Sahil Sanjay Chalke

Test ID: 432003529686130 | 9022516901 | sahilchalke1011@gmail.com

Test Date: February 2, 2024

Logical Ability	English Comprehension	Quantitative Ability (Advanced)	Computer Science
56 /100	61 /100	53 /100	35 /100

Data Science	Automata	Automata Fix	Personality
33 /100	4 /100	15 /100	Completed

Logical Ability — 56 / 100

Inductive Reasoning	Deductive Reasoning	Abductive Reasoning
58 / 100	57 / 100	54 / 100

English Comprehension — 61 / 100

Grammar	Vocabulary	Comprehension
67 / 100	57 / 100	60 / 100

Quantitative Ability (Advanced) — 53 / 100

Basic Mathematics	Advanced Mathematics	Applied Mathematics
54 / 100	57 / 100	49 / 100

Computer Science — 35 / 100

OS and Computer Architecture	DBMS	Computer Networks
32 / 100	70 / 100	0 / 100

© 2024 SHL and/or its affiliates. All rights reserved..

Data Science

33 / 100

ML Algorithms and Implementation	Probability and Statistics	ML Experiments
20 / 100	50 / 100	33 / 100

Automata

4 / 100

Programming Ability	Programming Practices	Functional Correctness
10 / 100	0 / 100	10 / 100

*This can potentially be a non-serious attempt.

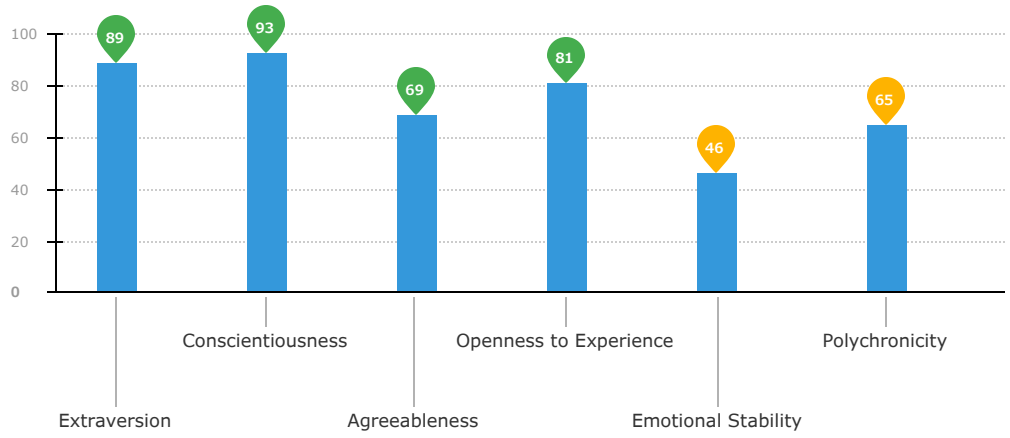
Automata Fix

15 / 100

Logical Error	Code Reuse	Syntactical Error
25 / 100	0 / 100	0 / 100

Personality

Completed



Competencies

People Interaction

Self-Drive

Trainability

Repetitive Job Suitability

Work Attributes

© 2024 SHL and/or its affiliates. All rights reserved..

1 | Introduction

About the Report

This report provides a detailed analysis of the candidate's performance on different assessments. The tests for this job role were decided based on job analysis, O*Net taxonomy mapping and/or criterion validity studies. The candidate’s responses to these tests help construct a profile that reflects her/his likely performance level and achievement potential in the job role

This report has the following sections:

The **Summary** section provides an overall snapshot of the candidate’s performance. It includes a graphical representation of the test scores and the subsection scores.

The **Insights** section provides detailed feedback on the candidate’s performance in each of the tests. The descriptive feedback includes the competency definitions, the topics covered in the test, and a note on the level of the candidate’s performance.

The **Response** section captures the response provided by the candidate. This section includes only those tests that require a subjective input from the candidate and are scored based on artificial intelligence and machine learning.

The **Learning Resources** section provides online and offline resources to improve the candidate's knowledge, abilities, and skills in the different areas on which s/he was evaluated.

