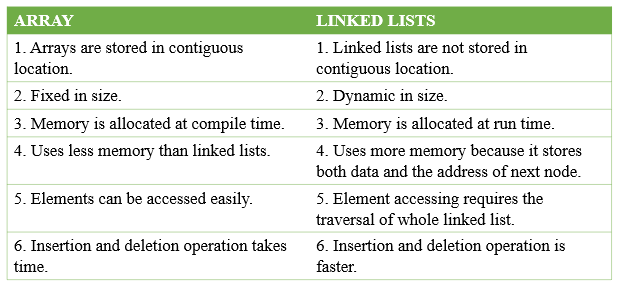
**1.Differentiate between the macros and the functions.**

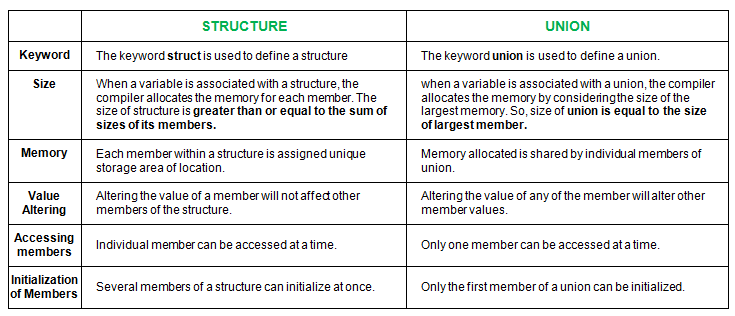
The differences between macros and functions can be explained as follows:

| **Macros** | **Functions** |
| --- | --- |
| It is preprocessed rather than compiled. | It is compiled not preprocessed. |
| It is preprocessed rather than compiled. | Function checks for compilation errors. |
| Code length is increased. | Code length remains the same. |
| Macros are faster in execution. | Functions are a bit slower in execution. |
| Macros are useful when a small piece of code is used multiple times in a program. | Functions are helpful when a large piece of code is repeated a number of times. |

2.Differenece between array and linkedlist.



3.Difference between structure and union.



4.Difference between malloc and calloc.

|  |  |
| --- | --- |
| **malloc()** | **calloc()** |
| **1.** | It is a function that creates one block of memory of a fixed size. | It is a function that assigns more than one block of memory to a single variable. |
| **2.** | It only takes one argumemt | It takes two arguments. |
| **3.** | It is faster then calloc. | It is slower than malloc() |
| **4.** | It has high time efficiency | It has low time efficiency |
| **5.** | It is used to indicate memory allocation | It is used to indicate contiguous memory allcoation |
|  |  |  |

5. Difference between static and dynamic memory allocation.

|  |  |  |
| --- | --- | --- |
| **S.No** | **Static Memory Allocation** | **Dynamic Memory Allocation** |
| 1 | In the static memory allocation, variables get allocated permanently, till the program executes or function call finishes. | In the Dynamic memory allocation, variables get allocated only if your program unit gets active. |
| 2 | Static Memory Allocation is done before program execution. | Dynamic Memory Allocation is done during program execution. |
| 3 | It uses [stack](https://www.geeksforgeeks.org/stack-data-structure/) for managing the static allocation of memory | It uses [heap](https://www.geeksforgeeks.org/heap-data-structure/) for managing the dynamic allocation of memory |
| 4 | It is less efficient | It is more efficient |
| 5 | In Static Memory Allocation, there is no memory re-usability | In Dynamic Memory Allocation, there is memory re-usability and memory can be freed when not required |
| 6 | In static memory allocation, once the memory is allocated, the memory size can not change. | In dynamic memory allocation, when memory is allocated the memory size can be changed. |
| 7 | In this memory allocation scheme, we cannot reuse the unused memory. | This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it. |
| 8 | In this memory allocation scheme, execution is faster than dynamic memory allocation. | In this memory allocation scheme, execution is slower than static memory allocation. |
| 9 | In this memory is allocated at compile time. | In this memory is allocated at run time. |
| 10 | In this allocated memory remains from start to end of the program. | In this allocated memory can be released at any time during the program. |
| 11 | **Example:** This static memory allocation is generally used for [array](https://www.geeksforgeeks.org/introduction-to-arrays/). | **Example:** This dynamic memory allocation is generally used for [linked list](https://www.geeksforgeeks.org/data-structures/linked-list/). |

6.

