Outline

- → Structures recap
- → Pointers to Structures
- → Memory
- → Dynamic Memory Allocation
- → LINKED → LISTS !

Structures

```
struct student {
    int id;
    char *name;
};
```

```
struct student dale;
dale.id = 54;
dale.name = "Dale";
```

```
struct student dale = {54, "Dale"};
struct student dale = {.name="Dale", .id=54};
```

Pointer to a struct

```
struct student *s_ptr;
s_ptr = &dale;
```

dale . ID = 100 ;

(or)

(*s_ptr). ID = 100;

(or)

s_ptr -> ID = 100; ->

Memory

Stack

1) Stack stores variables created in functions (incl main)
Eg: int a = 4;
int b;
2) Use it when you know the size of the variable at compile time
3) Fixed
4) Automatically managed

Heap

1) Have to request allocation during runtime. (that request may be denied)
2) Can grow
3) Manually allocate & deallocate it

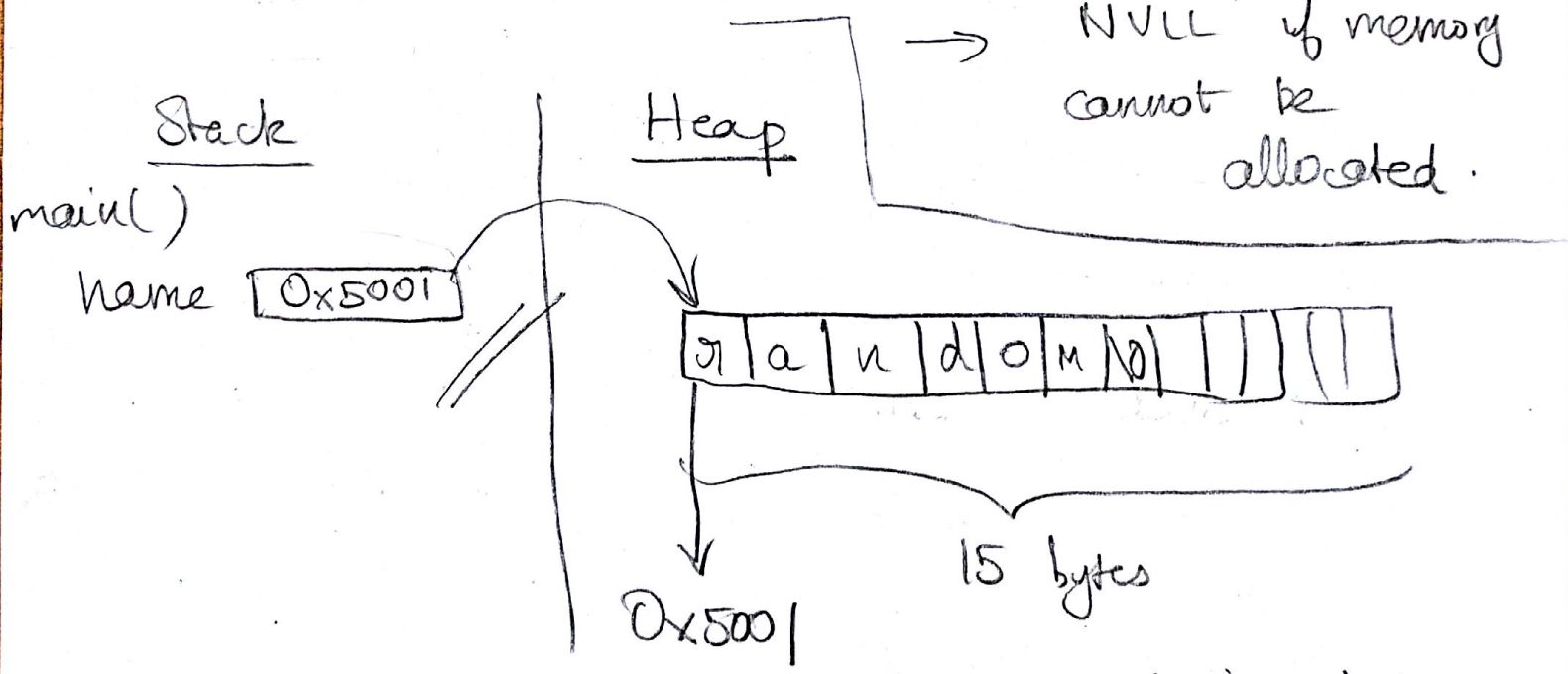
How do we allocate / get memory on the heap?

malloc

→ allocates the requested memory (in bytes) in the heap and <u>returns a pointer</u> to it

```
(void *) malloc (size_t size)
```

→ NULL if memory cannot be allocated.

