Day 3

NAME - MEGHA SHREE

UID -22BCS10381

Function $ RECURSION DSA QUESTION

Very Easy

Q .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), 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

Example 2:

Input: n = 3

Output: 2

Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.

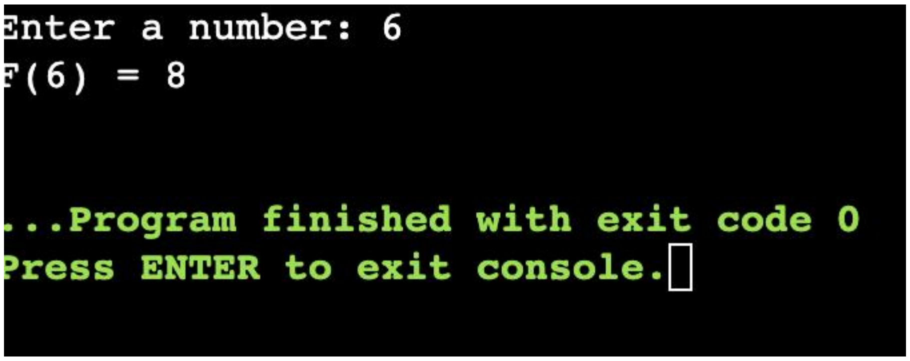
Constraints:

0 <= n <= 30

CODE-

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
  if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }
 return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
if (n < 0 || n > 30) {
  cout << "Input is out of range. Please enter a number between 0 and 30." << endl;
    } else {
        cout << "F(" << n << ") = " << fibonacci(n) << endl;
    }return 0;
}
```

OUTPUT:



```
Enter a number: 6
F(6) = 8


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

Easy

Q. 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]

Example 2:

Input: head = [1,2]

Output: [2,1]

Constraints:

The number of nodes in the list is the range [0, 5000].

-5000 <= Node.val <= 5000

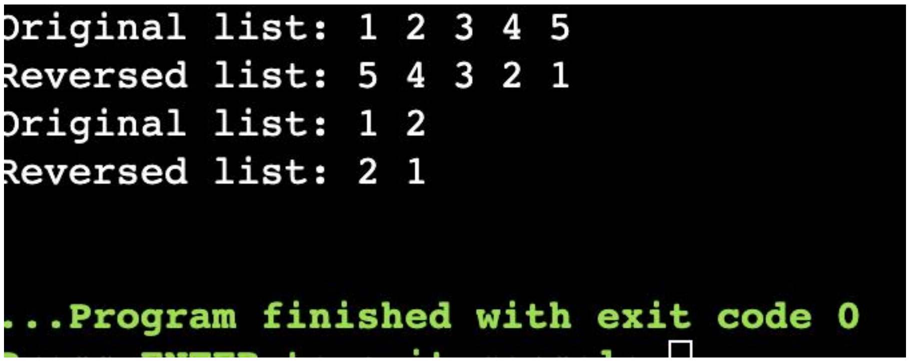
Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

CODE:

```
#include <iostream>

#include<vector>

using namespace std;

struct ListNode {

   int val;

   ListNode* next;

   ListNode() : val(0), next(nullptr) {}

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

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

};

ListNode* reverseList(ListNode* head) {

   ListNode* prev = nullptr;

   ListNode* curr = head;

  while (curr != nullptr)
ListNode* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
   }
   return prev;
}
void printList(ListNode* head) {
   ListNode* curr = head;
   while (curr != nullptr) {
        cout << curr->val << " ";
        curr = curr->next;
   }
   cout << endl;
}
ListNode* createList(const vector<int>& values) {
   if (values.empty()) return nullptr;
    ListNode* head = new ListNode(values[0]);
   ListNode* curr = head;
   for (int i = 1; i < values.size(); ++i) {
        curr->next = new ListNode(values[i]);
        curr = curr->next;
   }return head;}
```

```
   int main() {
vector<int> values1 = {1, 2, 3, 4, 5};
   ListNode* head1 = createList(values1);
   cout << "Original list: ";
   printList(head1);
   head1 = reverseList(head1);
   cout << "Reversed list: ";
   printList(head1);
  vector<int> values2 = {1, 2};
   ListNode* head2 = createList(values2);
   cout << "Original list: ";
   printList(head2);
   head2 = reverseList(head2);
   cout << "Reversed list: ";
   printList(head2);
   return 0;
}
```

