Assignment – 1

1. **Write a program to find the maximum element in an array.**

#include <stdio.h>

#include <stdlib.h>

int main()

{

    int n;

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

    scanf("%d", &n);

    int arr[n];

    if (n < 1)

    {

        printf("Invalid input\n");

        exit(0);

    }

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

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &arr[i]);

    }

    int max = arr[0];

    for (int i = 1; i < n; i++)

    {

        if (arr[i] > max)

        {

            max = arr[i];

        }

    }

    printf("The maximum element in the array is: %d\n", max);

    return 0;

}

Source Code

Set 1

Enter the number of elements in the array: 0

Invalid input

Set 2

Enter the number of elements in the array: 5

Enter the elements of the array: 2 1 9 7 3

The maximum element in the array is: 9

Output

1. **Implement a function to revere an array in place.**

#include <stdio.h>

#include <stdlib.h>

#define max 10

void reverse(int arr[], int n);

int main()

{

    int arr[max], num, pos;

    printf("Enter how many elements you want: ");

    scanf("%d", &num);

    if (num < 1)

    {

        printf("Invalid input\n");

        exit(0);

    }

    printf("Enter the array elements: ");

    for (int i = 0; i < num; ++i)

    {

        scanf("%d", &arr[i]);

    }

    printf("The array is:");

    for (int i = 0; i < num; ++i)

    {

        printf(" %d", arr[i]);

    }

    printf("\n");

    reverse(arr, num);

    printf("The reverse array is:");

    for (int i = 0; i < num; ++i)

    {

        printf(" %d", arr[i]);

    }

    printf("\n");

    return 0;

}

Source Code: main()

void reverse(int arr[], int n)

{

    int temp;

    for (int i = 0; i < n / 2; i++)

    {

        temp = arr[i];

        arr[i] = arr[n - i - 1];

        arr[n - i - 1] = temp;

    }

}

Source Code: reverse()

Set 1

Enter the number of elements in the array: 0

Invalid input

Set 2

Enter how many elements you want: 5

Enter the array elements: 1 2 3 4 5

The array is: 1 2 3 4 5

The reverse array is: 5 4 3 2 1

Output

1. **Implement a function to reverse an array in place.**

#include <stdio.h>

#include <stdlib.h>

#define max 10

int \*intersection(int arr1[], int arr2[], int n1, int n2, int \*size);

void main()

{

    int arr1[max], arr2[max], n1, n2, size = 0;

    printf("Enter the number of elements in the first array: ");

    scanf("%d", &n1);

    printf("Enter the elements in the first array: ");

    for (int i = 0; i < n1; i++)

        scanf("%d", &arr1[i]);

    printf("Enter the number of elements in the second array: ");

    scanf("%d", &n2);

    printf("Enter the elements in the second array: ");

    for (int i = 0; i < n2; i++)

        scanf("%d", &arr2[i]);

    int \*temp = intersection(arr1, arr2, n1, n2, &size);

    printf("The intersection of the two arrays is: ");

    for (int i = 0; i < size; i++)

        printf("%d ", temp[i]);

    printf("\n");

}

Source Code: main()

int \*intersection(int arr1[], int arr2[], int n1, int n2, int \*size)

{

    int \*temp = (int \*)malloc(max \* sizeof(int)), k = 0;

    if (n1 > n2)

        intersection(arr2, arr1, n2, n1, size);

    for (int i = 0; i < n1; i++)

        for (int j = 0; j < n2; j++)

            if (arr1[i] == arr2[j])

            {

                int found = 0;

                for (int l = 0; l < k; l++)

                {

                    if (temp[l] == arr1[i])

                    {

                        found = 1;

                        break;

                    }

                }

                if (!found)

                    temp[k++] = arr1[i];

                break;

            }

    \*size = k;

    return temp;

}

Source Code: \*intersection()

()

Enter the number of elements in the first array: 5

Enter the elements in the first array: 6 4 5 8 2

Enter the number of elements in the second array: 3

Enter the elements in the second array: 2 4 3

The intersection of the two arrays is: 4 2

Output

1. **Write an algorithm to rotate an array given number of positions.**

#include <stdio.h>

#define max 10

void rotate(int arr[], int n, int pos);

int main()

{

    int num, arr[max], pos;

    printf("Enter how many elements you want: ");

    scanf("%d", &num);

    printf("Enter the array elements: ");

    for (int i = 0; i < num; ++i)

        scanf("%d", &arr[i]);

    printf("The position of rotation: ");

    scanf("%d", &pos);

if (pos < 0)

    {

        printf("Invalid input\n");

        return 0;

    }

    printf("The array is:");

    for (int i = 0; i < num; ++i)

        printf(" %d", arr[i]);

    printf("\n");

    rotate(arr, num, pos);

    printf("The rotated array is:");

    for (int i = 0; i < num; ++i)

        printf(" %d", arr[i]);

    printf("\n");

    return 0;

}

Source Code: main()

void rotate(int arr[], int n, int pos)

{

    // Adjust position to be within bounds

    if (pos > n)

        pos = pos % n;

    // Create a temporary array to hold the rotated values

    int temp[max];

    for (int i = 0; i < n; i++)

        temp[(i + pos) % n] = arr[i];

    // Step 3: Copy back from temp to arr

    for (int i = 0; i < n; i++)

        arr[i] = temp[i];

}

Source Code: rotate()

()

Enter how many elements you want: 5

Enter the array elements: 1 2 3 4 5

The position of rotation: 3

The array is: 1 2 3 4 5

The rotated array is: 3 4 5 1 2

Output

1. **Implement an algorithm to find the missing number in an array of integers from 1 to N.**

#include <stdio.h>

#include <stdlib.h>

#define max 10

int main(int argc, char const \*argv[])

{

int arr[max], num, sum = 0, expected\_sum = 0;

printf("Enter how many elements you want (up to %d): ", max);

scanf("%d", &num);

if (num > max)

{

printf("Number exceeds maximum limit of %d.\n", max);

return 1; // Exit if the number exceeds the limit

}

printf("Enter the elements (from 1 to %d): ", num);

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

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

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

sum += arr[i]; // Sum of entered numbers

// Calculate the expected sum of the first 'num' natural numbers

expected\_sum = num \* (num + 1) / 2;

// Find the missing number

printf("The missing number is: %d\n", expected\_sum - sum);

return 0;

}

Source Code: main()

Enter how many elements you want (up to 10): 5

Enter the elements (from 1 to 5): 1 3 4 5 6

The missing number is: 2

Output

Assignment – 2

1. **Write a function to remove duplicates from sorted array.**

#include <stdio.h>

#define MAX 10

void remove\_duplicate(int \*arr, int \*num);

int main()

{

    int arr[MAX], num;

    printf("Enter how many elements you want (max %d): ", MAX);

    scanf("%d", &num);

    if (num <= 0 || num > MAX)

    {

        printf("\n\tInvalid input!\n\n");

        return 1; // Exit on invalid input

    }

    printf("Enter the array elements: ");

    for (int i = 0; i < num; i++)

    {

        scanf("%d", &arr[i]);

    }

    remove\_duplicate(arr, &num);

    printf("The array after remove duplicate is: ");

    for (int i = 0; i < num; i++)

