Manav Rachna International Institute of Research and Studies Bachelor's in computer applications

Data Structures using C

Submitted by: Urvashi Pahuja

Department: School Of Computer Applications

Course: Bcahelors in Computers 1pplications

Roll no. : 24/SCA/BCA/087

Semester: 2nd

Subject: Data Structures Using C

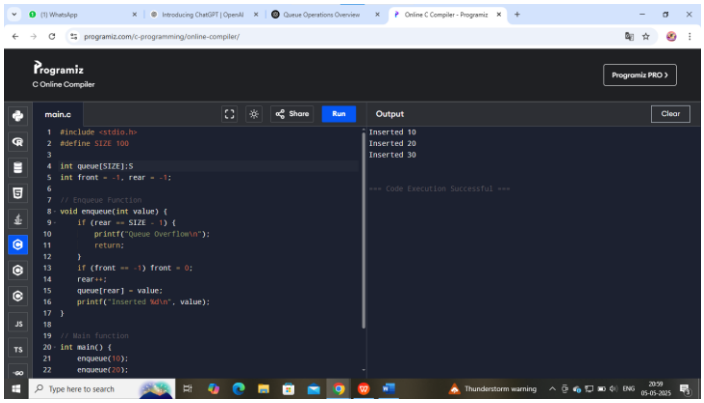
DS FILE

AIM1: ADDING AN ELEMENT AT THE BEGINNING OF THE LINKED LIST.

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

// Define node structure
struct node {
    int data;
    struct node *next;
};

// Global pointers
struct node *head = NULL;

// Function to insert at the beginning
void insertatbegin(int data) {
    struct node *lk = (struct node*) malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
```

```
    lk->next = head;
    head = lk;
}

// Function to print the list
void printList() {
    struct node *p = head;
    printf("\n[");
    while (p != NULL) {
        printf(" %d ", p->data);
        p = p->next;
    }
    printf("]\n");
}

int main() {
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(44);
    insertatbegin(50);

    printf("Linked List: ");
    printList();

    return 0;
}
```

OUTPUT:



<u>AIM 2:</u> ADDING AN ELEMENT AT THE ENDING OF THE LINKED LIST.

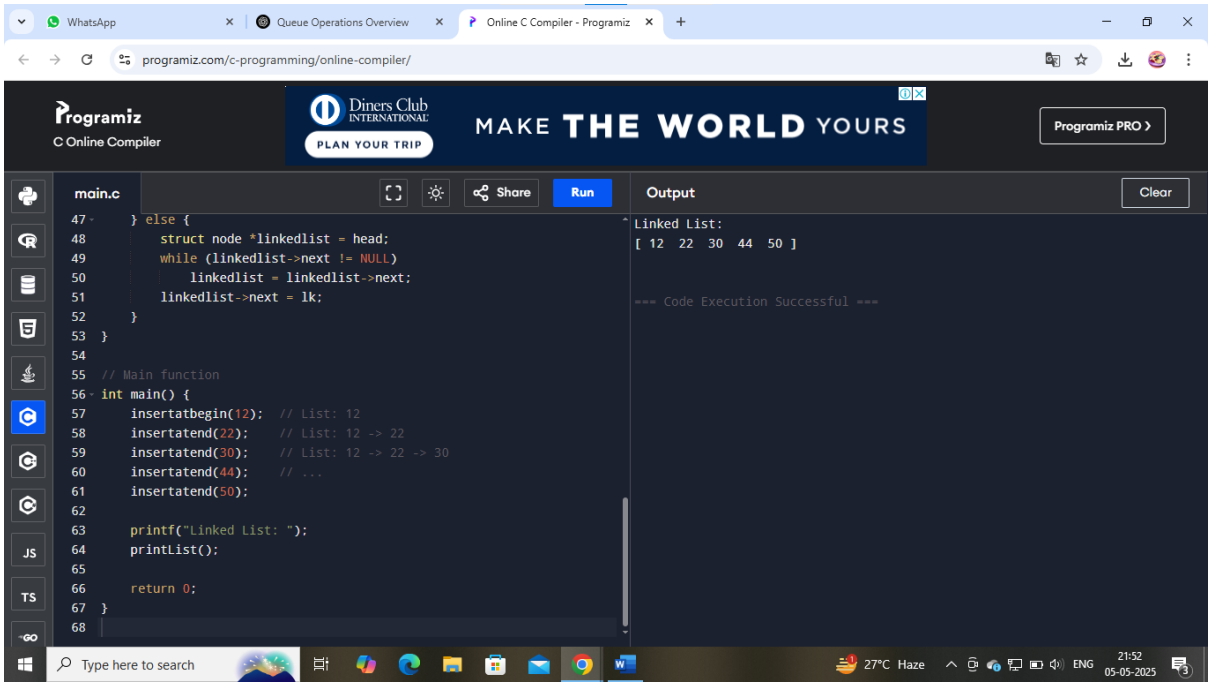
```
#include <stdio.h>
#include <stdlib.h>