Score Interpretation

All the test scores are on a scale of 0-100. All the tests except personality and behavioural evaluation provide absolute scores. The personality and behavioural tests provide a norm-referenced score and hence, are percentile scores. Throughout the report, the colour codes used are as follows:

- Scores between 67 and 100
- Scores between 33 and 67
- Scores between 0 and 33

© 2024 SHL and/or its affiliates. All rights reserved..

2 | Insights

English Comprehension

61 / 100

This test aims to measure your vocabulary, grammar and reading comprehension skills.

You have a good understanding of commonly used grammatical constructs. You are able to read and understand articles, reports and letters/mails related to your day-to-day work. The ability to read, understand and interpret business-related documents is essential in most jobs, especially the ones that involve research, technical reading and content writing.

Logical Ability

56 / 100

Inductive Reasoning

58 / 100

This competency aims to measure the your ability to synthesize information and derive conclusions.

You are able to work out rules based on specific information and solve general work problems using these rules. This skill is required in data-driven research jobs where one needs to formulate new rules based on variable trends.

Deductive Reasoning

57 / 100

This competency aims to measure the your ability to synthesize information and derive conclusions.

You are able to work out rules based on specific information and solve general work problems using these rules. This skill is required in data-driven research jobs where one needs to formulate new rules based on variable trends.

Abductive Reasoning

54 / 100

Quantitative Ability (Advanced)

53 / 100

This test aims to measure your ability to solve problems on basic arithmetic operations, probability, permutations and combinations, and other advanced concepts.

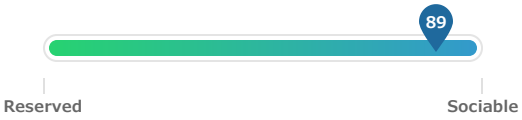
You are able to solve word problems on basic concepts of percentages, ratio, proportion, interest, time and work. Having a strong hold on these concepts can help you understand the concept of work efficiency and how interest is accrued on bank savings. It can also guide you in time management, work planning, and resource allocation in complex projects.

© 2024 SHL and/or its affiliates. All rights reserved..

Personality

Competencies

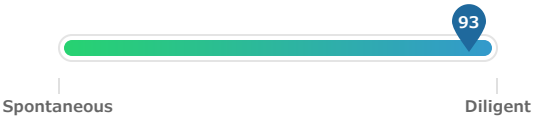
Extraversion



Extraversion refers to a person's inclination to prefer social interaction over spending time alone. Individuals with high levels of extraversion are perceived to be outgoing, warm and socially confident.

- You are outgoing and seek out opportunities to meet new people.
- You tend to enjoy social gatherings and feels comfortable amongst strangers and friends equally.
- You display high energy levels and like to indulge in thrilling and exciting activities.
- You may tend to be assertive about your opinions and prefer action over contemplation.
- You take initiative and are more inclined to take charge than to wait for others to lead the way.
- Your personality is well suited for jobs demanding frequent interaction with people.

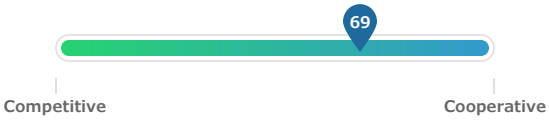
Conscientiousness



Conscientiousness is the tendency to be organized, hard working and responsible in one's approach to your work. Individuals with high levels of this personality trait are more likely to be ambitious and tend to be goal-oriented and focused.

- You value order and self discipline and tends to pursue ambitious endeavours.
- You believe in the importance of structure and is very well-organized.
- You carefully review facts before arriving at conclusions or making decisions based on them.
- You strictly adhere to rules and carefully consider the situation before making decisions.
- You tend to have a high level of self confidence and do not doubt your abilities.
- You generally set and work toward goals, try to exceed expectations and are likely to excel in most jobs, especially those which require careful or meticulous approach.

Agreeableness



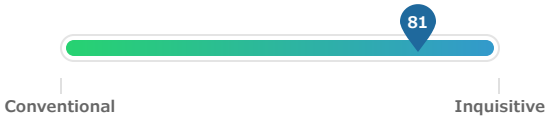
Agreeableness refers to an individual's tendency to be cooperative with others and it defines your approach to interpersonal relationships. People with high levels of this personality trait tend to be more considerate of people around them and are more likely to work effectively in a team.

