

Data structures & algorithm

Practical File



Submitted by :

Kunal

B.E. Information Technology

2nd year -3rd semester

Roll No. - UE218058

Submitted to :

Prof. Preetjot Kaur

**Experiment-1**

* Program for Insertion of values in array

1. ***In the beginning***

#include <iostream>

using namespace std;

int main(){

//no. of elements in array

int n;

cout<<"enter the no. of elments: ";

cin>>n;

//initialization of array

int arr[n];

cout<<"input your array: ";

//array input

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

cin>>arr[i];

}

//array output before insertion

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

cout<<arr[i]<<" ";

}cout<<endl;

//input the insertion element

int insert;

cout<<"enter the insertion element: ";

cin>>insert;

//inserting the element

for(int i=n;i>=0;i--){

arr[i+1]=arr[i];

}

arr[0]=insert;

//array output after insertion

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

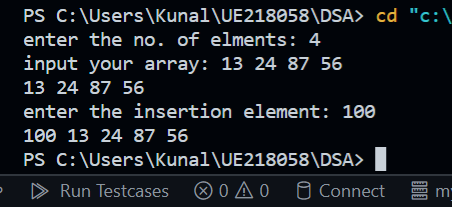
cout<<arr[i]<<" ";

}cout<<endl;

return 0;

}

Output:



1. ***At the given index***

#include <iostream>

using namespace std;

int main(){

//no. of elements in array

int n;

cout<<"enter the no. of elments: ";

cin>>n;

//initialization of array

int arr[n];

cout<<"input your array: ";

//array input

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

cin>>arr[i];

}

//array output before insertion

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

cout<<arr[i]<<" ";

}cout<<endl;

//input the insertion element

int insert;

cout<<"enter the insertion element: ";

cin>>insert;

//input the insertion position

int index;

cout<<"enter the index of insertion: ";

cin>>index;

//inserting the element

for(int i=n;i>index;i--){

arr[i]=arr[i-1];

}

arr[index]=insert;

//array output after insertion

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

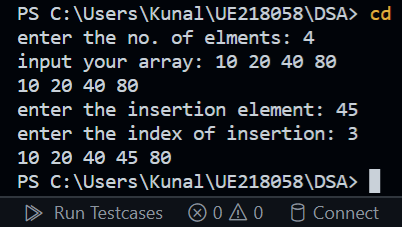
cout<<arr[i]<<" ";

}cout<<endl;

return 0;

}

Output:



1. ***At the ending***

#include <iostream>

using namespace std;

int main(){

//no. of elements in array

int n;

cout<<"enter the no. of elments: ";

cin>>n;

//initialization of array

int arr[n];

cout<<"input your array: ";

//array input

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

cin>>arr[i];

}

//array output before insertion

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

cout<<arr[i]<<" ";

}cout<<endl;

//input the insertion element

int insert;

cout<<"enter the insertion element: ";

cin>>insert;

//input the insertion position

int index=n;

//inserting the element

for(int i=n;i>index;i--){

arr[i]=arr[i-1];

}

arr[index]=insert;

//array output after insertion

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

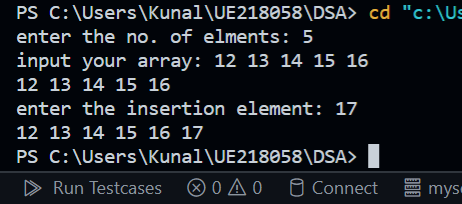
cout<<arr[i]<<" ";

}cout<<endl;

return 0;

}

Output:



* Program for deleting element in an array:

#include<iostream>

using namespace std;

int main(){

int arr[100], size, i, del, count=0;

cout<<"Enter the size of element: ";

cin>>size;

cout<<"\nEnter array elements: \n";

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

cin>>arr[i];

}

cout<<"\nEnter element to be delete : ";

cin>>del;

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

if(arr[i]==del)

{

for(int j=i; j<(size-1); j++)

{

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

}

count++;

break;

}

}

if(count==0){

cout<<"\nElement not found.!\n";

}

else{

cout<<"\nElement deleted successfully!\n";

cout<<"new array is:\n";

for(i=0; i<(size-1); i++)

{

cout<<arr[i]<<" ";

