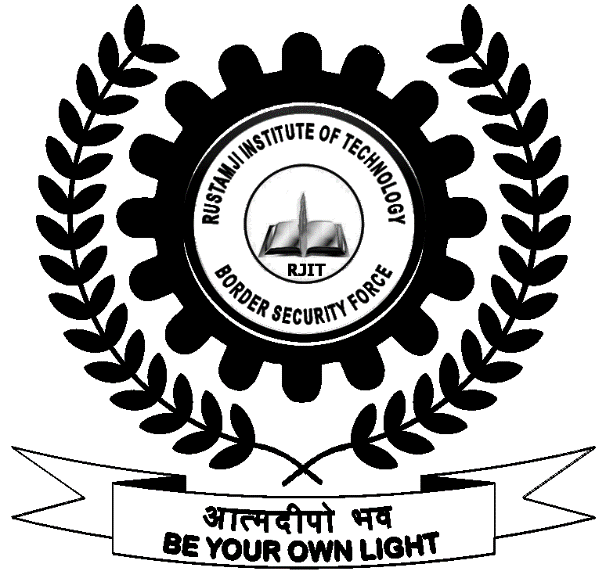
**RUSTAMJI INSTITUTE OF TECHNOLOGY**

**BSF ACADEMY, TEKANPUR**

**Lab File for**

**CS303 (Data Structure)**



Submitted by

*SUBHRAJIT BARMAN (0902CS211052)*

B.Tech. Computer Science & Engineering 3rd Semester

(2021-2025 batch)

|  |  |
| --- | --- |
| Subject Teacher  Dr. Jagdish Makhijani | File Checked by  Mr. Yashwant Pathak |

**TABLE OF CONTENTS**

**Section-A (Linked List)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Practical Description** | **Page Nos.** |
| 1 | Implementation of Linked List using array. |  |
| 2 | Implementation of Linked List using Pointers. |  |
| 3 | Implementation of Doubly Linked List using Pointers. |  |
| 4 | Implementation of Circular Single Linked List using Pointers. |  |
| 5 | Implementation of Circular Doubly Linked List using Pointers. |  |
| 6 | A program to maintain the record of students which contains their rollno (should be unique), name (only in capital letters), date of birth, course and semester using linked list. User should have utility to insert, modify, delete and search a record in the list. |  |

**Section-B (Stack)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Practical Description** | **Page Nos.** |
| 1 | Implementation of Stack using Array. |  |
| 2 | Implementation of Stack using Pointers. |  |
| 3 | Evaluation of Arithmetic Expression in PreOrder/PostOrder/ InOrder using Stack. |  |
| 4 | Program for Tower of Hanoi using recursion. |  |

**Section-C (Queue)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Practical Description** | **Page Nos.** |
| 1 | Implementation of Queue using Array. |  |
| 2 | Implementation of Queue using Pointers. |  |
| 3 | Implementation of Circular Queue using Array. |  |
| 4 | Implementation of DeQue/Priority Que. |  |

**Section-D (Trees)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Practical Description** | **Page Nos.** |
| 1 | Implementation of Binary Search Tree. |  |
| 2 | Conversion of BST PreOrder/PostOrder/InOrder. |  |

**Section-E (Sorting & Searching)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Practical Description** | **Page Nos.** |
| 1 | Implementation of Sorting  a. Bubble  b. Selection  c. Insertion  d. Quick  e. Merge |  |
| 2 | Implementation of Binary Search on a list of numbers stored in an Array |  |
| 3 | Implementation of Binary Search on a list of strings stored in an Array |  |
| 4 | Implementation of Linear Search on a list of strings stored in an Array |  |
| 5 | Implementation of Binary Search on a list of strings stored in a Single Linked List |  |

**Practical No.: 1**

Program Description: Implementation of Linked List using array.

Solution: -

#include<stdio.h>

struct node

{

int nData;

struct node\* pNode;

};

/\*

Function to create Linked List from Array elements.