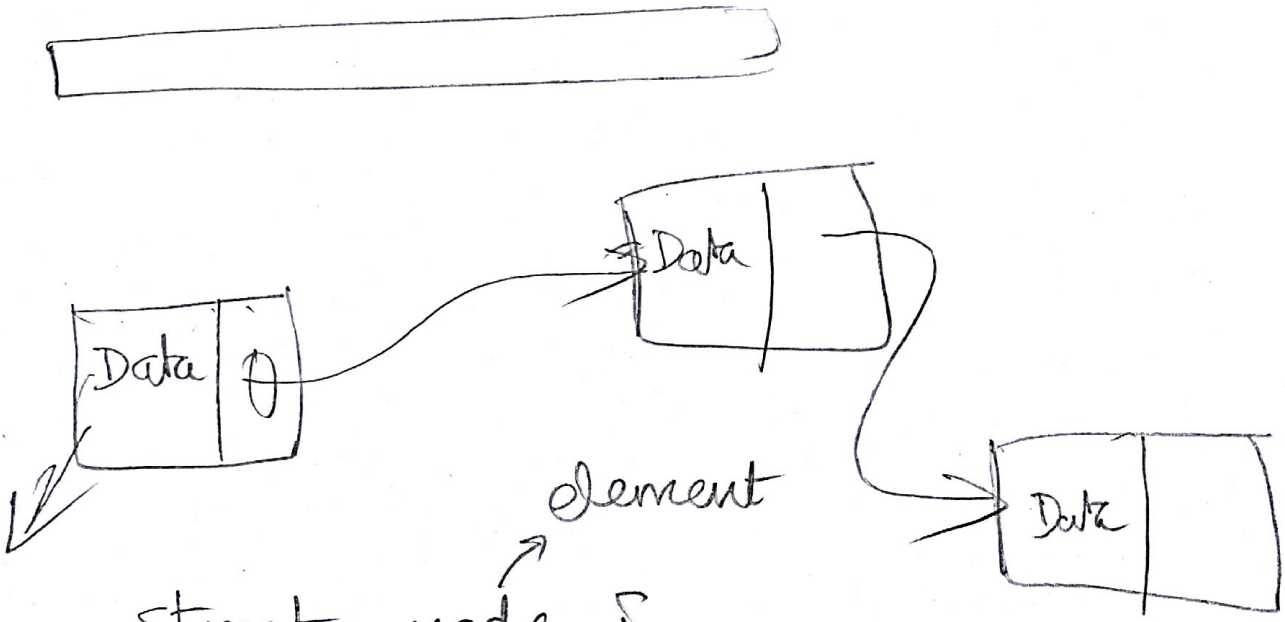


String.h

```
char * name;
name = (char*) malloc (15);
name = "random";   strcpy (name, "random");
```

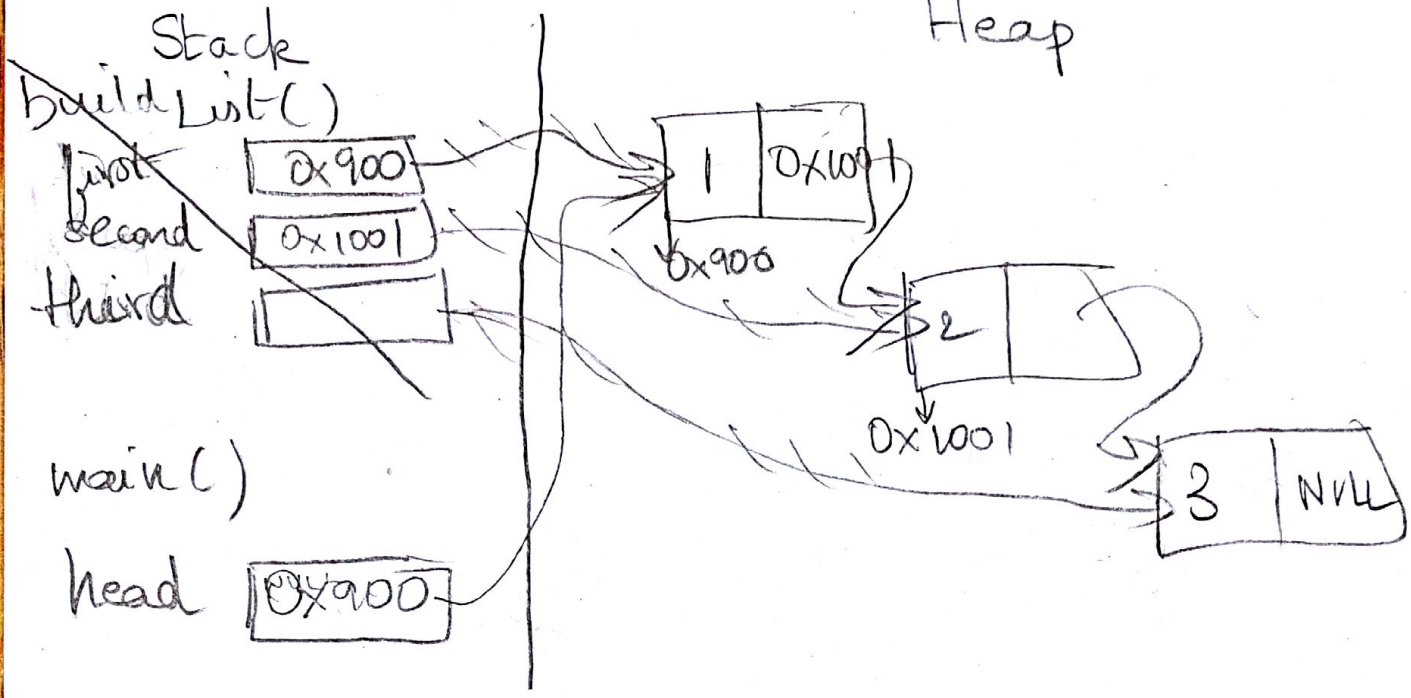
free (name) ; // free the allocated memory on the heap

