Rohit Sahu
22BCS13677
IOT -615-B

DAY-3

Q1-Fibonacci Series Using Recursion

```
#include <iostream>
using namespace std;

int fibonacci(int n) {
if (n <= 1) {
return n;      } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    int n_terms;

    cout << "Enter the number of terms in the Fibonacci series: ";     cin
>> n_terms;

    if (n_terms <= 0) {
        cout << "Please enter a posiMve integer." << endl;
    } else {
        cout << "Fibonacci series:" << endl;
for (int i = 0; i < n_terms; i++) {
            cout << fibonacci(i) << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output-

```
Enter the number of terms in the Fibonacci series: 7
Fibonacci series:
0 1 1 2 3 5 8
```

Q2- Reverse Linked List

```
#include <iostream>
using namespace std;

struct ListNode {
int value;
   ListNode* next;

   ListNode(int val) : value(val), next(nullptr) {}
};
ListNode* reverseLinkedList(ListNode* head) {
   ListNode* prev = nullptr;
   ListNode* current = head;

   while (current != nullptr) {
      ListNode* nextNode = current->next;         current-
>next = prev;                    prev =
current;
      current = nextNode;
   }

   return prev;
}

ListNode* createLinkedList() {
   int n;
   cout << "Enter the number of nodes: ";
cin >> n;

   if (n <= 0) {
```

```
        cout << "The list is empty." << endl;
        return nullptr;
    }

    cout << "Enter the values for the nodes:" << endl;     int
value;
    cin >> value;
    ListNode* head = new ListNode(value);
    ListNode* current = head;

    for (int i = 1; i < n; i++) {
cin >> value;
        current->next = new ListNode(value);
current = current->next;
    }

    return head;
}
void  printLinkedList(ListNode*  head)
{     ListNode* current = head;     while
(current != nullptr) {         cout <<
current->value;
        if (current->next != nullptr) cout << " -> ";
current = current->next;
    }
    cout << endl;
}

int main() {
    ListNode* head = createLinkedList();

    cout << "Original list:" << endl;
printLinkedList(head);

    ListNode* reversedHead = reverseLinkedList(head);      cout << "Reversed
list:" << endl;
printLinkedList(reversedHead);
```

```
    return 0;
}
```

Output-

```
Enter the number of nodes: 3
Enter the values for the nodes:
2 33 5
Original list:
2 -> 33 -> 5
Reversed list:
5 -> 33 -> 2
```

Q3- Add Two Numbers

```
#include <iostream>
using namespace std;

struct ListNode {
int value;
    ListNode* next;
    ListNode(int val) : value(val), next(nullptr) {}
};
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummyHead = new
ListNode(0);      ListNode* current =
dummyHead;      int carry = 0;

    while (l1 != nullptr || l2 != nullptr || carry != 0) {
int val1 = (l1 != nullptr) ? l1->value : 0;         int val2
= (l2 != nullptr) ? l2->value : 0;

        int sum = val1 + val2 + carry;
carry = sum / 10;
        current->next = new ListNode(sum % 10);

        current = current->next;
if (l1 != nullptr) l1 = l1->next;
```

```
if (l2 != nullptr) l2 = l2->next;
    }

    return dummyHead->next;
}

ListNode* createLinkedList() {
    int n;
    cout << "Enter the number of nodes: ";
cin >> n;

    if (n <= 0) {
        cout << "The list is empty." << endl;
return nullptr;
    }

    cout << "Enter the values for the nodes:" << endl;     int
value;
    cin >> value;
    ListNode* head = new ListNode(value);
    ListNode* current = head;

    for (int i = 1; i < n; i++) {
        cin >> value;
        current->next = new ListNode(value);
current = current->next;
    }

    return head;
}

void  printLinkedList(ListNode*  head)
{     ListNode* current = head;     while
(current != nullptr) {         cout <<
current->value;
        if (current->next != nullptr) cout << " -> ";
current = current->next;
    }
```

```
    cout << endl;
}

int main() {
    cout << "Enter the first number as a linked list:" << endl;
ListNode* l1 = createLinkedList();

    cout << "Enter the second number as a linked list:" << endl;
    ListNode* l2 = createLinkedList();

    ListNode* result = addTwoNumbers(l1, l2);

    cout << "The sum is:" << endl;
    printLinkedList(result);

    return 0;
}
```

Output-

```
Enter the first number as a linked list:
Enter the number of nodes: 3
Enter the values for the nodes:
2 33 5
Enter the second number as a linked list:
Enter the number of nodes: 2
Enter the values for the nodes:
12 3
The sum is:
4 -> 7 -> 8
```

Q4-Wildcard Matching

```
#include <iostream>
#include <vector> #include
<string> using namespace std; bool
isMatch(string s, string p) {
```

```
    int m = s.size(), n = p.size();

    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));
dp[0][0] = true;

    for (int j = 1; j <= n; ++j) {
if (p[j - 1] == '*') {
            dp[0][j] = dp[0][j - 1];
        }
    }

    for (int i = 1; i <= m; ++i) {
for (int j = 1; j <= n; ++j) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
dp[i][j] = dp[i - 1][j - 1];            } else if (p[j -
1] == '*') {                dp[i][j] = dp[i][j - 1]
|| dp[i - 1][j];
            }
        }
    }

    return dp[m][n];
}

int main() {
    string s, p;

    cout << "Enter the input string (s): ";
cin >> s;
    cout << "Enter the pajern (p): ";
cin >> p;

    if (isMatch(s, p)) {
        cout << "The string matches the pajern." << endl;
    } else {
        cout << "The string does not match the pajern." << endl;
    }
```

```
    return 0;
}
```

Output-

```
Enter the input string (s): abcdefgh
Enter the pattern (p): abcdefgh
The string matches the pattern.
```

Q5- Special Binary String

```
#include <iostream>
#include <string>
#include <vector> #include
<algorithm>  using
namespace std;

string makeLargestSpecial(string s) {
vector<string> substrings;     int count
= 0;
    int start = 0;

    for (int i = 0; i < s.size(); i++) {
count += (s[i] == '1' ? 1 : -1);         if
(count == 0) {              string substring = "1" +
makeLargestSpecial(s.substr(start + 1, i - start -
1)) + "0";
            substrings.push_back(substring);
            start = i + 1;
        }
    }

    sort(substrings.rbegin(), substrings.rend());

    string result;
    for (const string& sub : substrings) {
result += sub;
    }
```

```
    return result;
}

int main() {
    string s;
    cout << "Enter a special binary string: ";
cin >> s;

    string result = makeLargestSpecial(s);      cout << "The lexicographically
largest special binary string is: " << result <<
endl;

    return 0;
}
```

Output-

```
Enter a special binary string: 0101011
The lexicographically largest special binary string is: 101010
```