#include <stdio.h>

#include <stdlib.h>

// Define a structure for a doubly linked list node

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(int value) {

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

if (newNode == NULL) {

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

exit(1);

}

newNode->data = value;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the list

struct Node\* insertAtBeginning(struct Node\* head, int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

return newNode;

}

newNode->next = head;

head->prev = newNode;

return newNode;

}

// Function to insert a node at the end of the list

struct Node\* insertAtEnd(struct Node\* head, int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

return newNode;

}

struct Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

return head;

}

// Function to insert a node at a specified position in the list

struct Node\* insertAtPosition(struct Node\* head, int value, int position) {

struct Node\* newNode = createNode(value);

if (position == 1) {

newNode->next = head;

if (head != NULL) {

head->prev = newNode;

}

return newNode;

}

struct Node\* current = head;

int i;

for (i = 1; i < position - 1 && current != NULL; i++) {

current = current->next;

}

if (current == NULL) {

printf("Invalid position.\n");

return head;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != NULL) {

current->next->prev = newNode;

}

current->next = newNode;

return head;

}

// Function to delete a node at a specified position in the list

struct Node\* deleteAtPosition(struct Node\* head, int position) {

if (head == NULL) {

printf("List is empty. Cannot delete.\n");

return NULL;

}

if (position == 1) {

struct Node\* newHead = head->next;

free(head);

if (newHead != NULL) {

newHead->prev = NULL;

}

return newHead;

}

struct Node\* current = head;

int i;

for (i = 1; i < position && current != NULL; i++) {

current = current->next;

}

if (current == NULL) {

printf("Invalid position. Cannot delete.\n");

return head;

}

current->prev->next = current->next;

if (current->next != NULL) {

current->next->prev = current->prev;

}

free(current);

return head;

}

// Function to search for an element in the list

int searchElement(struct Node\* head, int value) {

struct Node\* current = head;

int position = 1;

while (current != NULL) {

if (current->data == value) {

return position;

}

current = current->next;

position++;

}

return -1; // Element not found

}

// Function to display the elements of the list

void displayList(struct Node\* head) {

struct Node\* current = head;

printf("Doubly Linked List: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

// Function to free the memory allocated for the list

void freeList(struct Node\* head) {

struct Node\* current = head;

struct Node\* next;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

}

int main() {

struct Node\* head = NULL;

int choice, value, position;

do {

printf("\n1. Insert at the beginning\n");

printf("2. Insert at the end\n");

printf("3. Insert at a position\n");

printf("4. Delete at a position\n");

printf("5. Search for an element\n");

printf("6. Display the list\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

head = insertAtBeginning(head, value);

break;

case 2:

printf("Enter the value to insert: ");

scanf("%d", &value);

head = insertAtEnd(head, value);

break;

case 3:

printf("Enter the value to insert: ");

scanf("%d", &value);

printf("Enter the position: ");

scanf("%d", &position);

head = insertAtPosition(head, value, position);

break;

case 4:

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

scanf("%d", &position);

head = deleteAtPosition(head, position);

break;

case 5:

printf("Enter the element to search: ");

scanf("%d", &value);

position = searchElement(head, value);

if (position != -1) {

printf("Element found at position %d\n", position);

} else {

printf("Element not found in the list\n");

}

break;

case 6:

displayList(head);

break;

case 7:

freeList(head);

printf("Program exiting.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 7);

return 0;

}

Inserting a element at the beginning of the linked list

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

struct node\*next;

};

struct node\*head=NULL;

void insert(int new\_data)

{

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

newnode->data=new\_data;

newnode->next=head;

head=newnode;

}

void display()

{struct node\*ptr;

ptr=head;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

void insertbeg(int item)

{

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

newnode->data=item;

newnode->next=head;

head=newnode;

}

int main()

{

insert(3);

insert(1);

insert(7);

insert(2);

insert(9);

cout<<"The linked list is :";

display();

int ch,item;

do

{

cout<<"\n Enter the new item to insert \n";

cin>>item;

insertbeg(item);

cout<<"press 0 to add more elements\n";

cin>>ch;

}

while(ch==0);

display();

return(0);