}

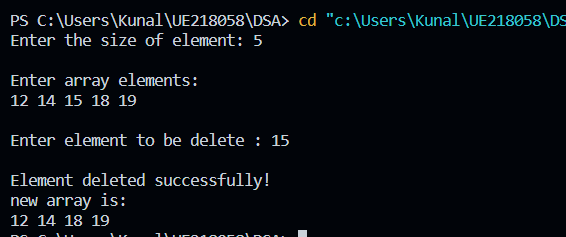
}

cout<<"\n";

return 0;

}

Output:

****

**Experiment-2**

* Program for linear search in an array:

#include <iostream>

using namespace std;

int linearSearch(int arr[],int N, int key){

for(int i=0; i<N;i++){

if(arr[i]==key){

return i;

}

}

return -1;

}

int main(){

int N,key;

cin>>N;

int arr[N];

for(int i=0;i<N;i++){

cin>>arr[i];

}

cin>>key;

cout<<linearSearch(arr,N, key);

return 0;

}

**Output:**



**Experiment-3**

* Program for binary search in an array:

#include <iostream>

using namespace std;

int binarySearch(int array[],int n, int key){

int s=0;

int e=n;

while(s<=e){

int mid=(s+e)/2;

if(array[mid]==key){

return mid;

}

else if(array[mid]>key){

e=mid-1;

}

else{

s=mid+1;

}

}

return -1;

}

int main(){

int n;

cin>>n;

int array[n];

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

cin>>array[i];

}

int key;

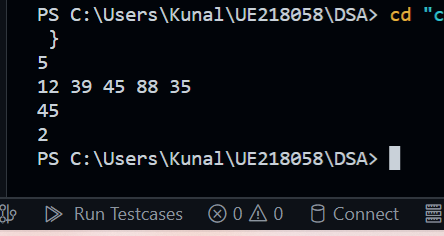
cin>>key;

cout<<binarySearch(array, n, key)<<endl;

return 0;

}

**Output:**



**Experiment- 4**

* **Program for implementing stack using array:**

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void push(int val){

if(top>=n-1){

cout<<"stack overflow"<<endl;

}

else{

top++;

stack[top]=val;

}

}

void pop(){

if(top<=-1){

cout<<"stack underflow"<<endl;

}else{

cout<<"the poped element is"<<stack[top]<<endl;

top--;

}

}

void display(){

if(top>=0){

cout<<"the elements are: ";

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

cout<<stack[i]<<" ";

}cout<<endl;

}else{

cout<<"stack is empty"<<endl;

}

}

int main(){

int ch, val;

cout<<"1 for push"<<endl;

cout<<"2 for pop"<<endl;

cout<<"3 for display"<<endl;

cout<<"4 for exit"<<endl;

do {

cout<<"Enter choice: "<<endl;

cin>>ch;

switch(ch) {

case 1: {

cout<<"Enter value to be pushed: ";

cin>>val;

push(val);

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"Exit"<<endl;

break;

}

default: {

cout<<"Invalid Choice"<<endl;

}

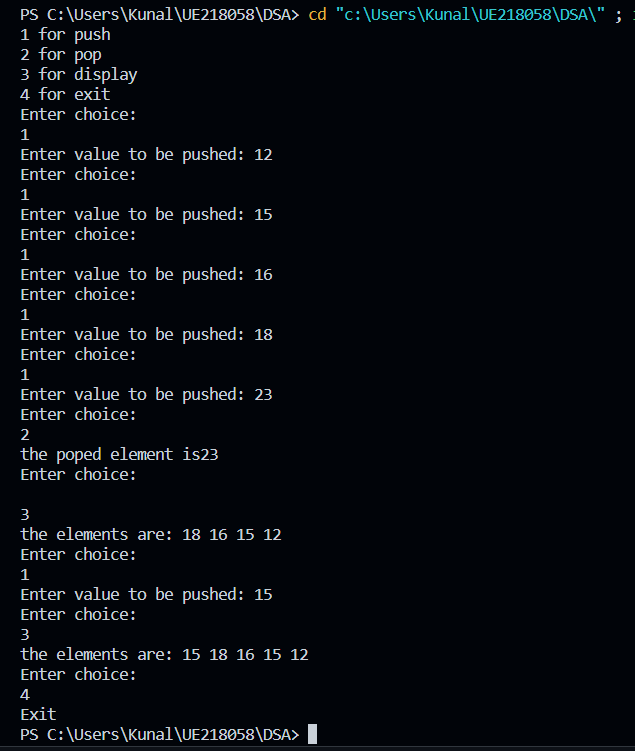
}

}while(ch!=4);

return 0;

