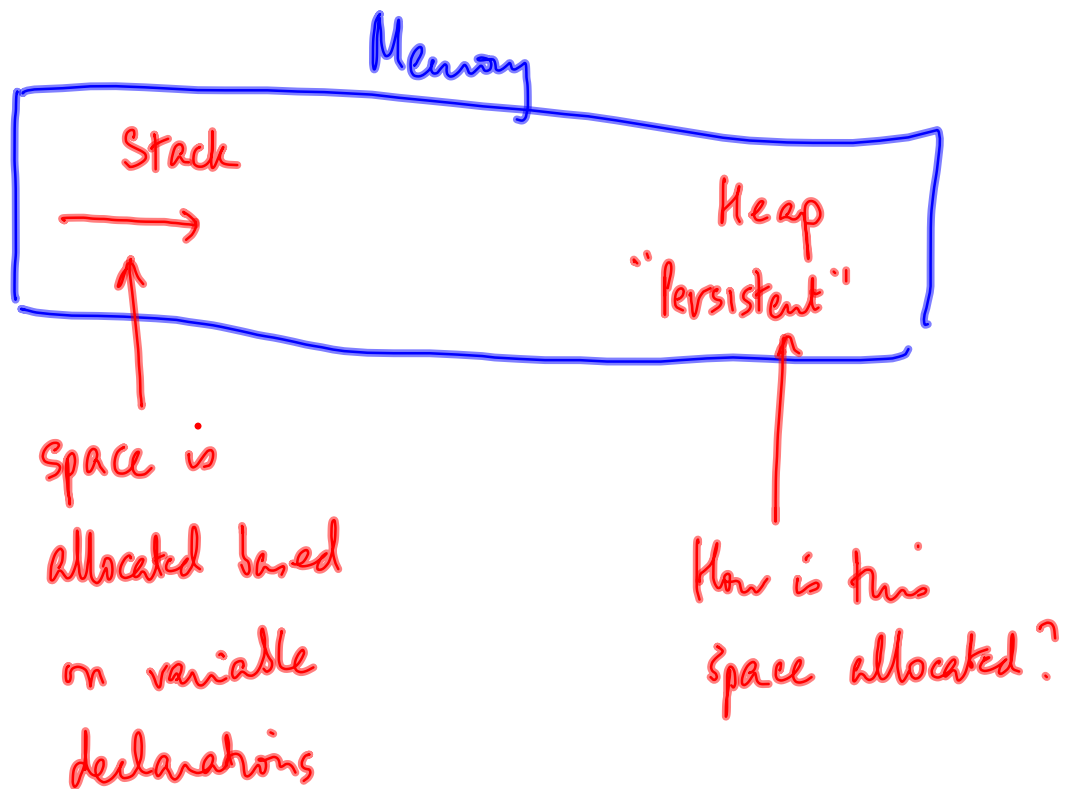


Memory management



```
class Node:
```

```
    def append(self, n):
```

```
        :
```

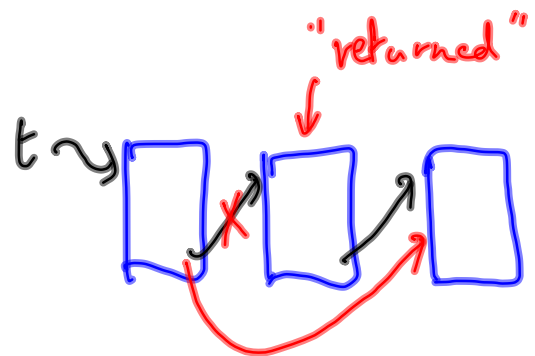
```
        t.next = Node(n)
```

← implicitly creates
space on the "heap"

```
    def delete(self, n):
```

```
        :
```

```
        t.next = t.next.next
```



In C:

malloc

allocate some space (how much?)
return a reference/pointer to
the allocated space

`t.next = Node(x)`

↳ allocates space for one Node
reference is returned

ptr to new space **malloc** (size info)

We need to obtain size of a datatype

`sizeof(int)`

`sizeof(float)`

Can then use this like a number

`int *a;`

`a = malloc(75 * sizeof(int))`
↑ (not quite!)