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

struct node *head = NULL;
```

```
// Display the list
void printList() {
    struct node *p = head;
    printf("\n[");
    while (p != NULL) {
        printf(" %d ", p->data);
        p = p->next;
    }
    printf("]\n");
}

// Insert at the beginning
void insertatbegin(int data) {
    struct node *lk = (struct node*) malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = head;
    head = lk;
}
```

```
// Insert at the end
void insertatend(int data) {
    struct node *lk = (struct node*) malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = NULL;

    if (head == NULL) {
        // If the list is empty
        head = lk;
    } else {
        struct node *linkedlist = head;
        while (linkedlist->next != NULL)
            linkedlist = linkedlist->next;
        linkedlist->next = lk;
    }
}

// Main function
int main() {
    insertatbegin(12);  // List: 12
```

```
    insertatend(22);    // List: 12 -> 22
    insertatend(30);    // List: 12 -> 22 -> 30
    insertatend(44);    // ...
    insertatend(50);

    printf("Linked List: ");
    printList();

    return 0;
}
```

OUTPUT:

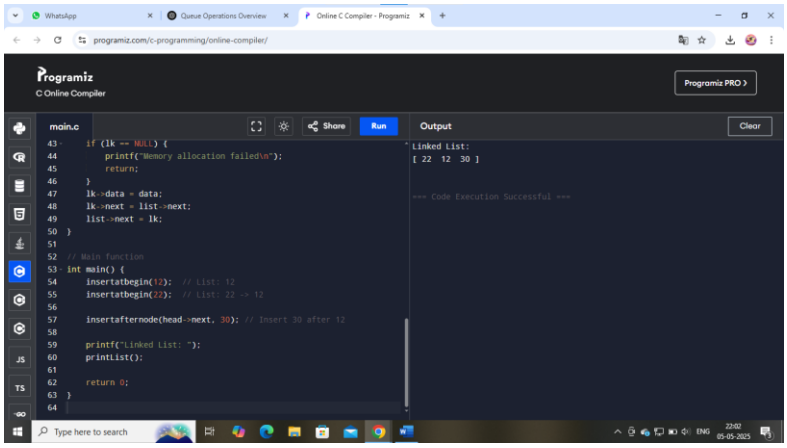


<u>AIM 3:</u> ADDING AN ELEMENT AT ANY POSITION WITHIN THE LIST.

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

// Node structure
struct node {
   int data;
   struct node *next;
};

// Global head pointer
struct node *head = NULL;

// Display the list
void printList() {
   struct node *p = head;
   printf("\n[");
   while (p != NULL) {
      printf(" %d ", p->data);
      p = p->next;
```

```
    }
    printf("]\n");
}

// Insert at beginning
void insertatbegin(int data) {
    struct node *lk = (struct node *)malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = head;
    head = lk;
}

// Insert after a given node
void insertafternode(struct node *list, int data) {
    if (list == NULL) {
        printf("The given previous node is NULL.\n");
        return;
    }
```

```
    struct node *lk = (struct node *)malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = list->next;
    list->next = lk;
}

// Main function
int main() {
    insertatbegin(12);  // List: 12
    insertatbegin(22);  // List: 22 -> 12

    insertafternode(head->next, 30); // Insert 30 after 12

    printf("Linked List: ");
    printList();

    return 0;
}
```

OUTPUT: 

<u>AIM 4:</u> DELETION AT THE BEGINNING OF THE LINKED LIST.

