A picture containing graphical user interface

Description automatically generated

**SUBJECT CODE-** CSC 2152

**SUBJECT NAME-** FOUNDATION OF DATA STRUCTURE LABORATORY

BTECH AI & DS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEC 2021

**NAME:** SHAIK MOHAMMAD ASRAR AHAMMAD

**RRN:** 200171601048

**SEMESTER & SECTION:** III, AI&DS

**A picture containing graphical user interface

Description automatically generated**

**BONAFIDE CERTIFICATE**

Certified that this is the bonafide record of the work done by **SHAIK MOHAMMAD ASRAR AHAMMAD**RRN **200171601048** of III semester B.Tech **ARTIFICIAL INTELLIGENCE & DATA SCIENCE** in the Foundations Of Data Structures Lab for the year 2021.

Course Faculty: Mrs.Niyati Behera Head of the department

Date:

Submitted for the practical examination held on ………………………

Register No:

Examiner:

A picture containing graphical user interface

Description automatically generated

**1) Course Title**: Foundations of Data Structures Lab **5) Semester:** III

**2) Course Code:** CSC 2155 **6) Academic year:** 2021 – 2022

**3) Course Faculty:** Mrs.Niyati Behera **7) Department:** B.Tech AI & DS

**4) Theory/Practical:** Laboratory **8) No. of Credits:** 1

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Title** | **Page No.** | **Faculty Signature** |
| **1.** | **03/09/2021** | **Single Linked List Implementation** |  |  |
| **2.** | **10/09/2021** | **Linked List Implementation of Stack ADT** |  |  |
| **3.** | **17/09/2021** | **Linked List Implementation of Queue ADT** |  |  |
| **4.** | **24/09/2021** | **Use recursion to generate Fibonacci series** |  |  |
| **5.** | **01/10/2021** | **Array Implementation of Binary tree** |  |  |
| **6.** | **01/11/2021** | **Implementation of Binary Tree Traversal** |  |  |
| **7.** | **15/11/2021** | **Binary search** |  |  |
| **8.** | **17/11/2021** | **Bubble Sorting** |  |  |
| **9.** | **22/11/2021** | **Selection Sorting** |  |  |
| **10.** | **24/11/2021** | **Implement Hash Table using Linear probing**  **and Quadratic probing** |  |  |

**1. Single Linked List Implementation**

**AIM :** To Implement Single Linked List program in c++ .

**ALGORITHM :**

* Allocate the space for the new node and store data into the data part of the node. This will be done by

the following statements.

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

p → data = item

* Make the link part of the new node pointing to the existing first node of the list. This will be done by using the following statement.

p->next = head;

* At the last, we need to make the new node as the first node of the list this will be done by using the

following statement.

* Step 1: IF PTR = NULL

Write overflow

Go to Step: 7

* Step 2: SET NEW\_NODE = PTR
* Step 3: SET PTR = PTR → NEXT
* Step 4: SET NEW\_NODE → DATA = VAL
* Step 5: SET NEW\_NODE → NEXT = HEAD
* Step 6: SET HEAD = NEW\_NODE
* Step 7: EXIT

**PROGRAM :**

#include<iostream>

using namespace std;

struct node{

int data;

struct node\* link;

};

struct node\* root=NULL;

void append(){

int val;

struct node\* temp;

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

struct node\* p;

cout<<"\nEnter value :";

cin>>val;

temp->data=val;

temp->link=NULL;

if(root==NULL){

root=temp;

}

else{

p=root;

while(p->link!=NULL){

p=p->link;

}

p->link=temp;

}

cout<<"\nItem pushed.";

}

void atbegin(){

struct node\* temp;

int val;

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

cout<<"\nEnter value :";

cin>>val;

temp->data=val;

if(root==NULL){

root=temp;

}

else{

temp->link=root;

root=temp;

}

cout<<"\nItem pushed.";

}

int length(){

int count=0;

struct node\* p;

p=root;

while(p->link!=NULL){

p=p->link;

count++;

}

return count;

}

void atnode(){

int loc,i=1;

int len=length();

struct node\* temp;

struct node\* p;

cout<<"\nEnter location :";

cin>>loc;

if(loc>len){

cout<<"\nThe value is more than lenght of list.";

}

else{

p=root;

while(loc>i){

p=p->link;

i++;

}

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

temp->link=p->link;

p->link=temp;

cout<<"\nEnter data :";

cin>>temp->data;