- You are considerate and sensitive to the needs of others.
- You tend to put the needs of others ahead of your own.
- You are likely to trust others easily without doubting their intentions.
- You are compassionate and may be strongly affected by the plight of both friends and strangers.
- You are humble and modest and prefer not to talk about personal accomplishments.

© 2024 SHL and/or its affiliates. All rights reserved..

- Your personality is more suitable for jobs demanding cooperation among employees.

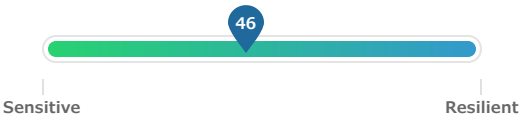
Openness to Experience



Openness to experience refers to a person's inclination to explore beyond conventional boundaries in different aspects of life. Individuals with high levels of this personality trait tend to be more curious, creative and innovative in nature.

- You tend to be curious in nature and is generally open to trying new things outside your comfort zone.
- You may have a different approach to solving conventional problems and tend to experiment with those solutions.
- You are creative and tends to appreciate different forms of art.
- You are likely to be in touch with your emotions and is quite expressive.
- Your personality is more suited for jobs requiring creativity and an innovative approach to problem solving.

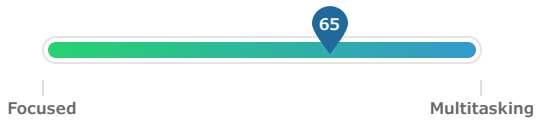
Emotional Stability



Emotional stability refers to the ability to withstand stress, handle adversity, and remain calm and composed when working through challenging situations. People with high levels of this personality trait tend to be more in control of their emotions and are likely to perform consistently despite difficult or unfavourable conditions.

- You are calm and relaxed in most situations.
- You experience a range of emotions in high pressure situations. You tend to worry when working in critical conditions.
- You do not like attention drawn towards you. You take some time to become confident and comfortable around people.
- You subdue your impulses and tend to act in a rational manner.
- Your personality is suited for jobs that have a moderate amount of stress.

Polychronicity



Polychronicity refers to a person's inclination to multitask. It is the extent to which the person prefers to engage in more than one task at a time and believes that such an approach is highly productive. While this trait describes the personality disposition of a person to multitask, it does not gauge their ability to do so successfully.

- You neither have a strong preference nor dislike to perform multiple tasks simultaneously.
- You are open to both options - pursuing multiple tasks at the same time or working on a single project at a time.
- Whether or not you will succeed in a polychronous environment depends largely on your ability to do so.

© 2024 SHL and/or its affiliates. All rights reserved..

© 2024 SHL and/or its affiliates. All rights reserved..

3 | Response

Automata

4 / 100 Code Replay

Question 1 (Language: Java)

A company has a sales record of N products for M days. The company wishes to know the maximum revenue received from a given product of the N products each day. Write an algorithm to find the highest revenue received each day.

Scores

Programming Ability

0 / 100

Programming ability score cannot be generated. This is because source code has syntax/ runtime errors and is unparseable.

Programming Practices

0 / 100

Programming practices score cannot be generated. This is because source code has syntax/runtime errors and is unparseable or the source code does not meet the minimum code-length specifications.

Functional Correctness

0 / 100

Syntactically incorrect code. The source code has syntax errors in it.

Final Code Submitted — **Compilation Status: Fail**

```
1  import java.util.*;
2  import java.lang.*;
3  import java.io.*;
4
5
6  /*
7   *
8   */
9  public class Solution
10 {
11
12    public static void main(String[] args)
13    {
14       BufferedReader br= new BufferReader(new InputStreamReader
   (System.in));
15
16       String[] str=br.readLine().split(" ");
17       int m =Integer.parseInt(str[0]);
         int n =Integer.parseInt(str[1]);
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code: O(N logN)

*N represents days/products in the sales record.

Errors/Warnings

```
Compiling failed with exitcode 1, compiler output:
Solution.java:14: error: cannot find symbol
BufferedReader br= new BufferReader(new
InputStreamReader(System.in));
^
symbol: class BufferReader
```

© 2024 SHL and/or its affiliates. All rights reserved..

