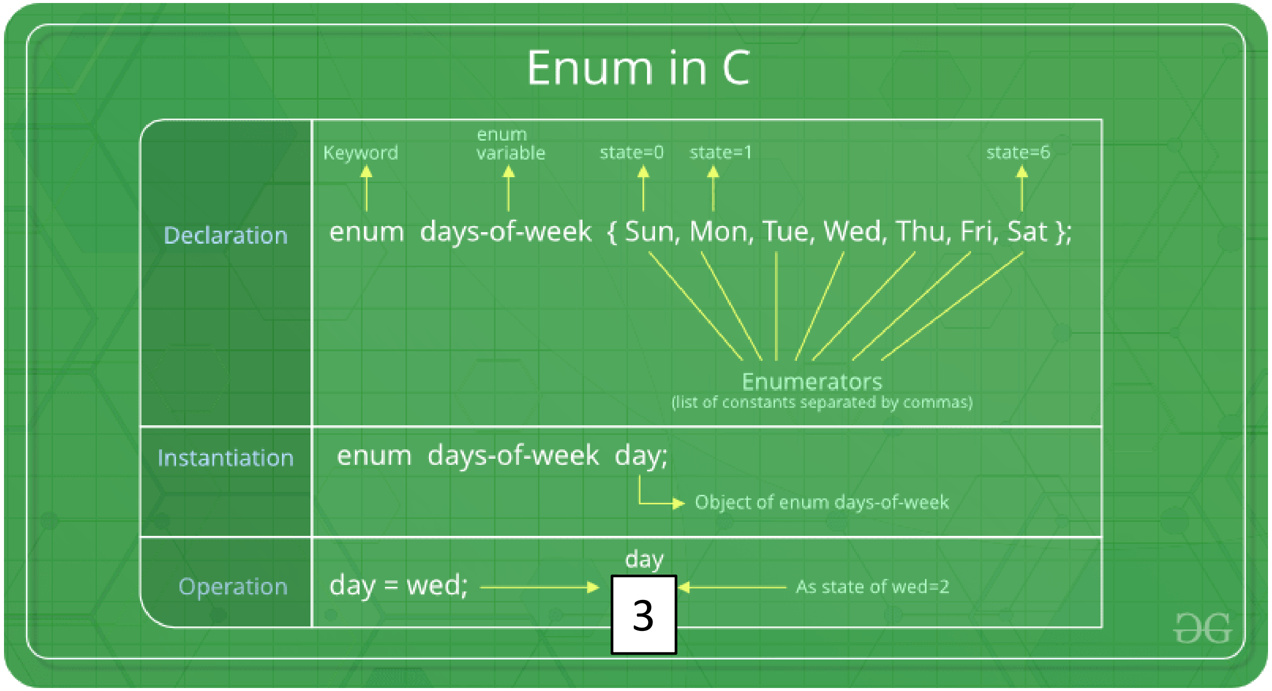
Unit 12: Additional Features of C

Er. Nipun Thapa

Enumeration (or enum) in C

- Enumeration (or enum) is a user defined data type in C.
- It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.



Enumeration (or enum) in C

- Variables of type enum can also be defined. They can be defined in two ways:

```
// In both of the below cases, "day" is
// defined as the variable of type week.

enum week{Mon, Tue, Wed};
enum week day;
// Or

enum week{Mon, Tue, Wed}day;
```

Enumeration (or enum) in C

```
// An example program to demonstrate working of enum in C

#include<stdio.h>
 enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
 int main()
{
   enum week day;
   day = Wed;
   printf("%d",day);
   return 0;
}
```

```
Output
2
```

Enumeration (or enum) in C

```
// Another example program to demonstrate working of enum in C

#include<stdio.h>
 enum year{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

int main()
{
  int i;
  for (i=Jan; i<=Dec; i++)
    printf("%d ", i);
  return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9 10 11
```

Enumeration (or enum) in C

```
// An example program to demonstrate working of enum in C

#include<stdio.h>
 enum week{Mon = 1, Tue, Wed, Thur, Fri, Sat, Sun};
 int main()
{
   enum week day;
   day = Mon;
   printf("%d",day);
   return 0;
}
```

```
Output
1
```

Macros in C

- A **macro** is a piece of code in a program that is replaced by the value of the macro.
- Macro is defined by **#define** directive.
- Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro.
- Macro definitions need not be terminated by a semi-colon(**;**).

