

**Basics of**

**Data Structures and Algorithms**

Prateek Panwar

Index

|  |  |  |
| --- | --- | --- |
| SN | Program | Page No. |
| 1 | Program to find an average of an array using AVG function | 3 |
| 2 | Program that can insert, delete and edit an element in array | 4 |
| 3 | Algorithm for insert and delete operations of circular queue | 9 |
| 4 | Program to implement stack with static memory allocation | 13 |
| 5 | Program to stack with dynamic memory allocation | 17 |
| 6 | Program to linear queue static memory allocation | 22 |
| 7 | Program to implement linear queue with dynamic memory allocation | 30 |
| 8 | Program to implement linear linked list | 37 |
| 9 | Program to implement circular linked list | 44 |
| 10 | Program for implementation of Bubble sort | 46 |
| 11 | Program for insertion sort | 46 |
| 12 | Program to merge sort | 48 |

1. Develop a program to find an average of an array using AVG function.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For  Visual Studio, Remove in other compilers\*/

#include <stdio.h>

void avg(int a[], int n)

{

    float sum = 0, avg = 0;

    for (int i = 0; i < n; i++)

    {

        sum = sum + a[i];

    }

    avg = sum / n;

    printf("Answer = %f", avg);

}

void main()

{

    int n, a[100];

    printf("Enter number of elements: ");

    scanf("%d", &n);

    printf("Enter values in array: ");

    for (int i = 0; i < n; i++)

    {

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

        scanf("%d", &a[i]);

    }

    avg(a, n);

}

Output:

Graphical user interface, text

Description automatically generated

1. Implement a program that can insert, delete and edit an element in array.

Ans. Program:

//Array Calc

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

#include <stdlib.h>

//Global variables

int arr[100], len\_arr = 0; //len\_arr stores length of array

//Declaring Functions

void insert\_elem();

void edit\_elem();

void delete\_elem();

void display\_arr();

void main()

{

int q = 0, choice=0; // (q) For quitting program / Continuing loop

printf("Array Calculator");

while (choice != 5)

{

printf("\n\nEnter following number to select:");

printf("\n1.Insert Element \n2.Edit Element \n3.Delete Element \n4.Display Array \n5.Quit: ");

scanf("%d", &choice);

switch (choice)

{

case 1: insert\_elem();

break;

case 2: edit\_elem();

break;

case 3: delete\_elem();

break;

case 4: display\_arr();

break;

case 5: exit(0);

default: printf("Error 1: Enter correct value!");

}

}

}

//Functions

//To insert values in the end

void insert\_elem()

{

int n=0;

printf("\nEnter number of elements to insert: ");

scanf("%d", &n);

printf("\nEnter values: ");

if (n > 0)

{

for (int i = len\_arr; i < len\_arr +n; i++)

{

printf("Element %d> ",i+1);

scanf("%d", &arr[i]);

}

len\_arr += n;

}

else

{

printf("Error 2: Enter correct length of elements!");

}

}

//To edit an element and replace

void edit\_elem()

{

int elem;

printf("\nEnter element to edit: ");

scanf("%d", &elem);

printf("\nEnter value: ");

scanf("%d", &arr[elem-1]);

}

//To delete an element

void delete\_elem()

{

int elem;

printf("\nEnter element to delete: ");

scanf("%d", &elem);

for (int i = elem; i < len\_arr; i++) //Loop to shift all elements

{

arr[i - 1] = arr[i];

}

len\_arr--; //Reduce length of array as we yeeted 1 element

}

//To display whole array in table

void display\_arr()

{

printf("\n+----+-----+ \n|Elem|Value| \n+----+-----+");

for (int i = 0; i < len\_arr; i++)

{

printf("\n| %d  |  %d  |",i+1,arr[i]);

}

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

}

Output:

Text

Description automatically generatedPage 1:

2.