**sizeof vs strlen()**

* 1. **Type:** Sizeof operator is a unary operator whereas strlen() is a predefined function in C
  2. **Data types supported:**Sizeof gives actual size of any type of data (allocated) in bytes (including the null values) whereas get the length of an array of chars/string.
  3. **Evaluation size:**sizeof() is a compile-time expression giving you the size of a type or a variable’s type. It doesn’t care about the value of the variable.  
     Strlen on the other hand, gives you the length of a C-style NULL-terminated string.
  4. **Summary:**The two are almost different concepts and used for different purposes.
  5. **In context of C++:** In C++, you do not need any of them as such.  
     **strlen()** in C-style strings can be replaced by **C++ std::strings**.  
     **sizeof() in C** is as an argument to functions like malloc(), memcpy() or memset() can be replaced by**C++ (use new, std::copy(), and std::fill() or constructors**
  6. Sizeof is count with include null value, strlen is count only the given string count.
  7. Example: str= ”Jyothi”
  8. 🡪sizeof(str)= 7, strlen(str) = 6

7. Difference between Stack and Queue Data Structures are as follows:

| **Stacks** | **Queues** |
| --- | --- |
| 1.Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | 1.Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| 2.Insertion and deletion in stacks takes place only from one end of the list called the top. | 2.Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| 3.Insert operation is called push operation. | 3.Insert operation is called enqueue operation. |

|  |  |
| --- | --- |
| 4.Delete operation is called pop operation. | 4.Delete operation is called dequeue operation. |
| 5.In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | 5.In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| 6.Stack is used in solving problems works on recursion. | 6.Queue is used in solving problems having sequential processing. |
| 7.Stack does not have any types. | 7.Queue is of three types – 1. Circular Queue 2. Priority queue 3. Double-ended queue. |
| 8.Can be considered as a vertical collection visual. | 8.Can be considered as a horizontal collection visual. |

9. Full condition when top == max-1, 9. When rear==max-1, queue is full.

10. Empty condition: when top == -1 10. When front=rear+1 or front==-1, queue empty.

**8. Difference between Actual and Formal Parameters :**

| **Actual Parameters** | **Formal Parameters** |
| --- | --- |
| When a function is called, the values (expressions) that are passed in the function call are called the arguments or actual parameters. | The parameter used in function definition statement which contain data type on its time of declaration is called formal parameter. |
| These are the variables or expressions referenced in the parameter list of a subprogram call. | These are the variables or expressions referenced in the parameter list of a subprogram specification. |
| Actual Parameters are the parameters which are in calling subprogram. | Formal Parameters are the parameters which are in called subprogram. |
| There is no need to specify datatype in actual parameter. | The datatype of the receiving value must be defined. |
| The parameters are written in function call are known as actual parameters. | The parameters are written in function definition are known as formal parameters. |
| Actual Parameters can be constant values or variable names. | Formal Parameters can be treated as local variables of a function in which they are used in the function header. |

9. Difference between compiler and interpreter.

| **S.No.** | **Compiler** | **Interpreter** |
| --- | --- | --- |
| 1. | The compiler scans the whole program in one go. | Translates the program one statement at a time. |
| 2. | As it scans the code in one go, the errors (if any) are shown at the end together. | Considering it scans code one line at a time, errors are shown line by line. |
| 3. | The main advantage of compilers is its execution time. | Due to interpreters being slow in executing the object code, it is preferred less. |
| 4. | It converts the source code into object code. | It does not convert source code into object code instead it scans it line by line |
| 5 | It does not require source code for later execution. | It requires source code for later execution. |
| 6 | Execution of the program takes place only after the whole program is compiled. | Execution of the program happens after every line is checked or evaluated. |
| 7 | The machine code is stored in the disk storage. | Machine code is nowhere stored. |
| 8 | Compilers more often take a large amount of time for analyzing the source code. | In comparison, Interpreters take less time for analyzing the source code. |
| 9. | It is more efficient. | It is less efficient. |
| 10. | CPU utilization is more. | CPU utilization is less. |
| Eg. | C, C++, C#, etc are programming languages that are compiler-based. | Python, Ruby, Perl, SNOBOL, MATLAB, etc are programming languages that are interpreter-based. |

