**DSA(C++)**

* **Datatypes:-**

DATATYPES

Primitive Derived User-defined

*Integer,float,character,Boolean function,array,pointer,reference class,structure,union,enum*

INT:- The first bit show the sign of number , first bit also called MSB(most significant bit), if MSB=1 then number is –ve else number is +ve.

* **C++:-**

#include<iostream>

int main(){

std::cout<<”hello”;

return 0;

}

In above c++ program there is some basic concept:

* #include:- Preprocessor Directives used to include files
* Iostream:- Header file for taking inpt and printing output
* Main:- the execution of code begins from main function
* Cout: used to display ouput in quotation marks
* Return 0: exit status of a function
* **Operator:**

Operator are he symbols that tell the compiler to perform some specific operations.

1. **Arithmetic Operators:**

* Binary Operators (+,-,\*,/,%)
* Unary Operators(++{Incrementer},--{Decrementer})

**Pre Incrementer(++a)**

Int a=10;

Int b;

B=++a;

Cout<<a<<””<<b;

Ouput:

A=11

B=11

**Post Incrementer(a++)**

Int a=10;

Int b;

B=a++;

Cout<<a<<’’’<<b;

Output:

A=11

B=10

1. **Relational operators:**

Return the Boolean values(==,!=,<,>,<=,>=)

1. **Logical operators:**

Used to connect multiple conditions together or to reverse logical value.

* **&&:** AND gives us true if both operands are true , else false.
* **||:** OR gives us true if at least one of the operands are true.
* **! :**  NOT give the opposite logical value of the operand.

1. **Bitwise operator:**

Operate on bits and perform bit by bit operations.

AND(&), OR(|), XOR(^), Ones complement(~),Left shift operator(<<), Right shift operator(>>)

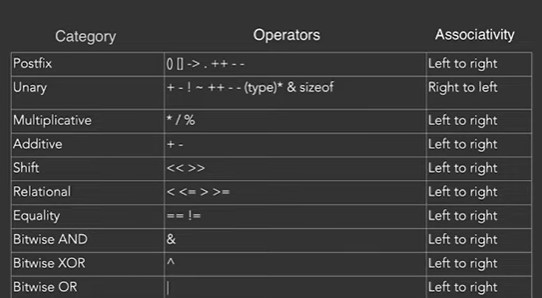
1. **Assignment operator**
2. **Miscellaneous operators:**

* **Sizeof():** Returns the size of variables.
* **Condition?X:Y -** Returns value of X if condition is true or else value of

Y.

* **Cast :**  Convert one data type to another.
* **Comma(,):** Causes a sequence of operations to be performed.
* **& :** Returns the address of a variable.
* **\* :**  Pointer to a variable.

**Operator Precedence:**



****



* **Functions:**

Functions is a piece of code that perform a specific task.

**Syntax:**

returnType functionName(parameter1, parameter2,…..){

//function body}

**NOTE:**

1. Only names of variables are passed and not their type while calling a function.
2. We pass the values, not variables themselves. Local variables are created in functions which are destroyed on returning from them.
3. A function can be called from any other function and main function.

* **Time Complexity**:

Time complexity of an algorithm quantifies the amount of time by a program to run as a function of length of the input.

* **Space Complexity:**

Space complexity of an algorithm quantifies the amount of time taken by a program to run as a function of length of the input. It is directly proportional to the largest memory your program acquires at any instance during run time.

**NOTE:**

Worst Case[(big oh) notation], Best Case[(big omega) notation], Average Case[(big theta) notation]

* **Array:**

Array is a list of variables of similar type.

**Syntax:**

Datatype array\_name[size]

* **Searching:**  Linear Search{Time complexity: O(n)}, Binary Search{Time complexity: O(log(base 2)n}.
* **Sorting:** Selection Sort, Bubble sort, Insertion sort

**Subarray:** Continuous part of the array. Number of subarrays of an array with n elements= nC2+n=n\*(n+1)/2

**Subsequence:**  A subsequence is a sequence that can be derived an array by selecting zero or more elements, without changing the order of the remaining elements.

Number of subsequence of an array with n elements= 2n.

