C Programming Language

1. sizeof Operator:

Code implementation of own sizeof:

```
#define my_sizeof(type)   (char *)(&type+1)-(char*)(&type)
```

2. Infinite loop:

i. while(1)

ii. for(;;)

3. Jump Statements:

i. break:

- The break statement is used to terminate the loop or statement in which it present.
- After that, the control will pass to the statements that present after the break statement, if available.
- If the break statement presents in the nested loop, then it terminates only those loops which contains break statement.

ii. continue:

- This statement is used to skip over the execution part of the loop on a certain condition.
- After that, it transfers the control to the beginning of the loop.
- Basically, it skips its following statements and continues with the next iteration of the loop.

iii. goto:

- This statement is used to transfer control to the labeled statement in the program.
- The label is the valid identifier and placed just before the statement from where the control is transferred.
- And it works within the functions.

iv. return:

- This statement terminates the execution of the method and returns the control to the calling method.
- It returns an optional value.
- If the type of method is void, then the return statement can be excluded.

4. <u>**Switch Statements:**</u>

- The expression used in a switch statement must have an integral or enumerated type.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.

5. <u>**Qualifiers:**</u>

- The keywords which are used to change the properties of a variable is called qualifiers.
- They are two types of qualifiers:
  - volatile
  - const

i. volatile:

- When a variable is defined as volatile, the program may not change the value of the variable explicitly.
- But these variable values might keep on changing without any explicit assignment by the program.
- When "volatile" comes in picture then it instructs the compiler that variable is special so that no optimization are allowed on this variable.
- An object that has volatile-qualified type may be modified in ways unknown to the implementation or have other unknown side effects.
- A volatile declaration may be used to describe an object corresponding to a *memory-mapped input/output port* or an object accessed by an *asynchronously interrupting function*.
- Actions on objects are declared so it should not be ''optimized out'' by an implementation or reordered except as permitted by the rules for evaluating expressions.
- Basically, C standard says that "volatile" variables can change from outside the program and that's why compilers are not supposed to optimize their access. Now, if we have too many volatile variables in our program would also result in lesser compiler optimization.

- <u>Syntax:</u>
  - volatile data_type variable_name;
  - volatile data_type *variable_name;

- Theoretical Example:
    - If global variable address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.

- Code Example:
    - volatile uint32 status = 0;
    - volatile uint32 *statusPtr = 0xF1230000

ii. const:

- When a variable is declared as const we can't change the value of that variable in its lifetime, once it is assigned.
- Syntax:
    - const Data_Type Identifier = Value;
- Example:
    - const float PI = 3.1415;

- const qualifier can also be used with pointers, which are follows:
    - ❖ **Pointer to a Constant:**
        - ✓ Example:
            - **const int *ptr;**
              or
            - **int const *ptr;**
        - ✓ The above declaration described that the identifier is pointing to a constant integer variable. So, we cannot change the value of the pointed integer variable using the pointer but change the pointer to point to any other integer variable.

    - ❖ **Constant Pointer to variable:**
        - ✓ Example:
            - **int *const ptr;**
        - ✓ The above declaration described that constant pointer is pointing to an integer variable. It means pointer is itself not modifiable i.e. identifier cannot point to the other object, but we can change the value of the integer variable pointed by the pointer.

- **Constant Pointer to a constant:**
  - <u>Example:</u>
    - **const int *const ptr;**
  - The above declaration described that constant pointer is pointing to a constant integer variable. It means we cannot change value pointed by the pointer as well as we cannot point the pointer to other integer variables.

- **static const:**
  - const means you cannot change the value.
  - static (inside a function) means the value survives to future executions of the function. const values do this too!
  - static (outside a function) means that the scope of the declaration is only the current source file, not the entire program.
  - So, a static const int x = 3; declared inside a function is the same thing as a const int x = 3;
  - Outside a function, a const int x = 3; would be visible to the entire program while a static const int x = 3; would only be visible in the current source file.