\*/

struct node\* createLL(int\* nArr, int n)

{

static int i=0;

struct node\* t = NULL;

if(n==0)

return NULL;

// Create New Node

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

t->nData = nArr[i++];

t->pNode = createLL(nArr, --n);

return t;

}

void displayLL(struct node \*t)

{

while(t)

{

printf("%d ", t->nData);

t=t->pNode;

}

}

int main()

{

int n=0, i=0, arr[100]={0};

struct node \*t = NULL;

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

scanf("%d", &n);

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

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

printf("\nCreate linked list from array");

t =createLL(arr, n);

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

if(t)

displayLL(t);

}

Output: -

Enter the number of elements: 5

12

32

45

65

78

Create linked list from array

Display Linked List:

12 32 45 65 78

**Practical No.: 2**

Program Description: - Implementation of Linked List using pointers.

Solution: -

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

void implemantation()

{

struct node

{

int data;

struct node \*next;

};

struct node \*head, \*newnode, \*temp;

head = 0;

int choice;

while (choice)

{

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

printf("Enter the data\n");

scanf("%d", &newnode->data);

newnode->next = 0;

if (head == 0)

{

head = temp = newnode;

}

else

{

temp->next = newnode;

temp = newnode;

}

printf("Your previous data is added \nDo you want to add more press (0,1) ?\n");

scanf("%d", &choice);

}

temp = head;

printf("ENTERED DATA ARE:-\n");

while (temp != 0)

{

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

temp = temp->next;

}

}

int main()

{

implemantation();

return 0;

}

OUTPUT: -

Enter the data

5

Your previous data is added

Do you want to add more press (0,1)?

1

ENTERED DATA ARE: -

5

Enter the data

5

**Practical No.: 3**

Program Description: - Implementation of Doubly Linked List using Pointers.

Solution: -

#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 display();

void main()

{

int choice = 0;

while (choice != 5)

{

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

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Show\n5.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:

display();

break;

case 5:

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 display()

{

struct node \*ptr;

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

ptr = head;

while (ptr != NULL)

{

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

ptr = ptr->next;

}

}

OUTPUT: -

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

4

printing values...

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

1

Enter Item value56

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

1

Enter Item value45

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

1

Enter Item value56

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

4

printing values...

56

45

56

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Show

5.Exit

Enter your choice?

5

**Practical No.: 4**

Program description: - Implementation of Circular Single Linked List using Pointers.

Solution: -

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert()

{

struct node \*ptr, \*temp;

int item;

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

if (ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter the node data?");

scanf("%d", &item);

ptr->data = item;

if (head == NULL)

{

head = ptr;

ptr->next = head;

}

else

{

temp = head;

while (temp->next != head)

temp = temp->next;

ptr->next = head;

temp->next = ptr;

head = ptr;

}

printf("\nnode inserted\n");

}

}

void display()

{

struct node \*ptr;

ptr = head;

if (head == NULL)

{

printf("\nnothing to print");

}

else

{

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

while (ptr->next != head)

{

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

ptr = ptr->next;

}

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

}

}

int main()

{

int choice = 0;

while (choice != 2)

{

printf("\n1.Insert in begining\n2.Display\n");

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

scanf("\n%d", &choice);

if (choice == 1)

beginsert();

if (choice == 2)

display();

}

}

OUTPUT: -

1.Insert in begining

2.Display

Enter your choice

1

Enter the node data?56

node inserted

1.Insert in begining

2.Display

Enter your choice

1

Enter the node data?54

node inserted

1.Insert in begining

2.Display

Enter your choice

2

printing values ...

54

56

**Practical No.: 5**

Program description: - Implementation of Circular Doubly Linked List using Pointers.

Solution: -

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning()

{

struct node \*newnode, \*temp;

int recieve\_data;

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

if (newnode == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Item value:- ");

scanf("%d", &recieve\_data);

newnode->data = recieve\_data;

if (head == NULL)

{

head = newnode;

newnode->next = head;

newnode->prev = head;

}

else

{

temp = head;

while (temp->next != head)

{

temp = temp->next;

}

temp->next = newnode;

newnode->prev = temp;

head->prev = newnode;

newnode->next = head;

head = newnode;

}

printf("\nNode inserted\n");

}

}

void display()

{

struct node \*ptr;

ptr = head;

if (head == NULL)

{

printf("\nnothing to print");

}

else

{

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

while (ptr->next != head)

{

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

ptr = ptr->next;

}

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

}

}

int main()

{

int choice = 0;

while (choice != 2)

{

printf("\n1.Insert in begining\n2.Display\n");

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

scanf("\n%d", &choice);

if (choice == 1)

insertion\_beginning();

if (choice == 2)

display();

}

}

OUTPUT: -

1.Insert in begining

2.Display

Enter your choice

1

Enter Item value:- 56

Node inserted

1.Insert in begining

2.Display

Enter your choice

1

Enter Item value:- 65

Node inserted

1.Insert in begining

2.Display

Enter your choice

2

printing values ...

