

Day 3

Function & Recursion DSA Questions

Name: HARLEEN

UID: 22BCS12125

Section: KPIT-901 B

Very Easy

Ques. Fibonacci Series Using Recursion

The Fibonacci numbers, commonly denoted $F(n)$ form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1.

That is,

$$F(0) = 0, F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$$

Given n , calculate $F(n)$.

Example 1:

Input: $n = 2$

Output: 1

Explanation: $F(2) = F(1) + F(0) = 1 + 0 = 1$

Program:

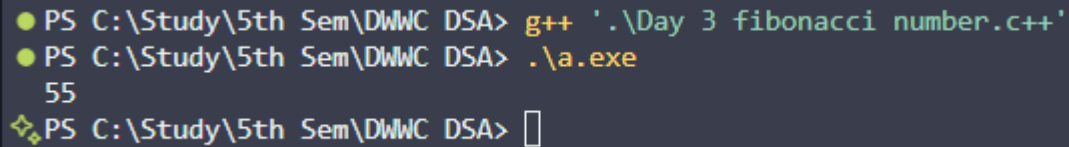
```
#include<bits/stdc++.h>

int fibo(int x)
{
    if(x<=1)
    {
        return x;
    }
    return fibo(x-1)+fibo(x-2);
}

int main()
{
```

```
int x=10;
std::cout<<fibo(x);
}
```

Output:



```
PS C:\Study\5th Sem\DWWC DSA> g++ '.\Day 3 fibonacci number.c++'
PS C:\Study\5th Sem\DWWC DSA> .\a.exe
55
PS C:\Study\5th Sem\DWWC DSA> 
```

Easy

Ques. Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:

Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Program:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* reverseListRecursive(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return head;
    ListNode* reversedList = reverseListRecursive(head->next);
    head->next->next = head;
    head->next = nullptr;
    return reversedList;
}
```

```

}

void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    ListNode* head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
    head->next->next->next->next = new ListNode(5);

    cout << "Original List: ";
    printList(head);

    head = reverseListRecursive(head);
    cout << "Reversed List: ";
    printList(head);

    return 0;
}

```

Output:

```

PS C:\Study\5th Sem\DWWC DSA> g++ '.\Day 3 Reverse linked list.c++'
PS C:\Study\5th Sem\DWWC DSA> .\a.exe
Original List: 1 2 3 4 5
Reversed List: 5 4 3 2 1
PS C:\Study\5th Sem\DWWC DSA> 

```

Medium

Ques. Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: $342 + 465 = 807$.

Program:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* addTwoNumbersRecursive(ListNode* l1, ListNode* l2, int carry = 0) {
    if (!l1 && !l2 && carry == 0) return nullptr;

    int sum = (l1 ? l1->val : 0) + (l2 ? l2->val : 0) + carry;
    ListNode* node = new ListNode(sum % 10);
    node->next = addTwoNumbersRecursive(l1 ? l1->next : nullptr, l2 ? l2->next : nullptr, sum / 10);
    return node;
}
```

```
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    ListNode* l1 = new ListNode(2);
    l1->next = new ListNode(4);
    l1->next->next = new ListNode(3);

    ListNode* l2 = new ListNode(5);
    l2->next = new ListNode(6);
    l2->next->next = new ListNode(4);

    cout << "List 1: ";
    printList(l1);
    cout << "List 2: ";
    printList(l2);

    ListNode* result = addTwoNumbersRecursive(l1, l2);
    cout << "Sum: ";
    printList(result);

    return 0;
}
```

Output:

```
● PS C:\Study\5th Sem\DWWC DSA> g++ '.\Day 3 Add Two Numbers.c++'
● PS C:\Study\5th Sem\DWWC DSA> .\a.exe
List 1: 2 4 3
List 2: 5 6 4
Sum: 7 0 8
❖ PS C:\Study\5th Sem\DWWC DSA> █
```

Hard

Ques. Wildcard Matching

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

'?' Matches any single character.

'*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".

Program:

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

bool isMatchRecursive(string s, string p, int i = 0, int j = 0) {
    if (j == p.size()) return i == s.size();
    if (p[j] == '*') {
        return isMatchRecursive(s, p, i, j + 1) || (i < s.size() && isMatchRecursive(s, p, i + 1, j));
    }
    return i < s.size() && (p[j] == '?' || p[j] == s[i]) && isMatchRecursive(s, p, i + 1, j + 1);
}

int main() {
```

```

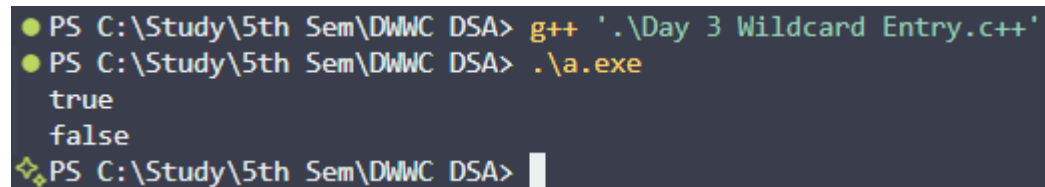
string s = "aa", p = "*";
cout << (isMatchRecursive(s, p) ? "true" : "false") << endl;

s = "cb", p = "?a";
cout << (isMatchRecursive(s, p) ? "true" : "false") << endl;

return 0;
}

```

Output:



```

PS C:\Study\5th Sem\DWWC DSA> g++ '.\Day 3 Wildcard Entry.cpp'
PS C:\Study\5th Sem\DWWC DSA> .\a.exe
true
false
PS C:\Study\5th Sem\DWWC DSA>

```

Very Hard

Ques. Special Binary String

Special binary strings are binary strings with the following two properties:

The number of 0's is equal to the number of 1's.

Every prefix of the binary string has at least as many 1's as 0's.

You are given a special binary string *s*.

A move consists of choosing two consecutive, non-empty, special substrings of *s*, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.

Example 1:

Input: *s* = "11011000"

Output: "11100100"

Explanation: The strings "10" [occurring at *s*[1]] and "1100" [at *s*[3]] are swapped.

This is the lexicographically largest string possible after some number of swaps.

Program:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric> // Include this for accumulate
using namespace std;

string makeLargestSpecialRecursive(string s) {
    vector<string> subs;
    int count = 0, start = 0;

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

    sort(subs.rbegin(), subs.rend());
    return accumulate(subs.begin(), subs.end(), string());
}

int main() {
    string s = "11011000";
    cout << makeLargestSpecialRecursive(s) << endl;

    s = "10";
    cout << makeLargestSpecialRecursive(s) << endl;

    return 0;
}
```



```
}
```

Output:

```
● PS C:\Study\5th Sem\DWMC DSA> g++ '.\Day 3 Special Binary String.c++'  
● PS C:\Study\5th Sem\DWMC DSA> .\a.exe  
11100100  
10  
❖ PS C:\Study\5th Sem\DWMC DSA> 
```