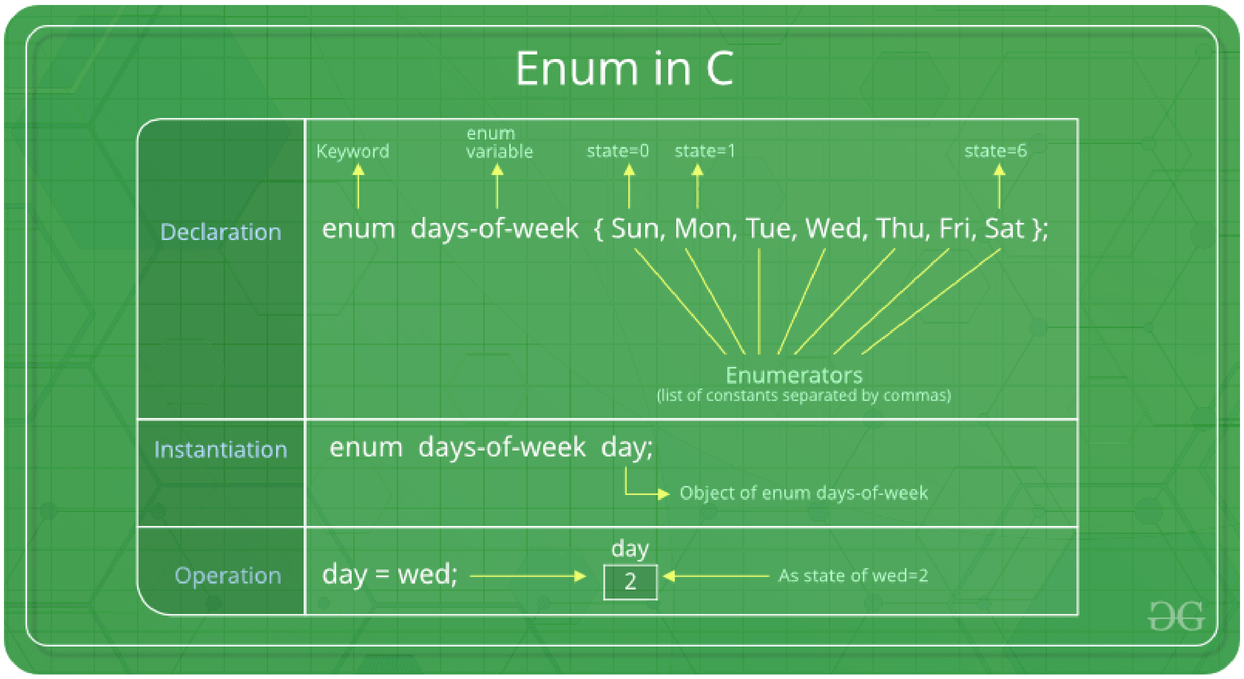
- **const volatile:**
  - const volatile means that the program cannot modify the variable's value, but the value can be modified from the outside, thus no optimizations will be performed on the variable.

6. <u>Type Qualifiers:</u>

i.	Size Qualifiers	-	short, long
ii.	Sign Qualifiers	-	signed, unsigned

7. <u>Enumeration or enum:</u>

- 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.
- By default it starts with ‘0’.
- <u>Example:</u> enum State {Working = 1, Failed = 0};



8. <u>typedef:</u>

- typedef is used to give a type a new name.
- <u>Example:</u>

typedef unsigned char	UInt8;
typedef unsigned short int	UInt16;
typedef unsigned int	UInt32;
typedef unsigned long int	UInt64;
typedef char	Int8;
typedef short int	Int16;
typedef int	Int32;
typedef long int	Int64;

