CS1010E: Programming Methodology

Tutorial 04: Repetition

13 Feb 2017 - 17 Feb 2017

1. Discussion Questions

(a) [Efficiency] What is the final value of `time` in the code fragment below? Can you give them in terms of n?

i.
```
int n = 10, i, ans = 0;
for(i=0; i<n; i++) ans++;
```

i. ____________________

ii.
```
int n = 10, i, ans = 0;
for(i=1; i<=n; i++) ans++;
```

ii. ____________________

iii.
```
int n = 10, i, j, ans = 0;
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    ans++;
```

iii. ____________________

iv.
```
int n = 10, i, j, ans = 0;
for(i=0; i<n; i++)
  for(j=i; j<n; j++)
    ans++;
```

iv. ____________________

v.
```
int n = 16, i, ans = 0;
for(i=n; i>=0; i/=2)
  ans++;
```

v. ____________________

vi.
```
int n = 10, i, j, ans = 0;
for(i=0, j=0; i<n; i++, j++)
  ans += j;
```

vi. ____________________

2. Program Analysis

(a) [Bad Practice] What is/are the output of *badly written* code fragments below?

i.
```
int n = 10, i, j, ans = 0;
for(i=0; i<n; i++)
  ans++;
  for(j=0; j<n; j++)
    ans++;
printf("%d", ans);
```

i. ____________________

ii.
```
int n = 10, ans = 0;
while(n-->0) ans++;
printf("%d", ans);
```

ii. ____________________

(b) [Complex Loop Reasoning] What is/are the output of code fragments below?

i.
```
int c = 1, ans = 100;
while(c != 0) {
  if(ans > 100) {
    ans -= 10; c--;
  } else {
    ans += 11; c++;
  }
} printf("%d", ans);
```

i. ____________________

ii.
```
int ans = 9999, g = 75, m = 65537;
while(ans)
  ans = g * ans % m;
printf("%d", ans);
```

ii. ____________________

(c) [Abstraction] What is/are the output of the code fragment below for `n = 5` and `n = 1000`?

i.
```
int i, j, ans = 0;
for(i=0; i<n; i++)
  for(j=0; j<2*n-1; j++)
    if(i >= n-1-j && i >= j+1-n && i <= n-1) ans++;
printf("%d", ans);
```

i. ____________________

ii.
```
int i, j, ans = 0;
for(i=1; i<=n; i++)
  for(j=1; j<=n; j++)
    if(i*j%2) ans++;
printf("%d", ans);
```

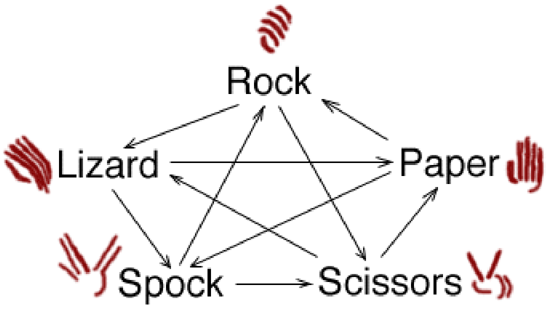
ii. ____________________

iii.
```
int i, j, ans = 0;
for(i=2; i<=n; i++) {
  for(j=2; j<i; j++)
    if(i%j == 0) break;
  if(j==i) ans++;
} printf("%d", ans);
```

iii. ____________________

3. Designing a Solution

(a) [Computation; Standard Input] The game of Rock Paper Scissors Spock Lizard (RPSSL) is an attempt at improving Rock Paper Scissors (RPS) from Sam Kass and Karen Bryla by reducing the probability of a draw. RPSSL is played by two people, each choosing one shape of hand according to Diagram 1. Unlike normal RPS, our game is played for an *undetermined* number of rounds.



Scissors cuts Paper covers Rock crushes Lizard poisons Spock smashes Scissors decapitates Lizard eats Paper disproves Spock vaporizes Rock crushes Scissors.

```
Representations:
- Rock     = 0    - Spock  = 3
- Paper    = 1    - Lizard = 4
- Scissors = 2
```

Diagram 1: Rock Paper Scissors Spock Lizard game description and representation modified from http://www.samkass.com/theories/RPSSL.html.

Since the number of rounds is undetermined, your job is to keep on reading from user input (i.e. *standard input*), all the input to the games *until there are no more inputs*. Every line in the input consist of **two (2)** `integer` numbers representing the shape of hand of player 1 (P1) and player 2 (P2). The numbers on each line are separated by a single [`space`].

