CS507 Computing Foundation for Computational Science HW4

Shell Programming

Min Long

Instruction

Due time: Due: 10/28/2022, 23:59
Submission command: `submit minlong CS507 HW4`

Instruction to electronic submission: http://cs.boisestate.edu/~cs221/SubmissionProcedure.html

- The written assignment can be done in a pure text format (*.txt, for example, problem1.txt) on Onyx.
- The programming assignment should be presented with source codes.
- Each problem should have its own working directory, such as HW2/prob1, HW2/prob2 ... For example, the following table shows the structure of HW1 from the user "student1" and how to submit HW to us through Onyx

```
[student1@onyx:HW1]$ ls -l
drwxr-x---. 2 student1 Students  13 Sep 19  2021 prob1
drwxr-xr-x. 3 student1 Students  14 Sep 19  2021 prob2
[student1@onyx:HW1]$ cd prob1/
[minlong@onyx:prob1]$ ls
-rw-r-----. 1 student1 Students 943 Sep 19  2021 problem1.txt
$ cd ..
[student1@onyx:HW1]$ pwd
/home/student1/CS507/HW1
[student1@onyx:HW1]$ submit minlong CS507 HW1
```

Listing 1: A sample structure of homework and submission procedure.

- Your source codes (if any) must compile and run on Onyx.
- Documentation is important and proper comments are expected in your source code.
  - comments giving description of: purpose, parameters, and return value if applicable
  - other comments where clarification of source code is needed
  - proper and consistent indentation
  - proper structure and modularity

Don't ask us or your classmates directly for solutions (it happened); just try as much as possible. Be patient and enjoy coding!

Programming Problems

The grading will be based on the simplicity and quality of the code. The simpler, shorter, well-designed code will get the higher score.

1. (10 pts) **Angle between Stars**. Given two stars with angles of declination and right ascension (d_1, a_1) and (d_2, a_2), respectively, the angle they subtend is given by the formula

$$2 \arcsin \left\{ [\sin^2(d/2) + \cos(d_1)\cos(d_2)\sin^2(a/2)]^{1/2} \right\},$$

where a_1 and a_2 are angles between -180 and 180 degrees, d_1 and d_2 are angles between -90 and 90 degrees, $a = a_2 - a_1$, and $d = d_2 - d_1$. Compose a program to take the declination and right ascension of two stars as command-line arguments and write the angle they subtend. Select the coordinates of the following two bright stars. **Hint**: Be careful about converting from degrees to radians.

Table 1: Coordinates of Altair and Vega stars

Star's name	Right Ascension (RA)	Declination (DEC)
Altair	19h 50m 46.99855s	+08° 52' 05.9563"
Vega	18h 36m 56.33635s	+38° 47' 01.2802"

Here the right ascension is represented using "hour angles", so you need to convert it to degrees first. Basically, 3600 s = 60 m = 1 h = 15°. For example, 23h=345°=-15°.

Extended reading. Any units of angular measure could have been chosen for right ascension, but it is customarily measured in hours (h), minutes (m), and seconds (s), with 24h being equivalent to a full circle (360 °). Astronomers have chosen this unit to measure right ascension because they measure a star's location by timing its passage through the highest point in the sky as the Earth rotates. The line which passes through the highest point in the sky, called the meridian, is the projection of a longitude line onto the celestial sphere.

Since a complete circle contains 24h of right ascension or 360° (degrees of arc), 1/24 of a circle is measured as 1h of right ascension, or 15°; 1/1440 of a circle is measured as 1m of right ascension, or 15 minutes of arc (also written as 15'); and 1/86400 of a circle contains 1s of right ascension, or 15 seconds of arc (also written as 15"). A full circle, measured in right-ascension units, contains $24 \times 60 \times 60 = 86400$s or $24 \times 60 = 1440$m, or 24h.

2. (10 pts) **Matrix multiplication.** Compose a function multiply() that takes two square matrices of the same dimension and returns their product. Make sure your output is formatted.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \cdot \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = ?$$

Extended Reading. If you are not familiar with the matrix multiplication, check your math book for details. We briefly summarize it below. If you have 2 matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, their product

$\boldsymbol{C}$ is represented as,

$$\begin{aligned}
&\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B} \\
&\boldsymbol{A} = [a_{ij}] \quad i, j = 1, 2, ..., n \\
&\boldsymbol{B} = [b_{ij}] \\
&\boldsymbol{C} = [c_{ij}]
\end{aligned}$$

The matrix forms are,

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Each term in $\boldsymbol{C}$ is computed as,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