```
18
19
20      for(int i =0;i<m;i++){
21      str=br.readLine().split(" ");
22      int max = Integer.parseInt(str[0]);
23      for(int j=1;j<n;j++){
24        if(max<Integer.parseInt(str[j])){
25          max=Integer.parseInt(str[j]);
26        }
27      }
28
29      System.out.println(max);
30
31      }
32    }
33 }
34
```

location: class Solution
1 error

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

3	1	2	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:31:35**
Average time taken between two compile attempts:	**00:10:32**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

© 2024 SHL and/or its affiliates. All rights reserved..

Question 2 (Language: C)

Charlie has a magic mirror that shows the right-rotated versions of a given word. To generate different right rotations of a word, the word is written in a circle in a clockwise order and read it starting from any given character in a clockwise order until all the characters are covered. For example, in the word "*sample*", if we start with '*p*', we get the right rotated word as "*plesam*".

Write an algorithm to output 1 if the *word1* is a right rotation of *word2* otherwise output -1.

Scores

Programming Ability

20 / 100

Code seems to be unrelated to the given problem.

Programming Practices

0 / 100

Programming practices score cannot be generated. This is because source code has syntax/runtime errors and is unparseable or the source code does not meet the minimum code-length specifications.

Functional Correctness

20 / 100

The source code does not pass any basic test cases. It is either due to incorrect logic or runtime errors. Some advanced or edge cases may randomly pass.

Final Code Submitted

Compilation Status: Pass

```
1  //Header Files
2  #include<stdio.h>
3  #include<stdlib.h>
4  #include<string.h>
5  #include<stdbool.h>
6
7  /* only used in string related operations */
8  typedef struct String string;
9  struct String
10 {
11    char *str;
12 };
13
14 char *input(FILE *fp, int size, int has_space)
15 {
16    int actual_size = 0;
17    char *str = (char *)malloc(sizeof(char)*(size+actual_size));
18    char ch;
19    if(has_space == 1)
20    {
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code: O(N)

*N represents maximum of length of both the input strings

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

© 2024 SHL and/or its affiliates. All rights reserved..

```
21      while(EOF != (ch=fgetc(fp)) && ch != '\n')
22      {
23        str[actual_size] = ch;
24        actual_size++;
25        if(actual_size >= size)
26        {
27          str = realloc(str,sizeof(char)*actual_size);
28        }
29      }
30    }
31    else
32    {
33      while(EOF != (ch=fgetc(fp)) && ch != '\n' && ch != ' ')
34      {
35        str[actual_size] = ch;
36        actual_size++;
37        if(actual_size >= size)
38        {
39          str = realloc(str,sizeof(char)*actual_size);
40        }
41      }
42    }
43    actual_size++;
44    str = realloc(str,sizeof(char)*actual_size);
45    str[actual_size-1] = '\0';
46    return str;
47  }
48  /* only used in string related operations */
49
50
51  /*
52   * word1, represents the first word.
53  word2, represents the second word.
54   */
55  int  isSameReflection(string word1, string word2)
56  {
57    int  answer;
58    // Write your code here
59
60
61
62    return answer;
63  }
64
65  int main()
66  {
67    string word1;
68    string word2;
69
70
```

© 2024 SHL and/or its affiliates. All rights reserved..

```
71  //input for word1
72  word1.str = input(stdin, 100, 1);
73
74
75  //input for word2
76  word2.str = input(stdin, 100, 1);
77
78
79  int result = isSameReflection(word1, word2);
80  printf("%d", result);
81
82  return 0;
83 }
84
```

Test Case Execution

Passed TC: **0%**

Total score

0/18

0%	0%	0%
Basic(**0**/8)	Advance(**0**/8)	Edge(**0**/2)

Compilation Statistics

0	0	0	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:06:05**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**0%**

© 2024 SHL and/or its affiliates. All rights reserved..

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Automata Fix

15 / 100 Code Replay

Question 1 (Language: C++)

The function/method ***arrayReverse*** modify the input list by reversing its element
The function/method ***arrayReverse*** accepts two arguments - *len,* an integer representing the length of the list and *arr,* list of integers representing the input list, respectively.