For instance, the following sequence:
– `0 1` (P1 plays Rock & P2 plays Paper [P2 wins])
– `4 3` (P1 plays Lizard & P2 plays Spock [P1 wins])
– `2 2` (P1 plays Scissors & P2 plays Scissors [draw])
ends in a draw since both players accumulated 1 win each.

Your job is to write a program such that given an *undetermined* number of rounds, read all the shape of hands in the given round *until there are no more inputs*, and determine the winner of the game. You can check the correctness of your program in CodeCrunch. Write your program below (*hint: what value does* `val` *get from* `val = scanf(...);` *?*):

```
int main() {
  int P1_shape, P2_shape, P1_wins = 0, P2_wins = 0;
   /* Your Solution Here */









  if(P1_wins == P2_wins) printf("draw\n");
  if(P1_wins  > P2_wins) printf("P1 wins\n");
  if(P1_wins  < P2_wins) printf("P2 wins\n");
  return 0;
}
```

(b) [Simulation; Past Question (*modified from 15/16 Sem 2 Assessed Lab 2*)] The New Earth Calendar is an experimental type calendar proposed by James A. Reich that consists of 13 months with 28 days in each month. Thus, a year in New Earth Calendar has 364 days. The shape of the calendar is shown in Diagram 2. Note how every 1st day of the month is always Monday and there is a middle month called Luna added.

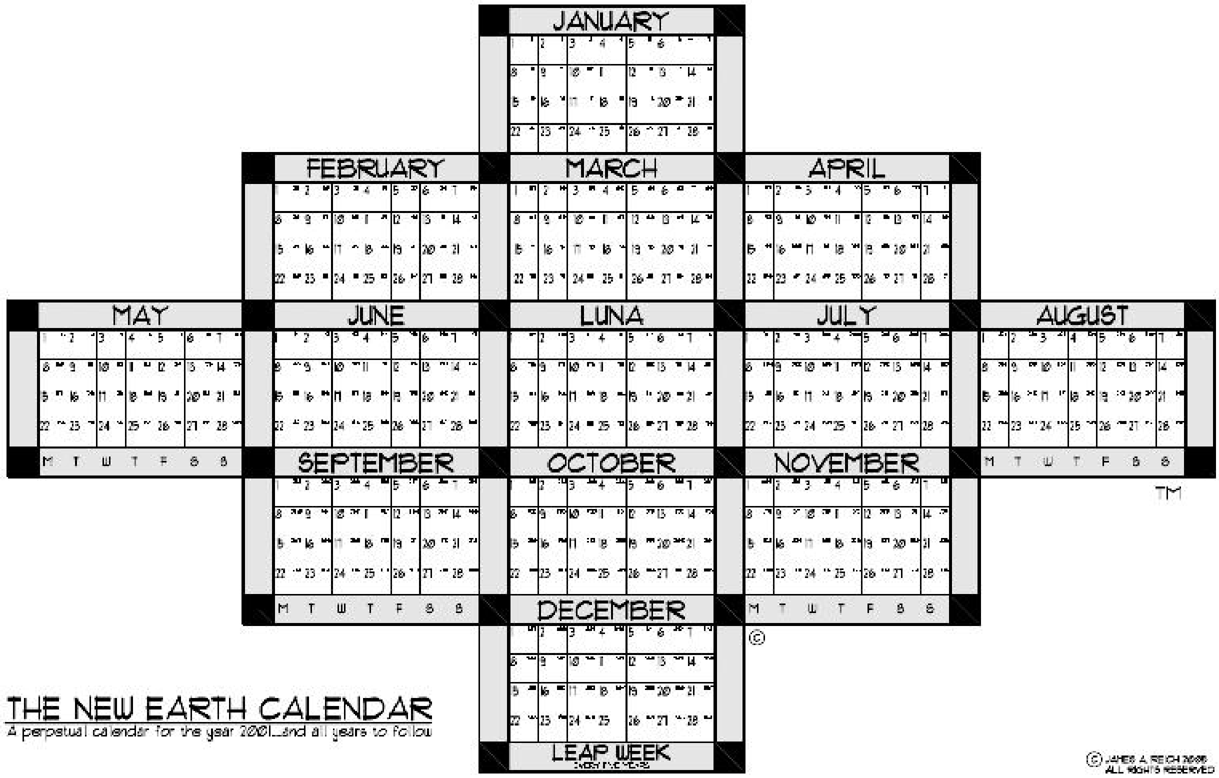


Diagram 2: New Earth Calendar taken from http://calendars.wikia.com/wiki/New_Earth_Calendar.