}

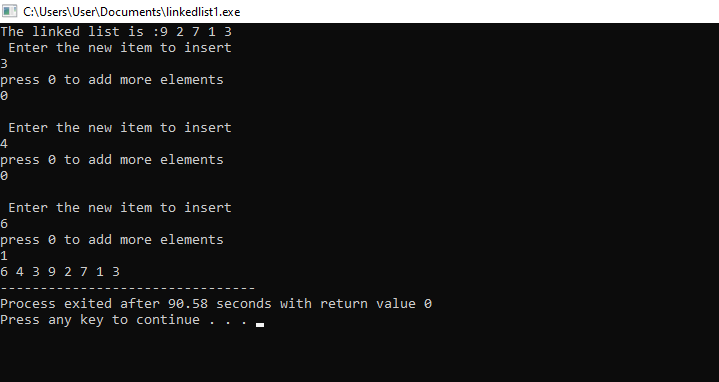
insert(3);

insert(1);

insert(7);

insert(2);

insert(9);



linked list program to insert a node at the end of the list

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

struct node\*next;

};

struct node\*head=NULL;

void insert(int new\_data)

{

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

newnode->data=new\_data;

newnode->next=head;

head=newnode;

}

void display()

{

struct node\*ptr;

ptr=head;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

void insertbeg(int item)

{

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

newnode->data=item;

newnode->next=head;

head=newnode;

}

int main()

{

insert(3);

insert(1);

insert(7);

insert(2);

insert(9);

cout<<"The linked list is :";

display();

int ch,item;

do

{

cout<<"\n Enter the new item to insert \n";

cin>>item;

insertbeg(item);

cout<<"press 0 to add more elements\n";

cin>>ch;

}

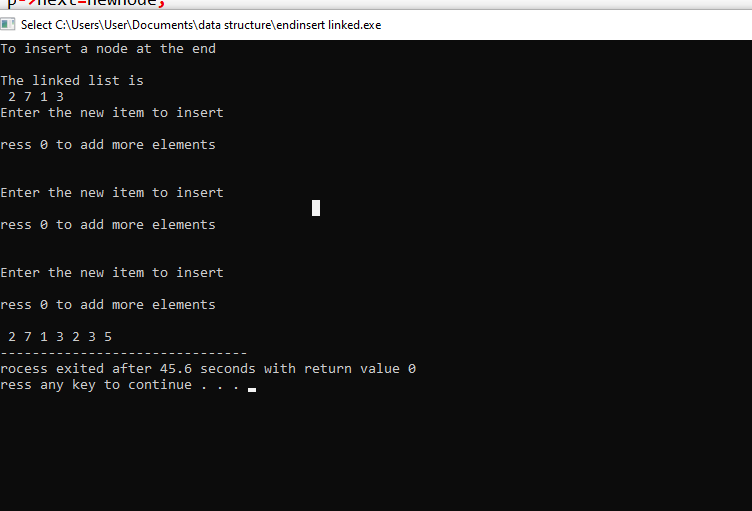
while(ch==0);

display();

return(0);

}

output:



linked list program to insert a node at the the given position

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

struct node\*next;

};

struct node\*head=NULL;

void insert(int new\_data)

{

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

newnode->data=new\_data;

newnode->next=head;

head=newnode;

}

void display()

{

struct node\*ptr;

ptr=head;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

void insertbeg(int item)

{

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

newnode->data=item;

newnode->next=head;

head=newnode;

}

void insertend(int item)

{

struct node\*p;

p=head;

while(p->next!=NULL)

p=p->next;

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

newnode->data=item;

newnode->next=NULL;

p->next=newnode;

}

void insertpos(int item)

{

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

struct node\*p;

struct node\*p1;

struct node\*p2;

int count=1,pos=1;

p1=head;

while(p1->next!=NULL)

{

count=count+1;

p1=p1->next;

}

cout<<"Enter the position :";

cin>>pos;

if(pos==1)

insertbeg(item);

else if(pos==count+1)

insertbeg(item);

else if(pos<=count)

{

p1=head;

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

{

p=p1;

p1=p1->next;

}

p2=newnode;

p2->data=item;

p2->next=p1;

p->next=p2;

}

else

{

cout<<"invalid position"<<endl;

}

}

int main()

