

Beginning #1 - Parenthesis Matching

You are to write a program that determines whether a given string is a correctly balanced string of parentheses. Your program must verify that the only characters in the string are '(' and ')', and that the characters may be paired according to the following rules:

- Each '(' is paired with a ')' and vice versa.
- Each '(' appears to the left of the ')' to which it is paired.

Your program must produce one of the following outputs:

- "Balanced." - if the string is properly balanced
- "Not balanced." - if all characters are legal, but the string is not properly balanced
- "Invalid character." - if some character other than '(' or ')' appears in the string.

Example 1:

```
Enter string: ()()  
Balanced.
```

Example 2:

```
Enter string: ())()  
Not balanced.
```

Example 3:

```
Enter string: ((a))  
Invalid character.
```

Beginning #2 - Longest Common Prefix

You are to write a program that displays the longest common prefix of two given strings; i.e., the longest substring that appears at the beginning of both strings. In some cases, the result will be the empty string.

Example 1:

```
Enter string 1: abcde
Enter string 2: abab
Longest common prefix: ab
```

Example 2:

```
Enter string 1: abcdef
Enter string 2: ghijklmn
Longest common prefix:
```

Beginning #4 - Polynomial Root Finder

You are to write a program to find a root of a given cubic polynomial (i.e., find a point at which the polynomial evaluates to 0). Your output must be an integer within a distance strictly less than 1 from a real root. Thus, if the only real root is 3, your program must output 3; on the other hand, if the only real root is 3.2, your program may output either 3 or 4. The coefficients will be integers, and the leading coefficient (i.e., the coefficient of the cubic term) will be nonzero. You may find the following facts helpful (the polynomial is denoted by p):

- If the leading coefficient is positive, then $p(x)$ approaches positive infinity as x approaches positive infinity, and $p(x)$ approaches negative infinity as x approaches negative infinity.
- If the leading coefficient is negative, then $p(x)$ approaches negative infinity as x approaches positive infinity, and $p(x)$ approaches positive infinity as x approaches negative infinity.
- If $p(x)$ is positive and $p(y)$ is negative, then there is a real root strictly between x and y .

Example 1:

```
Enter coefficient for x^3: 1
Enter coefficient for x^2: 2
Enter coefficient for x^1: 3
Enter coefficient for x^0: 0
0
```

Example 2:

```
Enter coefficient for x^3: -1
Enter coefficient for x^2: 0
Enter coefficient for x^1: 0
Enter coefficient for x^0: -27
-3
```

Example 3:

```
Enter coefficient for x^3: -1
Enter coefficient for x^2: 0
Enter coefficient for x^1: 0
Enter coefficient for x^0: 30
4
```

Beginning #6 - Nim

The following game is a variation on Nim. The game is played with some number of stones (at least 2). Each player alternates taking stones according to the following rules:

- Each play consists of taking at least one stone.
- On the very first play, the player must leave at least one stone.
- On each play after the first, the player must take no more than twice as many as were taken by the opponent on the immediately preceding play.

The player who takes the last stone wins.

You are to write a program to referee a game of Nim between two human players, identified as Player 1 and Player 2. After obtaining the initial number of stones from the players, the program then alternately prompts each player for the number of stones to remove, with Player 1 going first. The prompt must include the limit on the number of stones which may be taken; this limit must never be more than the number of stones remaining. Following each play, the program must report the number of stones remaining. When the game is over, the program must identify the winner.

The program must check that each play is legal. You may assume that the initial number of stones will always be an integer greater than 1, but that the players may enter arbitrary integers on their plays.

Example:

```
Enter the number of stones: 10
Player 1, how many stones do you take? (limit 9) 2
That leaves 8 stones.
Player 2, how many stones do you take? (limit 4) 5
Invalid play.
Player 2, how many stones do you take? (limit 4) 0
Invalid play.
Player 2, how many stones do you take? (limit 4) 1
That leaves 7 stones.
Player 1, how many stones do you take? (limit 2) 2
That leaves 5 stones.
Player 2, how many stones do you take? (limit 4) 1
That leaves 4 stones.
Player 1, how many stones do you take? (limit 2) 1
That leaves 3 stones.
Player 2, how many stones do you take? (limit 2) 2
That leaves 1 stones.
Player 1, how many stones do you take? (limit 1) 1
That leaves 0 stones.
Congratulations, Player 1 you win!
```

Advanced #1 - Card Shuffle

You are to write a program to shuffle a deck of cards a given number of times, and to display the first five cards in the resulting sequence. Each card is denoted by two characters, the first giving a rank, and the second giving a suit. The thirteen ranks are, in order:

A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K

and the four suits are, in order:

S, H, D, C

Initially, the cards are ordered first by suit, then by rank; i.e.:

AS, 2S, 3S, ..., QC, KC

Each shuffle splits the deck in half, and alternates between the two halves, one card at a time, beginning with the second half. Thus, after the first shuffle, the cards are ordered:

AD, AS, 2D, 2S, ..., KC, KH

You may assume the user always inputs a nonnegative integer.

Example 1:

```
Enter number of shuffles: 1
AD AS 2D 2S 3D
```

Example 2:

```
Enter number of shuffles: 20
3C 5D 7H 9S QC
```

Advanced #2 - Longest Common Substring

You are to write a program that displays a longest common substring of two given strings. In some cases, the result will be the empty string. You may break ties in any way you wish.