65

56

**Practical No.: 6**

Program Description: - A program to maintain the record of students which contains their rollno (should be unique), name (only in capital letters), date of birth, course and semester using linked list. User should have utility to insert, modify, delete and search a record in the list.

Solution: -

Output: -

**Stack**

**Practical No.: 7**

Program description: - Implementation of Stack using Array.

SOLUTION: -

#include <stdio.h>

int stack[100], choice, n, top, x, i;

void push(void);

void pop(void);

void display(void);

int main(){

top = -1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d", &n);

printf("\n\t STACK OPERATIONS USING ARRAY");

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

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do{

printf("\n Enter the Choice:");

scanf("%d", &choice);

switch (choice){

case 1:{

push();

break;

}

case 2:{

pop();

break;

}

case 3:{

display();

break;

}

case 4:{

printf("\n\t EXIT POINT ");

break;

}

default:{

printf("\n\t Please Enter a Valid Choice (1/2/3/4)");

}

}

} while (choice != 4);

return 0;

}

void push(){

if (top >= n - 1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d", &x);

top++;

stack[top] = x;

}

}

void pop(){

if (top <= -1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d", stack[top]);

top--;

}

}

void display(){

if (top >= 0)

{

printf("\n The elements in STACK \n");

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

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

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

OUTPUT: -

Enter the size of STACK[MAX=100]:10

STACK OPERATIONS USING ARRAY

--------------------------------

1.PUSH

2.POP

3.DISPLAY

4.EXIT

Enter the Choice:1

Enter a value to be pushed:12

Enter the Choice:1

Enter a value to be pushed:24

Enter the Choice:1

Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98

24

12

Press Next Choice

Enter the Choice:2

The popped elements is 98

Enter the Choice:3

The elements in STACK

24

12

Press Next Choice

**Practical No.: 8**

Program Description: - Implementation of Stack using Pointers.

SOLUTION: -

#include<stdio.h>

#include<stdlib.h>

struct stackarr;

typedef struct stackarr \* Stack;

struct stackarr

{

int Capacity;

int TopOfStack;

int \*Array;

};

void MakeEmpty(Stack s)

{

s->TopOfStack = -1;

}

Stack CreateStack(int MaxElements)

{

Stack s;

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

s->Array = (int \*)malloc(sizeof(int) \* MaxElements);

s->Capacity = MaxElements;

MakeEmpty(s);

return s;

}

void DisposeStack(Stack s)

{

if(s != NULL)

{

free(s->Array);

free(s);

}

}

int isFull(Stack s)

{

return s->TopOfStack == s->Capacity - 1;

}

int isEmpty(Stack s)

{

return s->TopOfStack == -1;

}

void Push(int x, Stack s)

{

if(isFull(s))

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

else

s->Array[++s->TopOfStack] = x;

}

void Pop(Stack s)

{

if(isEmpty(s))

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

else

s->TopOfStack--;

}

int Top(Stack s)

{

if(isEmpty(s))

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

else

return s->Array[s->TopOfStack];

}

int TopAndPop(Stack s)

{

if(isEmpty(s))

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

else

return s->Array[s->TopOfStack--];

}

void Display(Stack s)

{

int i;

if(isEmpty(s))

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

else

{

printf("The Stack Elements are :: ");

for(i=s->TopOfStack; i >= 0; i--)

printf("%d ",s->Array[i]);

printf("\n\n");

}

}

int main()

{

int n, x, ch, i;