At the bottom of the diagram, there is a Leap Week. This additional 7 days is added at the end of December every five years with certain exceptions. The rule for determining is a given year is a leap year or not is given below:

1. `year` that is divisible by 5 is leap year, *except for*
2. `year` that is divisible by 40 –which is common year– *unless it is*
3. `year` that is divisible by 400 –which is leap year.

The year in New Earth Calendar starts from the year 2001 (*to coincide with Gregorian Calendar currently in use*). Write the code to determine how many days has passed between two given dates in a format: `day month year` (*i.e.* `5 7 2010` *is the* 5th *of Luna, 2010*). For instance, between `5 7 2010` and `5 13 2010`, there are `168` days in between. For simplicity, assume that years are in the range of [2001, 9999]. You can check the correctness of your program in CodeCrunch. Write your program below:

```
#define LEAP(y)     (y%400==0 || (y%5==0 && y%40))
int main() {
  int day1, day2, month1, month2, year1, year2, days = 0;
  scanf("%d %d %d", &day1, &month1, &year1);
  scanf("%d %d %d", &day2, &month2, &year2);
  /* Your Solution Here */
```

```
  printf("%d", days);
  return 0;
}
```

4. Challenge

(a) [Mathematics; Exhaustive Search] Linear Diophantine equation (LDE) is an equation with the form given in Equation 1. An LDE can be characterized by three values: A, B, and C. An LDE can have *infinitely* many solution (i.e. *infinitely many x and y satisfying the LDE*). However, in our case, we are only interested in **integer** solution from -100 to 100 (*inclusive*) written as $[-100, 100]$.

Furthermore, a more interesting question is the *simultaneous* LDE. That is, given **two (2)** LDEs, find if there are any pair of **integer** (x, y) that satisfies **both** LDEs. For instance, $3x + 4y = 13$ and $-3x + 3y = 36$ has exactly **one (1)** solution in the range $[-100, 100]$ which is $x = -1$ and $y = 7$.

$$Ax + By = C \tag{1}$$

Given the values of A1, B1, C1, A2, B2, and C2, find the solution (*if any*) for both $A_1x + B_1y = C_1$ and $A_2x + B_2y = C_2$. You can check the correctness of your program in CodeCrunch. Write your program below:

```
#define UNSOLVABLE -99999
int main() {
  int A1, B1, C1, A2, B2, C2, x, y, solutionX, solutionY;
   /* Your Solution Here */











  if(solutionX != UNSOLVABLE) {
    printf("(%d %d)", solutionX, solutionY);
  } else {
    printf("The LDE has no solution”);
  }
  return 0;
}
```

(b) [Approximation; Past Question (*modified from 15/16 Sem 2 Assessed Lab 2*)] Euler number e is approximately `2.7182`. However, as a *transcendent* and *irrational* number, it is hard to approximate. A simple approximation using Equation 2 where $k!$ is the factorial of k which is defined as $k! = k \times (k-1) \times (k-2) \times ... \times 1$.

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \frac{x^k}{k!} + ... \tag{2}$$

The problem with this approximation is the *very large* value of the factorial of k for some *very small* value of k. For instance $13! = 6,227,020,800$. Remember that `unsigned integer` has a maximum value $2^{32} = 4,294,967,296$. Using `double` does not solve the underlying problem. Note how the value of e is *very very very* small compared to 13! that we have to compute for the approximation.

A much better approximation is using the *continued fraction* method given in Equation 3.

$$e^x = 1 + \cfrac{x}{1 + \cfrac{-x}{2 + x + \cfrac{-2x}{3 + x + \cfrac{-3x}{4 + x + \cfrac{-4x}{\ddots + \cfrac{\ddots}{(k-1) + x + \cfrac{-(k-1)x}{k + x + \cfrac{\ddots}{\ddots}}}}}}}} \tag{3}$$

Given the value of x and k, approximate the value e^x the k^{th} term in the approximation without using `#include <math.h>` library rounded to 3 decimal places. You can check the correctness of your program in CodeCrunch. Write your program below (hint: *it is easier [but not required] to compute each approximation from the bottom*):

```
int main() {
  double x, approx; int k; scanf("%lf %d", &x, &k);
   /* Approximation of e^x up to k terms */









  return 0;
}
```