}

**Output:**



**Experiment-5**

* **Program for implementing queue using array:**

#include <iostream>

using namespace std;

int queue[100], n = 100, front = - 1, rear = - 1;

void Insert() {

int val;

if (rear == n - 1)

cout<<"Queue Overflow"<<endl;

else {

if (front == - 1)

front = 0;

cout<<"Insert the element in queue : "<<endl;

cin>>val;

rear++;

queue[rear] = val;

}

}

void Delete() {

if (front == - 1 || front > rear) {

cout<<"Queue Underflow ";

return ;

} else {

cout<<"Element deleted from queue is : "<< queue[front] <<endl;

front++;;

}

}

void Display() {

if (front == - 1)

cout<<"Queue is empty"<<endl;

else {

cout<<"Queue elements are : ";

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

cout<<queue[i]<<" ";

cout<<endl;

}

}

int main() {

int ch;

cout<<"1) Insert element to queue"<<endl;

cout<<"2) Delete element from queue"<<endl;

cout<<"3) Display all the elements of queue"<<endl;

cout<<"4) Exit"<<endl;

do {

cout<<"Enter your choice : "<<endl;

cin>>ch;

switch (ch) {

case 1: Insert();

break;

case 2: Delete();

break;

case 3: Display();

break;

case 4: cout<<"Exit"<<endl;

break;

default: cout<<"Invalid choice"<<endl;

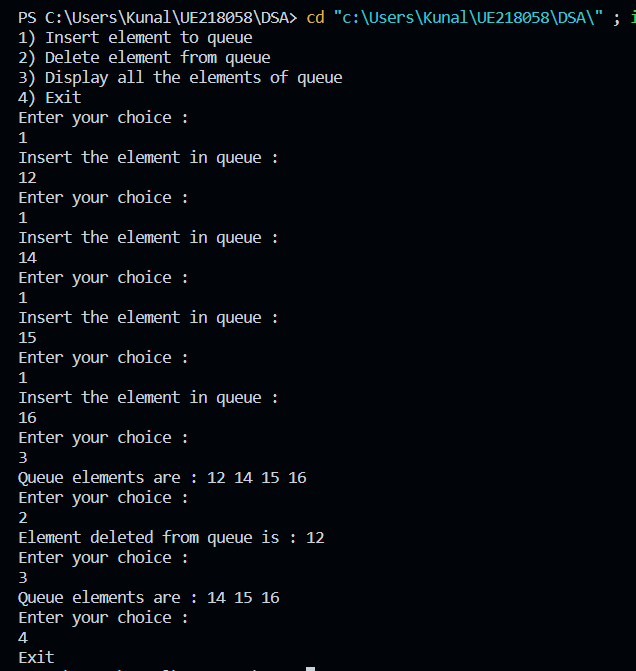
}

} while(ch!=4);

return 0;

}

**Output:**

****

**Experiment: 6**

* **Program for infix to postfix conversion using stack:**

#include <bits/stdc++.h>

using namespace std;

int prec(char c)

{

if (c == '^')

return 3;

else if (c == '/' || c == '\*')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1;

}

void infixToPostfix(string s)

{

stack<char> st;

string result;

for (int i = 0; i < s.length(); i++) {

char c = s[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')

|| (c >= '0' && c <= '9'))

result += c;

else if (c == '(')

st.push('(');

else if (c == ')') {

while (st.top() != '(') {

result += st.top();

st.pop();

}

st.pop();

}

else {

while (!st.empty()

&& prec(s[i]) <= prec(st.top())) {

result += st.top();

st.pop();

}

st.push(c);

}

}

while (!st.empty()) {

result += st.top();

st.pop();

}

cout << result << endl;

}

int main()

{

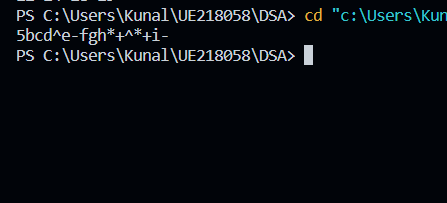
string exp = "5+b\*(c^d-e)^(f+g\*h)-i";

infixToPostfix(exp);

return 0;