}

cout<<"\nItem pushed.";

}

void display(){

struct node\* p;

p=root;

if(p==NULL){

cout<<"\nThe list is empty.";

}

else{

while(p!=NULL){

cout<<p->data<<"->";

p=p->link;

}

cout<<"NULL";

}

}

void delete\_node(){

struct node\* temp;

int loc,i;

int len=length();

cout<<"\nEnter location :";

cin>>loc;

if(loc>len){

cout<<"\nInvalid location.";

}

else if(loc==1){

temp=root;

root=temp->link;

temp->link=NULL;

free(temp);

}

else{

struct node \*p,\*q;

p=root;

i=1;

while(loc-1>i){

p=p->link;

i++;

}

q=p->link;

p->link=q->link;

q->link=NULL;

free(q);

}

}

int main(){

int ch;

cout<<"\n1. Insert at beginning";

cout<<"\n2. Insert at middle";

cout<<"\n3. Insert at end";

cout<<"\n4. Display";

cout<<"\n5. Delete";

cout<<"\n6. Exit";

do{

cout<<"\nEnter choice :";

cin>>ch;

switch(ch){

case 1 :

atbegin();

break;

case 2:

atnode();

break;

case 3:

append();

break;

case 4 :

display();

break;

case 5 :

delete\_node();

break;

case 6 :

cout<<"\nSucessfully exited.";

break;

default :

cout<<"\nEnter a valid choice.";

break;

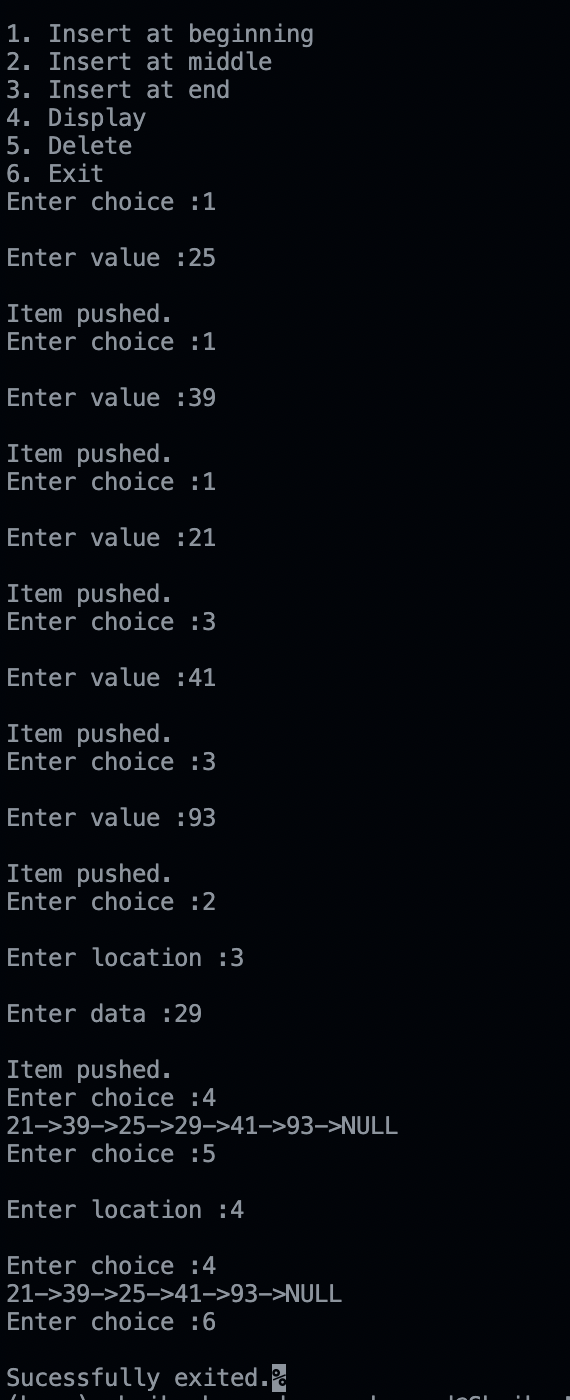
}

}while(ch!=6);

return 0;

}

**OUTPUT :**

****

**Result :**

**2. Linked List Implementation of Stack ADT**

**AIM :** To Implement Stack using Linked List program in c++ .

**ALGORITHM :**

* Step 1 - Include all the header files which are used in the program. And declare all the user defined
* functions.
* Step 2 - Define a 'Node' structure with two members data and next.
* Step 3 - Define a Node pointer 'top' and set it to NULL.
* Step 4 - Implement the main method by displaying Menu with list of operations and make suitable
* function calls in the main method.

