**Data Structures(3130702)**

**PRACTICAL**

**1 TO 20**

1. **Introduction to Call by Value and Call by reference and Call by address.**

**Call by value:**

#include <stdio.h>

#include<conio.h>

*void* call\_by\_value(*int* *x*)

{

    printf("\nInside call\_by\_value x = %d before adding 10.\n", *x*);

*x* += 10;

    printf("Inside call\_by\_value x = %d after adding 10.\n", *x*);

}

*int* main()

{

*int* a=10;

    printf("a = %d before function call\_by\_value.\n", a);

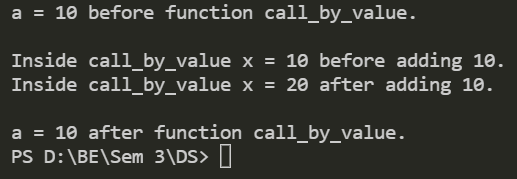
    call\_by\_value(a);

    printf("\na = %d after function call\_by\_value.\n", a);

    return 0;

}

**Output:**



**Call by reference**

#include <stdio.h>

*void* call\_by\_reference(*int* \**y*)

{

    printf("\nInside call\_by\_reference y = %d before adding 10.\n", \**y*);

    (\**y*) += 10;

    printf("Inside call\_by\_reference y = %d after adding 10.\n", \**y*);

}

*int* main()

{

*int* b=10;

    printf("b = %d before function call\_by\_reference.\n", b);

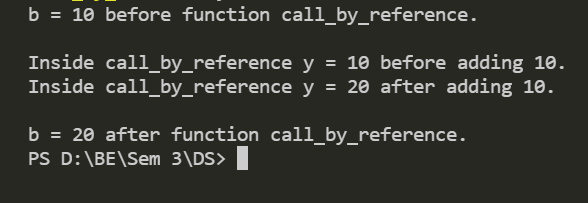
    call\_by\_reference(&b);

    printf("\nb = %d after function call\_by\_reference.\n", b);

    return 0;

}

**Output**:



1. Introduction to Dynamic Memory Allocation. DMA functions malloc(), calloc(), realloc(), free()

**Realloc()**

#include <stdio.h>

#include <stdlib.h>

*int* main() {

*int* \*ptr, i , n1, n2;

  printf("Enter size: ");

  scanf("%d", &n1);

  ptr = (*int*\*) malloc(n1 \* sizeof(*int*));

  printf("Addresses of previously allocated memory:\n");

  for(i = 0; i < n1; ++i)

    printf("%pc\n",ptr + i);

  printf("\nEnter the new size: ");

  scanf("%d", &n2);

  // rellocating the memory

  ptr = realloc(ptr, n2 \* sizeof(*int*));

  printf("Addresses of newly allocated memory:\n");

  for(i = 0; i < n2; ++i)

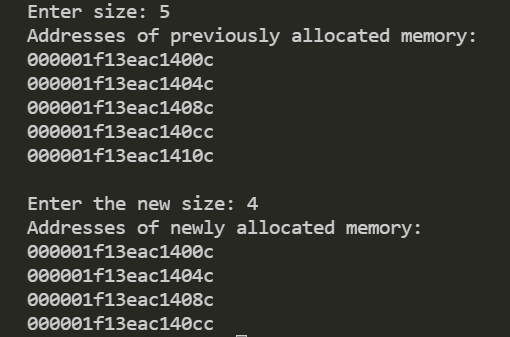
    printf("%pc\n", ptr + i);

  free(ptr);

  return 0;

}

**Output:**

****

**Malloc()**

#include <stdio.h>

#include <stdlib.h>

*int* main() {

*int* n, i, \*ptr, sum = 0;

  printf("Enter number of elements: ");

  scanf("%d", &n);

  ptr = (*int*\*) malloc(n \* sizeof(*int*));

  if(ptr == NULL) {

    printf("Error! memory not allocated.");

    exit(0);

  }

  printf("Enter elements: ");

  for(i = 0; i < n; ++i) {

    scanf("%d", ptr + i);

    sum += \*(ptr + i);

  }

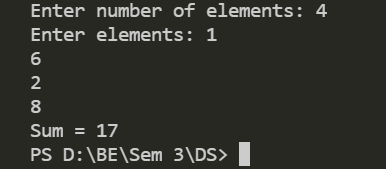
  printf("Sum = %d", sum);

  free(ptr);

  return 0;

}

**Output:**



**Calloc()**

#include <stdio.h>

#include <stdlib.h>

*int* main() {

*int* n, i, \*ptr, sum = 0;

  printf("Enter number of elements: ");

  scanf("%d", &n);

  ptr = (*int*\*) calloc(n, sizeof(*int*));

  if(ptr == NULL) {

    printf("Error! memory not allocated.");

    exit(0);

  }

  printf("Enter elements: ");

  for(i = 0; i < n; ++i) {

    scanf("%d", ptr + i);

    sum += \*(ptr + i);

  }

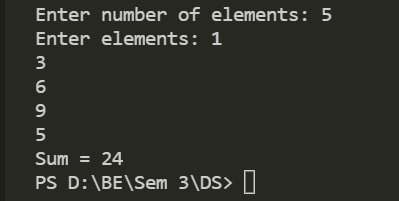
  printf("Sum = %d", sum);

  free(ptr);

  return 0;

}

**Output:**



1. Write a program of stack that performs following operations using array. (a) PUSH (b) POP (c) PEEP (d) CHANGE (e) DISPLAY

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

*struct* stack

{

*int* size;

*int* top;

*int* \*array;

};

*void* push(*struct* stack \**sp*)

{

    if(*sp*->top == *sp*->size-1)

    {

        printf("\nstack overflow");

    }

    else

    {

*int* no;

        printf("\nEnter value :");

        scanf("%d",&no);

*sp*->top = *sp*->top + 1;

*sp*->array[*sp*->top] = no;

        printf("\n%d is inserted at position %d",no,*sp*->top);

    }

}

*void* pop(*struct* stack \**sp*)

{

*int* no;

    if(*sp*->top == -1)

    {

        printf("\nstack underflow");

    }

    else

    {

        no = *sp*->array[*sp*->top];

*sp*->top = *sp*->top - 1;

        printf("\n%d is deleted from position %d",no,*sp*->top+1);

    }

}

*void* peep(*struct* stack \**sp*)

{

*int* p;

   printf("\nEnter the position : ");

   scanf("%d",&p);

   if(*sp*->top-p<=-1)

   {

      printf("\nSTACK is overflow !!!");

   }

   else

   {

      printf("\nThe Elements is : %d",*sp*->array[*sp*->top-p]);

   }

}

*void* change(*struct* stack \**sp*)

{

*int* v1,v2;

  printf("\nEnter Position for change : ");

  scanf("%d",&v1);

  printf("\nEneter the Number for change : ");

  scanf("%d",&v2);

  if(*sp*->top-v1<=-1)

  {

     printf("\nSTACK is overflow !!!");

  }

  else

  {

*sp*->array[*sp*->top-v1]=v2;

    printf("\nCHANGE successfull !!!");

  }

}

*void* display(*struct* stack \**sp*)

{

*int* i;

    for(i=0;i<=*sp*->top;i++)

    {

        printf("\n position %d : %d",i,*sp*->array[i]);

    }

}

*void* main()

{

*struct* stack \*s = (*struct* stack \*)malloc(sizeof(*struct* stack));

    s->top = -1;

    s->size = 100;

    s->array = (*int* \*)malloc(s->size \* sizeof(*int*));

*int* ch,no;

    while(ch != 6)

    {

        printf("\nTop : %d",s->top);

        printf("\n\n1. PUSH\n2. POP\n3. PEEP\n4. CHANGE\n5. DISPLAY\n6. EXIT");

        printf("\nEnter your choice : ");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

                push(s);

                break;

            case 2:

                pop(s);

                break;

            case 3:

                if(s->top == -1)

                {

                    printf("stack is empty");

                }

                else

                {

                    peep(s);

                }

                break;

            case 4:

                if(s->top == -1)

                {

                    printf("stack is empty");

                }

                else

                {

                    change(s);

                }

                break;

            case 5:

                display(s);

                break;

            default:

                printf("enter valid choice...");

