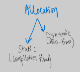
☆ Run-time Memory Allocation / Dynamic M.A

→ allows to define memory requirement during Runtime

Memory

→ 3 Different pool of Memories

1. Static → Global variable, permanent for the entire run of the program
2. stack → Local variable Storage (automatic, continous Memory)
3. Heap → dynamic storage (Large pool of memory, not allocated in contiguous order)



Dynamic Memory Allocation

☆ Static Memory Allocation / Compilation time M.A

In, this the memory is not actually allocated at the compilation time (into the RAM) but the Sizes (how much memory will be needed) and Location (Address of the variable) is fixed at the compilation time.

During Runtime, Mapping b/w Virtual Memory and physical Memory where actual allocation takes place is performed just before program execution.

Size's and Location are stored inside the Symbol table

Eg.
```
int main()
{ int x;      4bytes
  char ch;    1bytes
  int a[100]; 400 bytes
}
```

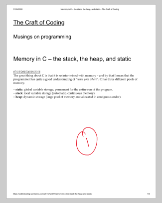
Symbol table (Sizes and Location)

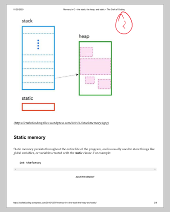
a	400
ch	401
x	402

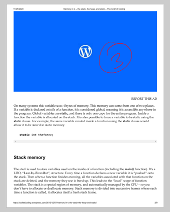
☆ Allocation takes place inside the Stack Memory

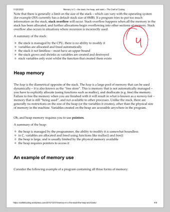
ADVANTAGES Vs DISADVANTAGES of Static Memory Allocation

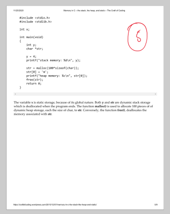
1) Fast | 1) Less flexible (Cannot grow or shrink

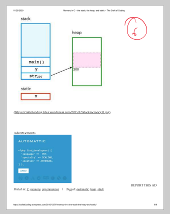












→ Reuse

→ heap memory is used for D.M.A

→ keyword in C++ → new, Delete → Given control to Programmer
(new ↓ Allocate, Delete ↓ Deallocate.)

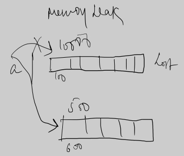
Eg.
```
int main ()
{  int *p = new int;   // new int returns address
                       thus, p will store the
                       address of the new variable.
   (int *p → static, new int → Heap)
   return 0;
}
```

```
// Array allocation
int a[1000];
cout<< sizeof(a);
// this will give the size of Array
```

```
int *a = new int[10000];
cout<<sizeof(a);
// this will give the size of pointer a.

a = new int[500]; //Memory leak
```

Memory leak



To avoid memory leak use. delete or Store the previous address to some other pointer. and after the use delete operator should be used.

```
// delete operator

int *p; int *x = new int;
p = new int a[500]; int val=1;
for (int i = 0; i< 500 ; i++)
{   *(p+i) = val;
    val++; }

delete []p; // for array
delete x; // for variable
```