Notas y ejercicios para PR1

Leandro Rabindranath León

19 de noviembre de 2011

1. Especificación y solución de un problema de programación

En lo que sigue, como instrumento ejemplar de explicación, consideremos el enunciado de problema siguiente: Dada una ecuación de segundo con la forma general $ax^2 + bx + c = 0$, diseñe e implemente una función que la resuelva.

Cada vez que se te plantee un problema de programación, es necesario que realices los siguientes pasos tendientes a delimitarlo.

1.1. El entendimiento del problema

Una cosa debe serte clara: no lograrás programar la solución a un problema si no entiendes cuál es el problema que resolvería tu programa. Esto puede parecer una trivialidad o perogrullada, pero habida cuenta de la gran cantidad de estudiantes (inclusive en las últimas etapas de su carrera) que se sientan a programar sin tener esto claro, debo insistirte en esta observación.

EL entendimiento comienza con una sensación de fluidez en tu pensamiento acerca de lo que te dice tu interlocutor; es decir, no te bloqueas pensando qué es lo que quiere decir tu interlocutor. Así, una primera señal de entendimiento es que tú sientas entiendes sobre el problema que se está planteando sin lagunas o partes oscuras.

A veces no es fácil sentir la seguridad de que se entiende cuál es el problema. Todo depende de la complejidad del problema, de tu destreza (en inteligencia pero también en experiencia) y, hay que decirlo, en algunos caso de la suerte.

En su celebérrimo libro "How to Solve It", George Polya plantea tres preguntas motoras (que mueven) que conducen a entender el problema:

1. **¿Qué es lo desconocido?**: enfrentar esta pregunta te revela el problema en si, lo que deseas resolver. Por lo general, las respuestas que obtengas expresan la solución.

   Lo respondido de esta pregunta te da la salida de tu programa.

2. **¿Cuáles son los datos que se requieren para poder resolver el problema?** Una vez que sabes qué es lo que quieres responder como solución, debes preguntarte qué información requieres para resolver el problema.

Lo respondido en esta pregunta te da los datos de entrada de tu programa.

3. ¿Cuál es (o cuáles son) la condición? o, dicho de otra manera, ¿cuál es la condición que relaciona los datos que se te dan como entrada para resolver el problema y así poder expresar su solución (o salida)?

   Esta es la pregunta más compleja y apunta al cómo utilizas los datos de entradas para obtener la solución. Polya recomienda (y yo también) que busques relaciones entre lo conocido (los datos de entrada) y lo desconocido (los datos de salida).

Por lo general, la pregunta más fácil es la primera y la más difícil la última.

Si no eres capaz de responder las dos primeras preguntas, entonces ni de vaina serás capaz de responder la tercera.

Las dos primeras preguntas (en ese orden) son la mitad del problema. Así, si logras responderlas bien, entonces tendrás cubierta la mitad del problema, lo que es digno y meritorio.

La otra mitad, o sea, la tercera pregunta, que es la más difícil deberás responderla para poder programar la solución del problema. Dicho de otro modo, si no logras responder la tercera pregunta, entonces no será capaz de escribir el programa.

Reflexionemos sobre esta preguntas motoras con el ejemplo propuesto al principio con la ecuación de segundo grado.

1. ¿Qué es lo desconocido? Respuesta: un valor particular de x tal que al sustituirlo en la expresión $ax^2 + bx + c$ te arroje como resultado cero.

   Así, la salida de un supuesto programa que resuelva una ecuación de segundo sería el (o los) valores que al sustituirlos en la expresión ax^2+bx+c te den cero.

   ¿Entendiste? ... ¡espero que sí!

2. ¿Cuáles datos requieres para resolver una ecuación de segundo grado? Respuesta, los valores de los coeficientes a, b y c.

   Así, los datos de entrada de tu programa son los valores de los coeficientes.

3. ¿Cuál es la condición? la respuesta es fácil si vives en el siglo XXI y cursaste bien el 9no año de primaria (3ro del antiguo bachillerato).

