Write a program in C to perform merging of two one-dimensional arrays, after taking all the necessary

**ASSIGNMENT 01: -**

**Write a program in C to perform merging of two one-dimensional arrays, after taking all the necessary inputs from the user.**

**SOLUTION:-**

### **Algorithm: Merge Two Arrays**

**Step 1:** [INITIALIZATION] Declare integer variables arr1[03], arr2[03], merge\_array[n1 + n2], n1, n2, and i.

**Step 2:** Print "Enter the element number in first array:".  
**Step 3:** Read n1.

**Step 4:** Print "Enter the elements in first array:".  
**Step 5:** Repeat Steps 6 and 7 while i < n1:  
**Step 6:** Read arr1[i].  
**Step 7:** Set i = i + 1.

**Step 8:** Print "Enter the element number in second array:".  
**Step 9:** Read n2.

**Step 10:** Print "Enter the elements in second array:".  
**Step 11:** Repeat Steps 12 and 13 while i < n2:  
**Step 12:** Read arr2[i].  
**Step 13:** Set i = i + 1.

**Step 14:** Initialize i = 0.

**Step 15:** Repeat Steps 16 and 17 while i < n1:  
**Step 16:** Set merge\_array[i] = arr1[i].  
**Step 17:** Set i = i + 1.

**Step 18:** Initialize i = 0.

**Step 19:** Repeat Steps 20 and 21 while i < n2:  
**Step 20:** Set merge\_array[n1 + i] = arr2[i].  
**Step 21:** Set i = i + 1.

**Step 22:** Print "MERGE ARRAY IS :".

**Step 23:** Initialize i = 0.

**Step 24:** Repeat Steps 25 and 26 while i < (n1 + n2):  
**Step 25:** Print merge\_array[i].  
**Step 26:** Set i = i + 1.

**Step 27:** EXIT.

* **SOURCE CODE:**

#include<stdio.h>

int main(){

int arr1[03],arr2[03];

int n1,n2,i;

printf("Enter the element number in first array: ");

scanf("%d",&n1);

printf("Enter the elemet in first array : ");

for(i=0;i<n1;i++){

scanf("%d",&arr1[i]);

}

printf("Enter the element number in second array: ");

scanf("%d",&n2);

printf("Enter the elemet in second array : ");

for(i=0;i<n2;i++){

scanf("%d",&arr2[i]);

}

int merge\_array[n1+n2];

for(i=0;i<n1;i++){

merge\_array[i]=arr1[i];

}

for(i=0;i<n2;i++){

merge\_array[n1+i]=arr2[i];

}

//print merge array

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("MERGE ARRAY IS : ");

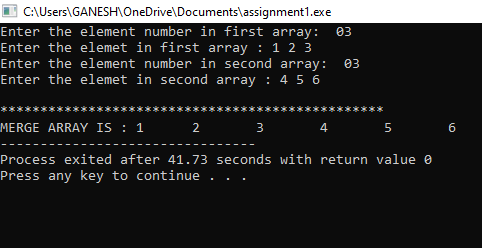
for(i=0;i<n1+n2;i++){

printf("%d\t",merge\_array[i]);

}

return 0;

}

* **OUTPUT:** 

### **Conclusion:**

Merging two arrays in a third array means first copying the contents of the first array into the array and then copying the contents of the second array into the third. Hence, the merged array contains contents of the first array followed by the contents of the second array.

**ASSIGNMENT 02:-**

**Write a program in C to implement the following matrix operations:**

**(a) Multiplication of two matrices**

* + 1. **(b)Transpose of a given matrix**

**SOLUTION:-**

1. **Multiplication of two matrices**

### **Algorithm: Matrix Multiplication of Two 2×2 Matrices**

**Input:** Two 2×2 matrices, matrix1 and matrix2  
**Output:** Resultant matrix result[2][2] after multiplication

#### **Step 1:** Start

#### **Step 2:** Declare variables

* matrix1[2][2], matrix2[2][2], result[2][2]
* Integer variables i, j, k for loops

#### **Step 3:** Input first matrix

* Print: "Enter the elements of the first 2x2 matrix:"
* **For** i = 0 to 1 **do**
  + **For** j = 0 to 1 **do**
    - Read matrix1[i][j]

#### **Step 4:** Input second matrix

* Print: "Enter the elements of the second 2x2 matrix:"
* **For** i = 0 to 1 **do**
  + **For** j = 0 to 1 **do**
    - Read matrix2[i][j]