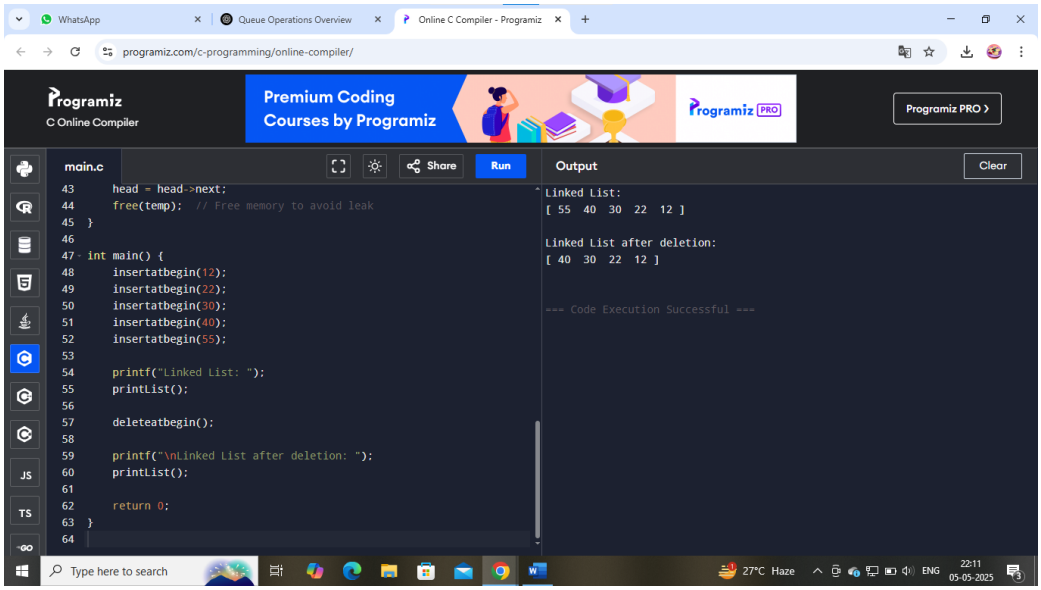
```
#include <stdio.h>

#include <stdlib.h>


// Node structure

struct node {

    int data;

    struct node *next;

};


// Global head pointer

struct node *head = NULL;


// Display the list
```

```
void printList() {
    struct node *p = head;
    printf("\n[");
    while (p != NULL) {
        printf(" %d ", p->data);
        p = p->next;
    }
    printf("]\n");
}

// Insertion at the beginning
void insertatbegin(int data) {
    struct node *lk = (struct node *)malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = head;
    head = lk;
}
```

```
// Delete at beginning
void deleteatbegin() {
    if (head == NULL) {
        printf("List is already empty.\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    free(temp);  // Free memory to avoid leak
}

int main() {
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(40);
    insertatbegin(55);

    printf("Linked List: ");
    printList();

    deleteatbegin();
```

```
    printf("\nLinked List after deletion: ");
    printList();

    return 0;
}
```

OUTPUT:



AIM 5: DELETING AN ELEMENT AT THE ENDING OF THE LINKED LIST.

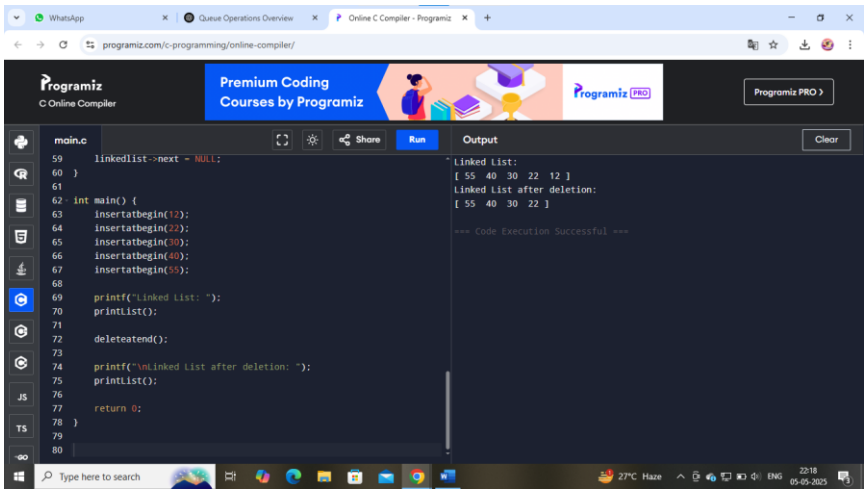
```
#include <stdio.h>
#include <stdlib.h>
```

```
// Node structure
struct node {
    int data;
    struct node *next;
};

// Global head pointer
struct node *head = NULL;

// Display the list
void printList() {
    struct node *p = head;
    printf("\n[");
    while (p != NULL) {
        printf(" %d ", p->data);
        p = p->next;
    }
    printf("]");
}

// Insert at the beginning
void insertatbegin(int data) {
    struct node *lk = (struct node *)malloc(sizeof(struct
node));
```

```
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = head;
    head = lk;
}

// Delete the last node
void deleteatend() {
    if (head == NULL) {
        printf("\nList is empty.");
        return;
    }

    // Only one node
    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }

    struct node *linkedlist = head;
```

```
    // Traverse to second last node
    while (linkedlist->next->next != NULL) {
        linkedlist = linkedlist->next;
    }

    // Delete last node
    free(linkedlist->next);
    linkedlist->next = NULL;
}

int main() {
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(40);
    insertatbegin(55);

    printf("Linked List: ");
    printList();

    deleteatend();

    printf("\nLinked List after deletion: ");
    printList();
```

```
    return 0;
}
```

OUTPUT: 

<u>AIM 6:</u> DELETING AN ELEMENT AT ANY POSITION WITHIN THE LIST.

