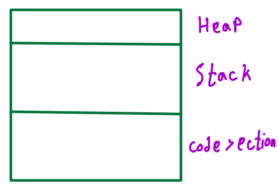
data structure is Performed in the main memory (Ram)

activation recored of a Function: the Portion of memory taken by a function, Programm.

Stack memory: How many bytes of memory is required by this Function was decided at compile time.

Static: Size of memory | when decided? Compile time | LIFO

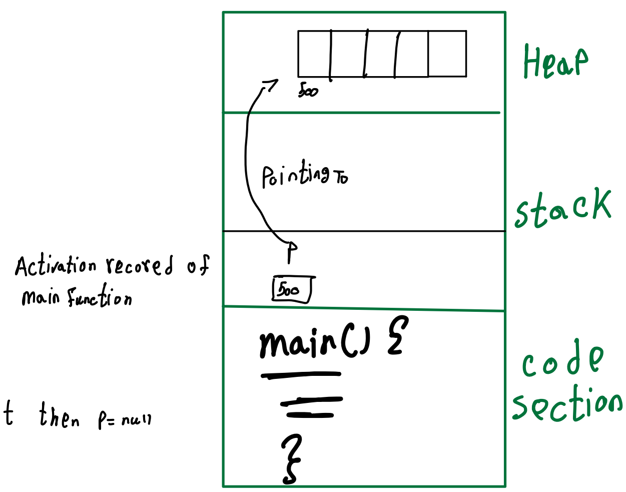
As we Know combiler divid main memory (Ram) into 3 Pieces (code section / Stack / Heap)



Programs can't directly Access Heap memory, but it can using Pointer

```
void main() {
    int *p;

    p = new int [5];
// if we want to delete the Array First delet contant then p= null
// if directly p=null will cause memory leak
    delete [ ]p;
    p= null;
```



Conclusion, stack is static Allocation vs Heap dynamic Allocation

---

Data structures implemented using 2 Physical datastructure (Array, linked list)

Stack, Queue ⟶ liner

Tree, Graph ⟶ non-liner

Hash ⟶ tabuler or liner

---

time & Space complexity

```
Void swap(x,y)
{
  int t;
  t = x;  ⟶ 1
  x = y;  ⟶ 1
  y = t;  ⟶ 1
}
```

$f(n) = 3n^0 = 3$

$O(1)$ ⟶ أس n

```
int sum(int A[], int n)
{
    int s, i;
  s = 0;  ⟶ 1
```

$\text{for}(\underbrace{i=0}_{1};\ \underbrace{i<n}_{n+1};\ \underbrace{i++}_{n})\ \underbrace{S = S + A[i];}_{n}$

$\underbrace{\text{return } S;}_{1}$

}

$f(n) = 2n + 3 \rightsquigarrow$ constants

$O(n)$

recursion 5

```
Void fun2(int n){
    if(n>0)
    {
        fun2(n-1)
        cout<<n;
    }
}
```

```
void main(){
    int x=3;
    fun2(x);
}
```

O/P ⟶ 1 2 3

```
Void fun2(int n){
    if(n>0)
    {
        cout<<n;
        fun2(n-1)
    }
}
```

```
void main(){
    int x=3;
    fun2(x);
}
```