#### **Step 5:** Initialize result[2][2] with 0

#### **Step 6:** Perform Matrix Multiplication

* **For** i = 0 to 1 **do**
  + **For** j = 0 to 1 **do**
    - Set result[i][j] = 0
    - **For** k = 0 to 1 **do**
      * result[i][j] = result[i][j] + (matrix1[i][k] \* matrix2[k][j])

#### **Step 7:** Display the Resultant Matrix

* Print: "The resulting matrix after multiplication is:"
* **For** i = 0 to 1 **do**
  + **For** j = 0 to 1 **do**
    - Print result[i][j]
  + Print new line

#### **Step 8:** End

* **SOURCE CODE**:

#include <stdio.h>

int main() {

int i, j, k;

int matrix1[2][2], matrix2[2][2], result[2][2] = {0};

// Input for the first matrix

printf("Enter the elements of the first 2x2 matrix:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

scanf("%d", &matrix1[i][j]);

}

}

// Input for the second matrix

printf("Enter the elements of the second 2x2 matrix:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

scanf("%d", &matrix2[i][j]);

}

}

// Matrix multiplication logic

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

result[i][j] = 0; // Initialize each element of result to 0

for (k = 0; k < 2; k++) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

// Display the resulting matrix

printf("The resulting matrix after multiplication is:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

printf("\t%d", result[i][j]);

}

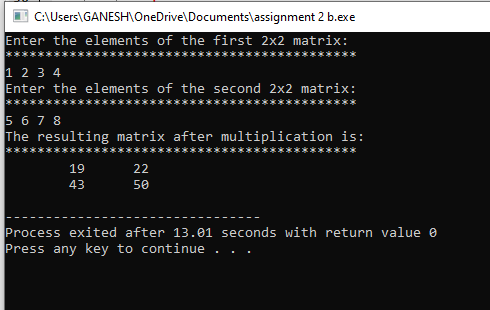
printf("\n"); // New line for each row

}

return 0;

}

* **OUTPUT**:



### **Conclusion:**

Matrix multiplication is a mathematical operation that takes two matrices and produces a new matrix as the result. In this program, we implemented the multiplication of two 2x2 matrices using nested loops. The algorithm involves multiplying the elements of the rows of the first matrix with the corresponding columns of the second matrix and summing the results to compute each element of the resultant matrix.

**(b)Transpose of a given matrix**

### **Algorithm: Transpose Matrix of 3×3 Matrix**

**Step 1:** Declare matrices

DECLARE matrix[3][3], Transposed\_matrix[3][3]

**Step 2:** Input the matrix elements

PRINT "Enter the elements of the matrix (3x3):"

FOR i ← 0 TO 2 DO

FOR j ← 0 TO 2 DO

READ matrix[i][j]

END FOR

END FOR

**Step 3:** Display the original matrix

PRINT "The elements of the matrix are:"

FOR i ← 0 TO 2 DO

FOR j ← 0 TO 2 DO

PRINT matrix[i][j] WITH TAB

END FOR

PRINT NEW LINE

END FOR

**Step 4:** Transpose the matrix

FOR i ← 0 TO 2 DO

FOR j ← 0 TO 2 DO

Transposed\_matrix[i][j] ← matrix[j][i]

END FOR

END FOR

**Step 5:** Display the transposed matrix

PRINT "The elements of the transposed matrix are:"

FOR i ← 0 TO 2 DO

FOR j ← 0 TO 2 DO

PRINT Transposed\_matrix[i][j] WITH TAB

END FOR

PRINT NEW LINE

END FOR

END

* **SOURCE CODE**:

#include <stdio.h>

int main() {

int i, j, matrix[3][3], Transposed\_matrix[3][3];

// Input the matrix elements

printf("Enter the elements of the matrix (3x3): \n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

scanf("%d", &matrix[i][j]);

}

}

// Display the original matrix

printf("The elements of the matrix are: \n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

printf("\t%d", matrix[i][j]);

}

printf("\n");

}

// Transpose the matrix

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

Transposed\_matrix[i][j] = matrix[j][i];

}

}

// Display the transposed matrix

printf("The elements of the transposed matrix are: \n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

printf("\t%d", Transposed\_matrix[i][j]);

}

printf("\n");

}

return 0;

}

* **OUTPUT**:

### **Conclusion:**

In this program, we created the transpose of a 3x3 matrix. Transpose means converting the rows of a matrix into columns and the columns into rows. First, the user inputs the elements of the matrix, and then nested loops are used to calculate the transpose. In this process, each element of the matrix is copied to its new position. Finally, both the original and transposed matrices are displayed separately. This program is a basic yet powerful example of matrix manipulation and nested loops, making linear algebra and programming concepts simple and interesting. Such operations are essential in real-world applications like computer graphics, scientific computing, and data processing. This program also serves as a great practice example for new programmers.