}

**Output:**



**Experiment- 7**

* **Program for evaluating arithmetic expression using stack**

#include<iostream>

#include<cmath>

#include<stack>

using namespace std;

float scanNum(char ch) {

int value;

value = ch;

return float(value-'0'); //return float from character

}

int isOperator(char ch) {

if(ch == '+'|| ch == '-'|| ch == '\*'|| ch == '/' || ch == '^')

return 1; //character is an operator

return -1; //not an operator

}

int isOperand(char ch) {

if(ch >= '0' && ch <= '9')

return 1; //character is an operand

return -1; //not an operand

}

float operation(int a, int b, char op) {

//Perform operation

if(op == '+')

return b+a;

else if(op == '-')

return b-a;

else if(op == '\*')

return b\*a;

else if(op == '/')

return b/a;

else if(op == '^')

return pow(b,a); //find b^a

else

return INT\_MIN; //return negative infinity

}

float postfixEval(string postfix) {

int a, b;

stack<float> stk;

string::iterator it;

for(it=postfix.begin(); it!=postfix.end(); it++) {

//read elements and perform postfix evaluation

if(isOperator(\*it) != -1) {

a = stk.top();

stk.pop();

b = stk.top();

stk.pop();

stk.push(operation(a, b, \*it));

}else if(isOperand(\*it) > 0) {

stk.push(scanNum(\*it));

}

}

return stk.top();

}

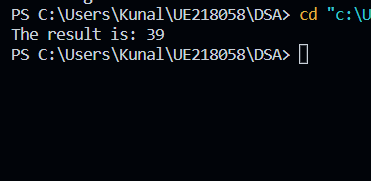
int main() {

string post = "53+62/\*35\*+";

cout << "The result is: "<<postfixEval(post);

}

**Output:**

****

**Experiment- 8**

* **Program for implementing a linked list**

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node \*next;

};

struct Node\* head = NULL;

void insert(int new\_data) {

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = head;

head = new\_node;

}

void display() {

struct Node\* ptr;

ptr = head;

while (ptr != NULL) {

cout<< ptr->data <<" ";

ptr = ptr->next;

}

}

int main() {

insert(3);

insert(1);

insert(7);

insert(2);

insert(9);

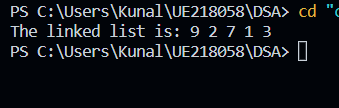
cout<<"The linked list is: ";

display();

return 0;

}

**Output:**

****

**Experiment- 9**

* **Program to reverse a linked list**

#include <bits/stdc++.h>

using namespace std;

Struct Node {

int data;

struct Node\* next;

Node(int data)

{

this->data = data;

next = NULL;

}

};

struct LinkedList {

Node\* head;

LinkedList() { head = NULL; }

void reverse()

{

Node\* current = head;

Node \*prev = NULL, \*next = NULL;

while (current != NULL) {

// Store next

next = current->next;

// Reverse current node's pointer

current->next = prev;

// Move pointers one position ahead.

prev = current;

current = next;

}

head = prev;

}

void print()

{

struct Node\* temp = head;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

}

void push(int data)

{

Node\* temp = new Node(data);

temp->next = head;

head = temp;

}

};

int main()

{

LinkedList ll;

ll.push(99);

ll.push(67);

ll.push(13);

ll.push(68);

ll.push(34);

ll.push(25);

ll.push(56);

cout << "Given linked list\n";

ll.print();

ll.reverse();

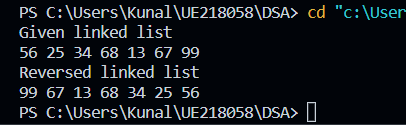
cout << "\nReversed linked list \n";

ll.print();

return 0;

}

**Output:**

****

**Experiment- 10**

* **Program to detect and remove a loop in the linked list**

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

struct Node\* next;

};

void removeLoop(struct Node\*, struct Node\*);

int detectAndRemoveLoop(struct Node\* list)

{

struct Node \*slow\_p = list, \*fast\_p = list;

while (slow\_p && fast\_p && fast\_p->next) {

slow\_p = slow\_p->next;

fast\_p = fast\_p->next->next;

if (slow\_p == fast\_p) {

removeLoop(slow\_p, list);

return 1;

}

}

return 0;

}

void removeLoop(struct Node\* loop\_node, struct Node\* head)

{

struct Node\* ptr1 = loop\_node;

struct Node\* ptr2 = loop\_node;

unsigned int k = 1, i;

while (ptr1->next != ptr2) {

ptr1 = ptr1->next;

k++;

