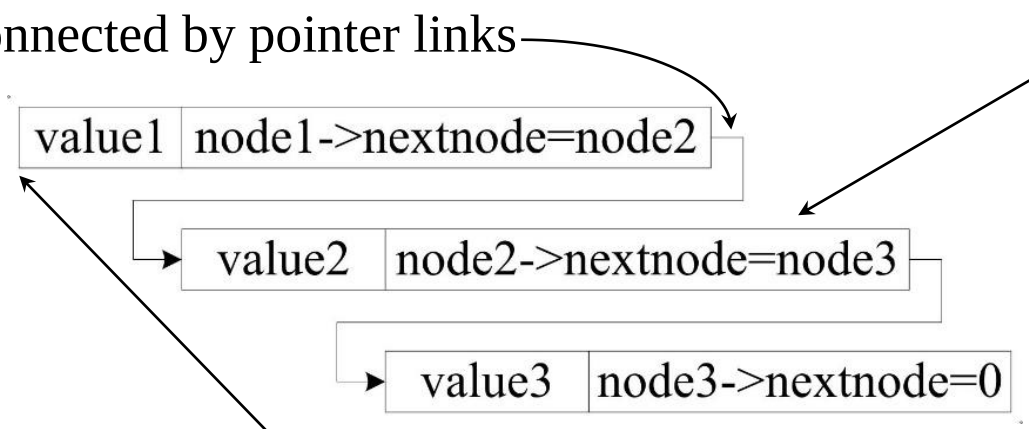
Linked-List Basic Examples

- A linked-list is
  - Linear collection of self-referential class objects, called nodes
  - Connected by pointer links



  - Accessed via a pointer to the first node of the list
  - Subsequent nodes are accessed via the link-pointer member of the current node
  - Link pointer in the last node is set to NULL to mark the list's end

```
struct node
{
    char data;
    struct node *nextnode;
}
```

- Use a linked list instead of an array when
  - You have an unpredictable number of data elements
  - Your list needs to be sorted quickly

- An array can be declared to contain more elements than number of data items expected,
  - but this can waste memory.
  - Linked lists can provide better memory utilization in these situations.

- Using dynamic memory allocation (instead of arrays) for data structures that grow and shrink at execution time can save memory

- Lets have a look at very simple example on the use of linked list

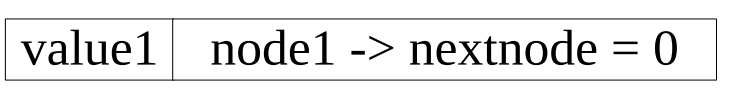
- The example involves simply creating first node of a linked list and printing it on screen

- The data structure (struct node) has 2 members, a character value (char c) and a pointer to the next node

- The first node is usually called rootnode and it can not be deleated. All nodes have some stored values and a link to the next node

- Final node has no pointer to the next node or a NULL pointer

// Linked_List_To_Enter_a_Char_and_Print_It.C
// The following program defines a data structure called node, with a character variable c
// and a link to the nextnode. Then declares a variable called rootnode of type struct node,
// assignes the value entered from keyboard to this node and prints values on screen

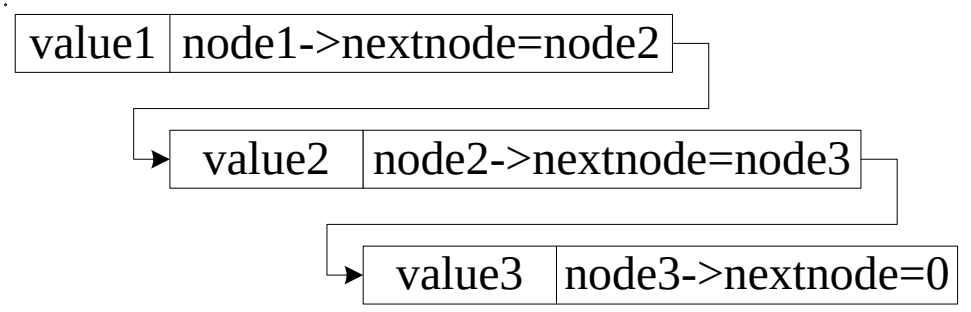



```
#include <stdio.h>
#include <stdlib.h>

struct node {
  char c;
  struct node *nextnode;
};

int main()
{
    struct node *rootnode;  // Declare a variable called roornodeto be of type struct node

    rootnode = malloc( sizeof(struct node) );
    rootnode->nextnode = 0;
```