**push(value) - Inserting an element into the Stack**

We can use the following steps to insert a new node into the stack...

* Step 1 - Create a newNode with given value.
* Step 2 - Check whether stack is Empty (top == NULL)
* Step 3 - If it is Empty, then set newNode → next = NULL.
* Step 4 - If it is Not Empty, then set newNode → next = top.
* Step 5 - Finally, set top = newNode.

**pop() - Deleting an Element from a Stack**

We can use the following steps to delete a node from the stack...

* Step 1 - Check whether stack is Empty (top == NULL).
* Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the

function

* Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
* Step 4 - Then set 'top = top → next'.
* Step 5 - Finally, delete 'temp'. (free(temp)).

**display() - Displaying stack of elements**

We can use the following steps to display the elements (nodes) of a stack...

* Step 1 - Check whether stack is Empty (top == NULL).
* Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
* Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
* Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches

to the first node in the stack. (temp → next != NULL).

* Step 5 - Finally! Display 'temp → data ---> NULL'.

**PROGRAM :**

#include<iostream>

using namespace std;

struct node{

int data;

struct node\*link;

};

struct node\* root=NULL;

void push(){

struct node\* temp;

int val;

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

cout<<"Enter value :";

cin>>val;

temp->data=val;

if(root==NULL){

root=temp;

}

else{

temp->link=root;

root=temp;

}

cout<<"\nItem pushed.";

}

void pop(){

int loc=1;

struct node\* temp;

if(root==NULL){

cout<<"The stack is empty.";

}

else{

temp=root;

cout<<"The popped element :"<<temp->data;

root=temp->link;

temp->link=NULL;

free(temp);

}

}

void display(){

struct node\* p;

p=root;

if(root==NULL){

cout<<"The stack is empty.";

}

else{

cout<<"The elements in stack :\n";

while(p!=NULL){

cout<<p->data<<"->";

p=p->link;

}

cout<<"NULL";

}

}

int main(){

int ch;

do{

cout<<"\n1. Push";

cout<<"\n2. Pop";

cout<<"\n3. Display";

cout<<"\n4. Exit";

cout<<"\nEnter choice :";

cin>>ch;

switch(ch){

case 1:

push();

break;

case 2 :

pop();

break;

case 3 :

display();

break;

case 4 :

cout<<"\n-----Successfully Exited-----\n";

break;

default :

cout<<"\nInvalid choice.";

break;

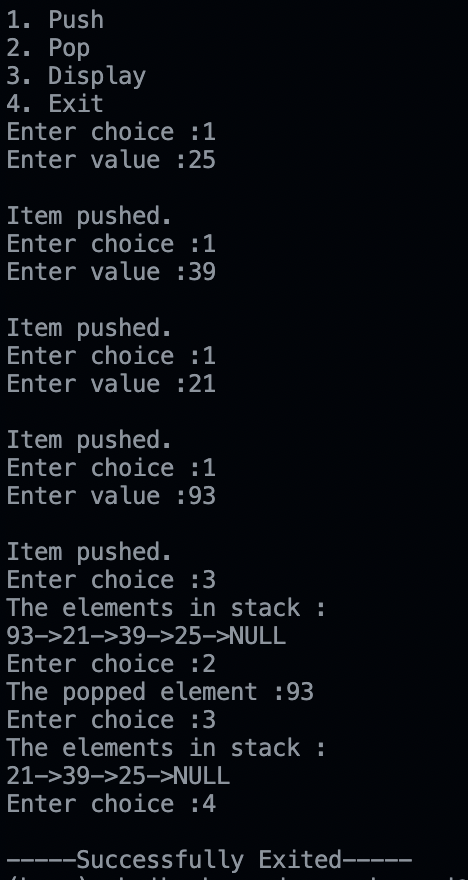
}

}while(ch!=4);

return 0;

}

**OUTPUT:**



**RESULT :**

**3. Linked List Implementation of Queue ADT**

**AIM :** To Implement Queue using Linked List program in c++ .

**ALGORITHM :**

* Step 1 - Include all the header files which are used in the program. And declare all the user defined

functions.