Stack s;

printf("Enter the maximum number of elements in the stack :: ");

scanf("%d", &n);

s = CreateStack(n);

printf("ARRAY IMPLEMENTATION OF STACK ADT\n\n");

do

{

printf("\n\n1. PUSH\t 2. POP\t 3.TOP \t 4. TOPANDPOP\t 5. PRINT\t 6. QUIT\n\nEnter the choice :: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

printf("Enter the element to be pushed :: ");

scanf("%d",&x);

Push(x,s);

break;

case 2:

Pop(s);

break;

case 3:

printf("The Top element in the stack :: %d\n\n", Top(s));

break;

case 4:

printf("The popped top element in the stack :: %d\n\n", TopAndPop(s));

break;

case 5:

Display(s);

break;

}

}

while(ch<6);

DisposeStack(s);

return 0;}

Output: -

enter the maximum number of elements in the stack: 5

ARRAY IMPLEMENTATION OF STACK ADT

1. PUSH 2. POP 3.TOP 4. TOPANDPOP 5. PRINT 6. QUIT

Enter the choice: 1

Enter the element to be pushed: 10

1. PUSH 2. POP 3.TOP 4. TOPANDPOP 5. PRINT 6. QUIT

Enter the choice: 1

Enter the element to be pushed: 20

1. PUSH 2. POP 3.TOP 4. TOPANDPOP 5. PRINT 6. QUIT

Enter the choice: 1

Enter the element to be pushed: 30

1. PUSH 2. POP 3.TOP 4. TOPANDPOP 5. PRINT 6. QUIT

Enter the choice: 5

The Stack Elements are: 30 20 10

1 Enter the choice: 2

1. PUSH 2. POP 3.TOP 4. TOPANDPOP 5. PRINT 6. QUIT

Enter the choice: 5

The Stack Elements are: 20 10

**Practical No.: 9**

Program Description: - Evaluation of Arithmetic Expression in PreOrder/PostOrder/ InOrder using Stack.

Solution: -

Output: -

**Practical No.: 10**

Program description: - Program for Tower of Hanoi using recursion.

SOLUTION: -

#include <stdio.h>

int step = 0;

void towerofHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

step++;

if (n == 1)

{

printf("\nStep-%d: Move disk 1 from rod %c to rod %c ", step, from\_rod, to\_rod);

return;

}

towerofHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

printf("\nStep-%d: Move disk %d from rod %c to rod %c", step, n, from\_rod, to\_rod);

towerofHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

int n;

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

scanf("%d", &n);

towerofHanoi(n, 'A', 'C', 'B');

return 0;

}

OUTPUT: -

Enter the number of disks: 3

Step-3: Move disk 1 from rod A to rod C

Step-3: Move disk 2 from rod A to rod B

Step-4: Move disk 1 from rod C to rod B

Step-4: Move disk 3 from rod A to rod C

Step-6: Move disk 1 from rod B to rod A

Step-6: Move disk 2 from rod B to rod C

Step-7: Move disk 1 from rod A to rod C

**Queue**

**Practical No.: 11**

Program description: - Implementation of Queue using Array.

SOLUTION: -

#include<stdio.h>

#define

int main()

{

int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;

printf("Queue using Array");

printf("\n1.Insertion \n3.Display \n4.Exit");

while(ch)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else

{

printf("\n Enter no %d:",j++);

scanf("%d",&queue[rear++]);

}

break;

case 3:

printf("\nQueue Elements are:\n ");

if(front==rear)

printf("\n Queue is Empty");

else

{

for(i=front; i<rear; i++)

{

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

printf("\n");

}

break;

case 4:

break;

default:

printf("Wrong Choice: please see the options");

}

}

} return 0;}

Output: -

Queue using Array

1.Insertion

3.Display

4.Exit

Enter the Choice:1

Enter no 1:10

Enter the Choice:1

Enter no 2:54

Enter the Choice:1

Enter no 3:98

Enter the Choice:1

Enter no 4:234

Enter the Choice:3

Queue Elements are:

10

54

98

234

Enter the Choice:3

Queue Elements are:

10

54

98

234

Enter the Choice:4

**Practical No.: 12**

Program description: - Implementation of Queue using Pointers.