   En efecto, en aquel momento (9no año) se te explicó bien la solución general de una ecuación de 2do grado, a saber:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

   Esta es la relación o condición que requieres para resolver el problema.

   Nota que cuando planteé ¿qué es lo desconocido? respondí: “un valor particular ...” ¡en singular, no en plural!. Ahora bien, una vez que conoces

"el cómo resolverlo" te das cuenta, al mirar (1), que pueden existir dos valores, y no uno como te respondiste al principio, que son solución.

Lo que sucedió es que cuando develaste el cómo te diste cuenta de que había algo más desconocido al principio: que pueden existir dos valores y no uno como suponías que satisfacen la ecuación.

Lo anterior no es atípico y ocurre con cierta frecuencia. Se trata de un refinamiento ganado cuando se termina de entender el problema.

1.2. Estructura general de un problema de programación

Escribes programas para resolver problemas. Consecuentemente, para poder escribir un programa, auténticamente hablando, es decir, no transcribirlo, debes entender el problema.

En esta sección plantearemos una extensión de la táctica de Polya para resolver problemas. En lo que sigue, se enuncian los requisitos que debes cumplir para poder escribir auténticamente un programa.

1.2.1. Especificación de los datos de entrada

Aquí debes identificar todos los datos que se requerirían para resolver el problema en cuestión; o sea, la segunda pregunta de Polya.

Es muy importante que a cada dato se le asigne un nombre que sea representativo del dato. Lo idóneo es que el nombre exprese el significado del dato.

También es esencial definir es tipo de dato con el cual se representaría en un programa.

Finalmente, debes definir lo que se conocen como "precondiciones", sobre los datos. Una precondición es una condición que se debe cumplir para que el programa funcione correctamente. Se trata de una especie de contrato con clausulas del tipo: me comprometo a resolverte el problema a condición de que cumplas con esta "precondición".

Con los datos de entrada es muy importante especificar precondiciones, cuando sea necesario. Las precondiciones pueden expresarse en forma coloquial o en predicado lógicos.

En nuestro ejemplo, como ya lo dijimos, los datos de entrada serían los coeficientes a, b y c. Puesto que estos nombres sólo tienen sentido en el contexto de la ecuación $ax^2 + bx + c = 0$, debemos mencionar la ecuación, pues fuera de este contexto los nombres a, b y c no significan nada. También, teniendo el conocimiento de la ecuación (1), sabemos que el valor del coeficiente a no puede ser cero, pues sino cometeríamos el pecado de dividir por cero. Así, $a \neq 0$ es una precondición sobre el dato a.

El mejor tipo de dato del que disponemos para definir los tres datos de entrada es el `float`.

Nota la riqueza expresiva de especificar el tipo de dato. Cuando decimos que a, b y c son de tipo `float` ya estamos diciendo que son números reales representados en punto flotante. Por tanto, no es necesario establecer como

precondiciones frases o predicados del estilo "a debe ser un número real" o "a no puede ser una cadena".

Así, los datos de entrada se podrían resumir en la siguiente tabla:

Nombre del dato	Tipo	Descripción	Precondición
a	`float`	Coeficiente constante a en $ax^2 + bx + c = 0$	$a \neq 0$
b	`float`	Coeficiente constante b en $ax^2 + bx + c = 0$	
c	`float`	Coeficiente constante c en $ax^2 + bx + c = 0$	

1.2.2. Valores de salida o resultados

Los datos de salida están conformados por lo que se expresaría como solución del problema, o sea, la primera pregunta de Polya. Tan simple como suena.

Toma en cuenta que no podrás estar absolutamente seguro de todos los datos de salida hasta el momento en que sepas con claridad cómo se resuelve el problema. A pesar de ello, a un programador novicio yo le recomiendo que se plantee esta pregunta primero antes de plantearse el cómo. Después, cuando tengas más experiencia, tú decidirás el orden.

Al igual que con los datos de entrada, los resultados deben tener un nombre, el cual debe ser representativo de lo que el resultado expresa. Del mismo modo, también debes especificar el tipo de dato al que pertenece un resultado. Cuando el resultado sea propenso a darse como valor de retorno de la función que resuelve tu problema y así lo decidas, entonces al resultado lo llamarás "valor de retorno".