Macros in C

```
// C program to illustrate macros

#include <stdio.h>
 // Macro definition
#define LIMIT 5

int main()
{
   // Print the value of macro defined
   printf("The value of LIMIT is %d“, LIMIT);
    return 0;
}
```

```
Output:
The value of LIMIT is 5
```

Macros in C

```
// C program to illustrate macros
#include <stdio.h>
#define AREA(l, b) (l * b)
int main()
{

   int l1 = 10, l2 = 5, area;
   area = AREA(l1, l2);
   printf("Area of rectangle is: %d",area);

   return 0;
}
```

Output

```
Area of rectangle is:50
```

Command line parameters

- The most important function of C is main() function.
- It is mostly defined with a return type of int and without parameters :

```
int main() { /* ... */ }
```

- We can also give command-line arguments in C .
- Command-line arguments are given after the name of the program in command-line shell of Operating Systems.
- To pass command line arguments, we typically define main() with two arguments :
  - first argument is the number of command line arguments and
  - second is list of command-line arguments.

```
int main(int argc, char *argv[]) { /* ... */ }
```

or

```
int main(int argc, char **argv) { /* ... */ }
```

Command line parameters

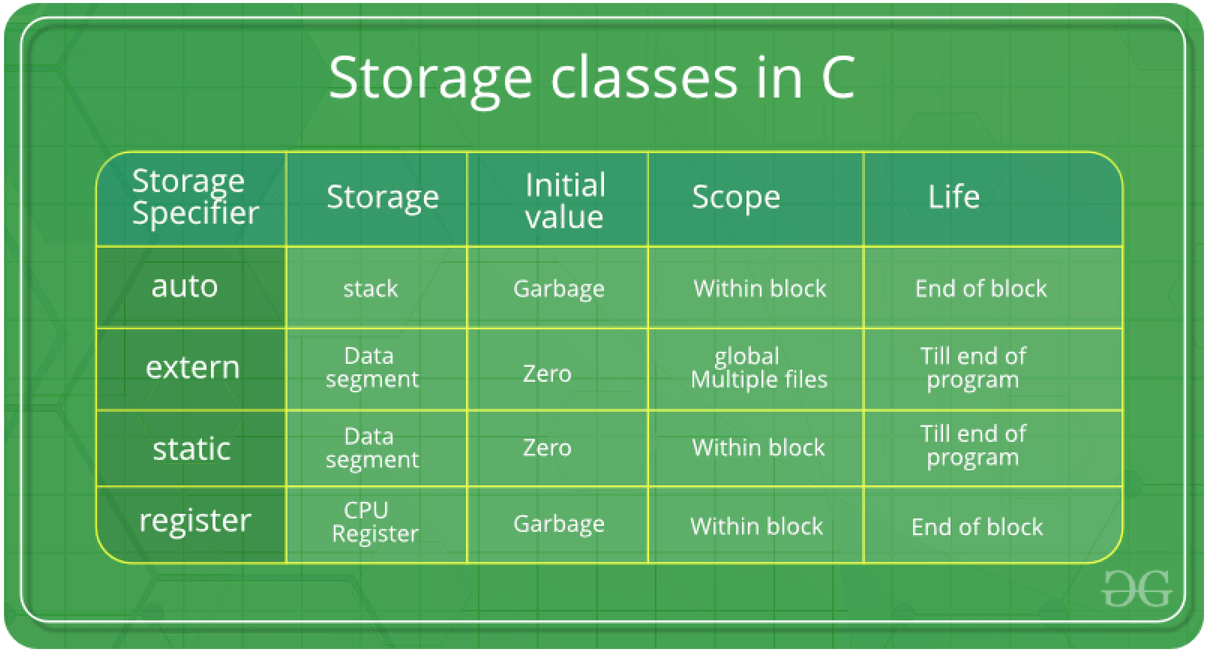
- **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non negative.
- **argv(ARGument Vector)** is array of character pointers listing all the arguments.
- If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

Properties of Command Line Arguments:

1. They are passed to main() function.
2. They are parameters/arguments supplied to the program when it is invoked.
3. They are used to control program from outside instead of hard coding those values inside the code.
4. argv[argc] is a NULL pointer.
5. argv[0] holds the name of the program.
6. argv[1] points to the first command line argument and argv[n] points last argument.

Storage classes in C

- Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.
- **C language uses 4 storage classes**, namely:



Storage classes in C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

Storage classes in C

auto:

- This is the default storage class for all the variables declared inside a function or a block.
- Hence, the keyword auto is rarely used while writing programs in C language.
- Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope).
- Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared.
- However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables reside.
- They are assigned a garbage value by default whenever they are declared.

Storage classes in C

extern:

- Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.
- Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well.
- So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere.
- It can be accessed within any function/block.
- Also, a normal global variable can be made extern as well by placing the ‘extern’ keyword before its declaration/definition in any function/block.
- This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only.
- The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.

Storage classes in C

static:

- This storage class is used to declare static variables which are popularly used while writing programs in C language.
- Static variables have the property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope.
- So we can say that they are initialized only once and exist till the termination of the program.
- Thus, no new memory is allocated because they are not re-declared.
- Their scope is local to the function to which they were defined.
- Global static variables can be accessed anywhere in the program.
- By default, they are assigned the value 0 by the compiler.

Storage classes in C

register:

- This storage class declares register variables that have the same functionality as that of the auto variables.
- The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free registration is available.
- This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.
- If a free registration is not available, these are then stored in the memory only.
- Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program.
- An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

Storage classes in C

To specify the storage class for a variable, the following syntax is to be followed:

Syntax:

storage_class var_data_type var_name;

Finished
Unit 12