    {

        printf(" %d", arr[i]);

    }

    printf("\n");

    return 0;

}

Source Code: main()

void remove\_duplicate(int \*arr, int \*num)

{

    int temp[MAX], k = 0; // Start k at 0

    for (int i = 0; i < \*num; i++)

    {

        int flag = 0;               // Reset flag for each element

        for (int j = 0; j < k; j++) // Check against current unique elements

        {

            if (arr[i] == temp[j])

            {

                flag = 1;

                break; // Break early if a duplicate is found

            }

        }

        if (!flag)

        {

            temp[k++] = arr[i]; // Only add if not a duplicate

        }

    }

    \*num = k; // Update the count of unique elements

    for (int i = 0; i < k; i++)

    {

        arr[i] = temp[i]; // Copy unique elements back to original array

    }

}

Source Code: remove\_duplicate()

()

Enter how many elements you want (max 10): 5

Enter the array elements: 1 2 2 8 8

The array after remove duplicate is: 1 2 8

Output

1. **Implement an algorithm to find the majority element in an array.**

#include <stdio.h>

#define MAX 10

int main(int argc, char const \*argv[])

{

int arr[MAX], num;

printf("How many elements you want: ");

scanf("%d", &num);

printf("Enter the array elements: ");

for (int i = 0; i < num; ++i) // For inserting array elements

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

int foundMajority = 0; // Flag to track if a majority element is found

for (int i = 0; i < num; ++i) // Calculating the frequency of each element

{

int frequency = 0;

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

{

if (arr[i] == arr[j])

{

frequency++;

}

}

if (frequency > num / 2)

{

printf("Majority element is: %d", arr[i]);

foundMajority = 1; // Set the flag to indicate a majority element is found

break;

}

}

if (!foundMajority) // Check if no majority element was found

printf("No majority element exists");

return 0;

}

Source Code: main()

How many elements you want: 5

Enter the array elements: 6 5 5 4 2

Majority element is: 5

Output

1. **Create a program to find the largest subarray with a sum less than or equal to a given value.**

#include <iostream>

#include <climits> // Use INT\_MIN from <climits>

using namespace std;

int main()

{

    int arr[] = {1, 2, 3, 4, 5}, len = INT\_MIN, givenSum = 6;

    int start = -1, end = -1; // Initialize start and end

    cout << "The array is: ";

    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)

        cout << arr[i] << " ";

    cout << "\nThe given sum is: " << givenSum;

    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)

    {

        int sum = 0;

        for (int j = i; j < sizeof(arr) / sizeof(arr[0]); j++)

        {

            // for (int k = i; k <= j; k++) // Change to <= to include arr[j]

            sum += arr[j];

            if (sum <= givenSum)

            {

                if (j - i + 1 > len) // Update length and start/end indices

                {

                    len = j - i + 1;

                    start = i;

                    end = j;

                }

            }

        }

    }

    if (start == -1 && end == -1)

        cout << "No subarray found with the given sum." << endl;

    else

    {

        cout << "\nThe longest subarray of sum " << givenSum << " is: [";

        for (int i = start; i <= end; i++)

        {

            cout << arr[i];

            if (i < end)

                cout << ", ";

        }

        cout << "]" << endl;

    }

    return 0;

}

Source Code: main()

The array is: 1 2 3 4 5

The given sum is: 6

The longest subarray of sum 6 is: [1, 2, 3]

Output

1. **Write a function to find the contiguous subarray with the largest sum (Kadane’s algorithm).**

#include <iostream>

#include <climits>

using namespace std;

int largestSumSubarray(int \*, int \*);

int main()

{

    int arr[] = {3, -4, 5, 4, -1, 7, -8};

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

    cout << "The array is: ";

    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)

        cout << arr[i] << " ";

    cout << endl;

    cout << " with sum " << largestSumSubarray(arr, &size);

}

Source Code: main()

int largestSumSubarray(int \*arr, int \*size)

{

    int maxSum = INT\_MIN, currSum = 0;

    int start = 0, end = 0, tempStart = 0;

    for (int i = 0; i < \*size; i++)

    {

        currSum += arr[i];

        maxSum = max(currSum, maxSum);

        start = tempStart, end = i;

        if (currSum < 0)

        {

            currSum = 0;

            tempStart = i + 1;

        }

    }

    cout << "The largest subarray is: [ ";

    for (int i = start; i <= end; i++)

        cout << arr[i] << " ";

    cout << "]";

    return maxSum;

}

Source Code: largestSumSubarray()

The array is: 3 -4 5 4 -1 7 -8

The largest subarray is: [ 5 4 -1 7 -8 ] with sum 15

Output

1. **Implement an algorithm to search for an element in a sorted and rotated array (rotated binary search).**

#include <iostream>

#include <climits>

using namespace std;

int binarySearch(int \*, int \*, int \*, int \*);

int main()

{

    int arr[] = {4, 5, 6, 1, 2, 3}, target = 5;

    int start = 0, end = (sizeof(arr) / sizeof(arr[0]) - 1);

    int result = binarySearch(arr, &start, &end, &target);

    cout << "The array is: ";

    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)

        cout << arr[i] << " ";

    cout << "\nThe target element is: " << target;

    if (result == -1)

    {

        cout << "\nElement not present in the array!\n\n";

        exit(0);

    }

    cout << "\nThe element found at " << result + 1 << "th position\n\n";

}

Source Code: main()

The array is: 4 5 6 1 2 3

The target element is: 5

The element found at 2th position

Output

int binarySearch(int \*arr, int \*start, int \*end, int \*target)

{

    int mid = \*start + (\*end - \*start) / 2;

    if (arr[mid] == \*target)

        return mid;

    if (arr[\*start] <= arr[mid]) // Left sorted or not

    {

        if (arr[\*start] <= \*target <= arr[mid]) // Checks the target will exists in left or not

        {

            if (\*target < arr[mid])

            {

                \*end = mid - 1;

                return binarySearch(arr, start, end, target);

            }

            else

            {

                \*start = mid + 1;

                return binarySearch(arr, start, end, target);

            }

        }

    }

    else // Right sorted

    {

        if (arr[mid] <= \*target <= arr[\*end]) // Checks the target will exists in right or not

        {

            if (\*target < arr[mid])

            {

                \*end = mid - 1;

                return binarySearch(arr, start, end, target);

            }

            else

            {

                \*start = mid + 1;

                return binarySearch(arr, start, end, target);

            }

        }

    }

    return -1;

}

Source Code: binarySearch()

()

Assignment – 3

1. **Create a single linked list and write a function to reverse it.**

#include <iostream>

#include <cstdlib>

using namespace std;

typedef struct node

{

    int data;

    node \*next;

} node;

node \*head = NULL;

node \*createNode(int);

void insertNode(int);

void displayList();

void freeList();

void reverseList();

int main()

{

    int choice, value;

    while (true)

    {

        cout << "\nMenu:\n";

        cout << "1. Insert Node\n";

        cout << "2. Reverse List\n";

        cout << "0. Exit\n";

        displayList();

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter value to insert: ";

            cin >> value;

            insertNode(value);

            break;

        case 2:

            reverseList();

            break;

        case 0:

            freeList();

            cout << "Exiting...\n"