**Note: \*\* Every Subarray is a Subsequence but every Subsequence is not a Subarray.\*\***

* **POINTERS:-**

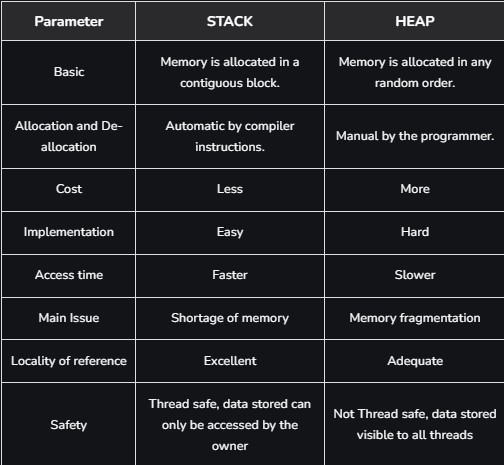
Pointers are variables that store the address of other variables.

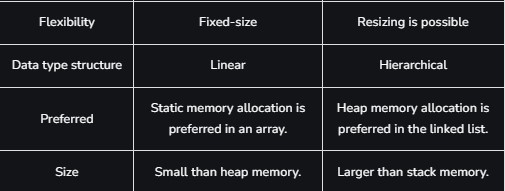
**Pointer Arithmetic:** ++,--, +,-

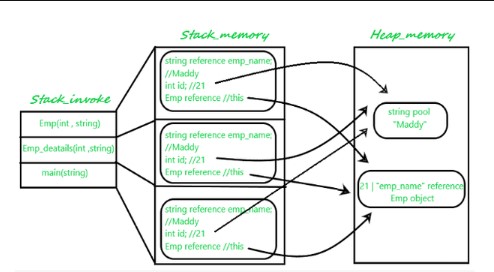
**Pointer and array**

**Pointer to Pointer**

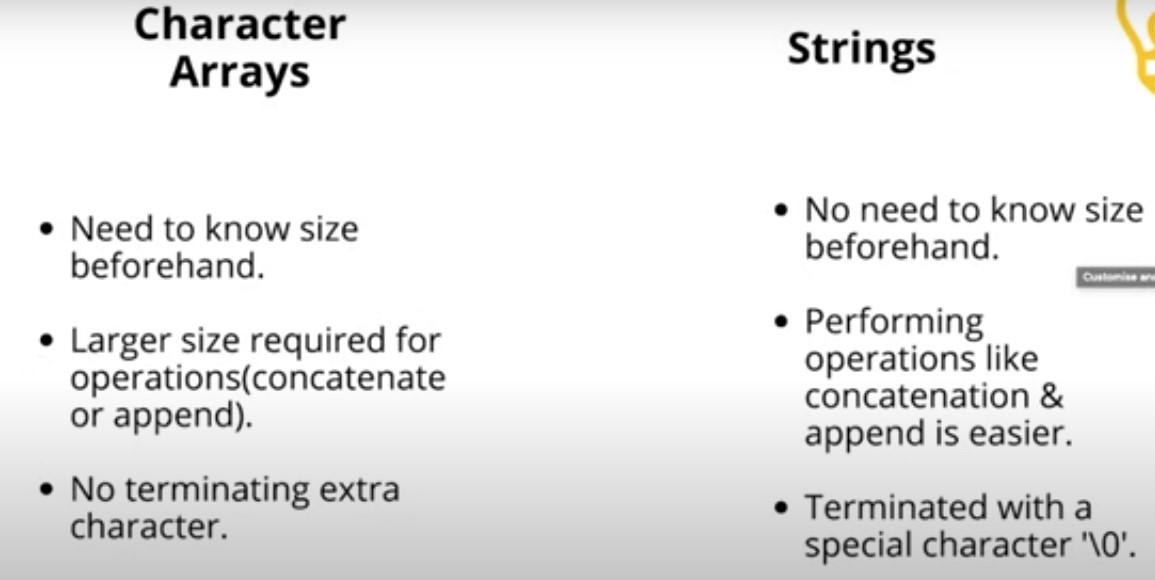
* **STACK & HEAP:-**

****

****



* **STRING:-**



* **BIT manipulation:-**
* **Get bit:**

n=0101

suppose we need to get bit at position, i=2

1<<i=0100

0101&0100=0100

If n&(1<<i)!=0 then bit is 1

* **Set bit :**

n=0101

Suppose we need to set bit at position , i=1

1<<i=0010

0101|0010=0111

* **Clear bit:**

n=0101

suppose we need to set bit at position, i=2

1<<i=0100

~0100=1011

0101&1011=0001

* **Update bit:**

N=0101

suppose we need to set bit at position, i=1

1<<i=0010

~0100=1101

0101&1101=0101

1<<i=0010

0101|0010=0111