**ASSIGNMENT 05:-**

**Write a menu driven program in C to perform insertion and deletion operations in a queue. Check for exception conditions.**

**SOLUTION**

* **Algorithm:Queue Implementation using Linked List**

**Input:** User selects an operation (Insertion, Deletion, or Display)  
**Output:** Queue is modified according to the selected operation

**Step 1:** Start

### **Step 2:** Define the Node Structure

* Create a structure node with two members:
  + data → to store the integer value
  + next → pointer to the next node

### **Step 3:** Define the Queue Structure

* Create a structure queue with two pointers:
  + front → points to the first node of the queue
  + rear → points to the last node of the queue

### **Step 4:** Initialize the Queue

* Set front = NULL and rear = NULL

### **Step 5:** Display the Main Menu

1. Print the following menu:
   * 1. Insertion
   * 2. Deletion
   * 3. Display
   * 4. Exit
2. Accept user input as option
3. Perform the corresponding operation

**Step 6:** Insert an Element into the Queue

1. Allocate memory for a new node
2. If memory allocation fails, print "Memory allocation failed" and return
3. Store the input value in data of the new node
4. Set next of the new node as NULL
5. **If queue is empty (front == NULL)**, then:
   * Set front = rear = new node
6. **Else**, link the new node to the rear and update rear
7. Print "Element inserted successfully"

**Step 7:** Delete an Element from the Queue

1. **If queue is empty (front == NULL)**, print "Queue Underflow" and return
2. Store the front node in a temporary pointer
3. Print the value of the front node (to be deleted)
4. Move front to the next node
5. If front becomes NULL, set rear = NULL (queue is now empty)
6. Free the memory of the deleted node

**Step 8:** Display the Queue Elements

1. **If queue is empty (front == NULL)**, print "QUEUE IS EMPTY" and return
2. Traverse from front to rear, printing each node’s data

**Step 9:** Repeat Until User Selects Exit (option == 4)

### **Step 10:** End

* **SOURCE CODE**:

#include <stdio.h>

#include <stdlib.h>

// Define the node structure

struct node {

int data;

struct node \*next;

};

// Define the queue structure

struct queue {

struct node \*front;

struct node \*rear;

};

// Function prototypes

void create\_queue(struct queue \*q);

void insert(struct queue \*q, int val);

void delete\_element(struct queue \*q);

void display(struct queue \*q);

int main() {

struct queue q; // Define queue

create\_queue(&q); // Initialize queue

int val, option;

do {

printf("\n\*\*\*\*\*\*\*\*\*\* MAIN MENU \*\*\*\*\*\*\*\*\*\*\n");

printf("1. Insertion\n");

printf("2. Deletion\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter your option: ");

scanf("%d", &option);

switch (option) {

case 1:

printf("Enter the number to be inserted in the queue: ");

scanf("%d", &val);

insert(&q, val);

break;

case 2:

delete\_element(&q);

break;

case 3:

display(&q);

break;

case 4:

printf("Exiting the program.\n");

break;

default:

printf("Invalid option! Please try again.\n");

}

} while (option != 4);

return 0;

}

// Function to initialize the queue

void create\_queue(struct queue \*q) {

q->front = NULL;

q->rear = NULL;

}

// Function to insert an element into the queue

void insert(struct queue \*q, int val) {

struct node \*ptr = (struct node \*)malloc(sizeof(struct node));

if (!ptr) {

printf("Memory allocation failed.\n");

return;

}

ptr->data = val;

ptr->next = NULL;

if (q->front == NULL) { // If the queue is empty

q->front = ptr;

q->rear = ptr;

} else { // If the queue is not empty

q->rear->next = ptr;

q->rear = ptr;

}

printf("%d has been inserted into the queue.\n", val);

}

// Function to delete an element from the queue

void delete\_element(struct queue \*q) {

if (q->front == NULL) { // Check for underflow

printf("Queue Underflow! Cannot delete element.\n");

return;