                 << endl;

            exit(0);

        default:

            cout << "Invalid choice. Try again." << endl;

        }

    }

    return 0;

}

Source Code: main()

void insertNode(int data)

{

    node \*p, \*temp = createNode(data);

    if (head == NULL)

        head = temp;

    else

    {

        p = head;

        while (p->next != NULL)

            p = p->next;

        p->next = temp;

    }

}

Source Code: insertNode(data)

()

node \*createNode(int data)

{

    node \*newNode = new node();

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

Source Code: \*createNode(data)

()

void displayList()

{

    if (!head)

    {

        cout << "\n\tList is empty!\n"

             << endl;

        return;

    }

    node \*p = head;

    cout << "\n\tList is: ";

    while (p != NULL)

    {

        cout << p->data << "->";

        p = p->next;

    }

    cout << "NULL\n"

         << endl;

}

Source Code: displayList()

()

void reverseList()

{

    if (head == NULL || head->next == NULL)

        return;

    node \*prevNode = head,

         \*currNode = prevNode->next;

    while (currNode != NULL)

    {

        node \*nextNode = currNode->next;

        currNode->next = prevNode;

        // Updated

        prevNode = currNode;

        currNode = nextNode;

    }

    head->next = NULL;

    head = prevNode;

}

Source Code: reverseList()

()

void freeList()

{

    node \*temp;

    while (head != nullptr)

    {

        temp = head;

        head = head->next;

        delete temp;

    }

    cout << "\nAll nodes freed." << endl;

}

Source Code: freeList()

()

Menu:

1. Insert Node

2. Reverse List

0. Exit

List is empty!

Enter your choice: 1

Enter value to insert: 10

Menu:

1. Insert Node

2. Reverse List

0. Exit

List is: 10🡪NULL

Enter your choice: 1

Enter value to insert: 20

Menu:

1. Insert Node

2. Reverse List

0. Exit

List is: 10🡪20🡪NULL

Enter your choice: 1

Enter value to insert: 30

Menu:

1. Insert Node

2. Reverse List

0. Exit

List is: 10🡪20🡪30🡪NULL

Enter your choice: 2

Menu:

1. Insert Node

2. Reverse List

0. Exit

List is: 30🡪20🡪10🡪NULL

Enter your choice: 0

All nodes freed.

Exiting...

Output

1. **Implement a program to detect if a linked list has a cycle.**

#include <iostream>

using namespace std;

typedef struct node

{

    int data;

    node \*next;

} node;

node \*head = NULL;

node \*createNode(int);

node \*checkCycle(); // Function to detect and handle cycle (to be implemented by you)

void createCyclicList();

void displayList();

void freeList();

int main()

{

    createCyclicList(); // Create a hardcoded list with a cycle

    displayList();      // Display the list

    if (!checkCycle()) // Call the function to detect a cycle

        cout << "\n\tIn this list cycle not present!"

             << endl;

    else

        cout << "\n\tIn this list cycle present!"

             << endl;

    freeList(); // Cleanup

    cout << "\tExiting...\n"

         << endl;

    return 0;

}

Source Code: main()

node \*createNode(int data)

{

    node \*newNode = new node();

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

Source Code: \*createNode(data)

()

void createCyclicList()

{

    // Create nodes

    node \*node1 = createNode(1);

    node \*node2 = createNode(2);

    node \*node3 = createNode(3);

    node \*node4 = createNode(4);

    node \*node5 = createNode(5);

    // Link nodes to form a list

    head = node1;

    node1->next = node2;

    node2->next = node3;

    node3->next = node4;

    node4->next = node5;

    // Create a cycle (node5 points back to node2)

    node5->next = node2; // Change to node2 to create a cycle

}

Source Code: createCyclicList()

()

void displayList()

{

    if (!head)

    {

        cout << "\n\tList is empty!\n"

             << endl;

        return;

    }

    node \*p = head;

    int count = 0; // Prevent infinite loop for cyclic list

    cout << "\n\tList is: ";

    while (p != NULL && count < 12)

    {

        cout << p->data << "->";

        p = p->next;

        count++;

    }

    if (count == 12)

        cout << "...\n";

    else

        cout << "NULL\n";

}

Source Code: displayList()

()

// Placeholder function for cycle detection

node \*checkCycle()

{

    node \*turtle = head, \*rabbit = head;

    while (rabbit != NULL && rabbit->next != NULL)

    {

        turtle = turtle->next;

        rabbit = rabbit->next->next;

        if (turtle == rabbit)

            return turtle;

    }

    return NULL;

}

Source Code: \*checkCycle()

()

void freeList()

{

    if (!head)

    {

        cout << "\n\tList is already empty." << endl;

        return;

    }

    // Use checkCycle to detect the meeting point if there's a cycle

    node \*meetingPoint = checkCycle();

    if (meetingPoint != NULL) // If a cycle exists

    {

        node \*slow = head;

        // Find the start of the cycle

        while (slow->next != meetingPoint->next)

        {

            slow = slow->next;

            meetingPoint = meetingPoint->next;

        }

        // Break the cycle

        meetingPoint->next = NULL;

    }

    // Free the list nodes

    node \*temp;

    while (head != nullptr)

    {

        temp = head;

        head = head->next;

        delete temp;

    }

    cout << "\n\tAll nodes freed." << endl;

}

Source Code: freeList()

()

List is: 1->2->3->4->5->2->3->4->5->2->3->4->...

In this list cycle present!

All nodes freed.

Exiting...

Output

List is: 1->2->3->4->5->NULL

In this list cycle not present!

All nodes freed.

Exiting...

Output

1. **Write a function to merge two sorted linked list into a single linked list.**

#include <iostream>

using namespace std;

typedef struct node

{

    int data;

    node \*next;

} node;

node \*head1 = NULL;

node \*head2 = NULL;

node \*createNode(int);

node \*createList();

node \*mergeList();

void displayList(node \*);

void freeList(node \*);

int main()

{

    cout << "\nFor the 1st list\n";

    cout << "-----------------\n";

    head1 = createList();

    cout << "\nFor the 2nd list\n";

    cout << "-----------------\n";

    head2 = createList();

    cout << '\n';

    cout << "1st List: ";

    displayList(head1);

    cout << "2nd List: ";

    displayList(head2);

    cout << "Merge List: ";

    node \*mergedList = mergeList();

    displayList(mergedList);

    freeList(mergedList); // Cleanup

    cout << "Exiting...\n";

    return 0;

}

Source Code: main()

void displayList(node \*head)

{

    if (!head)

    {

        cout << "Empty!\n"

             << endl;

        return;

    }

    node \*p = head;

    while (p != NULL)

    {

        cout << p->data << "->";

        p = p->next;

    }

    cout << "NULL\n"

         << endl;

}

Source Code: displayList()

()

node \*createNode(int data)

{

    node \*newNode = new node();

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

Source Code: \*createNode(data)

()

void freeList()

{

    node \*temp;

    while (head1 != nullptr)

    {

        temp = head1;

        head1 = head1->next;

        delete temp;

    }

    while (head2 != nullptr)

    {

        temp = head2;

        head2 = head2->next;

        delete temp;

    }

    cout << "\nAll nodes freed." << endl;

}

Source Code: freeList()

()

node \*mergeList()