OUTPUT:



```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
Original list: 1 2
Reversed list: 2 1


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

Medium

Q. 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.

Example 2:

Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

The number of nodes in each linked list is in the range [1, 100].

0 <= Node.val <= 9

It is guaranteed that the list represents a number that does not have leading zeros.

CODE

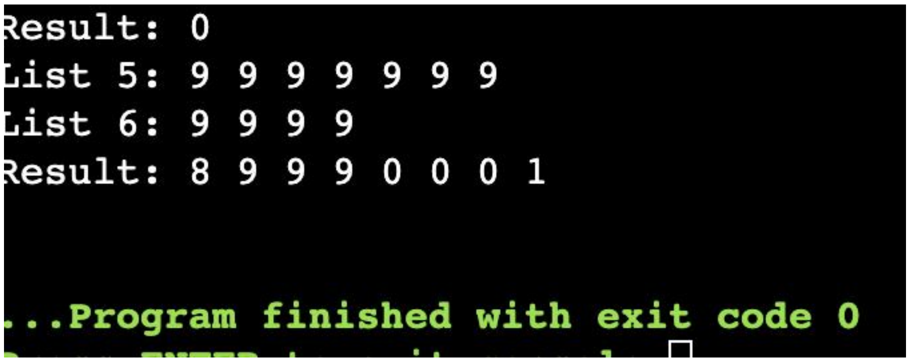
```
#include <iostream>

#include<vector>

using namespace std;

struct ListNode {

   int val;

   ListNode* next;

   ListNode() : val(0), next(nullptr) {}

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

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

};

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;
}
void printList(ListNode* head) {
 ListNode* current = head;
  while (current != nullptr) {
   cout << current->val << " ";
    current = current->next;
    }
    cout << endl;
}
ListNode* createList(const vector<int>& values) {
    if (values.empty()) return nullptr;
   ListNode* head = new ListNode(values[0]);
    ListNode* current = head;
    for (int i = 1; i < values.size(); ++i) {
        current->next = new ListNode(values[i]);
        current = current->next;
    }
    return head;}
```

```
int main() {

    vector<int> values1 = {2, 4, 3};

    vector<int> values2 = {5, 6, 4};

    ListNode* l1 = createList(values1);

    ListNode* l2 = createList(values2);

    cout << "List 1: ";

    printList(l1);

    cout << "List 2: ";

    printList(l2);

    ListNode* result = addTwoNumbers(l1, l2);

    cout << "Result: ";

    printList(result);

    vector<int> values3 = {0};

    vector<int> values4 = {0};

    ListNode* l3 = createList(values3);

    ListNode* l4 = createList(values4);

    cout << "List 3: ";

    printList(l3);

    cout << "List 4: ";

    printList(l4);

    result = addTwoNumbers(l3, l4);

    cout << "Result: ";

    printList(result);
```

```
 vector<int> values5 = {9, 9, 9, 9, 9, 9, 9};
    vector<int> values6 = {9, 9, 9, 9};
    ListNode* l5 = createList(values5);
    ListNode* l6 = createList(values6);
    cout << "List 5: ";
    printList(l5);
    cout << "List 6: ";
    printList(l6);
     result = addTwoNumbers(l5, l6);
    cout << "Result: ";
    printList(result);
 return 0;
}
```

OUTPUT



```
Result: 0
List 5: 9 9 9 9 9 9 9
List 6: 9 9 9 9
Result: 8 9 9 9 0 0 0 1


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

[Hard]

Q. 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".

Example 2:

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

Output: true

Explanation: '*' matches any sequence.

Example 3:

Input: s = "cb", p = "?a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

Constraints:

0 <= s.length, p.length <= 2000

s contains only lowercase English letters.

