**Problem Done By Hand:**

1. From 15th October, 1999 to 25th October, 1999

Days from 15th to 25th October:

25 - 15 = 10

Total days:

10

2. From 23rd October, 1999 to 1st December, 1999

Days remaining in October:

31 - 23 = 8

Days in November:

30

Days in December up to 1st:

1

Total days:

8 + 30 + 1 = 39

3. From 21st October, 1999 to 4th March, 2004

Days remaining in October:

31 - 21 = 10

Days in November:

30

Days in December:

31

Total for 1999:

10 + 30 + 31 = 71

Full years from 2000 to 2003:

2000: 366 (leap year)

2001: 365

2002: 365

2003: 365

Total for these four years:

366 + 365 + 365 + 365 = 1461

Days in 2004 up to 4th March:

January: 31

February: 29 (leap year)

March: 4

Total for 2004:

31 + 29 + 4 = 64

Overall total days:

71 + 1461 + 64 = 1596

**Approach:**

The problem needing to be solved by this algorithm is a rather complex one. The best way for me to tackle the issue of manually creating a program to calculate the number of days between two dates (excluding the start day) that does not use datetime modules or any other python library is to separate the algorithm into different functions. I will have one that determines whether or not the year is a leap year, one that calculates the number of days in each month, and one that brings it all together and calculates the number of days between the dates. The logic for number of days total in a year will be within the days\_between\_dates function.

**Psuedocode Attempt #1:** (it is assumed that the function is\_leap\_year() already exists)

days\_in\_month(month, year)

IF month IN [4, 6, 9, 11]

RETURN 30

ELSE IF month == 2

RETURN 29 IF is\_leap\_year(year) ELSE 28

ELSE

RETURN 31

days\_between\_dates(start\_date, end\_date)

SET start\_day, start\_month, start\_year <- start\_date

SET end\_day, end\_month, end\_year <- end\_date

SET days <- 0

IF start\_year == end\_year AND start\_month == end\_month

SET days <- end\_day – start\_day

RETURN days

ELSE IF start\_year == end\_year

SET days <- days + (days\_in\_month(start\_month, start\_year) - start\_day)

FOR month FROM start\_month + 1 TO end\_month - 1

SET days <- days + days\_in\_month(month, start\_year)

SET days <- days + end\_day

RETURN days

ELSE

SET days <- days + (days\_in\_month(start\_month, start\_year) - start\_day)

FOR month FROM start\_month + 1 TO 12

SET days <- days + days\_in\_month(month, start\_year)

FOR year FROM start\_year + 1 TO end\_year – 1

IF is\_leap\_year(year)

SET days <- days + 366

ELSE

SET days <- days + 365

FOR month FROM 1 TO end\_month - 1

SET days <- days + days\_in\_month(month, end\_year)

SET days <- days + end\_day

RETURN days

**Microsoft Copilot Pseudocode:**

function is\_leap\_year(year):

if (year % 4 == 0):

if (year % 100 != 0) or (year % 400 == 0):

return True

return False

function days\_in\_month(month, year):

days\_per\_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

if month == 2 and is\_leap\_year(year):

return 29

return days\_per\_month[month]

function parse\_date(date\_string):

# Assuming date\_string is in "YYYY-MM-DD" format

year, month, day = map(int, date\_string.split('-'))

return year, month, day

function days\_between\_dates(date1, date2):

year1, month1, day1 = parse\_date(date1)

year2, month2, day2 = parse\_date(date2)

total\_days = 0

for year in range(year1, year2 + 1):

if year == year1:

total\_days += days\_in\_month(month1, year) - day1

elif year == year2:

total\_days += day2

else:

total\_days += 365 if not is\_leap\_year(year) else 366

return total\_days

# Example usage:

date1 = "2024-06-15"

date2 = "2024-07-10"

print(f"Days between {date1} and {date2}: {days\_between\_dates(date1, date2)}")

**Compare and Contrast:**

1. Honestly, I think my code came out better this time, although maybe that is because I am not fully understanding what the Copilot code is doing, but the logic looks flawed. They are both equally modularized, has a clear separation of tasks between the functions, and is very readable. Mine is better in that I do all of the direct calculations and I do not use any built in functions or libraries. I also included better logic for handling different potential cases accurately. I do think that the Copilot code is better in that some of it's logic is handled on one line rather than 4 and it is a bit shorter.
2. I could make my solution better by maybe seeing if I could further simplify the logic for my calculations and using better variable names. After further study, the ways that copilot got the number of days in a month was more simple and efficient than my solution.
3. The Copilot solution could be improved by doing some more of the direct calculations and not using any built in libraries or functions. Its logic is also flawed for calculating number of days across different years and could use my logic instead.
4. Yes, both pseudocode’s match the algorithm performed by hand at the beginning of the lab.

**Final Pseudocode:** (it is assumed that the function is\_leap\_year() already exists)

days\_in\_month(month, year)

SET days\_per\_month <- [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

IF month == 2 AND is\_leap\_year(year)

RETURN 29

RETURN days\_per\_month[month]

days\_between\_dates(start\_date, end\_date)

SET start\_day, start\_month, start\_year <- start\_date

SET end\_day, end\_month, end\_year <- end\_date

SET days <- 0

IF start\_year == end\_year AND start\_month == end\_month

SET days <- end\_day – start\_day

RETURN days

ELSE IF start\_year == end\_year

SET days <- days + (days\_in\_month(start\_month, start\_year) - start\_day)

FOR month FROM start\_month + 1 TO end\_month - 1

SET days <- days + days\_in\_month(month, start\_year)

SET days <- days + end\_day

RETURN days

ELSE

SET days <- days + (days\_in\_month(start\_month, start\_year) - start\_day)

FOR month FROM start\_month + 1 TO 12

SET days <- days + days\_in\_month(month, start\_year)

FOR year FROM start\_year + 1 TO end\_year – 1

IF is\_leap\_year(year)

SET days <- days + 366

ELSE

SET days <- days + 365

FOR month FROM 1 TO end\_month - 1

SET days <- days + days\_in\_month(month, end\_year)

SET days <- days + end\_day

RETURN days

main()

SET date1 <- (15, 10, 1999)

SET date2 <- (25, 10, 1999)

SET date3 <- (21, 10, 1999)

SET date4 <- (4, 3, 2004)

SET result1 <- days\_between\_dates(date1, date2)

SET result2 <- days\_between\_dates(date3, date4)

PUT "Days between {date1} and {date2}: {result1}"

PUT "Days between {date3} and {date4}: {result2}"

main()

**Program Trace:**

A table with numbers and a number on it

Description automatically generated

**Algorithmic Efficiency:**

Starting with the days\_in\_month() function, the function takes the month as an index and uses it to look up the number of days in that month from a list. If the month is 2, it checks if it is a leap year and then returns 29 if it is. All of this will take constant time no matter the input size, so it is O(1). The days\_between\_dates() function can have varying time complexity. If the start and end years and months are the same, it computes the difference in days directly, which would be an O(1) operation. However, the worst case scenarios are that the start and end years/months are different, in which case the number of iterations through the loops would be proportional to the difference in years and months, which would make the efficiency O(n). Going off worst case scenarios, the overall efficiency could be represented by O(1) x O(n), resulting in an overall efficiency of O(n).