}

ptr1 = head;

ptr2 = head;

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

ptr2 = ptr2->next;

while (ptr2 != ptr1) {

ptr1 = ptr1->next;

ptr2 = ptr2->next;

}

while (ptr2->next != ptr1)

ptr2 = ptr2->next;

ptr2->next = NULL;

}

void printList(struct Node\* node)

{

while (node != NULL) {

cout << node->data << " ";

node = node->next;

}

}

struct Node\* newNode(int key)

{

struct Node\* temp = new Node();

temp->data = key;

temp->next = NULL;

return temp;

}

int main()

{

struct Node\* head = newNode(50);

head->next = newNode(20);

head->next->next = newNode(15);

head->next->next->next = newNode(4);

head->next->next->next->next = newNode(10);

head->next->next->next->next->next = head->next->next;

detectAndRemoveLoop(head);

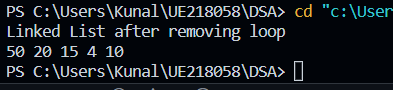
cout << "Linked List after removing loop \n";

printList(head);

return 0;

}

**Output:**

****

**Experiment- 11**

* **Program to split the circular linked list into two halves**

#include <bits/stdc++.h>

using namespace std;

class Node

{

public:

int data;

Node \*next;

};

void splitList(Node \*head, Node \*\*head1\_ref,

Node \*\*head2\_ref)

{

Node \*slow\_ptr = head;

Node \*fast\_ptr = head;

if(head == NULL)

return;

while(fast\_ptr->next != head &&

fast\_ptr->next->next != head)

{

fast\_ptr = fast\_ptr->next->next;

slow\_ptr = slow\_ptr->next;

}

if(fast\_ptr->next->next == head)

fast\_ptr = fast\_ptr->next;

\*head1\_ref = head;

if(head->next != head)

\*head2\_ref = slow\_ptr->next;

fast\_ptr->next = slow\_ptr->next;

slow\_ptr->next = head;

}

void push(Node \*\*head\_ref, int data)

{

Node \*ptr1 = new Node();

Node \*temp = \*head\_ref;

ptr1->data = data;

ptr1->next = \*head\_ref;

if(\*head\_ref != NULL)

{

while(temp->next != \*head\_ref)

temp = temp->next;

temp->next = ptr1;

}

else

ptr1->next = ptr1;

\*head\_ref = ptr1;

}

void printList(Node \*head)

{

Node \*temp = head;

if(head != NULL)

{

cout << endl;

do {

cout << temp->data << " ";

temp = temp->next;

} while(temp != head);

}

}

int main()

{

int list\_size, i;

Node \*head = NULL;

Node \*head1 = NULL;

Node \*head2 = NULL;

push(&head, 46);

push(&head, 56);

push(&head, 98);

push(&head, 81);

push(&head, 24);

cout << "Original Circular Linked List";

printList(head);

splitList(head, &head1, &head2);

cout << "\nFirst Circular Linked List";

printList(head1);

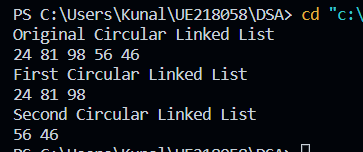
cout << "\nSecond Circular Linked List";

printList(head2);

return 0;

}

**Output:**

****

**Experiment- 12**

* **Program for deleting a node from the doubly linked list**

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

void deleteNode(struct Node\*\* head\_ref, struct Node\* del)

{

if (\*head\_ref == NULL || del == NULL)

return;

if (\*head\_ref == del)

\*head\_ref = del->next;

if (del->next != NULL)

del->next->prev = del->prev;

if (del->prev != NULL)

del->prev->next = del->next;

free(del);

}

void deleteNodeAtGivenPos(struct Node\*\* head\_ref, int n)

{

if (\*head\_ref == NULL || n <= 0)

return;

struct Node\* current = \*head\_ref;

int i;

for (int i = 1; current != NULL && i < n; i++)

current = current->next;

if (current == NULL)

return;

deleteNode(head\_ref, current);

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node =

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

new\_node->data = new\_data;

new\_node->prev = NULL;

new\_node->next = (\*head\_ref);

if ((\*head\_ref) != NULL)

(\*head\_ref)->prev = new\_node;

(\*head\_ref) = new\_node;

}

void printList(struct Node\* head)