Example 1:

```
Enter string 1: Mississippi  
Enter string 2: resistance  
Longest common substring: sis
```

Example 2:

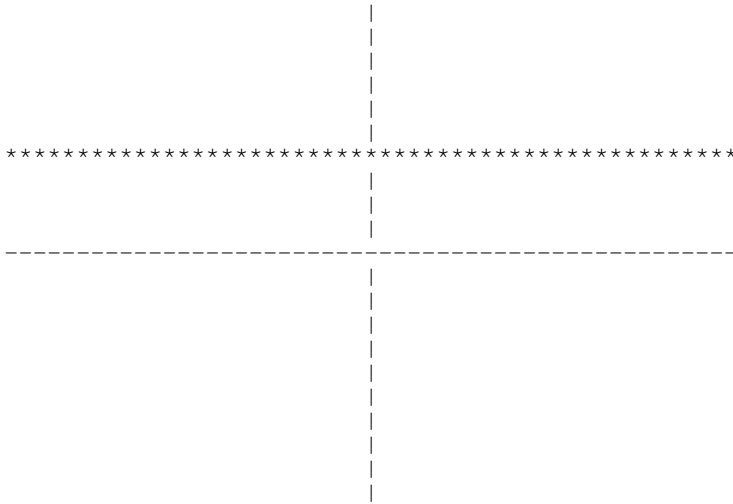
```
Enter string 1: abcde  
Enter string 2: ghijklm  
Longest common substring:
```

Advanced #4 - Polynomial Graphing

You are to write a program to graph a polynomial. The program will be given the degree of the polynomial, which is assumed to be a nonnegative integer no greater than 5, followed by the coefficients in decreasing order of degree. The coefficients are all assumed to be integers. The program then draws on the screen a graph of the polynomial over the domain -25 to 25, inclusive, showing only values that lie in the range -10 to 10, inclusive. The picture will be 21 row by 51 columns, each column representing a different x value, and each row representing a different y value. The x-axis should be drawn using the '-' character, and the y-axis should be drawn using the '|' character. The function points should be drawn using the '*' character. If a function point lies on one of the axes, the function point should be visible, and if no function point lies on the origin, it should be drawn with the '-' character.

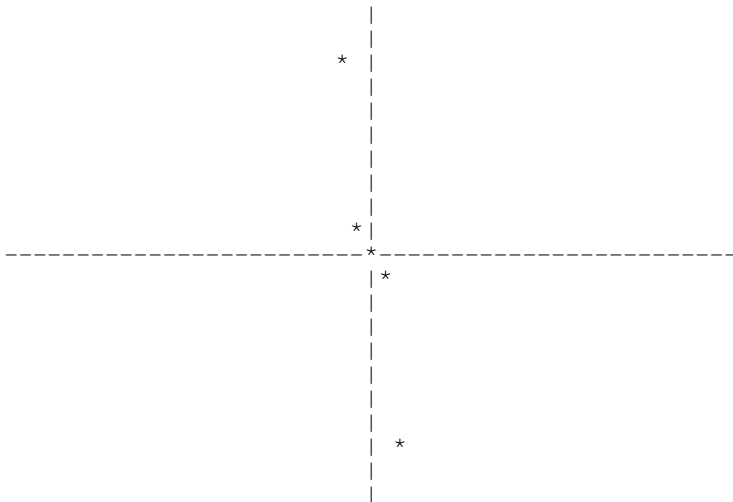
Example 1:

```
Enter degree of polynomial: 0
Enter coefficient for x^0: 4
```



Example 2:

```
Enter degree of polynomial: 3
Enter coefficient for x^3: -1
Enter coefficient for x^2: 0
Enter coefficient for x^1: 0
Enter coefficient for x^0: 0
```



Advanced #6 - Nim

The following game is a variation on Nim. The game is played with some number of stones (at least 2). Each player alternates taking stones according to the following rules:

- Each play consists of taking at least one stone.
- On the very first play, the player must leave at least one stone.
- On each play after the first, the player must take no more than twice as many as were taken by the opponent on the immediately preceding play.

The player who takes the last stone wins.

The first player can force a win whenever the initial number of stones is not one of the following Fibonacci numbers:

2, 3, 5, 8, 13, ...

where each successive number of the sequence is the sum of the preceding two.

The winning strategy follows these rules, assuming n stones remain:

- If it's possible to take all remaining stones, do so.
- Otherwise, if n is no more than 3, take 1 stone.
- Otherwise, use the rule that would apply if:
 - $n - m$ stones remain, where m is the largest Fibonacci number less than n ; and
 - the legal limit were the minimum of the current limit and $(n - 1) / 3$ (rounded down).This rule will eventually reduce to one of the first two.

You are to write a program to play a perfect game of Nim using the above strategy. The program must display the number of stones remaining after each play, the number of stones the computer takes, the limit the human player is allowed to take (this limit may exceed the number of stones left), and the winner. You may assume the human always takes a legal number of stones.

Example 1:

```
Enter the number of stones: 7
I take 2 stones, leaving 5
How many do you take? (limit 4) 1
That leaves 4 stones.
I take 1 stones, leaving 3
How many do you take? (limit 2) 2
That leaves 1 stones.
I take 1 stones, leaving 0
I win!
```

Example 2:

```
Enter the number of stones: 8
I take 1 stones, leaving 7
How many do you take? (limit 2) 2
That leaves 5 stones.
I take 1 stones, leaving 4
How many do you take? (limit 2) 1
That leaves 3 stones.
I take 1 stones, leaving 2
How many do you take? (limit 2) 2
That leaves 0 stones.
Congratulations, you win!
```