Array

Array is a list of same type of elements.
all the elements are in contigous location we can access it through index value.

```
int num[15];   // declare
int num[0] = 1;   // assign
cout << num[0] << num[3];
```

Output

1 0
↳ garbage value.

* when we pass an array to the function we pass the starting add. of an array, so any update will occur in actual array.

char array.

initilizing

```
char ch[10];
```

cin stop executing when it get space, Enter, In ;

for using space we use.

cin.getline (string name, length of string)

ex:- Hello world.

* cin.getline

cin.getline() is a library function used to read the space from the input.

→ method 1

```
#define Max_name_Len 60 // define length
#define Max_addres_len 120

int main(){
  char y_name[Max_name_Len 0],
       y_address[Max_address_len];
                    → put length

cin.getline (y_name, Max_name_len);
cin.getline (y_addrs, Max_addres_len);

return 0;
}
```

→ method 2

```
int main() {
```

```
string s, T;
getline (cin, s);
stringstream x(s);
while( getline(x, T, ' ')) {
    cout << T << endl;
}
return 0;
}
```

→ Input Hello Roshani
output Hello

* char inbuilt functions

1) length

```
int len = strlen(name);
```

2) Compare

```
strcmp(s1, s2)
```

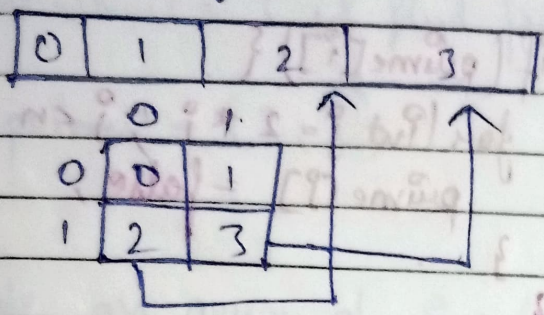
!= 0 (not equal)
== 0 (equal)

3) copy

```
strcpy (dest, src);
```

2D Array.

2D Array is stored in memory as in linear form



→ Creating

```
int arr[row][col]; // create
cin >> arr[i][j]; // input
cout << arr[i][j]; // output
```

Sieve of ~~Eratho~~ Eratosthenes

→ we take no. from 2 to n.
→ and mark all of it as prime
→ than take 2 and mark false all the the element which is divided by its.
→ than move to next., and futher go.
→ finally we get the all prime.

→ Code

```
void sieveeratothens(int n) {
```

```
vector<bool> prime (n+1, true);
prime[0] = prime[1] = false;

for(int i=2; i<n; i++) {
    if(prime[i]) {
        for(int j=2*i; j<n; j=j+i){
            prime[j] = false;
        }
    }
}
for(int i=2; i<n; i++){
    if(prime[i])
        cout << i << endl;
}
}
```

n(log(logn))

segmented sieve

→ Drawback of sieve eratothenes.

1) An array of size O(N) may not fit in memory.

* use divide the range into different segments such that the size of every segment is at most $\sqrt{n}$.
* we have a range [low, high] as input and (high) is very large

→ low = 6 , high = 10

1) Generate all prime number from 0 to floor($\sqrt{high}$) In above example. floor($\sqrt{high}$) = 3, I find prime till 3

primes = {2, 3}

$\sqrt{n}(\log(\log n))$

$\downarrow$

$n(\log(\log n))$

→ Create an array of size (h - l + 1), i.e. (10 - 6 + 1) = 5

→ Mark multiple of prime[] = {2, 3} as false

2 ⇒ 6, 8, 10

3 → 9

left is 7, so 7 is prime.

static & dynamic memory.

static — stack { small, defined }

dynamic → heap { large memory }

whenever we need to create a variable define array we must used heap memory, size in variable define array.
memory is declared after the compilation, at run time. so may be stack is not as capable to contains as that much of memory.
that why we use heap memory allocation for that.

⟹ new keyword is use to define the heap allocation.

```
char *ptr = new char;
```

↓

store the add. of heap memory provide by heap (new char;)

→ Release the allocated heap array

syntax:- delete [] arr;

macro

It is a piece of code in a program that is replace by the value macro. It doesn't contain extra space for it

syntax : # define name expression.

ex: # define X 14

Inline function

It is same as macro there function is get replace just before compilation.

It's a function of one line of code.

syntax: inline int demo(int a, int b) {
return a > b ? a : b;
}

→ No need of extra space.
→ No need of function call override

Recursion

Recursion when a function call itself till the Base case is not true.

Base case is the condition which define when the recursion is to be stop.
Base case is mandatory. (return)

ex :- fibbinacci found.
0, 1, 1, 2, 3, 5, 8

$f(n) = f(n-1) + f(n-2)$

```
int fib(int n){

  // Base case
  if(n == 0)
     return 0;
  if(n == 1)
     return 1;

int ans = f(n-1) + f(n-2);

return ans;
}
```

→ Recursion tree

→ Searching in array :

→ Linear searching in array.

```
int main() {
int n
cin >> n;
int a[n];
for(int i=0; i<n; i++)
  {
   cin >> a[i];
  }
int key;
cin >> key;
cout << linearsearch (a, n, key);
return 0;
}
```

```
int linearsearch (int a[n], int n, int key)
{
    for(int i= 0; i<n ; i++){
        if(a[i] == key){
            return i;
        }
    }
    return -1;
}
```

→ output

```
4                   // size of array.
12,13 14 15         // stored values.
15                  // key to find
3                   // This is index value of 15
```

Linear searching is done in one way from starting to ending in sequence.

→ Binary search in array.

* Concept is Binary search divide the list in to 2 equal part from mid. & check weather the finding element is equal, smaller or greater to the mid no.

according to that if it smaller it goes right or it goes left

do the same with the rest of half of the list.

next we will understand by programs

→ program for Binary search in array.

```
int binarysearch(int a[], int n, int key){
    int s = 0;          // start with index 0
    int e = n;          // end with given size of n

    while(s <= n) // run till start is <= n
    {
①       int mid = s+e/2;

        if(a[mid] == key)
        {
            return mid;
        }
        else if(a[mid] > key)
②           e = mid - 1;
        else
③           s = mid + 1;
    }
    return -1;
```

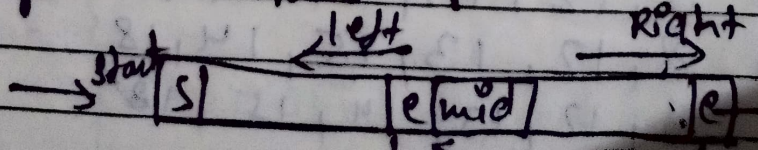
```
}
int main() {
    int n;
    cin >> n;
    int a[n];
    for(int i=0; i<n; i++)
    {
        cin >> a[i];
    }
int key;
cin>>key;
cout << binarysearch(a, n, key);

return 0;
}
```

* (explanation.)

1) find the mid of term of the list

2) e = mid - 1 mean when the mid term is greater than "key" then it has to search at the left side of the list
that why the end term is shifted to mid - 1



3) Now here s = mid+1 is same as logic in (2) eg.

when mid is smaller than key the start index shift to "mid+1"