**Roll No…………….. Total No. of Pages:……**

**ST-2 (SET-III)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. What is the result of calling a recursive function with a very large parameter value in C++?
   1. It will cause a segmentation fault.
   2. **It will throw a stack overflow exception.**
   3. It will cause an infinite loop.
   4. It will execute the function normally.
2. In C++, which type of recursion involves a function calling itself indirectly through other functions?
   1. **Indirect recursion**
   2. Tail recursion
   3. Linear recursion
   4. Direct recursion
3. In backtracking, how do you know when a solution is valid?
   1. By checking if the solution satisfies the base case condition
   2. By comparing it to a predefined solution
   3. By ensuring it doesn't involve any recursive calls
   4. **By testing it against a set of constraints**
4. What is the purpose of the "visited" array in a backtracking solution?
   1. To keep track of the number of recursive calls
   2. **To mark which elements have been explored or used in the solution**
   3. To store the computed results of subproblems
   4. To control the flow of execution in the recursive function
5. In a problem involving finding connected components in a grid, what is a connected component?
   1. **A set of cells that are adjacent to each other**
   2. A subset of the matrix
   3. A sequence of cells that can be traversed linearly
   4. A group of cells that are the same color
6. What is the purpose of a base case in a recursive algorithm on a matrix?
   1. To mark the starting point of the recursion
   2. **To handle the smallest sub-problems directly**
   3. To store the computed values for memoization
   4. To keep track of visited cells
7. When is the destructor called?
   1. When an object is created
   2. **When an object goes out of scope**
   3. When an object is copied
   4. When an object is passed by value to a function
8. Can a class have both a copy constructor and a copy assignment operator?
   1. **Yes, but they should have different names**
   2. No, they are mutually exclusive
   3. Yes, but the copy assignment operator is optional
   4. Yes, but they should take different arguments
9. What is the primary disadvantage of using a linked list over an array?
   1. **Slower random access**
   2. Inability to store data
   3. Limited memory usage
   4. Difficulty in implementation
10. Which type of linked list is used to implement a dynamic array?
    1. **Singly linked list**
    2. Doubly linked list
    3. Circular linked list
    4. Array-based list

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

11) struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

int main() {

Node\* head = new Node(1);

Node\* temp = head;

head = nullptr;

delete temp;

cout << temp->data;

return 0;

}

**a) 0**

b) 1

c) Error

d) Garbage Value

12) #include <iostream>

using namespace std;

void fun(int n) {

if (n <= 0)

return;

fun(n / 10);

cout << n % 10 << " ";

}

int main() {

fun(572);

return 0;

}

a) 2 7 5

**b) 5 7 2**

c) 7 5 2

d) 5 2 7

13) What is the role of below function?

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

if (board[i][col] == 1) {

return false;

}

}

a) check if there is 1 in upper left diagonal

b) check if there is 1 in upper right diagonal

**c) check if there is 1 in same column**

d) check if there is 1 in same row

14) class Shape {

public:

virtual void draw() { cout << "Shape"; }

};

class Circle : public Shape {

public:

void draw() { cout << "Circle"; }

};

int main() {

Shape \*shapes[2];

shapes[0] = new Shape;

shapes[1] = new Circle;

for (int i = 0; i < 2; ++i)

shapes[i]->draw();

return 0;

}

a) ShapeShape

b) CircleShape

**c) ShapeCircle**

d) CircleCircle

15) What is the time complexity of the following recursive function, which calculates the nth Fibonacci number using a naive approach?

int fibonacci(int n) {

if (n <= 1)

return n;

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

}

a) O(n)

**b) O(2^n)**

c) O(n^2)

d) O(log n)

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Implement the Merge Sort algorithm using recursion to sort the elements in array.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Arr= 8 4 12 42 7 | Arr= 21 45 16 7 4 | Arr= 99 14 6 87 101 |
| **Output** | Sorted array: 4 7 8 12 42 | Sorted array: 4 7 16 21 45 | Sorted array: 6 14 87 99 101 |

Solution :