* Step 2 - Define a 'Node' structure with two members data and next.
* Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.
* Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function
* calls in the main method to perform user selected operation.

**enQueue(value) - Inserting an element into the Queue**

We can use the following steps to insert a new node into the queue...

* Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.
* Step 2 - Check whether queue is Empty (rear == NULL)
* Step 3 - If it is Empty then, set front = newNode and rear = newNode.
* Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

**deQueue() - Deleting an Element from Queue**

We can use the following steps to delete a node from the queue...

* Step 1 - Check whether queue is Empty (front == NULL).
* Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from

the function

* Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
* Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

**display() - Displaying the elements of Queue**

We can use the following steps to display the elements (nodes) of a queue...

* Step 1 - Check whether queue is Empty (front == NULL).
* Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
* Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
* Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches
* to 'rear' (temp → next != NULL).
* Step 5 - Finally! Display 'temp → data ---> NULL'.

**PROGRAM :**

#include<iostream>

using namespace std;

struct node{

int data;

struct node \*link;

};

struct node\* root=NULL;

void enqueue(){

int val;

struct node \*temp;

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

cout<<"\nEnter value :";

cin>>val;

temp->data=val;

temp->link=NULL;

if(root==NULL){

root=temp;

}

else{

temp->link=root;

root=temp;

}

cout<<"\nItem pushed.";

}

int length(){

int count=0;

struct node\* p;

p=root;

while(p->link!=NULL){

count++;

p=p->link;

}

return count;

}

void dequeue(){

struct node \*p,\*q;

int len=length();

int i=0;

// cout<<"\nlength:"<<len;

if(root==NULL){

cout<<"The Queue is empty.";

}

else{

p=root;

while(len-1>i){

p=p->link;

i++;

}

// cout<<"The element popped :"<<p->data<<endl;

q=p->link;

p->link=q->link;

q->link=NULL;

free(q);

}

}

void display(){

struct node \*p;

if(root==NULL){

cout<<"\nQueue is empty";

}

else{

p=root;

while(p!=NULL){

cout<<p->data<<"->";

p=p->link;

}

cout<<"NULL";

}

}

int main(){

int ch;

cout<<"\n1. Enqueue";

cout<<"\n2. Dequeue";

cout<<"\n3. Dispaly";

cout<<"\n4. Exit";

do{

cout<<"\nEnter choice :";

cin>>ch;

switch(ch){

case 1 :

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4 :

cout<<"\n-----Successfully Exited-----\n";

break;

default :

cout<<"\nEnter a valid choice.";

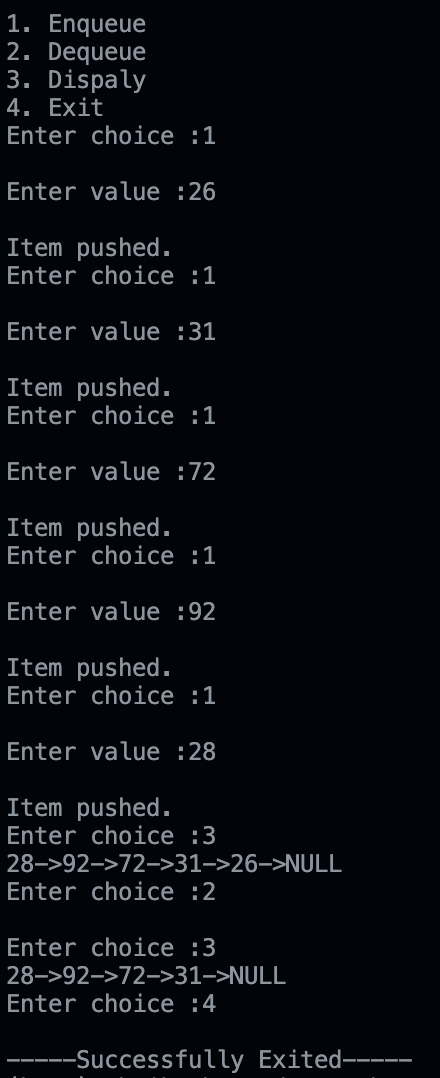
break;

}

}while(ch!=4);

}

**OUTPUT :**

****

**RESULT :**

**4. Use recursion to generate Fibonacci series**

**AIM :** To generate Fibonacci series program Using recursion in c++ .

**ALGORITHM :**

* Firstly we declare a function for recursion named as Fibbo with parameter as integer n, Where n is any positive integer.
* Now we need conditional statements to check the value after every recursion.
* First condition is that the integer number should go until 1.
* In the else part we give the recursion series of the number as fibbo(n - 1) + fibbo(n - 2).
* The recursion will continue till the given range.

