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

**ST-2 (SET-IV)**

**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. In C++, what is the result of calling a recursive function with a negative base case value?
   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 trigger the base case immediately.**
2. Which type of recursion involves a function calling itself directly at the end of the function?
   1. Indirect recursion
   2. Head recursion
   3. **Tail recursion**
   4. Multiple recursion
3. Which operation is not possible in a circular linked list?
   1. Insertion at the beginning
   2. Insertion at the end
   3. Deletion from the beginning
   4. **Deletion from the middle**
4. What is abstraction in C++?
   1. A way to provide multiple implementations for a function based on its arguments
   2. A way to create multiple instances of a class
   3. **A way to hide the complexity of implementation and show only the necessary features**
   4. A way to expose all class members publicly
5. Which concept is closely related to abstraction in C++?
   1. Inheritance
   2. **Data hiding**
   3. Polymorphism
   4. Dynamic binding
6. Which of the following is not a storage class in C++?
   1. auto
   2. register
   3. **class**
   4. static
7. What is the purpose of the "this" pointer in C++?
   1. **To point to the current object**
   2. To point to the previous object
   3. To point to the next object
   4. To point to a null object
8. In a maze-solving problem using recursive backtracking, what does the term "backtracking" refer to?
   1. Going back to the beginning of the maze
   2. Revisiting cells that have already been visited
   3. **Undoing a move and exploring a different path**
   4. Tracing the steps taken in the maze
9. What is the role of recursion in problems that involve permutation of elements in a grid?
   1. To ensure that all permutations are unique
   2. To create a new grid with permuted elements
   3. **To generate all possible permutations**
   4. To sort the elements in the grid
10. In backtracking, what is the term for the process of moving back one step to a previous state?
    1. **Undoing**
    2. Pruning
    3. Reverting
    4. Back-jumping

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

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

11) #include <iostream>

using namespace std;

int recursiveSum(int n) {

if (n <= 0)

return 0;

return n + recursiveSum(n - 2);

}

int main() {

cout << recursiveSum(7);

return 0;

}

**a) 16**

b) 20

c) 18

d) 15

12) #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

int main() {

Node\* head = nullptr;

Node\* temp = new Node;

temp->data = 5;

temp->next = nullptr;

head = temp;

cout << head->data;

return 0;

}

1. 0
2. **5**
3. Error
4. Garbage Value

13) class Animal {

public:

void speak() { cout << "Animal"; }

};

class Dog : public Animal {

public:

void speak() { cout << "Dog"; }

};

int main() {

Dog d;

Animal \*ptr = &d;

ptr->speak();

return 0;

}

1. **Animal**
2. Dog
3. Compile Error
4. Undefined Behavior

14) What is the operation performed by given function?

void fun(string str, int l, int r) {

if (l == r) {

cout << str << endl;

return;

}

for (int i = l; i <= r; i++) {

swap(str[l], str[i]);

fun(str, l + 1, r);

swap(str[l], str[i]); // Backtrack

}

}

a) print string in reverse order

**b) print permutations of string**

c) print anagrams of string

d) swapping string character at first and last location

15) What is the time complexity of the following recursive function that calculates the factorial of a positive integer n?

int factorial(int n) {

if (n <= 1)

return 1;

return n \* factorial(n - 1);

}

**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 a linked list and find the middle node.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Linked List: 4 6 8 9 12 | Linked List: 55 14 12 8 | Linked List: 9 8 7 5 4 6 17 |
| **Output** | Middle Node: 8 | Middle Node: 12 | Middle Node: 5 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Node structure for 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 find the middle node of the linked list**

**Node\* findMiddleNode(Node\* head) {**

**if (head == nullptr || head->next == nullptr) {**

**return head;**

**}**

**Node\* slow = head;**

**Node\* fast = head;**

**while (fast != nullptr && fast->next != nullptr) {**

**slow = slow->next;**

**fast = fast->next->next;**

**}**

**return slow;**

**}**

**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**

**cout << "Linked List: ";**

**Node\* current = head;**

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

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

**current = current->next;**

**}**

**// Find the middle node of the linked list**

**Node\* middleNode = findMiddleNode(head);**

**if (middleNode != nullptr) {**

**cout << "\nMiddle Node: " << middleNode->data;**

**} else {**

**cout << "\nLinked List is empty.";**

**}**

**// Free memory by deleting nodes**

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

**Node\* temp = head;**

**head = head->next;**

**delete temp;**

**}**

**return 0;**

**}**

Q17) Write a recursive function to find the Greatest Common Divisor (GCD) of two non-negative integers.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter two non-negative integers: 35 90 | Enter two non-negative integers: 56 16 | Enter two non-negative integers: 17 9 |
| **Output** | GCD of 35 and 90 is 5 | GCD of 56 and 16 is 8 | GCD of 17 and 9 is 1 |

Solution :

**#include <iostream>**

**// Recursive function to calculate GCD**

**int gcd(int a, int b) {**

**// Base case: If the second number (b) is 0, the GCD is found and it is 'a'.**

**if (b == 0)**

**return a;**

**// Recursive step: Calculate the GCD by calling the function with the second number (b)**

**// as the first number and the remainder of 'a' divided by 'b' as the second number.**

**return gcd(b, a % b);**

**}**

**int main() {**

**int a, b;**

**std::cout << "Enter two non-negative integers: ";**

**std::cin >> a >> b;**

**// Call the gcd function with the two input non-negative integers.**

**int result = gcd(a, b);**

**// Output the GCD.**

**std::cout << "GCD of " << a << " and " << b << " is " << result << std::endl;**

**return 0;**

**}**

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

Q18) Given two words, beginWord and endWord, and a dictionary of word transformations, find the length of the shortest transformation sequence from beginWord to endWord. Each transformation should be made only if the resulting word is in the dictionary.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | beginWord = "hit"  endWord = "cog"  wordList = {"hot", "dot", "dog", "lot", "log", "cog"} | beginWord = "hit"  endWord = "cog"  wordList = {"hot","dot","dog","lot","log"} | beginWord = "bit";  endWord = "bot";  wordList = {"kit","sit","fog","kot","bot"} |
| **Output** | Length of the shortest transformation sequence: 5 | Length of the shortest transformation sequence: 0 | Length of the shortest transformation sequence: 2 |

Solution :

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**#include <unordered\_set>**

**using namespace std;**

**int ladderLength(string beginWord, string endWord, vector<string>& wordList) {**

**unordered\_set<string> dict(wordList.begin(), wordList.end());**

**if (dict.find(endWord) == dict.end())**

**return 0;**

**queue<string> q;**

**q.push(beginWord);**

**int level = 1;**

**while (!q.empty()) {**

**int size = q.size();**

**for (int i = 0; i < size; i++) {**

**string word = q.front();**

**q.pop();**

**for (int j = 0; j < word.length(); j++) {**

**char original = word[j];**

**for (char ch = 'a'; ch <= 'z'; ch++) {**

**word[j] = ch;**

**if (word == endWord)**

**return level + 1;**

**if (dict.find(word) != dict.end()) {**

**q.push(word);**

**dict.erase(word);**

**}**

**}**

**word[j] = original;**

**}**

**}**

**level++;**

**}**

**return 0;**

**}**

**int main() {**

**string beginWord = "hit";**

**string endWord = "cog";**

**vector<string> wordList = {"hot", "dot", "dog", "lot", "log", "cog"};**

**int length = ladderLength(beginWord, endWord, wordList);**

**cout << "Length of the shortest transformation sequence: " << length;**

**return 0;**

**}**