Linked Lists




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

Example

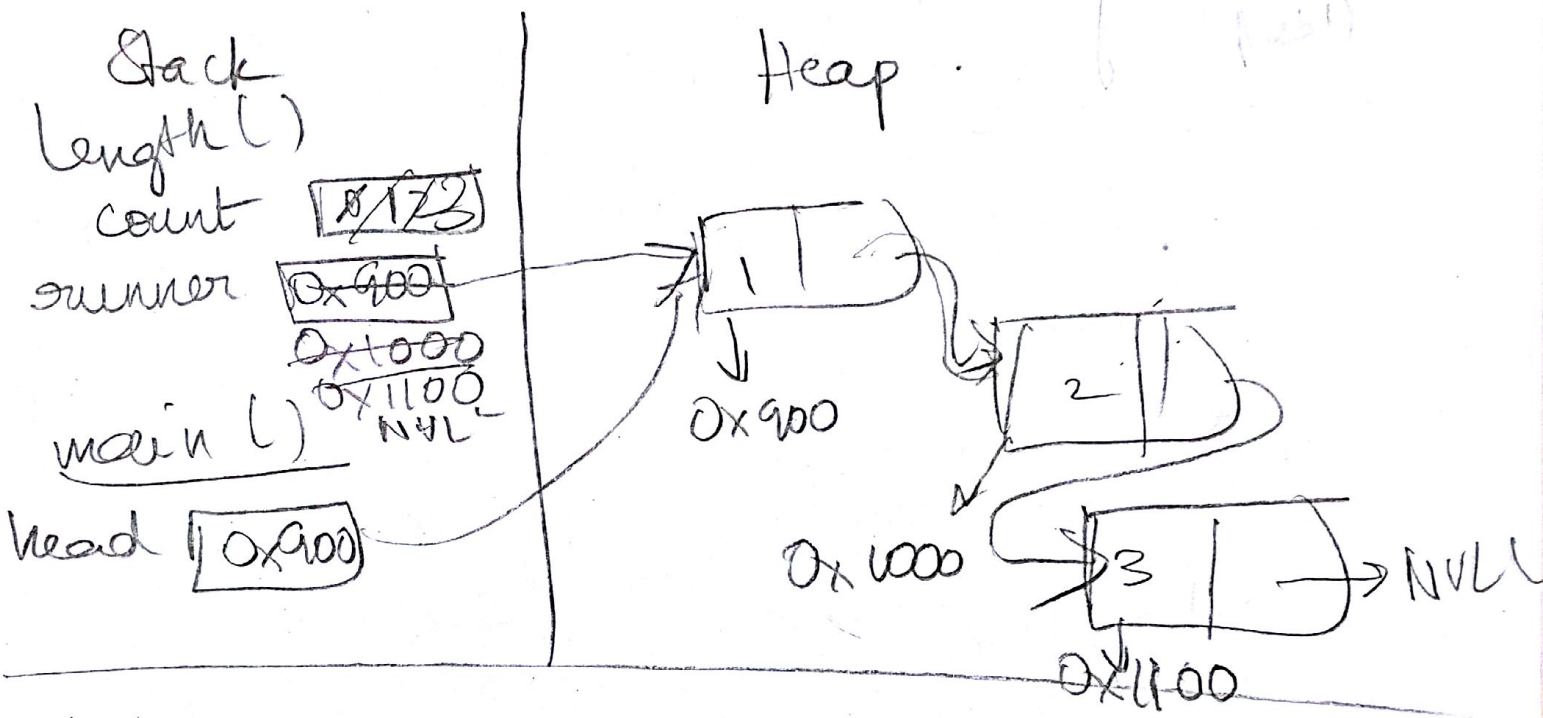



```
struct node * buildList()
{ struct node * first = NULL
   //      //   * second = NULL
   //      //   * third = NULL
first = (struct node *)
        malloc (sizeof
                (struct node))
second = //
third = //
first -> data = 1;
first -> next = second;
second -> data = 2
second -> next = third
third -> data = 3
third -> next = _____;
return first;
}

main ( ) {
  struct node * head = NULL
  head = buildList ( );
}
```

Length ( ).

Given the head pointer, return the number of nodes in the list.




```
int length (struct node * runner)
{
    int count = 0;
    while (runner != NULL)
    {
        count++;
        runner = runner -> next;
    }
    return count
}
```

```
main ( )
{
    head = buildList ( );
    printf ("%d", length (head));
}
```