En otro ámbitos es útil establecer condiciones para los resultados. En este caso les llamamos postcondiciones y son herramientas muy útiles para validar y corregir programas. Sin embargo, por simplicidad, como estás comenzando a programar, en este curso no las consideramos.

En el ejemplo de la ecuación de segundo grado, bajo el conocimiento que ya tenemos de la solución (1), planteamos los siguientes resultados o salidas:

Nombre	Tipo	Descripción
valor de retorno	`float`	cantidad de soluciones de $ax^2 + bx + c = 0$
x_1	`float`	primera solución (si existe)
x_2	`float`	segunda solución (si existe)

1.2.3. Definición de la interfaz

A la excepción de su inconstancia, nada es constante en la vida. Sin embargo, en este curso vamos a tomar como constante la siguiente regla: todo problema que se nos plantee programar definiremos una función que lo resuelve.

Así, siempre que se te plantee un problema de programación, pensarás en una función que resuelva el problema.

A la función debes ponerle un nombre, que no puede ser cualquier nombre. Debe ser un nombre que exprese lo mejor posible cuál es el problema que resuelve la función.

Por tanto, una vez definidos los datos de entrada y los resultados, debes definir el nombre de la función.

Los datos de entrada son parámetros de la función. Si la solución del problema tiene un solo resultado, entonces es preferible que este sea el valor de retorno de la función. En este caso, el nombre de la función debe decir lo que ella retorna (la solución al problema). En este curso, la mayoría de las veces los parámetros correspondientes a los datos de entrada son por valor.

Si la solución del problema se expresa en más de un resultado, entonces es preferible colocar los resultados en parámetros por referencia.

Si la función no tiene valor de retorno, entonces es recomendable que el nombre contenga un (o varios) verbo(s) que exprese(n) lo que ella hace.

El nombre de la función, el tipo de su valor de retorno, los nombres de los parámetros que recibe, sus tipos y sus significados conforman la "interfaz".

Interfaz puede verse como inter-faz, o sea, entre caras, lo que está entre dos caras. En el caso de un programa, una cara es el solucionador del problema, es decir, nuestro programa; la otra cara es algún "agente", que puede ser una persona u otro programa que requiera resolver el problema a través de nuestro programa.

En inglés, "interfaz" se escribe "interface". Quizá esto, aunado a que lo plural de lo que termina en z se vuelve "ces" y que por eso el plural de "interfaz" es "interfaces", ha propiciado confusión con el término literal en inglés o con la palabra interfase, que sí existe en castellano, pero que no es lo mismo que interfaz.

La definición de la interfaz es crucial y tiene varias y diversas bondades:

1. Tu programa se hace "reutilizable" e independiente del cómo se resuelve. Otros pueden utilizar tu programa sin necesidad de programarlo o inclusive sin necesidad de entenderlo completamente (el cómo se resuelve).

2. Permite la programación cooperativa. Muchas veces un problema se descompone en otros problemas o subproblemas. En este caso, el trabajo de solución puede repartirse entre varios programadores repartiendo los subproblemas. Para ello, los programadores deben entender y acordar las entradas y salidas de los programas, pero no todos tienen que comprender el cómo se resuelve.

3. También te permite muchísima flexibilidad para programar y obtener la solución a tu problema. Una vez que divides el problema en subproblemas, defines y comprendes sus interfaces, puedes acometer cada subproblema en orden en que prefieras. También puedes buscar soluciones en otras fuentes y reutilizarlas. En este último caso, no olvides de reconocer el trabajo del que realizó el subprograma que estás reutilizando mencionando la fuente.

   En algunos casos, los programas tienen licencias. Una licencia es como si en lugar de permitirte adquirir un bien, un lápiz o un carro, por ejemplos, se te prestase el bien (lápiz o carro) con un contrato en el cual aceptas algunas condiciones de uso. Trata de evadir estas situaciones y de no usar

estos programas, pues aparte o incurres en una ilegalidad en el sistema jurídico o te hace cómplice de un modo de explotación que es muy injusto. Las ideas, entre ellas soluciones a los problemas, no son susceptibles de propiedad individual.