**PROGRAM :**

/\* This is a program to print fibonacci series\*/

/\* The output is 0 1 1 2 3 5 8 ...... \*/

#include<iostream>

using namespace std;

int fibonacci(int x){

if(x==1 || x==0){

return (x);

}

else{

return (fibonacci(x - 1) + fibonacci(x - 2));

}

}

int main(){

int n,i=0;

cout<<"Enter number of terms to be printed :";

cin>>n;

cout<<"\nThe Fibonacci series :";

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

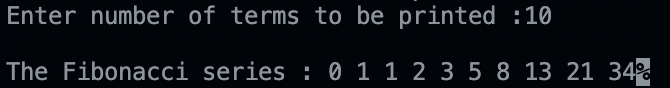
cout<<" "<<fibonacci(i);

}

return 0;

}

**OUTPUT :**

****

**RESULT :**

**5. Implementation of Binary Tree.**

**AIM :** To Implementation of Binary Tree program in c++ .

**ALGORITHM :**

* Step 1 - Create a newNode with given value and set its left and right to NULL.
* Step 2 - Check whether tree is Empty.
* Step 3 - If the tree is Empty, then set root to newNode.
* Step 4 - If the tree is Not Empty, then check whether the value of newNode is smaller or larger than

the node (here it is root node).

* Step 5 - If newNode is smaller than or equal to the node then move to its left child. If newNode

is larger than the node then move to its right child.

* Step 6- Repeat the above steps until we reach to the leaf node (i.e., reaches to NULL).
* Step 7 - After reaching the leaf node, insert the newNode as left child if the newNode is smaller or

equal to that leaf node or else insert it as right child.

**PROGRAM :**

#include<iostream>

using namespace std;

int tree[100];

/\*Inserting at root node\*/

void root\_node(int root){

if(tree[0]!=0){

cout<<"The root node already exists. ";

}

else{

cout<<"\nEnter root data :";

cin>>root;

tree[0]=root;

}

}

//Inserting at left node

void left\_node(int left, int parent){

cout<<"\nEnter parent index :";

cin>>parent;

if(tree[parent]==0){

cout<<"\nParent node donot exist at "<<(parent\*2)+1;

}

else{

cout<<"\nEnter left node :";

cin>>left;

tree[(parent\*2)+1]=left;

}

}

//Inserting at right node

void right\_node(int right, int parent){

cout<<"\nEnter Parent index :";

cin>>parent;

if(tree[parent]==0){

cout<<"\nParent node donot exist at"<<(parent\*2)+2;

}

else{

cout<<"\nEnter right node :";

cin>>right;

tree[(parent\*2)+2]=right;

}

}

void display(){

for(int i=0;i<10;i++){

if(tree[i]!=0){

cout<<tree[i]<<" ";

}

else{

cout<<"\*"<<" ";

}

}

}

int main(){

int parent,root,left,right,ch;

cout<<"\n1. To Enter root node";

cout<<"\n2. To Enter left node";

cout<<"\n3. To Enter right node";

cout<<"\n4. To display tree";

cout<<"\n5. Exit";

do{

cout<<"\nEnter choice :";

cin>>ch;

switch(ch){

case 1 :

root\_node(root);

break;

case 2:

left\_node(left,parent);

break;

case 3 :

right\_node(right,parent);

break;

case 4 :

display();

break;

case 5:

cout<<"\nSucessfully Exited";

break;

default :

cout<<"\nEnter a valid choice ";

break;

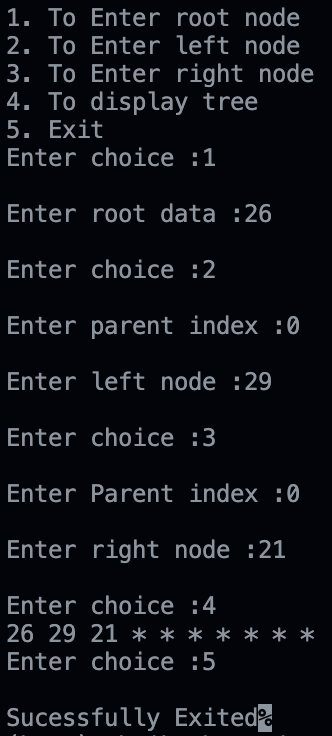
}

}while(ch!=5);

return 0;

}

**OUTPUT :**

****

**RESULT :**

**6. Implementation of Binary Tree Traversal**

**AIM :** To Implementation of Binary Tree Traversal program in c++ .

**ALGORITHM :**

**In-order Traversal**

Until all nodes are traversed −

* Step 1 − Recursively traverse left subtree.
* Step 2 − Visit root node.
* Step 3 − Recursively traverse right subtree.