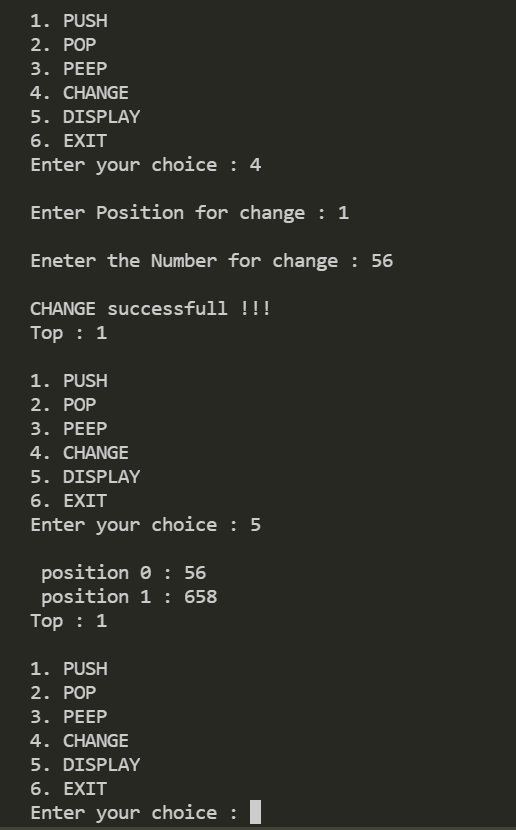
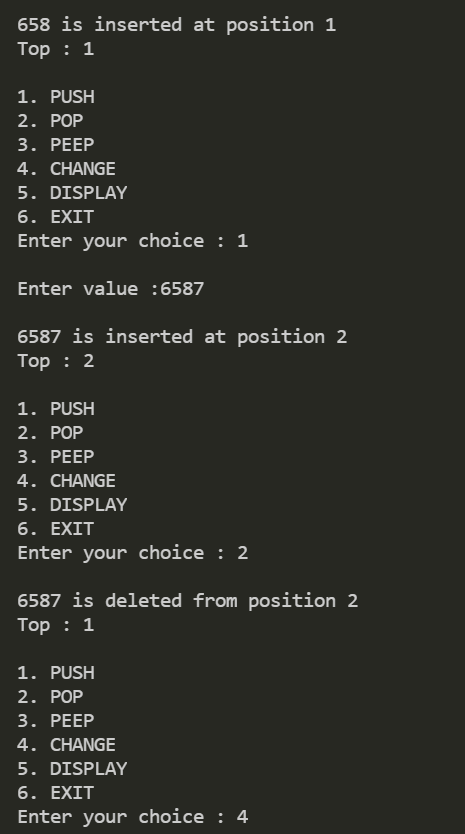
        }

    }

    free;

}

**Output:**



1. **Write a program to convert infix notation to postfix notation using stack (with parenthesis)**

#include<stdio.h>

#include<ctype.h>

*char* stack[100];

*int* top = -1;

*void* push(*char* *x*)

{

    stack[++top] = *x*;

}

*char* pop()

{

    if(top == -1)

        return -1;

    else

        return stack[top--];

}

*int* priority(*char* *x*)

{

    if(*x* == '(')

        return 0;

    if(*x* == '+' || *x* == '-')

        return 1;

    if(*x* == '\*' || *x* == '/')

        return 2;

    return 0;

}

*int* main()

{

*char* exp[100];

*char* \*e, x;

    printf("Enter the expression : ");

    scanf("%s",exp);

    printf("\n");

    e = exp;

    while(\*e != '\0')

    {

        if(isalnum(\*e))

            printf("%c ",\*e);

        else if(\*e == '(')

            push(\*e);

        else if(\*e == ')')

        {

            while((x = pop()) != '(')

                printf("%c ", x);

        }

        else

        {

            while(priority(stack[top]) >= priority(\*e))

                printf("%c ",pop());

            push(\*e);

        }

        e++;

    }

    while(top != -1)

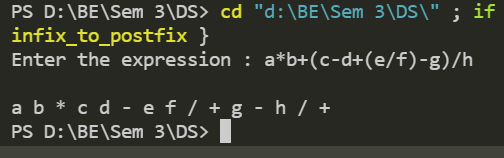
    {

        printf("%c ",pop());

    }return 0;

}

**Output:**



1. **Write a program for evaluation of postfix expression.**

#include<stdio.h>

#include<conio.h>

*int* stack[20];

*int* top = -1;

*void* push(*int* *x*)

{

    stack[++top] = *x*;

}

*int* pop()

{

    return stack[top--];

}

*int* main()

{

*char* exp[20];

*char* \*e;

*int* n1,n2,n3,num;

    printf("Enter the expression :: ");

    scanf("%s",exp);

    e = exp;

    while(\*e != '\0')

    {

        if(isdigit(\*e))

        {

            num = \*e - 48;

            push(num);

        }

        else

        {

            n1 = pop();

            n2 = pop();

            switch(\*e)

            {

            case '+':

            {

                n3 = n1 + n2;

                break;

            }

            case '-':

            {

                n3 = n2 - n1;

                break;

            }

            case '\*':

            {

                n3 = n1 \* n2;

                break;

            }

            case '/':

            {

                n3 = n2 / n1;

                break;

            }

            }

            push(n3);

        }

        e++;

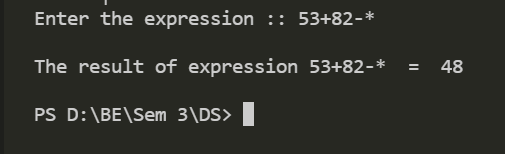
    }

    printf("\nThe result of expression %s  =  %d\n\n",exp,pop());

    return 0;

}

**Output:**



1. **Write a program to implement QUEUE using array that performs following operations. (a) INSERT (b) DELETE (c) DISPLAY:**

#include<stdio.h>

#include<conio.h>

#define MAX 10

*void* insert(*int*);

*int* del();

*int* queue[MAX], rear=0, front=0;

*void* display();

*int* main()

{

*char* ch , a='y';

*int* choice, token;

    printf("1.Insert");

    printf("\n2.Delete");

    printf("\n3.show or display");

    do

    {

        printf("\nEnter your choice for the operation: ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

                insert(token);

                display();

            break;

            case 2:

                token=del();

                printf("\nThe element deleted is %d",token);

                display();

            break;

            case 3:

                display();

                break;

            default:

                printf("Wrong choice");

                break;

        }

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

        ch=getch();

    }while(ch=='y'||ch=='Y');

    getch();

}

*void* display()

{

*int* i;

    printf("\nThe queue elements are:");

    for(i=rear;i<front;i++)

    {

        printf("%d ",queue[i]);

    }

}

*void* insert(*int* *token*)

{

*char* a;

    if(rear==MAX)

    {

        printf("\nQueue full");

        return;

    }

    do

    {

        printf("\nEnter the element to be inserted:");

        scanf("%d",&*token*);

        queue[front]=*token*;

        front=front+1;

        printf("\n\ndo you want to continue insertion Y/N");

        a=getch();

    }

    while(a=='y');

}

*int* del()

{

*int* t;

    if(front==rear)

    {

        printf("\nQueue empty");

        return 0;

    }

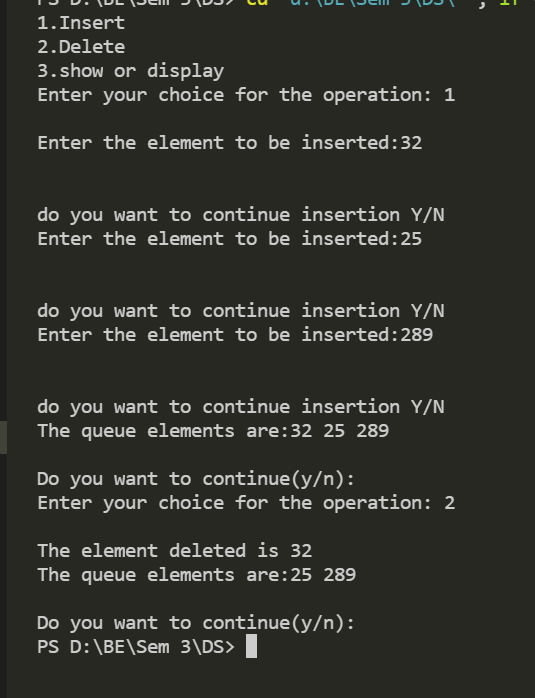
    rear=rear+1;

    t=queue[rear-1];

    return t;

}

**Output**:



1. **Write a program to implement Circular Queue using array that performs following operations.. (a) INSERT (b) DELETE (c) DISPLAY**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define max 10

*int* q[10],front=0,rear=-1;

*void* main()

{

*int* ch;

*void* insert();

*void* delet();

*void* display();

    printf("\n---Circular Queue operations---\n");

    printf("1.insert\n2.delete\n3.display\n4.exit\n");

    while(1)

    {

        printf("Enter your choice:");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

                insert();

                break;

            case 2:

                delet();

                break;

            case 3:

                display();

                break;

            case 4:

                exit(0);

            default:

                printf("Invalid option\n");

        }

    }

}

*void* insert()

{

*int* x;

    if((front==0&&rear==max-1)||(front>0&&rear==front-1))

        printf("Queue is overflow\n");

    else

    {

        printf("Enter element to be insert:");

        scanf("%d",&x);

        if(rear==max-1&&front>0)

        {

            rear=0;

            q[rear]=x;

        }

        else

        {

            if((front==0&&rear==-1)||(rear!=front-1))

            q[++rear]=x;

        }

    }

}

*void* delet()

{

*int* a;

    if((front==0)&&(rear==-1))

    {

        printf("Queue is underflow\n");

        getch();

        exit(0);

    }

    if(front==rear)

    {

        a=q[front];

        rear=-1;

        front=0;

    }

    else if(front==max-1)

    {

        a=q[front];

        front=0;

    }

    else

        a=q[front++];

    printf("Deleted element is:%d\n",a);

}

*void* display()

