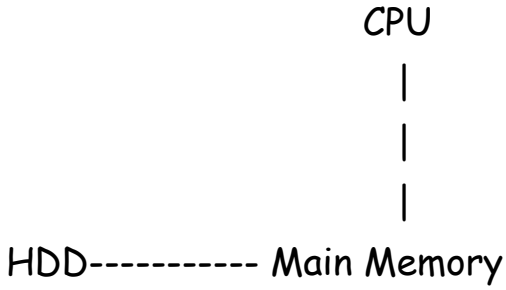
Writing the entire program inside the main function is known as Monolithic programming and breaking down it into different parts(functions) is known as procedural programming or Modular programming

When Data is big and Commercial it is Relational Data. Data is stored in harddisk (permanent Storage).---> DataBase

Relational Data :
1)operational Data : Data used on Daily Basis.
2)Legacy Data: OLd Data used rarely

DSA

Data Structures are Formed Inside the **Main Memory** during the execution of a Program. **Data Structures** are the arrangement of data inside the main Memory.



Let the size of Memory is 64KB i.e of a segment size.
The main Memory [64kB (0-65535)] is divided into three sections namely **Code section, Stack, Heap.**

The **area occupied by the program** in the main section is known as **Code section**.The machine code of the program is loaded here.

The **memory for the variables declared in the main function** is allocated in **stack**. That section of stack is known as stack Frame or Activation Record.

The memory is allocated according to the function calls starting from the main function; after executing all the function calls it deletes one

by one and deletes the main function at last (Similar to stack in microprocessors) hence it is named Stack.

**Memory is also created for the parameters passed i.e formal parameters

Static Memory Allocation:
The amount of memory required is decided at the compile time only. This memory allocation is done automatically; Programmer has nothing to do with it.

Heap is just piling up (kuppa). It is used for Unorganised Memory. It should be used like a resource (when required we take it and else we release it). A program can never uses the heap memory directly; it is used indirectly with the help of pointers.

Memory allocation and deallocation :
Memory is created in heap using a "new" keyword.

```
int *P;                    //pointer (address variable) is created in stack

P = new int[5];        // an array of size 5 is created in heap memory      // C++
P = (int *) malloc(5*sizeof(int))                                          // C

                              // int *P = new int[5];          -> in a single line of C++
delete  [] P;          // array inside the heap memory is deleted      //C++
free(P);                                                               // C
```

P =null; // pointer stops pointing to the heap memory

If we do not deallocate/release the heap memory it leads to memory leakage/loss.

Abstract Data Types (ADT):

Any class in C++ that has data representations and operations is an Abstract Data Type.

Recursion :

Tracing Tree of Recursive Function is a way we represent recursive functions with lines and nodes.

In a Recursive Function any statements **before** the function call are executed during **call time (Ascending)** and statements with and **after** the function call are executed during the **return time (Descending)**.

Every Recursive function has base cases and the function gets terminated at the base case.

Recursive Functions utilize stack(internally); hence they are Memory Consuming Functions. For input-**n** there will be **n+1 recursive calls** i.e. n+1 Activation Records.

Time Complexity is found using Recurrence Relation; followed by Induction Method (Successive Substitution Method).

Static Variables in Recursion :

Static Variables are created only once i.e at the beginning/loading of a program. It will not be created for every function call.Static Variables are created in the code section of the main memory. Only a single value exists for a static variable even after update. The same applies for Global variables as well.

An iteration can be converted into a recursive function and vice versa. The main difference between an iteration and Recursive Function is that an iteration will have only ascending phase whereas Recursive Function will have both ascending and descending phase.

Types of Recursion:

1. Tail Recursion
2. Head Recursion
3. Tree Recursion
4. Indirect Recursion
5. Nested Recursion

If the **Recursive call** in a Recursive Function is the **last statement** then it is a **Tail Recursion**.All operations will be performed in calling time and no operation is performed at return time.

The time complexity of both loop and recursive function remains same [o(n)], but the space complexity varies [o(n) for recursive function and constant for loop].

It's good to use an iterative statement for tail recursion as it consumes less space.

If the **Recursive call** in a Recursive Function is the **first statement** then it is a **Head Recursion.** All operations will be performed in return time and no operation is performed at the time of calling. Head Recursions can also be converted into loops but not through similar statements a different approach is required for loops to obtain the same result.

It's good to use a recursive function for head recursion as iteration consumes more space.

If a **Recursive Function is calling itself more than one time** then it is known as **Tree Recursion.** The time complexity of recursive function is depends on no.of recursive calls (for 2 calls it is [o(2^n)]) and **s**pace complexity is height of the tree i.e. **O(n)** .

If a recursive function has more than one function and is calling one another in a circular fashion then it is an Indirect Recursion.