{

    node \*p1 = head1, \*p2 = head2;

    node \*dummy = new node(), \*p3 = dummy;

    while (p1 != NULL && p2 != NULL)

    {

        if (p1->data < p2->data)

        {

            p3->next = p1;

            p1 = p1->next;

        }

        else

        {

            p3->next = p2;

            p2 = p2->next;

        }

        p3 = p3->next;

    }

    while (p1 != NULL)

    {

        p3->next = p1;

        p1 = p1->next;

        p3 = p3->next;

    }

    while (p2 != NULL)

    {

        p3->next = p2;

        p2 = p2->next;

        p3 = p3->next;

    }

    return dummy->next;

}

Source Code: \*mergeList()

()

For the 1st list

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

Enter the number of nodes: 5

Enter value for node 1: 1

Enter value for node 2: 2

Enter value for node 3: 4

Enter value for node 4: 6

Enter value for node 5: 8

For the 2nd list

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

Enter the number of nodes: 4

Enter value for node 1: 3

Enter value for node 2: 5

Enter value for node 3: 7

Enter value for node 4: 9

1st List: 1->2->4->6->8->NULL

2nd List: 3->5->7->9->NULL

Merge List: 1->2->3->4->5->6->7->8->9->NULL

All nodes freed.

Exiting...

Output

node \*createList()

{

    int n, value;

    cout << "Enter the number of nodes: ";

    cin >> n;

    if (n <= 0)

    {

        cout << "List size must be greater than 0.\n";

        return NULL;

    }

    // Create the head node

    cout << "Enter value for node 1: ";

    cin >> value;

    node \*head = createNode(value);

    node \*current = head;

    // Create the remaining nodes

    for (int i = 2; i <= n; ++i)

    {

        cout << "Enter value for node " << i << ": ";

        cin >> value;

        current->next = createNode(value);

        current = current->next;

    }

    return head;

}

Source Code: \*createList()

()

1st List: 1->2->4->6->8->NULL

2nd List: 3->5->7->9->NULL

Merge List: 1->2->3->4->5->6->7->8->9->NULL

All nodes freed.

Exiting...

Output

1. **Implement an algorithm to find the Nth node from the end of a linked list.**

#include <iostream>

using namespace std;

typedef struct Node

{

    int data;

    Node \*next;

    Node(int data, Node \*next = NULL)

    {

        this->data = data;

        this->next = next;

    }

} Node;

Node \*head = NULL;

Node \*createNode(int);

Node \*findNthFromEnd(int);

void createList();

void displayList();

void freeList();

int main()

{

    cout << '\n';

    createList();

    displayList();

    int n;

    cout << "\tEnter the n'th node: ";

    cin >> n;

    Node \*result = findNthFromEnd(n);

    if (result)

        cout << "\tThe " << n << "'th node from end of list is: "

             << result->data << endl;

    else

        cout << "Invalid value of n!" << endl;

    freeList(); // Cleanup

    cout << "\tExiting...\n\n";

    return 0;

}

Source Code: main()

Node \*createNode(int data)

{

    return new Node(data, NULL);

}

Source Code: \*createNode(data)

()

void displayList()

{

    if (!head)

    {

        cout << "\n\tList is empty!\n"

             << endl;

        return;

    }

    Node \*p = head;

    cout << "\n\tList is: ";

    while (p != NULL)

    {

        cout << p->data << "->";

        p = p->next;

    }

    cout << "NULL\n"

         << endl;

}

Source Code: displayList()

()

void freeList()

{

    Node \*temp;

    while (head != nullptr)

    {

        temp = head;

        head = head->next;

        delete temp;

    }

    cout << "\n\tAll nodes freed." << endl;

}

Source Code: freeList()

()

void createList()

{

    // Create Nodes

    Node \*node1 = createNode(1);

    Node \*node2 = createNode(2);

    Node \*node3 = createNode(3);

    Node \*node4 = createNode(4);

    Node \*node5 = createNode(5);

    // Link nodes to form a list

    head = node1;

    node1->next = node2;

    node2->next = node3;

    node3->next = node4;

    node4->next = node5;

    node5->next = NULL;

}

Source Code: createList()

()

Node \*findNthFromEnd(int n)

{

    Node \*slow = head, \*fast = head;

    for (int i = 0; i < n; i++)

    {

        if (fast == NULL)

            return NULL;

        fast = fast->next;

    }

    while (fast != NULL)

    {

        slow = slow->next;

        fast = fast->next;

    }

    return slow;

}

Source Code: \*findFromEnd(n)

()

List is: 1->2->3->4->5->NULL

Enter the n'th node: 2

The 2'th node from end of list is: 4

All nodes freed.

Exiting...

Output

List is: 1->2->3->4->5->NULL

Enter the n'th node: -1

Invalid value of n!

All nodes freed.

Exiting...

Output

1. **Create a program to delete a node with a given value from a linked list.**

#include <iostream>

#include <cstdlib>

using namespace std;

typedef struct Node

{

    int data;

    Node \*next;

} Node;

Node \*head = NULL;

Node \*createNode(int);

void insertNode(int);

void displayList();

void freeList();

void deleteNodeByValue();

int main()

{

    int choice, value;

    while (true)

    {

        cout << "\nMenu:\n";

        cout << "1. Insert Node\n";

        cout << "2. Delete By Value\n";

        cout << "0. Exit\n";

        displayList();

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter value to insert: ";

            cin >> value;

            insertNode(value);

            break;

        case 2:

            deleteNodeByValue();

            break;

        case 0:

            freeList();

            cout << "Exiting...\n"

                 << endl;

            exit(0);

        default:

            cout << "Invalid choice. Try again." << endl;

        }

    }

    return 0;

}

Source Code: main()

Node \*createNode(int data)

{

    Node \*newNode = new Node();

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

Source Code: \*createNode(data)

()

void displayList()

{

    if (!head)

    {

        cout << "\n\tList is empty!\n"

             << endl;

        return;

    }

    Node \*p = head;

    cout << "\n\tList is: ";

    while (p != NULL)

    {

        cout << p->data << "->";

        p = p->next;

    }

    cout << "NULL\n"

         << endl;

}

Source Code: displayList()

()

void freeList()

{

    Node \*temp;

    while (head != nullptr)

    {

        temp = head;

        head = head->next;

        delete temp;

    }

    cout << "\nAll nodes freed." << endl;

}

Source Code: freeList()

()

void deleteNodeByValue()

{

    int value;

    cout << "Enter the value of the node to delete: ";

    cin >> value;

    if (head == NULL)

    {

        cout << "\n\tThe list is empty. Nothing to delete.\n"

             << endl;

        return;

    }

    // Special case: if the head node contains the value

    if (head->data == value)

    {

        Node \*temp = head;

        head = head->next; // Move head to the next node

        free(temp);        // Free the old head

        // cout << "Node with value " << value << " deleted from the head." << endl;

        return;

    }

    // General case: traverse the list to find and delete the node

    Node \*currNode = head->next; // Start from the second node

    Node \*prevNode = head;       // Previous node starts as head

    while (currNode != NULL)

    {

        if (currNode->data == value)

        {

            prevNode->next = currNode->next; // Bypass the current node

            free(currNode);                  // Free the node

            // cout << "Node with value " << value << " deleted." << endl;

            return;

        }

        // Move to the next pair of nodes

        prevNode = currNode;

        currNode = currNode->next;

    }

    // If no node is found with the given value

    cout << "\n\tNode with value " << value << " not found in the list.\n"

         << endl;

}

Source Code: deleteNodeByValue()

()

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is empty!

