Day- 3

Student Name: Rajan Mishra   UID: 22BCS13621

Branch: BE-CSE   Sec/Group:22BCS-IOT-615

1.Reverse An Array Using Function An array is a type of data structure that stores elements of the same type in a contiguous block of memory. In an array A , of size N, each memory location has some unique index i , (where 0<=i<=N), that can be referenced as or A[i] or Ai Reverse an array of integers. (Using Function) Example: A=[1,2,3] Return . [3,2,1] Function Description Complete the function reverseArray in the editor below. reverseArray has the following parameter(s): ● int A[n]: the array to reverse Returns ● int[n]: the reversed array Input Format The first line contains an integer N, , the number of integers in A. The second line contains N space-separated integers that make up A. Constraints: 1<=N<=10^3 1<=A[i]<=10^4

```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<int> reverseArray(vector<int>& A) {
    int left = 0, right = A.size() - 1;
while    (left    <    right)    {
swap(A[left],          A[right]);
left++;        right--;
}
return A;
}


int main() {    int N;    cin
>> N;    vector<int> A(N);
for (int i = 0; i < N; i++) {
cin >> A[i];
}
```

```
vector<int>reversed=reverseArray(A);
for (int i = 0; i < N; i++) {          cout
<< reversed[i] << " ";
   }
   cout << endl;
return 0;  }
```

Output

```
Enter size of array:5
Enter original array
1
2
3
4
5
Reversed array is:5 4 3 2 1
```

2. Balanced Brackets A bracket is defined as one of the following characters: (,), {, }, [, or]. Two brackets are considered to be a matched pair if an opening bracket (i.e., (, [, or {) occurs to the left of the corresponding closing bracket (i.e.,),], or }). There are three types of matched pairs of brackets: [], {}, and (). A matching pair of brackets is considered not balanced if the set of brackets it encloses is not matched properly. For example, {[(])} is not balanced because the contents in between { and } are not balanced: the pair of square brackets encloses an unmatched opening bracket (, and the pair of parentheses encloses an unmatched closing square bracket]. We define a sequence of brackets as balanced if the following conditions are met: 1. It contains no unmatched brackets. 2. The subset of brackets enclosed within a matched pair of brackets is also a matched pair of brackets. Task: Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return "YES". Otherwise, return "NO". Function Description: Complete the function isBalanced in the editor below. The function has the following parameter(s): ● string s: a string consisting of brackets. Returns: ● string: Either "YES" if the sequence is balanced, or "NO" if it is not balanced. Input Format: ● The first line contains a single integer n, the number of strings. ● Each of the next n lines contains a single string s, which is a sequence of brackets. Output Format: For each string, return "YES" if the string is balanced, otherwise return "NO". Example 1: Input: lua Copy code 3 {[()]} {[(])} {{[[(())]]}} Output: objectivec Copy code YES NO YES Explanation: ● The string {[()]} is balanced because all brackets are properly matched and nested. ● The string {[(])} is not balanced because the brackets enclosed by the matched pair { and } are not balanced: [(]). ● The string {{[[(())]]}} is balanced because all brackets are properly matched and nested.

Answer

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

string    isBalanced(string   s)   {
stack<char> stk;      for
(char c : s) {
if (c == '(' || c == '{' || c == '[') {
stk.push(c);
} else {
if     (stk.empty())  return  "NO";
char top = stk.top();           if ((c
== ')' && top == '(') || (c =='}' &&
top == '{') || (c == ']' && top ==
'['))
{            stk.pop();
} else {           return
"NO";
       }
     }
   }
   return stk.empty() ? "YES" : "NO";
}

int main() {
int n;
cin    >>    n;
while (n--) {       string s;
     cin >> s;        cout <<
isBalanced(s) << endl;
```

```
    }
return 0;
}
```

Output:

```
Enter size of stack:5
Enter paranthesis:{{()}}
YES
```

3.Power Of Three Given an integer n, return true if it is a power of three. Otherwise, return false. An integer n is a power of three, if there exists an integer x such that n == 3x. Example 1: Input: n = 27 Output: true Explanation: 27 = 33 Example 2: Input: n = 0 Output: false Explanation: There is no x where 3x = 0. Constraints: -231 <= n <= 231 - 1

Answer

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

bool isPowerOfThree(int n) {    if
(n <= 0) return false;
while (n % 3 == 0) {        n
/= 3;
    }
 return n == 1;
}

int main() {
int n;
cin >> n;    cout        <<
        (isPowerOfThree(n)   ?
"true""false") << endl;
return 0;
}
```

Output:

```
}

int main() {
int n, k;     cin
>> n >> k;
cout << getPermutation(n, k) << endl;
return 0;
}
```

Output:

```
Enter size:3
Enter kth Permutation:3
213
```

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".