O/P ⟶ 3 2 1

Every recursion should have

condition to terminate function

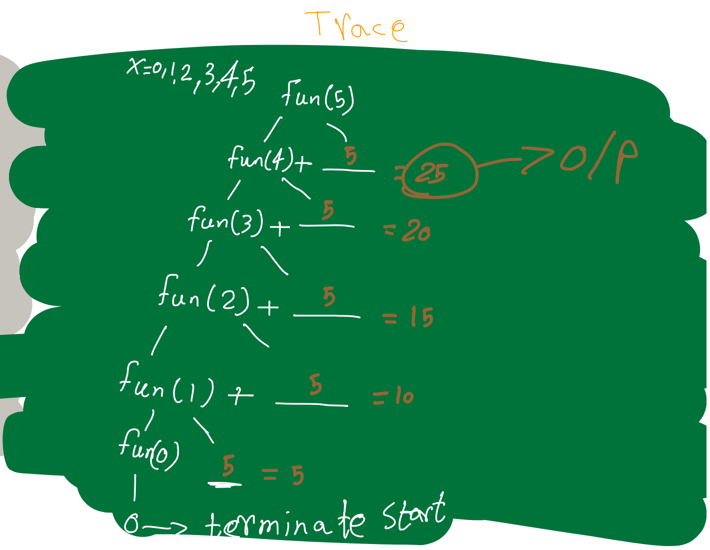
Static & global variables only have 1 copy, so its intialize once only.

To trace recursion use tree teqnige, if the variable is static or global don't put it in the tree, Ex:-

```
int fun(int n){
   if (n>0){
   x++;
   return fun(n-1)+x;
   }
 return 0; }
int main(){
int a=5;
cout << fun(a);
}
```

أول يعدي الكول
وآخر شيء إذا
رجع يضيف x

X is global static



Types of recursion

Tail - Head - tree - indirect - nested

Tail: when the call is last statement in the function

```
fun(){
=
=
fun();}
```

every loop can be recursivefunction and visaversa

```
v oid fun(int n)
{
 if (n>0)
 {
 cout << n;
 fun(n-1)
 }
 }
```

```
void fun(int n)
{
while (n>0)
{
  cout << n;
n--;
}
}
```

loops is more efficient than recursive in space, time both same

---

Head : calling is first statement

```
Void fun(int n) {
if(n>0)
{ fun(n-1);
  ==
}}
```

```
void fun(int n){
   if (n>0){
   fun(n-1);
  cout << n;
   }}
```

```
Void fun(int n){
 int i= 1;
 while(i<=n)
{
 cout << i;
 ++i;
}
}
```

Head recursive cannot directly converted to loop

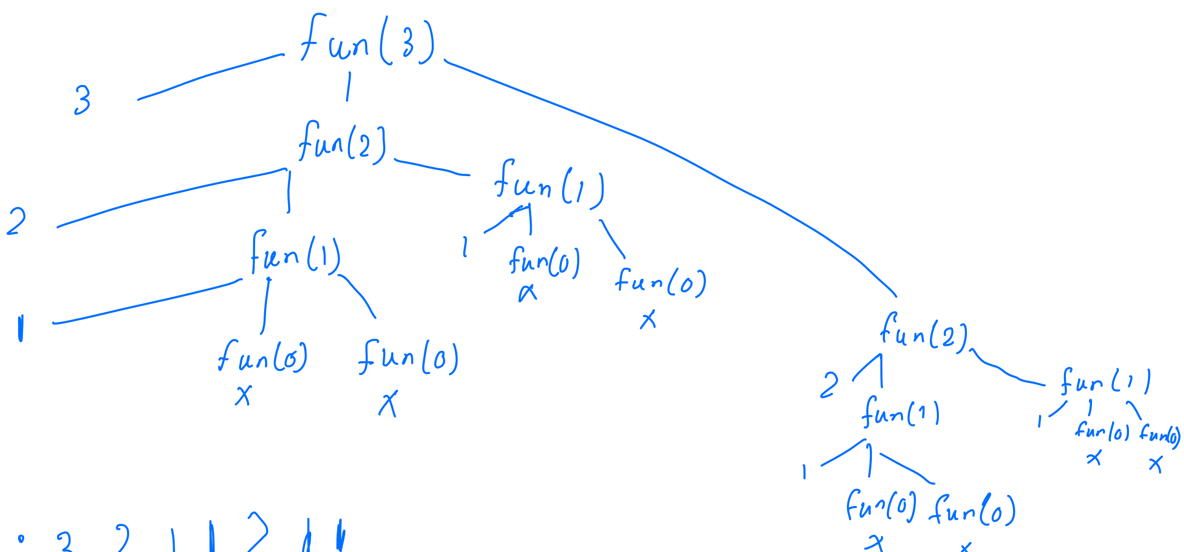
Tree: that call itself more than 1 time

```
fu(n){
 if(n>0){
  fun(n-1);
  - -- . --
  fun(n-1);
}}
```