3. (10pts) **Polynomials using array**. Compose a program `polynomial.py` with a function `evaluate(x, a)` that evaluates the polynomial $a(x)$ whose coefficients are the elements in the array $a[]$:

$$a_0 + a_1 x + a_2 x^2 + ... + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

An efficient way to perform the computations that is suggested by the following parenthesization:

$$a_0 + x(a_1 + x(a_2 + ... + x(a_{n-2} + x a_{n-1})...))$$

Then compose a function exp() that calls evaluate() to compute an approximation to e^x, using the first n terms of the Taylor series expansion $e^x = 1 + x + x^2/2! + x^3/3! + ...$ Take an argument x (e.g., try e^1 =?) from the command line, and compare your result against that computed by `math.exp(x)`.

4. (10pts) **Day of the week and Calendar**. Compose a program that accepts a date as input and writes the day of the week that date falls on. Your program should accept three command-line arguments: m (month), d (day), and y (year). For m, use 1 for January, 2 for February, and so forth. For output, write 0 for Sunday, 1 for Monday, 2 for Tuesday, and so forth. Use the following formulas for the Gregorian calendar:

$$\begin{aligned}
y_0 &= y - (14 - m)//12 \\
x &= y_0 + y_0//4 - y_0//100 + y_0//400 \\
m_0 &= m + 12 \times ((14 - m)//12) - 2 \\
d_0 &= (d + x + (31 \times m_0)//12)\%7
\end{aligned}$$

where "//" denotes floored division operator and "%" denotes the remainder operator. Example: On what day of the week was February 14, 2000?

$$\begin{aligned}
y_0 &= 2000 - (14 - 2)//12 = 1999 \\
x &= 1999 + 1999//4 - 1999//100 + 1999//400 = 1999 + 499 - 19 + 4 = 2483 \\
m_0 &= 2 + 12 \times ((14 - 2)//12) - 2 = 2 + 12 \times 1 - 2 = 12 \\
d_0 &= (14 + 2483 + (31 \times 12)//12)\%7 = (14 + 2483 + 31)\%7 = 1(\text{Monday})
\end{aligned}$$

```
$python dateofweek.py 2 14 2000
1
```

Listing 2: results of dayofweek.py.

Based on this program and the leapyear.py listed below,

```
import sys

# Accept an int year as a command-line argument. Write True to
# standard output if year is a leap year.  Otherwise write False.

year = int(sys.argv[1])

isLeapYear = (year % 4 == 0)
isLeapYear = isLeapYear and (year % 100 != 0)
isLeapYear = isLeapYear or  (year % 400 == 0)

print(sys.argv[1]+' is a leap year? ', isLeapYear)
```

Listing 3: leapyear.py.

compose a program cal.py that takes two command-line arguments m and y and writes the monthly calendar for the mth month of year y, as in this example:

```
    October 2022
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Listing 4: results of cal.py.

5. (10 pts) **Binary representation using recursion.** Compose a program that takes a positive integer n (in decimal) from the command line and writes its binary representation. Recall that in binary.py we used the method of subtracting out powers of 2.

```
import sys
n = int(sys.argv[1])

# Compute v as the largest power of 2 <= n.
v = 1
while v <= n//2:
    v *= 2

# Cast out powers of 2 in decreasing order.
while v > 0:
    if n < v:
        print(0, end="")
    else:
        print(1, end="")
        n -= v
    v //= 2
print()
```

Listing 5: binary.py.

```
$ python integertobinary.py 8
1000
```

Listing 6: results of binary.py.

Instead, use the following simpler method: repeatedly divide 2 into n and read the remainders backward. First, compose a while loop to carry out this computation and write the bits in the wrong order. Then, use recursion to write the bits in the correct order.

6. (10 pts) **Using Object's methods to process string.** We will discuss this problem on Monday's class. There are three strings, a='now is ', b='the time ', c='to'. Use built-in `str` class and its instance method to print out the following results in a **formatted** style.

```
$ python string-processing.py
a                    : "now is "
b                    : "the time "
c                    : "to"
len(a)               : 7
a[4]                 : "i"
a[2:5]               : "w i"
c.upper()            : "TO"
b.startswith("the")  : True
a.find("is")         : 4
a+c                  : "now is to"
b.replace("t", "T")  : "The Time "
a.split()            : ['now', 'is ']
b==c                 : False
a.strip()            : "now is"
```

Listing 7: Sample output.

Think: can you do it in less than 10 lines of code? If you want to make your code short, you may need to use Python's built-in functions isinstance(), eval().

Overall, the grading will be based on the simplicity and quality of the code. The simpler, shorter, well-designed code will get a higher score.