{

*int* i,j;

    if(front==0&&rear==-1)

    {

        printf("Queue is underflow\n");

        getch();

        exit(0);

    }

    if(front>rear)

    {

        for(i=0;i<=rear;i++)

            printf("\t%d",q[i]);

        for(j=front;j<=max-1;j++)

        printf("\t%d",q[j]);

        printf("\nrear is at %d\n",q[rear]);

        printf("\nfront is at %d\n",q[front]);

    }

    else

    {

        for(i=front;i<=rear;i++)

        {

            printf("\t%d",q[i]);

        }

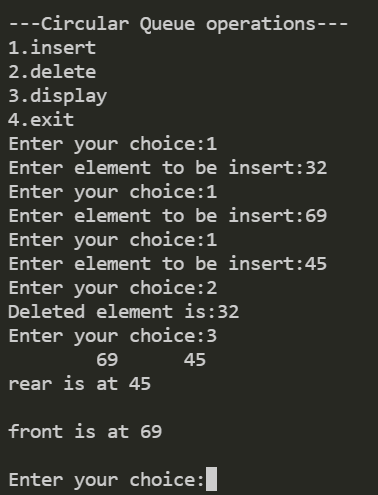
        printf("\nrear is at %d\n",q[rear]); printf("\nfront is at %d\n",q[front]);

    }

    printf("\n");

}

**Output:**



1. **Write a menu driven program to implement following operations on the singly linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node after a given element (value) (d) Insert a node at a given position (e) Delete a first node of the linked list. (f) Delete a last node of the linked list. (g) Delete a node at specified position. (h) Delete a specified node (based on value).**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

*struct* node

{

*int* data;

*struct* node \*link;

};

*struct* node \*header, \*ptr,\*ptr1, \*temp;

*void* insert\_front();

*void* insert\_end();

*void* insert\_any();

*void* delete\_front();

*void* delete\_end();

*void* delete\_any();

*void* display();

*void* main()

{

*int* choice;

*int* count = 1;

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

    header->data = NULL;

    header->link = NULL;

    while(count == 1)

    {

        printf("\n1. Insert at front\n");

        printf("\n2. Insert at end\n");

        printf("\n3. Insert at any position\n");

        printf("\n4. Delete from front\n");

        printf("\n5. Delete from end\n");

        printf("\n6. Delete from anywhere\n");

        printf("\n7. Display linked list\n");

        printf("\nEnter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                insert\_front();

                break;

            case 2:

                insert\_end();

                break;

            case 3:

                insert\_any();

                break;

            case 4:

                delete\_front();

                break;

            case 5:

                delete\_end();

                break;

            case 6:

                delete\_any();

                break;

            case 7:

                display(); break;

        }

        printf("\n\nDo you want to continue? (1 / 0): ");

        scanf("%d", &count);

    }

    getch();

}

*void* insert\_front()

{

*int* data\_value;

    printf("\nEnter data of the node: ");

    scanf("%d", &data\_value);

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

    temp->data = data\_value;

    temp->link = header->link;

    header->link = temp;

}

*void* insert\_end()

{

*int* data\_value;

    printf("\nEnter data of the node: ");

    scanf("%d", &data\_value);

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

    ptr = header;

    while(ptr->link != NULL)

    {

        ptr = ptr->link;

    }

    temp->data = data\_value;

    temp->link = ptr->link;

    ptr->link = temp;

}

*void* insert\_any()

{

*int* data\_value, key;

    printf("\nEnter data of the node: ");

    scanf("%d", &data\_value);

    printf("\nEnter data of the node after which new node is to be inserted: ");

    scanf("%d", &key);

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

    ptr = header;

    while(ptr->link != NULL &&ptr->data != key)

    {

        ptr = ptr->link;

    }

    if(ptr->data == key)

    {

        temp->data = data\_value;

        temp->link = ptr->link;

        ptr->link = temp;

    }

    else

    {

        printf("\nValue %d not found\n",key);

    }

}

*void* delete\_front()

{

    if(header->link == NULL)

    {

        printf("\nEmpty Linked List. Deletion not possible.\n");

    }

    else

    {

        ptr = header->link;

        header->link= ptr->link;

        free(ptr);

        printf("\nNode deleted from the front.\n");

    }

}

*void* delete\_end()

{

    if(header->link == NULL)

    {

        printf("\nEmpty Linked List. Deletion not possible.\n");

    }

    else

    {

        ptr = header;

        while(ptr->link != NULL)

        {

            ptr1 = ptr;

            ptr = ptr->link;

        }

        ptr1->link = ptr->link;

        free(ptr);

        printf("\nNode deleted from the end.\n");

    }

}

*void* delete\_any()

{

*int* key;

    if(header->link == NULL)

    {

        printf("\nEmpty Linked List. Deletion not possible.\n");

    }

    else

    {

        printf("\nEnter the data of the node to be deleted: ");

        scanf("%d", &key);

        ptr = header;

        while((ptr->link != NULL) && (ptr->data != key))

        {

            ptr1 = ptr;

            ptr = ptr->link;

        }

        if(ptr->data == key)

        {

            ptr1->link = ptr->link;

            free(ptr);

            printf("\nNode with data %d deleted.\n", key);

        }

        else

        {

            printf("\nValue %d not found. Deletion not possible.\n", key);

        }

    }

}

*void* display()

{

    printf("\nContents of the linked list are: \n");

    ptr = header;

    while(ptr->link != NULL)

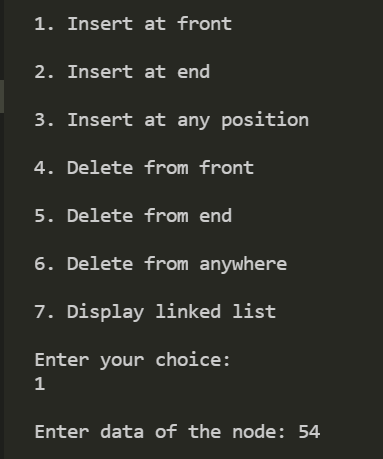
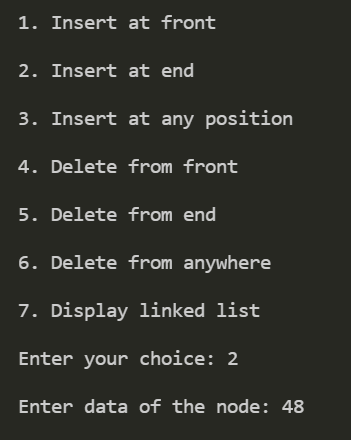
    {

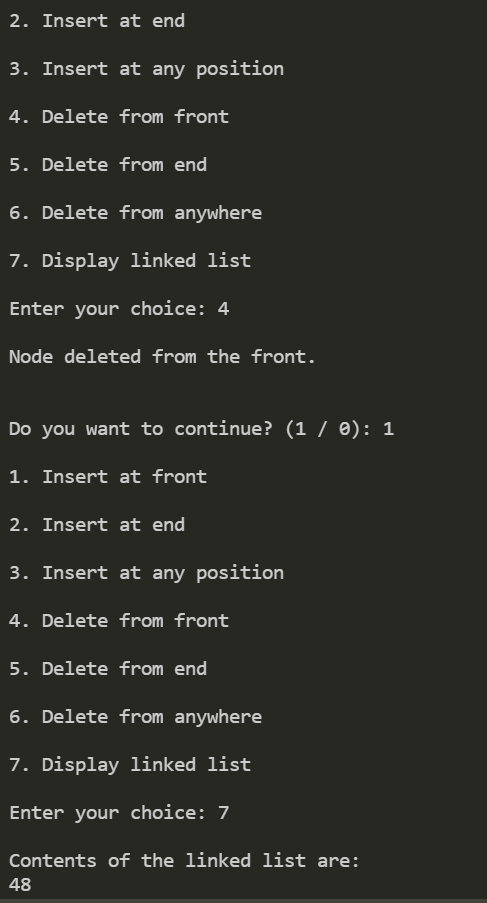
        ptr = ptr->link;

        printf("%d ", ptr->data);

    }

}

**Output:**



1. **Write a program to implement stack using linked list**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

*struct* Node

{

*int* Data;

*struct* Node \*next;

}\*top;

*void* popStack()

{

*struct* Node \*temp, \*var=top;

    if(var==top)

    {

        top = top->next;

        free(var);

    }

    else

    printf("\nStack Empty");

}

*void* push(*int* *value*)

{

*struct* Node \*temp;

    temp=(*struct* Node \*)malloc(sizeof(*struct* Node));

    temp->Data=*value*;

    if (top == NULL)

    {

        top=temp;

        top->next=NULL;

    }

    else

    {

        temp->next=top;

        top=temp;

    }

}

*void* display()

{

*struct* Node \*var=top;

    if(var!=NULL)

    {

        printf("\nElements are as:\n");

        while(var!=NULL)

        {

            printf("\t%d\n",var->Data);

            var=var->next;

        }

        printf("\n");

    }

    else

    printf("\nStack is Empty");

}

*void* main()

{

*int* i=0,value;

    top=NULL;

