**DAY 1 - Arrays, Linked List and String**

**Problem 1: Given an array arr[] of non-negative integers and an integer sum, find a subarray that adds to a given sum**

**Solution:**

In this code, the findSubarrayWithSum function takes an array arr and a target sum targetSum as input and returns a vector containing the subarray that adds up to the given sum. It uses the sliding window technique to efficiently find the subarray.

The main function demonstrates the usage by creating an array arr and a target sum targetSum. It calls the findSubarrayWithSum function and prints the result.

Note: This code assumes that there is at least one subarray that adds up to the given sum. If there can be cases where no subarray is found, you can modify the code accordingly to handle such scenarios.

C++ code

#include <iostream>

#include <vector>

using namespace std;

vector<int> findSubarrayWithSum(vector<int>& arr, int targetSum)

{

int n = arr.size();

int start = 0, end = 0, currentSum = 0;

vector<int> result;

while (end <= n) {

if (currentSum == targetSum) {

// Found subarray with the given sum

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

result.push\_back(arr[i]);

}

break;

}

if (currentSum < targetSum) {

// Include the next element in the current sum

currentSum += arr[end];

end++;

} else {

// Remove elements from the start to adjust the sum

currentSum -= arr[start];

start++;

}

}

return result;

}

int main() {

vector<int> arr = {1, 4, 20, 3, 10, 5};

int targetSum = 33;

vector<int> subarray = findSubarrayWithSum(arr, targetSum);

if (subarray.empty()) {

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

}

else

{

cout << "Subarray found with the given sum: ";

for (int num : subarray)

{

cout << num << " ";

}

cout << endl;

}

return 0;

}

**Problem 2: An array contains both positive and negative numbers in random order. Rearrange the array elements so that all negative numbers appear before all positive numbers.**

Solution: In this code, the function **rearrangeArray** takes a vector of integers (arr) as input and modifies it in place to rearrange the elements. The main function demonstrates an example usage by initializing an array, calling rearrangeArray on it, and then printing the rearranged array. The output will display the rearranged array with negative numbers appearing before positive numbers.

**C++ Code**

#include <iostream>

#include <vector>

void rearrangeArray(std::vector<int>& arr)

{

int left = 0;

int right = arr.size() - 1;

while (left <= right) {

if (arr[left] > 0 && arr[right] < 0) {

std::swap(arr[left], arr[right]);

left++;

right--;

} else if (arr[left] < 0) {

left++;

} else if (arr[right] >= 0) {

right--;

}

}

}

int main() {

std::vector<int> arr = {3, -2, 1, -5, 6, -4};

rearrangeArray(arr);

std::cout << "Rearranged array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

**Problem 3: Given an array, print the Next Greater Element (NGE) for every element.**

**Input: arr[] = [ 4 , 5 , 2 , 25 ]**

Output: 4 –> 5

5 –> 25

2 –> 25

25 –> No greater element found"

Solution: In this code, the findNextGreaterElement function takes an input array (arr) and returns a vector containing the next greater element for each element in the array. It uses a stack to keep track of the indices of elements that are yet to find their next greater element. The main function demonstrates an example usage by initializing an array, calling findNextGreaterElement on it, and then printing the input array and the next greater elements. The output will display the input array and the corresponding next greater elements for each element.

To find the Next Greater Element (NGE) for every element in an array, you can use a stack-based approach. Here's the implementation in C++:

C++ Code:

#include <iostream>

#include <stack>

#include <vector>

std::vector<int> findNextGreaterElement(const std::vector<int>& arr)

{

std::vector<int> result(arr.size(), -1); // Initialize result array with -1

std::stack<int> st;

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

{

while (!st.empty() && arr[i] > arr[st.top()]) {

result[st.top()] = arr[i]; // arr[i] is the next greater element for elements at indices stored in the stack

st.pop();

}

st.push(i); // Push the current index to the stack

}

return result;

}

int main() {

std::vector<int> arr = {4, 5, 2, 25};

std::vector<int> nextGreater = findNextGreaterElement(arr);

std::cout << "Input array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Next Greater Elements: " << std::endl;

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

{

std::cout << arr[i] << " --> ";

if (nextGreater[i] == -1) {

std::cout << "No greater element found";

} else

{

std::cout << nextGreater[i];

}

std::cout << std::endl;

}

return 0;

}

**Problem 4: Two lists are to be maintained containing integer values. The values are not stored in contiguous memory locations. Let one list be 7->1->5 and another list be 5->9->2. Create a third list such that the third list contains the sum of the first two list i.e 517+295 =812 and save it as 2->1>8.**