Page 2:

Text

Description automatically generated

Page 3:

Text

Description automatically generated

1. Implement an algorithm for insert and delete operations of circular queue same using array.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

int max = 5;

int q[5], front=0, rear=0, count=0, inpt=0, ch=0;

void insert()

{

if (count == max)

{

printf("\nQueue is Full");

}

else

{

int elem; //To enter multiple elements at once

printf("\nEnter number of elements to insert:");

scanf("%d", &elem);

if (elem == 0 || elem>max)

{

printf("\nError 101: Enter correct number of elements");

}

else

{

printf("\nEnter elements: \n");

for (int i = 0; i < elem; i++)

{

printf("> ");

scanf("%d", &inpt);

q[rear] = inpt;

rear = (rear + 1) % max;

count++;

}

printf("\nData inserted in queue");

}

}

}

void delete()

{

if (count == 0)

{

printf("\nQueue is Empty");

}

else

{

printf("\n%d is deleted from queue", q[front]);

front = (front + 1) % max;

count--;

}

}

void display()

{

if (count == 0)

{

printf("\nQueue is Empty");

}

else

{

printf("\nElements of Queue:");

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

int j=0;

for (int i = front; j < count; j++)

{

printf("\n|  %d  |", q[i]);

i = (i + 1) % max;

}

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

}

}

intMenu()

{

ch = 0;

printf("\n\nSelect an option: \n1.Insert Elements \n2.Delete Element \n3.Display Queue \n4.Quit ");

scanf("%d", &ch);

return ch;

}

void main()

{

printf("\n\tCircular Queue Program");

while (!0)

{

ch =Menu();

switch (ch)

{

case 1: insert(); break;

case 2: delete(); break;

case 3: display(); break;

case 4: return; break;

default: printf("\nError 101: Enter correct choice");

}

}

}

Output:

Page 1:

Text

Description automatically generated

Page 2:

Text

Description automatically generated

Page 3:

Text

Description automatically generated

1. Menu driven program to implement the push, pop, and display option of the stack with the help of static memory allocation.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

int arr[20], top, max = 20;

void push()

{

if (top == max) //Checking if stack is full

{

printf("\nStack is full!");

}

else

{

int elem, num;

printf("\nEnter total number of elements: ");

scanf("%d", &elem);

printf("\nEnter elements into Stack: ");

for (int i = 0; i < elem; i++) //To enter multiple elements at once

{

printf("> ");

scanf("%d", &num);

top++;

arr[top] = num;

}

}

}

void pop()

{

if (top == -1) //Checking if stack is empty

{

printf("\nStack is empty!");

}

else

{

printf("\n%d removed from Stack", arr[top]);

top--;

}

}

void display()

{

if (top == -1)

{

printf("\nStack is empty!");

}

else

{

printf("\nElements of Stack:\n");

for (int i = 1; i <= top; i++)

{

printf(" %d", arr[i]);

}

printf("\n%d / %d elements used", top, max);

}

}

intMenu()

{

int ch;

ch = 0;

printf("\n\nSelect an option: \n1.Push \n2.Pop \n3.Display Stack \n4.Quit ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

printf("\nStack using static memory");

while (!0)

{

ch =Menu();

switch (ch)

{

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

case 4: return; break;

default: printf("\nError 101: Enter correct choice");

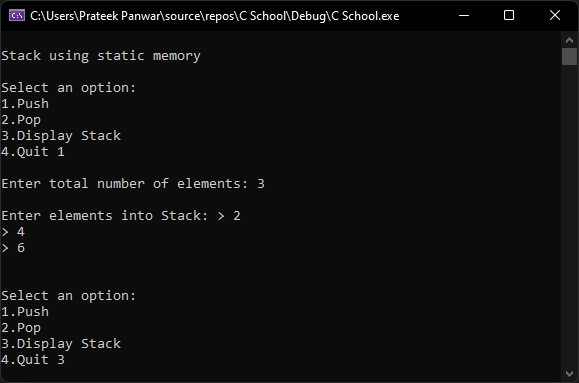
}

}

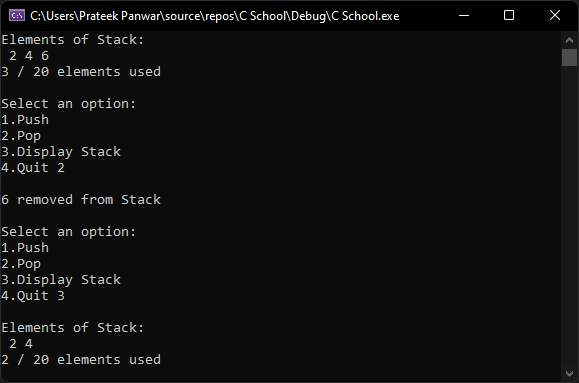
}

Output:

Page 1:



Page 2:



1. Menu driven program to implement the push, pop and display option of the stack with the help of dynamic memory allocation.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

#include <stdlib.h>

int max = 20;

//Node stores data and it's link

struct node

{

int data;

struct node\* link;

}\*top = NULL;

void push()

{

int num, count;

struct node\* newnode;

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

count = stackcount();

if (count <= max - 1)

{

printf("\nEnter element into Stack: ");

scanf("%d", &num);

newnode->data = num;

newnode->link = top;

top = newnode;

}

else

{

printf("\nStack is full!");

}

}

void pop()

{

//Creating pointer of node

struct node\* newnode;

if (top == NULL) //Checking if stack is empty

{

printf("\nStack is empty!");

}

else

{

newnode = top;

printf("\n%d removed from Stack", newnode->data);

top = top->link;

free(newnode);

}

}

void display()

{

int count = 0;

struct node\* newnode; //Creating ptr of node

newnode = top;

if (top == NULL)

{

printf("\nStack is empty!");

}

else

{

printf("\nElements of Stack:\n");

while (newnode != NULL)

{

printf(" %d", newnode->data);

newnode = newnode->link;

count++;

}

printf("\n(%d / %d elements used)", count, max);

}

}

int stackcount() //Count num of elements in stack

{

int count = 0;

struct node\* newnode;

newnode = top;

while (newnode != NULL)

{

newnode = newnode->link;

count++;

}

return count;

}

intMenu()

{

int ch;

ch = 0;

printf("\n\nSelect an option: \n1.Push \n2.Pop \n3.Display Stack \n4.Quit ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

printf("\nStack using dynamic memory");

while (!0)

{

ch =Menu();

switch (ch)

{

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

case 4: return; break;

default: printf("\nError 101: Enter correct choice");

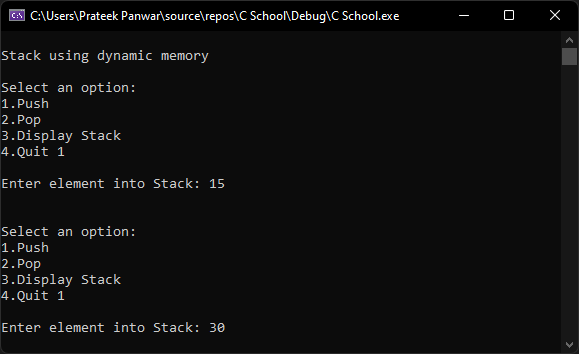
}

}

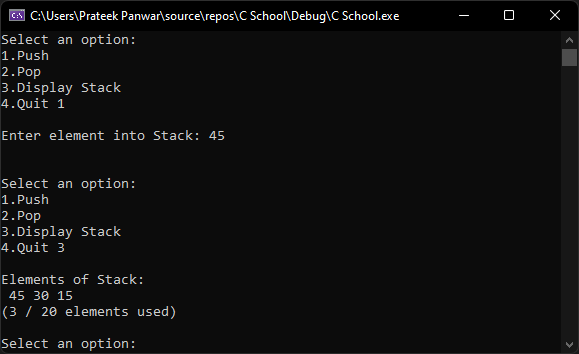
}

Output:

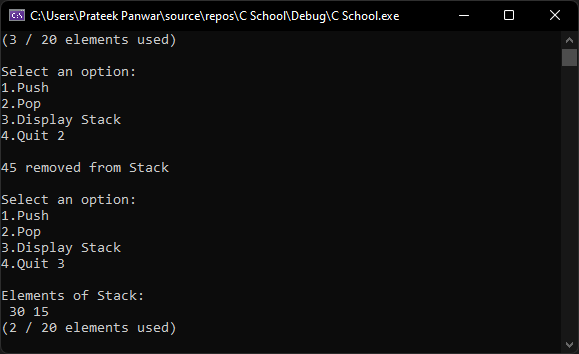
Page 1:



Page 2:



Page 3:



1. Menu driven program to implementing the various operations on a linear queue with the help of static memory allocation.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

int max = 3;

int arr[3], front, rear;

void insert()

{

int num,elem;

if (rear == max)

{

printf("\n Linear Queue is full");

return;

}

else

{

printf("\nEnter total number of elements: ");

scanf("%d", &elem);

printf("\nEnter elements into Queue: ");

for (int i = 0; i < elem; i++) //To enter multiple elements at once

{

printf("> ");

scanf("%d", &num);

arr[rear] = num;

rear++;

}

printf("\nElement is Inserted in the Queue");

}

}

void delete()

{

if (rear == front)

{

printf("\nQueue is Empty..");

return;

}

else

{

printf("\nDeleted element from Queue is %d", arr[front]);

front++;

}

}

void display()

{

if (front == rear)

{

printf("\nQueue is Empty!");

return;

}

else

{

printf("\nElements of Queue: ");

for (int i = front; i < rear; i++)

{

printf(" %d", arr[i]);

}

}

}

intMenu()

{

int ch;

printf("\n\nLinear Queue with static memory");

printf("\n1. (+) Insert \n2. (X) Delete \n3. Display \n4. Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

do

{

ch =Menu();

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

return;

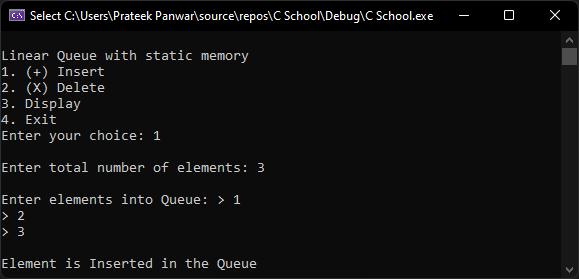
}

} while (1);

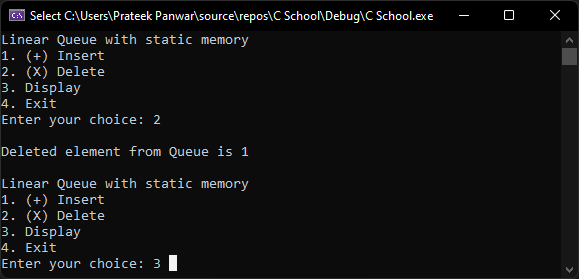
}

Output:

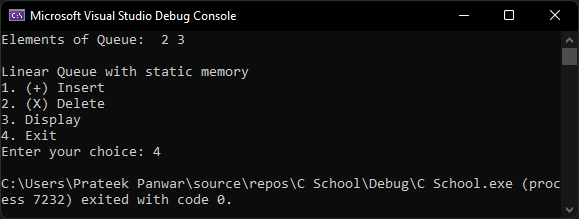
Page 1:



Page 2:



Page 3:



1. Menu driven program to implementing the various operations on a linear queue with the help of dynamic memory allocation.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

struct node

{

int data;

struct node\* link;

}\*front, \* rear;

void insert()

{

int elem;

struct node\* newnode;

printf("\nEnter total number of elements: ");

scanf("%d", &elem);

printf("\nEnter elements into Queue: ");

for (int i = 0; i < elem; i++) //To enter multiple elements at once

{

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

printf("> ");

scanf("%d", &newnode->data);

newnode->link = NULL;

if (rear == NULL)

{

front = rear = newnode;

}

else

{

rear->link = newnode;

rear = newnode;

}

}

printf("\nElement is Inserted in the Queue");

}

void delete()

{

struct node\* newnode;

newnode = front;

if (front == NULL)

{

printf("\nQueue is Empty!");

return;

}

else

{

printf("\nDeleted element from Queue is %d", front->data);

front = front->link;

free(newnode);

}

}

void display()

{

struct node\* newnode;

newnode = front;

if (front == NULL)

{

printf("\nQueue is Empty!");

}

else

{

printf("\nElements of Queue: ");

while (newnode)

{

printf(" %d", newnode->data);

newnode = newnode->link;

}

}

}

intMenu()

{

int ch;

printf("\n\nLinear Queue with dynamic memory");

printf("\n1. (+) Insert \n2. (X) Delete \n3. Display \n4. Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

do

{

ch =Menu();

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

return;