If a **recursive function passes a recursive function as a parameter** in the **recursive call** then it is a **Nested Recursive Function.**

Recursion Examples :

1. Sum of n numbers
2. Factorial of a number
3. Power of a number
4. Taylor Series
5. Fibonacci Series
6. Towers of Hanoi
7. Combination

Power of a number:

Method 1 :

```
int pow(m,n)
{
  if(n==0)
   return 1;
else
  return m* pow(m,n-1);
}
```

Method 2 : Efficient Method

```
int pow(m,n)
{
  if(n==0)
   return 1;
else if(n%2==0)
 return  pow(m*m, n/2);
else
  return m* pow(m*m, (n-1)/2);
}
```

Taylor series:

Method 1 :

```
int e(x, int k)
{
  static int p=1,f=1;
  int r;

   if(k == 0)
    return 1;
  else
   r = e(x,k-1);
   P = p*x;
   f = f*k;
  return  r+ p/f;
}
```

Method 2 : Efficient Method

```
int e(x, int k)
{
  static int s =1;
  if(k==0)
  return 1;
else
 s = 1 + s(x/n);
return e(x,n-1);
}
```

Method 3 :

```
int e(x,int k)
{
   int s =1;
  if(k==0)
  return 1;
else
for(;n>0;n--)
   {
       s = 1 + s(x/n);
```

```
    }
}
```

Combination:

```
 int  c(int n, int r)
{
    if(r==0 || n==r)
     return 1;
   else
  return  c(n-1,r-1) + c(n-1,r);
}
```

Towers of Hanoi:

```
 int Toh(int n, int A, int B, int C)
{
    if(n>0)
    {
    Toh(n-1, A, C, B);
     printf("from %d to %d", A,C);
     Toh(n-1, B, A, C);
  }
}
```

Fibonacci Series:

Method 1 :

```
int fib(int n)
{
    if(n<=1)
     return n;
  else
     return  fib(n-1) + fib(n-2);
```

```
}

Method 2 :
int fib(int n)
{
   int a,b,c,i;
    if(n<=1)
     return n;
  else
   for(i=0;i<=n;i++)
   {
      c = a+b;
       a  = b;
       b =  c;
     }
  return c;
}
```

A recursive function calling itself multiple times for the same values is known as excessive recursion. Ex: Fibonacci Series

In this type of recursion we just need to avoid the repeating calls while moving in order

Arrays:

- It is a collection of similar data type elements grouped under one name. Memory is allocated contiguously.
- It is a vector. Indices begin from 0.An element can be accessed by its index. A[i] or i[A] or *(A+i)
- **Default** Initialization with **0** happens when all the elements of specified size are not initialized .
- Size once declared can't be altered for a static Array; whereas the Dynamic Array has variable size.
- In C programming the size of an Array is decided at compile time and memory of an Array is allocated at runtime.
- But in C++ we can decide the size & memory at the run time itself .

Allocating an array:	de-allocating an array
int *p; p = new int[5];	delete []p; p = null;

Array ADT

- Searching an element in an unsorted array :
  - T(n) = O(n)
  - S(n) = O(1)
- Searching an element in a sorted array :
  - T(n) = O(logn)
  - S(n) = O(1)
- Inserting an element in an unsorted array
  - T(n) = O(n)
  - S(n) = O(1)
-

Data : Array Space, size, length(no.of elements)

- Length of an array and its size need not be the same always.

Operations : Display(), Add/Append(ele), Insert(index,ele), Delete(index), search(x), Get(index), Set(index), Max()/Min(), Reverse(), Shift()/Rotate().

All these **operations result in an alter in length of the array**. So **accordingly length should be incremented or decremented** after the operation.

Sorting

Merging:

It is a binary operation; it means there would be a requirement of two arrays; some more binary operations are Append(), concat(), compare(), copy().

Merging can be done only on sorted lists

Set Operations : Union ,Intersection, Difference, Set Membership

Strings:

The end of a string is known by its length in Java. In c/c++ a string ends with a **null character ['\0']**. It is also known as string delimiter, end of a String class, string terminator

Without the null character they are just an array of characters. The append of a null character to a character array makes it a string.

Declaring a string

Char name[10] = {'J','o','h','n','\0'} ; or

Char name[] = {'J','o','h','n','\0'} ; or

Char name[] = {"John"};

Char * name[] = {"John"} ;

No other keywords or functions are required to declare a string inside heap memory.

cout function works until it finds a \0 and cin function will automatically include \0 we no need to mention explicitly.cin function stops accepting string values when it finds a space. gets() comes to the rescue it can accept string values with multiple spaces.

1. Length of a string
2. Toggling of characters in a string
3. Validation of a string
4. Reversing a string
5. Comparing strings and Palindrome check