{

while (head != NULL) {

cout << head->data << " ";

head = head->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 5);

push(&head, 2);

push(&head, 4);

push(&head, 8);

push(&head, 10);

cout << "Doubly linked list before deletion:n";

printList(head);

int n = 2;

deleteNodeAtGivenPos(&head, n);

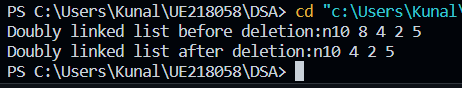
cout << "\nDoubly linked list after deletion:n";

printList(head);

return 0;

}

**Output:**

****

**Experiment- 13**

* **Program for implementing stack and queue using a linked list**

#include <bits/stdc++.h>

using namespace std;

struct QNode {

int data;

QNode\* next;

QNode(int d)

{

data = d;

next = NULL;

}

};

struct Queue {

QNode \*front, \*rear;

Queue() { front = rear = NULL; }

void enQueue(int x)

{

QNode\* temp = new QNode(x);

if (rear == NULL) {

front = rear = temp;

return;

}

rear->next = temp;

rear = temp;

}

void deQueue()

{

if (front == NULL)

return;

QNode\* temp = front;

front = front->next;

if (front == NULL)

rear = NULL;

delete (temp);

}

};

int main()

{

Queue q;

q.enQueue(10);

q.enQueue(20);

q.deQueue();

q.deQueue();

q.enQueue(30);

q.enQueue(40);

q.enQueue(50);

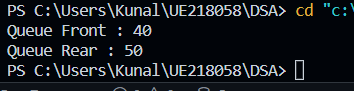
q.deQueue();

cout << "Queue Front : " << (q.front)->data << endl;

cout << "Queue Rear : " << (q.rear)->data;

}

**Output:**

****

**Experiment- 14**

* **Program to find the kth largest/smallest element in an unsorted array**

#include <stdio.h>

#include <stdlib.h>

int cmpfunc(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

int kthSmallest(int arr[], int N, int K)

{

qsort(arr, N, sizeof(int), cmpfunc);

return arr[K - 1];

}

int main()

{

int arr[] = { 12, 3, 5, 7, 19 };

int N = sizeof(arr) / sizeof(arr[0]), K = 2;

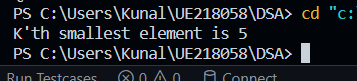
printf("K'th smallest element is %d",

kthSmallest(arr, N, K));

return 0;

}

**Output:**

****

* **Program for kth largest element in unsorted array**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

// Function that returns the Kth largest element

int kth\_largest\_element(int arr[], int k, int n){

// Sorts the array

sort(arr, arr + n);

// Reverses the array

reverse(arr, arr+n);

// Returns the required element

return arr[k-1];

}

int main(){

// Given array

int arr[] = {12, 15, 7, 3, 8, 16, 25};

// n represents the size of the array

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

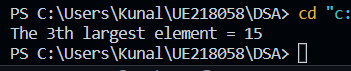
int k = 3;

cout << "The "<< k << "th largest element = " << kth\_largest\_element(arr, k, n) << endl;

return 0;

}

**Output:**

****

**Experiment- 15**

* **Program to implement two stacks in an array**

#include <bits/stdc++.h>

using namespace std;

class twoStacks {

int\* arr;

int size;

int top1, top2;

public:

twoStacks(int n)

{

size = n;

arr = new int[n];

top1 = n / 2 + 1;

top2 = n / 2;

}

void push1(int x)

{

if (top1 > 0) {

top1--;

arr[top1] = x;

}

else {

cout << "Stack Overflow"

<< " By element : " << x << endl;

return;

}

}

void push2(int x)

{

if (top2 < size - 1) {

top2++;

arr[top2] = x;

}

else {

cout << "Stack Overflow"

<< " By element : " << x << endl;

return;

}

}

int pop1()

{

if (top1 <= size / 2) {

int x = arr[top1];

top1++;

return x;

}

else {

cout << "Stack UnderFlow";

exit(1);

}

}

int pop2()

{

if (top2 >= size / 2 + 1) {

int x = arr[top2];

top2--;

return x;

}

else {

cout << "Stack UnderFlow" << endl;

exit(1);

}

}

};

int main()

{

twoStacks ts(5);

ts.push1(5);

ts.push2(10);

ts.push2(15);

ts.push1(11);

ts.push2(7);

cout << "Popped element from stack1 is "

<< ": " << ts.pop1() << endl;

ts.push2(40);

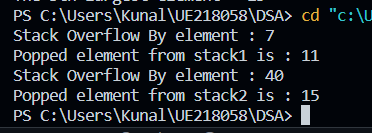
cout << "Popped element from stack2 is "

<< ": " << ts.pop2() << endl;

return 0;

}

**Output:**



**Experiment-16**

* **Program for implementing binary search tree:**

#include <iostream>

using namespace std;

class BST {

int data;

BST \*left, \*right;

public:

// Default constructor.