}

struct node \*ptr = q->front;

printf("The value being deleted is: %d\n", ptr->data);

q->front = q->front->next;

if (q->front == NULL) { // If the queue becomes empty

q->rear = NULL;

}

free(ptr);

}

// Function to display the elements of the queue

void display(struct queue \*q) {

if (q->front == NULL) { // Check if the queue is empty

printf("QUEUE IS EMPTY.\n");

return;

}

struct node \*ptr = q->front;

printf("The elements in the queue are: ");

while (ptr != NULL) {

printf("%d ", ptr->data);

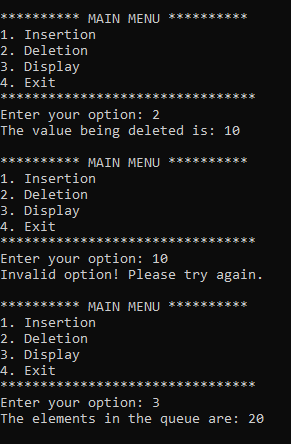
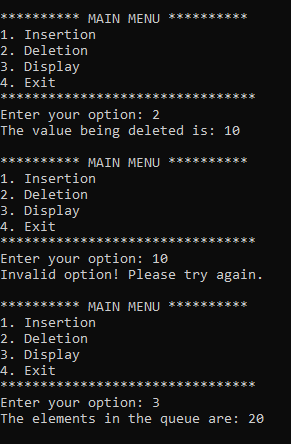
ptr = ptr->next;

}

printf("\n");

}

* **OUTPUT:**



**Fig.01 Fig.02**

### **Conclusion:**

In this program, we implemented a menu-driven approach to perform queue operations such as insertion and deletion. A queue is a linear data structure that works on the "First In, First Out" (FIFO) principle. The program uses a linked list for dynamic memory allocation, overcoming the limitations of fixed-size arrays.

We effectively handled exception conditions such as queue underflow (when attempting to delete from an empty queue) and memory allocation failure (when memory cannot be allocated). Additionally, the display function was used to show the current elements of the queue, helping to understand the real-time state of the queue.

This program serves as an excellent example of practical implementation of data structures and error handling concepts. It helps beginners strengthen their understanding of data structures while improving their coding skills.

**SOLUTION:-**

**ASSIGNMENT 07:-**

**Write a program in C to create a single linked list and implement the following functions on it—**

**a. Insert after a specific location**

* 1. **b. Deletion at end**
  2. **c. Delete an element from a specified location.**
  3. **d. Count the number of nodes.**
  4. **e. Separate the linked list into odd numbered and even numbered list**
  5. **f. Display the list.**
* **ALGORITHM:**
* **SOURCE CODE**:

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

struct node \*start = NULL, \*temp = NULL, \*newnode, \*prev, \*forw;

void create\_ll();

void insert\_after\_location();

void delete\_from\_location();

void delete\_last();

void count\_node();

void separate\_odd\_even();

void display\_ll();

int main() {

int choice;

do {

printf("\n\*\*\*\*\* Linked List Menu \*\*\*\*\*\*\n");

printf("01) Create linked list\n");

printf("02) Insert after specific location\n");

printf("03) Delete from last\n");

printf("04) Delete from specific location\n");

printf("05) Count total nodes\n");

printf("06) Separate into odd & even lists\n");

printf("07) Display linked list\n");

printf("08) Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: create\_ll(); break;

case 2: insert\_after\_location(); break;

case 3: delete\_last(); break;

case 4: delete\_from\_location(); break;

case 5: count\_node(); break;

case 6: separate\_odd\_even(); break;

case 7: display\_ll(); break;

case 8: printf("Exiting program.\n"); break;

default: printf("Invalid choice!\n");

}

} while (choice != 8);

return 0;

}

void create\_ll() {

char ch;

int n;

printf("Enter an element: ");

scanf("%d", &n);

start = (struct node \*)malloc(sizeof(struct node));

start->data = n;

start->next = NULL;

temp = start;

printf("Do you want to continue (Y/N)? ");

getchar();

ch = getchar();

while (ch == 'Y' || ch == 'y') {

printf("Enter next element: ");

scanf("%d", &n);

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = n;

newnode->next = NULL;

temp->next = newnode;

temp = newnode;

printf("Do you want to continue (Y/N)? ");

getchar();

ch = getchar();

}

}

void insert\_after\_location() {

int pos, i = 1, n;

if (start == NULL) {

printf("List is empty\n");

return;

