Memory Allocation

Data → Memory → static allocation/compile time
→ Dynamic allocation

① Compile time Allocation → you are basically deciding this much will be the memory requirement and I will map these variables at these particular location in the RAM.

→ during compilation compiler makes a table called symbol table, and in that symbol table it will map the variable name with virtual address

→ sizes & locations is fixed during the compile time

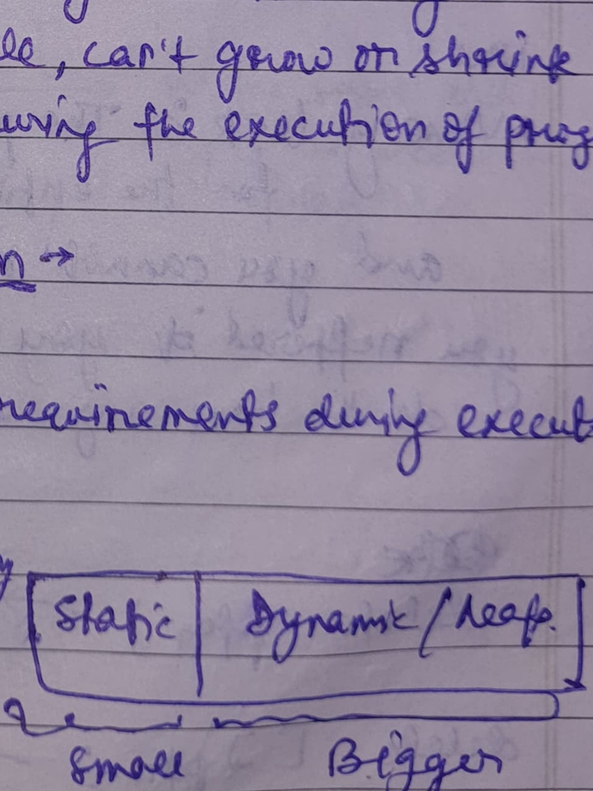
→ advantages:- faster than dynamic memory allocation
disadvantages:- less flexible, can't grow or shrink the memory during the execution of programe

② Runtime / Dynamic allocation →

→ allow us to define memory requirements during execution of prg.

→ ~~Heap~~ It uses heap memory

→ we can reuse the heap memory



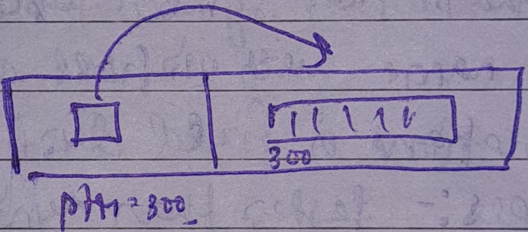
→ new → allocate
delete → free
→ when to allocate → programme → code.
when to free

static
↓ heap

eg int *ptr = new int ;

↳ this method is usefule when you want to create an array

int *ptr = new int [10] ;



Now

Memory leak :- The part of the memory is now occupied for the entire duration of the programme and you cannot reffer/reuse this memory because you reffered it you forgot to delete it and you don't have any way to reach this.

delete ptr ; → for single variable

delete [ ] ptr ; → for array

example → //Allocation

```
int b[100];
cout << sizeof(b) <<endl;
```

//Dynamic allocation (on the fly)

```
int n;
cin >>n;
int *a = new int[n]; → or   new int[n]{0};
cout<< sizeof(a) <<endl;
```

all n are initialize with 0.