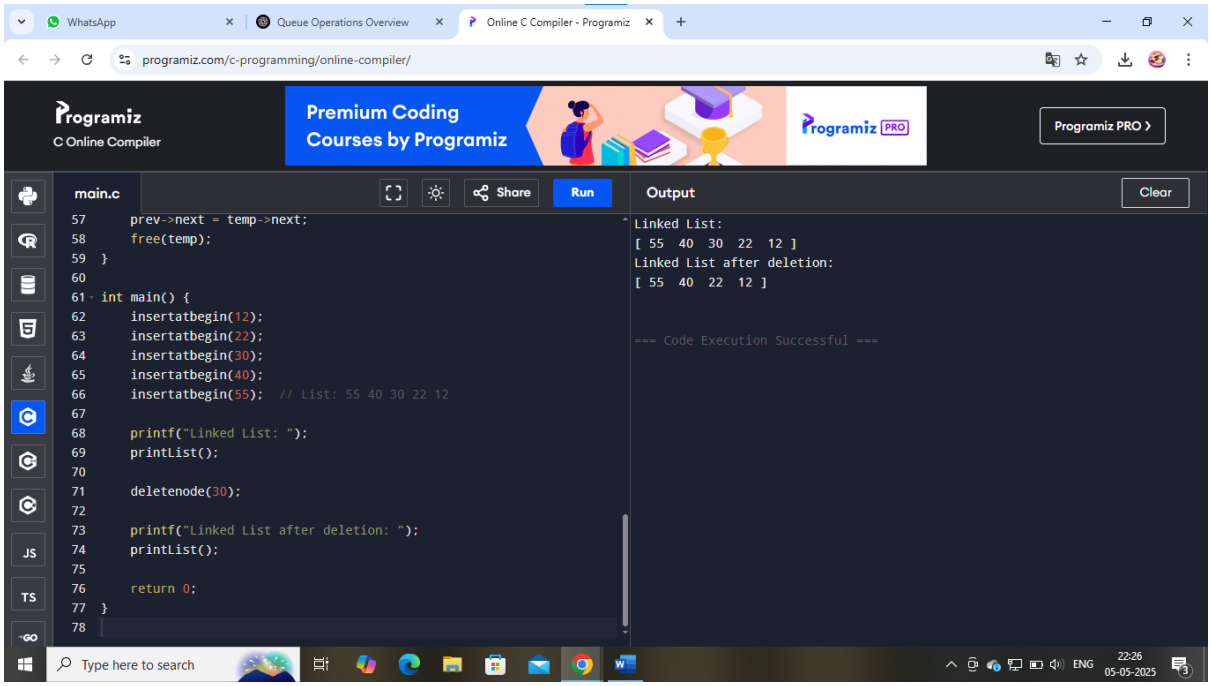
```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct node {
    int data;
    struct node *next;
};
```

```
// Head pointer
struct node *head = NULL;

// Function to print the list
void printList() {
    struct node *p = head;
    printf("\n[");
    while (p != NULL) {
        printf(" %d ", p->data);
        p = p->next;
    }
    printf("]\n");
}

// Insert at beginning
void insertatbegin(int data) {
    struct node *lk = (struct node *)malloc(sizeof(struct node));
    if (lk == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    lk->data = data;
    lk->next = head;
    head = lk;
```

```
}

// Delete a node by key
void deletenode(int key) {
    struct node *temp = head, *prev = NULL;

    // If head node holds the key
    if (temp != NULL && temp->data == key) {
        head = temp->next;
        free(temp);
        return;
    }

    // Search for the key
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // If key not found
    if (temp == NULL) return;

    // Unlink and delete the node
    prev->next = temp->next;
```

```
    free(temp);
}

int main() {
    insertatbegin(12);
    insertatbegin(22);
    insertatbegin(30);
    insertatbegin(40);
    insertatbegin(55);  // List: 55 40 30 22 12

    printf("Linked List: ");
    printList();

    deletenode(30);

    printf("Linked List after deletion: ");
    printList();

    return 0;
}
```

OUTPUT:


WhatsApp
Queue Operations Overview
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Premium Coding
Courses by Programiz
Programiz PRO
Programiz PRO ›
main.c
Share
Run
Output
Clear
prev->next = temp->next;
free(temp);
}
int main() {
insertatbegin(12);
insertatbegin(22);
insertatbegin(30);
insertatbegin(40);
insertatbegin(55); // List: 55 40 30 22 12
printf("Linked List: ");
printList();
deletenode(30);
printf("Linked List after deletion: ");
printList();
return 0;
}
Linked List:
[55 40 30 22 12]
Linked List after deletion:
[55 40 22 12]
=== Code Execution Successful ===
Type here to search
ENG
22:26
05-05-2025