

Contestant Number: _____

Time: _____

Rank: _____



C++ PROGRAMMING

(335)

REGIONAL 2023

APPLICATION KNOWLEDGE:

Credit Card Verification _____ (350 points)

TOTAL POINTS _____ (***350 points***)

Test Time: 90 minutes

GENERAL GUIDELINES:

Failure to adhere to any of the following rules will result in disqualification:

1. Contestant must hand in this test booklet and all printouts if any. Failure to do so will result in disqualification.
2. No equipment, supplies, or materials other than those specified for this event are allowed in the testing area. No previous BPA tests and/or sample tests (handwritten, photocopied, or keyed) are allowed in the testing area.
3. Electronic devices will be monitored according to ACT standards.

Credit Card Verification

All credit card readers need to verify that a credit card is valid before an attempt is made to pull money from the bank account. In this exercise, you will create a C++ console application that performs credit card verification using the Luhn Algorithm.

The process to verify credit card numbers first involves verifying that the value is between 13 and 16 digits – note that this will require 64bits when stored as an integer value. Additionally, proper prefix must be present. The list of valid prefixes are as follows: “4”, “5”, “37”, and “6”, and will indicate what company the card is from.

The Luhn algorithm is a Mod 10 based verification process, that involves grouping each digit based on whether it is in an even or odd position in the number, starting from right and going left. Digits in an even position are doubled (if this results in a 2-digit number, add the digits together such that it is a single-digit value again) prior to adding them together, and odd position are simply added together.

The final step of the check is to add both sums together, and then verify that the result is divisible by 10 – if it is, then the card number is valid.

Sample output showing various valid credit card numbers and application flow.

```
Enter a Credit Card number to check validity for (-1 to exit application): test
Invalid input – only enter numerical values that can be contained in a 64bit signed value (max
9223372036854775807).
Enter a Credit Card number to check validity for (-1 to exit application): o
Invalid input – only enter numerical values that can be contained in a 64bit signed value (max
9223372036854775807).
Enter a Credit Card number to check validity for (-1 to exit application): 0
0 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 4670009360252821
4670009360252821 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 374319177519123
374319177519123 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 5329858139046159
53298581390461 is valid
Enter a Credit Card number to check validity for (-1 to exit application): 6670009760252821
6670009760252821 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): 6670009960252821
6670009960252821 is invalid
Enter a Credit Card number to check validity for (-1 to exit application): -1
Closing application.
```

Requirements:

1. You must create a C++ console application named CPP_335_ContestantNumber, where ContestantNumber is your BPA assigned contestant number (including dashes). For example, CPP_335_01_2345_6789.
2. Your contestant number must appear as a comment at the top of the main source code file.
3. When application starts, the user shall be prompted to input a credit card number to check validity for
 - a. Entering '-1' shall cause the application to be closed
 - b. After validity is reported, the user shall be prompted for an additional number to check
 - c. The user's input shall also be validated to ensure that it is not text input
4. Application shall properly employ the described credit card verification method using the Luhn algorithm
 - a. Verify card number is between 13 and 16 digits long
 - b. Starting right to left, sum all even position digits (double them first) and sum all odd position digits
 - c. Sum results from step b (odd summation and doubled even summation)
 - d. Verify result from step c is divisible by 10 (valid if so)
 - e. Report validity as shown
5. The application shall make use of functions, specifically a function for checking validity, verifying prefix matches, and for performing the summation functionality for both odd and even digits.

Your application will be graded on the following criteria:

Solution and Project

The project is present on the flash drive	_____ 10 pts
The project is named according to the naming conventions	_____ 10 pts

Program Execution

Code copied to USB drive and the program runs from USB	_____ 10 pts
--	--------------

If the program does not execute, then the remaining items in this section receive a score of zero.

Application displays error message if invalid text input	_____ 20 pts
Application displays error message if invalid numerical input	_____ 20 pts
Application successfully reports validity for test values	_____ 50 pts
Application properly loops to allow additional inputs to be entered	_____ 30 pts
Application pauses once execution is complete	_____ 20 pts

Source Code Review

Code is commented at the top, for each function, and as needed	_____ 20 pts
Code uses reasonable and consistent variable naming conventions	_____ 20 pts
Data types are handled in a logical and consistent manner	_____ 30 pts
Application user input logic is constructed properly	_____ 30 pts
The specified functions are present and are constructed in a logical manner	_____ 40 pts
Algorithm is constructed in manner described	_____ 40 pts

Total Points: 350 pts