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

**ST-2 (SET-VI)**

**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 space complexity of a linked list?
   1. O(1)
   2. **O(n)**
   3. O(log n)
   4. O(n^2)
2. Which node must be updated when inserting a new node at the beginning of a singly linked list?
   1. Previous node
   2. Last node
   3. **Head node**
   4. Tail node
3. Which type of inheritance involves one base class and one derived class?
   1. **Single inheritance**
   2. Multiple inheritance
   3. Hierarchical inheritance
   4. Multilevel inheritance
4. Which keyword is used to derive a class publicly in C++?
   1. extends
   2. base
   3. inherits
   4. **public**
5. What is the purpose of the "const" keyword in a member function declaration?
   1. **To indicate that the function is constant and won't modify the object**
   2. To indicate that the function cannot be called
   3. To indicate that the function is for internal use only
   4. To indicate that the function has a constant return value
6. Which of the following statements is true about destructors?
   1. Destructors are called explicitly using their names
   2. Destructors can take parameters
   3. **Destructors are automatically called when an object goes out of scope**
   4. Destructors can return values
7. What type of recursion involves a function calling itself with a modified parameter?
   1. Indirect recursion
   2. Tail recursion
   3. Helper recursion
   4. **Direct recursion**
8. When implementing a problem involving movement on a grid, what can be done to prevent going out of bounds?
   1. Use a try-catch block
   2. **Check the boundaries before making a move**
   3. Change the starting position
   4. Use a bigger grid
9. Which of the following problems is best suited for backtracking?
   1. Finding the maximum element in an array
   2. Calculating the factorial of a number
   3. **Solving a maze to find the shortest path**
   4. Counting the number of occurrences of a character in a string
10. A recursive function is called with an argument value of 4. How many times will the function be called if it contains only one recursive call and the base case is reached when the argument value becomes 1?
    1. 1
    2. **2**
    3. 3
    4. 4

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

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

11) #include <iostream>

using namespace std;

int fun(int a, int b) {

if (b == 0)

return 1;

return base \* power(a, b - 1);

}

int main() {

cout << fun(2, 3);

return 0;

}

a) 2

b) 6

**c) 8**

d) 16

12) #include <iostream>

using namespace std;

void printSubsetSum(int arr[], int n, int sum, int idx, int currentSum) {

if (idx == n) {

if (currentSum == sum) {

cout << "Subset found with sum " << sum << endl;

}

return;

}

printSubsetSum(arr, n, sum, idx + 1, currentSum);

printSubsetSum(arr, n, sum, idx + 1, currentSum + arr[idx]);

}

int main() {

int arr[] = {2, 4, 6, 8};

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

int sum = 10;

printSubsetSum(arr, n, sum, 0, 0);

return 0;

}

a) Subset found with sum 10

**b) Subset found with sum 10**

**Subset found with sum 10**

c) Subset found with sum 18

d) No subset found with the given sum

13) struct Node {

int data;

Node\* next;

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

};

int main() {

Node\* head = new Node(2);

head->next = new Node(4);

cout << head->next->data;

return 0;

}

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

14) class Base {

public:

void show() { cout << "Base"; }

};

class Derived : public Base {

public:

void show() { cout << "Derived"; }

};

int main() {

Derived d;

Base \*ptr = &d;

ptr->show();

return 0;

}

1. **Base**
2. Derived
3. Compile Error
4. Undefined Behavior

15) Consider the following recursive function that calculates the sum of elements in an array:

int arraySum(int arr[], int n) {

if (n <= 0)

return 0;

return arr[n - 1] + arraySum(arr, n - 1);

}

What is the time complexity of this function?

a) O(1)

b) O(log n)

**c) O(n)**

d) O(n^2)

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

Q16) Write a recursive function to check if a string is a palindrome.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | madam | tools | racecar |
| **Output** | The string is a palindrome. | The string is not a palindrome. | The string is a palindrome. |

Solution :

**#include <iostream>**

**#include <string>**

**// Recursive function to check palindrome**

**bool isPalindrome(const std::string& str, int start, int end) {**

**// Base case: If the start index is greater than or equal to the end index,**

**// it means all characters in the string have been compared, and it is a palindrome.**

**if (start >= end)**

**return true;**

**// If the characters at the start and end positions are not equal,**

**// it means the string is not a palindrome.**

**if (str[start] != str[end])**

**return false;**

**// Recursive step: Call the function with updated start and end positions.**

**// Continue checking characters towards the middle of the string.**

**return isPalindrome(str, start + 1, end - 1);**

**}**

**int main() {**

**std::string str;**

**std::cout << "Enter a string: ";**

**std::cin >> str;**

**// Call the isPalindrome function with start index 0 and end index (length - 1).**

**bool palindrome = isPalindrome(str, 0, str.length() - 1);**

**// Output whether the string is a palindrome or not.**

**if (palindrome)**

**std::cout << "The string is a palindrome." << std::endl;**

**else**

**std::cout << "The string is not a palindrome." << std::endl;**

**return 0;**

**}**

Q17) Implement a linked list and insert elements at the beginning.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter elements for linked list (enter -1 to stop): 4 5 3 -1 | Enter elements for linked list (enter -1 to stop): 12 17 19 8 -1 | Enter elements for linked list (enter -1 to stop): 7 25 -1 |
| **Output** | Linked List: 3 5 4 | Linked List: 8 19 17 12 | Most frequent element: 25 7 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Node structure for linked list**

**struct Node {**

**int data;**

**Node\* next;**

**};**

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

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

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

**newNode->data = data;**

**newNode->next = head;**

**return newNode;**

**}**

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

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

**Node\* current = head;**

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

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

**current = current->next;**

**}**

**}**

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

**}**

**// Display the linked list**

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

**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) Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

Constraints: String of digits should contain digits from 2 to 9 only.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter the string of digits (2-9): 23 | Enter the string of digits (2-9): 73 | Enter the string of digits (2-9): 8 |
| **Output** | Letter Combinations:  ad  ae  af  bd  be  bf  cd  ce  cf | Letter Combinations:  pd  pe  pf  qd  qe  qf  rd  re  rf  sd  se  sf | Letter Combinations:  t  u  v |

Solution :

**#include <iostream>**

**#include <vector>**

**#include <string>**

**using namespace std;**

**const vector<string> phoneMap = {**

**"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"**

**};**

**void generateLetterCombinations(string digits, int index, string current, vector<string>& result) {**

**if (index == digits.length()) {**

**result.push\_back(current);**

**return;**

**}**

**string letters = phoneMap[digits[index] - '2'];**

**for (char ch : letters) {**

**generateLetterCombinations(digits, index + 1, current + ch, result);**

**}**

**}**

**vector<string> letterCombinations(string digits) {**

**vector<string> result;**

**if (digits.empty())**

**return result;**

**generateLetterCombinations(digits, 0, "", result);**

**return result;**

**}**

**int main() {**

**string digits;**

**cout << "Enter the string of digits (2-9): ";**

**cin >> digits;**

**vector<string> result = letterCombinations(digits);**

**cout << "Letter Combinations:\n";**

**for (const string& combination : result) {**

**cout << combination << endl;**

**}**

**return 0;**

**}**