En el ejemplo de la ecuación de segundo grado podríamos escoger la siguiente interfaz:

```
short raices_ec_2do_grado(float a, float b, float c, float & x1, float & x2)
```

De este modo, puedes escribir programas que se expresen por sí mismo qué es lo que hacen.

1.2.4. Solución del problema = programa

Finalmente, en teoría la parte más difícil, debes poder plantear cómo un computador resuelve el problema. Para ello, debes tener claro el "algoritmo".

"Un algoritmo es una secuencia finita de instrucciones que acomete la consecución de un fin. Usamos algoritmos en diversos contextos de la vida; por ejemplo, cuando preparamos un plato de comida según alguna receta. La cultura nos ha inculcado algunos "algoritmos", culturales, no naturales, para desenvolvernos socialmente; por ejemplos, el algoritmo de conducir un automóvil o el algoritmo para cruzar una calle. En ambos ejemplos, el fin que se plantea es arribar a un sitio.

En esta definición "fin" debe entenderse en el sentido de propósito o meta.

En nuestro caso, un algoritmo es la secuencia de instrucciones que ejecutaría el computador para resolver el problema. En tu situación de programador novel, el fin del algoritmo es resolver el problema. Así de simple. Ahora bien, es imposible que puedas explicar un algoritmo si tú no entiendes el problema. De allí pues la importancia y mi insistencia en el entendimiento del problema.

El cómo tú

Tienes los datos de entrada, los valores de salida y la interfaz bien definida. Como ya te lo entredije, cuando llegas a este punto ya tienes la mitad del problema resuelto. Ahora te toca enfrentar el cómo resolverlo, el descubrir la condición que relaciona la entrada para producir, calcular, deducir, encontrar o conocer la solución. Si ya conoces la solución, entonces entiendes el problema y sólo te queda escribir el algoritmo que lo resuelva.

Si aún no conoces una solución, entonces no estás en aprietos, pero tienes que hacer un poco de esfuerzo para descubrirla o, como te sucederá en la mayoría de los casos, para terminar de entenderla.

Una vez que conoces una solución, descubierta por ti o aprendida, debes ser capaz de explicarla coherentemente a otro sin mucha dificultad. Quizá esta es la parte más difícil de aprender a programar, pues en cierta forma un algoritmo o programa es una explicación al computador acerca de lo que él debe ejecutar para solucionar el problema. Por esta razón, es muy importante que verifiques que puedes ejecutar por ti mismo el algoritmo tal como si tú fueses el computador.

Por la razón anterior, un buen principio para deducir algoritmos de problemas sencillos o medianos y un buen ejercicio para aprender a programar, es analizar cómo tú mismo haces para resolver el problema.

Regresemos al ejemplo de la ecuación de 2do grado, el cual quizá te parezca muy simple. Si se te encomendase resolver una ecuación de 2do grado, ¿cómo la resolverías? Probablemente responderías, aplicando la ecuación (1) . Este es un caso en el cual ya conoces la solución. Al recordar esa fórmula, te percatarás de que requieres los valores de los coeficientes, lo que te ayudará a identificar los datos de entrada. Del mismo modo, mirando la expresión (1) también podrías percatarte de que es posible tener dos soluciones distintas, así como situaciones en las cuales no hay solución. Si te detienes a pensar un poco, entonces quizá te digas que el coeficiente a no debe ser cero, pues podría ocurrir una división pecaminosa. Finalmente, si piensas en cómo haces cuando tú mismo calculas la solución (tal como lo hacías hace poco y lo seguirás haciendo), entonces verás que es preferible efectuar el cálculo $\sqrt{b^2 - 4ac}$ antes que el de $2a$, que la última operación que haces es dividir entre $2a$ y demás cosas por el estilo.