For example, if the input list *arr* is {20 30 10 40 50}, the function/method is supposed to print {50 40 10 30 20}.

The function/method ***arrayReverse*** compiles successfully but fails to get the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Scores

Final Code Submitted **Compilation Status: Pass**

```
1  // You can print the values to stdout for debugging
2  void arrayReverse(int len, int* arr)
3  {
4     int i, temp, originalLen=len;
5     for(i=0;i<originalLen/2;i++)
6     {
7        temp = arr[len-1];
8        arr[len-1] = arr[i];
         arr[i] = temp;
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

© 2024 SHL and/or its affiliates. All rights reserved..

```
 9
10      len -= 1;
11    }
12 }
```

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **100%**

Total score

8/8

100%	100%	0%
Basic(**6**/6)	Advance(**2**/2)	Edge(**0**/0)

Compilation Statistics

4	4	0	3	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:05:06**
Average time taken between two compile attempts:	**00:01:17**
Average test case pass percentage per compile:	**31.3%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 2 (Language: C++)

© 2024 SHL and/or its affiliates. All rights reserved..

The function/method **countDigits** return an integer representing the remainder when the given number is divided by the number of digits in it.
The function/method**countDigits** accepts an argument - *num,* an integer representing the given number.

The function/method **countDigits** compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Scores

Final Code Submitted — Compilation Status: Pass

```
1  // You can print the values to stdout for debugging
2  using namespace std;
3  int countDigits(int num)
4  {
5    int count =0;
6    while(num!=0){
7      num=num/10;
8      count++;
9    }
10   return (num%count);
11 }
12
13
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution — Passed TC: **37.5%**

Total score 3/8

33%	50%	0%
Basic(**2**/6)	Advance(**1**/2)	Edge(**0**/0)

Compilation Statistics

7	7	0	7	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:05:30**
Average time taken between two compile attempts:	**00:00:47**
Average test case pass percentage per compile:	**8.9%**

© 2024 SHL and/or its affiliates. All rights reserved..

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 3 (Language: C++)

The function/method ***removeElement*** prints space separated integers that remains after removing the integer at the given index from the input list.

The function/method ***removeElement*** accepts three arguments - *size*, an integer representing the size of the input list, *indexValue*, an integer representing given index and *inputList*, a list of integers representing the input list.

The function/method ***removeElement*** compiles successfully but fails to print the desired result for some test cases due to incorrect implementation of the function/method ***removeElement***. Your task is to fix the code so that it passes all the test cases.

Note:
Zero-based indexing is followed to access list elements.

Scores

Final Code Submitted **Compilation Status: Pass**

```
1 // You can print the values to stdout for debugging
2 using namespace std;
3 void removeElement(int size, int indexValue, int *inputList)
4 {
5   int i,j;
6   if(indexValue<size)
7   {
8     for(i=indexValue;i<size-1;i++)
9     {
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

© 2024 SHL and/or its affiliates. All rights reserved..

```
10        inputList[i]=inputList[i++];
11      }
12      for(i=0;i<size-1;i++)
13        cout<<inputList[i]<<" ";
14    }
15    else
16    {
17      for(i=0;i<size;i++)
18        cout<<inputList[i]<<" ";
19    }
20  }
21
```

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **62.5%**

Total score

5/8

40%	100%	100%
Basic(**2**/5)	Advance(**2**/2)	Edge(**1**/1)

Compilation Statistics

1	1	0	1	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:01:32**
Average time taken between two compile attempts:	**00:01:32**
Average test case pass percentage per compile:	**12.5%**

© 2024 SHL and/or its affiliates. All rights reserved..

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 4 (Language: C++)

The function/method ***findMaxElement*** return an integer representing the largest element in the given two input lists. The function/method ***findMaxElement*** accepts four arguments - *len1,* an integer representing the length of the first list, *arr1*, a list of integers representing the first input list, *len2*, an integer representing the length of the second input list and *arr2*, a list of integers representing the second input list, respectively.

Another function/method ***sortArray*** accepts two arguments - *len,* an integer representing the length of the list and *arr*, a list of integers, respectively and return a list sorted ascending order.

Your task is to use the function/method ***sortArray*** to complete the code in ***findMaxElement*** so that it passes all the test cases.

Scores

Final Code Submitted **Compilation Status: Pass**