BST();

// Parameterized constructor.

BST(int);

// Insert function.

BST\* Insert(BST\*, int);

// Inorder traversal.

void Inorder(BST\*);

};

// Default Constructor definition.

BST ::BST()

: data(0)

, left(NULL)

, right(NULL)

{

}

// Parameterized Constructor definition.

BST ::BST(int value)

{

data = value;

left = right = NULL;

}

// Insert function definition.

BST\* BST ::Insert(BST\* root, int value)

{

if (!root) {

// Insert the first node, if root is NULL.

return new BST(value);

}

// Insert data.

if (value > root->data) {

// Insert right node data, if the 'value'

// to be inserted is greater than 'root' node data.

// Process right nodes.

root->right = Insert(root->right, value);

}

else if (value < root->data){

// Insert left node data, if the 'value'

// to be inserted is smaller than 'root' node data.

// Process left nodes.

root->left = Insert(root->left, value);

}

// Return 'root' node, after insertion.

return root;

}

// Inorder traversal function.

// This gives data in sorted order.

void BST ::Inorder(BST\* root)

{

if (!root) {

return;

}

Inorder(root->left);

cout << root->data << endl;

Inorder(root->right);

}

// Driver code

int main()

{

BST b, \*root = NULL;

root = b.Insert(root, 50);

b.Insert(root, 30);

b.Insert(root, 20);

b.Insert(root, 40);

b.Insert(root, 70);

b.Insert(root, 60);

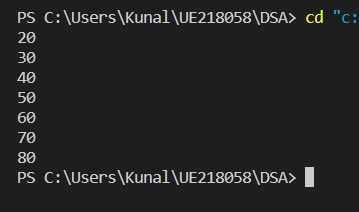
b.Insert(root, 80);

b.Inorder(root);

return 0;

}

**Output:**

****

**Experiment-17**

* **Program for implementing heap sort:**

// C++ program for implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i

// which is an index in arr[].

// n is size of heap

void heapify(int arr[], int N, int i)

{

// Initialize largest as root

int largest = i;

// left = 2\*i + 1

int l = 2 \* i + 1;

// right = 2\*i + 2

int r = 2 \* i + 2;

// If left child is larger than root

if (l < N && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest

// so far

if (r < N && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

swap(arr[i], arr[largest]);

// Recursively heapify the affected

// sub-tree

heapify(arr, N, largest);

}

}

// Main function to do heap sort

void heapSort(int arr[], int N)

{

// Build heap (rearrange array)

for (int i = N / 2 - 1; i >= 0; i--)

heapify(arr, N, i);

// One by one extract an element

// from heap

for (int i = N - 1; i > 0; i--) {

// Move current root to end

swap(arr[0], arr[i]);

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

// A utility function to print array of size n

void printArray(int arr[], int N)

{

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

cout << arr[i] << " ";

cout << "\n";

}

// Driver's code

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

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

// Function call

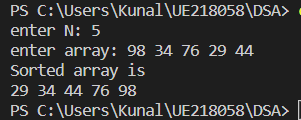
heapSort(arr, N);

cout << "Sorted array is \n";

printArray(arr, N);

}

**Output:**

****

**Experiment-18**

* **Program for implementing bubble sort:**

#include <iostream>

using namespace std;

int main()

{

int n;

cin>>n;

int array[n];

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

{

cin>>array[i];

}

int counter=1;

while(counter<n)

{

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

{

if(array[i]>array[i+1])

{

int temp=array[i];

array[i]=array[i+1];

array[i+1]=temp;

}

}

counter++;

}

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

{

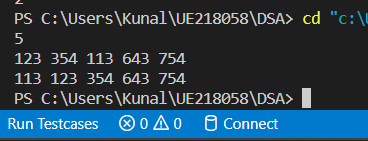
cout<<array[i]<<" ";