Enter your choice: 1

Enter value to insert: 10

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is: 10->NULL

Enter your choice: 20

Invalid choice. Try again.

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is: 10->NULL

Enter your choice: 1

Enter value to insert: 20

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is: 10->20->NULL

Enter your choice: 2

Enter the value of the node to delete: 30

Node with value 30 not found in the list.

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is: 10->20->NULL

Enter your choice: 2

Enter the value of the node to delete: 20

Output

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is: 10->NULL

Enter your choice: 2

Enter the value of the node to delete: 10

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is empty!

Enter your choice: 2

Enter the value of the node to delete: 10

The list is empty. Nothing to delete.

Menu:

1. Insert Node

2. Delete By Value

0. Exit

List is empty!

Enter your choice: 0

All nodes freed.

Exiting...

Output

1. **Write a function to check if 2 linked list intersect and if they do find intersection node.**

#include <iostream>

using namespace std;

typedef struct Node

{

    int data;

    Node \*next;

    Node(int data, Node \*next = NULL)

    {

        this->data = data;

        this->next = next;

    }

} Node;

Node \*head1 = NULL, \*head2 = NULL;

Node \*findIntersect();

void createList();

void displayList(Node \*);

void freeList();

int main()

{

    createList();

    cout << "\n1st List: ";

    displayList(head1);

    cout << "2nd List: ";

    displayList(head2);

    Node \*intersect = findIntersect();

    if (intersect)

        cout << "The intersect node is: " << intersect->data << '\n'

             << endl;

    else

        cout << "\tNo intersection between the two lists." << endl;

    freeList(); // Cleanup

    cout << "Exiting...\n";

    return 0;

}

Source Code: main()

void displayList(Node \*head)

{

    if (!head)

    {

        cout << "Empty!\n"

             << endl;

        return;

    }

    Node \*p = head;

    while (p != NULL)

    {

        cout << p->data << "->";

        p = p->next;

    }

    cout << "NULL\n"

         << endl;

}

Source Code: displayList()

()

void freeList()

{

    Node \*temp;

    while (head1 != nullptr)

    {

        temp = head1;

        head1 = head1->next;

        delete temp;

    }

    while (head2 != nullptr)

    {

        temp = head2;

        head2 = head2->next;

        delete temp;

    }

    cout << "\nAll nodes freed." << endl;

}

Source Code: freeList()

()

void createList()

{

    // 1st list nodes

    Node \*node1 = new Node(1);

    Node \*node2 = new Node(2);

    Node \*node3 = new Node(3);

    Node \*node4 = new Node(4);

    Node \*node5 = new Node(5);

    // 2nd List nodes

    Node \*node6 = new Node(6);

    Node \*node7 = new Node(7);

    Node \*node8 = new Node(8);

    // Link nodes to form a list

    head1 = node1;

    node1->next = node2;

    node2->next = node3;

    node3->next = node4;

    node4->next = node5;

    head2 = node6;

    node6->next = node7;

    node7->next = node8;

    node8->next = node3;

    node5->next = NULL;

}

Source Code: createList()

()

Node \*findIntersect()

{

    Node \*p1 = head1, \*p2 = head2;

    // Traverse both lists. When one pointer reaches the end, switch to the other list.

    while (p1 != p2)

    {

        p1 = (p1 == NULL) ? head2 : p1->next;

        p2 = (p2 == NULL) ? head1 : p2->next;

    }

    // Either intersection node or NULL (if no intersection)

    return p1;

}

Source Code: \*findIntersect()

()

1st List: 1->2->3->4->5->NULL

2nd List: 6->7->8->3->4->5->NULL

The intersect node is: 3

Output

1. **Implement a function to add 2 numbers represented by linked lists (e.g., 342 + 465 = 807).**

#include <iostream>

using namespace std;

// Node structure

typedef struct Node

{

    int data;

    Node \*next;

    Node(int data, Node \*next = NULL)

    {

        this->data = data;

        this->next = next;

    }

} Node;

// Function prototypes

void createList(Node \*&head);

void displayList(Node \*head);

void freeList(Node \*head);

void insertAtTail(Node \*&head, Node \*&tail, int data);

Node \*reverseList(Node \*head);

Node \*add(Node \*first, Node \*second);

Node \*addTwoLists(Node \*first, Node \*second);

int main()

{

    Node \*first = NULL, \*second = NULL;

    // Create lists

    cout << '\n'

         << "Creating the first list...." << endl;

    createList(first);

    cout << "Creating the second list...." << endl;

    createList(second);

    // Display the lists

    cout << "\nFirst List: ";

    displayList(first);

    cout << "\nSecond List: ";

    displayList(second);

    cout << '\n';

    // Add the two lists

    Node \*sum = addTwoLists(first, second);

    // Display the result

    cout << "Sum List: ";

    displayList(sum);

    // Free memory

    freeList(first);

    freeList(second);

    freeList(sum);

    cout << "\nExiting...\n";

    return 0;

}

Source Code: main()

void displayList(Node \*head)

{

    if (!head)

    {

        cout << "Empty!\n";

        return;

    }

    Node \*p = head;

    while (p != NULL)

    {

        cout << p->data;

        if (p->next != NULL)

            cout << "->";

        p = p->next;

    }

    cout << "->NULL\n";

}

Source Code: displayList()

()

void freeList(Node \*head)

{

    Node \*temp;

    while (head != nullptr)

    {

        temp = head;

        head = head->next;

        delete temp;

    }

    // cout << "All nodes freed." << endl;

}

Source Code: freeList()

()

void createList(Node \*&head, int number)

{

    if (number == 0)

    {

        head = new Node(0); // Handle the case for 0

        return;

    }

    while (number > 0)

    {

        int digit = number % 10; // Extract the last digit

        Node \*newNode = new Node(digit);

        newNode->next = head; // Insert at the beginning (most significant digit)

        head = newNode;       // Move the head to the new node

        number /= 10; // Remove the last digit

    }

}

Source Code: createList()

()

Node \*reverseList(Node \*head)

{

    if (head == NULL || head->next == NULL)

        return head;

    Node \*prevNode = NULL, \*currNode = head;

    while (currNode != NULL)

    {

        Node \*nextNode = currNode->next; // Store the next node

        currNode->next = prevNode;       // Reverse the link

        prevNode = currNode;             // Move prevNode forward

        currNode = nextNode;             // Move currNode forward

    }

    return prevNode; // New head of the reversed list

}

Source Code: \*reverseList()

()

void insertAtTail(Node \*&head, Node \*&tail, int data)

{

    Node \*temp = new Node(data);

    if (head == NULL)

    {

        head = temp;

        tail = temp;

        return;

    }

    else

    {

        tail->next = temp;

        tail = temp;

    }

}

Source Code: insertAtTail()

()

Node \*addTwoLists(Node \*first, Node \*second)

{

    first = reverseList(first);

    second = reverseList(second);

