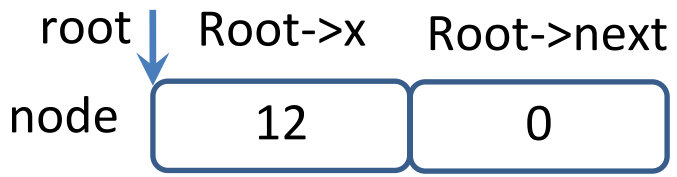
```c
/* Linked_List_Example 1: Just shows how linked lists are created. It does
    nothing */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int x;
    struct node *next;
};

int main()
{
    /*  This won't change, or we would lose the list in memory */
    struct node *root;

    /*  This will point to each node as it traverses the list */
    struct node *conductor;

    /*  malloc allocates memory enough to store the size of struct node */
    root = malloc( sizeof(struct node) );
    root->next = 0;
    root->x = 12;
```



/* Two operators are used to access members of structures: the **structure member operatör (.)**—also called the dot operator—and the **structure pointer operator (->)**—also called the **arrow operator**. The structure member operator accesses a structure member via the structure variable name. For example, to print member suit of structure variable aCard defined in Section 10.3, use the statement

```
printf( "%s", aCard.suit ); /* displays Hearts */
```

The structure pointer operator (consisting of a minus (-) sign and a greater than (>) sign with no intervening spaces) accesses a structure member via a **pointer to the structure**. Assume that the pointer cardPtr has been declared to point to struct card and that the address of structure aCard has been assigned to cardPtr. To print member suit of structure aCard with pointer cardPtr, use the statement

```
printf( "%s", cardPtr->suit ); /* displays Hearts */
```

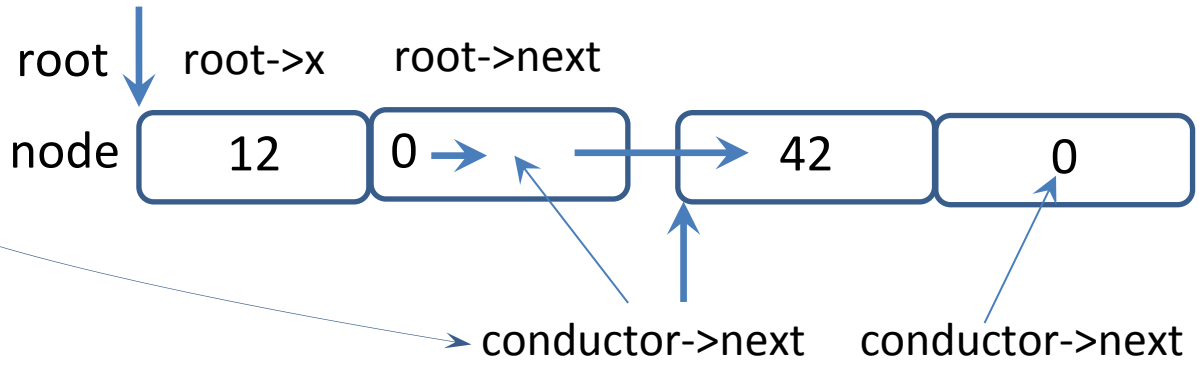
The expression cardPtr->suit is equivalent to (*cardPtr).suit, which dereferences the pointer and accesses the member suit using the structure member operator. The parentheses are needed here because the structure member operator (.) has a higher precedence than the pointer dereferencing operator (*). The structure pointer operator and structure member operator, along with parentheses (for calling functions) and brackets ([]) used for array subscripting, have the highest operator precedence and associate from left to right.
*/

```
    conductor = root;


    if ( conductor != 0 )
    {
        while ( conductor->next != 0)
        {
            conductor = conductor->next;
        }
    }

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

    if ( conductor == 0 )
    {
        printf( "Out of memory" );
        return 0;
    }
    /* initialize the new memory */
    conductor->next = 0;
    conductor->x = 42;
    return 0;
}
```