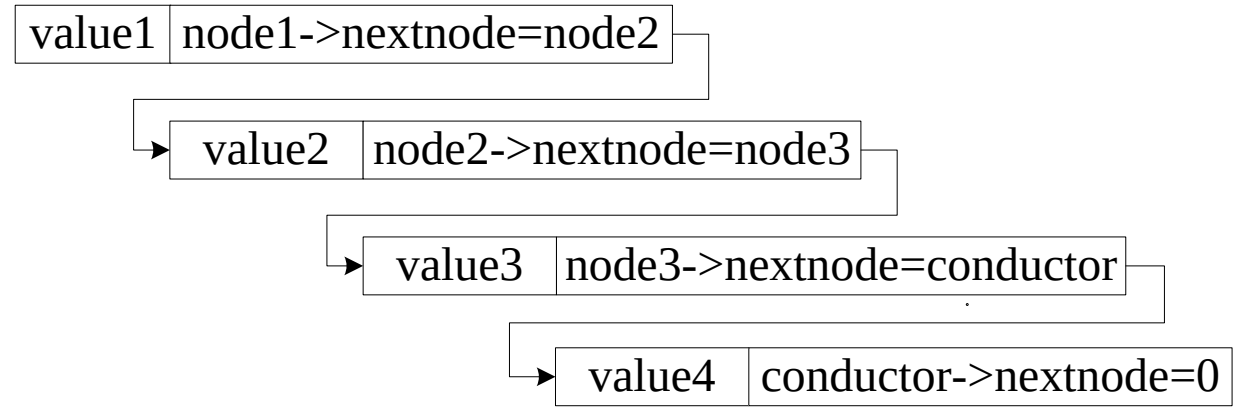
```
    printf("Please enter a value from the keyboard : ");
    scanf("%c", &rootnode->c);

       printf("%c => ", rootnode->c);
    return 0;
}
```

The following example creates a linked list and adds 3 nodes to the linked list and then adds a fourth node.

// Linked_List_To_Enter_3_Chars_And_Add_New_One.C
// The following program defines a data structure called node, with a character variable c
// and a link to the nextnode, then declares three variables node1, node2, node3 of type
// struct node, assignes values to these 3 nodes, prints them on the screen, then creates a
// new variable called conductor, which traces the list till end and adds a new node at the
// end of the list. Before adding the last node, the program should check if the new memory
// block is allocated successfully




```
#include <stdio.h>
#include <stdlib.h>

struct node
{
  char c;
  struct node *nextnode;
};

int main()
{
```



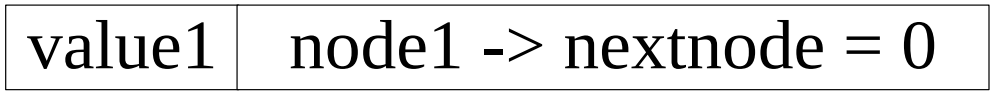
```
struct node *node1, *node2, *node3;  // node1 will be root node which can’t be deleted

struct node *conductor;  // point to each node and traverses list
node1 = malloc( sizeof(struct node) );
node1->nextnode = 0;
node1->c = 'x';

   node2 = malloc( sizeof(struct node) );
   node1->nextnode = node2;
   node2->nextnode = 0;
   node2->c = 'y';

         node3 = malloc( sizeof(struct node) );
         node2->nextnode = node3;
         node3->nextnode = 0;
         node3->c = 'z';

   printf("%c => ", node1->c);
         printf("%c => ", node2->c);
               printf("%c => ", node3->c);
```

```
// assign node1 address to conductor & advance conductor till end
// of linked-list
conductor = node1;
      if ( conductor != 0 )
            {
            while ( conductor->nextnode != 0)
            {
            conductor = conductor->nextnode;
            }
      }

      /* Creates a node at the end of the list */
      conductor->nextnode = malloc( sizeof(struct node) );
      conductor = conductor->nextnode;

      if ( conductor == 0 )
      {
            printf( "Out of memory" );
            return 0;
      }
```

```
    /* assign a value to the new memory and print it */
    conductor->nextnode = 0;
    conductor->c = 't';
       printf("%c => ", conductor->c);

    return 0;
}
```

// Linked_List_To_Enter_a_String_of_Characters_Using_Pointer.c
// Following program defines a data structure called node, with a character variable c and a
// link to the nextnode. Then declares a variable called rootnode of type struct node,
// assignes the value entered from the keyboard to this node and prints value on screen

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
  char *c;
  struct node *nextnode;
};

int main()
{
    /* Declare a variable called rootnode to be of type struct node */
       struct node *rootnode;
       char c[10];
```



```
        rootnode = malloc( sizeof(struct node) );
        rootnode->nextnode = 0;

        printf("Please enter a value from the keyboard : ");
        scanf("%s", rootnode->c);
        printf("%s => ", rootnode->c);
    return 0;
}
```

```
// Linked_List_To_Enter_3_Chars_and_Print_Using_Pointers.C
// The program defines a data structure called node, with a character variable c and a link to
// the nextnode. Then declares three variables node1, node2, node3 of type struct node,
// assignes values to these 3 nodes and prints them on the screen

#include <stdio.h>
#include <stdlib.h>

struct node {
  char a, b, c;
  struct node *next;
};

int main()
{
      struct node *node1, *node2, *node3;
      struct node *conductor;  // Points to nodes and traverses list
            node1 = malloc( sizeof(struct node) );
            node1->next = 0;
            node1->a = 'x';
```

```
        node2 = malloc( sizeof(struct node) );
        node1->next = node2;
        node2->next = 0;
        node2->b = 'y';

        node3 = malloc( sizeof(struct node) );
        node2->next = node3;
        node3->next = 0;
        node3->c = 'z';

        printf("%c => ", node1->a);
              printf("%c => ", node2->b);
                    printf("%c => ", node3->c);

    return 0;
}
```