# Problem 1 test plan

**Problem 1:**

Write a program to calculate the number of days between two dates. Each date is represented by three integers: day, month, and year. Validate the user input and if it is invalid, ask the user to enter again. If user enters three times an invalid day or month, announce that the program will exit and exit the program. Use the following rules for determining leap years: A leap year is one whose number is exactly divisible by four. Century years, however, are only leap years if they are exactly divisible by 400. Hence 1900 was not a leap year but 2000 was. Your program should not use arrays or vectors.

**Structure chart:Diagram

Description automatically generated**

**Checkpoints:**

1. The program should prompt the user to enter the first date and validate the input until it is valid.
2. The program should prompt the user to enter the second date and validate the input until it is valid.
3. The program should calculate the number of days between the two dates using the given rules for leap years.
4. The program should output the number of days between the two dates.

**Test Data:**

|  |  |  |
| --- | --- | --- |
| First/Second Date | Expected Result (manually) | Actual Result |
| 01 01 2022 / 31 12 2022 | 364 | 364 |
| 15 02 2022 / 01 03 2022 | 14 | 14 |
| 13 12 2004 / 13 03 2023 | 6664 | 6664 |
| 29 02 2020 / 01 03 2024 | 1461 | 1462 |

Brief Analysis: This program does not have any known errors since it has passed through multiple testing data examinations, which made me conclude that it’s well programmed.

# Problem 2 Test Plan

**Problem 2:**

The game of Nim. Two players alternately take marbles from a pile. In each move, a player chooses how many marbles to take. The player must take at least one, but at most half of the marbles. Then the other player takes a turn. The player who takes the last marble loses.

Write a program in which the computer plays against a human opponent. The computer can play in two modes: smart or stupid. In stupid mode, the computer simply takes a random legal value (between 1 and half of the marbles) from the pile whenever it has a turn. In smart mode the computer takes off enough marbles to make the size of the pile a power of two minus 1 (that is 3, 7, 15, 63, etc.). That is always a legal move, except if the size of the pile is currently one less than a power of 2. In that case, the computer makes a random legal move. (Note that the computer cannot be beaten in smart mode when it has the first move unless the pile size happens to be a power of 2 minus 1. Of course, a human player who has the first turn and knows the winning strategy can win against the computer.)

Minimum requirements: ask the user to enter the initial size of the pile between 10 and 100, to decide who has the first turn, and to decide whether the computer plays in stupid or smart mode. Then play the game accordingly.

**Structure chart:**

**Diagram

Description automatically generated**

**Checkpoints:**

1. The program should prompt the user to enter the initial size of the pile between 10 and 100.
2. The program should prompt the user to choose who has the first turn (human player or computer player).
3. The program should prompt the user to choose the mode for the computer player (smart or stupid).
4. The program should simulate the game of Nim with the chosen parameters until a player wins.
5. The program should print the winner of the game.

Test Data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial size of pile | First Turn | Computer Mode | Expected Result | Actual Result |
| 10 | Human | Smart | Human Wins | Human Wins |
| 25 | Computer | Stupid | Computer Wins | Human Wins |
| 50 | Human | Smart | Computer Wins | Computer Wins |
| 75 | Computer | Stupid | Human Wins | Human Wins |

Brief Analysis: In the end of the program still runs the loop (when there is 1 marble left), other than that the program structure and algorithm is correctly implemented.

# Problem 3 Test Plan

**Problem 3:**

Credit card number check. The last digit of a credit card number is the check digit, which protects again transcription errors such as an error in a single digit or switching two digits. The following method is used to verify actual credit card numbers, but for simplicity we will describe it for 8 digits instead of 16:

• Starting from the rightmost digit, calculate the sum of every other digit. For example, if the credit card number is 43589795, then you form the sum 5 + 7 + 8 + 3 = 23.

• Double all the digits that were not included in the preceding step. For example, with the number give above you double the digits 9, 9, 5, and 4, which yield 18, 18, 10, and 8. Adding all the digits in these numbers gives 1+8+1+8+1+ 0+8 = 27.

• Add the sums of the two preceding steps. If the last digit of the result is 0, the number is valid. In our example, 23 + 27 = 50, so the number is valid.

Write a program that implements this algorithm. Do not use arrays or vectors. Do not use a string representation of numbers. The user should supply a number (not individual digits, but a single number with any number of digits that you can store in a variable of type long), and you should print out whether the number is valid or not. Suggested extension: also print the value of the check digit that would make the number valid.

**Structure chart:Diagram

Description automatically generated**

**Checkpoints:**

1. The program should prompt the user to enter a credit card number.
2. The program should read in the credit card number from the user.
3. The program should calculate the sum of every other digit starting from the rightmost digit.
4. The program should double all the digits that were not included in the preceding step and add up the digits in the resulting numbers.
5. The program should add the sums of the two preceding steps.
6. If the last digit of the total sum is 0, the program should print that the credit card number is valid.
7. If the last digit of the total sum is not 0, the program should print that the credit card number is not valid.
8. The program should calculate the check digit that would make the credit card number valid.
9. The program should print the check digit that would make the credit card number valid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Credit Card Number | Expected Validity | Expected check digit | Actual Validity | Check digit |
| 43589795 | Valid | 0 | Valid | 0 |
| 12345678 | Invalid | 4 | Invalid | 4 |
| 79927398713 | Valid | 3 | Valid | 3 |
| 9999999999999999 | Invalid | 8 | Invalid | 8 |

Brief Analysis: This program does not have any known errors since I have tried multiple credit card number combinations and the output is always accurate, therefore it is well programmed.