**Pre-order Traversal**

Until all nodes are traversed −

* Step 1 − Visit root node.
* Step 2 − Recursively traverse left subtree.
* Step 3 − Recursively traverse right subtree.

**Post-order Traversal**

Until all nodes are traversed −

* Step 1 − Recursively traverse left subtree.
* Step 2 − Recursively traverse right subtree.
* Step 3 − Visit root node.

**PROGRAM :**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct node

{

    int data;

    struct node \*right;

    struct node \*left;

};

struct node \*root = NULL;

void adding\_node()

{

    struct node \*temp, \*p;

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

    cout << "\nEnter a value: ";

    cin >> temp->data;

    temp->left = NULL;

    temp->right = NULL;

    p = root;

    if (root == NULL)

    {

        root = temp;

    }

    else

    {

        struct node \*current;

        current = root;

        while (current)

        {

            p = current;

            if (temp->data > current->data)

            {

                current = current->right;

            }

            else

            {

                current = current->left;

            }

        }

        if (temp->data > p->data)

        {

            p->right = temp;

        }

        else

        {

            p->left = temp;

        }

    }

}

void InOrder\_Traversal(struct node \*temp)

{

    if (temp->left)

    {

        InOrder\_Traversal(temp->left);

    }

    cout << temp->data<<" -> ";

    if (temp->right)

    {

        InOrder\_Traversal(temp->right);

    }

}

void PostOrder\_Traversal(struct node \*temp)

{

    if (temp->left)

    {

        PostOrder\_Traversal(temp->left);

    }

    if (temp->right)

    {

        PostOrder\_Traversal(temp->right);

    }

    cout << temp->data<<" -> ";

}

void PreOrder\_Traversal(struct node \*temp)

{

    cout << temp->data<<" -> ";

    if (temp->left)

    {

        PreOrder\_Traversal(temp->left);

    }

    if (temp->right)

    {

        PreOrder\_Traversal(temp->right);

    }

}

int main()

{

int num;

cout << "\n1.Insert\n2.Inorder\n3.Postorder\n4.Preorder\n5.Exit\n";

    do

    {

        cout << "\nEnter your choice: ";

        cin >> num;

        switch (num)

        {

        case 1:

            adding\_node();

            break;

        case 2:

            cout << "\tIn-order traversal\n";

            InOrder\_Traversal(root);

            cout << "\n";

            break;

        case 3:

            cout << "\Post-order traversal\n";

            PostOrder\_Traversal(root);

            cout << "\n";

            break;

        case 4:

            cout << "\Pre-order traversal\n";

            PreOrder\_Traversal(root);

            cout << "\n";

            break;

        case 5:

            cout << "\nExiting...";

            break;

        default:

            cout << "\nInvalid Input";