SOLUTION: -

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front = 0;

struct node \*rear = 0;

void enqueu(int x)

{

struct node \*newnode;

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

newnode->data = x;

newnode->next = 0;

if (front == 0 && rear == 0)

front = rear = newnode;

else

{

rear->next = newnode;

rear = newnode;

}

}

void display()

{

struct node \*temp;

if (front == 0 && rear == 0)

printf("NOTHING TO SHOW\n");

else

{

temp = front;

while (temp != 0)

{

printf("The number is %d\n", temp->data);

temp = temp->next;

}

}

}

void peek()

{

if (front == 0 && rear == 0)

printf("NOTHING TO SHOW\n");

else

{

printf("The above data is:-", front->data);

}

}

int main()

{printf("\n\t QUEUQ OPERATIONS USING ARRAY");

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

printf("\n\t 1.Enqueue\n\t 2.DISPLAY\n\t 3.EXIT");

int ch;

do

{

printf("\n Enter the Choice:");

scanf("%d", &ch);

switch (ch)

{

case 1:

{

int p;

printf("Enter number to add\n");

scanf("%d", &p);

enqueu(p);

break;

}

case 2:

{

display();

break;

}

case 3: {

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

} while (ch != 4);

return 0;

}

OUTPUT: -

QUEUQ OPERATIONS USING ARRAY

--------------------------------

1.Enqueue

2.DISPLAY

3.EXIT

Enter the Choice:1

Enter number to add

123

Enter the Choice:1

Enter number to add

654

Enter the Choice:1

Enter number to add

7845

Enter the Choice:2

The number is 123

The number is 654

The number is 7845

Enter the Choice:3

**Practical No.: 13**

Program description: - Implementation of Circular Queue using Array.

SOLUTION: -

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull()

{

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))

return 1;

return 0;

}

int isEmpty()

{

if (front == -1)

return 1;

return 0;

}

void enQueue(int element)

{

if (isFull())

printf("\n Queue is full!! \n");

else

{

if (front == -1)

front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted -> %d", element);

}

}

int deQueue()

{

int element;

if (isEmpty())

{

printf("\n Queue is empty !! \n");

return (-1);

}

else

{

element = items[front];

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

front = (front + 1) % SIZE;

}

printf("\n Deleted element -> %d \n", element);

return (element);

}

}

void display()

{

int i;

if (isEmpty())

printf(" \n Empty Queue\n");

else

{

printf("\n Front -> %d ", front);

printf("\n Items -> ");

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

{

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

}

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

printf("\n Rear -> %d \n", rear);

}

}

int main()

{

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

enQueue(6);

display();

deQueue();

display();

enQueue(7);

display();

enQueue(8);

return 0;

}

OUTPUT: -

Queue is empty !!

Inserted -> 1

Inserted -> 2

Inserted -> 3

Inserted -> 4

Inserted -> 5

Queue is full!!

Front -> 0

Items -> 1 2 3 4 5

Rear -> 4

Deleted element -> 1

Front -> 1

Items -> 2 3 4 5

Rear -> 4

Inserted -> 7

Front -> 1

Items -> 2 3 4 5 7

Rear -> 0

Queue is full!!

**Practical No.: 14**

Program description: -Implementation of DeQue/Priority Que.

SOLUTION: -

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front = 0;

struct node \*rear = 0;

void enqueu(int x)

{

struct node \*newnode;

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

newnode->data = x;

newnode->next = 0;

if (front == 0 && rear == 0)

front = rear = newnode;

else

{

rear->next = newnode;

rear = newnode;

}

}

void display ()

{

struct node \*temp;

if (front == 0 && rear == 0)

printf("NOTHING TO SHOW\n");

else

{

temp = front;

while (temp != 0)

{

printf("The number is %d\n", temp->data);

temp = temp->next;

}

}

}

void dequeue()

{

struct node \*temp;

temp = front;

if (front == 0 && rear == 0)

printf("NOTHING TO SHOW\n");

else

{

printf(" The dequeued element is %d\n", front->data);

front = front->next;

free(temp);

}

}void peek()

{

if (front == 0 && rear == 0)

printf("NOTHING TO SHOW\n");

else

{

printf("The above data is:-", front->data);

}

}

int main()

