Manav Rachna International Institute of Research and Studies

Bachelor's in computer applications

Data Structures using C

Name: Yashika Saini

Roll No: 24/SCA/BCA/055

Department: School of Computer Applications

Course: Bachelor of Computer Applications

Semester: 2nd

Subject: Data Structures using C

Program 1

AIM: Write a program in C to implement Insertion in 1-D Arrays.

CODE:

```
#include <stdio.h>
int main() {
  int arr[5];
  int n, count=0, loc, upd;
  printf("Enter the elements of array\n");
  for(int i=0; i<5; i++){
     scanf("%d",&arr[i]);
  }
   return 0;
}
```

OUTPUT:

```
Output
Enter the elements of array
3
7
8
9
2


=== Code Execution Successful ===
```

PROGRAM 2

AIM: Write a program in C to implement deletion in 1-D arrays.

CODE:

```
#include <stdio.h>
int main(){
   int count = 0;
   int x;
   int arr1[] = {1,2,3,4,5};
   printf("Enter the element you want to delete: \n");
   scanf("%d", &x);
   for(int i = 0; i<5; i++){
      if(arr1[i] == x){
         for(int j = i; j<5; j++){
            arr1[j] = arr1[j + 1];
            }
         count = count + 1;
      }
 }
   if(count == 0){
      printf("Element is not found");
   } else{
      for(int i = 0; i<4; i++){
       printf("%d\t", arr1[i]);
      }
       return 0;
  }
}
```

OUTPUT:

```
Output
Enter the element you want to delete:
3
1   2   4   5

=== Code Execution Successful ===
```

PROGRAM 3

AIM: Write a program in C to implement searching in 1-D arrays.

CODE:

```
#include <stdio.h>
int main() {
   int arr[5];
   int n,count=0 , loc, upd;
   printf(" enter the elements of array \n");
   for(int i=0; i<5; i++){
      scanf("%d",&arr[i]);
   }
   printf("enter the element you want to find\n");
   scanf("%d",&n);

   for(int i=0; i<5; i++){
      if(arr[i] == n){
         printf("%d found at location %d\n",n,i+1);
         count +=1;
      }
   }
   if(count == 0){
```

```
    printf("%d not founded\n", n);
    }
return 0;
}
```

OUTPUT:

```
Output

 enter the elements of array
5
8
2
4
6
enter the element you want to find
2
2 found at location 3


=== Code Execution Successful ===
```

PROGRAM 4

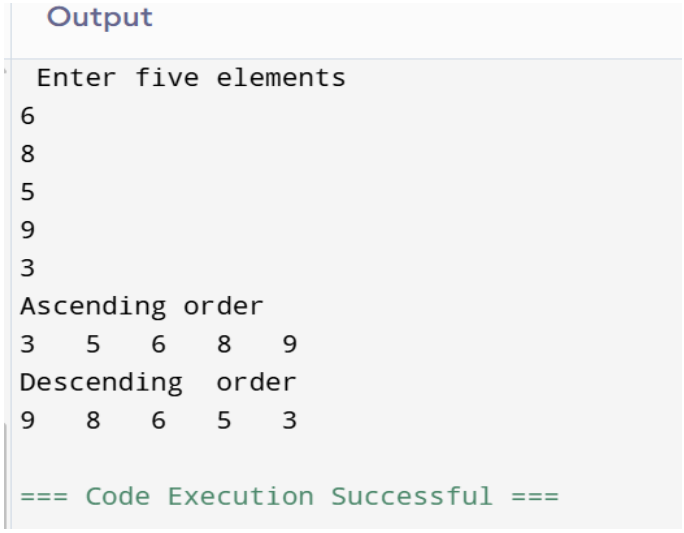
AIM: Write a program in C to implement sorting in 1-D arrays.

CODE:

```
#include <stdio.h>
int main() {
    int arr[5];
    printf(" Enter five elements \n");
    for(int i=0; i<5; i++){
        scanf("%d",&arr[i]);
    }
    int temp;
    for(int i=0; i<5; i++){
```

```
    for(int j=0; j<4-i; j++){
     if(arr[j] > arr[j+1]){
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
     }
   }
}
 printf("Ascending order\n");
 for(int i=0; i<5; i++){
     printf("%d\t",arr[i]);
 }
    for(int i=0; i<5; i++){
    for(int j=0; j<4-i; j++){
     if(arr[j] < arr[j+1]){
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
     }
   }
}
 printf("\nDescending  order\n");
 for(int i=0; i<5; i++){
     printf("%d\t",arr[i]);
 }
 return 0;
   }
```