*struct* Node \*temp;

    printf(" \n1. Push to stack");

    printf(" \n2. Pop from Stack");

    printf(" \n3. Display data of Stack");

    printf(" \n4. Exit\n");

    while(1)

    {

        printf(" \nChoose Option: "); scanf("%d",&i);

        switch(i)

        {

            case 1:

                printf("\nEnter a valueber to push into Stack: ");

                scanf("%d",&value);

                push(value);

                display();

                break;

            case 2:

                popStack();

                display(); break;

            case 3:

                display(); break;

            case 4:

                while(top!=NULL)

                {

                    temp = top->next;

                    free(top);

                    top=temp;

                }

                exit(0);

            default:

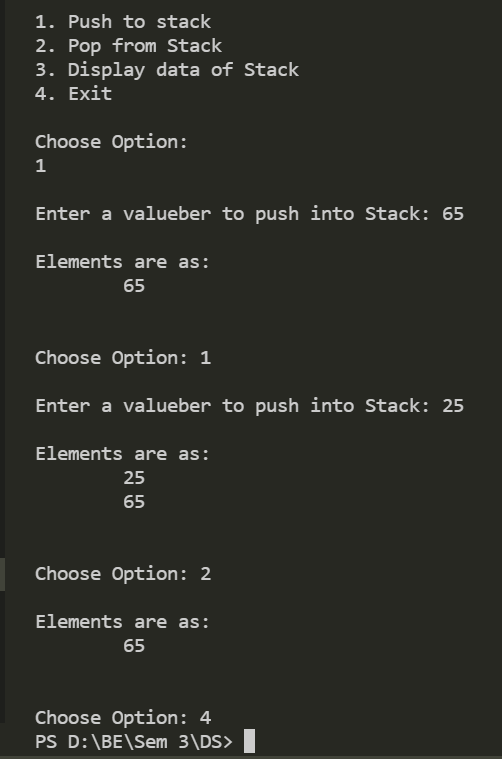
                printf("\nwrong choice for operation");

        }

    }

}

**Output:**



1. **Write a program to implement queue using linked list.**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

typedef *struct* node

{

*int* data;

*struct* node \*link;

} NODE;

*void* Insert(*int*);

*int* Delete();

*void* Display();

NODE \*front, \*rear;

/\* Global Declarations \*/

*void* main()

{

    /\* Main Program \*/

*int* opn, elem;

    front = rear = NULL;

        printf("\n --- Linked List Implementation of QUEUE Operations --- \n\n");

    do

    {

        printf("\n \n 1-Insert,\n 2-Delete,\n3-Display,\n4-Exit\n");

        printf("\n Your option : ");

        scanf("%d", &opn);

        switch (opn)

        {

            case 1:

                printf("\n\n Enter the Element to be Inserted :");

                scanf("%d", &elem);

                Insert(elem);

                break;

            case 2:

                elem = Delete();

                if (elem != -1)

                    printf(" Deleted Node(From Front)with the Data: %d\n", elem);

                break;

            case 3:

                printf("Your Queue:\n");

                Display();

                break;

            case 4:

                printf("\n\n Terminating \n\n");

                break;

            default:

                printf("\n\nInvalid Option !!! Try Again !! \n\n");

                break;

        }

    getch();

    } while (opn != 4);

}

*void* Insert(*int* *info*)

{

    NODE \*temp;

    temp = (NODE \*) malloc(sizeof(NODE));

    if (temp == NULL)

    printf(" Out of Memory !! Overflow !!!");

    else

    {

        temp->data = *info*;

        temp->link = NULL;

        if (front == NULL)

        {

            front = rear = temp;

        } /\* First Node? \*/

        else

        {

            rear->link = temp;

            rear = temp;

        } /\* Insert End \*/

        printf(" Node has been inserted at End Successfully !!");

    }

}

*int* Delete()

{

*int* info;

    NODE \*t;

    if (front == NULL)

    {

        printf(" Underflow!!!"); return -1;

    }

    else {

        t = front;

        info = front->data;

        if (front == rear)

            rear = NULL;

        front = front->link;

        t->link = NULL;

        free(t);

        return (info);

    }

}

*void* Display()

{

    NODE \*t;

    if (front == NULL)

        printf("Empty Queue\n");

    else

    {

        t = front;

        printf("Front->");

        while (t)

        {

            printf("[%d]->", t->data);

            t = t->link;

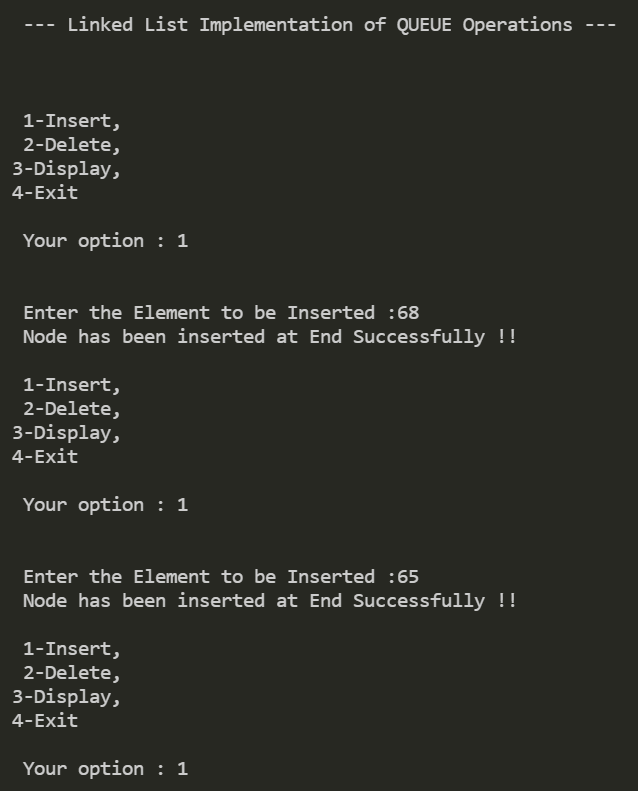
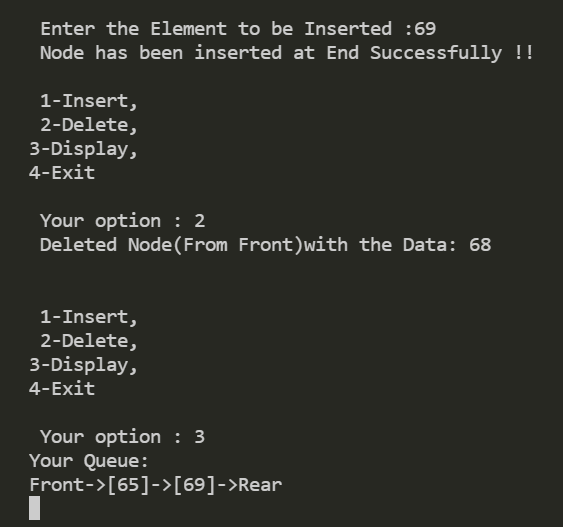
        }

        printf("Rear\n");

    }

}

**Output:**



1. **Write a program to implement following operations on the circular linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node at specified position. (d) Delete a first node of the linked list. (e) Delete a last node of the linked list. (f) Delete a node at specified position.**

#include <stdio.h>

#include<conio.h>

#include <stdlib.h>

*struct* node

{

*int* data;

*struct* node \*link;

};*struct* node \*head = NULL, \*x, \*y, \*z;

*void* create();

*void* ins\_at\_beg();

*void* ins\_at\_pos();

*void* del\_at\_beg();

*void* del\_at\_pos();

*void* traverse();

*void* main()

{

*int* ch;

    printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at specified position \n 4.Deletion at beginning \n 5.Deletion at remaining \n 6.display \n 7. exit \n");

    while (1)

    {

        printf("\n Enter your choice:"); scanf("%d", &ch);

        switch(ch)

        {

            case 1:

                create();

                break;

            case 2:

                ins\_at\_beg();

                break;

            case 3:

                ins\_at\_pos();

                break;

            case 4:

                del\_at\_beg();

                break;

            case 5:

                del\_at\_pos();

                break;

            case 6:

                traverse();

            case 7:

                exit(0);

            default:

                printf("\nWrong choice");

        }

    }

}

/\*Function to create a new circular linked list\*/

*void* create()

{

*int* c;

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

    printf("\n Enter the data:");

    scanf("%d", &x->data); x->link = x;

    head = x;

    printf("\n If you wish to continue press 1 otherwise 0:");

    scanf("%d", &c);

    while (c != 0)

    {

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

        printf("\n Enter the data:");

        scanf("%d", &y->data); x->link = y;

        y->link = head;

        x = y;

        printf("\n If you wish to continue press 1 otherwise 0:");

        scanf("%d", &c);

    }

}

/\*Function to insert an element at the begining of the list\*/

*void* ins\_at\_beg()

{

    x = head;

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

    printf("\n Enter the data:");

    scanf("%d", &y->data);

    while (x->link != head)

    {

         x = x->link;

    }

    x->link = y;

    y->link = head; head = y;

}

/\*Function to insert an element at any position the list\*/

*void* ins\_at\_pos()