10. Difference between call by value and call by reference.

| **Call By Value** | **Call By Reference** |
| --- | --- |
| While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”. | While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References. |
| In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function. | In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function. |
| With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function. | With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them. |

#### 11. **How are global variables different from static variables?**

Global variables are variables with global scope, i.e., they are accessible throughout the program, unless shadowed. These variables are defined outside a function or code block.

Static variables are variables allocated statically, i.e., their value can’t be changed. It is fixed for the entire run of a program. They can be defined outside as well as inside functions. Moreover, they can be accessed from anywhere inside the program.

**12.List out the differences between reference and Pointer?**

|  |  |
| --- | --- |
| **Reference** | **Pointer** |
| It is an alternative name for a variable. | Stores the address of a variable. |
| Declared using \* : int \*ptr. | Declared using & : int &refvar. |
| Cannot have null values. | Can have null values assigned. |
| Can be accessed through pass by value. | Uses pass by reference. |
| Must be initialized upon declaration, i.e. int &ref; will give an error. | No need for initialization during declaration itself, i.e. int \*ptr is correct. |
| Shares same memory address as the original variable and takes up some space on the stack. | Has its size and memory address on the stack. |

13.

#### **What is the difference between getch() and getche() functions?**

Both getch() and getche() functions are used for reading a single character from the keyboard. The difference between the two, however, lies in terms of displaying the output. The getche() function displays the data, the entered character, on the output screen while the getch() function doesn’t. Use Alt+F5 to see the entered character.

14.Difference between ‘=’ and ‘==’?

= > Single equal is an assignment operator used to assign the values to the variables.

== > But, double equal is relational operator used to compare two variable values whether they are equal are not.

15. Difference between while and do-while?

While loop is executed only when given condition is true.

Whereas, do-while loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.

#### 16**. WHAT IS THE DIFFERENCE BETWEEN AUTO VARIABLE AND REGISTER VARIABLE IN C?**

Storage class of all variables are auto by default unless we specify a variable is register or static or extern in C program.

* Both auto variable and register variable are local variables. Register variables are stored in register memory. Whereas, auto variables are stored in main CPU memory.
* Register variables will be accessed very faster than the normal/auto variables since they are stored in register memory rather than main memory.
* But, only limited variables can be used as register since register size is very low. (16 bits, 32 bits or 64 bits)

#### 17. **WHAT IS THE DIFFERENCE BETWEEN ARRAY AND STRING IN C?**

Array can hold any of the data types. But, string can hold only char data type.

Array size must be a constant value. So, array size can’t be changed once declared. But, string size can be modified using char pointer.

Array is not ended with null character by default. But, string is ended with null (‘\0’) character by default.



18.Difference between signed and unsigned integer?

Ans::

**SIGNED INTEGER**

1.The **int** type in C is a signed integer, which means it can represent both negative and positive numbers.

2. A signed integer is a 32-bit datum that encodes an integer in the range [-2147483648 to 2147483647].

3. The signed integer is represented in twos complement notation. The most significant byte is 0 and the least significant is 3.

4. For signed char we need not to write the **signed** keyword.

**UNSIGNED INTEGER**

1.In the case of an unsigned integer, only positive numbers can be stored.

2. An unsigned integer is a 32-bit datum that encodes a nonnegative integer in the range [0 to 4294967295].

3. The unsigned integer is represented by an unsigned binary number whose most significant byte is 0; the least significant is 3.

4.But for unsigned, we have to mention the keyword.

5. Both of the Signed and Unsigned char, they are of 8-bits. So for signed char it can store value from -128 to +127, and the unsigned char will store 0 to 255.