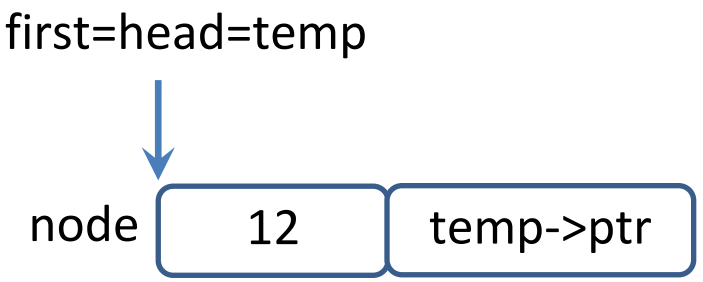
```
/*Linked_List_Example_4.C
   C Program to Create a Linked List & Display the Elements in the List. Linked
   list is an ordered set of data elements, each containing a link to its
   successor. This program creates a linked list and displays all the elements
   present in the created list. */

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

void main()
{
struct node
{
    int num;
    struct node *ptr;
}; typedef struct node NODE;

NODE *head, *first, *temp = 0;

int count = 0;
int choice = 1;
first = 0;
```

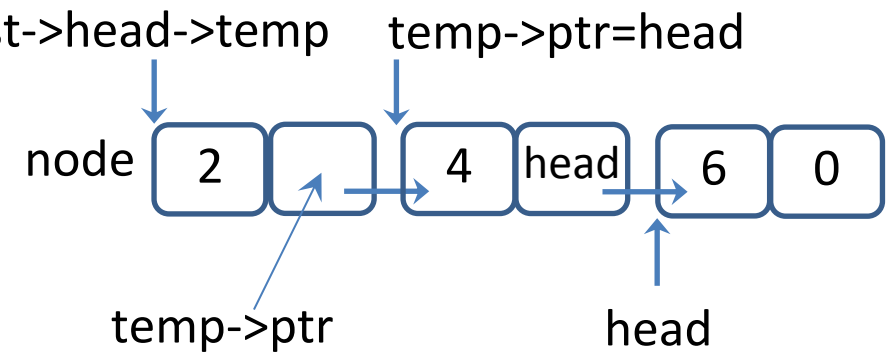


```
while (choice)
{
    head  = (NODE *)malloc(sizeof(NODE));
    printf("Enter the data item\n");
    scanf("%d", &head-> num);

    if (first != 0)
    {
        temp->ptr = head;
        temp = head;
    }
    else
    {
        first = temp = head; /* first points to first node of the linked-list */
    }                        /* head points to beginning address */
    fflush(stdin);           /* of the current node*/

    printf("Do you want to continue (Type 0 or 1)?\n");
    scanf("%d", &choice);
}

temp->ptr = 0;
```



```
    /*  reset temp to the beginning */
    temp = first;

    printf("\n status of the linked list is\n");

    while (temp != 0)
    {
        printf("%d=>", temp->num);
        count++;
        temp = temp -> ptr;
    }

    printf("NULL\n");
    printf("No. of nodes in the list = %d\n", count);
}
```

OUTPUT

2 => 4 => 6 =>

```c
/*  Linked_List_8.c
    Example to define the root node and add 10 notes to the list
*/
#include <stdio.h>
#include <stdlib.h>

/* Define a structure called node with two members, named item and *next*/

struct node
{
    int item;
    struct node *next;
};

int main()
{
    int x;

    /*  Defibe a new variable named *root and data type struct node
        node will also have 2 members of type int and struct node. This
        will be the unchanging first node */

   struct node *root;
```

```
struct node *conductor;

/*  This will be like,
    struct conductor
    {
        int item;
        struct conductor *next;
    }
This will point to each node as it traverses the list */

/* allocating space for the root */

root = malloc(sizeof(struct node));

/*  Functions malloc and free, and operator sizeof, are essential to
    dynamic memory allocation. Function malloc takes as an argument the
    number of bytes to be allocated and returns a pointer of type void *
    (pointer to void) to the allocated memory.
```

A void * pointer may be assigned to a variable of any pointer type. Function malloc is normally used with the sizeof operator. For example, the statement

```
newPtr = malloc( sizeof( struct node ) );
```

evaluates sizeof(struct node) to determine the size in bytes of a structure of type struct node, allocates a new area in memory of that number of bytes and stores a pointer to the allocated memory in variable newPtr. The allocated memory is not initialized. If no memory is available, malloc returns NULL.