{

int ch,item;

insert(9);

insert(7);

insert(2);

insert(4);

insert(3);

cout<<"The elements in the linked list :";

display();

cout<<endl;

while(ch!=3)

{

cout<<"\n choice"<<endl<<"1.Insert"<<endl<<"2.display"<<endl<<"3.exit"<<endl;

cout<<"Enter your choice :";

cin>>ch;

switch(ch)

{

case 1:cout<<"Enter the elements :"<<endl;

cin>>item;

insertpos(item);

break;

case 2:display();

break;

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

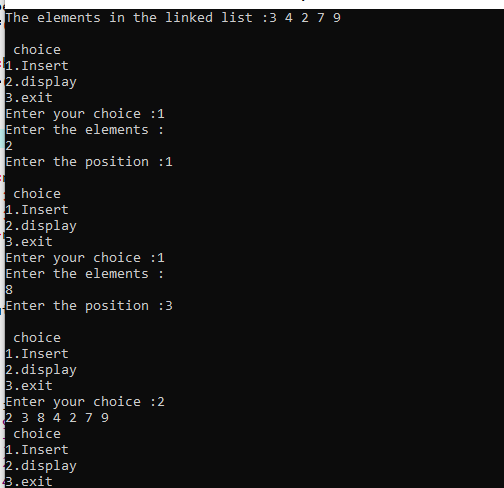
}

}

return(0);

}

output:



#include<iostream>

using namespace std;

struct Node{

int value;

struct Node \*next;

};

struct Node\*head=NULL;

struct Node\*sHead=NULL;

struct Node\*temp=NULL;

void insert(int new\_data){

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

new\_node->value=new\_data;

new\_node->next=head;

head=new\_node;

}

/\*int n;

int ele;

int splitIndex;\*/

int main(){

int i,n,ele,splitIndex;

cout<<"Enter number of elements you want in the list\t";

cin>>n;

cout<<"Enter elements:"<<endl;

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

cin>>ele;

insert(ele);

}

cout<<"\nList of elements:"<<endl;

Node \*t;

t=head;

while(t!=NULL){

cout<<t->value<<"\t";

t=t->next;

}

cout<<"\n\n Enter the position you want the list to split:";

cin>>splitIndex;

while(splitIndex<0||splitIndex>n-1){

cout<<"Invalid Position.Try Again!!"<<endl;

cin>>splitIndex;

}

temp=head;

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

if(i==splitIndex-1){

Node \*tN;

tN=temp->next;

sHead=tN;

temp->next=NULL;

break;

}

temp=temp->next;

}

temp=head;

if(temp==NULL){

cout<<"\nFirst list is empty"<<endl;

} else {

cout<<"\n\nFirst list element:"<<endl;

while(temp!=NULL){

cout<<temp->value<<"\t";

temp=temp->next;

}

}

temp=sHead;

if(temp==NULL){

cout<<"\nSecond list is empty"<<endl;

}else{

cout<<"\n\nSecond list elements "<<endl;

while(temp != NULL){

cout<<temp->value<<"\t";

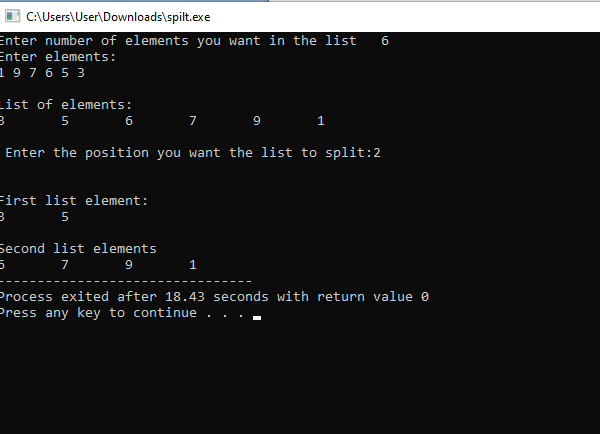
temp = temp->next;

}

}

return 0;

}



PROGRAM TO HEAP SORT USING MIN HEAP

#include<iostream>

#include<conio.h>

using namespace std;

void min\_heap(int\*a,int m,int n)