    Node \*result = add(first, second);

    // Reverse the result to maintain proper order

    return reverseList(result);

}

Source Code: \*addTwoLists()

()

Node \*add(Node \*first, Node \*second)

{

    int carry = 0;

    Node \*ansHead = NULL, \*ansTail = NULL;

    while (first != NULL && second != NULL)

    {

        int sum = carry + first->data + second->data;

        int digit = sum % 10;

        insertAtTail(ansHead, ansTail, digit);

        carry = sum / 10;

        first = first->next;

        second = second->next;

    }

    while (first != NULL)    {

        int sum = carry + first->data;

        int digit = sum % 10;

        insertAtTail(ansHead, ansTail, digit);

        carry = sum / 10;

        first = first->next;

    }

    while (second != NULL)    {

        int sum = carry + second->data;

        int digit = sum % 10;

        insertAtTail(ansHead, ansTail, digit);

        carry = sum / 10;

        second = second->next;

    }

    while (carry != 0)    {

        int sum = carry;

        int digit = sum % 10;

        insertAtTail(ansHead, ansTail, digit);

        carry = sum / 10;

    }

    return ansHead;

}

Source Code: \*add()

()

First List: 3->4->5->NULL

Second List: 4->5->NULL

Sum List: 3->9->0->NULL

Exiting...

Output

Assignment – 4

1. **Create a function to sort n array using bubble sort.**

#include <iostream>

using namespace std;

void swap(int \*, int \*);

void bubble\_sort(int \*, int);

int main()

{

    int arr[], size = sizeof(arr) / sizeof(arr[0]) - 1;

    cout << "How many element you want to insert: ";

    cin >> size;

    if (size <= 0) {cout << "Invalid input"; exit(0);}

    cout << "Enter the array elements: ";

    for (int i = 0; i < size; i++)

        cin >> arr[i];

    cout << "The unsorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    bubble\_sort(arr, size);

    cout << "The sorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    return 0;

}

Source Code: main()

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

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

Source Code: swap(a,b)

void bubble\_sort(int \*arr, int size)

{

    for (int i = 0; i < size; i++)

    {

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

        {

            if (arr[j] > arr[j + 1])

                swap(arr[j], arr[j + 1]);

        }

    }

}

Source Code: bubbleSort(arr,size)

How many elements you want to insert: 5

Enter the array elements: 5 4 3 2 1

The unsorted array is: 5 4 3 2 1

The sorted array is: 1 2 3 4 5

Output

1. **Write a program to perform the quick sort.**

#include <iostream>

using namespace std;

void swap(int \*, int \*);

void quick\_sort(int \*, int, int);

int partition(int \*arr, int start, int end);

int main()

{

    int arr[10], size;

    cout << "How many element you want to insert: ";

    cin >> size;

    if (size <= 0){cout << "Invalid input";exit(0);}

    cout << "Enter the array elements: ";

    for (int i = 0; i < size; i++)

        cin >> arr[i];

    cout << "The unsorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    quick\_sort(arr, 0, size);

    cout << "The sorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    return 0;

}

Source Code: main()

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

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

Source Code: swap(a,b)

void quick\_sort(int \*arr, int start, int end)

{

    if (start < end)

    {

        int pivot = partition(arr, start, end);

        quick\_sort(arr, start, pivot - 1);

        quick\_sort(arr, pivot + 1, end);

    }

}

Source Code: quickSort(arr,start,end)

int partition(int \*arr, int start, int end)

{

    int pivot = arr[start], i = start, j = end;

    while (i < j)

    {

        while (arr[i] <= pivot)

            i++;

        while (arr[j] > pivot)

            j--;

        if (j > i)

            swap(&arr[i], &arr[j]);

    }

   swap(&arr[start], &arr[j]);

    return j;

}

Source Code: partition(arr,start,end)

How many element you want to insert: 5

Enter the array elements: 3 1 9 4 6

The unsorted array is: 3 1 9 4 6

The sorted array is: 1 3 4 6 9

Output

1. **Implement the heapsort algorithm.**

#include <iostream>

using namespace std;

void swap(int \*, int \*);

void heapSort(int \*arr, int);

void build\_max\_heap(int \*, int size);

void heapify(int \*, int, int);

int main()

{

    int size;

    cout << "How many element you want to insert: ";

    cin >> size;

    if (size <= 0)

    {

        cout << "Invalid input";

        exit(0);

    }

    int arr[size]; // Dynamically allocate based on the size input

    cout << "Enter the array elements: ";

    for (int i = 0; i < size; i++)

        cin >> arr[i];

    cout << "The unsorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    heapSort(arr, size);

    cout << "The sorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    return 0;

}

Source Code: main()

void heapSort(int \*arr, int size)

{

    build\_max\_heap(arr, size); // Build the initial max heap

    // Perform heap sort

    for (int i = size - 1; i > 0; i--) // Start from the end of the array

    {

        swap(&arr[0], &arr[i]); // Move the largest element to the end

        heapify(arr, i, 0);     // Heapify the reduced heap

    }

}

Source Code: heapSort(arr,size)

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

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

Source Code: swap(a,b)

void heapify(int \*arr, int size, int i)

{

    int largest = i;       // Assume the current node is the largest

    int left = 2 \* i + 1;  // Left child

    int right = 2 \* i + 2; // Right child

    // Check if the left child exists and is greater than the current largest

    if (left < size && arr[left] > arr[largest])

        largest = left;

    // Check if the right child exists and is greater than the current largest

    if (right < size && arr[right] > arr[largest])

        largest = right;

    // If the largest element is not the current node

    if (largest != i)

    {

        swap(&arr[i], &arr[largest]); // Swap the current node with the largest

        heapify(arr, size, largest);  // Recursively heapify the affected subtree

    }

}

Source Code: heapify(arr,size,i)

void build\_max\_heap(int \*arr, int size)

{

    for (int i = size / 2 - 1; i >= 0; i--) // Start heapifying from the last non-leaf node

        heapify(arr, size, i);

}

Source Code: build\_max\_heap(arr,size)

How many element you want to insert: 5

Enter the array elements: 3 1 9 4 6

The unsorted array is: 3 1 9 4 6

The sorted array is: 1 3 4 6 9

Output

1. **Implement the merge algorithm.**

#include <iostream>

#include <climits>

using namespace std;

void merge(int \*, int, int, int);

void mergeSort(int \*, int, int);

int main()

{

    int size;

    cout << "How many element you want to insert: ";

    cin >> size;

    if (size <= 0)

    {

        cout << "Invalid input";

        exit(0);

    }

    int arr[size]; // Dynamically allocate based on the size input

    int start = 0, end = size;

    cout << "Enter the array elements: ";

    for (int i = 0; i < size; i++)

        cin >> arr[i];

    cout << "Before sorting the array: ";

    for (int i = start; i <= end; i++)

        cout << " " << arr[i];

    cout << endl;

    mergeSort(arr, start, end);

    cout << "The sorted array is: ";

    for (int i = start; i <= end; i++)

        cout << " " << arr[i];

    cout << endl;

    return 0;

}

Source Code: main()

void mergeSort(int \*arr, int start, int end)

{

    if (start < end)