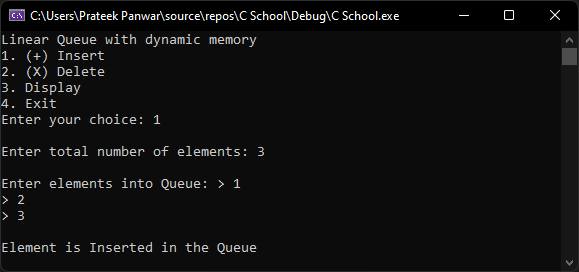
}

} while (1);

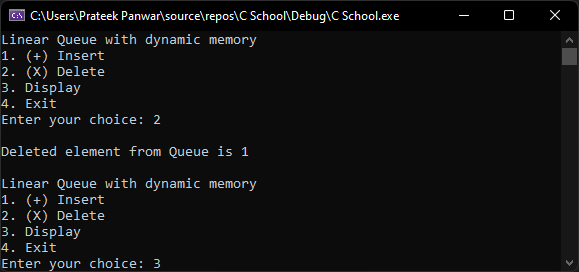
}

Output:

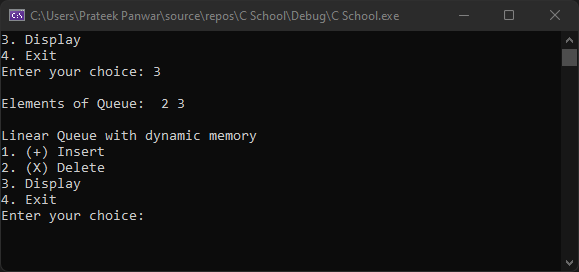
Page 1:



Page 2:



Page 3:



1. Menu driven program to implement various operations on a linear linked list.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

#include <stdlib.h>

int max = 20;

//Node stores data and it's link

struct node

{

int data;

struct node \*next;

};

struct node \*start = NULL;

struct node\* getnode()

{

struct node\* newnode;

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

printf("> ");

scanf("%d", &newnode->data);

newnode->next = NULL;

return newnode;

}

void createlist()

{

if (start == NULL)

{

int n;

struct node\* newnode;

struct node\* temp;

printf("\nEnter number of nodes: ");

scanf("%d", &n);

for (int i = 0; i < n; i++)

{

newnode = getnode();

if (start == NULL)

{

start = newnode;

}

else

{

temp = start;

while (temp->next != NULL)

{

temp = temp->next;

}

temp -> next = newnode;

}

}

}

else

{

printf("\nList is already created!");

}

}

void insertfront()

{

struct node\* newnode;

newnode = getnode();

if (start == NULL)

{

start = newnode;

}

else

{

newnode->next = start;

start = newnode;

}

}

void insertrear()

{

struct node\* newnode, \*temp;

newnode = getnode();

if (start == NULL)

{

start = newnode;

}

else

{

temp = start;

while (temp->next != NULL)

temp = temp->next;

temp->next = newnode;

}

}

void deletefront()

{

struct node\* newnode;

if (start == NULL)

{

printf("No nodes are present");

return;

}

else

{

newnode = start;

start = newnode->next;

free(newnode);

printf("\n Node deleted");

}

}

void deleterear()

{

struct node\* newnode, \*lastnode;

if (start == NULL)

{

printf("No nodes are present");

return;

}

else

{

newnode = start;

lastnode = start;

while (newnode -> next != NULL)

{

lastnode = newnode;

newnode = newnode->next;

}

printf("\n Node deleted");

}

}

void traverse()

{

struct node\* newnode;

newnode = start;

if (start == NULL)

{

printf("\nList is Empty!");

return;

}

else

{

printf("\nElements in list (left -> right):\n");

while (newnode != NULL)

{

printf(" %d ->", newnode->data);

newnode = newnode->next;

}

printf(" X");

}

}

void traverseopp(struct node \*start)

{

if (start == NULL)

{

return;

}

else

{

traverseopp(start->next);

printf(" %d ->", start->data);

}

}

intMenu()