**Solution**

#include <iostream>

// Definition of a node in the linked list

struct Node {

int value;

Node\* next;

};

// Function to create a new node with the given value

Node\* createNode(int value)

{

Node\* newNode = new Node();

newNode->value = value;

newNode->next = nullptr;

return newNode;

}

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

void insertNode(Node\*\* head, Node\* newNode) {

if (\*head == nullptr) {

\*head = newNode;

} else

{

Node\* temp = \*head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Function to add two linked lists and return the result

Node\* addLists(Node\* list1, Node\* list2) {

Node\* result = nullptr;

Node\* current = nullptr;

int carry = 0;

while (list1 != nullptr || list2 != nullptr) {

int sum = carry + (list1 != nullptr ? list1->value : 0) + (list2 != nullptr ? list2->value : 0);

carry = sum / 10; // Calculate the carry

Node\* newNode = createNode(sum % 10); // Calculate the value to be stored in the new node

if (result == nullptr) {

result = newNode;

current = result;

} else {

current->next = newNode;

current = current->next;

}

if (list1 != nullptr) {

list1 = list1->next;

}

if (list2 != nullptr) {

list2 = list2->next;

}

}

if (carry != 0) {

Node\* newNode = createNode(carry);

current->next = newNode;

}

return result;

}

// Function to display the linked list

void displayList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

std::cout << temp->value;

if (temp->next != nullptr) {

std::cout << "->";

}

temp = temp->next;

}

std::cout << std::endl;

}

// Function to deallocate memory for the linked list

void deleteList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

Node\* nextNode = temp->next;

delete temp;

temp = nextNode;

}

}

int main() {

// Create the first linked list: 7->1->5

Node\* list1 = createNode(7);

insertNode(&list1, createNode(1));

insertNode(&list1, createNode(5));

// Create the second linked list: 5->9->2

Node\* list2 = createNode(5);

insertNode(&list2, createNode(9));

insertNode(&list2, createNode(2));

// Add the two linked lists

Node\* sumList = addLists(list1, list2);

// Display the result

std::cout << "Sum of the lists: ";

displayList(sumList);

// Deallocate memory for the linked lists

deleteList(list1);

deleteList(list2);

deleteList(sumList);

return 0;

}

In this code, we define a Node struct to represent each node in the linked list

**Problem 5 - You are in the process of creating a text editor (like notepad /Ms Word). The text editor saves a particular word in the form of a string. We need to provide the feature of Find and Replace a particular substring in the given text editor.**

To provide the feature of Find and Replace in a text editor using C++, you can follow these steps:

1. Obtain the input text: Start by asking the user to enter the text they want to edit. You can store the input text in a string variable.

std::string text;

std::cout << "Enter the text: ";

std::getline(std::cin, text);

1. Get the substring to find: Ask the user to input the substring they want to find in the text. Store it in a string variable.

std::string findSubstring;

std::cout << "Enter the substring to find: ";

std::getline(std::cin, findSubstring);

1. Get the replacement string: Ask the user to enter the replacement string they want to use. Store it in another string variable.

std::string replaceString;

std::cout << "Enter the replacement string: ";

std::getline(std::cin, replaceString);

1. Perform the find and replace operation: Use the **find()** function of the **std::string** class to locate the first occurrence of the substring within the text. If a match is found, use the **replace()** function to replace the substring with the replacement string. Repeat this process until all occurrences of the substring are replaced.

size\_t pos = 0;

while ((pos = text.find(findSubstring, pos)) != std::string::npos) {

text.replace(pos, findSubstring.length(), replaceString);

pos += replaceString.length();

}

1. Display the modified text: Finally, print the modified text after the find and replace operation is complete.

std::cout << "Modified Text: " << text << std::endl;

Here's the complete code for the Find and Replace feature in a text editor:

#include <iostream>

#include <string>

int main() {

std::string text;

std::cout << "Enter the text: ";

std::getline(std::cin, text);

std::string findSubstring;

std::cout << "Enter the substring to find: ";

std::getline(std::cin, findSubstring);

std::string replaceString;

std::cout << "Enter the replacement string: ";

std::getline(std::cin, replaceString);

size\_t pos = 0;

while ((pos = text.find(findSubstring, pos)) != std::string::npos) {

text.replace(pos, findSubstring.length(), replaceString);

pos += replaceString.length();

}

std::cout << "Modified Text: " << text << std::endl;

return 0;

}

**Problem 6 -** **Cyclic Rotation of Array**

Cyclic rotation of an array involves shifting each element of the array to the right by one position, and moving the last element to the first position. This operation is performed repeatedly for a given number of rotations.