    {

        int mid = (start + end) / 2;

        mergeSort(arr, start, mid);

        mergeSort(arr, mid + 1, end);

        merge(arr, start, mid, end);

    }

}

Source Code: mergeSort(arr,start,end)

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

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

Source Code: swap(a,b)

void merge(int \*arr, int start, int mid, int end)

{

    int temp[end];

    int i = start, j = mid + 1, k = 0;

    while (i <= mid && j <= end)

    {

        if (arr[i] < arr[j])

            temp[k++] = arr[i++];

        else

            temp[k++] = arr[j++];

    }

    while (i <= mid)

        temp[k++] = arr[i++];

    while (j <= end)

        temp[k++] = arr[j++];

    for (int i = start; i <= end; i++)

    {

        arr[i] = temp[i - start];

    }

}

Source Code: merge(arr,start,mid,end)

How many element you want to insert: 5

Enter the array elements: 3 1 9 4 6

The unsorted array is: 3 1 9 4 6

The sorted array is: 1 3 4 6 9

Output

1. **Implement the radix sort algorithm for integers.**

#include <iostream>

#include <climits>

#define N 10

using namespace std;

int get\_digit(int, int);

int find\_max(int \*, int);

void radix\_sort(int \*, int);

void count\_sort(int \*, int, int);

int main()

{

    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

    int size = sizeof(arr) / sizeof(arr[0]) - 1;

    cout << "The unsorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    radix\_sort(arr, size);

    cout << "The sorted array is: ";

    for (int i = 0; i < size; i++)

        cout << " " << arr[i];

    cout << endl;

    return 0;

}

Source Code: main()

void radix\_sort(int \*arr, int size)

{

    for (int place = 1; find\_max(arr, size) / place > 0; place \*= 10)

        count\_sort(arr, size, place);

}

Source Code: swap(a,b)

void count\_sort(int \*arr, int size, int place)

{

    int output[size], count[N] = {0};

    for (int i = 0; i < size; i++)

    {

        int digit = get\_digit(arr[i], place);

        count[digit]++;

    }

    for (int i = 1; i < N; i++)

        count[i] += count[i - 1];

    for (int i = size - 1; i >= 0; i--)

    {

        int digit = get\_digit(arr[i], place);

        output[--count[digit]] = arr[i];

    }

    for (int i = 0; i < size; i++)

        arr[i] = output[i];

}

Source Code: count\_sort(arr,size,place)

int get\_digit(int num, int place)

{

    return (num / place) % 10;

}

Source Code: get\_digit(num,palce)

int find\_max(int \*arr, int size)

{

    int max = INT\_MIN;

    for (int i = 0; i < size; i++)

    {

        if (max < arr[i])

            max = arr[i];

    }

    return max;

}

Source Code: find\_max(arr,size)

The unsorted array is: 170 45 75 90 802 24 2

The sorted array is: 2 24 45 75 90 170 802

Output

Assignment – 5

1. **Implement an binary search tree and write functions for insertion & deletion.**

#include <iostream>

using namespace std;

typedef struct Node

{

    Node \*left, \*right;

    int data;

    Node(int data)

    {

        this->left = NULL;

        this->data = data;

        this->right = NULL;

    }

} Node;

Node \*root = NULL;

void createBST();

Node \*createNode(int);

void freeTree(Node \*root);

void printTree(Node \*, int);

void insertBST(Node \*&, int);

Node \*deleteNodeBST(Node \*, int);

Node \*inOrderPredecessor(Node \*ptr);

int main(){

    createBST(); // Initialize the BST with some predefined values.

    while (1)    {

        cout << "\nMenu:\n";

        cout << "1. Insert Node in the tree\n";

        cout << "2. Delete Node from tree\n";

        cout << "0. Exit\n";

        printTree(root, 0);

        cout << "\nEnter your choice: ";

        int choice, value;

        cin >> choice;

        switch (choice)       {

        case 1:

            cout << "Enter value to insert: ";

            cin >> value;

            insertBST(root, value);

            break;

        case 2:

            cout << "Enter value to delete: ";

            cin >> value;

            root = deleteNodeBST(root, value);

            break;

        case 0:

            freeTree(root);

            cout << "\nExiting...\n\n";

            exit(0);

        default:

            cout << "Invalid choice. Try again." << endl;

        }

    }

    return 0;

}

Source Code: main()

void insertBST(Node \*&ptr, int value)

{

    if (ptr == NULL)

    {

        ptr = createNode(value);

        return;

    }

    if (value < ptr->data)

        insertBST(ptr->left, value);

    else if (value > ptr->data)

        insertBST(ptr->right, value);

    else

        cout << "\nDuplication is not allowed! " << value << " already exists.\n";

}

Source Code: insertBST()

void createBST()

{

    root = NULL; // Reset root for initialization

    insertBST(root, 8);

    insertBST(root, 3);

    insertBST(root, 10);

    insertBST(root, 1);

    insertBST(root, 6);

    insertBST(root, 14);

    insertBST(root, 4);

    insertBST(root, 7);

    insertBST(root, 13);

}

Source Code: createBST()

void freeTree(Node \*root)

{

    if (root == nullptr)

        return;

    freeTree(root->left);

    freeTree(root->right);

    delete root;

}

Source Code: freeTree()

void printTree(Node \*root, int space = 0)

{

    if (root == nullptr)

        return;

    space += 5;

    printTree(root->right, space);

    cout << '\n';

    for (int i = 5; i < space; i++)

    {

        cout << ' ';

    }

   out << root->data;

    printTree(root->left, space);

}

Source Code: printTree()

Node \*inOrderPredecessor(Node \*ptr)

{

    ptr = ptr->left;

    while (ptr->right != NULL)

        ptr = ptr->right;

    return ptr;

}

Source Code: inOrderPredecessor ()

Node \*createNode(int data)

{

    return new Node(data);

}

Source Code: createNode()

Node \*deleteNodeBST(Node \*ptr, int value)

{

    if (ptr == NULL)

    {

        cout << "Value " << value << " not found in the tree.\n";

        return NULL;

    }

    if (value < ptr->data)

    {

        ptr->left = deleteNodeBST(ptr->left, value);

    }

    else if (value > ptr->data)

    {

        ptr->right = deleteNodeBST(ptr->right, value);

    }

    else

    {

        // Node with one child or no child

        if (ptr->left == NULL)

        {

            Node \*temp = ptr->right;

            delete ptr;

            return temp;

        }

        else if (ptr->right == NULL)

        {

            Node \*temp = ptr->left;

            delete ptr;

            return temp;

        }

        // Node with two children

        Node \*inPre = inOrderPredecessor(ptr);

        ptr->data = inPre->data;

        ptr->left = deleteNodeBST(ptr->left, inPre->data);

    }

    return ptr;

}

Source Code: deleteNodeBST()

Menu:

1. Insert Node in the tree

2. Delete Node from tree

0. Exit

14

13

10

8

7

6

4

3

1

Enter your choice: 1

Enter value to insert: 5

Menu:

1. Insert Node in the tree

2. Delete Node from tree

0. Exit

14

13

10

8

7

6

5

4

3

1

Enter your choice: 2

Enter value to delete: 6

Menu:

1. Insert Node in the tree

2. Delete Node from tree

0. Exit

14

13

10

8

7

5

4

3

1

Enter your choice: 0

Exiting...