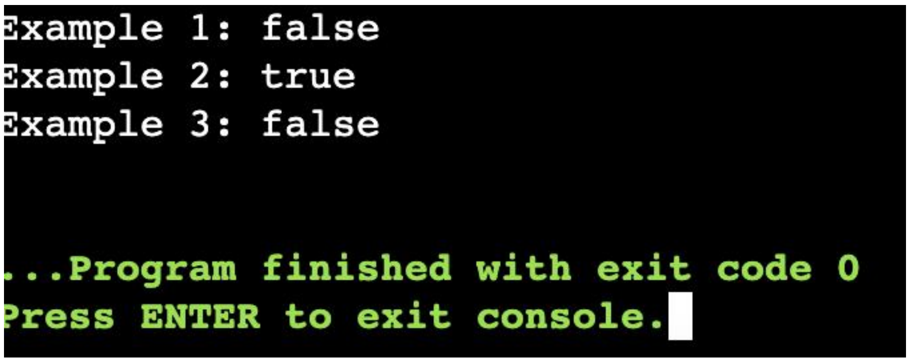
p contains only lowercase English letters, '?' or '*'.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;

bool isMatch(string s, string p) {

    int m = s.length();

    int n = p.length();

      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-1][j] || dp[i][j-1];
           }
        }
    }
       return dp[m][n];
}


int main() {
  cout << boolalpha;
      cout << "Example 1: " << isMatch("aa", "a") << endl;
    cout << "Example 2: " << isMatch("aa", "*") << endl;
    cout << "Example 3: " << isMatch("cb", "?a") << endl;
       return 0;
}
```

OUTPUT:



```
Example 1: false
Example 2: true
Example 3: false


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

Very Hard

Q. 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" [occuring at s[1]] and "1100" [at s[3]] are swapped.

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

Example 2:

Input: s = "10"

Output: "10"

Constraints:

1 <= s.length <= 50

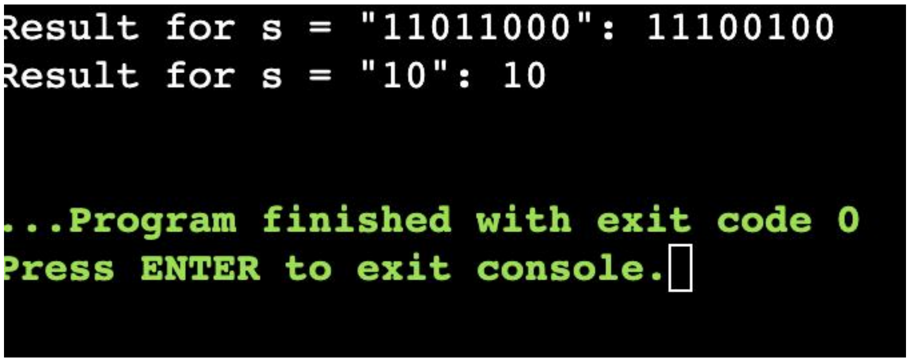
s[i] is either '0' or '1'.

s is a special binary string.

CODE:

```
#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

string makeLargestSpecial(string s) {

    if (s.size() <= 1) return s;

   vector<string> substrings;

    int balance = 0, start = 0;

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

        balance += (s[i] == '1') ? 1 : -1;

        if (balance == 0) {

            substrings.push_back("1" + makeLargestSpecial(s.substr(start + 1, i - start - 1)) +
"0");

            start = i + 1;

        }
    }

        sort(substrings.begin(), substrings.end(), greater<string>());

        string result;

    for (const string& sub : substrings) {

        result += sub;

    }

        return result;

}


int main() {

  string s1 = "11011000";

    cout << "Result for s = \"" << s1 << "\": " << makeLargestSpecial(s1) << endl;

      string s2 = "10";

    cout << "Result for s = \"" << s2 << "\": " << makeLargestSpecial(s2) << endl;

       return 0;

}
```