}

cout<<endl;

}

**Output:**

****

**Experiment-19**

* **Program for implementing merge sort:**

#include <iostream>

using namespace std;

void merge(int array[], int const left, int const mid,

int const right)

{

auto const subArrayOne = mid - left + 1;

auto const subArrayTwo = right - mid;

// Create temp arrays

auto \*leftArray = new int[subArrayOne],

\*rightArray = new int[subArrayTwo];

// Copy data to temp arrays leftArray[] and rightArray[]

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne

= 0, // Initial index of first sub-array

indexOfSubArrayTwo

= 0; // Initial index of second sub-array

int indexOfMergedArray

= left; // Initial index of merged array

// Merge the temp arrays back into array[left..right]

while (indexOfSubArrayOne < subArrayOne

&& indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne]

<= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

// Copy the remaining elements of

// right[], if there are any

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

delete[] leftArray;

delete[] rightArray;

}

void mergeSort(int array[], int const begin, int const end)

{

if (begin >= end)

return; // Returns recursively

auto mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

void printArray(int A[], int size)

{

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

cout << A[i] << " ";

}

int main()

{int arr\_size;

cout<<"enter arr\_size: ";

cin>>arr\_size;

int arr[arr\_size];

cout<<"enter array: ";

for(int i=0;i<arr\_size;i++){

cin>>arr[i];

}

cout << "Given array is \n";

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

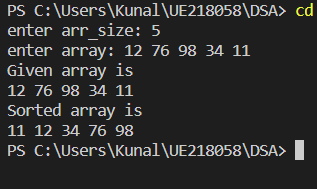
cout << "\nSorted array is \n";

printArray(arr, arr\_size);

return 0;

}

**Output:**

****

**Experiment-20**

* **Program for implementing quick sort:**

#include <bits/stdc++.h>

using namespace std;

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

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

{

int pivot = arr[high]; // pivot

int i

= (low

- 1);

for (int j = low; j <= high - 1; j++) {

// If current element is smaller than the pivot

if (arr[j] < pivot) {

i++; // increment index of smaller element

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

}

}

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

return (i + 1);

}

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

{

if (low < high) {

/\* pi is partitioning index, arr[p] is now

at right place \*/

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

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

cout << arr[i] << " ";

cout << endl;

}

int main()

{int N;

cout<<"enter N: ";

cin>>N;

int arr[N];

cout<<"enter array: ";

for(int i=0;i<N;i++){

cin>>arr[i];

}

quickSort(arr, 0, N - 1);

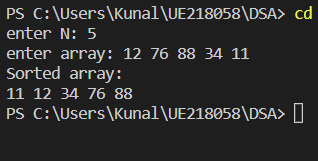
cout << "Sorted array: \n";

printArray(arr, N);

return 0;

}

**Output:**



**University Institute of Engineering and Technology**

**Panjab University, Chandigarh**



Data Structure Practical File

Submitted To: Submitted By:

*Dr. Preetjot Kaur Kunal*

Dept. of Information UE218058

Technology, UIET B.E. 2nd Year- 3rd sem

Panjab University, Information Technology

Chandigarh UIET, Panjab University

Chandigarh

**INDEX**

***Contents***

1.WAP in insert and delete the elements of an array. 11/08/22

2.WAP to find an element using Linear search in an array. 25/08/22

3.WAP to find an element using Binary search in an array. 25/08/22

4.WAP to implement stack using array. 01/09/22

5.WAP to implement Queue using array. 01/09/22

6.WAP to convert infix expression to postfix expression. 08/09/22

7.WAP to evaluate an arithmetic expression using stack. 15/09/22

8.WAP to implement a linked list. 15/09/22

9.WAP to reverse a linked list. 22/09/22

10.WAP to detect and remove a loop in the linked list. 22/09/22

11.WAP to split the circular linked list into two halves. 13/10/22

12.WAP to delete a node from the doubly linked list. 13/10/22

13.WAP to implement stack and queue using a Linked list. 20/10/22

14.WAP to find the kth largest/smallest element in an

unsorted array . 27/10/22

15.WAP to implement two stacks in an array. 03/11/22

16.Write a program to create binary search tree. 03/11/22

17.WAP for heap sort. 10/11/22

18.WAP for bubble sort. 10/11/22

19.WAP for merge sort. 17/11/22

20.WAP for quick sort. 17/11/22