{

*struct* node \*ptr;

*int* c = 1, pos, count = 1;

    y = (*struct* node\*)malloc(sizeof(*struct* node)); if (head == NULL)

    {

        printf("cannot enter an element at this place");

    }

    printf("\n Enter the data:");

    scanf("%d", &y->data);

    printf("\n Enter the position to be inserted:");

    scanf("%d", &pos);

    x = head;

    ptr = head;

    while (ptr->link != head)

    {

        count++; ptr = ptr->link;

    }

    count++;

    if (pos> count)

    {

        printf("OUT OF BOUND");

        return;

    }

    while (c <pos)

    {

        z = x;

        x = x->link;

        c++;

    }

    y->link = x;

    z->link = y;

}

    /\*Function to delete an element at any begining of the list\*/

*void* del\_at\_beg()

    {

        if (head == NULL) printf("\n List is empty");

        else

        {

            x = head;

            y = head;

            while (x->link != head)

            {

                x = x->link;

            }

            head = y->link;

            x->link = head;

            free(y);

        }

}

/\*Function to delete an element at any position the list\*/

*void* del\_at\_pos()

{

    if (head == NULL) printf("\n List is empty");

    else

    {

*int* c = 1, pos;

        printf("\n Enter the position to be deleted:");

        scanf("%d", &pos);

        x = head;

        while (c <pos)

        {

            c++;

        }

        y = x;

        x = x->link;

        y->link = x->link;

        free(x);

    }

}

/\*Function to display the elements in the list\*/

*void* traverse()

{

    if (head == NULL)

    printf("\n List is empty");

    else

    {

        x = head;

        while (x->link != head)

        {

            printf("%d->", x->data);

            x = x->link;

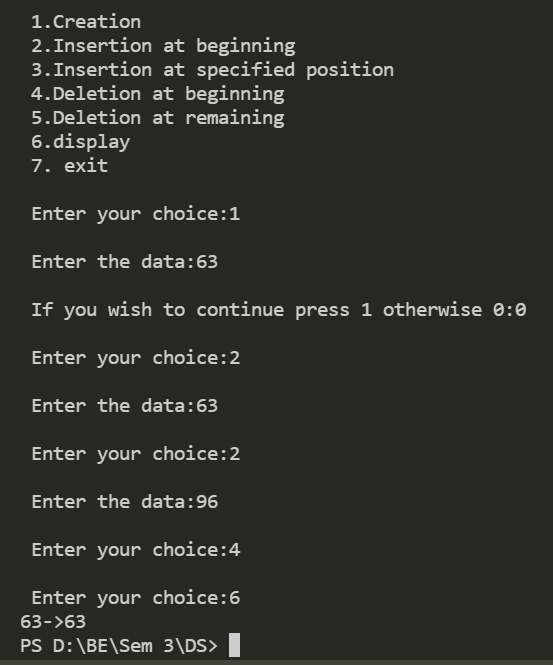
        }

        printf("%d", x->data);

    }

}

**Output:**



1. **Write a program which create binary search tree.**

**&**

1. **Implement recursive tree traversing methods for inorder, preorder and postorder traversal.**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

*struct* btnode

{

*int* value; *struct* btnode \*l; *struct* btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

*void* delete1();

*void* insert();

*void* delete();

*void* inorder(*struct* btnode \**t*);

*void* create();

*void* search(*struct* btnode \**t*);

*void* preorder(*struct* btnode \**t*);

*void* postorder(*struct* btnode \**t*);

*void* search1(*struct* btnode \**t*,*int* *data*);

*int* smallest(*struct* btnode \**t*);

*int* largest(*struct* btnode \**t*);

*int* flag = 1;

*void* main()

{

*int* ch;

    printf("\nOPERATIONS ---");

    printf("\n1 - Insert an element into tree\n");

    printf("2 - Delete an element from the tree\n");

     printf("3 - Inorder Traversal\n");

    printf("4 - Preorder Traversal\n");

    printf("5 - Postorder Traversal\n");

    printf("6 - Exit\n");

    while(1)

    {

        printf("\nEnter your choice : "); scanf("%d", &ch);

        switch (ch)

        {

            case 1:

                insert();

                break;

            case 2:

                delete();

                break;

            case 3:

                inorder(root);

                break;

            case 4:

                preorder(root);

                break;

            case 5:

                postorder(root);

                break;

            case 6:

                exit(0);

            default :

                printf("Wrong choice, Please enter correct choice ");

                break;

        }

    }

    getch();

}

/\* To insert a node in the tree \*/

*void* insert()

{

    create();

    if (root == NULL)

    {

        root = temp;

    }

    else

    {

        search(root);

    }

}

/\* To create a node \*/

*void* create()

{

*int* data;

    printf("Enter data of node to be inserted : ");

    scanf("%d", &data);

    temp = (*struct* btnode \*)malloc(1\*sizeof(*struct* btnode));

    temp->value = data;

    temp->l = temp->r = NULL;

}

/\* Function to search the appropriate position to insert the new node \*/

*void* search(*struct* btnode \**t*)

{

    if ((temp->value > *t*->value) && (*t*->r != NULL)) /\* value more than root node value insert at right \*/

        search(*t*->r);

    else if ((temp->value > *t*->value) && (*t*->r == NULL))

*t*->r = temp;

    else if ((temp->value < *t*->value) && (*t*->l != NULL))    /\* value less than root node value insert at left \*/

        search(*t*->l);

    else if ((temp->value < *t*->value) && (*t*->l == NULL))

*t*->l = temp;

}

/\* recursive function to perform inorder traversal of tree \*/

*void* inorder(*struct* btnode \**t*)

{

    if (root == NULL)

    {

        printf("No elements in a tree to display"); return;

    }

    if (*t*->l != NULL)

        inorder(*t*->l);

    printf("%d -> ", *t*->value);

    if (*t*->r != NULL)

        inorder(*t*->r);

}

/\* To check for the deleted node \*/

*void* delete()

{

*int* data;

    if (root == NULL)

    {

        printf("No elements in a tree to delete");

        return;

    }

    printf("Enter the data to be deleted : ");

    scanf("%d", &data);

    t1 = root;

    t2 = root;

    search1(root, data);

}

    /\* To find the preorder traversal \*/

*void* preorder(*struct* btnode \**t*)

{

    if (root == NULL)

    {

        printf("No elements in a tree to display"); return;

    }

    printf("%d -> ", *t*->value);

    if (*t*->l != NULL)

        preorder(*t*->l);

    if (*t*->r != NULL)

        preorder(*t*->r);

}

/\* To find the postorder traversal \*/

*void* postorder(*struct* btnode \**t*)

{

    if (root == NULL)

    {

        printf("No elements in a tree to display ");

        return;

    }

    if (*t*->l != NULL)

        postorder(*t*->l);

    if (*t*->r != NULL)

        postorder(*t*->r);

    printf("%d -> ", *t*->value);

}

/\* Search for the appropriate position to insert the new node \*/

*void* search1(*struct* btnode \**t*, *int* *data*)

{

    if ((*data*>*t*->value))

    {

        t1 = *t*;

        search1(*t*->r, *data*);

    }

    else if ((*data* < *t*->value))

    {

        t1 = *t*;

        search1(*t*->l, *data*);

    }

    else if ((*data*==*t*->value))

    {

        delete1(*t*);

    }

}

/\* To delete a node \*/

*void* delete1(*struct* btnode \**t*)

{

*int* k;

    /\* To delete leaf node \*/

    if ((*t*->l == NULL) && (*t*->r == NULL))

    {

        if (t1->l == *t*)

        {

            t1->l = NULL;

        }

        else

        {

            t1->r = NULL;

        }

*t* = NULL;

        free(*t*);

        return;

    }

/\* To delete node having one left hand child \*/

    else if ((*t*->r == NULL))

    {

        if (t1 == *t*)

        {

            root = *t*->l;

            t1 = root;

        }

        else if (t1->l == *t*)

        {

            t1->l = *t*->l;

        }

        else

        {

            t1->r = *t*->l;

        }

*t* = NULL;

        free(*t*);

        return;

    }

    /\* To delete node having right hand child \*/

    else if (*t*->l == NULL)

    {

        if (t1 == *t*)

        {

            root = *t*->r;

            t1 = root;

        }

        else if (t1->r == *t*)

            t1->r = *t*->r;

        else

            t1->l = *t*->r;

*t* == NULL;

        free(*t*);

        return;

    }

    /\* To delete node having two child \*/

    else if ((*t*->l != NULL) && (*t*->r != NULL))

    {

        t2 = root;

        if (*t*->r != NULL)

        {

            k = smallest(*t*->r);

            flag = 1;

        }

        else

        {

            k =largest(*t*->l);

            flag = 2;

        }

            search1(root, k);

*t*->value = k;

    }

}

/\* To find the smallest element in the right sub tree \*/

*int* smallest(*struct* btnode \**t*)

{

    t2 = *t*;

    if (*t*->l != NULL)

    {

        t2 = *t*;

        return(smallest(*t*->l));

    }

    else

        return (*t*->value);

}

/\* To find the largest element in the left sub tree \*/

*int* largest(*struct* btnode \**t*)