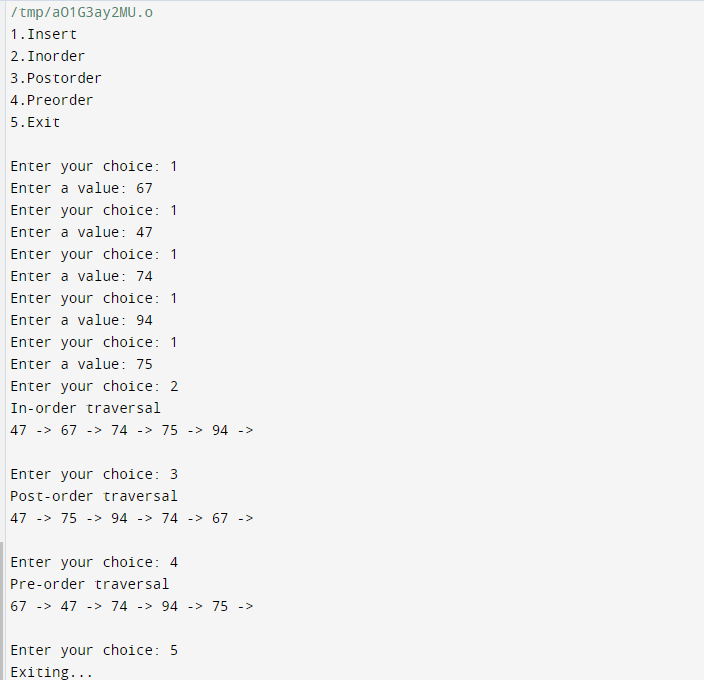
        }

    } while (num != 5);

    return 0;

}

**OUTPUT :**



**RESULT :**

**7. Implementation of Binary search.**

**AIM :** To Implementation of Binary search program using c++ .

**ALGORITHM :**

* Compare x with the middle element.
* If x matches with the middle element, we return the mid index.
* Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid

element. So we recur for the right half.

* Else (x is smaller) recur for the left half.

**PROGRAM :**

#include <iostream>

#include <stdlib.h>

using namespace std;

int BinarySearch(int arr[], int ele, int low, int high)

{

if (low <= high)

{

int mid;

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

if (arr[mid] > ele)

{

return BinarySearch(arr, ele, mid + 1, high);

}

else if (arr[mid] < ele)

{

return BinarySearch(arr, ele, low, mid - 1);

}

return mid;

}

else

{

return -1;

}

}

int main()

{

int array[] = {12, 23, 34, 45, 56, 67, 78};

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

cout<<"Enter the element to search: ";

cin>>ele;

result = BinarySearch(array, ele, 0, n-1);

(result == -1) ? cout<<"\nElement is not found." : cout<<"\nElement found at position: "<<result;

return 0;

}

**OUTPUT :**



**RESULT :**

**8. Implementation of Bubble Sorting.**

**AIM :** To Implement of Bubble Sorting program using c++ .

**ALGORITHM :**

**First Iteration (Compare and Swap)**

* Starting from the first index, compare the first and the second elements.
* If the first element is greater than the second element, they are swapped.
* Now, compare the second and the third elements. Swap them if they are not in order.
* The above process goes on until the last element.

**PROGRAM :**

#include<iostream>

using namespace std;

int main(){

int n=100, temp;

int arr[n];

cout<<"Enter size of array :";

cin>>n;

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

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

cin>>arr[i];

}

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

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

if(arr[j]>arr[j+1]){

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

cout<<"The sorted array is :";

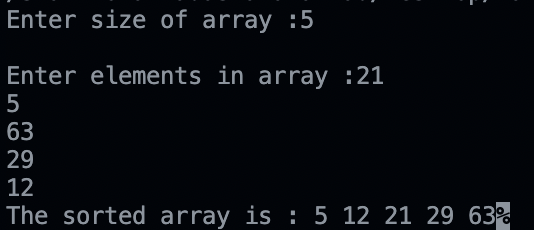
for(int i=0;i<n;i++){

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

}

}

**OUTPUT :**

****

**RESULT :**

**9. Implementation of Selection Sorting.**

**AIM :** To Implement of Selection Sorting program in c++ .

**ALGORITHM :**

* Set the first element as minimum.
* Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.
* After each iteration, minimum is placed in the front of the unsorted list.
* For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

**PROGRAM :**

#include<iostream>

using namespace std;

int main(){

int n=100,temp;

int arr[n];

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

cin>>n;

cout<<"Enter elements in array :";

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

cin>>arr[i];

}

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

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

if(arr[j]<arr[i]){

temp = arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

}

}

cout<<"The array after sorting :";

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

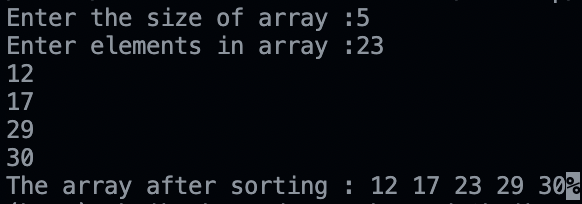
cout<<" "<<arr[i];

}

return 0;

}

**OUTPUT :**

****

**RESULT :**

**10. Implement Hash Table using Linear probing method.**

**AIM :** To Implement Hash Table using Linear probing method using c++ .

**ALGORITHM :**

In linear probing, we linearly probe for next slot.

