How linked lists are arranged in memory?

Linked list basically consists of memory blocks that are located at random memory locations. Now, one would ask how are they connected or how they can be traversed? Well, they are connected through pointers. Usually, a block in a linked list is represented through a structure like this:

```c
struct node
{
    int val;
    struct node *next;
};
```

So as you can see here, this structure contains a value ‘val’ and a pointer to a structure of same type. The value ‘val’ can be any value (depending upon the data that the linked list is holding) while the pointer ‘next’ contains the address of next block of this linked list. So linked list traversal is made possible through these ‘next’ pointers that contain address of the next node. The ‘next’ pointer of the last node (or for a single node linked list) would contain a NULL.

Simple linked list example

```c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
struct node {
    int val;
    struct node *next;
};
void main()
{
    /* This will be the unchanging first node */
    struct node *root;
    /* Now root points to a node struct */
    root = (struct node *) malloc( sizeof(struct node) );
    /* The node root points to has its next pointer equal to a null pointer set */
    root->next = 0;
    /* By using the -> operator, you can modify what the node, a pointer, (root in this case) points to. */
    root->val = 5;
    getch();
}
```
Executable data object

In most languages, especially compiled ones like C, executable source programs and the data objects they manipulate are separate structures. But in some languages like LISP and Prolog, executable statements may be data that are accessible by the program, and that may be manipulated by it. For example, LISP stores all of its data in lists. The scheme expression:

\[
(\text{define myfunction (cons (a b c)(d e f))})
\]

simply defines the function name *myfunction* to be the cons operation defined earlier. This is stored as a linked list much like any another list structure.