{

    if (*t*->r != NULL)

    {

        t2 = *t*;

        return(largest(*t*->r));

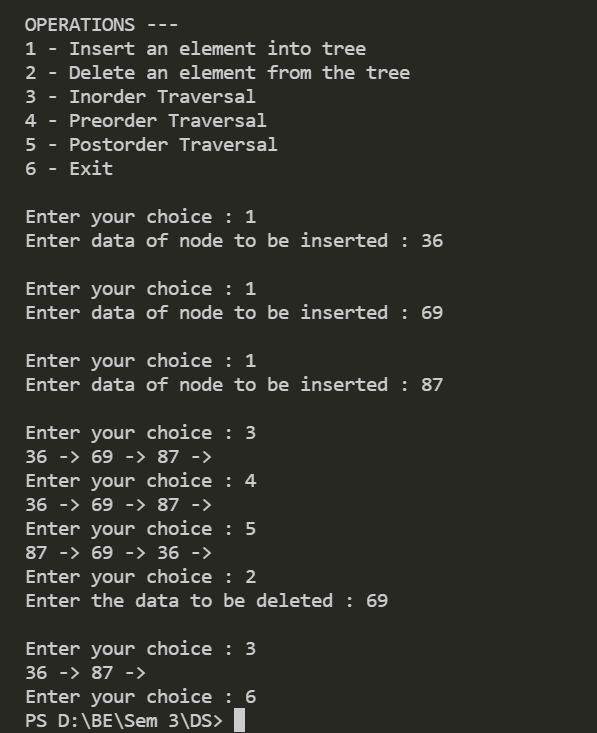
    }

    else

    return(*t*->value);

}

**Output:**



1. **Write a program to implement Selection Sort:**

#include <stdio.h>

#include<conio.h>

*void* selection\_sort();

*void* print\_array();

*int* a[30], n;

*void* main()

{

*int* i;

    printf("\nEnter size of an array: ");

    scanf("%d", &n);

    printf("\nEnter elements of an array:\n");

    for(i=0; i<n; i++)

        scanf("%d", &a[i]);

    printf("\n\n\t------- before sort :");

    print\_array();

    selection\_sort();

    printf("\n\n\t------- after sort : ");

    print\_array();

    getch();

}

*void* selection\_sort()

{

*int* i, j, min, temp;

    for (i=0; i<n; i++)

    {

        min = i;

        for (j=i+1; j<n; j++)

        {

            if (a[j] < a[min])

                min = j;

        }

        temp = a[i];

        a[i] = a[min];

        a[min] = temp;

    }

}

*void* print\_array()

{

*int* i;

    for(i=0;i<n;i++)

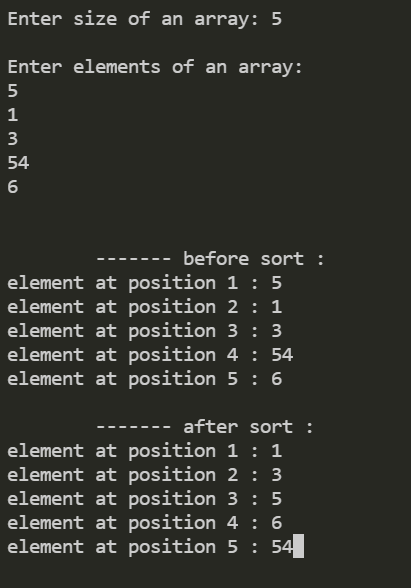
    {

        printf("\nelement at position %d : %d",i+1,a[i]);

    }

}

**Output:**



1. **Write a program to implement Bubble Sort and Insertion Sort.:**

**Bubble sort:**

#include<stdio.h>

#include<conio.h>

*void* bubble\_sort(*int* \**a*, *int* *n*)

{

*int* i,j,temp;

    for(i=0; i<*n*-1; i++)

    {

        for(j=0; j<*n*-1-i; j++)

        {

            if(*a*[j]>*a*[j+1])

            {

                temp = *a*[j];

*a*[j] = *a*[j+1];

*a*[j+1] = temp;

            }

        }

    }

}

*void* print\_array(*int* \**a*, *int* *n*)

{

*int* i;

    for(i=0;i<*n*;i++)

    {

        printf("\nelement at position %d : %d",i+1,*a*[i]);

    }

}

*void* main()

{

*int* n,i;

*int* a[n];

    printf("\n\t------- bubble sorting method   \n");

    printf("\nenter the size : ");

    scanf("%d",&n);

    printf("\n");

    for(i=0; i<n; i++)

    {

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

        scanf("%d", &a[i]);

    }

    printf("\n\n\t------- before sort :");

    print\_array(a,n);

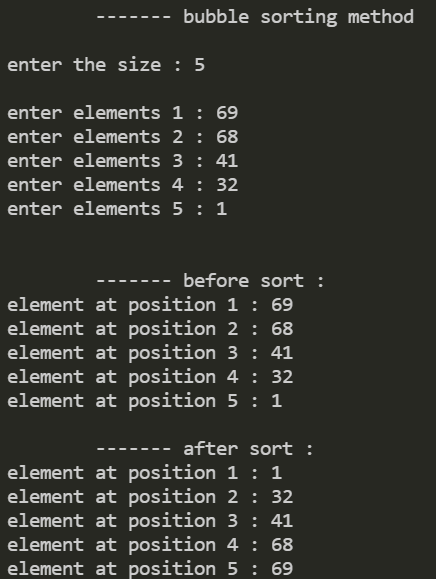
    bubble\_sort(a,n);

    printf("\n\n\t------- after sort : ");

    print\_array(a,n);

}

**Output:**



**Insertion sort:**

#include<stdio.h>

*void* InsertionSort(*int* *a*[], *int* *n*)

{

*int* j, p;

*int* tmp;

    for(p = 1; p < *n*; p++)

    {

        tmp = *a*[p];

        for(j = p; j > 0 && *a*[j-1] > tmp; j--)

*a*[j] = *a*[j-1];

*a*[j] = tmp;

    }

}

*void* print\_array(*int* *arr*[], *int* *n*)

{

*int* i;

    for(i=0;i<*n*;i++)

    {

        printf("\nelement at position %d : %d",i+1,*arr*[i]);

    }

}

*int* main()

{

*int* i, n, a[10];

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

    scanf("%d",&n);

    printf("Enter the elements :: ");

    for(i = 0; i < n; i++)

    {

        scanf("%d",&a[i]);

    }

    printf("\n\n\t------- before sort :");

    print\_array(a,n);

    InsertionSort(a,n);

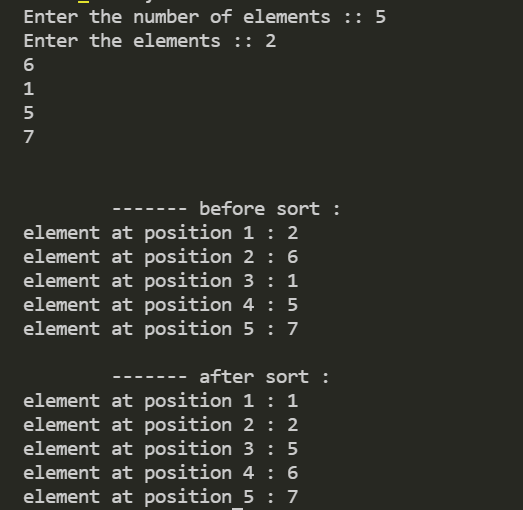
    printf("\n\n\t------- after sort : ");

    print\_array(a,n);

    return 0;

}

**Output:**



1. **Write a program to implement Merge Sort**

#include<stdio.h>

#include<conio.h>

*void* merge(*int* [],*int* ,*int* ,*int* );

*void* part(*int* [],*int* ,*int* );

*void* print\_array(*int* [], *int* );

*void* main()

{

*int* arr[30];

*int* i,size;

    printf("\n\t------- Merge sorting method    \n\n");

    printf("\nEnter total no. of elements : ");

    scanf("%d",&size);

    for(i=0; i<size; i++)

    {

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

        scanf("%d",&arr[i]);

    }

    printf("\n\n\t------- before sort :");

    print\_array(arr,size);

    part(arr,0,size-1);

    printf("\n\n\t------- after sort : ");

    print\_array(arr,size);

    getch();

}

*void* part(*int* *arr*[],*int* *min*,*int* *max*)

{

*int* mid; if(*min*<*max*)

    {

        mid=(*min*+*max*)/2;

        part(*arr*,*min*,mid); part(*arr*,mid+1,*max*); merge(*arr*,*min*,mid,*max*);

    }

}

*void* merge(*int* *arr*[],*int* *min*,*int* *mid*,*int* *max*)

{

*int* tmp[30]; *int* i,j,k,m;

    j=*min*;

    m=*mid*+1;

    for(i=*min*; j<=*mid* && m<=*max* ; i++)

    {

        if(*arr*[j]<=*arr*[m])

        {

        tmp[i]=*arr*[j]; j++;

        }

        else

        {

        tmp[i]=*arr*[m]; m++;

        }

    }

    if(j>*mid*)