```
1  // You can print the values to stdout for debugging
2  using namespace std;
3  int* sortArray(int len, int* arr)
4  {
5    int i=0,j=0,temp=0;
6    for(i=0;i<len;i++)
7    {
8     for(j=i+1;j<len;j++)
9     {
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

© 2024 SHL and/or its affiliates. All rights reserved..

```
10      if(arr[i]>arr[j])
11      {
12        temp = arr[i];
13        arr[i] = arr[j];
14        arr[j] = temp;
15      }
16     }
17   }
18   return arr;
19 }
20
21 int findMaxElement(int len1, int* arr1, int len2, int* arr2)
22 {
23  int max=0;
24  for(int i=0;i<sizeof(arr1);i++) {
25   if(arr1[i] > max){
26    max=arr1[i];
27   }
28  }
29
30  for(int i=0;i<sizeof(arr2);i++) {
31   if(arr2[i] > max){
32    max=arr2[i];
33   }
34  }
35 return  max;
36
37 }
38
```

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution

Passed TC: **37.5%**

Total score

3/8

50%	33%	0%
Basic(**1**/2)	Advance(**2**/6)	Edge(**0**/0)

Compilation Statistics

5	2	3	2	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:06:32**
Average time taken between two compile attempts:	**00:01:18**
Average test case pass percentage per compile:	**2.5%**

© 2024 SHL and/or its affiliates. All rights reserved..

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 5 (Language: C++)

You are given predefined structure ***Time*** containing *hour*, *minute*, and *second* as members. A collection of functions/methods for performing some common operations on times is also available. You must make use of these functions/methods to calculate and return the difference.

The function/method ***difference_in_times*** accepts two arguments - *time1*, and *time2*, representing two times and is supposed to return an integer representing the difference in the number of seconds.

You must complete the code so that it passes all the test cases.

.

Helper Description

The following class is used to represent the time and is already implemented in the default code (Do not write this definition again in your code):

```
class Time

{

  int hour;

  int minute;

  int second;

  int Time :: Time_compareTo( Time* time2)

  {
```

© 2024 SHL and/or its affiliates. All rights reserved..

```
    /*Return 1, if time1 is greater than time2.
    Return -1 if time1 is less than time2
    or, Return 0, if time1 is equal to time2
    This can be called as -
    *  If time1 and time2 are two Time then -
    *  time1.compareTo(time2) */
}
void Time :: Time_addSecond()
{
    /* Add one second in the time;
    This can be called as -
    *  If time1 is Time then -
    *  time1.addSecond() */
}
```

Scores

Final Code Submitted — Compilation Status: Fail

```
1 // You can print the values to stdout for debugging
2 using namespace std;
3 int difference_in_times(Time *time1, Time *time2)
4 {
5    // write your code here
6 }
7
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

Errors/Warnings

```
In file included from main_24.cpp:8:
source_24.cpp: In function 'int
difference_in_times(Time*, Time*)':
source_24.cpp:6:1: error: no return statement in
function returning non-void [-Werror=return-type]
}
```

© 2024 SHL and/or its affiliates. All rights reserved..

^
cc1plus: some warnings being treated as errors

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Compilation Statistics

0	0	0	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:00:05**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 6 (Language: C++)

The function/method ***countElement*** returns the number of elements in the input list arr which are greater than twice the input number K. The function/method ***countElement*** accepts three arguments - *size*, an integer representing the size of the input list, *numK*, an integer representing the input number K and *inputList*, a list of integers.

The function/method compiles unsuccessfully due to syntactical error. Your task is to fix the code so that it passes all the test cases.

© 2024 SHL and/or its affiliates. All rights reserved..

Scores

Final Code Submitted — Compilation Status: Fail

```
1  // You can print the values to stdout for debugging
2  using namespace std;
3  int countElement(int size, int numK, int *inputList)
4  {
5    int i,count=0;
6    for(i=0,i<size,i++)
7    {
8      if(inputList[i]>numK)
9        count+=1;
10   }
11   return count;
12 }
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