}

printf("Enter position after which to insert: ");

scanf("%d", &pos);

printf("Enter element to insert: ");

scanf("%d", &n);

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = n;

newnode->next = NULL;

temp = start;

while (i < pos && temp != NULL) {

temp = temp->next;

i++;

}

if (temp == NULL) {

printf("Invalid position!\n");

return;

}

newnode->next = temp->next;

temp->next = newnode;

}

void delete\_from\_location() {

int pos, i = 1;

if (start == NULL) {

printf("List is empty\n");

return;

}

printf("Enter position to delete: ");

scanf("%d", &pos);

temp = start;

if (pos == 1) {

start = start->next;

printf("Deleted node: %d\n", temp->data);

free(temp);

return;

}

while (i < pos - 1 && temp->next != NULL) {

temp = temp->next;

i++;

}

if (temp->next == NULL) {

printf("Invalid position!\n");

return;

}

struct node \*del = temp->next;

temp->next = del->next;

printf("Deleted node: %d\n", del->data);

free(del);

}

void delete\_last() {

if (start == NULL) {

printf("List is empty\n");

return;

}

temp = start;

if (temp->next == NULL) {

printf("Deleted node: %d\n", temp->data);

free(temp);

start = NULL;

return;

}

while (temp->next->next != NULL) {

temp = temp->next;

}

printf("Deleted node: %d\n", temp->next->data);

free(temp->next);

temp->next = NULL;

}

void count\_node() {

int count = 0;

temp = start;

while (temp != NULL) {

count++;

temp = temp->next;

}

printf("Total nodes: %d\n", count);

}

void separate\_odd\_even() {

struct node \*odd = NULL, \*even = NULL, \*odd\_tail = NULL, \*even\_tail = NULL;

temp = start;

while (temp != NULL) {

struct node \*newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = temp->data;

newnode->next = NULL;

if (temp->data % 2 == 0) {

if (even == NULL) even = even\_tail = newnode;

else {

even\_tail->next = newnode;

even\_tail = newnode;

}

} else {

if (odd == NULL) odd = odd\_tail = newnode;

else {

odd\_tail->next = newnode;

odd\_tail = newnode;

}

}

temp = temp->next;

}

printf("Odd List: ");

temp = odd;

while (temp) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\nEven List: ");

temp = even;

while (temp) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

void display\_ll() {

if (start == NULL) {

printf("List is empty\n");

return;

}

temp = start;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

### **Output:**

### **Conclusion:**

This program implements various operations on a singly linked list, including creation, insertion, deletion, counting nodes, separating odd and even elements, and displaying the list.

* **Insertion:** A new node can be inserted after a specific location.
* **Deletion:** Nodes can be deleted either from a specific position or from the last.
* **Counting Nodes:** The total number of nodes in the linked list is counted.
* **Odd-Even Separation:** The program separates odd and even elements into two separate linked lists.
* **Display:** The linked list is printed in a structured format.

This implementation ensures efficient manipulation of linked list elements, making it useful for dynamic data storage and processing.

ASSIGNMENT 07:-

**Write a program in C to implement stack operations using array and perform Insertion and Deletion operations. Show all possible exception/error cases**.

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include<stdio.h>

#include<conio.h>

#define maxsize 20

int stack[maxsize],top=-1;

int main(){

int choice;

do{

printf("\n\*\*\*\*\*Stack\*\*\*\*\*\*\n");

printf("------------------------\n");

printf("01)push\n 02)pop\n 03)display\n 04)exit");

printf("-------------------------\n\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

default :printf("Invalid choice");

}

}while(choice!=4);

return 0;

}

void push(){

int n;

if(top==maxsize-1) printf("Stack is overflow");

else{

printf("Enter an element :");

scanf("%d",&n);

top++;

stack[top]=n;

}

}

void display(){

int i;

if(top==-1) printf("Stack is empty");

else{

printf("\n Elemts of stacks: ");

for(i=top;i>=0;i--){

printf(" %d \t",stack[i]);

}

}

}

void pop(){

int n;

if(top==-1) printf("Stack is Empty");

else{

n=stack[top];

top--;

printf("poped element %d \n",n);

}

}

* **OUTPUT:**
* **CONCLUSION**:

ASSIGNMENT 11:-

**Write a program in C to create a singly linked list with even number of elements. Then swap the odd and even positioned elements to create a new list. Display both the original and changed list.**

1. Write a program in C to create a singly linked list with even number of elements. Then swap the odd and even positioned elements to create a new list. Display both the original and changed list.