    {

        for(k=m; k<=*max*; k++)

        {

         tmp[i]=*arr*[k]; i++;

        }

    }

    else

    {

        for(k=j; k<=*mid*; k++)

        {

        tmp[i]=*arr*[k]; i++;

        }

    }

    for(k=*min*; k<=*max*; k++)

*arr*[k]=tmp[k];

}

*void* print\_array(*int* *arr*[], *int* *n*)

{

*int* i;

    for(i=0;i<*n*;i++)

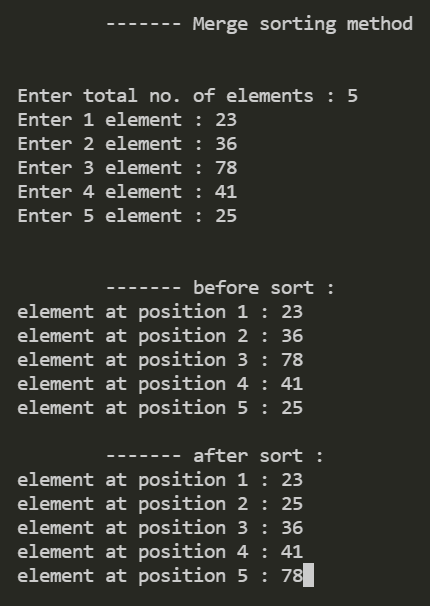
    {

        printf("\nelement at position %d : %d",i+1,*arr*[i]);

    }

}

**Output:**



1. **Write a program to implement Quick Sort**

#include<stdio.h>

#include<conio.h>

*void* quicksort(*int* [10],*int*,*int*);

*void* print\_array(*int* [], *int* );

*void* main()

{

*int* x[20],size,i;

    printf("\n\t------- Quick sorting method    \n\n");

    printf("\nEnter size of the array: ");

    scanf("%d",&size);

    for(i=0; i<size; i++)

    {

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

        scanf("%d",&x[i]);

    }

    printf("\n\n\t------- before sort :");

    print\_array(x,size);

    quicksort(x,0,size-1);

    printf("\n\n\t------- after sort : ");

    print\_array(x,size);

    getch();

}

*void* quicksort(*int* *x*[10],*int* *first*,*int* *last*)

{

*int* pivot,j,temp,i;

    if(*first*<*last*)

    {

        pivot=*first*;

        i=*first*;

        j=*last*;

        while(i<j)

        {

            while(*x*[i]<=*x*[pivot]&&i<*last*)

            i++;

            while(*x*[j]>*x*[pivot])

            j--;

            if(i<j)

            {

                temp=*x*[i];

*x*[i]=*x*[j];

*x*[j]=temp;

            }

        }

        temp=*x*[pivot];

*x*[pivot]=*x*[j];

*x*[j]=temp;

        quicksort(*x*,*first*,j-1);

        quicksort(*x*,j+1,*last*);

    }

}

*void* print\_array(*int* *arr*[], *int* *n*)

{

*int* i;

    for(i=0;i<*n*;i++)

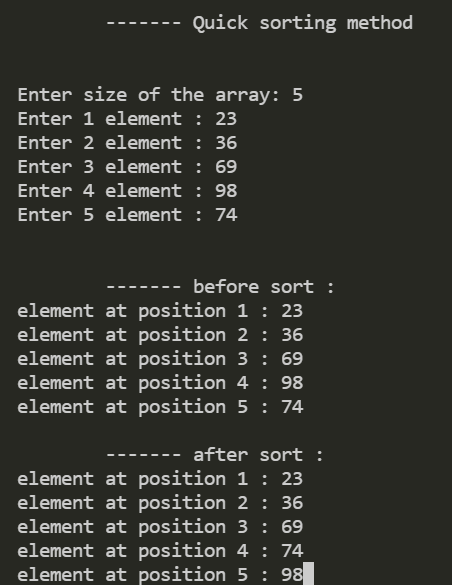
    {

        printf("\nelement at position %d : %d",i+1,*arr*[i]);

    }

}

**Output:**



1. **Write a program to implement Linear Search.**

# include<stdio.h>

# include<conio.h>

*int* linear\_search(*int* *a*[], *int* *element*)

{

*int* i;

    for(i=0; i<5; i++)

    {

        if(a[i]==element)

        {

            return i;

        }

    }

        return -1;

}

*void* main()

{

*int* n,i,result,a[5];

    for(i=0; i<5; i++){

         printf("Enter element of array : ");

         scanf("%d",&a[i]);

    }

    printf("\nenter element to search :");

    scanf("%d",&n);

    result = linear\_search(a,n);

    if(result == -1)

    {

        printf("\nElement %d is not found!!! ",n);

    }

    else

    {

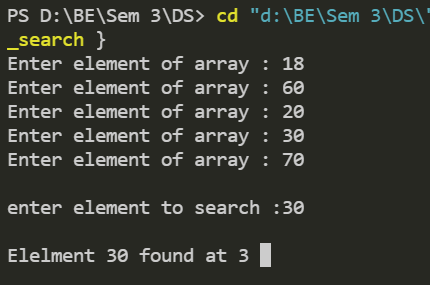
        printf("\nElelment %d found at %d ",n,result);

    }

    getch();

}

**Output:**



1. Write a program to implement Binary Search

# include<stdio.h>

# include<conio.h>

*void* sort\_array(*int* *a*[])

{

*int* i,j,temp;

    for(i=0; i<15-1; i++)

    {

        for(j=0; j<15-1-i; j++)

        {

            if(*a*[j]>*a*[j+1])

            {

                temp = *a*[j];

*a*[j] = *a*[j+1];

*a*[j+1] = temp;

            }

        }

    }

    printf("\n Array after sort : ");

    for(i=0;i<15;i++)

    {

        printf("\nelement at position %d : %d",i,*a*[i]);

    }

}

*int* binary\_search(*int* *a*[], *int* *element*)

{

*int* low, mid, high;

    low = 0;

    high = 15-1;

    while(low<=high)

    {

        mid = (low + high)/2;

        if(*a*[mid] == *element*)

        {

            return mid;

        }

        if(*a*[mid]<*element*)

        {

            low = mid+1;

        }

        else

        {

            low = mid-1;

        }

    }

    return -1;

}

*void* main()

{

*int* n,i,result,a[15];

    for(i=0; i<15; i++){

         printf("\nEnter element of array : ");

         scanf("%d",&a[i]);

    }

    sort\_array(a);

    printf("\n\nenter element to search :");

    scanf("%d",&n);

    result = binary\_search(a,n);

    if(result == -1)

    {

        printf("\nElement %d is not found!!! ",n);

    }

    else

    {

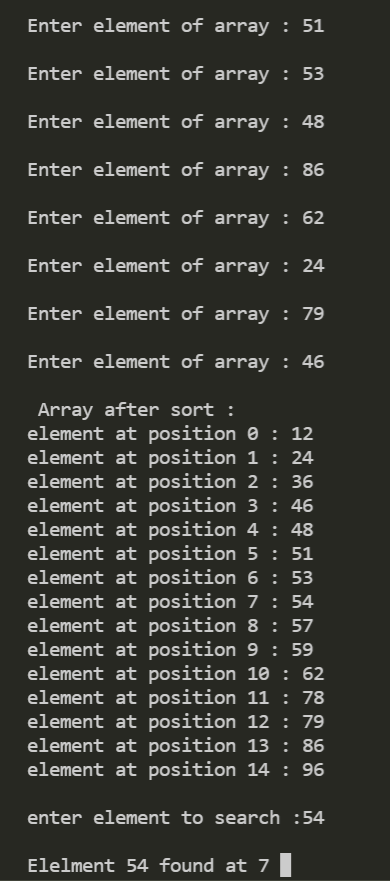
        printf("\nElelment %d found at %d ",n,result);

    }

    getch();

}

**Output**:



**12. Write a program to implement following operations on the doubly linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node after a given element (value) (d) Insert a node at a given position (e) Delete a first node of the linked list. (f) Delete a last node of the linked list. (g) Delete a node at specified position. (h) Delete a specified node (based on value).**

#include<stdio.h>

 #include<stdlib.h>

*struct* node

{

*struct* node \*prev;       *struct* node \*next;       *int* data;

    };

*struct* node \*head;

*void* insertion\_beginning();

*void* insertion\_last();

*void* insertion\_specified();

*void* deletion\_beginning();

*void* deletion\_last();

*void* deletion\_specified();

*void* display();

*void* search();

*void* main ()

    {

*int* choice =0;

    while(choice != 9)

    {

        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random loca tion\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Display\n9.Exit\n ");

        printf("\nEnter your choice: \n");

        scanf("\n%d",&choice);

        switch(choice)

        {

            case 1:

            insertion\_beginning();

            break;

            case 2:

            insertion\_last();

            break;

            case 3:

            insertion\_specified();

            break;

            case 4:

            deletion\_beginning();

            break;

            case 5:

            deletion\_last();

            break;

            case 6:

            deletion\_specified();

            break;

            case 7:

            search();

            break;

            case 8:

            display();

            break;

            case 9:

            exit(0);

            break;

            default:

            printf("Please enter valid choice..");