OUTPUT:



PROGRAM 5

AIM: Write a program in C to implement push operation in stacks.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

struct Stack {
   int arr[MAX];
   int top;
};
void initStack(struct Stack* stack) {
   stack->top = -1;
}

int isFull(struct Stack* stack) {
   return stack->top == MAX - 1;
}
```

```
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        stack->arr[++(stack->top)] = value;
        printf("%d pushed to stack\n", value);
    }
}

void printStack(struct Stack* stack) {
    if (stack->top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= stack->top; i++) {
            printf("%d ", stack->arr[i]);
        }printf("\n");
    }
}
int main() {
    struct Stack stack;
    initStack(&stack);
     push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);
    push(&stack, 60);
    printStack(&stack);
    return 0;
}
```

OUTPUT:

```
Output

10 pushed to stack
20 pushed to stack
30 pushed to stack
40 pushed to stack
50 pushed to stack
Stack Overflow! Cannot push 60
Stack elements: 10 20 30 40 50


=== Code Execution Successful ===
```

PROGRAM 6

AIM: Write a program in c to implement pop operation in stacks.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Stack {
    int arr[MAX_SIZE];
    int top;
};

void initialize(struct Stack *stack) {
    stack->top = -1;
}
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}
int pop(struct Stack *stack) {
```

```
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop element from empty stack.\n");
        return -1;
    }
    return stack->arr[stack->top--];
}
int main() {
    struct Stack stack;
    initialize(&stack);

    stack.arr[++stack.top] = 10;
    stack.arr[++stack.top] = 20;
    stack.arr[++stack.top] = 30;

    printf("Popped: %d\n", pop(&stack));
    printf("Popped: %d\n", pop(&stack));
    printf("Popped: %d\n", pop(&stack));
    printf("Popped: %d\n", pop(&stack));
    return 0;
}
```

OUTPUT:

Output

```
Popped: 30
Popped: 20
Popped: 10
Stack Underflow! Cannot pop element from empty stack.
Popped: -1


=== Code Execution Successful ===
```

PROGRAM 7

AIM: Write a program in c to implement insertion in linked list (beg, mid, end)

a.) Insertion in beginning

CODE:

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

struct Node {
   int data;
   struct Node* next;
};
void insertAtBeginning(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   new_node->data = new_data;
   new_node->next = (*head_ref);
   (*head_ref) = new_node;
}
void printList(struct Node* node) {
   while (node != NULL) {
      printf("%d ", node->data);
      node = node->next;
   }
   printf("\n");
}
int main() {
   struct Node* head = NULL;
   insertAtBeginning(&head, 70);
   insertAtBeginning(&head, 60);
   insertAtBeginning(&head, 50);
```

```
    insertAtBeginning(&head, 40);
    insertAtBeginning(&head, 30);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 10);

    printf("Linked list after insertions: ");
    printList(head);
    return 0;
}
```

OUTPUT:

```
Output

Linked list after insertions: 10 20 30 40 50 60 70


=== Code Execution Successful ===
```

b.) Insertion at middle

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void insertAtMiddle(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    if (new_node == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    new_node->data = new_data;
    new_node->next = NULL;
```

```
    if (*head_ref == NULL) { // If the list is empty
        *head_ref = new_node;
        return;
    }

    struct Node* slow_ptr = *head_ref;
    struct Node* fast_ptr = *head_ref;
    struct Node* prev_slow = NULL;

    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next->next;
        prev_slow = slow_ptr;
        slow_ptr = slow_ptr->next;
    }

    new_node->next = slow_ptr;
    if (prev_slow != NULL) {
        prev_slow->next = new_node;
    } else {
        *head_ref = new_node;
    }
}
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");

void freeList(struct Node* head) {
```

```
    struct Node* tmp;
    while (head != NULL) {
        tmp = head;
        head = head->next;
        free(tmp);
    }
int main() {
    struct Node* head = NULL;
    insertAtMiddle(&head, 1);
    insertAtMiddle(&head, 2);
    insertAtMiddle(&head, 4);
    insertAtMiddle(&head, 5);
    printf("Linked list before insertion: \n");
    printList(head);

    insertAtMiddle(&head, 3);
    printf("Linked list after insertion: \n");
    printList(head);

    freeList(head); // Free allocated memory
    return 0;
}
```

OUTPUT:

Output

```
Linked list before insertion:
2 5 4 1
Linked list after insertion:
2 5 3 4 1


=== Code Execution Successful ===
```

c.) Insertion at ending

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
void insertAtEnd(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   struct Node *last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;

    if (*head_ref == NULL) {
      *head_ref = new_node;
      return;
   }
   while (last->next != NULL)
      last = last->next;
   last->next = new_node;
   return;
}
void printList(struct Node* node) {
   while (node != NULL) {
      printf("%d ", node->data);
      node = node->next;
   }
```

```
    printf("\n");
}
int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);

  printf("Created Linked list is: ");
    printList(head);

    return 0;
}
```

OUTPUT:

```
Output
Created Linked list is: 1 2 3 4


=== Code Execution Successful ===
```

PROGRAM 8

AIM: Write a program in c to implement deletion in linked list (beg, mid, end)

a.) Deletion at beginning

CODE:

```
    #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
```

```
};
struct Node* deleteFromBeginning(struct Node* head) {
    if (head == NULL) {
        printf("List is empty, cannot delete.\n");
        return NULL;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    return head;
}
void printList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

int main() {
    struct Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
```

```
    printf("Linked list before deletion: ");
    printList(head);
    head = deleteFromBeginning(head);

    printf("Linked list after deletion: ");
    printList(head);
    head = deleteFromBeginning(head);

    printf("Linked list after second deletion: ");
    printList(head);
    head = deleteFromBeginning(head);

    return 0;
}
```

OUTPUT:

```
Output

Linked list before deletion: 10 20 30
Linked list after deletion: 20 30
Linked list after second deletion: 30


=== Code Execution Successful ===
```

b.) Deletion at middle

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
```

```
};

void insertEnd(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL) {
      *head_ref = new_node;
      return;
   }
   while (last->next != NULL)
      last = last->next;
   last->next = new_node;
   return;
}
void deleteMiddle(struct Node** head_ref) {
   if (*head_ref == NULL || (*head_ref)->next == NULL) {
      free(*head_ref);
      *head_ref = NULL;
      return;
   }
   struct Node* slow_ptr = *head_ref;
   struct Node* fast_ptr = *head_ref;
   struct Node* prev = NULL;

   while (fast_ptr != NULL && fast_ptr->next != NULL) {
      fast_ptr = fast_ptr->next->next;
      prev = slow_ptr;
      slow_ptr = slow_ptr->next;
   }
```

```
    prev->next = slow_ptr->next;
    free(slow_ptr);
}
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    insertEnd(&head, 1);
    insertEnd(&head, 2);
    insertEnd(&head, 3);
    insertEnd(&head, 4);
    insertEnd(&head, 5);

    printf("Linked list before deletion: ");
    printList(head);

    deleteMiddle(&head);

    printf("Linked list after deletion of middle element: ");
    printList(head);

    return 0;
}
```

OUTPUT:

```
Output

Linked list before deletion: 1 2 3 4 5
Linked list after deletion of middle element: 1 2 4 5


=== Code Execution Successful ===
```

c.) Deletion at ending

CODE:

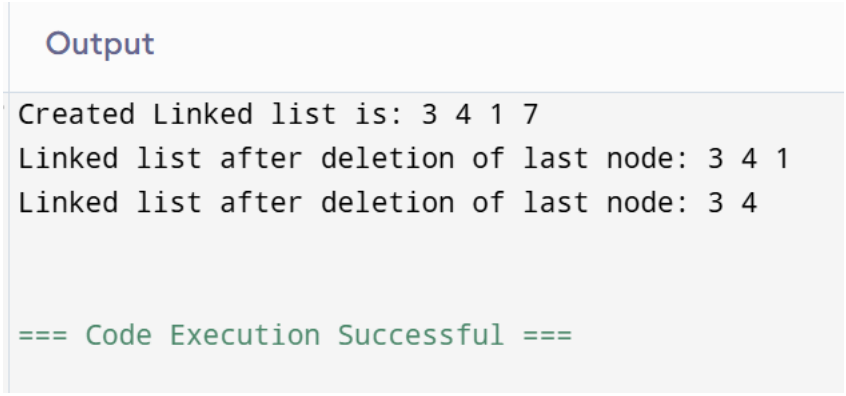
```
 #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void deleteLastNode(struct Node** head_ref) {
    if (*head_ref == NULL) {
        return;
    }
    if ((*head_ref)->next == NULL) {
        free(*head_ref);
        *head_ref = NULL;
        return;
    }
    struct Node* second_last = *head_ref;
    while (second_last->next->next != NULL) {
        second_last = second_last->next;
    }
    free(second_last->next);
```

```
    second_last->next = NULL;
}
void push(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = NULL;

    push(&head, 7);
    push(&head, 1);
    push(&head, 4);
    push(&head, 3);
    printf("Created Linked list is: ");
    printList(head);
    deleteLastNode(&head);
    printf("Linked list after deletion of last node: ");
    printList(head);
    deleteLastNode(&head);
     printf("Linked list after deletion of last node: ");
    printList(head);
    return 0;
```