Este estilo de pensamiento, que en el caso de la ecuación de 2do grado probablemente tú ya lo haces tan bien que ya no eres consciente, es la manera de pensar que requieres para aprender a programar, para aprender a responder "el cómo se hace", o cuál es la condición, en los términos de Polya. Con el tiempo y la práctica, se te instituirá y se te volverá inconsciente de la misma manera en que ahora no te detienes a meditar cuando se te pide que resuelvas una ecuación de 2do grado. En el ínterin, mientras aprendes, te recomiendo que te esfuerces en reflexionar cómo tú ejecutas la resolución de un problema.

El conocimiento de la solución La mayoría de las veces te tocará resolver un problema cuya solución ya ha sido conocida por otro ser humano. Vives en una época tildada como "la era del conocimiento". Aunque yo tengo mis discrepancias con este mote, tanto en lo de era como en lo de conocimiento, debo admitir que en este tiempo es mucho más fácil que otrora encontrar una solución hallada por otro.

Descubrir la solución, haya o no sido descubierta previamente, es un asunto que no nos atañe directamente en este curso. De hecho, no es durante un curso que se aprende a descubrir soluciones a problemas nuevos para ti o para todo el mundo. Esto es un asunto que se aprende durante toda la vida. De todos modos, aprovecho la oportunidad para hacerte algunas recomendaciones importantes:

1. Repasa tus conocimientos de matemática y física. En lo particular de este curso, te recomiendo que leas o releas la Aritmética de Baldor.

2. Nunca, pero nunca, desprecies el valor de la matemática y de la física. Así que ponle todo el esmero a los cursos de cálculo y de física. Egos aparte, son cursos más importantes que este de programación.

   También, por si acaso, si escuchas a un compañero hablar mal de la matemática o de la física, no lo critiques pero no te dejes llevar por él en un proyecto importante que ataña a la ingeniería. Si escuchas a un profesor

relacionado con la ingeniería hablar mal de la matemática o de la física, escúchalo atentamente pero ten muchísimo cuidado porque es bastante extraño. Si no lo logras comprender, es preferible que te alejes de él.

3. ¿Alguna recomendación? Léete "How to Solve It" de Polya. Está traducido al castellano y lo puedes encontrar por allí.

Los programadores debemos terminar rápidamente nuestros programas. Por consiguiente, si te encuentras con un problema cuya solución ya es conocida, entonces considera seriamente aprenderla y entenderla. Aunque te perderás el sabor de descubrirla por ti mismo, terminarás más pronto. En el mismo sentido, si te topas con un problema del cual sospechas ya se conoce una solución, entonces haz el esfuerzo de buscarla y encontrarla. En los dos caso, no lo olvides, haz el reconocimiento al autor.

Algunas veces no tendrás otra alternativa que descubrir por ti mismo la solución. No veas esto como una mala cosa, pues es una oportunidad de experienciar una de las sensaciones más importantes del ser humano: el descubrimiento.

El lenguaje de especificación del algoritmo

Por motivos prácticos y didácticos, en lo personal yo prefiero leer y escribir programas directamente en un lenguaje de programación. De hecho, por eso se les llaman "lenguajes de programación" y no lenguajes de otra cosa, pues son lenguajes para expresar programas. Te recomiendo que en el futuro escribas algoritmos en un lenguaje o pseudolenguaje de programación, pero que no los escribas en un pseudolenguaje castellanizado. Considera por ejemplo, este pedazo de programa:

```
Repita mientras m > 0 y m < n
  si a[m] < a[x] entonces
    m = m/2;
  de lo contrario
    m = 2*m
  fin si
Fin repita mientras
```

Ahora considera el siguiente pedazo de programa equivalente expresado en C++:

```
while (m > 0 and m < n)
{
  if (a[m] < a[x])
    m = m/2;
  else
    m = 2*m;
}
```

Reconozco que si no sabes inglés tendrás un poco de menos posibilidades de entender el segundo programa. Pero, una vez que conozcas los pocos términos

ingleses del segundo programa, tendrás muchísimas más ventajas sobre aquellas personas que lo escriben en castellano. Si los escribes en castellano, entonces, cuando tengas que irte al computador, tendrás que escribirlo en el lenguaje de programación; o sea, repetirás el trabajo. Créeme cuando te digo que trabajar el doble es bastante fastidioso.