* Formula for linear probing is (hash+ i) % m where hash = key % m, m is size of array.
* If slot hash(x) % S is full, then we try (hash(x) + 1) % S
* If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S
* If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S ......

**PROGRAM :**

#include <iostream>

#include <stdlib.h>

using namespace std;

#define n 10

int arr[n];

void insert()

{

int index, hash, key, data;

cout<<"\nEnter the key value: ";

cin>>key;

cout<<"\nEnter the data value: ";

cin>>data;

hash = key % n;

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

{

index = (hash + i) % n;

if (arr[index] == 0)

{

arr[index] = data;

break;

}

}

}

void display()

{

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

{

if (arr[i])

{

cout<<arr[i]<<” ”;

}

else

{

cout<<" -- ";

}

}

cout<<"\n";

}

int main()

{

int num;

do{

cout<<"\nEnter the number: ";

cin>>num;

switch(num)

{

case 1:

insert();

break;

case 2:

display();

break;

}

case 3:

cout<<"\nExiting...";

break;

}

default :

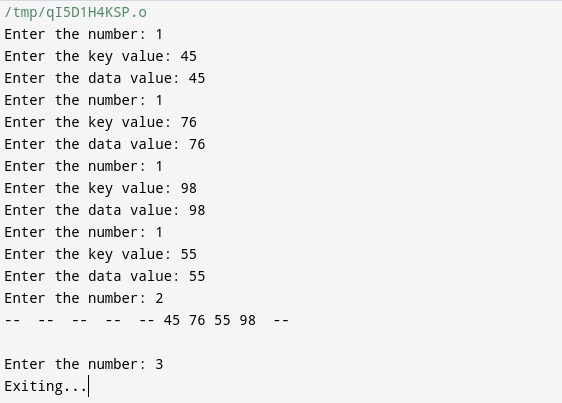
cout<<"\nInvalid Input";

}while(num != 3);

return 0;

}

**OUTPUT :**



**RESULT :**

**Implement Hash Table using Quadratic probing method.**

**AIM :** To Implement Hash Table using Quadratic probing method using c++ .

**ALGORITHM :**

In quadratic probing, we look for ith square slot in ith iteration.

* If slot hash(x) % S is full, then we try (hash(x) + 1\*2) % S
* If (hash(x) + 1\*2) % S is also full, then we try (hash(x) + 2\*2) % S
* If (hash(x) + 2\*2) % S is also full, then we try (hash(x) + 3\*2) % S.....

**PROGRAM :**

#include <iostream>

#include <stdlib.h>

using namespace std;

#define n 10

int arr[n];

void insert()

{

int index, hash, key, data, C = 1, P = 3;

cout<<"\nEnter the key value: ";

cin>>key;

cout<<"\nEnter the data value: ";

cin>>data;

hash = key % n;

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

{

index = (hash + (C\*i) + (P\*i\*i) ) % n;

if (arr[index] == 0)

{

arr[index] = data;

break;

}

}

}

void display()

{

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

{

if (arr[i])

{

cout<<arr[i]<<” “;

}

else

{

cout<<" -- ";

}

}

cout<<"\n";

}

int main()

{

int num;

do{

cout<<"\nEnter the number: ";

cin>>num;

switch(num)

{

case 1:

insert();

break;

case 2:

display();

break;

case 3:

cout<<"Exiting...";

break;

default:

cout<<"Invalid Input";

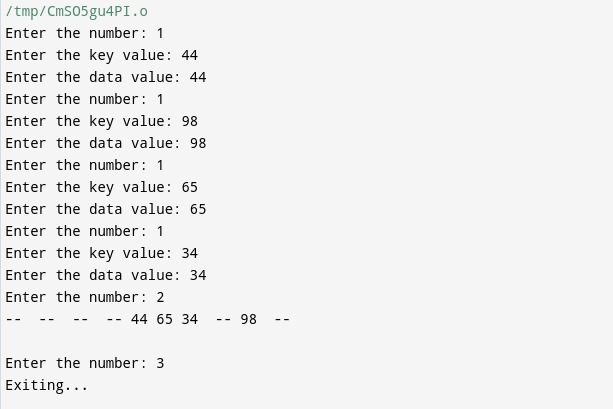
}

}while(num != 3);

return 0;

}

**OUTPUT :**



**RESULT :**