OUTPUT:



```
Result for s = "11011000": 11100100
Result for s = "10": 10


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

Function Questions

Very Easy

Q. Write a Function to print first name and last name using function

You are given the firstname and lastname of a person on two different lines. Your task is to read them and print the following using function:

Hello firstname lastname! You just delved into function.

Function Description:

Complete the print_full_name function in the editor below.

print_full_name has the following parameters:

string first: the first name

string last: the last name

Prints

string: 'Hello firstname lastname ! You just delved into using the function' where firstname and lastname are replaced with first and last.

Input Format

The first line contains the first name, and the second line contains the last name.

Constraints

The length of the first and last names are each ≤10 .

Sample Input 0

Ross

Taylor

Sample Output 0

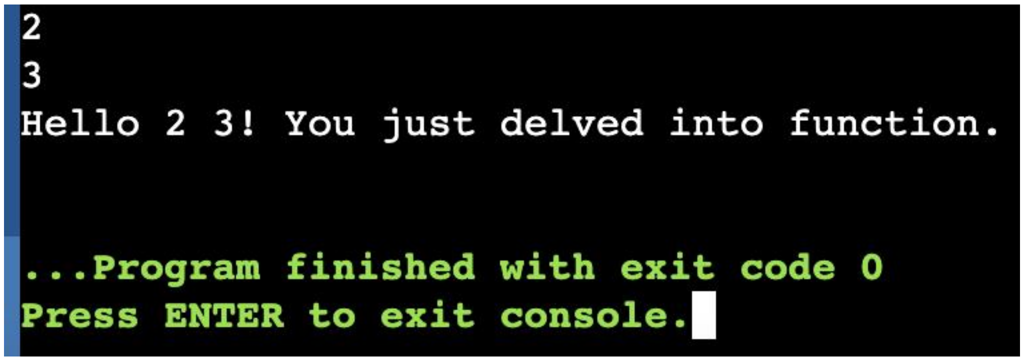
Hello Ross Taylor! You just delved into function

Explanation 0

The input read by the program is stored as a string data type. A string is a collection of characters.

```
#include <iostream>

#include <string>

using namespace std;

void print_full_name(string first, string last) {

    cout << "Hello " << first << " " << last << "! You just delved into function." << endl;

}

int main() {

    string first, last;

    cin >> first;

    cin >> last;

 print_full_name(first, last);

  return 0;

}
```

OUTPUT:



```
2
3
Hello 2 3! You just delved into function.


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

Easy

Q1 Find GCD of Number Using Function

Given an integer array nums, return the greatest common divisor of the smallest number and largest number in nums.

The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

Example 1:

Input: nums = [2,5,6,9,10]

Output: 2

Explanation:

The smallest number in nums is 2.

The largest number in nums is 10.

The greatest common divisor of 2 and 10 is 2.

Example 2:

Input: nums = [7,5,6,8,3]

Output: 1

Explanation:

The smallest number in nums is 3.

The largest number in nums is 8.

The greatest common divisor of 3 and 8 is 1.

Example 3:

Input: nums = [3,3]

Output: 3

Explanation:

The smallest number in nums is 3.

The largest number in nums is 3.

The greatest common divisor of 3 and 3 is 3.

Constraints:

2 <= nums.length <= 1000

1 <= nums[i] <= 1000

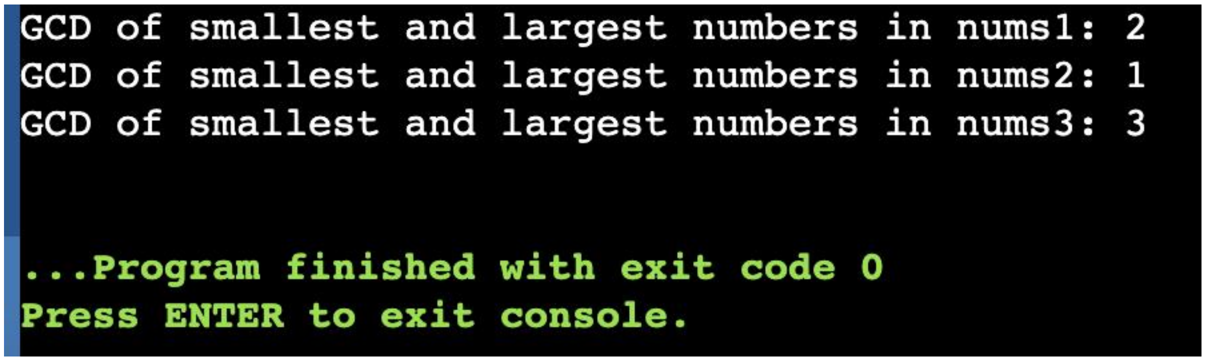
CODE:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;
```