`*a+i` is same as `a[i]` etc

Weak type of polymorphism

return type of malloc is (void *)

int *a; Type of a is (int *)

generic pointer

Use explicit type conversion = "type casting"

int x, y; float z;

z = ((float) x) / y ;

```
int *a;
```

```
a = (int *) malloc (75 * sizeof(int));
```

Returning free space to the heap:

```
free(a);
```

a must point to space got from
previous malloc

What would `free(a+5);` mean?

malloc may fail

returns NULL - for pointers, C equivalent
of Python None

Should always use malloc as follows:

```
a = (int *) malloc( -- );
```

```
if (a != NULL) { -- }
```

Defining dynamic data structures

Need to define "units" such as



struct pair {

int x;

int y;

};

name of new user defined type

struct pair p, q; - defines/declares two concrete variables of this new type

Access components of struct using .

p.x = 17;

q.y = p.x;

:

Recursive datatype

```
struct node {  
    int value;  
    struct node *next;  
}
```

A function to create a node with a given value

```
(struct node *)createNode (int v) {  
    struct node *temp;  
    temp = (struct node *) malloc (sizeof (struct node));  
    if (temp != NULL) {  
        (*temp).value = v;  
        (*temp).next = NULL;  
    }  
    return (temp);  
}
```

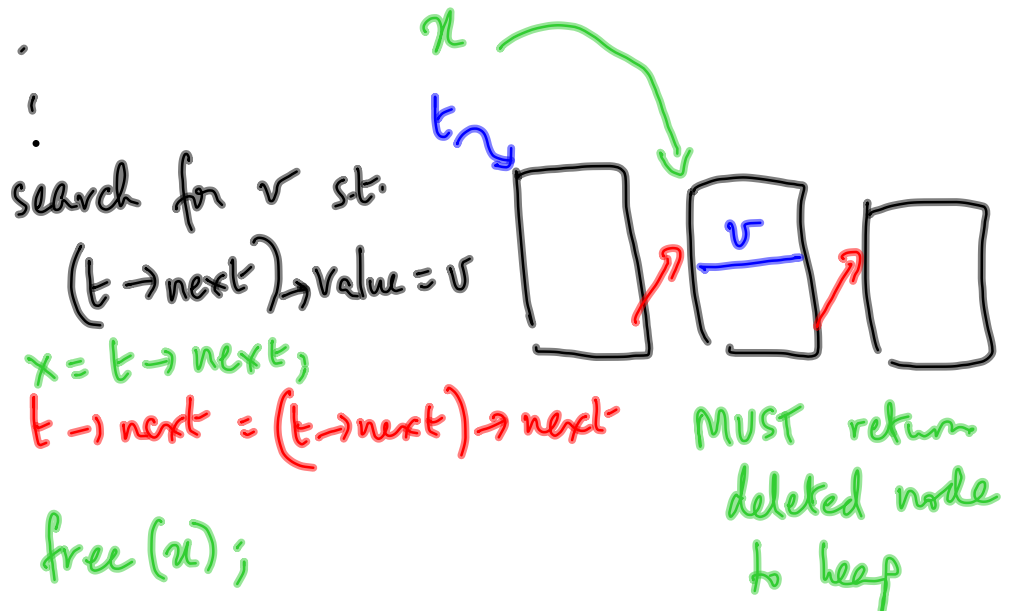
$(*p).v$ occurs very often

equivalently written as

$p \rightarrow v$

```
:  
if (temp != NULL) {  
    temp → value = v;  
    temp → next = NULL;  
}
```

... `delete (struct node *l, int v) {`



Function prototypes

Give a summary of a function's arguments & return types before it is used
(unless return type is int)

```
int factorial(int);  
int main() {  
    i ← factorial(x)  
}  
int factorial(int n) {  
    ≡  
}
```