{

int ch;

ch = 0;

printf("\n\nSelect an option: \n1. Create list \n2. (+) Insert element at front \n3. (+) Insert element at end");

printf("\n4. (X) Delete element from front \n5. (X) Delete element from end");

printf("\n6. (T) Traverse list(left->right) \n7. (T) Traverse list(right->left) \n8. Exit ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

printf("\nLinked list");

while (!0)

{

ch =Menu();

switch (ch)

{

case 1: createlist();

break;

case 2: insertfront();

break;

case 3: insertrear();

break;

case 4: deletefront();

break;

case 5: deleterear();

break;

case 6: traverse();

break;

case 7: printf("\nElements in list (right -> left):\n");

struct node\* newnode;

newnode = start;

traverseopp(start);

printf(" X");

break;

case 8: exit(0);

break;

default: printf("\nError 101: Enter correct choice");

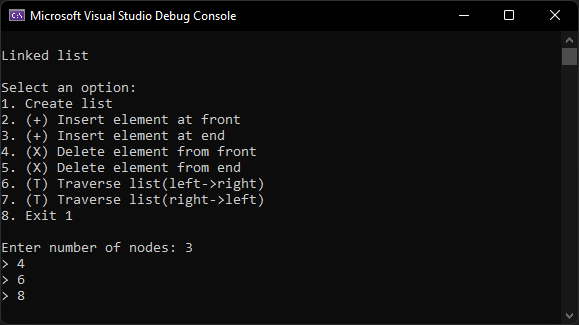
}

}

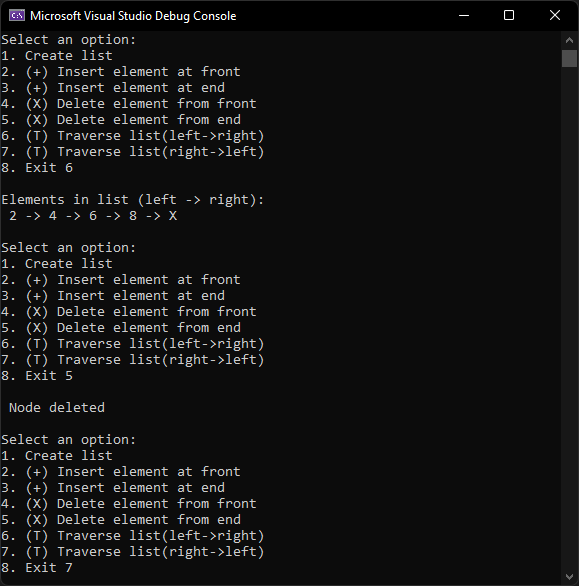
}

Output:

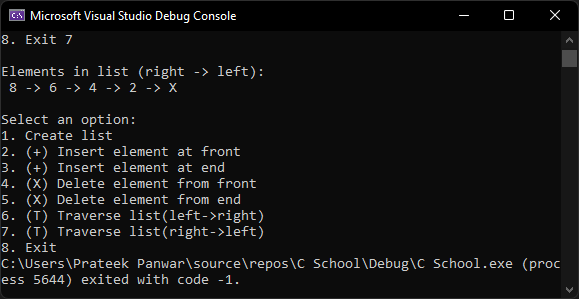
Page 1:



Page 2:



Page 3:



1. Menu driven program to implement various operations on a circular linked list.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

#include <stdlib.h>

int max = 20;

//Node stores data and it's link

struct node

{

int data;

struct node \*next;

};

struct node \*start = NULL;

int nodecount;

struct node\* getnode()

{

struct node\* newnode;

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

printf("> ");

scanf("%d", &newnode->data);

newnode->next = NULL;

return newnode;

}

void createlist()

{

if (start == NULL)

{

int n;

struct node\* newnode;

struct node\* temp;

printf("\nEnter number of nodes: ");

scanf("%d", &n);

nodecount = n;

newnode = NULL; //VS won't let me run uninitialised newnode in line 56

for (int i = 0; i < n; i++)

{

newnode = getnode();

if (start == NULL)

{

start = newnode;

}

else

{

temp = start;

while (temp->next != NULL)

{

temp = temp->next;

}

temp -> next = newnode;

}

}

newnode -> next = start;

}

else

{

printf("\nList is already created!");

}

}

void insertfront()

{

struct node\* newnode, \*last;

newnode = getnode();

if (start == NULL)

{

start = newnode;

newnode->next = start;