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

struct node \*start = NULL, \*temp = NULL, \*newnode;

void create\_even\_ll();

void display\_ll(struct node \*head);

void swap\_odd\_even();

int main() {

printf("Creating an even-length linked list:\n");

create\_even\_ll();

printf("\nOriginal Linked List:\n");

display\_ll(start);

swap\_odd\_even();

printf("\nLinked List after swapping odd and even positioned nodes:\n");

display\_ll(start);

return 0;

}

void create\_even\_ll() {

int n, count = 0;

char ch;

do {

printf("Enter an element: ");

scanf("%d", &n);

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = n;

newnode->next = NULL;

if (start == NULL) {

start = newnode;

temp = start;

} else {

temp->next = newnode;

temp = newnode;

}

count++;

printf("Do you want to continue (Y/N)? ");

scanf(" %c", &ch);

} while ((ch == 'Y' || ch == 'y') || count % 2 != 0); // Ensures even-length list

}

void display\_ll(struct node \*head) {

if (head == NULL) {

printf("List is empty\n");

return;

}

struct node \*temp = head;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void swap\_odd\_even() {

if (start == NULL || start->next == NULL) {

return;

}

struct node \*temp = start;

while (temp != NULL && temp->next != NULL) {

int swap = temp->data;

temp->data = temp->next->data;

temp->next->data = swap;

temp = temp->next->next;

}

}

* **OUTPUT:**

### **Conclusion:**

ASSIGNMENT 28:-

**Write a program in C to apply the following sorting techniques on a list of numbers, as per user's choice**

* 1. **a. Insertion Sort**
  2. **b. Selection Sort.**

**Show intermediate results between passes.**

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include <stdio.h>

#define MAX 100

void insertion\_sort();

void selection\_sort();

int main() {

int choice;

do {

printf("\n\*\*\*\*\* Sorting Technique \*\*\*\*\*\*\n");

printf("----------------------------------------------\n");

printf("01) Insertion Sort\n02) Selection Sort\n03) Exit\n");

printf("----------------------------------------------\n\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: insertion\_sort(); break;

case 2: selection\_sort(); break;

case 3: printf("Exiting program...\n"); break;

default: printf("Invalid choice. Please try again.\n");

}

} while (choice != 3);

return 0;

}

// Function to perform Insertion Sort with intermediate steps

void insertion\_sort() {

int arr[MAX], n, i, j, key;

printf("Enter the number of elements: ");

scanf("%d", &n);

printf("Enter the elements: ");

for (i = 0; i < n; i++)

scanf("%d", &arr[i]);

printf("\nInsertion Sort Process:\n");

for (i = 1; i < n; i++) {

key = arr[i];

j = i - 1;

// Shift elements to the right

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

// Display intermediate array

printf("After pass %d: ", i);

for (int k = 0; k < n; k++)

printf("%d ", arr[k]);

printf("\n");

}

printf("\nSorted Array: ");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Function to perform Selection Sort with intermediate steps

void selection\_sort() {

int arr[MAX], n, temp, i, j, index, min;

printf("Enter the number of elements: ");

scanf("%d", &n);

printf("Enter the elements: ");

for (i = 0; i < n; i++)

scanf("%d", &arr[i]);

printf("\nSelection Sort Process:\n");

for (i = 0; i < n - 1; i++) {

min = arr[i];

index = i;

for (j = i + 1; j < n; j++) {

if (min > arr[j]) {

min = arr[j];

index = j;

}

}

// Swap

temp = arr[i];

arr[i] = arr[index];

arr[index] = temp;

// Display intermediate array

printf("After pass %d: ", i + 1);

for (int k = 0; k < n; k++)

printf("%d ", arr[k]);

printf("\n");

}

printf("\nSorted Array: ");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

* **OUTPUT:**

### **Conclusion:**

ASSIGNMENT 20:-

**Write a program in C to input a list of integers and sort the list using bubble sort.**

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include<stdio.h>

#define MAX 10

int main() {

int temp, n, i, j;

int arr[MAX];

printf("Enter the number of elements (max %d): ", MAX);

scanf("%d", &n);

// Check for valid input size

if (n > MAX || n <= 0) {

printf("Invalid input! The number of elements should be between 1 and %d.\n", MAX);

return 1; // Exit with an error

}

printf("Enter the elements: ");

for (i = 0; i < n; i++) scanf("%d", &arr[i]);