Answer

```
#include <iostream> #include
<vector>
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 i = 1; i <= n; i++) {        if (p[i
- 1] == '*') dp[0][i] = dp[0][i - 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];
        }          if (p[j - 1] == '*') {
dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
        }
      }
}


    return dp[m][n];
}


int main() {     string s, p;    cin >>
s >> p;    cout << (isMatch(s, p) ?
"true" : "false") << endl;    return
0; }  Output:
```

```
Enter the string:aabbaba
Enter the pattern:ab
false
```

6.Palindrome Partitioning Given a string s, partition s such that every substring of the partition is a
Palindrome Return all possible palindrome partitioning of s. Example 1: Input: s = "aab" Output:
[["a","a","b"],["aa","b"]] Answer

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

bool isPalindrome(const string& s, int
```

```cpp
left, int right) {    while (left < right) {
if (s[left] != s[right]) return false;
left++;        right--;    }    return true;
}

void partitionHelper(const string& s, int
start, vector<string>& current,
vector<vector<string>>& result) {     if
     (start   ==   s.size())       {
result.push_back(current);
    return;
  }

  for (int end = start; end < s.size();
end++) {
     if (isPalindrome(s,  start,  end))  {
current.push_back(s.substr(start,  end  -
start + 1));        partitionHelper(s, end
+    1,    current,       result);
current.pop_back();
    }
  }
}

vector<vector<string>> partition(string
s) {    vector<vector<string>> result;
vector<string>        current;
partitionHelper(s, 0, current, result);
return result;
}

int main() {    string s;    cin >> s;
vector<vector<string>>   result   =
partition(s);       for (const auto&
```

part : result) {   for (const auto&
str : part) {   cout << str <<
" ";
      }   cout
<< endl;
}   return 0; } Output:

```
Enter the string:aab
a a b
aa b
```

7.Combination sum || Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target. Each number in candidates may only be used once in the combination. Note: The solution set must not contain duplicate combinations. Example 1: Input: candidates = [10,1,2,7,6,1,5], target = 8 Output: [[1,1,6], [1,2,5], [1,7], [2,6]]

Answer

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

combinationSum2Helper(vector<int>&
candidates,    int     target,  int
      start, vector<int>&      current,
vector<vector<int>>& result) {
   if (target == 0) {
result.push_back(current);      return;
   }

   for (int i = start; i < candidates.size();
i++) {
      if (i > start && candidates[i]  ==
candidates[i - 1]) continue;            if
(candidates[i] > target) break;
```

```cpp
        current.push_back(candidates[i]);

combinationSum2Helper(candidates,
target - candidates[i], i + 1, current,
result);
current.pop_back();
    }
}

vector<vector<int>>
combinationSum2(vector<int>&
candidates,    int    target) {
vector<vector<int>>  result;
vector<int>    current;
sort(candidates.begin(),
candidates.end());
combinationSum2Helper(candidates,
target, 0, current, result);    return result;
}

int main() {    vector<int> candidates
     =
{10,1,2,7,6,1,5};
    int        target      =       8;
vector<vector<int>>        result    =
combinationSum2(candidates, target);
for (const auto& comb : result) {
for (int num : comb) {         cout <<
num << " ";
    }       cout
<< endl;
}    return 0; }  Output:
```

```
Number of students:4
Total Sandwiches:4
Number of students who eat sandwiches:1
```

8.Number of Students Unable to Eat Sandwiches The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches. The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step: If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue. Otherwise, they will leave it and go to the queue's end. This continues until none of the queue students want to take the top sandwich and are thus unable to eat. You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat. Example 1: Input: students = [1,1,0,0], sandwiches = [0,1,0,1] Output: 0

Answer

```cpp
#include <iostream> #include
<queue>  using
namespace std;  int
        countStudents
(vector<int>&
students, vector<int>& sandwiches) {
queue<int> studentQueue;     for (int
student      :      students)      {
studentQueue.push(student);
  }

  int count = 0;    while (count <
students.size()) {        if
(studentQueue.front()        ==
sandwiches[0]) {
studentQueue.pop();

sandwiches.erase(sandwiches.begin());
      count = 0;
```

```
        } else {

studentQueue.push(studentQueue.front(
)
);
studentQueue.pop();
        count++;
      }
    }
    return studentQueue.size();
}


int main() {     vector<int> students =
{1,1,0,0};    vector<int> sandwiches =
{0,1,0,1};    cout        <<
        countStudents(students,
sandwiches) << endl;
    return 0; }
```

```
Number of students:4
Total Sandwiches:4
Number of students unable to eat sandwiches:1
```

9.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, nonempty, 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"
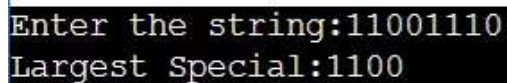
Answer

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

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

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

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

int    main() {   string s;
  cin     >> s;              cout    <<
makeLargestSpecial(s)        <<
      endl;   return 0;  }
```

```
Enter the string:11001110
Largest Special:1100
```