```
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
int findGCD(vector<int>& nums) {
  int min_num = *min_element(nums.begin(), nums.end());
    int max_num = *max_element(nums.begin(), nums.end());
       return gcd(min_num, max_num);
}

int main() {
vector<int> nums1 = {2, 5, 6, 9, 10};
    cout << "GCD of smallest and largest numbers in nums1: " << findGCD(nums1) << endl;
   vector<int> nums2 = {7, 5, 6, 8, 3};
    cout << "GCD of smallest and largest numbers in nums2: " << findGCD(nums2) << endl;
  vector<int> nums3 = {3, 3};
    cout << "GCD of smallest and largest numbers in nums3: " << findGCD(nums3) << endl;
  return 0;
```

}

OUTPUT:



```
GCD of smallest and largest numbers in nums1: 2
GCD of smallest and largest numbers in nums2: 1
GCD of smallest and largest numbers in nums3: 3


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

Medium

Q2.: Longest Substring Without Repeating Characters

You are working on building a text editor application. One of the features you're focusing on is a "word suggestion" system. The editor needs to identify the longest sequence of characters typed without repeating any letters to suggest potential words or phrases. To accomplish this, you must efficiently find the length of the longest substring of unique characters as the user types.

Description:
Write a function that takes a string as input and returns the length of the longest substring without repeating characters. A substring is a contiguous sequence of characters within the string.

Example 1:

Input: "abcabcbb"

Output: 3

Explanation: The longest substring without repeating characters is "abc", which has length 3.

Constraints:

- The input string will have a length between 111 and 10410^4104.
- The characters in the string are printable ASCII characters.

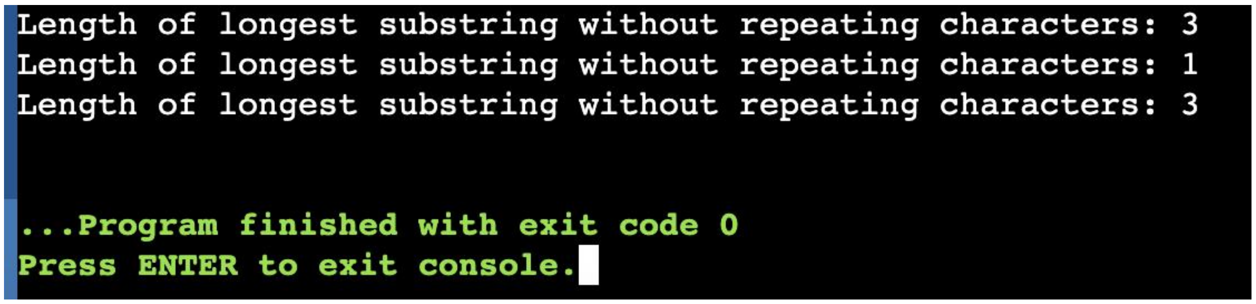
This problem is a classic example of the **sliding window** technique,

CODE:

```
include <iostream>

#include <unordered_set>

#include <string>

using namespace std;

int lengthOfLongestSubstring(string s) {

    unordered_set<char> char_set;

    int start = 0;

    int max_len = 0;

for (int end = 0; end < s.length(); ++end) {

while (char_set.find(s[end]) != char_set.end()) {

   char_set.erase(s[start]);

           ++start;

        }

 char_set.insert(s[end]);

    max_len = max(max_len, end - start + 1);

    }

    return max_len;

}
```

```
int main() {

    string s1 = "abcabcbb";

    cout << "Length of longest substring without repeating characters: " <<
lengthOfLongestSubstring(s1) << endl;

        string s2 = "bbbbb";

    cout << "Length of longest substring without repeating characters: " <<
lengthOfLongestSubstring(s2) << endl;

   string s3 = "pwwkew";

    cout << "Length of longest substring without repeating characters: " <<
lengthOfLongestSubstring(s3) << endl;

   return 0;

}
```

OUTPUT:



```
Length of longest substring without repeating characters: 3
Length of longest substring without repeating characters: 1
Length of longest substring without repeating characters: 3


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