}

else

{

last = start;

while (last->next != start)

{

last = last->next;

}

newnode->next = start;

start = newnode;

last->next = start;

}

nodecount++;

}

void insertrear()

{

struct node\* newnode, \*temp;

newnode = getnode();

if (start == NULL)

{

start = newnode;

}

else

{

temp = start;

while (temp->next != NULL)

temp = temp->next;

temp->next = newnode;

}

nodecount++;

}

void deletefront()

{

struct node\* newnode, \*last;

if (start == NULL)

{

printf("No nodes are present");

return;

}

else

{

last = newnode = start;

while (last->next != start)

last = last->next;

start = newnode->next;

last->next = start;

free(newnode);

printf("\n Node deleted");

}

}

void deleterear()

{

struct node \*newnode, \*lastnode;

if (start == NULL)

{

printf("No nodes are present");

return;

}

else

{

newnode = start;

lastnode = start;

while (newnode -> next != NULL)

{

lastnode = newnode;

newnode = newnode->next;

}

lastnode->next = start;

free(newnode);

nodecount--;

if (nodecount == 0)

start = NULL;

printf("\n Node deleted");

}

}

void traverse()

{

struct node\* newnode;

newnode = start;

if (start == NULL)

{

printf("\nList is Empty!");

return;

}

else

{

printf("\nElements in list (left -> right):\n");

do

{

printf(" %d ->", newnode->data);

newnode = newnode->next;

} while (newnode != start);

printf(" X");

}

}

void traverseopp(struct node \*start)

{

if (start == NULL)

{

return;

}

else

{

traverseopp(start->next);

printf(" %d ->", start->data);

}

}

intMenu()

{

int ch;

ch = 0;

printf("\n\nSelect an option: \n1. Create list \n2. (+) Insert element at front \n3. (+) Insert element at end");

printf("\n4. (X) Delete element from front \n5. (X) Delete element from end");

printf("\n6. (T) Traverse list(left->right) \n7. (T) Traverse list(right->left) \n8. Exit ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

printf("\nCircular Linked list");

while (!0)

{

ch =Menu();

switch (ch)

{

case 1: createlist();

break;

case 2: insertfront();

break;

case 3: insertrear();

break;

case 4: deletefront();

break;

case 5: deleterear();

break;

case 6: traverse();

break;

case 7: printf("\nElements in list (right -> left):\n");

struct node\* newnode;

newnode = start;

traverseopp(start);

printf(" X");

break;

case 8: exit(0);

break;

default: printf("\nError 101: Enter correct choice");

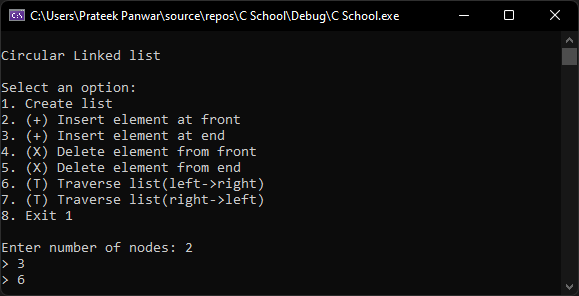
}

}

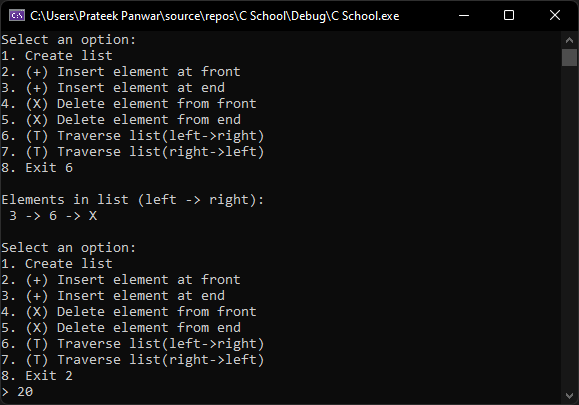
}

Output:

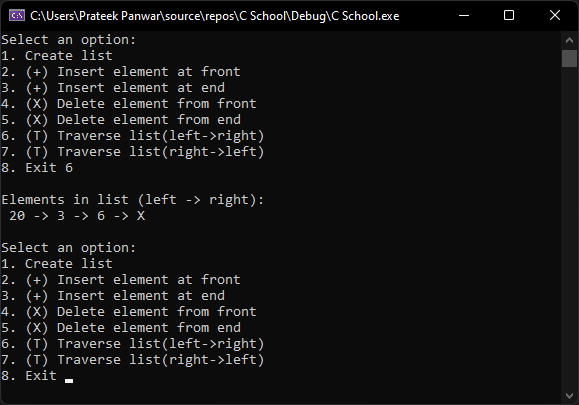
Page 1:



Page 2:



Page 3:



10. Program for implementation of Bubble sort.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

void bubblesort(int arr[], int n) //Bubble sort compares each element to rest of elem in array & replaces smaller elem with larger

{

int temp;

for (int i = 0; i < n; i++) //It checks for each pass

{

for (int j = 0; j<n-i-1; j++) //It compares each pass element to all the other elements

{

if (arr[j] > arr[j + 1]) //Swapping both elements

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main()

{

int arr[10],n;

printf("Bubble Sort");

printf("\nEnter number of elements: ");

scanf("%d", &n);

printf("\nEnter value of elements: ");

for (int i = 0; i < n; i++)

{

printf("> ");

scanf("%d", &arr[i]);

}

bubblesort(arr, n);

printf("\nSorted Array: ");

for (int i = 0; i < n; i++)

{

printf("%d ", arr[i]);

}

return 0;

}

Output:

Text

Description automatically generated

11. Program for Insertion sort.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <math.h>

#include <stdio.h>

int arr[10];

int n;

void getElem(int arr[])

{

printf("Enter total elements in list: ");

scanf("%d", &n);

//Entering elements

printf("Enter elements of sorted list: ");

for (int i = 0; i < n; i++)

{

printf("> ");

scanf("%d", &arr[i]);

}

}

int insertionSort(int arr[])

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void display(int arr[])

{

printf("Sorted elemets using quick sort: \n");

for (int i = 0; i < n; i++)

{

printf(" %d", arr[i]);

}

printf("\n");

}

int main()

{

getElem(arr);

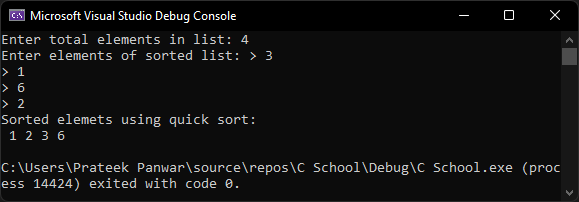
insertionSort(arr);

display(arr);

return 0;

}

Output:



12. Program for merge sort.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1 /\*Line 1: For    Visual Studio, Remove in other compilers\*/

#include <stdio.h>

int m = 0, n = 0, i, j;

int a[10], b[10];

void getElem()

{

    printf("Enter total elements in list A: ");

    scanf("%d", &m);

    printf("Enter total elements in list B: ");

    scanf("%d", &n);

    //Entering elements

    printf("Enter elements of sorted list A: ");

    for (int i = 0; i < m; i++)

    {

        printf("> ");

        scanf("%d", &a[i]);

    }

    printf("Enter elements of sorted list B: ");

    for (int i = 0; i < n; i++)

    {

        printf("> ");

        scanf("%d", &b[i]);

    }

}

void display(int c[])

{

    printf("Merged list: \n");

    for (int i = 0; i < m + n; i++)

    {

        printf("  %d", c[i]);

    }

}

void mergeSort()

{

    int c[m + n];

    int i = 0, j = 0, k = 0;

    while (i < m & j < n)

    {

        if (a[i] < b[j])

        {

            c[k] = a[i];

            k = k + 1;

            i = i + 1;

        }

        else

        {

            c[k] = b[i];

            k = k + 1;

            j = j + 1;

        }

    }

    while (i < m)

    {

        c[k] = a[i];

        k = k + 1;

        i = i + 1;

    }

    while (j < n)

    {

        c[k] = b[j];

        k = k + 1;

        j = j + 1;

    }

    display(c);

}

int main()

{

printf("Merge Sort\n");

    getElem();

    mergeSort();

    return 0;

}

Output:

Output:

