Dynamic Memory: (heap) → Runtime request → return | Programmer → when to allocate, when to deallocate

Dynamic Variable: 1) has no name, but (size, content, Address)
2) created at runtime, during program execution] compiler dependent.
3) process and destroyed (deallocated) through pointers.

operator: (new) → reserve word
→ creation (allocation of dynamic memory) C++

1) request for allocation of memory (variable)
2) on success return address of allocated memory.

```
int main()
{
  int *p = new int;
  cout<< p << *p;
}
return 0;
```

① 4 bytes of memory (heap)
② address return.

011 | 100 | heap 4 bytes
p | 011

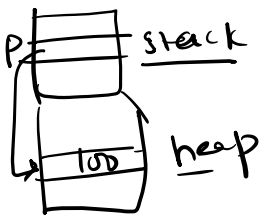
011 JUNK

int a;
{ a = 100;

int a = 100;

initialization.

① int *p = new int;
*p = 100;
cout<< *p;



② int *p = new int(6);
055 | 6
1) create
2) initialize
3) Address
p | 055
cout<< *p; 6

*p = *p + 1;

---

operator → (delete) for deallocation of dynamic memory.
reserve word.

1 → when needed can be used.
2 → it requires a valid address of allocated memory on heap.
3 → can only use for heap memory.

int *p = new int(6); all

{ cout<< *p;

055 deallocated dangling pointers

p | 055 → nullptr

if a pointer contains the address of a
1 deallocated or
2 unallocated memory.

delete p; deallocate.
p = nullptr;
cout<< p; → 055
cout<< *p; Junk value
*p = 100; ✗ Runtime error.

Errors:- delete → can use only once for a variable.

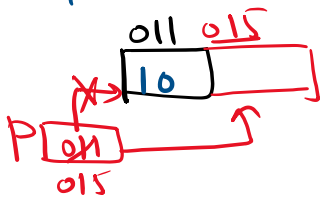
1) int x = 10;
int *p = &x;
delete p; ✗
p = nullptr;
Runtime Error
cannot deallocate stack memory.

2) int *p = new int(10);
delete p;
*p = 50; ✗
Runtime error
cannot write into deallocated memory.

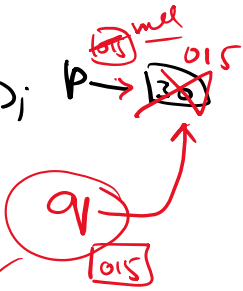
3) int *p = new int(150);
delete p; ✓
delete p; ✗
Runtime error.
cannot deallocate a memory multiple times.

Dangling pointers: → if contains addrss of unallocated or deallocated memory.
→ currupt data.
→ program termination with runtime error. (fatal)

int *p = new int(10);
p++; → dangling
delete p;
① *p = 100; R-E
② R-E unallocated memory addrss

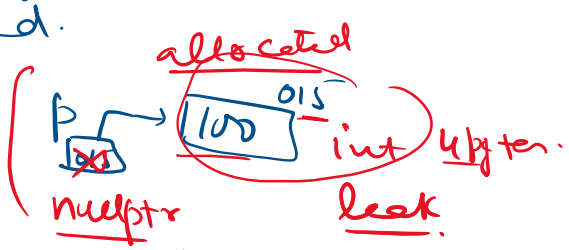


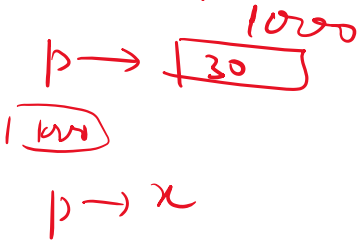
int *q, *p;
p = new int(30);
q = p;
delete p;
p = nullptr;
① *q = 500; dangling
② delete q;
Runtime error



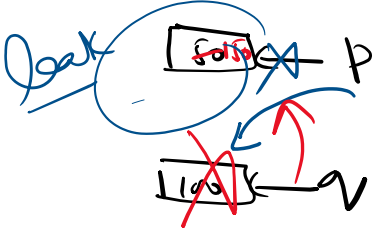
Memory leak: A portion of memory (variable dynamic)
→ which cannot be used (accen)
→ allocated for other procenes.
→ if we loose the track of dynamic memory addrss.
→ it will remain allocated.

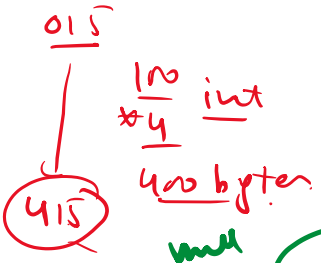
int x = 10;
int *p = new int(100);
1) p = nullptr;
2) p = new int(30);
3) p = &x;





```
int *p = new int(50);
int *q = new int (100);
*p = *q + *p;
```





1) p++;

2) p = p + (*q);

int

Jump

(*p) = (*q) * (*q) + (*p);

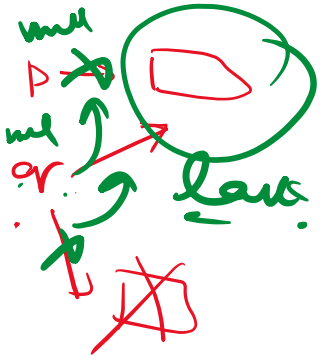
Runtime error

Junk

3) p = q; or q = p;

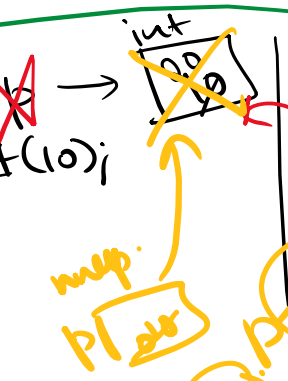
4) delete q;

```
q = p;
p = nullptr;
q = p;
```



return pointer from function

```
int * fun()
{
    int *p = new int(10);
    *p = 20;
    return p;
}
```




```
int main()
{
    int *q = fun();
    cout << *q;
    deallocate(q);
    cout << *q;
    *q = 100;
}
```

Address

20

q = nullptr;

20

Junk

Runtime error

```
void deallocate(int *p)
{
    delete p;
    p = nullptr;
}
```