}

OUTPUT:



PROGRAM 9

AIM: Write a program in C to implement insertion in queue (enqueue)

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

int isFull() {
   return (rear == MAX_SIZE - 1);
}
int isEmpty() {
   return (front == -1);
}

void enqueue(int value) {
   if (isFull()) {
      printf("Queue Overflow! Cannot insert element.\n");
      return;
```

```
    }
    if (isEmpty()) {
        front = 0;
    }
    rear++;
    queue[rear] = value;
    printf("%d enqueued to queue\n", value);
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    return 0;
```

OUTPUT:

```
Output

10 enqueued to queue
20 enqueued to queue
30 enqueued to queue


=== Code Execution Successful ===
```

PROGRAM 10

AIM: Write a program in C to implement deletion in queue (dequeue)

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
```

```
int queue[MAX_SIZE];
int front = -1;
int rear = -1;
int isEmpty() {
  return front == -1;
}
void dequeue() {
  if (isEmpty()) {
    printf("Queue is empty. Cannot delete element.\n");
    return;
  }
  printf("Deleted element: %d\n", queue[front]);
  if (front == rear) { // If only one element in queue
    front = rear = -1;
  } else {
    front++;
  }
}
void display() {
  if (isEmpty()) {
    printf("Queue is empty.\n");
    return;
  }
  printf("Queue elements: ");
  for (int i = front; i <= rear; i++) {
    printf("%d ", queue[i]);
  }
  printf("\n");
}
int main() {
```

```
    queue[++rear] = 10;
    if (front == -1) front = 0;
    queue[++rear] = 20;
    queue[++rear] = 30;

    printf("Initial ");
    display();

    dequeue();
    printf("After first deletion ");
    display();

    dequeue();
    printf("After second deletion ");
    display();

    dequeue();
    printf("After third deletion ");
    display();

    dequeue();
    return 0;
}
```

OUTPUT:

Output

```
Initial Queue elements: 10 20 30
Deleted element: 10
After first deletion Queue elements: 20 30
Deleted element: 20
After second deletion Queue elements: 30
Deleted element: 30
After third deletion Queue is empty.
Queue is empty. Cannot delete element.


=== Code Execution Successful ===
```

PROGRAM 11

AIM: Write a program in C to perform peek operation in queue

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

typedef struct {
    int arr[MAX_SIZE];
    int front;
    int rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
```

```
}
int isEmpty(Queue *q) {
    return (q->front == -1 && q->rear == -1);
}
int isFull(Queue *q) {
    return (q->rear == MAX_SIZE - 1);
}
void enqueue(Queue *q, int data) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->arr[q->rear] = data;
}
int peek(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot peek.\n");
        return -1;
    }
    return q->arr[q->front];
}
int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
```

```
    enqueue(&q, 30);

    printf("Front element of the queue: %d\n", peek(&q));
    return 0;
}
```

OUPUT:

```
Output

Front element of the queue: 10


=== Code Execution Successful ===
```

PROGRAM 12

AIM: Write a program in C to perform isEmpty operation in queue

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

typedef struct {
    int arr[MAX_SIZE];
    int front, rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}
```

```
int isEmpty(Queue *q) {
    return (q->front == -1 && q->rear == -1);
}

int isFull(Queue *q) {
    return (q->rear == MAX_SIZE - 1);
}

void enqueue(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->arr[q->rear] = value;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int value = q->arr[q->front];
    if (q->front == q->rear) {
        initializeQueue(q);
    } else {
        q->front++;
    }
```

```
    return value;
}

int main() {
    Queue q;
    initializeQueue(&q);
    printf("Is queue empty? %s\n", isEmpty(&q) ? "Yes" : "No");

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    printf("Is queue empty? %s\n", isEmpty(&q) ? "Yes" : "No");

    printf("Dequeued: %d\n", dequeue(&q));
    printf("Dequeued: %d\n", dequeue(&q));
    printf("Dequeued: %d\n", dequeue(&q));
    printf("Is queue empty? %s\n", isEmpty(&q) ? "Yes" : "No");
    return 0;
}
```

OUTPUT:

Output

```
Is queue empty? Yes
Is queue empty? No
Dequeued: 10
Dequeued: 20
Dequeued: 30
Is queue empty? Yes


=== Code Execution Successful ===
```