{

printf("\n\t QUEUQ OPERATIONS USING ARRAY");

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

printf("\n\t 1.Enqueue\n\t 2.Dequeue\n\t 3.DISPLAY\n\t 4.EXIT");

int ch;

do

{

printf("\n Enter the Choice:");

scanf("%d", &ch);

switch (ch)

{

case 1:

{

int p;

printf("Enter number to add\n");

scanf("%d", &p);

enqueu(p);

break;

}

case 2:

{

dequeue();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

} while (ch != 4);

return 0;}

OUTPUT: -

QUEUQ OPERATIONS USING ARRAY

1.Enqueue

2.Dequeue

3.DISPLAY

4.EXIT

Enter the Choice:1

Enter number to add

123

Enter the Choice:1

Enter number to add

456

Enter the Choice:1

Enter number to add

789

Enter the Choice:1

Enter number to add

963

Enter the Choice:2

The dequeued element is 123

Enter the Choice:3

The number is 456

The number is 789

The number is 963

**Trees**

**Practical No.: 15**

Program description: - Implementation of Binary Search Tree.

SOLUTION: -

#include <stdio.h>

#include <stdlib.h>

struct node {

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item) {

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node \*root) {

if (root != NULL) {

inorder(root->left); printf("%d -> ", root->key);

inorder(root->right);

}

}

struct node \*insert(struct node \*node, int key) {

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

struct node \*minValueNode(struct node \*node) {

struct node \*current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node \*deleteNode(struct node \*root, int key) {

if (root == NULL) return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

struct node \*temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct node \*temp = root->left;

free(root);

return temp;

}

struct node \*temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main() {

struct node \*root = NULL;

root = insert (root, 8);

root = insert (root, 3);

root = insert (root, 1);

root = insert (root, 6);

root = insert (root, 7);

root = insert (root, 10);

root = insert (root, 14);

root = insert (root, 4);

printf("Inorder traversal: ");

inorder(root);

printf("\nAfter deleting 10\n");

root = deleteNode(root, 10);

printf("Inorder traversal: ");

inorder(root);

}

OUTPUT: -

Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->

After deleting 10

Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->

**Practical No.: 16**

Program description: - Conversion of BST PreOrder/PostOrder/InOrder.

SOLUTION: -

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void printInorder(struct node \*node)

{

if (node == NULL)

return;

printInorder(node->left);

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

printInorder(node->right);

}

void printPreorder(struct node \*node)

{

if (node == NULL)

return;

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

printPreorder(node->left);

printPreorder(node->right);

}

void printPostorder(struct node \*node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

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

}

int main()

{

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("\nInorder traversal of binary tree is \n");

printInorder(root);

printf("\nPreorder traversal of binary tree is \n");

printPreorder(root);

printf("\nPostorder traversal of binary tree is \n");

printPostorder(root);

getchar();

return 0;

}

OUTPUT: -

Inorder traversal of binary tree is

4 2 5 1 3

Preorder traversal of binary tree is

1 2 4 5 3

Postorder traversal of binary tree is

4 5 2 3 1

**Sorting & Searching**

**Practical No.: 17**

Program description: - Implementation of Sorting

1. Bubble
2. Selection
3. Insertion
4. Quick
5. Merge

SOLUTION: -

1. Bubble Sort

#include <stdio.h>

int main()

{

int arr[20];

int n, temp, k;

printf("Enter the array size:- ");

scanf("%d", &n);

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

{

printf("Enter the element:- ");

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

}

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

{

for (k = 0; k < n - 1; k++)

{

if (arr[k] > arr[k + 1])

{

temp = arr[k];

arr[k] = arr[k + 1];

arr[k + 1] = temp;

}

}

}

printf("The sorted array is--\n");

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

{

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

}

return 0;

}

Output: -

Enter the array size: - 5

Enter the element: - 12

Enter the element: - 32

Enter the element: - 15

Enter the element: - 1

Enter the element: - 654

The sorted array is--

1 12 15 32 654

1. Selection Sort: -

SOLUTION: -

#include <stdio.h>

void selectionsort(int \*A, int n)

{

int indexofmin, temp;

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

indexofmin = i;

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

{

if (A[j] < A[indexofmin])