Here's an example to illustrate the cyclic rotation of an array:

Initial array: [1, 2, 3, 4, 5]

Cyclic rotation 1: [5, 1, 2, 3, 4]

Cyclic rotation 2: [4, 5, 1, 2, 3]

Cyclic rotation 3: [3, 4, 5, 1, 2]

#include <iostream>

using namespace std;

void cyclicRotation(int arr[], int n) {

int lastElement = arr[n - 1];

// Shift elements to the right

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

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

}

// Move the last element to the first position

arr[0] = lastElement;

}

int main() {

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

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

cout << "Original Array: ";

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

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

}

cyclicRotation(arr, n);

cout << "\nArray after cyclic rotation: ";

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

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

}

return 0;

}

Explanation:

1. The **cyclicRotation** function takes an array **arr** and its size **n** as input parameters.
2. It stores the last element of the array in a variable called **lastElement**.
3. The for loop starts from the second-to-last element and iterates until the first element of the array. In each iteration, it moves the current element one position to the right by assigning the value of the previous element to the current element (**arr[i] = arr[i - 1]**).
4. After the loop, the first position of the array is updated with the **lastElement**, effectively moving the last element to the first position.
5. In the **main** function, an example array **{1, 2, 3, 4, 5}** is created, and its size is calculated using the **sizeof** operator.
6. The original array is printed using a for loop.
7. The **cyclicRotation** function is called to rotate the array.
8. Finally, the array after cyclic rotation is printed using another for loop.

Original Array: 1 2 3 4 5

Array after cyclic rotation: 5 1 2 3 4

As you can see, the elements have been shifted to the right, and the last element (5) has moved to the first position, resulting in a cyclic rotation of the array.

**Problem 7: Converting General tree to Doubly Linked list**

Solution: To convert a general tree to a doubly linked list in C++, you can use a recursive approach. Here's an implementation:

#include <iostream>

#include <vector>

// Structure for a tree node

struct TreeNode {

int data;

std::vector<TreeNode\*> children;

TreeNode(int val) : data(val) {}

};

// Structure for a doubly linked list node

struct DoublyListNode {

int data;

DoublyListNode\* prev;

DoublyListNode\* next;

DoublyListNode(int val) : data(val), prev(nullptr), next(nullptr) {}

};

// Function to convert a general tree to a doubly linked list

DoublyListNode\* convertToDoublyLinkedList(TreeNode\* root)

{

if (root == nullptr) {

return nullptr;

}

DoublyListNode\* head = new DoublyListNode(root->data);

DoublyListNode\* current = head;

for (TreeNode\* child : root->children) {

DoublyListNode\* childList = convertToDoublyLinkedList(child);

current->next = childList;

if (childList != nullptr) {

childList->prev = current;

while (current->next != nullptr) {

current = current->next;

}

}

}

return head;

}

// Function to print the doubly linked list

void printDoublyLinkedList(DoublyListNode\* head) {

DoublyListNode\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

int main() {

// Create a general tree

TreeNode\* root = new TreeNode(1);

TreeNode\* node2 = new TreeNode(2);

TreeNode\* node3 = new TreeNode(3);

TreeNode\* node4 = new TreeNode(4);

TreeNode\* node5 = new TreeNode(5);

TreeNode\* node6 = new TreeNode(6);

TreeNode\* node7 = new TreeNode(7);

root->children = {node2, node3};

node2->children = {node4, node5};

node3->children = {node6, node7};

// Convert the general tree to a doubly linked list

DoublyListNode\* doublyLinkedList = convertToDoublyLinkedList(root);

// Print the doubly linked list

std::cout << "Doubly Linked List: ";

printDoublyLinkedList(doublyLinkedList);

return 0;

}

In this code, the **TreeNode** structure represents a node in the general tree, and the **DoublyListNode** structure represents a node in the doubly linked list. The c**onvertToDoublyLinkedList** function takes the root of the general tree and recursively converts it into a doubly linked list. It returns the head of the resulting list.

The **printDoublyLinkedList** function traverses the doubly linked list and prints its contents.

In the **main** function, a general tree is created, and then it is converted to a **doubly linked list** using the **convertToDoublyLinkedList** function. Finally, the resulting doubly linked list is printed using the **printDoublyLinkedList** function.