Por otra parte, aunque adoro el castellano, es mi lengua materna y me proclamo anti-imperialista, sé muy bien que todos los lenguajes de programación de hoy son en inglés sencillo. Por lo tanto, sé también que si escribiese programas en castellano muchas menos personas me los leerían.

Algunas personas te dirán que lo importante es aprender la "lógica de la programación" y que ello es independiente del lenguaje. Algunas de estas personas son muy respetables y estoy seguro que lo dicen con sentido. Sin embargo, discrepo de estas personas. En mi opinión, la lógica de programación es la misma lógica que necesitas para cocinar, para resolver un problema de física o matemática, para escribir correctamente, etc. Lo que tú no dominas bien en este momento es el lenguaje del mundo de la programación, así como alguna vez no sabías el lenguaje de la escritura (sus reglas, etc), o no sabías matemática. Para escribir programas empleas un lenguaje, casi de la misma manera en que para la matemática o la cocina utilizas un lenguaje especial. Lo que estas personas llaman lógica de la programación es en realidad un lenguaje nuevo, palabras, sintaxis y semántica obedecen a la lógica cotidiana. Por eso yo prefiero enseñarte a razonar un lenguaje de programación en un lenguaje de programación y no hacerte trabajar doble o que hagas trabajar doble a otros, castellanizando un lenguaje de programación.

Veamos ahora el algoritmo para resolver la ecuación de 2do grado:

```
short raices_ec_2do_grado(float a, float b, float c, float & x1, float & x2)
{
  float discriminante = b*b - 4*a*c;
  if (discriminante < 0)
    return 0; // no existe solución

  float dos_a = 2*a;

  if (discriminante == 0)
    {
      x1 = -b / dos_a;

      return 1; // una sola solución (en x1)
    }

  float raiz_cuadrada_disc = sqrt(discriminante);
  x1 = (-b + raiz_cuadrada_disc) / dos_a;
  x2 = (-b - raiz_cuadrada_disc) / dos_a;

  return 2; // dos soluciones (x1 y x2)
```

```
}
```

Sólo por esta vez, y sólo esta vez, te lo escribo castellanizado:

```
Entero raices_ec_2do_grado
Entrada: Real a, Real b, Real c
Salida: Real x1, Real x2

1. Real discriminante = b*b - 4*a*c

2. si discriminante < 0
     retornar 0;

3. Real dos_a = 2*a;

4. si discriminante == 0
     x1 = -b / dos_a;
     retornar 1
   fin si 4.

5. Real raiz_cuadrada_disc = sqrt(discriminante);
6. x1 = (-b + raiz_cuadrada_disc) / dos_a;
7. x2 = (-b - raiz_cuadrada_disc) / dos_a;

  retornar 2; // dos soluciones (x1 y x2)

Fin del programa
```

¡Ahora vete a probarlo en un computador! (te apuesto a que me agradecerás la versión en C).

2. Ejercicios

Para cada uno de los ejercicios que se presentan a continuación, se debe identificar:

1. Los datos de entrada
2. Los valores de salida o resultados
3. La interfaz
4. El algoritmo en pseudolenguaje C

Haz tu máximo esfuerzo por hacer los ejercicios tú mismo. Por cada ejercicio, plantéate las tres preguntas e intenta responder cada parte. Si te bloqueas en un ejercicio, parcial o totalmente, márcalo como pendiente, sigue con el siguiente

y después regresa a tratar de nuevo. Recuerda que como toda práctica la única manera de mejorar la programación es programando.

Programa tú mismo tus soluciones en el computador.

1. Dado un entero positivo n, determine si es o no un número primo. Por ejemplo, $n = 8$ no es primo, mientras que $n = 11$ sí lo es.