{

indexofmin = j;

}

}

temp = A[i];

A[i] = A[indexofmin];

A[indexofmin] = temp;

}

}

int main()

{

int A[20];

int n, temp, k;

printf("Enter the array size:- ");

scanf("%d", &n);

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

{

printf("Enter the element:- ");

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

}

selectionsort(A, n);

printf("The sorted array is--\n");

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

{

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

}

return 0;

}

OUTPUT: -

Enter the array size:- 5

Enter the element:- 100

Enter the element:- 50

Enter the element:- 25

Enter the element:- 8

Enter the element:- 1

The sorted array is--

1 8 25 50 100

1. Insertion Sort: -

SOLUTION: -

#include <stdio.h>

void insertionsort(int \*A, int n)

{

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

{

int current = A[i + 1];

int j = i - 1;

while (A[j] > current && j >= 0)

{

A[j + 1] = A[j];

j--;

}

A[j + 1] = current;

}

}

int main()

{

int A[] = {12, 65, 1, 78, 60, 9, 0};

int n = 7;

insertionsort(A, n);

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

{

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

}

return 0;

}

OUTPUT: -

THE SORTED ARRAY IS: -

0 1 9 12 60 78

1. Quick Sort: -

SOLUTION: -

#include <stdio.h>

void swap(int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int array[], int low, int high)

{

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++)

{

if (array[j] <= pivot)

{

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int array[], int low, int high)

{

if (low < high)

{

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

void printArray(int array[], int size)

{

for (int i = 0; i < size; ++i)

{

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

}

printf("\n");

}

int main()

{

int data[] = {8, 7, 2, 1, 0, 9, 6};

int n = sizeof(data) / sizeof(data[0]);

printf("Unsorted Array\n");

printArray(data, n);

quickSort(data, 0, n - 1);

printf("Sorted array in ascending order: \n");

printArray(data, n);

}

OUTPUT: -

Unsorted Array

8 7 2 1 0 9 6

Sorted array in ascending order:

1. 1 2 6 7 8 9
2. Merge Sort: -

SOLUTION: -

#include<stdio.h>

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

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

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

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

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int size)

{

for (int i = 0; i < size; i++)

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

printf("\n");

}

int main()

{

int arr[] = {6, 5, 12, 10, 9, 1};

int size = sizeof(arr) / sizeof(arr[0]);

mergeSort(arr, 0, size - 1);

printf("Sorted array: \n");

printArray(arr, size);

}

OUTPUT: -

The unsorted array is: -

6, 5, 12, 10, 9, 1

Sorted array:

1 5 6 9 10 12

**Practical No.: 18**

Program description: -Implementation of Binary Search on a list of numbers stored in an Array.

Solution: -

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

if (array[mid] == x)

return mid;

if (array[mid] > x)

return binarySearch(array, x, low, mid - 1);

return binarySearch(array, x, mid + 1, high);

}

return -1;

}

int main(void) {

int array[] = {3, 4, 5, 6, 7, 8, 9};

int n = sizeof(array) / sizeof(array[0]);

int x = 4;

int result = binarySearch(array, x, 0, n - 1);

if (result == -1)

printf("Not found");

else

printf("Element is found at index %d", result);

}

OUTPUT: -

Element is found at index 1

**Practical No.: 19**

Program Description: - Implementation of Linear Search on a list of strings stored in an Array

SOLUTION: -

#include <stdio.h>

int linearSearch(int a[], int n, int val)

{

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

{

if (a[i] == val)

return i + 1;

}

return -1;

}

int main()

{

int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52};

int val = 41;

int n = sizeof(a) / sizeof(a[0]);

int res = linearSearch(a, n, val);

printf("The elements of the array are- ");

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

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

printf("\nElement to be searched is- %d", val);

if (res == -1)

printf("\nElement is not present in the array");

else

printf("\nElement is present at %d position of array", res);

return 0;

}

OUTPUT: -

The elements of the array are- 70 40 30 11 57 41 25 14 52

Element to be searched is- 41

Element is present at 6 positions of array

**GITHUB PROFILE:** -

[Subhrajit303 (Subhrajit Barman) (github.com)](https://github.com/Subhrajit303)