- <u>Code:</u>
  - *<u>Input:</u>*

```
int main()
{
    printf("UInt8   : %dByte\n", sizeof(UInt8));
    printf("UInt16  : %dByte\n", sizeof(UInt16));
    printf("UInt32  : %dByte\n", sizeof(UInt32));
    printf("UInt64  : %dByte\n", sizeof(UInt64));
    printf("Int8    : %dByte\n", sizeof(Int8));
    printf("Int16   : %dByte\n", sizeof(Int16));
```

```
        printf("Int32   : %dByte\n", sizeof(Int32));
        printf("Int64   : %dByte\n", sizeof(Int64));
    }
```

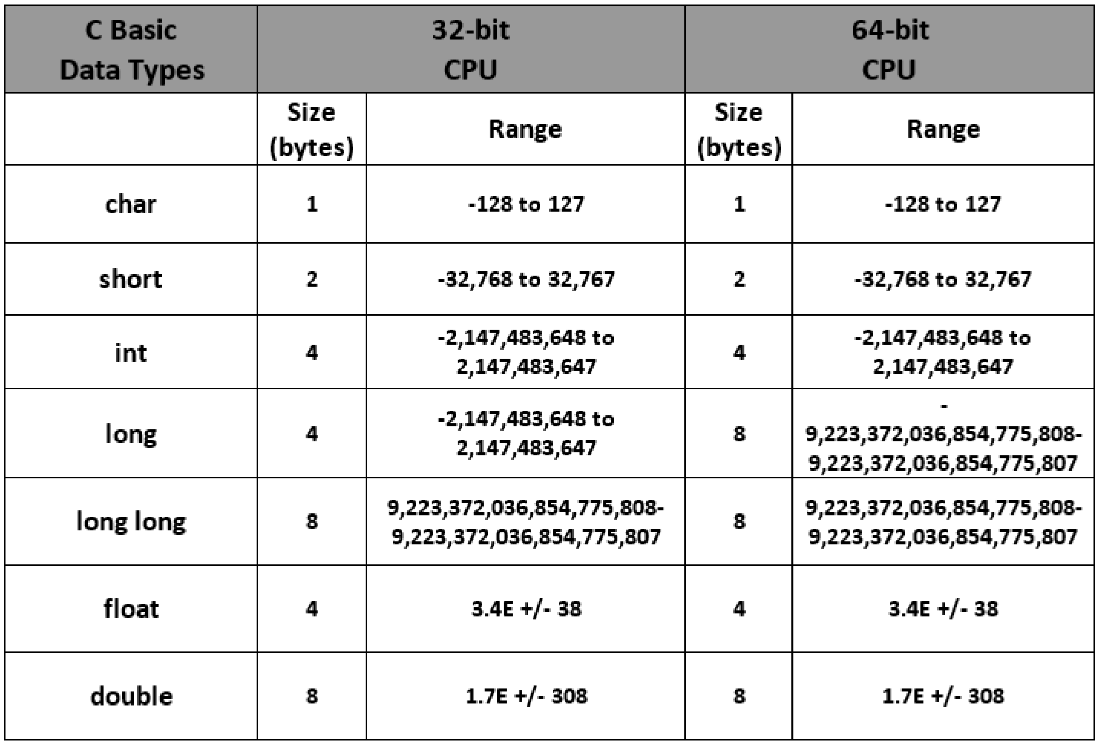
- *Output:*

```
UInt8    : 1Byte
UInt16   : 2Byte
UInt32   : 4Byte
UInt64   : 8Byte
Int8     : 1Byte
Int16    : 2Byte
Int32    : 4Byte
Int64    : 8Byte
```

9. typedef vs #define:

- typedef is limited to giving symbolic names to types only, whereas #define can be used to define an alias for values as well.
- typedef interpretation is performed by the compiler where #define statements are performed by preprocessor.
- #define should not be terminated with a semicolon, but typedef should be terminated with semicolon.
- #define will just copy-paste the definition values at the point of use, while typedef is the actual definition of a new type.
- typedef follows the scope rule which means if a new type is defined in a scope (inside a function), then the new type name will only be visible till the scope is there. In case of #define, when preprocessor encounters #define, it replaces all the occurrences, after that (No scope rule is followed).

10. Size char of data type:



C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308