2. Dado un número n, entero o real, calcula su raíz cuadrada con una precisión de dos decimales. Por ejemplo, para $n = 101$, la raíz cuadrada es $10,05$

3. Calcular los n primeros números primos. Por ejemplo, para $n = 5$, el resultado sería $1, 2, 3, 5, 7..$

4. Calcular el i-ésimo número primo.

5. Dados números enteros n y m calcular su máximo común divisor (mcd). Por ejemplos, $n = 5, m = 7, \text{mcd}(5, 7) = 1$, $n = 16, m = 24, \text{mcd}(16, 24) = 8, \text{mcd}(44, 72) = 4$.

6. Dado un número entero positivo n calcular su descomposición en factores primos. Por ejemplos, $n = 204 = 2^2 \times 3 \times 17$, $n = 25230 = 2 \times 3 \times \times 5 \times 29^2$.

7. Dados dos números enteros n y m hallar su mínimo común múltiplo o $\text{mcm}(n, m)$. Por ejemplos, $\text{mcm}(24, 36) = 72, \text{mcm}(25, 55) = 275$.

8. Dado un número entero n calcular su factorial denotado como $n!$ y definido como sigue:
$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times \cdots \times (n-1) \times n$$

9. La cantidad de maneras de combinar n elementos en conjuntos de tamaño m se le conoce como una "combinación" y se denota como:
$$\binom{n}{m} = \frac{n \times (n-1) \times (n-1) \times \cdots \times (n-k+1)}{1 \times 2 \times \cdots \times m} = \frac{n!}{m!(n-m)!} \quad (2)$$

   También se puede interpretar como la cantidad de subconjuntos distintos de m elementos de un conjunto de n elementos.

   También se puede interpretar como el coeficiente del término $x^m y^{n-m}$ del desarrollo o expansión de $(x+y)^n$; conocimiento que proviene del teorema de Newton[1], el cual se expresa así:
$$(x+y)^r = \sum_{i=0}^{\infty} \binom{r}{i} x^{r-i} y^i \quad (3)$$

[1] En realidad el teorema fue descubierto primero por el matemático persaAl-Karaji.

Donde $r \in \mathcal{R}$ y cada coeficiente $\binom{r}{i}$ se expresa como:

$$\binom{r}{i} = \frac{1}{i!}\prod_{k=0}^{i-1}(r-k) \tag{4}$$

Si r es un entero m, entonces (3) puede escribirse como:

$$(x+y)^n = \sum_{i=0}^{n}\binom{n}{i}x^{r-i}y^i \tag{5}$$

y cada coeficiente se define como (2).

Dados dos enteros n y m, $n \geq m$, calcular $\binom{n}{m}$.

10. Dado un entero n, calcular todos los factores $(x+y)^n$. Por ejemplo, $(x+y)^5 = x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + y^5$.

11. Dado un vector $\bar{v} = [v_1\ v_2\ v_3 \ldots\ v_n]$ de n elementos calcular su menor elemento. Por ejemplo, para $\bar{v} =$ [45 34 90 13 7 11] el menor es 7

12. Dado un vector $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ de n elementos calcular su mayor elemento.Por ejemplo, para $\bar{v} =$ [45 34 90 13 7 11] el mayor es 90.

13. Calcular la suma vectorial de dos vectores $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ y $\bar{w} = [w_1\ w_2\ w_3\ \ldots\ w_n]$ de tamaño n, dado que:

$$\bar{c} + \bar{w} = [v_1 + w_1 \quad v_2 + w_2 \quad \ldots \quad v_n + w_n]$$

14. Calcular el producto de un vector $\bar{v} = [v_1\ v_2\ v_3 \ldots\ v_n]$ de n elementos pro un escalar k definido como:

$$k\bar{v} = [kv_1\ kv_2\ kv_3 \ldots\ kv_n]$$

Por ejemplo, para $3\bar{v} =$ [3 7 0 8] = [9 21 0 24].