// Bubble sort logic

for (i = 0; i < n - 1; i++) { // n-1 passes

for (j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("Sorted array: ");

for (i = 0; i < n; i++) printf("%d\t", arr[i]);

printf("\n");

return 0;

}

* **OUTPUT:**

### **Conclusion:**

ASSIGNMENT 06:-

**Write a program in C to implement the standard circular queue operations (i.e. Insert, Delete, Display) using an array.**

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include<stdio.h>

#include<conio.h>

#define MAXSIZE 20

int cq[MAXSIZE],rear=-1,front=-1;

void insertion();

void deletion();

void display();

int main(){

int choice;

do{

printf("\n\*\*\*\*\*Circular Queue\*\*\*\*\*\*\n");

printf("------------------------\n");

printf("01)Insertion\n 02)Deletion\n 03)display\n 04)exit\n");

printf("-------------------------\n\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: insertion(); break;

case 2: deletion(); break;

case 3: display(); break;

default :printf("Invalid choice");

}

}while(choice!=4);

return 0;

}

void insertion(){

int n;

if((rear+1)%MAXSIZE==front)

printf("Queue is overflow");

else{

printf("Enter an element for insert: ");

scanf("%d",&n);

if(rear==-1 && front==-1){

rear=0;

front=0;

}

else

rear=(rear+1)%MAXSIZE;

cq[rear]=n;

}

}

void deletion(){

int n;

if(rear==-1 && front==-1)

printf("Queue is empty");

else{

n=cq[front];

if(front==rear){

front=-1;

rear=-1;

}

else

front=(front+1)%MAXSIZE;

printf("Deleted Element %d",n);

}

}

void display(){

int i;

if(rear==-1 && front==-1)

printf("Queue is empty");

else{

printf("Elements of queue: ");

for(i=front;i!=rear;i=(i+1)%MAXSIZE)

printf("%5d",cq[i]);

printf("%5d",cq[i]);

}

}

**OUTPUT:**

### **Conclusion:**

ASSIGNMENT 14:-

**Write a program in C to create a doubly linked list and perform the following operations:**

**a) Display**

**b) Deleting a node from any position**.

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:

#include<stdio.h>

#include<stdlib.h>

struct node {

struct node \*prev;

int data;

struct node \*next;

};

struct node \*start = NULL, \*temp = NULL, \*newnode, \*prev, \*forw, \*next;

void create\_ll();

void delete\_any\_position();

void display\_ll();

int main() {

int choice;

do {

printf("\n\*\*\*\*\* Doubly Linked List Menu\*\*\*\*\*\*\n");

printf("------------------------\n");

printf("01) Create linked list\n");

printf("02) Delete from any position\n");

printf("03) Display linked list\n");

printf("04) Exit\n");

printf("-------------------------\n\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1: create\_ll(); break;

case 2: delete\_any\_position(); break; // New case for deleting from any position

case 3: display\_ll(); break;

default: printf("Invalid choice\n");

}

} while(choice != 4);

return 0;

}

void create\_ll() {

char ch;

int n;

printf("Enter an element: ");

scanf("%d", &n);

// Create the first node

start = (struct node \*)malloc(sizeof(struct node));

start->prev = NULL;

start->data = n;

start->next = NULL;

temp = start;

printf("\nDo you want to continue (y/n)? ");

ch = getche(); // Reads user input for continuation

while (ch == 'Y' || ch == 'y') {

printf("\nEnter next element: ");

scanf("%d", &n);

// Allocate memory for a new node

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->prev = temp; // Link new node's prev to the current last node

newnode->data = n;

newnode->next = NULL;

temp->next = newnode; // Link the current last node to the new node

temp = newnode; // Move temp to the new last node

printf("\nDo you want to continue (y/n)? ");

ch = getche();

}

}

void display\_ll() {

if (start == NULL) {

printf("List is already empty\n");

return;

}

// Forward traversal

printf("Forward Traversal: ");

temp = start;

while (temp != NULL) {

printf("%d\t", temp->data);

if (temp->next == NULL) {

break; // Stop at the last node to use it for backward traversal

}

temp = temp->next;

}

// Backward traversal

printf("\nBackward Traversal: ");

while (temp != NULL) {

printf("%d\t", temp->data);

temp = temp->prev;

}

printf("\n");

}

void delete\_any\_position() { // New function for deleting a node from any position

int pos, i = 1;

if(start == NULL) {

printf("List is empty\n");