Errors/Warnings

```
In file included from main_30.cpp:7:
source_30.cpp: In function 'int countElement(int, int, int*)':
source_30.cpp:6:23: error: expected ';' before ')' token
for(i=0,i ^
;
source_30.cpp:11:5: error: expected primary-expression before 'return'
return count;
^~~~~~
source_30.cpp:10:6: error: expected ';' before 'return'
}
^
;
return count;
~~~~~~~
source_30.cpp:11:5: error: expected primary-expression before 'return'
return count;
^~~~~~
source_30.cpp:10:6: error: expected ')' before 'return'
}
^
)
return count;
~~~~~~~
source_30.cpp:6:8: note: to match this '('
for(i=0,i ^
```

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

© 2024 SHL and/or its affiliates. All rights reserved..

Compilation Statistics

1	0	1	0	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:00:58**
Average time taken between two compile attempts:	**00:00:58**
Average test case pass percentage per compile:	**0%**

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

Question 7 (Language: C++)

The function/method **printCharacterPattern** accepts an integer *num*. It is supposed to print the first *num* ($0 \leq num \leq 26$) lines of the pattern as shown below.

For example, if *num* = 4, the pattern is:
a
ab
abc
abcd

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

© 2024 SHL and/or its affiliates. All rights reserved..

Scores

Final Code Submitted — Compilation Status: Pass

```
1  // You can print the values to stdout for debugging
2  using namespace std;
3  void printCharacterPattern(int num){
4    int i, j;
5    char ch='a';
6    char print;
7    for(i=0;i<num;i++){
8      print = ch;
9      for(j=0;j<=i;j++)
10       cout<<(ch++);
11     cout<<"\n";
12   }
13 }
14
```

Code Analysis

Average-case Time Complexity

Candidate code: Complexity is reported only when the code is correct and it passes all the basic and advanced test cases.

Best case code:

*N represents

Errors/Warnings

There are no errors in the candidate's code.

Structural Vulnerabilites and Errors

There are no errors in the candidate's code.

Test Case Execution — Passed TC: **25%**

Total score 2/8

33%	0%	100%
Basic(**1**/3)	Advance(**0**/4)	Edge(**1**/1)

Compilation Statistics

1	1	0	1	0	0
Total attempts	Successful	Compilation errors	Sample failed	Timed out	Runtime errors

Response time:	**00:00:00**
Average time taken between two compile attempts:	**00:00:00**
Average test case pass percentage per compile:	**12.5%**

© 2024 SHL and/or its affiliates. All rights reserved..

Average-case Time Complexity

Average Case Time Complexity is the order of performance of the algorithm given a random set of inputs. This complexity is measured here using the Big-O asymptotic notation. This is the complexity detected by empirically fitting a curve to the run-time for different input sizes to the given code. It has been benchmarked across problems.

Test Case Execution

There are three types of test-cases for every coding problem:

Basic: The basic test-cases demonstrate the primary logic of the problem. They include the most common and obvious cases that an average candidate would consider while coding. They do not include those cases that need extra checks to be placed in the logic.

Advanced: The advanced test-cases contain pathological input conditions that would attempt to break the codes which have incorrect/semi-correct implementations of the correct logic or incorrect/semi-correct formulation of the logic.

Edge: The edge test-cases specifically confirm whether the code runs successfully even under extreme conditions of the domain of inputs and that all possible cases are covered by the code

© 2024 SHL and/or its affiliates. All rights reserved..

4 | Learning Resources

English Comprehension

- Improve your hold on the language by reading Shakespearan plays
- Learn about how to get better at reading
- Read opinions to improve your comprehension

Logical Ability

- Practice your Inductive Reasoning Skills!
- Learn about generalizing unknown trends
- Test your application of inductive logic!

Quantitative Ability (Advanced)

- Watch a video on the history of algebra and its applications
- Learn about proportions and its practical usage
- Learn about calculating percentages manually

Icon Index

Free Tutorial | Paid Tutorial | Youtube Video | Web Source

Wikipedia | Text Tutorial | Video Tutorial | Google Playstore

© 2024 SHL and/or its affiliates. All rights reserved..