Name : Sahil

UID : 22BCS10928

Section : 901_kpit(B)

Q1 .**Fibonnacci Series Using Recursion**

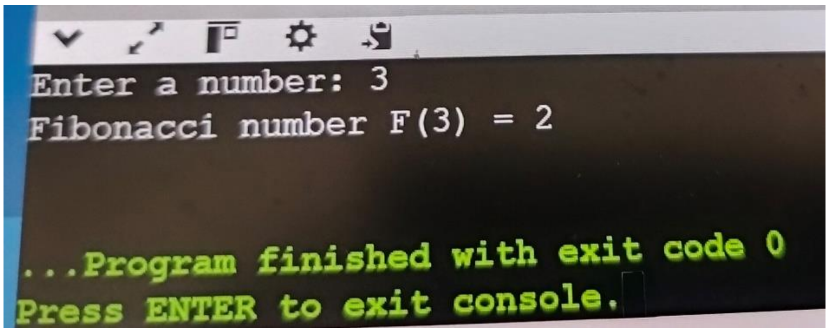
The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
Given n, calculate F(n).

Answer:

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

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    cout << "Fibonacci number F(" << n << ") = " << fibonacci(n) << endl;
    return 0;
}
```

Output:



```
Enter a number: 3
Fibonacci number F(3) = 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Q2) Add Two Numbers

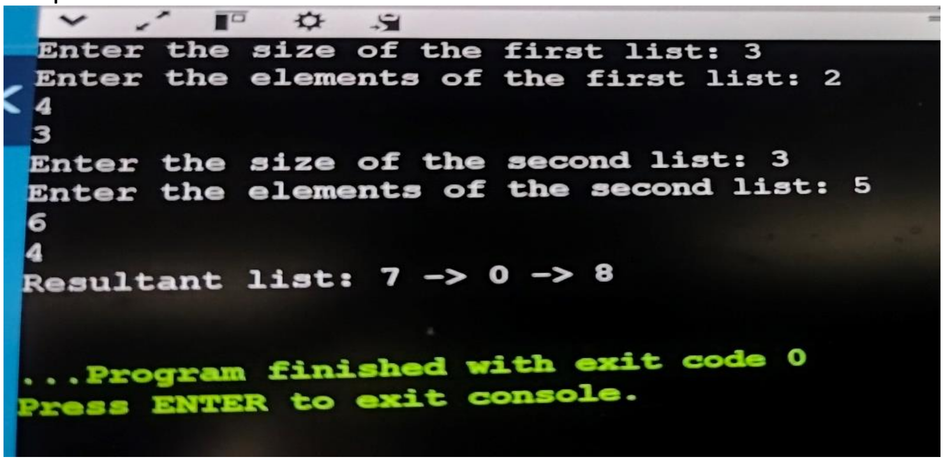
You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

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

Answer:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), 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 sum = carry;
        if (l1 != nullptr) {
            sum += l1->val;
            l1 = l1->next;
        }
        if (l2 != nullptr) {
            sum += l2->val;
            l2 = l2->next;
        }
        carry = sum / 10;
        current->next = new ListNode(sum % 10);
        current = current->next;
    }

    return dummyHead->next;
}

ListNode* createListFromInput(int size) {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    for (int i = 0; i < size; ++i) {
        int val;
        cin >> val;
        ListNode* newNode = new ListNode(val);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = tail->next;
        }
    }
    return head;
}
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val;
        if (head->next != nullptr) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    int size1, size2;
    cout << "Enter the size of the first list: ";
    cin >> size1;
    cout << "Enter the elements of the first list: ";
    ListNode* l1 = createListFromInput(size1);
    cout << "Enter the size of the second list: ";
    cin >> size2;
    cout << "Enter the elements of the second list: ";
    ListNode* l2 = createListFromInput(size2);
    ListNode* result = addTwoNumbers(l1, l2);
    cout << "Resultant list: ";
    printList(result);
    return 0;
}
```

Output:



```
Enter the size of the first list: 3
Enter the elements of the first list: 2
4
3
Enter the size of the second list: 3
Enter the elements of the second list: 5
6
4
Resultant list: 7 -> 0 -> 8


...Program finished with exit code 0
Press ENTER to exit console.
```

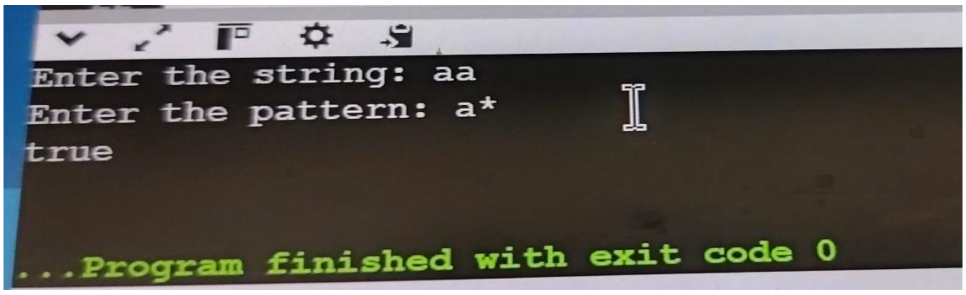
Q3) Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

'.' Matches any single character.
'*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).

Answer:
```
#include <iostream>
#include <string>
using namespace std;

bool isMatch(string s, string p) {
   int m = s.length(), n = p.length();
   bool dp[m + 1][n + 1];
   dp[0][0] = true;

   for (int i = 1; i <= m; ++i) dp[i][0] = false;
   for (int j = 1; j <= n; ++j) dp[0][j] = j > 1 && p[j - 1] == '*' && dp[0][j - 2];