} else {

printf("Enter the position to delete from: ");

scanf("%d", &pos);

temp = start;

while(i < pos && temp != NULL) {

temp = temp->next;

i++;

}

if(temp != NULL) {

prev = temp->prev;

next = temp->next;

if(prev != NULL) prev->next = next;

if(next != NULL) next->prev = prev;

printf("Deleted node %d\n", temp->data);

free(temp);

} else {

printf("Invalid position\n");

}

}

}

* **OUTPUT:**

### **Conclusion:**

ASSIGNMENT :-

**SOLUTION:-**

* **ALGORITHM:**
* **SOURCE CODE**:
* **OUTPUT:**

### **Conclusion:**

ASSIGNMENT 09:-

**Write a program in C to reverse a single linked list.**

**SOLUTION:-**

### **Algorithm for Reversing a Linked List**

1. **Initialize:**
   * Set prev = NULL (to store the previous node).
   * Set current = start (to traverse the list).
   * Set next = NULL (to store the next node).
2. **Traverse the List:**
   * While current is not NULL:
     + Store next = current->next (save the next node).
     + Set current->next = prev (reverse the link).
     + Move prev = current (move prev forward).
     + Move current = next (move current forward).
3. **Update Head:**
   * Set start = prev (new head of the reversed list).
   * Print "Linked List has been reversed."

* **SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node \*start=NULL,\*temp=NULL,\*newnode,\*next;

void create\_ll();

void display\_ll();

void reverse\_ll();

int main(){

int choice;

do{

printf("\n\*\*\*\*\*Linked List Menu\*\*\*\*\*\*\n");

printf("------------------------\n");

printf("01)Create linkeds list\n");

printf("02)Dsiplay\_ll\n");

printf("03)reverse\_ll\n");

printf("04)Exit\n");

printf("-------------------------\n\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: create\_ll(); break;

case 2: display\_ll(); break;

case 3: reverse\_ll(); break;

default :printf("Invalid choice");

}

}while(choice!=12);

return 0;

}

void create\_ll(){

char ch;

int n;

printf("Enter an element:\t");

scanf("%d",&n);

start=(struct node \*)malloc(sizeof(struct node));

start->data=n;

start->next=NULL;

temp=start;

printf("\nDo you Want to continue: ");

ch=getch();

while(ch=='Y'||ch=='y'){

printf("\nEnter next element: ");

scanf("%d",&n);

newnode=(struct node \*)malloc(sizeof(struct node));

newnode->data=n;

newnode->next=NULL;

temp->next=newnode;

temp=temp->next;

printf("\nDo you Want to continue: ");

ch=getche();

}

}

void display\_ll(){

if(start==NULL) printf("List is alredy empty");

else{

temp=start;

while(temp!=NULL){

printf("%d\t",temp->data);

temp=temp->next;

}

}

}

void reverse\_ll() {

struct node \*prev = NULL, \*current = start, \*next = NULL;

while (current != NULL) {

next=current->next; // Store the next node

current->next = prev; // Reverse the link

prev=current; // Move prev to the current node

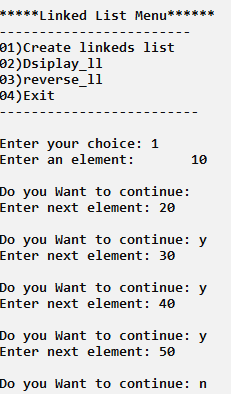
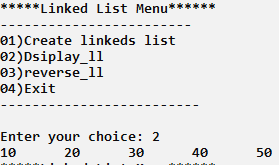
current=next; // Move to the next node

}

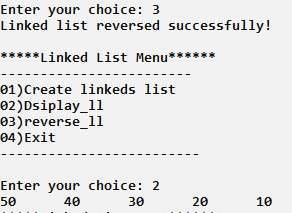
start=prev; // Update start to point to the new head

printf("Linked list reversed successfully!\n");

}:

* **OUTPUT:**

**Fig.02**



**Fig.01**

***Fiig.03***

### **Conclusion:**

This program implements a singly linked list with basic operations such as creation, display, and reversal. The create\_ll() function allows the user to add nodes dynamically. The display\_ll() function prints all elements of the linked list. The reverse\_ll() function reverses the linked list by adjusting the next pointers of each node without requiring additional memory allocation.

By using pointer manipulation, the program efficiently reverses the linked list while maintaining its structure. The implementation ensures proper memory management and user interaction through a menu-driven approach.