Output

1. **Write a program to find the height of a binary tree.**

#include <iostream>

using namespace std;

typedef struct Node

{

    Node \*left, \*right;

    int data;

    Node(int data)

    {

        this->left = NULL;

        this->data = data;

        this->right = NULL;

    }

} Node;

Node \*root = NULL;

void createBinaryTree();

Node \*createNode(int);

void freeTree(Node \*root);

void printTree(Node \*, int);

int heightCalculate(Node \*);

int main()

{

    createBinaryTree(); // Initialize the BinaryTree with some predefined values.

    printTree(root, 0);

    cout << "\n\n"

         << "The height of the current tree is: " << heightCalculate(root)

         << "\n\n"

         << endl;

    return 0;

}

Source Code: main()

void insertBST(Node \*&ptr, int value)

{

    if (ptr == NULL)

    {

        ptr = createNode(value);

        return;

    }

    if (value < ptr->data)

        insertBST(ptr->left, value);

    else if (value > ptr->data)

        insertBST(ptr->right, value);

    else

        cout << "\nDuplication is not allowed! " << value << " already exists.\n";

}

Source Code: insertBST()

void createBST()

{

    root = NULL; // Reset root for initialization

    insertBST(root, 8);

    insertBST(root, 3);

    insertBST(root, 10);

    insertBST(root, 1);

    insertBST(root, 6);

    insertBST(root, 14);

    insertBST(root, 4);

    insertBST(root, 7);

    insertBST(root, 13);

}

Source Code: createBST()

void freeTree(Node \*root)

{

    if (root == nullptr)

        return;

    freeTree(root->left);

    freeTree(root->right);

    delete root;

}

Source Code: freeTree()

void printTree(Node \*root, int space = 0)

{

    if (root == nullptr)

        return;

    space += 5;

    printTree(root->right, space);

    cout << '\n';

    for (int i = 5; i < space; i++)

    {

        cout << ' ';

    }

   out << root->data;

    printTree(root->left, space);

}

Source Code: printTree()

int heightCalculate(Node \*root)

{

    if (root == NULL)

        return 0;

    int left\_height = heightCalculate(root->left);

    int right\_height = heightCalculate(root->right);

    return max(left\_height, right\_height) + 1;

}

Source Code: heightCalculate()

Node \*createNode(int data)

{

    return new Node(data);

}

Source Code: createNode()

14

13

10

8

7

6

4

3

1

The height of the current tree is: 4

Output

1. **Implement an algorithm to check if a binary tree is a binary search tree (BST).**

#include <iostream>

using namespace std;

typedef struct Node

{

    Node \*left, \*right;

    int data;

    Node(int data)

    {

        this->left = NULL;

        this->data = data;

        this->right = NULL;

    }

} Node;

Node \*root = NULL;

void createBinaryTree();

Node \*createNode(int);

void freeTree(Node \*root);

void printTree(Node \*, int);

int heightCalculate(Node \*);

int main()

{

    createBinaryTree(); // Initialize the BinaryTree with some predefined values.

    printTree(root, 0);

    cout << "\n\n"

         << "The height of the current tree is: " << heightCalculate(root)

         << "\n\n"

         << endl;

    return 0;

}

Source Code: main()

void createTree()

{

    root = NULL; // Reset root for initialization

    root = createNode(1);

    Node \*node1 = createNode(2);

    Node \*node2 = createNode(3);

    Node \*node3 = createNode(4);

    Node \*node4 = createNode(5);

    Node \*node5 = createNode(6);

    Node \*node6 = createNode(7);

    root->left = node1;

    root->right = node2;

    node1->left = node3;

    node1->right = node4;

    node2->left = node5;

    node2->right = node6;

}

Source Code: createTree()

void printTree(Node \*root, int space = 0)

{

    if (root == nullptr)

        return;

    space += 5;

    printTree(root->right, space);

    cout << '\n';

    for (int i = 5; i < space; i++)

    {

        cout << ' ';

    }

    cout << root->data;

    printTree(root->left, space);

}

Source Code: printTree()

Node \*createNode(int data)

{

    return new Node(data);

}

Source Code: createNode()

void freeTree(Node \*root)

{

    if (root == nullptr)

        return;

    freeTree(root->left);

    freeTree(root->right);

    delete root;

}

Source Code: freeTree()

bool isBST(Node \*root)

{

    if (root == NULL)

        return true;

    if (sizeof(root) > 3)

        return false;

    if (root->left != NULL && root->left->data > root->data)

        return false;

    if (root->right != NULL && root->right->data < root->data)

        return false;

    if (!isBST(root->left) || !isBST(root->right))

        return false;

    return true;

}

Source Code: isBST()

7

3

6

1

5

2

4

The tree is not a Binary Search Tree.

Output

1. **Create a function to find the lowest common ancestor (LCA) Of two nodes in a binary tree.**

#include <iostream>

using namespace std;

// Definition for a binary tree node.

typedef struct Node

{

    int data;

    Node \*left;

    Node \*right;

    Node(int data, Node \*left = NULL, Node \*right = NULL)

    {

        this->data = data;

        this->left = left;

        this->right = right;

    }

} Node;

void printTree(Node \*, int);

Node \*createNode(int val);

Node \*createTree();

Node \*lowestCommonAncestor(Node \*root, Node \*p, Node \*q);

Node \*root = createTree();

int main()

{

    printTree(root, 0);

    Node \*p = root->left->left;

    Node \*q = root->left->right->right;

    Node \*lca = lowestCommonAncestor(root, p, q);

    cout << "\n\nThe lowest common ancestor of " << p->data << " and "

         << q->data << " is " << lca->data

         << endl;

    return 0;

}

Source Code: main()

Node \*createTree()

{

    Node \*root = createNode(5);

    root->left = createNode(3);

    root->right = createNode(1);

    root->left->left = createNode(6);

    root->left->right = createNode(2);

    root->right->left = createNode(0);

    root->right->right = createNode(8);

    root->left->right->left = createNode(7);

    root->left->right->right = createNode(4);

    return root;

}

Source Code: createTree()

void printTree(Node \*root, int space = 0)

{

    if (root == nullptr)

        return;

    space += 5;

    printTree(root->right, space);

    cout << '\n';

    for (int i = 5; i < space; i++)

    {

        cout << ' ';

    }

    cout << root->data;

    printTree(root->left, space);

}

Source Code: printTree()

Node \*createNode(int data)

{

    return new Node(data);

}

Source Code: createNode()

void freeTree(Node \*root)

{

    if (root == nullptr)

        return;

    freeTree(root->left);

    freeTree(root->right);

    delete root;

}

Source Code: freeTree()

Node \*lowestCommonAncestor(Node \*root, Node \*p, Node \*q)

{

    if (root == NULL || root == p || root == q)

        return root;

    Node \*left = lowestCommonAncestor(root->left, p, q);

    Node \*right = lowestCommonAncestor(root->right, p, q);

    if (left == NULL)

        return right;

    else if (right == NULL)

        return left;

    else

        return root;

}

Source Code: lowestCommonAncestor()

8

1

0

5

4

2

7

3

6

The lowest common ancestor of 6 and 4 is 3

Output