Ex:-

```
void fun(int n){
  if(n>0){
  cout<<n;
  fun(n-1);
  fun(n-1);
  }
}
```

let's consider in main fun(3);, let's trace it

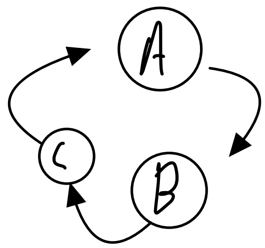


O/P: 3 2 1 1 2 1 1

Time complexity $O(2^n)$

Space complexity $O(n)$

Indirect recursion: Function calling another one like cycle




```
void A(int n) {
  if (n>0)
  {
    cout<<n;
      B(n-1);
  } }
```

```
void B(int n) {
   if (n>0)
  {
   cout<<n;
   A(n/2);
   }
  }
```

A(20)
20 — B(19)
19 — A(9)
9 — B(8)
8 —

O/P:
20 19 9 8 4 3 1

A(4)

4 — B(3)

3 — A(1)

1 — B(0)

x

Nested: Parameter of call is recursive

```
int fun(int n)
{
    if(n>100) return n-10;
    else return fun(fun(n+11));
}
```

fun(95)

fun(95)

↓

fun(fun(106)) fun(106) = 96

=

fun(96)

↓

fun(fun(107)) fun(107) = 97

fun(97)

↓

fun(fun(108))   fun(108) = 98

↕

fun(98)

↓

fun(fun(109)]   fun(109) = 99

↕

fun(99)

↓

fun(fun(110))   fun(110) = 100

↕

fun(100)

↓

fun(fun(111))   fun(111) = 101

↕

fun(101)

|

91

Conclusion types of recursion

1- tail : call statement is the last statement, nothing excuted in returning, excusion only in calling.

2- Head : call is the first statement in the function.

3- tree : more than one call in the function

4- indirect : functions make a cycle, each one call another one until finally back to first function

5- Nested: in calling statement Another call and both to the Same function

Practice

Power using recursive

$m^n = m \times m \times m \times \underbrace{\underline{n-1 \text{ times}}} \times m$

$m^n = Pow(m, n-1) \times m$

$$Pow(m,n) = \begin{cases} 1 & n = 0 \\ Pow(m, n-1) \times m & n > 0 \end{cases}$$

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

Pow(2, 3)

Pow(2,2) x 2 —> 8

Pow(2,1) x 2 —> 4

Pow(2,0) x 2 —> 2

1

Sum to n

$Sum(n) = \underbrace{1 + 2 + 3 + n-1} + n$

$Sum(n) = Sum(n-1) + n$

$$Sum(n) = \begin{cases} 0 & n = 0 \\ Sum(n-1) + n, & n > 0 \end{cases}$$

```
int sum(int n) {
  if(n==0) return 0;
```

sum(3)

3 sum(2) + 3 -> 6

1 sum(1) + 2 -> 3

0 sum(0) + 1 -> 1

0 x

```
sum(n-1)+n
}
```

factorial

$$f(n) = 1 * 2 * 3 * \dots n-1 * n$$

$$f(n) = f(n-1) * n$$

$$f(n) = \begin{cases} 1 & n == 1 \\ f(n-1) * n & n > 0 \end{cases}$$

f(5)

f(4) * 5 -> 120

f(3) * 4 -> 24

f(2) * 3 -> 6

f(1) * 2 -> 2

1 x

```
int f(n){
if(n==1) return n;
f(n-1)*n;
}
```