   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 - 2] || (dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'));
         } else {
            dp[i][j] = false;
         }
      }
   }
   return dp[m][n];
}

int main() {
   string s, p;
   cout << "Enter the string: ";
   cin >> s;
   cout << "Enter the pattern: ";
   cin >> p;

   if (isMatch(s, p)) {
      cout << "true" << endl;
   } else {
      cout << "false" << endl;
   }

   return 0;
}
```

Output:



```
Enter the string: aa
Enter the pattern: a*
true

...Program finished with exit code 0
```

Q4) Basic Calculator

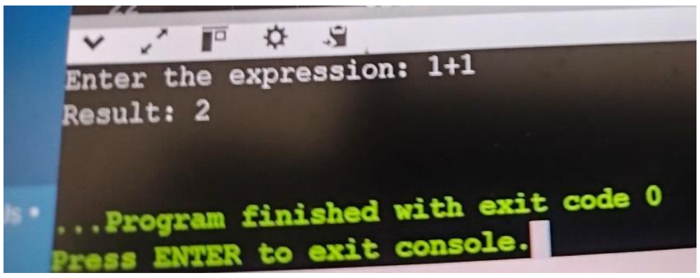
Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return the result of the evaluation.
Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

Answer:

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

int calculate(string s) {
   stack<int> operands, operators;
   int currentNumber = 0, result = 0, sign = 1;

   for (char c : s) {
      if (isdigit(c)) {
         currentNumber = currentNumber * 10 + (c - '0');
      } else if (c == '+') {
         result += sign * currentNumber;
         currentNumber = 0;
         sign = 1;
      } else if (c == '-') {
         result += sign * currentNumber;
         currentNumber = 0;
         sign = -1;
      } else if (c == '(') {
         operands.push(result);
```

```
            operators.push(sign);
            result = 0;
            sign = 1;
        } else if (c == ')') {
            result += sign * currentNumber;
            currentNumber = 0;
            result *= operators.top();
            operators.pop();
            result += operands.top();
            operands.pop();
        }
    }
    result += sign * currentNumber;
    return result;
}

int main() {
    string s;
    cout << "Enter the expression: ";
    getline(cin, s);

    cout << "Result: " << calculate(s) << endl;

    return 0;
}
```

Output:



```
Enter the expression: 1+1
Result: 2

...Program finished with exit code 0
Press ENTER to exit console.
```

Q5. Permutation Sequence

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.
By listing and labeling all of the permutations in order, we get the following sequence for n = 3:

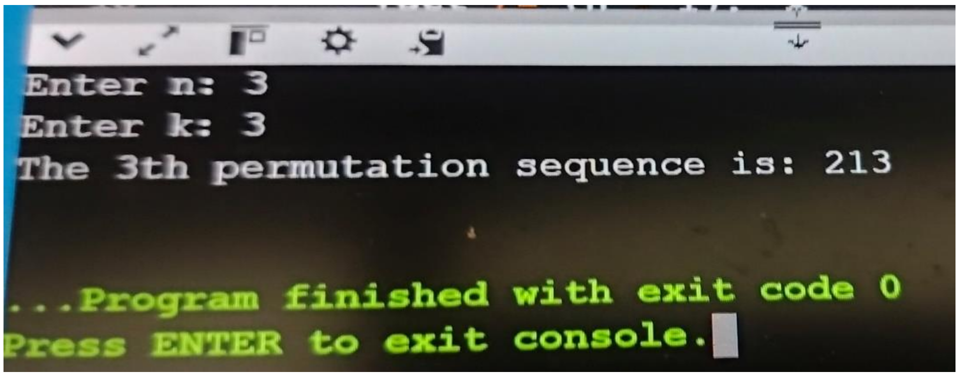
"123"
"132"
"213"
"231"

"312"
"321"
Given n and k, return the kth permutation sequence.

Answer:

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

string getPermutation(int n, int k) {
    vector<int> numbers;
    int fact = 1;
    for (int i = 1; i <= n; ++i) {
        numbers.push_back(i);
        fact *= i;
    }

    k--;
    string result;

    for (int i = 0; i < n; ++i) {
        fact /= (n - i);
        int index = k / fact;
        result += to_string(numbers[index]);
        numbers.erase(numbers.begin() + index);
        k %= fact;
    }

    return result;
}

int main() {
    int n, k;
    cout << "Enter n: ";
    cin >> n;
    cout << "Enter k: ";
    cin >> k;

    cout << "The " << k << "th permutation sequence is: " << getPermutation(n, k) << endl;

    return 0;
}
```

Output:



```
Enter n: 3
Enter k: 3
The 3th permutation sequence is: 213


...Program finished with exit code 0
Press ENTER to exit console.
```