15. Calcular el producto escalar de dos vectores $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ y $\bar{w} = [w_1\ w_2\ w_3\ \ldots\ w_n]$ de tamaño n, dado que:

$$\bar{c} \cdot \bar{w} = \sum_{i=1}^{n} v_i \cdot w_i$$

Por ejemplo, para [3 7 0 8] $\cdot$ [1 2 2 0] $= 3 + 14 + 0 + 0 = 17$.

16. Dado un vector $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ de n elementos, "agrandar" el vector mediante "inserción" de un elemento v_i en la posición k. Por ejemplo, insertar 13 en la posición 2 de $\bar{v} =$ [3 7 0 8] da como resultado $\bar{v} =$ [3 13 7 0 8].

17. Dado un vector $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ de n elementos, "achicar" el vector mediante "eliminación" de un elemento v_i en la posición k. Por ejemplo, eliminar el elemento de la posición 3 en $\bar{v} = [3\ 13\ 7\ 0\ 8]$ da como resultado $\bar{v} = [3\ 13\ 0\ 8]$.

18. Dado un vector $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ de n elementos, ordenar sus elementos de menor a mayor. Por ejemplo, ordenar $\bar{v} = [3\ 13\ 7\ 0\ 8]$ da como resultado $\bar{v'} = [0\ 3\ 7\ 8\ 13]$.

19. Dado un vector $\bar{v} = [v_1\ v_2\ v_3\ \ldots\ v_n]$ de n elementos verificar si un valor cualquiera k está o no contenido dentro de $\bar{v}$. Por ejemplo, para $\bar{v} = [3\ 13\ 7\ 0\ 8]$, el valor 7 está contenido, mientras que el valor 15 no está contenido.

20. Dado un número entero de a lo sumo cuatro dígitos decimales, escribir su valor en castellano. Por ejemplo, 2567 se escribe en castellano como: "dos mil quinientos sesenta y siete".

21. Calcular el área de triángulo cualquiera.

22. Calcular el área de una circunferencia.

23. Calcular el área de un rectángulo.

24. Calcular el área de un pentágono.

25. Calcular el área de un hexágono.

26. Calcular el área de un polígono regular de n lados.

27. Resolver una ecuación general de primer grado.

28. Resolver un sistema de ecuaciones con dos incógnitas de forma:

$$\begin{aligned} a_1x_1 + a_2x_2 &= a_3 \\ b_1x_2 + b_2x_2 &= b_3 \end{aligned}$$

    Donde $a_1, a_2, a_3, b_1, b_2, b_3$ son coeficientes constantes y x_1 y x_2 son las incógnitas.

29. Definir un tipo de dato que permita especificar un número mixto.

    Un número mixto es uno compuesto por un entero y un quebrado. Ejemplos: $5\frac{4}{5}, 10\frac{29}{77}$.

    Nota 1: cualquier quebrado de forma $\frac{n}{d}$ puede expresarse como el número mixto $1\frac{n}{d}$.

    Nota 2: la unidad puede expresarse como un número mixto así: $1\frac{1}{1}$.

    Nota 3: cualquier quebrado con numerador igual a su denominador es igual a la unidad ($\frac{k}{k} = 1 = 1\frac{1}{1}$).

30. Un quebrado propio es uno cuyo numerador es menor que su denominador. Inversamente, un quebrado impropio es uno cuyo numerador es mayor que su denominador.

    Un número mixto propio es uno cuyo quebrado es propio.

    Dado un número mixto impropio, convertirlo a uno propio.

31. Un quebrado reducido a su más simple expresión es uno cuyo denominador es el menor posible. Por ejemplo:

$$\frac{62}{84} = \frac{2\texttt{.}31}{2\texttt{.}2\texttt{.}3\texttt{.}7} = \frac{31}{42}$$

    Convertir un número mixto a uno cuyo quebrado está reducido a su más simple expresión.

32. Calcular la suma de dos números mixtos. El resultado debe ser propio.

33. Calcular la resta de dos números mixtos. El resultado debe ser propio.

34. Calcular el producto de dos números mixtos. El resultado debe ser propio.

35. Calcular la división de dos números mixtos. El resultado debe ser propio.