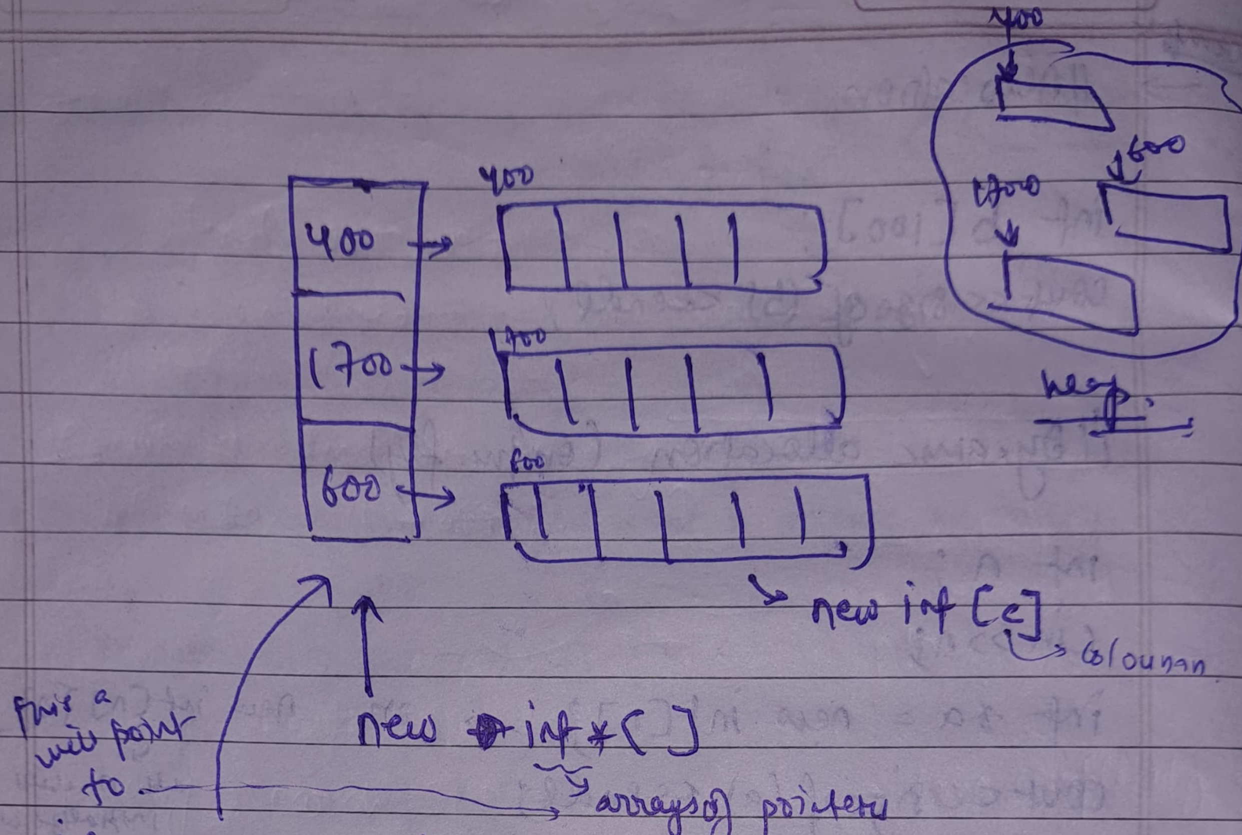
//No change

```
for (int i=0; i<n; i++) {
    cin>> a[i]; → in the same old way I can access the array.
    cout<< a[i] << " ";
 }
```

// free up the space

```
 delete [] a;
```

* 2D array → there no way to create 2D array directly so.



int * *a = new int*[ ].

eg

```
int **arr;
int r, c;
cin>>r>>c;
```

//create an array of row heads.

```
arr = new int*[r];
```

// create an 2d array

```
for (int i=0; i<r; i++) {
    arr[i] = new int[c];
}
```

{0} → initializ all with 0.

//take input and print the elements

```
int val=1;
for (int i= 0; i<n; i++) {

  for (int j=0; j<c; j++) {
        arri[i][j]= val;
        val++;
        cout << arri[i][j]<<" ";
  }
```

also //No change same old way to use 2D array.

⇒ we should never return a local variable

ie
```
#include<iostream>
using namespace std;
int* fun() {
  ~~int a=new int[5]~~ int a[] = {1, 2, 3, 4, 5}
  cout<<a<<endl;
  cout<<a[0]<<endl;
  return a;
}

int main() {
  int *b= fun();
  cout<<b<<endl;
  cout<< b[0]<<endl;
  return 0;
}
```

→ we only get 1/garbage / segmentation fault

because as soon as fun function call is over the ~~memory~~ stack frame is cleared from memory array is cleared. to avoid this use dynamic allocation