{

int j,t;

t=a[m];

j=2\*m;

while(j<=n)

{

if(j<n && a[j+1]<a[j])

j=j+1;

if(t<a[j])

break;

else if(t>=a[j])

{

a[j/2]=a[j];

j=2\*j;

}

}

a[j/2]=t;

return;

}

void build\_minheap(int\*a,int n)

{

int k;

for(k=n/2;k>=1;k--)

{

min\_heap(a,k,n);

}

}

int main()

{

int n,i;

cout<<"Enter the number of array\n";

cin>>n;

int a[30];

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

{

cout<<"enter element"<<""<<(i)<<endl;

cin>>a[i];

}

build\_minheap(a,n);

cout<<"Min Heap\n";

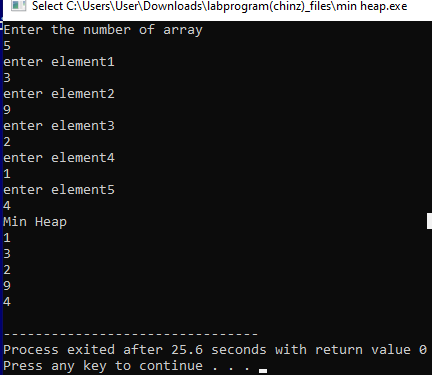
for(i=1;i<=n;i++)

{

cout<<a[i]<<endl;

}

}



#include<iostream>  
using namespace std;  
int s[10],d,n,set[10],count=0;  
void display(int);  
int flag = 0;  
int main()  
{  
int subset(int,int);  
int i;  
cout<<"ENTER THE NUMBER OF THE ELEMENTS IN THE SET : ";  
cin>>n;  
cout<<"ENTER THE SET OF VALUES : ";  
for(i=0;i<n;i++)  
 cin>>s[i];  
cout<<"ENTER THE SUM : ";  
cin>>d;  
cout<<"THE PROGRAM OUTPUT IS: ";  
subset(0,0);  
if(flag == 0)  
cout<<"There is no solution";  
}  
int subset(int sum,int i)  
{  
if(sum == d)  
{  
flag = 1;  
display(count);  
return 0;  
}  
if(sum>d || i>=n)  
return 1;  
else  
{  
set[count]=s[i];  
count++;  
subset(sum+s[i],i+1);  
count--;  
subset(sum,i+1);  
}  
}  
void display(int count)  
{  
int i;  
cout<<"{";  
for(i=0;i<count;i++)  
cout<<set[i]<<",";  
cout<<"}";  
}

|  |  |
| --- | --- |
|  |  |

#include<iostream>  
using namespace std;  
int grid[10][10];  
void print(int n) {  
for (int i = 0;i <= n-1; i++) {  
for (int j = 0;j <= n-1; j++) {  
cout <<grid[i][j]<< " ";  
}  
cout<<endl;  
}  
cout<<endl;  
cout<<endl;  
}  
bool isSafe(int col, int row, int n) {  
for (int i = 0; i < row; i++) {  
if (grid[i][col]) {  
return false;  
}  
}  
for (int i = row,j = col;i >= 0 && j >= 0; i--,j--) {  
if (grid[i][j]) {  
return false;  
}  
}  
for (int i = row, j = col; i >= 0 && j < n; j++, i--) {  
if (grid[i][j]) {  
return false;  
}  
}  
return true;  
}  
bool solve (int n, int row) {  
if (n == row) {  
print(n);  
return true;  
}  
bool res = false;  
for (int i = 0;i <=n-1;i++) {  
if (isSafe(i, row, n))  
{  
grid[row][i] = 1;  
res = solve(n, row+1) || res;  
grid[row][i] = 0; }}  
return res; }  
int main() {  
ios\_base::sync\_with\_stdio(false);  
cin.tie(NULL);  
int n;  
cout<<"Enter the number of queen"<<endl;  
cin >> n;  
for (int i = 0;i < n;i++) {  
for (int j = 0;j < n;j++) {  
grid[i][j] = 0; }}  
bool res = solve(n, 0);  
if(res == false) {  
cout <<  
-1 << endl;  
} else {  
cout << endl; }  
return 0;  
 }

|  |  |
| --- | --- |
|  |  |