### 19. Difference Between 32 And 64-bit Processors

We need to understand Bit first. In the computing world, Bit is the most basic unit of information and Bit is the short form of Binary Digit, which means it can be represented by two values – either 0 or 1. It is called Binary, as there are only two possible digits: 0 and 1. The Binary system is also called Base 2.

These Bits can be grouped together, and in multiples called Bytes, Kilobytes, Megabytes, Gigabytes, etc to store data and execute transactions.

**Some basic standards used in the market (relation between Bits and Bytes) are:**

1 Nibble = 4 Bits

1 Byte = 8 Bits

1 Kilobyte (KB) = 1000 Bytes

1 Megabyte (MB) = 1000 Kilobytes

1 Gigabyte (GB) = 1000 Megabytes

1 Terabyte (TB) = 1000 Gigabytes, and it goes on.

32-bit means that a microprocessor can execute 4 bytes of data in one instruction cycle while 64-bit means that a microprocessor can execute 8 bytes of data in one instruction cycle.

20. const int \*ptr and int \*const ptr difference?

Ans: const int \*ptr:: const int \* const ptr; Above declaration is a **constant pointer to a constant variable** which means we cannot change value pointed by the pointer as well as we cannot point the pointer to other variables.

Int const\* ptr::

**int const\*** is pointer to constant integer This means that the variable being declared is a pointer, pointing to a constant integer. Effectively, this implies that the pointer is pointing to a value that shouldn’t be changed. Const qualifier doesn’t affect the pointer in this scenario so the pointer is allowed to point to some other address.

21.difference between sysfs and procfs?

Ans::

## **what is procfs?-->** the process pseudo-file system (PROCFS)

**Procfs** or "**/proc**" is a special filesystem under Linux that is used to present process information and kernel processes. Although "/proc" is still used widely, much of the information found on systems running with a kernel level of 2.6 and above have been moved to another pseudo filesystem called "sysfs" which is generally mounted under "/sys". "/proc" is stored in memory, unlike other filesystems, which are stored on disk. If you list the "/proc" filesystem, you will notice that the majority of files are of a "0" byte length. However, if you view the contents of the file, you will see that there is quite a substantial amount of information within.

## **what is sysfs?-->**  system pseudo-file system (SYSFS)

**Many newer distributions of Linux are using "sysfs" mounted on /sys as a way of exporting information from the kernel to various applications.**

sysfs is a pseudo file system provided by the Linux kernel that **exports information about various kernel subsystems, hardware devices, and associated device drivers from the kernel's device model to user space through virtual files**.

### 22. Difference between const char\* p and char const\* p?

* const char\* p is a pointer to a const char.
* char const\* p is a pointer to a char const.

Since const char and char const are the same, it's the same.

### 23. What is an r-value and l-value?

* The term "r-value" refers to a data value stored in memory at a given address. An r-value is an expression that cannot have a value assigned to it, hence it can only exist on the right side of an assignment operator(=).
* The term "l-value" refers to a memory location that is used to identify an object. The l-value can be found on either the left or right side of an assignment operator(=). l-value is frequently used as an identifier.

### 24. What is the difference between #include "..." and #include <...>?

In practice, the difference is in the location where the preprocessor searches for the included file.

For #include <filename> the C pre-processor looks for the filename in the predefined list of system directories first and then to the directories told by the user(we can use -I option to add directories to the mentioned predefined list).

For #include "filename" the pre-processor searches first in the same directory as the file containing the directive, and then follows the search path used for the #include <filename> form. This method is normally used to include programmer-defined header files.

### 25. What is a near pointer and a far pointer in C?

* **Near Pointer**: In general, the near pointer can be considered because it is used to hold the address, which has a maximum size of just 16 bits. We can't store an address with a size larger than 16 bits using the near pointer. All other smaller addresses that are within the 16-bit limit, on the other hand, can be stored. Because we can only access 64kb of data at a time, you might assume the 16 bits are insufficient. As a result, it is regarded as one of the near-pointer's biggest drawbacks, which is why it is no longer commonly used.
* **Far Pointer:** A far pointer is considered a pointer of size 32 bits. It can, however, use the current segment to access information stored outside the computer's memory. Although, in order to use this type of pointer, we usually need to allocate the sector register to store the data address in the current segment.