**#include <iostream>**

**#include <vector>**

**// Merge two sorted subarrays and combine them into a single sorted array**

**void merge(std::vector<int>& arr, int left, int mid, int right) {**

**int n1 = mid - left + 1; // Size of the left subarray**

**int n2 = right - mid; // Size of the right subarray**

**// Create temporary vectors to hold the left and right subarrays**

**std::vector<int> leftArray(n1);**

**std::vector<int> rightArray(n2);**

**// Copy data from the original array to the temporary left and right subarrays**

**for (int i = 0; i < n1; ++i)**

**leftArray[i] = arr[left + i];**

**for (int j = 0; j < n2; ++j)**

**rightArray[j] = arr[mid + 1 + j];**

**int i = 0; // Initial index of the left subarray**

**int j = 0; // Initial index of the right subarray**

**int k = left; // Initial index of the merged subarray**

**// Merge the two subarrays back into the original array in sorted order**

**while (i < n1 && j < n2) {**

**if (leftArray[i] <= rightArray[j]) {**

**arr[k] = leftArray[i];**

**++i;**

**} else {**

**arr[k] = rightArray[j];**

**++j;**

**}**

**++k;**

**}**

**// Copy the remaining elements, if any, from the left subarray**

**while (i < n1) {**

**arr[k] = leftArray[i];**

**++i;**

**++k;**

**}**

**// Copy the remaining elements, if any, from the right subarray**

**while (j < n2) {**

**arr[k] = rightArray[j];**

**++j;**

**++k;**

**}**

**}**

**// Recursive function to sort an array using Merge Sort**

**void mergeSort(std::vector<int>& arr, int left, int right) {**

**if (left < right) {**

**int mid = left + (right - left) / 2; // Calculate the middle index**

**// Recursive call to sort the left and right halves of the array**

**mergeSort(arr, left, mid);**

**mergeSort(arr, mid + 1, right);**

**// Merge the sorted halves**

**merge(arr, left, mid, right);**

**}**

**}**

**int main() {**

**int n;**

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

**std::cin >> n;**

**std::vector<int> arr(n);**

**std::cout << "Enter the array elements: ";**

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

**std::cin >> arr[i];**

**mergeSort(arr, 0, n - 1);**

**std::cout << "Sorted array: ";**

**for (int num : arr)**

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

**std::cout << std::endl;**

**return 0;**

**}**

Q17) Implement a singly linked list and delete the first occurrence of a given value from the list.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Linked list=3 5 9 12 5 8  Value= 5 | Linked list= 98 14 65 78 9 4  Value= 65 | Linked list= 11 22 33 44 55  Value= 44 |
| **Output** | Linked List after deleting nodes: 3 9 12 5 8 | Linked List after deleting nodes: 98 14 78 9 4 | Linked List after deleting nodes: 11 22 33 55 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Node structure for singly linked list**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**// Function to insert an element at the end of the linked list**

**Node\* insertAtEnd(Node\* head, int data) {**

**Node\* newNode = new Node;**

**newNode->data = data;**

**newNode->next = nullptr;**

**if (head == nullptr) {**

**return newNode;**

**}**

**Node\* current = head;**

**while (current->next != nullptr) {**

**current = current->next;**

**}**

**current->next = newNode;**

**return head;**

**}**

**// Function to display the linked list**

**void displayLinkedList(Node\* head) {**

**Node\* current = head;**

**while (current != nullptr) {**

**cout << current->data << " ";**

**current = current->next;**

**}**

**}**

**// Function to delete the first occurrence of a given value from the linked list**

**Node\* deleteFirstNodeWithValue(Node\* head, int value) {**

**if (head == nullptr) {**

**return nullptr;**

**}**

**if (head->data == value) {**

**Node\* temp = head;**

**head = head->next;**

**delete temp;**

**return head;**

**}**

**Node\* current = head;**

**Node\* prev = nullptr;**

**while (current != nullptr) {**

**if (current->data == value) {**

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

**delete current;**

**break;**

**}**

**prev = current;**

**current = current->next;**

**}**

**return head;**

**}**

**int main() {**

**Node\* head = nullptr;**

**int data;**

**// Create the linked list with user input**

**cout << "Enter elements for linked list (enter -1 to stop):\n";**

**while (true) {**

**cin >> data;**

**if (data == -1)**

**break;**

**head = insertAtEnd(head, data);**

**}**

**// Display the linked list before deleting nodes**

**cout << "Linked List before deleting nodes: ";**

**displayLinkedList(head);**

**// Prompt the user to enter the value to be deleted**

**int valueToDelete;**

**cout << "\nEnter the value to be deleted from the linked list: ";**

**cin >> valueToDelete;**

**// Delete the first node with the given value from the linked list**

**head = deleteFirstNodeWithValue(head, valueToDelete);**

**// Display the linked list after deleting nodes**

**cout << "Linked List after deleting nodes: ";**

**displayLinkedList(head);**

**// Free memory by deleting nodes**

**while (head != nullptr) {**

**Node\* temp = head;**

**head = head->next;**

**delete temp;**

**}**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) You are given a lock which is made up of n-different circular rings and each ring has 0-9 digit printed serially on it. Initially all n-rings together show a n-digit integer but there is particular code only which can open the lock.

You can rotate each ring any number of time in either direction. You have to find the minimum number of rotation done on rings of lock to open the lock.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Input = 1919, Unlock code = 0000 | Input = 28756, Unlock code = 98234 | Input = 1232, Unlock code = 1232 |
| **Output** | Rotations required = 4 | Rotations required = 12 | Rotations required = 0 |

Solution :

**#include <bits/stdc++.h>**

**using namespace std;**

**// function for min rotation**

**int minRotation(int input, int unlock\_code)**

**{**

**int rotation = 0;**

**int input\_digit, code\_digit;**

**// iterate till input and unlock code become 0**

**while (input || unlock\_code) {**

**// input and unlock last digit as reminder**

**input\_digit = input % 10;**

**code\_digit = unlock\_code % 10;**

**// find min rotation**

**rotation += min(abs(input\_digit - code\_digit),**

**10 - abs(input\_digit - code\_digit));**

**// update code and input**

**input /= 10;**

**unlock\_code /= 10;**

**}**

**return rotation;**

**}**

**int main()**

**{**

**int input = 1232;**

**int unlock\_code = 1232;**

**cout << "Minimum Rotation = "**

**<< minRotation(input, unlock\_code);**

**return 0;**

**}**