        }

    }

}

*void* insertion\_beginning()

    {

*struct* node \*ptr;

*int* item;

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

        if(ptr == NULL)

        {

        printf("\nOVERFLOW");

        }

        else

        {

        printf("\nEnter Item value");

        scanf("%d",&item);

        if(head==NULL)

        {

        ptr->next = NULL;

        ptr->prev=NULL;

        ptr->data=item;

        head=ptr;

        }

        else

        {

        ptr->data=item;

        ptr->prev=NULL;

        ptr->next = head;

    head->prev=ptr;

        head=ptr;

        }

        printf("\nNode inserted\n");

        }

}

*void* insertion\_last()

    {

*struct* node \*ptr,\*temp;

*int* item;

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

    if(ptr == NULL)

    {

    printf("\nOVERFLOW");

    }

    else

    {

    printf("\nEnter value");

    scanf("%d",&item);

    ptr->data=item;

    if(head == NULL)

    {

    ptr->next = NULL;

    ptr->prev = NULL;

    head = ptr;

    }

    else

    {

    temp = head;

    while(temp->next!=NULL)

    {

    temp = temp->next;

    }

    temp->next = ptr;

    ptr ->prev=temp;

    ptr->next = NULL;

    }

    }

    printf("\nnode inserted\n");

    }

*void* insertion\_specified()

    {

*struct* node \*ptr,\*temp;

*int* item,loc,i;

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

    if(ptr == NULL)

    {

    printf("\n OVERFLOW");

}

    else

    {

    temp=head;

printf("Enter the location");

    scanf("%d",&loc);

    for(i=0;i<loc;i++)

    {

    temp = temp->next;

    if(temp == NULL)

    {

    printf("\n There are less than %d elements", loc);

    return;

    }

    }

    printf("Enter value");

    scanf("%d",&item);

    ptr->data = item;

    ptr->next = temp->next;

    ptr -> prev = temp;

    temp->next = ptr;

    temp->next->prev=ptr;

printf("\nnode inserted\n");

    }

    }

*void* deletion\_beginning()

    {

*struct* node \*ptr;

    if(head == NULL)

    {

printf("\n UNDERFLOW");

    }

    else if(head->next == NULL)

    {

    head = NULL;

    free(head);

    printf("\nnode deleted\n");

    }

    else

    {

    ptr = head;

    head = head -> next;

    head -> prev = NULL;

    free(ptr);

    printf("\nnode deleted\n");

    }

    }

*void* deletion\_last()

    {

*struct* node \*ptr;

    if(head == NULL)

    {

    printf("\n UNDERFLOW");

    }

    else if(head->next == NULL)

    {

    head = NULL;

    free(head);

    printf("\nnode deleted\n");

    }

    else

    {

    ptr = head;

    if(ptr->next != NULL)

    {

    ptr = ptr -> next;

    }

    ptr -> prev -> next = NULL;

    free(ptr);

    printf("\nnode deleted\n");

    }

    }

*void* deletion\_specified()

    {

*struct* node \*ptr, \*temp;

*int* val;

    printf("\n Enter the data after which the node is to be deleted : ");

    scanf("%d", &val);

    ptr = head;

    while(ptr -> data != val)

    ptr = ptr -> next;

    if(ptr -> next == NULL)

    {

    printf("\nCan't delete\n");

}

    else if(ptr -> next -> next == NULL)

    {

    ptr ->next = NULL;

    }

    else

    {

    temp = ptr -> next;

    ptr -> next = temp -> next;             temp -> next -> prev = ptr;             free(temp);

    printf("\nnode deleted\n");

    }

    }

*void* display()

    {

*struct* node \*ptr;

    printf("\n printing values...\n");

    ptr = head;

    while(ptr != NULL)

    {

    printf("%d\n",ptr->data);

    ptr=ptr->next;

    }

    }

*void* search()

    {

*struct* node \*ptr;

*int* item,i=0,flag;

    ptr = head;

    if(ptr == NULL)

{

    printf("\nEmpty List\n");

    }

    else

    {

    printf("\nEnter item which you want to search?\n");

    scanf("%d",&item);

    while (ptr!=NULL)

    {

    if(ptr->data == item)

    {

    printf("\nitem found at location %d ",i+1);

    flag=0;

    break;

    }

    else

    {

    flag=1;

    }

    i++;

    ptr = ptr -> next;

    }

    if(flag==1)

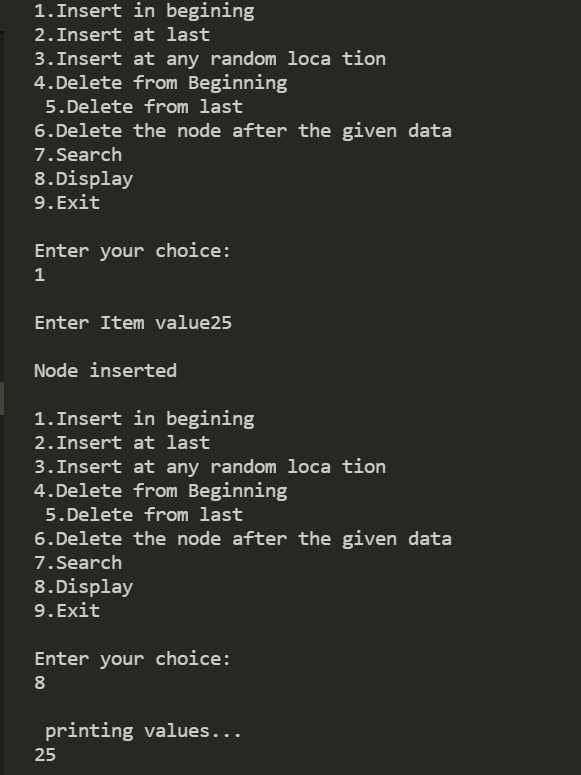
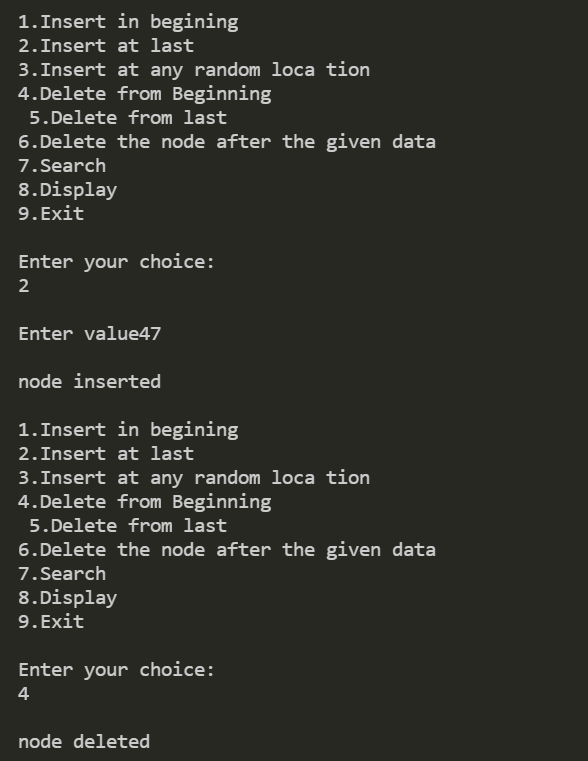
    {

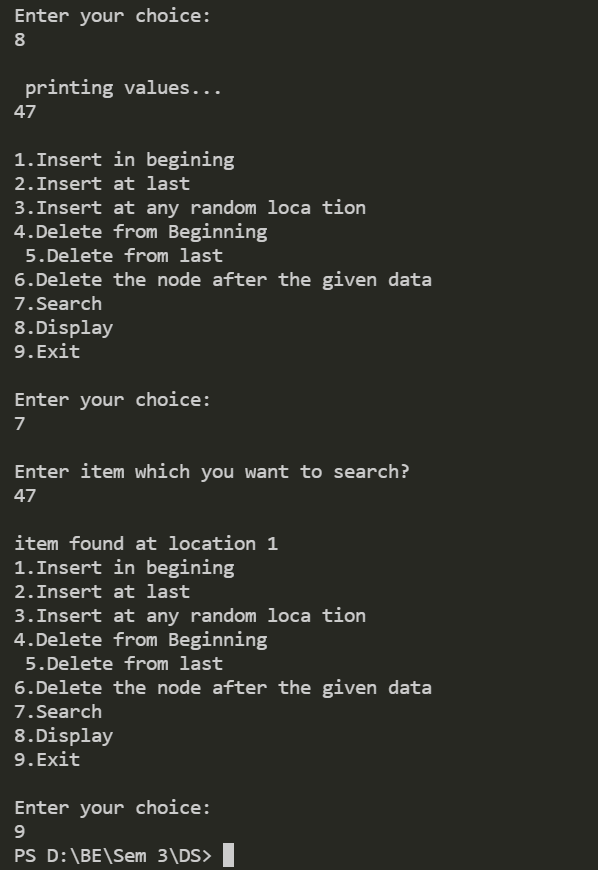
    printf("\nItem not found\n");

    }

    }

    }

**Output:**

****