Function free deallocates memory, i.e., the memory is returned to the system so that the memory can be reallocated in the future. To free memory dynamically allocated by the preceding malloc call, use the statement

```
free( newPtr );
```

*/

```
root->next = 0;

/*  Points to the allocated memory where the - struct node - is stored.
```

```
    In the struct node, the value 777 is assigned to the data (int item).

    The following line also means (*root).item = 777;
*/
root->item = 777;

conductor = root;

/* create ten new nodes */
for(x = 1; x <=10; ++x)
{
    /* Creates a node at the end of the list */

    conductor->next = malloc( sizeof(struct node) );

    /*checking that memory is allocated*/

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

```
        conductor = conductor->next;
        conductor->item = x;
    }

    //marking the last as NULL

    conductor->next = 0;

    //conductor points to root again
    conductor = root;

    //print the list
    if ( conductor != 0 )  /* Makes sure there is a place to start */
    {
        while(conductor->next != 0)
        {
            printf(" [%d]-> ",conductor->item);
            conductor = conductor->next;
        }
    }
getchar();
return 0;
}
```

```c
/*  Linked_List_Example_11.c
 *  C program to search for an element in linked list
*/
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int a;
    struct node *next;
};

void generate(struct node **, int);
void search(struct node *, int, int);
void delete(struct node **);

int main()
{
    struct node *head;
    int key, num;

    printf("Enter the number of nodes: ");
    scanf("%d", &num);
```

```
    generate(&head, num);
    printf("\nEnter key to search: ");
    scanf("%d", &key);
    search(head, key, num);
    delete(&head);

return 0;
}

void generate(struct node **head, int num)
{
    int i;
    struct node *temp;

    for (i = 0; i < num; i++)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        temp->a = rand() % num;
        printf("%d    ", temp->a);
        if (*head == NULL)
        {
            *head = temp;
            (*head)->next = NULL;
```

```
        }
        else
        {
            temp->next = *head;
            *head = temp;
        }
    }
}

void search(struct node *head, int key, int index)
{
    if (head->a == key)
    {
        printf("Key found at Position: %d\n", index);
    }

    if (head->next == NULL)
    {
        return;
    }

    search(head->next, key, index - 1);
}
```

```
void delete(struct node **head)
{
    struct node *temp;
    while (*head != NULL)
    {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}
```

```c
/*  Linked_List_Example_12.c
 * Recursive C program to display members of a linked list */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int a;
    struct node *next;
};

void generate(struct node **);
void display(struct node*);
void delete(struct node **);

int main()
{
    struct node *head = NULL;
    generate(&head);
    display(head);
    delete(&head);
    return 0;
}
```

```
void generate(struct node **head)
{
   int num = 10, i;
   struct node *temp;

   for (i = 0; i < num; i++)
   {
      temp = (struct node *)malloc(sizeof(struct node));
      temp->a = i;

      if (*head == NULL)
      {
         *head = temp;
         (*head)->next = NULL;
      }
      else
      {
         temp->next = *head;
         *head = temp;
      }
   }
}
```

```
void display(struct node *head)
{
   printf("%d    ", head->a);
   if (head->next == NULL)
   {
      return;
   }
   display(head->next);
}

void delete(struct node **head)
{
   struct node *temp;
   while (*head != NULL)
   {
      temp = *head;
      *head = (*head)->next;
      free(temp);
   }
}
```

```c
/*  Linked_List_example_13.c
 *
 * C Program find the Length of the Linked List without using Recursion
 */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int a;
    struct node *next;
};


void generate(struct node **);
int length(struct node*);
void delete(struct node **);

int main()
{
    struct node *head = NULL;
    int count;
```

```
    generate(&head);
    count = length(head);
    printf("The number of nodes are: %d\n", count);
    delete(&head);

    return 0;
}

void generate(struct node **head)
{
    /* for unknown number of nodes use num = rand() % 20; */
    int num = 10, i;
    struct node *temp;

    for (i = 0; i < num; i++)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        temp->a = i;
        if (*head == NULL)
        {
            *head = temp;
            (*head)->next = NULL;
        }
```

```
        else
        {
            temp->next = *head;
            *head = temp;
        }
    }
}

int length(struct node *head)
{
    int num = 0;
    while (head != NULL)
    {
        num += 1;
        head = head->next;
    }
    return num;
}

void delete(struct node **head)
{
    struct node *temp;
```

```
    while (*head != NULL)
    {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}
```